

Table of Contents

Introduction	1.1
1	1.2
1 on 1 Template	1.2.1
A	1.3
AB testing	1.3.1
ACID Transaction	1.3.2
AI Engineer	1.3.3
AI governance	1.3.4
API Driven Microservices	1.3.5
API	1.3.6
ARIMA	1.3.7
AUC	1.3.8
AWS Lambda	1.3.9
Accessing Gen AI generated content	1.3.10
Accuracy	1.3.11
Activation Function	1.3.12
Activation atlases	1.3.13
Active Learning	1.3.14
Ada boosting	1.3.15
Adam Optimizer	1.3.16
Adaptive Learning Rates	1.3.17
Addressing Multicollinearity	1.3.18
Addressing_Multicollinearity.py	1.3.19
Adjusted R squared	1.3.20
Agent Based Modelling	1.3.21
Agentic Solutions	1.3.22
Aggregation	1.3.23
Algorithms	1.3.24
Alternatives to Batch Processing	1.3.25
Amazon S3	1.3.26
Anomaly Detection in Time Series	1.3.27
Anomaly Detection with Clustering	1.3.28
Anomaly Detection with Statistical Methods	1.3.29
Anomaly Detection	1.3.30
Apache Airflow	1.3.31
Apache Kafka	1.3.32
Apache Spark	1.3.33
Asking questions	1.3.34

Introduction

Attention Is All You Need	1.3.35
Attention mechanism	1.3.36
Automated Feature Creation	1.3.37
Azure	1.3.38
B	1.4
B tree	1.4.1
BERT Pretraining of Deep Bidirectional Transformers for Language Understanding	1.4.2
BERT	1.4.3
BERTScore	1.4.4
Backpropagation	1.4.5
Bag of words	1.4.6
Bagging	1.4.7
Bandit_Example_Fixed.py	1.4.8
Bandit_Example_Nonfixed.py	1.4.9
Batch Normalisation	1.4.10
Batch Processing	1.4.11
Bellman Equations	1.4.12
Bernoulli	1.4.13
Bias and variance	1.4.14
Big Data	1.4.15
Big O Notation	1.4.16
BigQuery	1.4.17
Binary Classification	1.4.18
Binder	1.4.19
Boosting	1.4.20
Bootstrap	1.4.21
Boxplot	1.4.22
Business observability	1.4.23
business intelligence	1.4.24
C	1.5
CI CD	1.5.1
CRUD	1.5.2
Career Interest	1.5.3
Causal Inference	1.5.4
CatBoost	1.5.5
Central Limit Theorem	1.5.6
Chain of thought	1.5.7
Change Management	1.5.8
Checksum	1.5.9
Choosing a Threshold	1.5.10

Introduction

Choosing the Number of Clusters	1.5.11
Class Separability	1.5.12
Classification Report	1.5.13
Classification	1.5.14
Claude	1.5.15
Click_Implementation.py	1.5.16
Cloud Providers	1.5.17
Clustering	1.5.18
Clustering_Dashboard.py	1.5.19
Code Diagrams	1.5.20
Columnar Storage	1.5.21
Command Line	1.5.22
Command Prompt	1.5.23
Common Security Vulnerabilities in Software Development	1.5.24
Common Table Expression	1.5.25
Communication Techniques	1.5.26
Communication principles	1.5.27
Comparing LLM	1.5.28
Components of the database	1.5.29
Computer Science	1.5.30
Concatenate	1.5.31
Conceptual Model	1.5.32
Concurrency	1.5.33
Confidence Interval	1.5.34
Confusion Matrix	1.5.35
Continuous Delivery Deployment	1.5.36
Continuous Integration	1.5.37
Converting categorical variables to a dummy indicators	1.5.38
Convolutional Neural Networks	1.5.39
Correlation vs Causation	1.5.40
Correlation	1.5.41
Cosine Similarity	1.5.42
Cost Function	1.5.43
Covariance	1.5.44
Covering Index	1.5.45
Cron jobs	1.5.46
Cross Entropy	1.5.47
Cross Validation	1.5.48
Cross_Entropy.py	1.5.49
Cross_Entropy_Single.py	1.5.50

Introduction

Crosstab	1.5.51
Cryptography	1.5.52
Current challenges within the energy sector	1.5.53
cleaning terminal path	1.5.54
conceptual data model	1.5.55
D	1.6
DBScan	1.6.1
Dash	1.6.2
Dashboarding	1.6.3
Data AI Education at Work	1.6.4
Data Analysis Portal	1.6.5
Data Analysis	1.6.6
Data Analyst	1.6.7
Data Architect	1.6.8
Data Archive Graph Analysis	1.6.9
Data Cleansing	1.6.10
Data Collection	1.6.11
Data Distribution	1.6.12
Data Drift	1.6.13
Data Engineer	1.6.14
Data Engineering Portal	1.6.15
Data Engineering Tools	1.6.16
Data Engineering	1.6.17
Data Governance	1.6.18
Data Hierarchy of Needs	1.6.19
Data Ingestion	1.6.20
Data Integration	1.6.21
Data Integrity	1.6.22
Data Lake	1.6.23
Data Lakehouse	1.6.24
Data Leakage	1.6.25
Data Lifecycle Management	1.6.26
Data Management	1.6.27
Data Modelling	1.6.28
Data Observability	1.6.29
Data Pipeline to Data Products	1.6.30
Data Pipeline	1.6.31
Data Principles	1.6.32
Data Product	1.6.33
Data Quality	1.6.34

Introduction

Data Reduction	1.6.35
Data Roles	1.6.36
Data Science	1.6.37
Data Scientist	1.6.38
Data Selection in ML	1.6.39
Data Selection	1.6.40
Data Steward	1.6.41
Data Storage	1.6.42
Data Streaming	1.6.43
Data Terms	1.6.44
Data Transformation with Pandas	1.6.45
Data Transformation	1.6.46
Data Validation	1.6.47
Data Virtualization	1.6.48
Data Visualisation	1.6.49
Data Warehouse	1.6.50
Data transformation in Data Engineering	1.6.51
Data transformation in Machine Learning	1.6.52
Database Index	1.6.53
Database Management System (DBMS)	1.6.54
Database Schema	1.6.55
Database Storage	1.6.56
Database Techniques	1.6.57
Database	1.6.58
Databricks vs Snowflake	1.6.59
Databricks	1.6.60
Datasets	1.6.61
Debugging ipynb	1.6.62
Debugging	1.6.63
Debugging.py	1.6.64
Decision Tree	1.6.65
Deep Learning Frameworks	1.6.66
Deep Learning	1.6.67
Deep Q Learning	1.6.68
DeepSeek	1.6.69
Deleting rows or filling them with the mean is not always best	1.6.70
Demand forecasting	1.6.71
Dendrograms	1.6.72
Design Thinking Questions	1.6.73
Determining Threshold Values	1.6.74

Introduction

DevOps	1.6.75
Difference between Databricks vs. Snowflake	1.6.76
Difference between snowflake to hadoop	1.6.77
Differentation	1.6.78
Digital Transformation	1.6.79
Digital twin	1.6.80
Dimension Table	1.6.81
Dimensional Modelling	1.6.82
Dimensionality Reduction	1.6.83
Dimensions	1.6.84
Directed Acyclic Graph (DAG)	1.6.85
Directory Structure	1.6.86
Distillation	1.6.87
Distributed Computing	1.6.88
Distribution_Analysis.py	1.6.89
Distributions	1.6.90
Docker Image	1.6.91
Docker	1.6.92
Dropout	1.6.93
DuckDB in python	1.6.94
DuckDB vs SQLite	1.6.95
DuckDB	1.6.96
dagster	1.6.97
data asset	1.6.98
data lineage	1.6.99
data literacy	1.6.100
dbt	1.6.101
declarative	1.6.102
dependency manager	1.6.103
E	1.7
EDA	1.7.1
EDA_Pandas.py	1.7.2
ELT	1.7.3
ETL Pipeline Example	1.7.4
ETL vs ELT	1.7.5
ETL	1.7.6
Edge Machine Learning Models	1.7.7
Education and Training	1.7.8
Elastic Net	1.7.9
Embedded Methods	1.7.10

Introduction

Encoding Categorical Variables	1.7.11
Energy Storage	1.7.12
Energy	1.7.13
Environment Variables	1.7.14
Epoch	1.7.15
Epub	1.7.16
Estimator	1.7.17
EtLT	1.7.18
Evaluating Language Models	1.7.19
Evaluation Metrics	1.7.20
Event Driven Events	1.7.21
Event Driven Microservices	1.7.22
Event Driven	1.7.23
Event Driven Architecture	1.7.24
Everything	1.7.25
Excel & Sheets	1.7.26
Explain different gradient descent algorithms, their advantages, and limitations.	1.7.27
Explain the curse of dimensionality	1.7.28
Exploration vs. Exploitation	1.7.29
Exploration	1.7.30
embeddings for OOV words	1.7.31
emergent behavior	1.7.32
F	1.8
FAISS	1.8.1
Fabric	1.8.2
Fact Table	1.8.3
Factor Analysis	1.8.4
Factor_Analysis.py	1.8.5
Facts	1.8.6
FastAPI	1.8.7
FastAPI_Example.py	1.8.8
Feature Engineering	1.8.9
Feature Evaluation	1.8.10
Feature Extraction	1.8.11
Feature Importance	1.8.12
Feature Scaling	1.8.13
Feature Selection vs Feature Importance	1.8.14
Feature Selection	1.8.15
Feature selection and creation	1.8.16
Feature_Distribution.py	1.8.17

Introduction

Feed Forward Neural Network	1.8.18
Feedback Template	1.8.19
Filter method	1.8.20
Firebase	1.8.21
Fishbone diagram	1.8.22
Fitting weights and biases of a neural network	1.8.23
Folder Tree Diagram	1.8.24
Forecasting_AutoArima.py	1.8.25
Forecasting_Baseline.py	1.8.26
Forecasting_Exponential_Smoothing.py	1.8.27
Foreign Key	1.8.28
Forward Propagation	1.8.29
Fuzzywuzzy	1.8.30
filter methods	1.8.31
functional programming	1.8.32
G	1.9
GIS	1.9.1
GRU	1.9.2
GSheets	1.9.3
Gaussian Distribution	1.9.4
Gaussian Mixture Models	1.9.5
Gaussian_Mixture_Model_Implementation.py	1.9.6
General Linear Regression	1.9.7
Generative AI	1.9.8
Generative Adversarial Networks	1.9.9
Get data	1.9.10
Gini Impurity vs Cross Entropy	1.9.11
Gini Impurity	1.9.12
Git	1.9.13
Gitlab	1.9.14
Google Cloud Platform	1.9.15
Google My Maps Data Extraction	1.9.16
Gradient Boosting Regressor	1.9.17
Gradient Boosting	1.9.18
Gradient Descent	1.9.19
Gradio	1.9.20
Grain	1.9.21
Grammar method	1.9.22
Graph Analysis Plugin	1.9.23
Graph Neural Network	1.9.24

Introduction

Graph Theory Community	1.9.25
Graph Theory	1.9.26
GraphRAG	1.9.27
Grep	1.9.28
GridSearchCV	1.9.29
Groupby vs Crosstab	1.9.30
Groupby	1.9.31
Grouped plots	1.9.32
Guardrails	1.9.33
gitlab ci.yml	1.9.34
granularity	1.9.35
H	1.10
Hadoop	1.10.1
Handling Different Distributions	1.10.2
Handling_Missing_Data.ipynb	1.10.3
Handling_Missing_Data_Basic.ipynb	1.10.4
Handwritten Digit Classification	1.10.5
Hash	1.10.6
Heatmap	1.10.7
Heatmaps_Dendrograms.py	1.10.8
Hierarchical Clustering	1.10.9
High cross validation accuracy is not directly proportional to performance on unseen test data	1.10.10
Honkit	1.10.11
Hosting	1.10.12
How LLMs store facts	1.10.13
How businesses use Gen AI	1.10.14
How do we evaluate of LLM Outputs	1.10.15
How is reinforcement learning being combined with deep learning	1.10.16
How is schema evolution done in practice with SQL	1.10.17
How to model to improve demand forecasting	1.10.18
How to normalise a merged table	1.10.19
How to reduce the need for Gen AI responses	1.10.20
How to use Sklearn Pipeline	1.10.21
How would you decide between using TF IDF and Word2Vec for text vectorization	1.10.22
Hugging Face	1.10.23
Hyperparameter Tuning	1.10.24
Hyperparameter	1.10.25
Hypothesis testing	1.10.26
heterogeneous features	1.10.27
how do you do the data selection	1.10.28

Introduction

I		1.11
Imbalanced Datasets		1.11.1
Imbalanced_Datasets_SMOTE.py		1.11.2
Immutable vs mutable		1.11.3
Impact of multicollinearity on model parameters		1.11.4
Implementing Database Schema		1.11.5
In NER how would you handle ambiguous entities		1.11.6
Industries of interest		1.11.7
Input is Not Properly Sanitized		1.11.8
Interpreting logistic regression model parameters		1.11.9
Interquartile Range (IQR) Detection		1.11.10
Isolated Forest		1.11.11
imperative		1.11.12
in memory format		1.11.13
incremental synchronization		1.11.14
inference versus prediction		1.11.15
inference		1.11.16
information theory		1.11.17
interoperable		1.11.18
interpretability		1.11.19
interview notepad		1.11.20
ipynb		1.11.21
J		1.12
Java vs JavaScript		1.12.1
JavaScript		1.12.2
Json to Yaml		1.12.3
Json		1.12.4
Junction Tables		1.12.5
jinja template		1.12.6
K		1.13
K means		1.13.1
K nearest neighbours		1.13.2
K_Means.py		1.13.3
Kaggle Abalone regression example		1.13.4
Kerneling		1.13.5
Knowledge Graph		1.13.6
Knowledge Graphs with Obsidian		1.13.7
Knowledge Work		1.13.8
Knowledge graph vs RAG setup		1.13.9
kubernetes		1.13.10

Introduction

L	1.14
LBFGS	1.14.1
LLM Evaluation Metrics	1.14.2
LLM	1.14.3
LSTM	1.14.4
Label encoding	1.14.5
Labelling data	1.14.6
Langchain	1.14.7
Language Model Output Optimisation	1.14.8
Language Models	1.14.9
Lasso	1.14.10
Latency	1.14.11
Latent Dirichlet Allocation	1.14.12
Learning Styles	1.14.13
LightGBM vs XGBoost vs CatBoost	1.14.14
LightGBM	1.14.15
Linear Discriminant Analysis	1.14.16
Linear Regression	1.14.17
Linked List	1.14.18
Load Balancing	1.14.19
Local Interpretable Model agnostic Explanations	1.14.20
Logical Model	1.14.21
Logistic Regression Statsmodel Summary table	1.14.22
Logistic Regression does not predict probabilities	1.14.23
Logistic Regression	1.14.24
Looker Studio	1.14.25
Loss function	1.14.26
Loss versus Cost function	1.14.27
lambda architecture	1.14.28
learning rate	1.14.29
lemmatization	1.14.30
M	1.15
ML Engineer	1.15.1
MNIST	1.15.2
Machine Learning Algorithms	1.15.3
Machine Learning Operations	1.15.4
Machine Learning	1.15.5
Maintainable Code	1.15.6
Makefile	1.15.7
Manifold Learning	1.15.8

Introduction

Markov Decision Processes	1.15.9
Markov chain	1.15.10
Master Observability Datadog	1.15.11
Mathematical Reasoning in Transformers	1.15.12
Mathematics	1.15.13
Maximum Likelihood Estimation	1.15.14
Mean Squared Error	1.15.15
Melt	1.15.16
Memory Caching	1.15.17
Memory	1.15.18
Merge	1.15.19
Metadata Handling	1.15.20
Metric	1.15.21
Microsoft Access	1.15.22
Mini batch gradient descent	1.15.23
Mixture of Experts	1.15.24
Model Building	1.15.25
Model Cascading	1.15.26
Model Deployment	1.15.27
Model Ensemble	1.15.28
Model Evaluation vs Model Optimisation	1.15.29
Model Evaluation	1.15.30
Model Interpretability	1.15.31
Model Observability	1.15.32
Model Optimisation	1.15.33
Model Parameters Tuning	1.15.34
Model Parameters	1.15.35
Model Selection	1.15.36
Model Validation	1.15.37
Model parameters vs hyperparameters	1.15.38
Model preparation	1.15.39
Momentum	1.15.40
Momentum.py	1.15.41
MongoDB	1.15.42
Monolith Architecture	1.15.43
Monte Carlo Simulation	1.15.44
Multi Agent Reinforcement Learning	1.15.45
Multi head attention	1.15.46
Multi level index	1.15.47
Multicollinearity	1.15.48

Introduction

Multinomial Naive bayes	1.15.49
MySQL	1.15.50
maintainability	1.15.51
map reduce	1.15.52
master data management	1.15.53
mean absolute error	1.15.54
N	1.16
NLP	1.16.1
Naive Bayes	1.16.2
Named Entity Recognition	1.16.3
Network Design	1.16.4
Neural Network Classification	1.16.5
Neural Scaling Laws	1.16.6
Neural network in Practice	1.16.7
Neural network	1.16.8
Ngrams	1.16.9
NoSQL	1.16.10
Node.JS	1.16.11
Non parametric tests	1.16.12
Normalisation of Text	1.16.13
Normalisation of data	1.16.14
Normalisation vs Standardisation	1.16.15
Normalisation	1.16.16
Normalised Schema	1.16.17
NotebookLM	1.16.18
nbconvert	1.16.19
neo4j	1.16.20
nltk	1.16.21
npy Files A NumPy Array storage	1.16.22
O	1.17
OLAP (online analytical processing)	1.17.1
OLAP	1.17.2
OLTP	1.17.3
One hot encoding	1.17.4
One_hot_encoding.py	1.17.5
Optimisation function	1.17.6
Optimisation techniques	1.17.7
Optimising Neural Networks	1.17.8
Optuna	1.17.9
Ordinary Least Squares	1.17.10

Introduction

Orthogonalization	1.17.11
Outliers	1.17.12
Over parameterised models	1.17.13
Overfitting	1.17.14
oltp (online transactional processing)	1.17.15
P	1.18
PCA Explained Variance Ratio	1.18.1
PCA Principal Components	1.18.2
PCA Based Anomaly Detection	1.18.3
PCA_Analysis.ipynb	1.18.4
PCA_Based_Anomaly_Detection.py	1.18.5
PDF++	1.18.6
PDP and ICE	1.18.7
Page Rank	1.18.8
Pandas Dataframe Agent	1.18.9
Pandas Pivot Table	1.18.10
Pandas Stack	1.18.11
Pandas join vs merge	1.18.12
Pandas	1.18.13
Pandas_Common.py	1.18.14
Pandas_Stack.py	1.18.15
Parametric tests	1.18.16
Parquet	1.18.17
Part of speech tagging	1.18.18
Percentile Detection	1.18.19
Performance Dimensions	1.18.20
Performance Drift	1.18.21
Physical Model	1.18.22
Plotly	1.18.23
Poetry	1.18.24
Policy	1.18.25
Positional Encoding	1.18.26
PostgreSQL	1.18.27
PowerBI	1.18.28
PowerShell	1.18.29
Powerquery	1.18.30
Powershell versus cmd	1.18.31
Powershell vs Bash	1.18.32
Precision or Recall	1.18.33
Precision Recall Curve	1.18.34

Introduction

Precision	1.18.35
Prediction Intervals	1.18.36
Preprocessing	1.18.37
Prevention Is Better Than The Cure	1.18.38
Primary Key	1.18.39
Principal Component Analysis	1.18.40
Probability in other fields	1.18.41
Problem Definition	1.18.42
Prompt Engineering	1.18.43
Publish and Subscribe	1.18.44
Push Down	1.18.45
PyCaret	1.18.46
PyGraphviz	1.18.47
PySpark	1.18.48
PyTorch	1.18.49
Pycaret_Anomaly.ipynb	1.18.50
Pycaret_Example.py	1.18.51
Pydantic	1.18.52
Pydantic.py	1.18.53
Pydantic_More.py	1.18.54
Pyright vs Pydantic	1.18.55
Pyright	1.18.56
Python Click	1.18.57
Python	1.18.58
Pytorch vs Tensorflow	1.18.59
p values	1.18.60
p values in linear regression in sklearn	1.18.61
parametric vs non parametric models	1.18.62
parametric vs non parametric tests	1.18.63
parsimonious	1.18.64
pd.Grouper	1.18.65
pdoc	1.18.66
pmdarima	1.18.67
programming languages	1.18.68
Q	1.19
Q Learning	1.19.1
QUERY GSheets	1.19.2
Quartz	1.19.3
Query Optimisation	1.19.4
Query Plan	1.19.5

Introduction

Querying	1.19.6
QuickSort	1.19.7
R	1.20
R squared	1.20.1
R squared metric not always a good indicator of model performance in regression	1.20.2
R	1.20.3
RAG	1.20.4
ROC (Receiver Operating Characteristic)	1.20.5
ROC_Curve.py	1.20.6
Race Conditions	1.20.7
Random Forest Regression	1.20.8
Random Forests	1.20.9
React	1.20.10
Reasoning tokens	1.20.11
Recall	1.20.12
Recommender systems	1.20.13
Recurrent Neural Networks	1.20.14
Recursive Algorithm	1.20.15
Regression Metrics	1.20.16
Regression	1.20.17
Regression_Logistic_Metrics.ipynb	1.20.18
Regularisation of Tree based models	1.20.19
Regularisation	1.20.20
Regularisation.py	1.20.21
Reinforcement learning	1.20.22
Relating Tables Together	1.20.23
Relational Database	1.20.24
Relationships in memory	1.20.25
Reward Function	1.20.26
Ridge	1.20.27
Row based Storage	1.20.28
requirements.txt	1.20.29
reverse etl	1.20.30
rollup	1.20.31
S	1.21
SHapley Additive exPlanations	1.21.1
SMOTE (Synthetic Minority Over sampling Technique)	1.21.2
SMSS	1.21.3
SQL Groupby	1.21.4
SQL Injection	1.21.5

Introduction

SQL Joins	1.21.6
SQL Window functions	1.21.7
SQL vs NoSQL	1.21.8
SQL	1.21.9
SQLAlchemy	1.21.10
SQLite Studio	1.21.11
SQLite	1.21.12
SVM_Example.py	1.21.13
Sarsa	1.21.14
Scala	1.21.15
Scalability	1.21.16
Scaling Agentic Systems	1.21.17
Scaling Server	1.21.18
Scheduled Tasks	1.21.19
Schema Evolution	1.21.20
Scientific Method	1.21.21
Seaborn	1.21.22
Search	1.21.23
Security	1.21.24
Semantic Relationships	1.21.25
Sentence Similarity	1.21.26
Sharepoint	1.21.27
Silhouette Analysis	1.21.28
Similarity Search	1.21.29
Single Source of Truth	1.21.30
Sklearn Pipeline	1.21.31
Sklearn	1.21.32
Slowly Changing Dimension	1.21.33
Small Language Models	1.21.34
Smart Grids	1.21.35
Snowflake Schema	1.21.36
Soft Deletion	1.21.37
Software Design Patterns	1.21.38
Software Development Life Cycle	1.21.39
SparseCategoricalCrossentropy or CategoricalCrossEntropy	1.21.40
Specificity	1.21.41
Spreadsheets vs Databases	1.21.42
Stacking	1.21.43
Standard deviation	1.21.44
Standardisation	1.21.45

Introduction

Star Schema	1.21.46
Statistical Assumptions	1.21.47
Statistical Tests	1.21.48
Statistics	1.21.49
Stemming	1.21.50
Stochastic Gradient Descent	1.21.51
Stored Procedures	1.21.52
Strongly vs Weakly typed language	1.21.53
Structuring and organizing data	1.21.54
Summarisation	1.21.55
Supervised Learning	1.21.56
Support Vector Classifier (SVC)	1.21.57
Support Vector Machines	1.21.58
Support Vector Regression	1.21.59
Symbolic computation	1.21.60
Sympy	1.21.61
semantic layer	1.21.62
semi structured data	1.21.63
shapefile	1.21.64
sklearn datasets	1.21.65
spaCy	1.21.66
storage layer object store	1.21.67
structured data	1.21.68
syntactic relationships	1.21.69
T	1.22
T test	1.22.1
TF IDF	1.22.2
TOML	1.22.3
TS_Anomaly_Detection	1.22.4
TS_Anomaly_Detection.py	1.22.5
Tableau	1.22.6
Tags	1.22.7
Technical Debt	1.22.8
Telecommunications	1.22.9
Tensorflow	1.22.10
Terminal commands	1.22.11
Test Loss When Evaluating Models	1.22.12
Testing	1.22.13
Testing_Pytest.py	1.22.14
Testing_unittest.py	1.22.15

Introduction

Text2Cypher	1.22.16
Thinking Systems	1.22.17
Time Series Forecasting	1.22.18
Time Series Identify Trends and Patterns	1.22.19
Time Series	1.22.20
Tokenisation	1.22.21
Train Dev Test Sets	1.22.22
Transaction	1.22.23
Transfer Learning	1.22.24
Transformed Target Regressor	1.22.25
Transformer	1.22.26
Transformers vs RNNs	1.22.27
Turning a flat file into a database	1.22.28
TypeScript	1.22.29
Types of Computational Bugs	1.22.30
Types of Database Schema	1.22.31
Types of Neural Networks	1.22.32
Typical Output Formats in Neural Networks	1.22.33
t SNE	1.22.34
tool.ruff	1.22.35
tool.uv	1.22.36
topic modeling	1.22.37
transfer_learning.py	1.22.38
U	1.23
UML	1.23.1
Ubuntu	1.23.2
Unsupervised Learning	1.23.3
Untitled 1	1.23.4
Untitled 2	1.23.5
Untitled	1.23.6
Use Cases for a Simple Neural Network Like	1.23.7
Use of RNNs in energy sector	1.23.8
Utilities	1.23.9
unittest	1.23.10
univariate vs multivariate	1.23.11
unstructured data	1.23.12
V	1.24
Vacuum	1.24.1
Variance	1.24.2
Vector Embedding	1.24.3

Introduction

Vector_EMBEDDING.py	1.24.4
Vectorisation	1.24.5
Vercel	1.24.6
Views	1.24.7
Violin plot	1.24.8
Virtual environments	1.24.9
vanishing and exploding gradients problem	1.24.10
W	1.25
WCSS and elbow method	1.25.1
Weak Learners	1.25.2
Web Feature Server (WFS)	1.25.3
Web Map Tile Service (WMPS)	1.25.4
Webpages relevant	1.25.5
What algorithms or models are used within the energy sector	1.25.6
What algorithms or models are used within the telecommunication sector	1.25.7
What are the best practices for evaluating the effectiveness of different prompts	1.25.8
What can ABM solve within the energy sector	1.25.9
What is the difference between odds and probability	1.25.10
What is the role of gradient based optimization in training deep learning models.	1.25.11
When and why not to us regularisation	1.25.12
Why JSON is Better than Pickle for Untrusted Data	1.25.13
Why Type 1 and Type 2 matter	1.25.14
Why and when is feature scaling necessary	1.25.15
Why does increasing the number of models in a ensemble not necessarily improve the accuracy	1.25.16
Why does label encoding give different predictions from one hot encoding	1.25.17
Why is named entity recognition (NER) a challenging task	1.25.18
Why is the Central Limit Theorem important when working with small sample sizes	1.25.19
Why use ER diagrams	1.25.20
Wikipedia_API.py	1.25.21
Windows Subsystem for Linux	1.25.22
Word2Vec.py	1.25.23
Word2vec	1.25.24
WordNet	1.25.25
Wrapper Methods	1.25.26
X	1.26
XGBoost	1.26.1
Y	1.27
yaml	1.27.1
Z	1.28
Z Normalisation	1.28.1

Introduction

Z Score	1.28.2
Z Scores vs Prediction Intervals	1.28.3
Z Test	1.28.4

Data Archive Book

Welcome to the Data Archive Book.

Table of Contents

- 1-on-1 Template

1 On 1 Template

- Decisions
 - [Your name] *add decisions that need to be made*
 - [Other person's name] *add decisions that need to be made*
- Action items
 - [Your name] *add next steps from the discussion*
 - [Other person's name] *add next steps from the discussion*
- Topics to discuss (bi-directional)
 - [Your name] *add topics or questions to discuss together*
 - [Other person's name] *add topics or questions to discuss together*
- Updates (uni-directional - no action needed)
 - [Your name] *add updates with no action needed*
 - [Other person's name] *add updates with no action needed*

A

Table of Contents

- AB testing
- ACID Transaction
- AI Engineer
- AI governance
- API Driven Microservices
- API
- ARIMA
- AUC
- AWS Lambda
- Accessing Gen AI generated content
- Accuracy
- Activation Function
- Activation atlases
- Active Learning
- Ada boosting
- Adam Optimizer
- Adaptive Learning Rates
- Adding a database to PostgreSQL
- Addressing Multicollinearity
- Addressing_Multicollinearity.py
- Adjusted R squared
- Agent-Based Modelling
- Agentic Solutions
- Aggregation
- Algorithms
- Alternatives to Batch Processing
- Amazon S3
- Anomaly Detection in Time Series
- Anomaly Detection with Clustering
- Anomaly Detection with Statistical Methods
- Anomaly Detection
- Apache Airflow
- Apache Kafka
- Apache Spark
- Asking questions
- Attention Is All You Need
- Attention mechanism
- Automated Feature Creation
- Azure

Ab Testing

A/B testing is a method of performance testing two versions of a product like an app.

Acid Transaction

An ACID [Transaction](#) ensures that either all changes are successfully committed or rolled back, preventing the database from ending up in an inconsistent state. This guarantees the integrity of the data throughout the transaction process.

Key Properties of ACID Transactions

1. Atomicity: This property ensures that transactions are treated as a single, indivisible unit. If any part of the transaction fails, the entire transaction is rolled back, and none of the changes are applied. Users do not see intermediate states of the transaction.
2. Consistency: Transactions must leave the database in a valid state, adhering to all defined constraints. If a transaction violates a constraint, it is rolled back to maintain the database's stable state.
3. Isolation: This property ensures that concurrent transactions do not interfere with each other. Each transaction operates independently, and the results of one transaction are not visible to others until it is committed.
4. Durability: Once a transaction has been committed, the changes are permanent, even in the event of a system failure. The data remains intact and recoverable.

Ai Engineer

They know what

- [LSTM](#) means
- [Attention mechanism](#)
- [Prompting optimisation](#)
- [Neural network](#)

Ai Governance

AI Governance

[Data Governance](#)

Used in regulated sectors.

Constraints to using ai:

- legal,
- transparency,
- security,
- historical bias

AI acts and standards:

- eu and AI acts
- NIST standards framework
- Security OWASP standards for LLM

how will beaurcacy keeps up with ai innovation.

Governance can stifle innovation.

Api Driven Microservices

API-driven microservices refer to a [software architecture](#) approach where [microservices](#) communicate with each other and with external systems primarily through well-defined [API](#) (Application Programming Interfaces).

This architecture is designed to enhance modularity, scalability, and flexibility by breaking down an application into smaller, independent services that can be developed, deployed, and scaled independently.

API-driven microservices architecture is particularly beneficial for large, complex applications that require frequent updates and scaling. It allows organizations to innovate faster, improve fault isolation, and better align development efforts with business needs. However, it also introduces complexity in terms of service orchestration, data consistency, and network communication, which must be carefully managed.

Key characteristics of API-driven microservices include:

1. **Decoupled Services:** Each microservice is a separate, self-contained unit that performs a specific business function. Services are loosely coupled, meaning changes to one service do not directly impact others.
2. **API Communication:** Microservices interact with each other and with external clients through APIs. These APIs are typically RESTful, but they can also use other protocols like gRPC, GraphQL, or messaging systems like Kafka.
3. **Independent Deployment:** Each microservice can be developed, tested, deployed, and scaled independently of the others. This allows for more agile development and continuous deployment practices.
4. **Technology Agnostic:** Different microservices can be built using different technologies or programming languages, as long as they adhere to the agreed-upon API contracts.
5. **Scalability and Resilience:** Microservices can be scaled independently based on demand. If one service fails, it does not necessarily bring down the entire system, enhancing resilience.
6. **Focused Functionality:** Each microservice is designed to handle a specific business capability, making it easier to understand, develop, and maintain.
7. **API Gateway:** Often, an API gateway is used to manage and route requests to the appropriate microservices. It can also handle cross-cutting concerns like authentication, logging, and rate limiting.

Api

An API (Application Programming Interfaces) allows one system (client) to [request specific actions from another system](#) (server).

Using a predefined set of rules and [protocols](#).

Good API documentation is necessary for developers to integrate and use APIs effectively.

Resources:

- [Link](#)
- [REST API](#)
- [FastAPI](#)

API Principles

- **Controlled Access:** APIs provide access to certain parts of a system while keeping the core functionalities secure.
- **System Independence:** APIs function independently of changes in the underlying system.
- **Simplicity:** APIs are designed to be **user-friendly** and come with comprehensive documentation to guide developers.

Implementation

In [ML_Tools](#) see: [Wikipedia_API.py](#)

Example:

For instance, a weather app querying a weather API to fetch the current weather conditions involves sending a structured request and receiving a response.

Types of API Connections:

1. **Web APIs:** These facilitate communication between web clients (browsers or apps) and servers. For example, online shopping apps use APIs to process transactions on remote servers.
2. **Database APIs:** These allow applications to interact with databases, ensuring data is accessed and manipulated efficiently.
3. **Device APIs:** When apps like Instagram or WhatsApp request access to your phone's camera or microphone, they use device APIs.

Arima

ARIMA (AutoRegressive Integrated Moving Average) is a popular method for [Time Series Forecasting](#) that models the autocorrelations within the data. It is particularly useful for datasets with trends and patterns that are not seasonal. However, it is not perfect; for example, it struggles with predicting stock trading data.

ARIMA Components

- **AR (AutoRegressive):** Utilizes the dependency between an observation and a number of lagged observations.
- **I (Integrated):** Involves differencing the data to achieve stationarity.
- **MA (Moving Average):** Models the dependency between an observation and a residual error from a moving average model applied to lagged observations.

ARIMA Explained

ARIMA stands for:

- **AutoRegressive (AR):** Uses past values to predict the current one.
- **Integrated (I):** Differencing to make the series stationary.
- **MAving Average (MA):** Uses past forecast errors for prediction.

A typical **ARIMA(p,d,q)** model has:

- **\$p\$:** Number of **lagged values** (AR terms)
- **\$d\$:** Number of **differences** needed to make the series stationary
- **\$q\$:** Number of **lagged forecast errors** (MA terms)

What ARIMA Does:

1. **Checks for stationarity** – if not, applies differencing (d times).
2. **Models relationships** between current values and:
 - o Past values (AR part)
 - o Past errors (MA part)
3. **Fits** parameters by minimizing a loss (typically log-likelihood).
4. **Forecasts** future values using this learned structure.

Why ARIMA Isn't Enough for Seasonal Data

Your quarterly data might show **repeating patterns every 4 quarters** (seasonality), and ARIMA has **no built-in mechanism to model this periodic structure**. This is where **SARIMA** comes in.

SARIMA: Seasonal ARIMA

SARIMA extends ARIMA by adding **seasonal components**. It is written as:

$$\text{SARIMA}(p, d, q)(P, D, Q)_s$$

Where:

- (p, d, q) are **non-seasonal** ARIMA parameters (as above).
- $(P, D, Q)_s$ are **seasonal** ARIMA parameters:
 - o P : Seasonal autoregressive terms.
 - o D : Seasonal differences.
 - o Q : Seasonal moving average terms.
 - o s : Seasonality period (e.g., $s=4$ for quarterly data).

What SARIMA Adds:

- **Seasonal differencing** (D): e.g., subtract the value from 4 quarters ago to remove annual cycles.
- **Seasonal AR/MA terms**: Model relationships from past seasonal lags and errors.

Summary

Model	Handles Trend	Handles Seasonality	Use When...
ARIMA			Data has trend, but no regular cycles
SARIMA			Data has both trend and seasonality

Advanced Variants

SARIMAX (Seasonal ARIMA with Exogenous Variables)

- Incorporates external variables (e.g., interest rates, volume) into the forecasting model.
- Useful for [Datasets](#) where external factors influence the time series.

Related terms

In [ML_Tools](#) see:

- https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/TimeSeries/ARIMA.ipynb
 - Forecasting_AutoArima.py
 - pmdarima
-

AUC

AUC (Area Under the Curve) is a metric for binary classification problems, representing the area under the **ROC (Receiver Operating Characteristic)**

Key Concepts

Represents the area under the ROC curve.

AUC values range from 0 to 1, where 1 indicates perfect classification and 0.5 suggests no discriminative power (equivalent to random guessing).

Roc and Auc Score

The `roc_auc_score` is a function from the `sklearn.metrics` module in Python that computes the Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores. It is a widely used metric for evaluating the performance of binary classification models.

Key Points about `roc_auc_score` :

- **Purpose:** It quantifies the overall ability of the model to discriminate between the positive and negative classes across all possible classification thresholds.
- **Range:** The score ranges from 0 to 1, where:
 - 1 indicates perfect discrimination (the model perfectly distinguishes between the positive and negative classes).
 - 0.5 suggests no discriminative power (equivalent to random guessing).
 - Values below 0.5 indicate a model that performs worse than random guessing.
- **Input:** The function takes the true binary labels and the predicted probabilities (or decision function scores) as inputs.
- **Output:** It returns a single scalar value representing the AUC.

Example Code

```
from sklearn.metrics import roc_auc_score

# Actual and predicted values
y_act = [1, 0, 1, 1, 0]
y_pred = [1, 1, 0, 1, 0]

# Compute AUC
auc = roc_auc_score(y_act, y_pred)
print(f'AUC: {auc}')
```

Aws Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS) that allows you to run code without provisioning or managing servers.

AWS Lambda is a powerful tool for building scalable, event-driven applications without the overhead of managing server infrastructure.

With AWS Lambda, you can execute your code in response to various events, such as HTTP requests via Amazon API Gateway, changes to data in an Amazon S3 bucket, updates to a DynamoDB table, or messages arriving in an Amazon SQS queue.

Key features of AWS Lambda include:

1. **Event Driven Events:** AWS Lambda functions are triggered by events, which can come from a wide range of AWS services or custom applications.
2. **Automatic Scaling:** Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.
3. **Pay-as-You-Go:** You are charged based on the number of requests for your functions and the time your code executes. This means you only pay for the compute time you consume.
4. **No Server Management:** AWS Lambda abstracts the underlying infrastructure, so you don't need to manage servers, patch operating systems, or worry about scaling.
5. **Supports Multiple Languages:** AWS Lambda supports several programming languages, including Python, Java, Node.js, C#, Ruby, and Go, among others.
6. **Integration with AWS Services:** Lambda integrates seamlessly with other AWS services, allowing you to build complex, scalable applications.

Here's a simple example of how AWS Lambda might be used:

- You have an [S3 bucket](#) where users upload images.
- An AWS Lambda function is triggered whenever a new image is uploaded.
- The Lambda function processes the image, such as generating thumbnails or extracting metadata.
- The processed data is then stored back in S3 or sent to another AWS service for further processing.

Accessing Gen Ai Generated Content

To assess whether the content generated by a [Generative AI](#) is truthful and faithful, several methods and frameworks can be employed. Truthfulness refers to whether the generated content **is factually correct**, while faithfulness refers to whether it **accurately** reflects the input data or prompt.

[interpretability](#)

1. Frameworks for Truthfulness and Faithfulness

- Subject Matter Expert (SME) Reviews: One of the most reliable methods for verifying truthfulness and faithfulness is through SME validation. SMEs can manually check the content to ensure it aligns with domain-specific knowledge and is factually accurate.

- [Knowledge Graph](#) and External Data: Generative AI models can be linked to external sources of truth, such as knowledge graphs, databases, or other verified resources. This allows the system to cross-check facts and improve the truthfulness of the content.
- Retrieval-Augmented Generation ([RAG](#)): This framework involves retrieving relevant information from trusted sources before generating content. It helps ensure that the AI is providing up-to-date, reliable, and contextually accurate responses.
- Evaluation Metrics: Some metrics can be used to evaluate faithfulness:
 - Factual Consistency Metrics: Tools such as [BERTScore](#) or FactCC can compare generated text with reference text or factual databases to check for consistency.
 - Human Evaluation: In certain contexts, human evaluators rate the content on aspects of truthfulness and faithfulness. This can be part of quality assurance processes.
- Cross-Referencing Data: AI-generated content should be cross-referenced with existing, credible sources to confirm its accuracy. For example, if the AI makes a historical claim or provides statistical data, those facts should be verifiable through known data repositories.
- Fact-Checking Tools: Using automated fact-checking tools or models trained to detect false information can provide another layer of defence against untruthful content.

Accuracy

Definition

- Accuracy Score is the proportion of correct predictions out of all predictions made. In other words, it is the percentage of correct predictions.
- Accuracy can have issues with [Imbalanced Datasets](#) where there is more of one class than another.

Formula

- The formula for accuracy is: $\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{Total}}$ In the context of [Classification](#) problems, particularly binary classification, TN and TP are components of the confusion matrix:
- TP (True Positive): The number of instances that are correctly predicted as the positive class. For example, if the model predicts a positive outcome and it is indeed positive, it counts as a true positive.
- TN (True Negative): The number of instances that are correctly predicted as the negative class. For example, if the model predicts a negative outcome and it is indeed negative, it counts as a true negative.

The [Confusion Matrix](#) also includes:

- FP (False Positive): The number of instances that are incorrectly predicted as the positive class. This is also known as a "Type I error."
- FN (False Negative): The number of instances that are incorrectly predicted as the negative class. This is also known as a "Type II error."

These metrics are used to evaluate the performance of a classification model, providing insights into not just accuracy but also precision, recall, and other performance measures.

Exploring Accuracy in Python

To explore accuracy in Python, you can use libraries such as `scikit-learn`, which provides the `accuracy_score` function. This function compares the predicted labels with the true labels and calculates the accuracy.

Example Usage

```
from sklearn.metrics import accuracy_score
# Assuming pred and y_test are defined
accuracy = accuracy_score(y_test, pred)
print("Prediction accuracy: {:.2f}%".format(accuracy * 100.0))
```

- Make sure to replace `pred` and `y_test` with your actual prediction and test data variables.

Activation Function

Activation functions play a role in [Neural network](#) by introducing non-linearity, allowing models to learn from complex patterns and relationships in the data.

[How do we choose the right Activation Function](#)

Key Uses of Activation Functions:

1. Non-linearity: Without activation functions, neural networks would behave as linear models, unable to capture complex, non-linear patterns in the data
2. [Data transformation](#): Activation functions modify input signals from one layer to another, helping the model focus on important information while ignoring irrelevant data,
3. [Backpropagation](#): They enable gradient-based optimization by making the network differentiable, essential for efficient learning.

Purpose of Typical Activation Functions

Linear: Outputs a continuous value, suitable for regression.

ReLU (Rectified Linear Unit):

- Purpose: ReLU is used to introduce non-linearity by turning neurons "on" or "off." It outputs the input directly if it is positive; otherwise, it outputs zero. This helps in efficiently training deep networks by mitigating the vanishing gradient problem.
- Function: $f(x) = \max(0, x)$

Sigmoid:

- Purpose: Sigmoid is used primarily in [Binary Classification](#) tasks. It squashes input values to a range between 0 and 1, making it suitable for representing probabilities.
- Function: $f(x) = \frac{1}{1 + e^{-x}}$

Tanh:

- Purpose: Tanh is similar to the sigmoid function but outputs values in the range of -1 to 1. This zero-centered output can be beneficial for optimization in certain scenarios.
- Function: $f(x) = \tanh(x)$

Softmax:

- Purpose: Softmax is used in multi-class classification tasks. It converts a vector of raw scores (logits) into a probability distribution, where each value is between 0 and 1, and the sum of all values is 1. This allows the outputs to be interpreted as probabilities, with larger inputs corresponding to larger output probabilities.
- Application: In both softmax regression and neural networks with softmax outputs, a vector \mathbf{z} is generated by a linear function and then passed through the softmax function to produce a probability distribution. This enables the selection of one output as the predicted category.

The softmax function converts a vector of raw scores (logits) into a probability distribution. The formula for the softmax function for a vector $\mathbf{z} = [z_1, z_2, \dots, z_N]$ is given by:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$$

This ensures that the output values are between 0 and 1 and that they sum to 1, making them [interpretable](#) as probabilities.

Activation Atlases

is a viewing method for high dimensional space that AI system use for predictions.

Example AlexNet (cofounder of OpenAI)

Active Learning

Think captchas for training.

To help the [Supervised Learning](#) models when they are less confident.

Reducing labelling time or need for it.

Ada Boosting

Resources: [LINK](#)

Overview:

Ada Boosting short for [Adaptive Boosting](#), is a specific type of [Boosting](#) algorithm that focuses on improving the accuracy of predictions by combining multiple [weak learners](#) into a strong learner. It is particularly known for its [simplicity](#) and effectiveness in classification tasks.

How AdaBoost Works:

1. **Base Learners:** In AdaBoost, the base learners are typically low-depth trees, also known as [stumps](#). These are simple models that perform slightly better than random guessing.
2. **Sequential Training:** AdaBoost trains these stumps sequentially. Each stump is trained to correct the errors made by the previous stumps. This sequential approach ensures that each new model focuses on the data points that were misclassified by earlier models.

-
- 3. **Weighting:** After each stump is trained, AdaBoost assigns a weight to it based on its accuracy. More accurate stumps receive higher weights, giving them more influence in the final prediction.
 - 4. **Error Focus:** The algorithm increases the weights of the misclassified data points, making them more prominent in the training of the next stump. This ensures that subsequent models pay more attention to the difficult-to-classify instances.
 - 5. **Final Prediction:** The final prediction is a weighted sum of the predictions from all the stumps. The stumps with higher accuracy have more say in the final classification.

Further Understanding

Creating a Forest with AdaBoost:

To create a forest using AdaBoost, you start with a [Decision Tree](#) or [Random Forests](#) approach, but instead of using full-sized trees, you use stumps.

These stumps are trained sequentially, with each one focusing on the errors of the previous stumps.

The final prediction is a weighted sum of the predictions from all the stumps, where more accurate stumps have more influence on the final outcome.

Key Differences from Random Forests:

- **Tree Depth:** In [Random Forests](#), full-sized trees are used, and each tree gets an equal say in the final prediction. In contrast, AdaBoost uses low-depth trees (stumps) and assigns different weights to each stump based on its accuracy.
- **Order and Sequence:** In AdaBoost, the order of the stumps is important because errors are passed on in sequence. In [Random Forests](#), trees are built independently and simultaneously.

Advantages of AdaBoost:

- **Increased Accuracy:** By focusing on the errors of previous models, AdaBoost can significantly improve the accuracy of predictions.
- **Simplicity:** AdaBoost is relatively simple to implement and understand compared to other ensemble methods.
- **Flexibility:** It can be applied to various types of base models and is not limited to a specific algorithm.

Challenges of AdaBoost:

- **Sensitivity to Noisy Data:** AdaBoost can be sensitive to noisy data and outliers, as it focuses heavily on correcting errors.
- **Complexity:** While simpler than some other boosting methods, AdaBoost can still be computationally intensive due to its sequential nature.

Adam Optimizer

Adam (Adaptive Moment Estimation) is an advanced optimization algorithm that combines the benefits of both [Momentum](#) and adaptive learning rates. It is widely used due to its efficiency and effectiveness in training [deep learning](#) models.

Adam is particularly effective for large datasets and complex models, as it provides robust convergence and requires minimal tuning compared to other optimization algorithms. Its ability to **dynamically adjust learning rates** makes it a popular choice in the deep learning community.

Key Features of Adam:

Adaptive Learning Rates: Adam adjusts the **learning rate** for each parameter individually, based on the first and second moments of the gradients. This allows for more precise updates and better convergence.

Momentum and RMSProp Combination: Adam incorporates the concept of momentum by using moving averages of the gradients (first moment) and the squared gradients (**second moment**), similar to RMSProp.

Parameter Update Rule: The update rule involves computing biased estimates of the first and second moments, which are then corrected to provide unbiased estimates. These are used to update the parameters.

Hyperparameter:

- **Learning Rate** (α): Typically set to 0.001, but can be tuned for specific tasks.
- **Beta1 and Beta2**: Control the decay rates for the moving averages of the first and second moments. Common values are 0.9 and 0.999, respectively.
- **Epsilon** (ϵ): A small constant added for numerical stability, usually set to (1×10^{-8}) .

Implementation Challenges:

- **Parameter Tuning:** Careful tuning of learning rate, beta1, and beta2 is essential for optimal performance.
- **Numerical Stability:** Adjusting epsilon can help prevent division by zero and other numerical issues.

Related concepts

- Gradient Descent
- Why does the Adam Optimizer converge
-

Adaptive Learning Rates

Adam Optimizer

Adaptive **learning rate** adjust the learning rate for each parameter based on the estimates of the first and second moments of the gradients. Adam (short for Adaptive Moment Estimation) combines ideas from **Momentum** and adaptive learning rates to help the optimization process.

How to Add a Database to PostgreSQL

Using pgAdmin (GUI)

1. Open pgAdmin and log in.
2. In the Object Explorer, right-click Databases → Create → Database.
3. Enter Database Name (e.g., `mydatabase`).
4. Choose an Owner (optional).
5. Click Save.

Using Python (`psycopg2`)

If you're using Python (e.g., in a Jupyter Notebook), install the `psycopg2` package if needed:

```
!pip install psycopg2-binary
```

Then, run this script to create a PostgreSQL database:

```
import psycopg2

# Connect to the PostgreSQL server (default 'postgres' database)
conn = psycopg2.connect(
    dbname="postgres", # Default DB to connect before creating a new one
    user="postgres",
    password="your_password",
    host="localhost"
)
conn.autocommit = True # Required for CREATE DATABASE
cursor = conn.cursor()

# Create a new database
cursor.execute("CREATE DATABASE mydatabase;")

# Close connection
cursor.close()
conn.close()
print("Database 'mydatabase' created successfully!")
```

Addressing Multicollinearity

In [ML_Tools](#) see: [Addressing_Multicollinearity.py](#)

Multicollinearity can impact the performance and [interpretability](#) of regression models by causing instability in coefficient estimates and complicating the analysis of variable significance. Techniques like PCA can help by transforming correlated variables into uncorrelated principal components, thereby improving model stability and interpretability.

[Principal Component Analysis](#) (PCA) is a [dimensionality reduction](#) technique that can help address [multicollinearity](#) in regression models.

- 1. Combining Correlated Variables:** PCA transforms the correlated independent variables into a set of uncorrelated variables called principal components. These components capture the majority of the variance in the data while reducing redundancy.
- 2. Reducing Dimensionality:** By selecting a smaller number of principal components that explain most of the variance, PCA can simplify the model. This reduces the complexity and potential overfitting associated with having too many correlated predictors.
- 3. Improving Model Stability:** By using principal components instead of the original correlated variables, the regression model can achieve greater stability and reliability in coefficient estimates, as the issues caused by multicollinearity are mitigated.

4. **Enhanced Interpretability:** While the principal components may not have a direct interpretation in terms of the original variables, they can still provide insights into the underlying structure of the data and the relationships among variables.

Example Code

```
# edit this to explore how to address multi collinusing pca
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

variables = data_cleaned[['var1', 'var2', 'var3']] #var1-var2-var3
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(variables.values, i) for i in range(variables.shape[1])]
vif["features"] = variables.columns

# Drop feature with VIF > 10
data_no_multicollinearity = data_cleaned.drop(['Year'], axis=1)
```

Addressing_Multicollinearity.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/Regression/Addressing_Multicollinearity.py

Adjusted R Squared

Adjusted R-squared is a **Regression Metrics** for assessing the quality of a regression model, especially when multiple predictors are involved. It helps ensure that the model remains **parsimonious** while still providing a good fit to the data.

When evaluating a regression model, if you notice a **large difference** between **R squared** and adjusted R², it indicates that the additional predictors may not be improving the model's performance.

In such cases, it may be beneficial to drop those extra variables to simplify the model without sacrificing predictive power.

Key features:

1. Penalty for Number of Predictors:

- o Adjusted R² adjusts the R² value by penalizing the addition of **unnecessary predictors**. This means that if you add a variable that does not improve the model significantly, the adjusted R² will decrease, reflecting that the model may be overfitting.

2. Comparison with R-squared:

- o Adjusted R² is always less than or equal to R². While R² can artificially inflate with the addition of more predictors (even if they are not useful), adjusted R² provides a **more reliable** assessment of model fit by considering the number of predictors relative to the number of observations.

3. Interpretation:

- o Like R², adjusted R² values range from 0 to 1, where values closer to 1 indicate a better fit. However, a significant difference between R² and adjusted R² suggests that the model may be penalized for including

extra variables that do not contribute meaningfully to the prediction.

$$4. \text{ Formula: } R_{adj.}^2 = 1 - (1 - R^2) \cdot \frac{n-1}{n-p-1}$$

- o Where:
 - \$R^2\$ = R-squared value
 - \$n\$ = number of observations
 - \$p\$ = number of predictors (independent variables)

Agent Based Modelling

(ABM) is a computational approach that simulates the interactions of individual agents within a defined environment to observe complex phenomena and [emergent behavior](#) at a system level.

Agent-based modeling provides a robust framework for understanding and analyzing complex systems, particularly in the energy sector.

By simulating individual agents and their interactions, researchers and practitioners can gain insights into system dynamics, evaluate [Policy|policies](#), and optimize strategies for energy production and consumption.

Principles of Agent-Based Modelling

1. **Agents:** The primary components of ABM, agents can represent individuals, groups, or entities with defined behaviors and attributes. Each agent operates based on its rules and interactions with other agents and the environment.
2. **Environment:** The space in which agents operate, which can be a physical or abstract setting. The environment influences agent behavior and can include various elements like resources, obstacles, or rules governing interactions.
3. **Interactions:** Agents communicate and interact with each other and their environment. These interactions can be cooperative, competitive, or neutral, leading to complex system dynamics.
4. **Emergence:** ABM focuses on emergent phenomena, where the collective behavior of agents leads to unexpected outcomes not evident from examining individual agents alone. This principle helps understand complex systems' dynamics and behaviors.

Techniques in Agent-Based Modeling

1. **Model Development:**
 - o **Define Agents:** Specify agent types, behaviors, attributes, and decision-making processes.
 - o **Environment Design:** Create a representation of the environment, including spatial aspects and available resources.
 - o **Interaction Rules:** Establish rules governing how agents interact with each other and their environment.
2. **Simulation:**
 - o Execute the model over time, allowing agents to make decisions, interact, and adapt based on predefined rules.
 - o Collect data on agents' behaviors and system-wide outcomes during the simulation.
3. **Analysis:**
 - o Analyze the results to understand emergent patterns and behaviors. This can involve statistical analysis, visualization of agent interactions, and evaluating how different parameters influence outcomes.
4. **Validation:**

- Compare model outputs with real-world data to validate the model's accuracy. Calibration may be necessary to ensure the model reflects observed behaviors accurately.
-

Applications of Agent-Based Modeling

1. Energy Systems:

- **Demand Response:** ABM can simulate consumer behavior in response to dynamic pricing or demand response programs, providing insights into how to encourage energy conservation during peak demand.
- **Renewable Energy Integration:** It helps model the interactions between different energy producers (e.g., solar, wind) and consumers, examining how they adapt to changes in supply and demand.

2. Market Dynamics:

- Simulate interactions between different energy providers and consumers to understand competitive behavior, pricing strategies, and market outcomes.
- Evaluate the impact of regulatory changes on market behavior and investment decisions.

3. Resource Management:

- Model interactions among agents in managing shared resources, such as water or electricity, to study the effects of cooperation and competition on resource depletion or sustainability.

Example Frameworks and Tools

Several tools and frameworks are available for building and simulating agent-based models, including:

- **NetLogo:** A user-friendly platform for creating agent-based models, particularly in education and research.
 - **AnyLogic:** A powerful commercial tool that supports agent-based, system dynamics, and discrete event modeling.
 - **Repast:** An open-source framework for building ABMs, widely used in academic research.
 - **MASON:** A fast and flexible discrete-event simulation library for Java, suitable for developing complex ABMs.
-

Agent-Based Modelling

- **Multi-Agent Systems** in [LLM|LLMs](#)
 - The need for shared context across different agents, ensuring consistency and coherence in interactions.

Agent Interactions

How do Agents Interact?

- **Horizontal Collaboration:** Agents with local goals coordinate to achieve a shared objective.
- **Hierarchical Collaboration:** Primary agents oversee specialized agents to manage complex tasks effectively.

Understanding Agents

Core Components:

1. **Tools:** Resources such as APIs, databases, or GitHub.
 2. **Strategy:** Techniques like self-criticism, [chain of thought](#) (CoT), and planning to improve reasoning.
 3. **States:** Memory, context tracking, and microservices for modularity.
 4. **Goals:** Specific objectives defined for each agent.
-

Agent Planning and Interaction

1. **Planning:** Agents plan operations, such as managing workflows in a support center.
2. **Agent Collaboration:** Agents align their individual goals with shared objectives to enhance system performance.
3. Multi-step

Compounding Systems

Multi-Agent Systems

These systems reduce reliance on extensive [prompt engineering](#) by compartmentalizing tasks across specialized agents.

Example: A writer agent drafts content, while a reviewer agent ensures quality, both operating within defined scopes.

Agentic Solutions

Agentic solutions leverage multiple autonomous agents (usually [Small Language Models\(SLM\)](#)) to achieve goals collaboratively. These systems distribute tasks across agents that operate individually or collectively to solve complex problems.

Agents can model specific business functions. Role clarity enhances the effectiveness of these systems.

Related terms:

- [GraphRAG](#)
- [Agent-Based Modelling](#)

Types of Agentic Solutions

Reactive Solutions (Ask Approach):

Systems like chatbots and retrieval-augmented generation (RAG) tools respond to user queries.

Autonomous Solutions (Do Approach):

Agents perform tasks proactively, e.g., drafting documents or scheduling meetings.

Business Process Integration

Workflow:

1. Identify a business problem.
2. Define personas (agents) required.
3. Develop an agentic workflow.

Agentic Architectures:

1. **Vertical:** Hierarchical structures for task delegation.
2. **Horizontal:** Collaborative structures with high feedback loops.
3. **Mixed:** Combines vertical delegation with horizontal collaboration.

Vertical Example: A primary agent delegates tasks to lower-level agents for execution.

The Orleans Framework

A framework for building distributed applications in .NET.

- **Grains:** Individual agents performing specific tasks.
- **Silos:** Distributed nodes managing grains.
- **Clusters:** Collections of silos for scalability.

Benefits of Using Agents

1. **Performance Gains:** Task parallelization enhances throughput.
2. **Developer Abstraction:** Modular design simplifies system understanding and debugging.
3. **Workflow Integration:** Aligns AI agents with organizational processes.

Example Use Cases

1. **IT Helpdesk Agent:** Automates troubleshooting and network access requests.
2. **Device Refresh Agent:** Manages hardware upgrades and approvals.
3. **Lead Generation Agent:** Identifies and researches potential leads.
4. **Budget Management Agent:** Reviews financial data and aids in planning.
5. **Customer Support Agent:** Triage support issues for faster resolution.
6. **Project Tracker Agent:** Tracks project milestones and budget compliance.

Aggregation

Aggregation

Summarizing data for analysis ([Pandas Pivot Table](#) and [Groupby](#)).

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/group_by.ipynb
- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/pivot_table.ipynb

Algorithms

Recursive Algorithm

Backtracking

Backtracking is a method for solving problems incrementally, by trying partial solutions and then abandoning them if they are not valid.

Example: Graph coloring with the 4-color theorem.

Divide and Conquer

Divide and conquer is a strategy that involves breaking a problem into smaller subproblems of the same type, solving these subproblems recursively, and then combining their solutions to solve the original problem.

Example: Merge sort, where the array is split in half, and each smaller part is sorted.

Note: Subproblems do not generally overlap.

Dynamic Programming

Dynamic programming is used for optimization problems and involves storing past results to find new results efficiently.

- **Memoization:** This technique "remembers" past results to avoid redundant calculations.
- It is characterized by overlapping substructure and overlapping subproblems.

Examples: Fibonacci numbers, Towers of Hanoi, etc.

Greedy Algorithms

A greedy algorithm builds up a solution piece by piece, always choosing the next piece that offers the most immediate benefit without regard for future consequences. The hope is that by choosing a local optimum at each step, a global optimum will be reached.

Examples: Dijkstra's shortest path algorithm, Knapsack problem.

Branch and Bound

Branch and bound algorithms are typically used for optimization problems and are similar to backtracking.

- As the algorithm progresses, a tree of subproblems is formed, with the original problem as the "root problem."
- A method is used to construct upper and lower bounds for a given problem.
- At each node, bounding methods are applied:
 - If the bounds match, it is deemed a feasible solution for that subproblem.
 - If the bounds do not match, the problem represented by that node is partitioned into two subproblems, which become child nodes.

- The process continues, using the best known feasible solution to trim sections of the tree until all nodes have been solved or trimmed.
-

Example: Traveling salesman problem.

Brute Force

Brute force algorithms try all possible solutions until they find an optimized or satisfactory answer. Heuristics can be used to assist in this process.

Randomized Algorithms

Randomized algorithms use random numbers to influence their behaviour.

Example: [K-means](#) clustering initialization.

Alternatives To Batch Processing

If you're working with a streaming dataset ([Data Streaming](#)), why might [batch processing](#) not be suitable, and what alternatives would you consider?

Latency: Batch processing involves collecting data over a period and processing it in large chunks. This can introduce significant delays, making it unsuitable for applications that require real-time or near-real-time insights.

Timeliness: Streaming datasets often require immediate processing to respond to events as they occur. Batch processing cannot meet the demand for timely [data analysis](#).

Data Freshness: In streaming scenarios, data is continuously generated, and waiting for a batch interval can result in outdated information being analyzed.

Alternatives to Batch Processing

1. **Stream Processing:** This approach processes data in real-time as it arrives. Tools like [Apache Kafka](#), [Apache Flink](#), and [Apache Spark Streaming](#) are designed for handling streaming data efficiently.
2. **Event-Driven Architectures:** Implementing an [event-driven architecture](#) allows systems to react to data changes or events immediately, ensuring timely processing and response.
3. **Micro-batching:** This technique processes small batches of data at very short intervals, striking a balance between batch and stream processing. Tools like [Apache Spark Streaming](#) can utilize micro-batching to handle streaming data more effectively.
4. **Complex Event Processing (CEP):** CEP systems analyze and process streams of events in real-time, allowing for the detection of patterns and trends as they happen.

Amazon S3

Amazon S3

Amazon S3 buckets (onedrive essentially)

Amazon Simple Storage Service (S3) is a versatile **object storage solution** known for its scalability, data availability, security, and performance.

Stands for Simple Storage Service.

What is Amazon S3?

It's an object storage service, allowing you to upload and store various types of objects, including images, text, videos, and more. S3's structure resembles that of a typical file system, with folders, subfolders, and files. Notably, it's extremely cost-effective, with storage costs starting at only 0.023 cents per GB, and it offers high durability with data replicated across three availability zones.

Why is Amazon S3 Useful?

1. **Cost-Effective Storage:** S3 provides cheap, reliable storage for objects of any type.
2. **Low Latency, High Throughput:** It offers fast access to your data, making it suitable for hosting static websites.
3. **Integration with AWS Services:** S3 can be integrated with other AWS services like SNS, SQS, and Lambda for powerful event-driven applications.
4. **Lifecycle Management:** S3 offers lifecycle management to automatically transition data to lower-cost storage tiers based on access patterns.

Step-by-Step Walkthrough in the AWS Console

1. **Creating a Bucket:** Start by navigating to the AWS Management Console and selecting S3. Create a bucket with a unique name and choose the region closest to your application.
2. **Uploading a File:** Once the bucket is created, upload a file using the console. You can add metadata and set permissions as needed.
3. **Exploring Settings:** Dive into bucket settings to configure options like versioning, logging, encryption, and lifecycle management. Block public access to ensure data security.

Additional Features

1. **Transfer Acceleration:** Accelerate data transfer to and from S3 using optimized network paths.
2. **Events:** Configure event notifications to trigger actions in response to S3 events like object creation or deletion.
3. **Requester Pays:** Enable Requester Pays to allow bucket owners to pass on data transfer costs to requesters.

Anomaly Detection In Time Series

In [Time Series](#)

In [ML_Tools](#) see:

- [TS_Anomaly_Detection.py](#)

To perform anomaly detection specifically for time series data, you can utilize various techniques that account for the **temporal nature** of the data. Here are some common methods:

1. Statistical Methods:

- Moving Average: Calculate a moving average and identify points that deviate significantly from this average.
- Seasonal Decomposition: Decompose the time series into trend, seasonal, and residual components. Anomalies can be identified in the residuals.

2. Time Series Models:

- AutoRegressive Integrated Moving Average: Fit an ARIMA model to the time series data and analyze the residuals for anomalies.
- State Space Model (ETS): Similar to ARIMA, this model can be used to forecast and identify anomalies in the residuals.

3. Machine Learning Approaches:

- [LSTM](#) (Long Short-Term Memory): Use LSTM networks to model the time series and detect anomalies based on prediction errors.
- [Isolated Forest](#): This algorithm can be adapted for time series data by treating time as an additional feature.

4. Change Point Detection:

- Identify points in time where the statistical properties of the time series change significantly, which may indicate anomalies.

5. Visual Methods:

- Time Series Plots: Visual inspection of time series plots can help identify anomalies.
- Control Charts: Use control charts to monitor the time series and flag points that fall outside control limits.

Anomaly Detection With Clustering

Clustering

- Description: Outliers often form small clusters or are isolated from main clusters.

8. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- **Purpose:**

Finds anomalies based on density rather than explicit statistical assumptions.

- **Steps:**

- Identify points in low-density regions as anomalies.

[Isolated Forest](#)|[Forest](#)

2. Local Outlier Factor (LOF)

LOF is a density-based anomaly detection method that identifies anomalies by comparing the local density of a point with that of its neighbors.

Steps:

- For each point, calculate the local density based on the distance to its k-nearest neighbors.
- Compute the LOF score, which measures the degree of isolation of a point relative to its neighbors.

Introduction

- Points with a LOF score significantly greater than 1 are considered anomalies.
-

Anomaly Detection With Statistical Methods

Basic:

- [Z-Normalisation|Z-Score](#)
- [Interquartile Range \(IQR\) Detection](#)
- [Percentile Detection](#)

Advanced:

- [Gaussian Model](#)
- [Isolated Forest](#)

Grubbs' Test

Context:

Grubbs' test is a hypothesis test designed to detect a single outlier in a normally distributed dataset. It tests the largest deviation from the mean relative to the standard deviation. This test is iterative and removes one outlier at a time.

Purpose:

To determine whether the most extreme data point (either smallest or largest) is a statistical outlier.

Steps:

- Compute the test statistic:

$$G = \frac{\max(|X - \mu|)}{\sigma}$$

where:

- X : Data points
- μ : Mean of the dataset
- σ : Standard deviation of the dataset.
- Compare G to a critical value:
 - The critical value depends on the sample size n and significance level α (e.g., 0.05).
 - If G exceeds the critical value, the data point is considered an outlier.

Limitations:

- Assumes data follows a normal distribution.
 - Inefficient for detecting multiple outliers simultaneously.
-

Histogram-Based Outlier Detection (HBOS)

Context:

HBOS is a non-parametric method that detects anomalies by analyzing the distribution of individual features independently. It relies on histograms, which estimate feature density.

Purpose:

To identify outliers as data points falling in bins with low frequencies or densities.

Steps:

- Create histograms for each feature:
 - Divide each feature's range into bins.
 - Count the frequency of data points in each bin.
- Calculate scores for each data point:
 - Outliers are points in bins with significantly lower densities compared to others.

Advantages:

- Does not assume a specific data distribution.
- Scales well to large datasets.

Limitations:

- Assumes feature independence (not ideal for [multivariate data](#)).
- Sensitive to bin size selection.

One-Class SVM

One-Class Support Vector Machine is a variation of the SVM algorithm used for anomaly detection. It learns a decision boundary around the normal data points.

Steps:

- Train the model on the normal data points.
- The model attempts to find a hyperplane that separates the normal data from the origin.
- Points that fall outside this boundary are classified as anomalies.

Anomaly Detection

Anomaly detection involves identifying [standardised/Outliers|Outliers](#). Detecting these anomalies is crucial for maintaining [data integrity](#) and improving model performance.

Methods for Detecting Anomalies

In [ML_Tools](#) see: [Pycaret_Anomaly.ipynb](#)

Process of Detection

Data Preparation

- [Data Cleansing](#): Handle missing values and remove any irrelevant data points.
- [Normalisation/Standardisation](#): Scale the data if necessary, especially if using methods sensitive to the scale.

Anomaly Detection with a model: Use a chosen method to flag anomalies

- [Anomaly Detection with Clustering](#)
- [PCA-Based Anomaly Detection](#)
- [Anomaly Detection in Time Series](#)
- [Anomaly Detection with Statistical Methods](#)

Validation

- Validate the detected anomalies by comparing them against known anomalies (if available) or using domain knowledge.
- Adjust thresholds or methods based on validation results.

Visualization

- Visualize the results using plots (e.g., scatter plots, box plots) to understand the distribution of data and the identified anomalies.
- [Boxplot](#): Displays the distribution and identifies outliers using the interquartile range (IQR).
- Scatter Plot: Helps visually identify outliers.

Apache Airflow

Schedular think CROM jobs with python.

Apache Nifi may be better.

Airflow is a [data orchestrator](#) and the first that made task scheduling popular with [Python](#).

Airflow programmatically author, schedule, and monitor workflows. It follows the [imperative](#) paradigm of schedule as *how* a DAG [Directed Acyclic Graph \(DAG\)](#) is run has to be defined within the Airflow jobs. Airflow calls its *Workflow as code* with the main characteristics

- **Dynamic**: Airflow pipelines are configured as Python code, allowing for dynamic pipeline generation.
- **Extensible**: The Airflow framework contains operators to connect with numerous technologies. All Airflow components are extensible to easily adjust to your environment.
- **Flexible**: Workflow parameterization is built-in leveraging the [Jinja Templating](#) engine.

Apache Kafka

Apache Kafka is an open-source distributed event streaming platform used for building real-time data pipelines and data streaming applications. It is designed to handle high-throughput, fault-tolerant, and scalable messaging.

Features

Immutable commit log: Kafka maintains an append-only log of messages, which ensures [Data Integrity](#) and replicability.

Introduction

Kafka allows applications to [Publish and Subscribe](#) to streams of records, similar to a message queue or enterprise messaging system.

Durability and Reliability: Kafka stores streams of records in a fault-tolerant way, ensuring data durability and reliability. It replicates data across multiple nodes to prevent data loss.

Scalability: Kafka is designed to scale horizontally by adding more brokers (servers) to the cluster, which can handle increased load and data volume.

Kafka is optimized for high throughput, making it suitable for processing large volumes of data in real-time.

Data in Kafka is organized into topics, which are further divided into partitions. Each partition is an ordered, immutable sequence of records that is continually appended to.

Producers are applications that publish data to Kafka topics, while consumers are applications that subscribe to topics and process the data.

Kafka is commonly used for log aggregation, real-time analytics, [data integration](#), stream processing, and building event-driven architectures.

Kafka integrates ([Data Integration](#)) well with various data processing frameworks like Apache Spark, Apache Flink, and Apache Storm, as well as with databases and other [data storage](#) systems.

Apache Spark

Apache Spark is an open-source multi-language engine for executing [Data Engineer](#) and [Machine Learning](#) on single-node machines or clusters. It's optimized for large-scale data processing.

Spark runs well with [Kubernetes](#).

Spark is a highly popular framework for large-scale data processing. It allows [Data Engineer](#) to process massive datasets in memory, which makes it faster than traditional disk-based approaches. Spark is versatile, supporting batch processing, real-time data streaming, machine learning, and graph processing.

Asking Questions

Why Ask Better Questions?

Asking better questions enhances thinking, learning, problem-solving, and communication. Good questions:

- Guide inquiry and exploration.
- Clarify assumptions.
- Elicit insights or novel responses.
- Enable self-reflection and deeper understanding.

What Makes a Good Question?

A good question:

- **Elicits a novel or thoughtful response** — not just a fact or yes/no.
 - **Opens possibilities** rather than closing them.
 - **Reveals assumptions or forces re-evaluation** of mental models.
 - **Matches the context and audience** — good questions for brainstorming differ from those for debugging.
 - **Fosters chain-of-thought reasoning**, helping others articulate how they arrive at conclusions.
-

Types of Questions

Questions can be classified by their **function**, **depth**, or **structure**.

1. By Function

Type	Purpose	Example
Clarifying	Understand what's being said	"What do you mean by X?"
Probing	Dig deeper into reasoning or logic	"Why do you think that?"
Exploratory	Generate ideas or new perspectives	"What if we reversed the problem?"
Reflective	Encourage self-awareness	"What assumption am I making here?"
Critical	Test or challenge statements	"What evidence supports that?"

2. By Depth

- **Surface-level:** "What is X?"
- **Mid-level:** "How does X relate to Y?"
- **Deep-level:** "Why does X matter?" or "What are the implications of X?"

3. By Structure

- **Open-ended:** Encourage elaboration.

→ "*How might we design this differently?*"

- **Closed:** Seek a specific answer.

→ "*Is this implementation correct?*"

- **Leading:** Suggest an answer.

→ "*Wouldn't you agree that...?*"

- **Falsifiable/Testable:** Can be proven right or wrong.

→ "*Does increasing X always decrease Y?*"

Characteristics of Good Questions

Characteristic	Description
Purposeful	Serves a clear goal in the conversation or inquiry
Contextual	Relevant to the topic or the respondent
Open/Expansive	Invites multiple viewpoints or lines of reasoning
Challenging	Pushes beyond defaults or surface-level answers
Precise	Minimizes ambiguity while leaving room for elaboration
Sequenced	Ordered to build thought step-by-step (chain of thought)

Related Concepts

- [Chain of Thought](#)
- [Design Thinking Questions](#)
- [Prompting](#)

Questions:

- How LLMs generate or refine questions using [Prompting](#) or [Chain of thought](#) approaches?

Attention Is All You Need

Attention Mechanism

Think of attention like human reading behavior: when reading a complex sentence, we don't process all the words equally at every moment. Instead, we might "attend" more to certain words based on the context of what we've read so far and what we're trying to understand. This is similar to what the attention mechanism does in neural networks.

The attention mechanism is a key concept in modern [ML](#) particularly in natural language processing ([NLP/LLM](#)) and sequence-based models like neural machine translation (NMT). It was introduced to address the limitations of earlier models, like [Recurrent Neural Networks|RNN](#) and [LSTM](#) network), in handling long sequences and capturing important dependencies within data.

The attention mechanism improves a model's ability to focus on important parts of the input data, helping it manage long-range dependencies, which is especially useful in tasks like machine translation, text generation, and various NLP tasks.

Core Idea of Attention

The [Transformer|Transformer](#) architecture, introduced by Vaswani et al. in 2017 ("Attention Is All You Need"), is the most popular use of the attention mechanism. The key innovation of Transformers is the [self-attention mechanism](#), which [allows each token in a sequence to attend to all other tokens](#), making the model more efficient and scalable for parallel processing. [Transformers replaced traditional RNN-based](#) models and have become the foundation of models like [BERT](#), GPT, and T5.

The attention mechanism allows a model to focus on different parts of an input sequence when making predictions, rather than relying on a fixed-size hidden state to encode all information. This selective "focus" can greatly enhance the model's ability to handle long-range dependencies.

For example, in machine translation, while translating a sentence from one language to another, different words in the input sentence might hold varying degrees of relevance to the word currently being translated. Attention helps the model dynamically weigh the importance of different words at each step of the translation.

In its simplest form, attention assigns weights to each input token based on its relevance to the current output token being generated. These weights are typically calculated using a score function and then normalized (usually through a softmax) to produce a distribution over the input sequence. The weighted sum of the input tokens is then passed as context for generating the output token.

Key Components of Attention and Formula

I see! Here's the text with the math terms wrapped in as requested:

1. Query: Represents the current word or position that requires attention: $\$Q\$$
2. Key: Represents each word in the input sequence: $\$K\$$
3. Value: Represents the actual content or information in the input sequence: $\$V\$$
4. Attention Scores: The attention mechanism computes the relevance between the query and each key by computing a similarity score (such as dot-product or other scoring methods).
5. Softmax: These scores are then passed through a softmax function to form a probability distribution, which gives us the attention weights.
6. Context Vector: A weighted sum of the values ($\$V\$$), using the attention weights, is computed. This context vector is what the model uses to generate the output token.

Given a query matrix, key matrix, and value matrix, attention is calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- $\$Q\$, \$K\$,$ and $\$V\$$ are matrices of query, key, and value vectors.
- d_k is the dimension of the keys.
- The softmax is applied row-wise to produce attention weights.

Applications of Attention

- Machine Translation: Aligns source and target words in a more dynamic and context-dependent way.
- Text Summarization: Helps to identify the most relevant parts of a document.
- Speech Recognition: Enhances the model's ability to focus on important features over long audio sequences.
- Vision: Self-attention is now used in computer vision tasks (Vision Transformers or ViTs) to model dependencies between different parts of an image.

Types of Attention Mechanisms

1. Additive Attention: Introduced in the original attention paper by Bahdanau et al. (2014), this method uses a feed-forward neural network to compute the relevance between the query and each key.
2. Dot-Product (Multiplicative) Attention: Introduced in the Transformer paper by Vaswani et al. (2017), this method computes the relevance using a simple dot product between the query and key vectors. It is computationally efficient and widely used.

-
- 3. Scaled Dot-Product Attention: A variant of dot-product attention, used in the Transformer architecture, where the dot product is divided by the square root of the dimension of the key vectors to avoid excessively large gradients.
 - 4. Self-Attention: In this mechanism, the model applies attention to itself. This means each word in the input sequence attends to all other words in the sequence, including itself. Self-attention is used in models like Transformers to capture dependencies within a sentence.

Self attention vs multi-head attention

<https://www.youtube.com/shorts/Muvjex0nkes>

Take every word pays attention to every other word to capture context by:

- 1. take input word vectors,
- 2. break words into Q,K,V vectors,
- 3. compute attention matrix
- 4. generate final word vectors. these vectors

Multi-head attention: perform self attention in parallel.

- 1. take word vectors,
- 2. break words into Q,K,V vectors,
 - i. Break each Q,K,V vector into the number of heads parts
- 3. compute attention matrix for each head
- 4. generate final word vectors for each head
- 5. Combine back together

These have better understanding of the context.

Multi-head attention

- This allows the model to weigh the importance of different words in a sequence when making predictions. It captures relationships between words, even if they are far apart in the sequence.
 - The model computes attention scores for each pair of words to determine how much focus one word should place on another in a sequence.

Automated Feature Creation

Question: Can we autodetect meaningful features.

Feature Engineering is an ad-hoc manual process that depends on domain knowledge, intuition, data exploration, and creativity. However, this process is dataset-dependent, time-consuming, tedious, subjective, and not a scalable solution.

Automated Feature Creation automatically generates features using a framework; these features can be filtered using **Feature Selection** to avoid feature explosion.

Below are some popular open-source libraries for automated feature engineering:

- Pycaret – [PyCaret](#)
- Featuretools for advanced usage – [Home | What is Featuretools? — Featuretools 1.1.0 documentation](#)
- Optuna – [A hyperparameter optimization framework](#)
- Feature-engine – [A Python library for Feature Engineering for Machine Learning — 1.1.2](#)
- ExploreKit – [GitHub – giladkatz/ExploreKit](#)

Introduction

- <https://www.turintech.ai/blog/feature-generation-what-it-is-and-how-to-do-it>
-

Azure

Public cloud computing platform from Microsoft offering various services like infrastructure, data storage, and machine learning.

B

Table of Contents

- [B-tree](#)
- [BERT Pretraining of Deep Bidirectional Transformers for Language Understanding](#)
- [BERT](#)
- [BERTScore](#)
- [Backpropagation](#)
- [Bag of words](#)
- [Bag_of_Words.py](#)
- [Bagging](#)
- [Bandit_Example_Fixed.py](#)
- [Bandit_Example_Nonfixed.py](#)
- [Bash](#)
- [Batch Normalisation](#)
- [Batch Processing](#)
- [Bellman Equations](#)
- [Benefits of Data Transformation](#)
- [Bernoulli](#)
- [Bias and variance](#)
- [Big Data](#)
- [Big O Notation](#)
- [BigQuery](#)
- [Binary Classification](#)
- [Binder](#)
- [Boosting](#)
- [Bootstrap](#)
- [Boxplot](#)
- [Business observability](#)
- [business intelligence](#)

B Tree

Bert Pretraining Of Deep Bidirectional Transformers For Language Understanding

Bert

BERT ([Bidirectional Encoder Representations from Transformer](#)) is used in [NLP](#) processing, developed by [Google](#).

Introduced in the paper "[BERT Pretraining of Deep Bidirectional Transformers for Language Understanding](#)" in 2018.

It is forward & backward looking in the context.

BERT is a stack of encoders -learning context.

Input [Vector Embedding|embedding](#):

- [Positional Encoding](#): passes location info to encoder
- Sentence embeddings: differences between sentences
- Token embeddings

Training of BERT:

- Masked Language modelling (hiding words)
- Next Sentence Prediction

Fine tuning ([Transfer Learning](#)) BERT model:

- New output layer dependent

Resources:

- [What is BERT and how does it work? | A Quick Review](#)

What is BERT?

- BERT is based on the [Transformer](#) architecture and utilizes a bidirectional approach, meaning it considers the [context of a word based on both its left and right surroundings in a sentence](#). This allows BERT to capture nuanced meanings and relationships between words more effectively than unidirectional models
- Pre-training and Fine-tuning/[Transfer Learning](#) techniques. It learns to predict masked words in sentences (Masked Language Model) and to determine if one sentence follows another (Next Sentence Prediction).

What is BERT Used For?

- Text Classification: Assigning categories or labels to text documents, such as sentiment analysis or topic classification.
- Named Entity Recognition ([Named Entity Recognition|NER](#)): Identifying and classifying entities (e.g., names, organizations, locations) within text.
- Question Answering: Providing answers to questions based on a given context or passage of text.
- Text [Summarisation](#): Generating concise summaries of longer documents while retaining key information.
- Language Translation: Assisting in translating text from one language to another.
- [Sentence Similarity](#) :Measuring the similarity between sentences, which can be useful for tasks like paraphrase detection or duplicate question identification.

Bertscore

Backpropagation

[!Summary]

Backpropagation is an essential algorithm in the training of neural networks and iteratively correcting its mistakes. It involves a process of calculating the gradient of the loss function $L(\theta)$ concerning each weight in the network, allowing the system to update its weights via [Gradient Descent](#).

This process helps minimize the difference between predicted outputs and actual target values.

Mathematically, the chain rule of calculus is employed to propagate errors backward through the network.

Each layer in the network computes a partial derivative that is used to adjust the weights. This iterative approach continues until a convergence criterion is met, typically when the change in loss falls below a threshold.

The backpropagation algorithm is critical in [Supervised Learning](#), where labeled data is used to train models to recognize patterns.

[!Breakdown]

Key Components:

- **Algorithm:** Gradient Descent
- **Mathematical Foundation:** Chain Rule for derivatives
- **Metrics:** Loss function (e.g., Mean Squared Error, Cross-Entropy)

[!important]

- Gradient descent uses $\nabla L(\theta) = \frac{\partial L}{\partial \theta}$ to iteratively minimize the loss.
- Backpropagation optimizes deep learning models by adjusting weights based on error gradients.

[!attention]

- The method is computationally expensive for deep networks due to the need to compute gradients for each layer.
- Vanishing/exploding gradients in deep layers can prevent proper weight updates.

[!Example]

A feed-forward neural network trained on image classification data uses backpropagation to minimize cross-entropy loss. The gradient of the loss is calculated layer by layer, adjusting weights through an optimization algorithm like Adam.

[!Follow up questions]

- How does backpropagation compare with other optimization algorithms such as Newton's method or evolutionary strategies?

Introduction

- What role does [Regularisation](#) play in addressing overfitting when using backpropagation in deep neural networks?

[!Related Topics]

- Gradient Descent Optimizers ([Adam Optimizer](#), RMSprop)
- [vanishing and exploding gradients problem](#)

Backpropagation

Backpropagation is used to calc the gradient of the loss function with respect to the model parameters, when there are a lot of parameters - i.e in a neural network. Simple example of backpropagation

Simple example of computation graph Computation graph - calcng derivatives? Use [sympy](#) to calculate derivatives for the loss function.

The steps in backprop

Now that you have worked through several nodes, we can write down the basic method:\ working right to left, for each node:

- calculate the local derivative(s) of the node
- using the chain rule, combine with the derivative of the cost with respect to the node to the right.

The 'local derivative(s)' are the derivative(s) of the output of the current node with respect to all inputs or parameters.

Example of using [sympy](#) to calculate derivatives for the loss function. Use `diff` , `subs`

```
from sympy import symbols, diff
```

Sympy

Bag Of Words

In [ML_Tools](#) see: [Bag_of_Words.py](#)

In the context of natural language processing (NLP), the Bag of Words (BoW) model is a simple and commonly used [method for text representation](#). It converts text data into numerical form by treating each [document as a collection of individual words, disregarding grammar and word order](#). Here's how it works:

1. Vocabulary Creation: A vocabulary is created from the entire corpus, which is a list of all unique words appearing in the documents.
2. Vector Representation: Each document is represented as a vector, where each element corresponds to a word in the vocabulary. The value of each element is typically the count of occurrences of the word in the document.
3. Simplicity and Limitations: While BoW is easy to implement and useful for tasks like text classification, it has limitations. It ignores word order and context, and can result in large, sparse vectors for large vocabularies.

Despite its simplicity, BoW can be effective for certain NLP tasks, especially when combined with other techniques like [TF-IDF](#) to weigh the importance of words.

Takes key terms of a text in normalised [unordered](#) form.

`CountVectorizer` from scikit-learn to convert a collection of text documents into a matrix of token counts.

```
#Need normalize_document
from sklearn.feature_extraction.text import CountVectorizer

# Using CountVectorizer with the custom tokenizer
bow = CountVectorizer(tokenizer=normalize_document)
bow.fit(corpus) # Fitting text to this model
print(bow.get_feature_names_out()) # Key terms
```

Represent each sentence by a vector of length determined by `get_feature_names_out`. representing the tokens contained.

Summary of What the Script Does:

1. It takes a dataset of text (movie reviews in this case) and processes it to remove HTML tags, non-alphabetic characters, and stopwords.
2. It transforms the cleaned text into numerical features using the **Bag of Words** model, where each word in the reviews is counted and represented as a feature.
3. It prints a sample of the top features (words) that were extracted from the reviews.

This is a typical text preprocessing pipeline used to prepare textual data for machine learning models.

Bagging

Overview:

Bagging, short for Bootstrap Aggregating, is an [Model Ensemble](#) technique designed to improve the stability and accuracy of machine learning algorithms.

It works by [training multiple instances of the same learning algorithm on different subsets of the training data](#) and then [combining their predictions](#).

How Bagging Works:

1. **Bootstrap Sampling:** Bagging involves creating multiple subsets of the training data by sampling with replacement. This means that each subset, or "bootstrap sample," is drawn randomly from the original dataset, and some data points may appear multiple times in a subset while others may not appear at all.
2. **Parallel Training:** Each bootstrap sample is used to train a separate instance of the same base learning algorithm. These models are trained independently and in parallel, which makes bagging computationally efficient.
3. **Combining Predictions:** Once all models are trained, their predictions are combined to produce a final output. For regression tasks, this is typically done by [averaging](#) the predictions. For classification tasks, [majority voting](#) is used to determine the final class label.

Key Concepts of Bagging:

- **Reduction of Overfitting:** By averaging the predictions of multiple models, bagging reduces the variance and helps prevent overfitting, especially in high-variance models like decision trees.
- **Diversity:** The use of different subsets of data for each model introduces diversity among the models, which is crucial for the success of ensemble methods.
- **Parallelization:** Since each model is trained independently, bagging can be easily parallelized, making it scalable and efficient for large datasets.

Example of Bagging:

Random Forest: A well-known example of a bagging technique is the [Random Forests](#) algorithm.

It uses decision trees as base models and combines their predictions to improve accuracy and robustness.

Each tree in a random forest is trained on a different bootstrap sample of the data, and the final prediction is made by averaging the outputs (for regression) or majority voting (for classification).

Further Understanding

Advantages of Bagging:

- **Increased Accuracy:** By combining multiple models, bagging often achieves higher accuracy than individual models.
- **Robustness:** Bagging is less sensitive to overfitting, especially when using high-variance models like decision trees.
- **Flexibility:** It can be applied to various types of base models and is not limited to a specific algorithm.

Challenges of Bagging:

- **Complexity:** While bagging reduces overfitting, it can increase the complexity of the model, making it harder to interpret.
- **Computational Cost:** Training multiple models can be computationally intensive, although this can be mitigated by parallel processing.

Bandit_Example_Fixed.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Deployment/Bandit_Example_Fixed.py

Bandit_Example_Nonfixed.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Deployment/Bandit_Example_Nonfixed.py

Automation Scripts

In [ML_Tools](#), see: [Bash_folder](#)

Basic Commands

1. Show Current Directory:

```
pwd
```

2. Display Contents of a Text File:

```
cat filename.txt
```

3. Search for a Word in a File:

```
grep "word" filename.txt
```

4. Replace Text in a File (Output Only):

```
sed 's/old/new/g' filename.txt
```

Writing and Running a Bash Script

1. Create a Script:

```
nano hello.sh
```

Add:

```
#!/bin/bash
echo "Hello, $(whoami)! Welcome to Bash scripting!"
```

Save and exit: **Ctrl + O, Enter, Ctrl + X**.

2. Make the Script Executable:

```
chmod +x hello.sh
```

3. Run the Script:

```
./hello.sh
```

Useful Bash Automation Tips

- **Clear Screen:**

```
clear
```

- **Keyboard Shortcut: Ctrl + L.**

- **Clear Screen and Command History:**

```
clear && history -c
```

- **Reset Terminal:**

reset

Managing Command History

1. Clear Current Session's History:

```
history -c
```

2. Save History to a Custom File:

```
history > my_session_history.txt
```

3. Clear and Remove Saved History:

```
history -c  
> ~/.bash_history
```

4. Start a Fresh Bash Session:

```
exec bash
```

Example: Conditional Execution

```
if [ -f filename.txt ]; then  
    echo "File exists."  
else  
    echo "File does not exist."  
fi
```

Batch Normalisation

Links:

- [Batch normalization | What it is and how to implement it](#)

Can be used to handle [vanishing](#) and [exploding gradients](#) problem and [Overfitting](#) problems within Neural network.

First note: [Normalisation vs Standardisation](#)

How does Batch normalisation work?

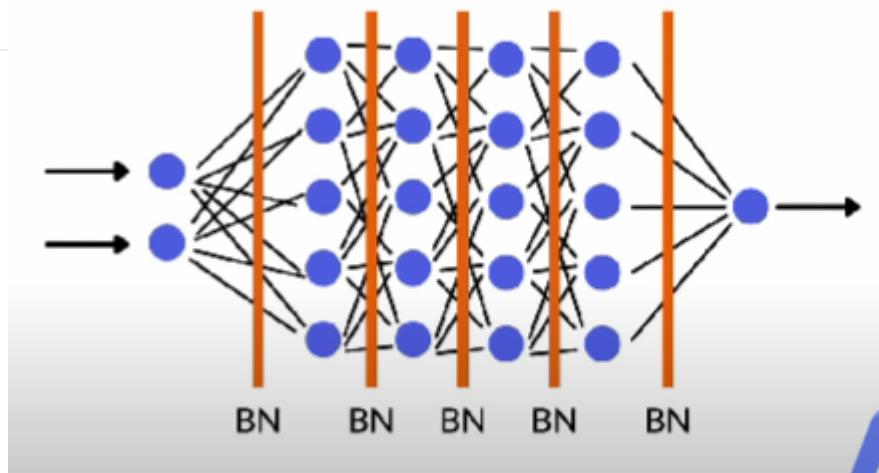
Batch normalisation works by first standardising the inputs, then scales linearly - coefficients determined through training. This occurs between each layer.

Outcomes of this process:

- epochs take longer, but less epochs are required.

Benefits:

- Batch normalisation occurs at each layer, so do not need separate normalisation step for input data.
- What about bias? We do not need bias in BN.



Example:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt

mnist = keras.datasets.mnist
(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()

plt.imshow(X_train_full[12], cmap=plt.get_cmap('gray' ))
X_valid, X_train = X_train_full[:5000] / 255, X_train_full[5000:]/255
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X test = x test/255

model = keras.models.Sequential([
keras.layers.Flatten(input_shape=[28,28]),
keras.layers.Dense(300, activation = "relu"),
keras.layers.Dense(100, activation = "relu"),
keras.layers.Dense(10, activation = "softmax")])
```

Introducing BN into this model.

Do you put BN before or after a activation function? Author of Paper suggests before.

```

# Dont need as have BN now
# X valid, X train = X_train_full[ :5000] / 255, X_train_full[5000:]/255
# y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
# X test = X test/255

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.BatchNormalization(), # normalisation layer.
    keras.layers.Dense(300,use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation('relu'),
    keras.layers.Dense(100,use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation('relu'),
    keras.layers.Dense(10, activation = "softmax")

])

```

Batch Processing

Batch Processing is a technique used to handle and process large datasets efficiently. It works by breaking the data into smaller chunks and processing them together in a single batch.

Apache Spark is the leading technology for batch processing, offering scalable and distributed data processing. It can handle unmanageable data sizes by using parallelism and [Distributed Computing](#)

A key concept in batch processing is **MapReduce**:

- **Map:** Splits the data into smaller, manageable pieces for parallel processing.
- **Reduce:** Aggregates the processed data results from the individual tasks.
- **Order:** The order of Map and Reduce steps is flexible; the primary focus is on splitting and then aggregating data.

Batch processing is widely supported by cloud infrastructures like [Amazon EMR](#) and [Databricks](#), which provide scalable environments for running batch jobs.

[Batch Processing Tags:](#) #data_processing, #data_workflow

Bellman Equations

[What are the Bellman equations that are used in RL?](#)

Equations here may not be accurate.

In reinforcement learning, Bellman's equations are fundamental to understanding how agents make decisions to maximize rewards over time. They are used to describe the relationship between the value of a state and the values of its successor states. There are two main types of Bellman's equations:

1. Bellman Equation for State Value Function (V):

- This equation expresses the value of a state as the expected return starting from that state and following a particular policy. It is defined as:

$$V(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V(s')]$$

- o Here, ($V(s)$) is the value of state (s), ($\pi(a|s)$) is the policy (probability of taking action (a) in state (s)), ($P(s'|s, a)$) is the transition probability to state (s') from state (s) taking action (a), ($R(s, a, s')$) is the reward received, and (γ) is the discount factor.

2. Bellman Equation for Action Value Function (Q):

- o This equation expresses the value of taking an action in a given state under a particular policy. It is defined as:

$$Q(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q(s', a')]$$

- o Here, ($Q(s, a)$) is the value of taking action (a) in state (s), and the other terms are similar to those in the state value function.

Bellman's equations are used in dynamic programming methods like Value Iteration and Policy Iteration to find optimal policies and value functions. They provide a recursive decomposition of the value functions.

Benefits of Data Transformation

- Efficiency: Faster query performance.
- Interoperability: Converting data into the required format for target systems.
- Enrichment: Adding contextual data for better insights.
- Data Quality: Validating, cleansing, and deduplicating data.

Bernoulli

Bias And Variance

Related to Overfitting

Ways to Reduce Bias and Variance:

- Regularisation
- Boosting
- Bagging

What is Bias in Machine Learning?

Bias occurs when a model produces **consistently unfair or inaccurate results**, usually caused during training due to design choices.

What Does High Bias Mean for a Machine Learning Model?

High bias refers to a situation where a model has a strong and often **simplistic assumption** about the underlying data, leading to underfitting.

Introduction

It is biased to the data.

What is the Variance of a Machine Learning Model?

Variance measures how much a **model's predictions change when trained on different subsets** of the training data. It indicates how much the model overfits the training data.

What is the Difference Between Bias and Variance in Machine Learning?

- Bias: The error that occurs when the model cannot learn the true relationship between input and output variables.
- Variance: The error that arises when the model is **too sensitive** to the training data and does not generalize well to new data.

Explain the Bias-Variance Trade-off in the Context of Model Complexity:

The bias-variance trade-off describes the relationship between model complexity and performance.

- High bias (underfitting) occurs when a model is too simple, leading to poor performance on both training and test data.
- High variance (overfitting) happens when a model is overly complex, performing well on training data but poorly on unseen data.

Big Data

The concept of Big Data revolves around datasets that are too large or complex to be managed using traditional data processing techniques. It's characterized by four main attributes, commonly referred to as the **Four V's**:

- Volume: The sheer amount of data being generated, often in terabytes, petabytes, or even exabytes.
- Variety: The diversity in data types, including structured, semi-structured, and unstructured data (e.g., text, images, videos).
- Velocity: The speed at which data is generated and needs to be processed in real-time or near-real-time.
- Veracity: The uncertainty or quality of the data, addressing issues like noise, biases, or incomplete data.

Big Data Technologies

- [Apache Spark](#)
- [Hadoop](#)
- [Scala](#)
- [Databricks](#)

Handling big data involves:

- Distributed storage systems/ [Data Storage](#): Ensuring that data is split and stored across multiple machines for redundancy and speed.
- Processing frameworks: Using tools like [Apache Spark|Spark](#) or Hadoop to process data efficiently in parallel.
- Cloud platforms: Leveraging cloud infrastructure (e.g., Azure, AWS, Google Cloud) to scale resources dynamically based on workload.

[Big Data Tags](#): #big_data, #data_processing

Big O Notation

Big-O Notation is an analysis of the algorithm using [Big – O asymptotic notation](#).

Mostly related to computing rather than storage

Doing things not exponentially, such as copying the same data many times, will save lots of performance and money.

We can express algorithmic complexity using the big-O notation. For a problem of size N:

- A constant-time function/method is “order 1” : $O(1)$
- A linear-time function/method is “order N” : $O(N)$
- A quadratic-time function/method is “order N squared” : $O(N^2)$

Bigquery

cloud-based [Data Warehouse](#)

BigQuery is a fully managed, serverless data warehouse offered by [Google](#) Cloud Platform (GCP). It is designed to handle large-scale data analytics and allows users to run fast SQL queries on massive datasets.

1. **Serverless Architecture:** BigQuery is serverless, meaning users do not need to manage any infrastructure. Google handles the provisioning of resources, scaling, and maintenance, allowing users to focus on analyzing data.
2. **Scalability:** BigQuery can scale to handle petabytes of data, making it suitable for large datasets and complex queries.
3. **SQL Support:** BigQuery supports standard SQL, making it accessible to users familiar with SQL syntax. It also offers extensions for advanced analytics.
4. **Real-Time Analytics:** BigQuery can ingest streaming data and perform real-time analytics, enabling users to gain insights from data as it arrives.
5. **Integration:** BigQuery integrates seamlessly with other Google Cloud services, such as Google Cloud Storage, Google Sheets, and Google Data Studio, as well as third-party tools for data visualization and ETL (Extract, Transform, Load).
6. **Machine Learning:** BigQuery ML allows users to build and deploy machine learning models directly within BigQuery using SQL, without needing to move data to another platform.
7. **Security and Compliance:** BigQuery provides robust security features, including data encryption, identity and access management, and compliance with various industry standards.
8. **Cost-Effective:** BigQuery uses a pay-as-you-go pricing model, where users are charged based on the amount of data processed by queries and the amount of data stored.

Binary Classification

Binary classification is a type of [Classification](#) task that involves predicting one of two possible classes or outcomes. It is used in scenarios where the goal is to categorize data into two distinct groups, such as spam vs. not spam in email filtering or disease vs. no disease in medical diagnosis.

Binder

<https://mybinder.org/>

Boosting

Boosting is a type of [Model Ensemble](#) in machine learning that focuses on improving the accuracy of predictions by building a [sequence of models](#). Each subsequent model focuses on correcting the errors made by the previous ones.

It combines [Weak Learners](#) (models that are slightly better than random guessing) to create a strong learner.

Key Concepts of Boosting:

1. Sequential Learning: Boosting involves training models sequentially. Each new model is trained to correct the errors made by the previous models. This means that the models are not independent of each other; instead, [each model is built on the mistakes of the previous ones](#).
2. Focus on Misclassified Data: As models are trained in sequence, more emphasis is placed on the data points that were misclassified by earlier models. This helps the ensemble model to gradually improve its performance by focusing on the difficult-to-classify instances.
3. [Weak Learners](#): Boosting combines multiple weak learners, which are models that perform slightly better than random guessing. By combining these weak learners, boosting creates a strong learner that has improved accuracy.
4. Examples of Boosting Algorithms: Some well-known boosting algorithms include [Ada boosting](#), [Gradient Boosting](#), and [XGBoost](#). Each of these algorithms has its own approach to boosting, but they all share the core principle of sequentially improving model performance.

Advantages of Boosting:

- Increased Accuracy: By focusing on the errors of previous models, boosting can significantly improve the accuracy of predictions.
- Flexibility: Boosting can be applied to various types of base models and is not limited to a specific algorithm.
- Robustness: Boosting can handle complex datasets and is effective in reducing bias and variance.

Challenges of Boosting:

- Complexity: Boosting models can be more complex and computationally intensive than single models.
- [Interpretability](#): The final model may be harder to interpret compared to simpler models like decision trees.

Bootstrap

sampling with replacement from an original dataset.

Boxplot

A boxplot, also known as a whisker plot, is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It can also highlight outliers in the dataset.

Key Components

Uses:

- Identifying [standardised/Outliers](#).
- Understanding the spread and skewness of the data [Distributions](#).
- Comparing distributions across different categories.
- Need to remove them in order to do [Data Cleansing](#).

Components:

- **Minimum:** The smallest data point excluding outliers.
- **First Quartile (Q1):** The median of the lower half of the dataset.
- **Median (Q2):** The middle value of the dataset.
- **Third Quartile (Q3):** The median of the upper half of the dataset.
- **Maximum:** The largest data point excluding outliers.
- **Outliers:** Data points that fall outside 1.5 times the interquartile range (IQR) above Q3 or below Q1.

Implementing Boxplot in Python

You can create a boxplot in Python using libraries like Matplotlib and Seaborn. Here's how you can do it:

Implementation

```
import matplotlib.pyplot as plt

# Sample data
data = [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

# Create a boxplot
plt.boxplot(data)

# Add title and labels
plt.title('Boxplot Example')
plt.ylabel('Values')

# Show plot
plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

# Create a boxplot
sns.boxplot(data=data)

# Add title and labels
plt.title('Boxplot Example')
plt.ylabel('Values')

# Show plot
plt.show()
```

Business Observability

Business [Model Observability](#) refers to the ability to gain insights into the internal state and performance of a business through the continuous monitoring and analysis of data.

It involves collecting, analyzing, and visualizing data from various sources to understand how different parts of the business are functioning and to identify areas for improvement.

Business observability aims to provide a comprehensive view of operations, customer interactions, and other critical aspects to enable data-driven decision-making.

It helps businesses to detect issues early, optimize operations, enhance customer experiences, and drive growth and innovation.

Key components of business observability include:

1. **Data Collection:** Gathering data from various sources such as customer interactions, sales transactions, operational processes, and external market conditions.
2. **Monitoring:** Continuously tracking key performance indicators (KPIs) and metrics to ensure that the business is operating efficiently and effectively.
3. **Analysis:** Using analytical tools and techniques to interpret the data, identify patterns, and uncover insights that can inform strategic decisions.
4. **Data Visualisation:** Presenting data in an accessible and understandable format, such as dashboards and reports, to facilitate quick comprehension and action by stakeholders.
5. **Feedback Loops:** Implementing mechanisms to use insights gained from observability to make adjustments and improvements in business processes and strategies.

Business Intelligence

Business intelligence (BI) leverages software and services to [transform data](#) into actionable insights that inform an organization's business decisions.

The new term is [Data Engineer](#). The language of a BI engineer is [SQL](#).

Goals of BI

BI should produce a simple overview of your business, boost efficiency, and automate repetitive tasks across your organization. In more detail:

- **rollup capability** - (data) [Visualization](#) over the most important [KPIs][2] (aggregations) - like a cockpit in an airplane which gives you the important information at one glance.
- **Drill-down possibilities** - from the above high-level overview drill down the very details to figure out why something is not performing as planned. **Slice-and-dice or pivot your data from different angles.
- **Single Source of Truth** - instead of multiple spreadsheets or other tools with different numbers, the process is automated and done for all unified. Employees can talk about the business problem instead of the various numbers everyone has. Reporting, budgeting, and forecasting are automatically updated and consistent, accurate, and in timely manner.
- **Empower users**: With the so-called self-service BI, every user can analyze their data instead of only BI or IT persons.

C

Table of Contents

- CI-CD
 - CRUD
 - Career Interest
 - Casual Inference
 - CatBoost
 - Central Limit Theorem
 - Chain of thought
 - Change Management
 - Checksum
 - Chi-Squared Test
 - Choosing a Threshold
 - Choosing the Number of Clusters
 - Class Separability
 - Classification Report
 - Classification
 - Claude
 - Click_Implementation.py
 - Cloud Providers
 - Clustering
 - Clustering_Dashboard.py
 - Clustermap
 - Code Diagrams
 - Columnar Storage
 - Command Line
 - Command Prompt
 - Common Security Vulnerabilities in Software Development
 - Common Table Expression
 - Communication Techniques
 - Communication principles
 - Comparing LLM
 - Components of the database
 - Computer Science
 - Concatenate
 - Conceptual Model
 - Concurrency
 - Confidence Interval
 - Confusion Matrix
 - Continuous Delivery - Deployment
 - Continuous Integration
 - Converting categorical variables to a dummy indicators
 - Convolutional Neural Networks
 - Correlation vs Causation
 - Correlation
 - Cosine Similarity
 - Cost Function
-

Introduction

- Covariance
- Covering Index
- Cron jobs
- Cross Entropy
- Cross Validation
- Cross_Entropy.py
- Cross_Entropy_Single.py
- Crosstab
- Cryptography
- Current challenges within the energy sector
- cleaning terminal path
- conceptual data model

Ci Cd

CI/CD stands for **Continuous Integration** and **Continuous Delivery/Deployment**. It is a set of practices aimed at streamlining and accelerating the **Software Development Life Cycle**. The main goals of CI/CD are to improve software quality, reduce integration issues, and deliver updates to users more frequently and reliably.

Tools and Technologies

- Gitlab
- Docker

Crud

Create,Read,Update,Delete.

Career Interest

This is a portal to notes that I find relevant to my career:

Casual Inference

missing data problem

Catboost

CatBoost is a **Gradient Boosting** library developed by Yandex, designed to handle **categorical** features efficiently and provide robust performance with minimal **Hyperparameter|Hyperparameter tuning**

It is particularly useful in scenarios where datasets contain a significant number of categorical variables.

Key Advantages

1. Handling Categorical Features:

- CatBoost natively processes categorical features without the need for extensive preprocessing like one-hot encoding, which simplifies the workflow and reduces the risk of introducing errors during data preparation.

2. Robustness to Overfitting:

- It employs techniques such as ordered boosting and per-feature scaling to reduce overfitting, making it a reliable choice for complex datasets.

3. Performance:

- CatBoost offers competitive performance with minimal hyperparameter tuning, making it suitable for quick experimentation and deployment.

Implementing CatBoost in Python

To implement CatBoost in Python, you need to install the CatBoost library and then follow these steps:

Step 1: Install CatBoost

You can install CatBoost using pip:

```
pip install catboost
```

Step 2: Import Necessary Libraries

```
import catboost as cb
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Step 3: Prepare Your Data

Assume you have a dataset with features `x` and target `y`. Split the data into training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 4: Identify Categorical Features

Identify the indices of categorical features in your dataset:

```
categorical_features_indices = [0, 1, 2] # Example indices of categorical features
```

Step 5: Create a CatBoost Pool

Create a Pool object for the training data, specifying the categorical features:

```
train_pool = Pool(data=X_train, label=y_train, cat_features=categorical_features_indices)
test_pool = Pool(data=X_test, label=y_test, cat_features=categorical_features_indices)
```

Step 6: Initialize and Train the Model

Initialize the CatBoostClassifier and fit it to the training data:

```
model = CatBoostClassifier(iterations=1000, depth=6, learning_rate=0.1, loss_function='Logloss', verbose=100)
model.fit(train_pool)
```

Step 7: Make Predictions and Evaluate

Make predictions on the test set and evaluate the model's performance:

```
y_pred = model.predict(test_pool)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Central Limit Theorem

The Central Limit Theorem states that the distribution of the sum (or average) of a large number of independent, identically distributed random variables approaches a normal distribution, regardless of the original distribution.

[Why is the Central Limit Theorem important when working with small sample sizes](#)

Key Points

- **Mean of Sampling Distribution:** The mean of the sampling distribution is equal to the mean of the original population.
- **Variance of Sampling Distribution:** The variance of the sampling distribution is the population variance divided by the sample size ((n)), making it (n) times smaller.
- **Applicability:** The CLT applies when calculating the sum or average of many variables, such as the sum of rolled numbers when rolling dice.

Importance

- The CLT allows us to assume normality for various variables, which is crucial for:
 - Confidence intervals
 - Hypothesis testing
 - Regression analysis

[Central Limit Theorem Explain the concept of the Central Limit Theorem.](#);;

Chain Of Thought

Chain of Thought (CoT) reasoning

Asking sequenced questions that guide someone (or yourself) through a reasoning path is a core technique in problem-solving and teaching. Examples:

- "What is the known information?"
- "What is being asked?"
- "What patterns can we observe?"

Introduction

- "What similar problems have we solved before?"

Used in AI systems is a cognitive-inspired framework that improves the performance of large [language models](#) (LLMs) by explicitly guiding the AI through intermediate reasoning steps.

Advantages of Chain of Thought:

- **Improved Interpretability:** Since the model outputs intermediate steps, it's easier for humans to understand how the final answer was reached.
- **Better Performance on Complex Tasks:** CoT allows the model to handle multi-step reasoning more effectively.
- **Easier Debugging:** If there's an error in reasoning, it can be spotted at a specific step in the chain, which aids in model fine-tuning and debugging.

Related to:

- [Model Ensemble](#)

Change Management

Change management is a structured approach to transitioning individuals, teams, and organizations from a current state to a desired future state. It involves

- preparing,
- supporting,
- and helping people to adopt change in order to drive organizational success and outcomes.

Effective change management helps

- minimize resistance,
- improves engagement,
- and increases the likelihood of successful outcomes.

The process typically includes:

1. **Planning:** Identifying the need for change, defining the change, and developing a strategy to implement it.
2. **Communication:** Clearly explaining the reasons for the change, the benefits, and the impact on the organization and its people.
3. **Training and Support:** Providing the necessary training and resources to help employees adapt to the change.
4. **Knowledge Sharing:** Provide training and resources to help teams understand best practices for data quality.
5. **Implementation:** Executing the change plan while managing any resistance or challenges that arise.
6. **Monitoring and Evaluation:** Assessing the effectiveness of the change and making adjustments as needed to ensure successful adoption.
7. **Sustainability:** Ensuring that the change is maintained over time and becomes integrated into the organization's culture and operations.

Why change fails:

- Change is hard, identify the pain points.
- Resistance is why change fails, due to loss aversion, uncertainty, unexpected change when not bought in.

How we can accomplish change:

- Story telling will help.
 - Introduce a hook i.e. can we reduce the processing time for tasks by X amount.
 - Put ourselves in a better position for tomorrow.
-

Checksum

A checksum is a value calculated from a data set that is used to verify the integrity of that data. It acts as a fingerprint for the data, allowing systems to detect errors or alterations that may occur during storage, processing, or transmission.

When data is sent or stored, a checksum is generated based on the contents of the data. This checksum is then sent or stored alongside the data. Upon retrieval or receipt, the checksum is recalculated from the data and compared to the original checksum. If the two checksums match, it indicates that the data has remained unchanged and is likely intact. If they do not match, it suggests that the data may have been corrupted or tampered with.

Checksums are commonly used in various applications, such as:

- **File transfers:** To ensure that files are not corrupted during transfer.
- **Data storage:** To verify that data has not changed over time.
- **Networking:** To check the integrity of packets sent over a network.

Example of a Checksum Calculation

- **Original Data:** Let's say we have the string "Hello, World!".
 - **Checksum Calculation:** A common method for calculating a checksum is to sum the ASCII values of each character in the string.
 - ASCII values:
 - H = 72
 - e = 101
 - l = 108
 - l = 108
 - o = 111
 - , = 44
 - (space) = 32
 - W = 87
 - o = 111
 - r = 114
 - l = 108
 - d = 100
 - ! = 33
 - Sum of ASCII values: $72 + 101 + 108 + 108 + 111 + 44 + 32 + 87 + 111 + 114 + 108 + 100 + 33 = 1,2,0$
 - Let's say we take the modulo 256 of the sum to get the checksum: $1,2,0 \text{ mod } 256 = 1,2,0$
 - **Sending Data:** The original data "Hello, World!" is sent along with the checksum value of 1, 2, 0.
 - **Receiving Data:** Upon receiving the data, the receiver calculates the checksum again using the same method.
 - **Verification:** If the calculated checksum matches the received checksum (1, 2, 0), the data is considered intact. If it does not match, it indicates that the data may have been corrupted during transmission.
-

This is a basic example, and in practice, checksums can be computed using more complex [algorithms](#) (like CRC32, MD5, or SHA-1) to provide better error detection and [Security](#).

Chi-Squared Test

The Chi-squared test is used to determine if there is a significant association between categorical variables. It assesses whether the observed frequencies in a contingency table differ from the expected frequencies, assuming the data is independent.

Choosing A Threshold

The optimal threshold depends on the specific problem and the desired trade-off between different types of errors:

1. Manual Selection: Based on domain expertise or prior knowledge, choose a threshold that seems reasonable.
2. Receiver Operating Characteristic ([ROC \(Receiver Operating Characteristic\)](#)) Curve Analysis: Plot the true positive rate (TPR) against the false positive rate (FPR) for different threshold values. The optimal threshold often lies near the "elbow" of the ROC curve, where a small increase in FPR results in a significant increase in TPR.
3. [Precision-Recall Curve](#) Analysis: Plot the precision against the recall for different threshold values. The optimal threshold often lies near the "elbow" of the precision-recall curve, where a small decrease in precision results in a significant increase in recall.
4. [Cost-Sensitive Analysis](#): Assign different costs to different types of errors (e.g., false positives vs. false negatives) and choose the threshold that minimizes the total cost.

Choosing The Number Of Clusters

The optimal number of clusters ([clustering](#)) depends on the data and the desired level of [granularity](#). Here are some common approaches:

1. Elbow Method: [WCSS and elbow method](#): Plot the within-cluster sum of squares (WCSS) as a function of the number of clusters. The optimal number of clusters is often the point where the WCSS starts to decrease slowly.
2. [Silhouette Analysis](#): Calculate the silhouette coefficient for each data point, which measures how similar a data point is to its own cluster compared to other clusters. The optimal number of clusters 1 is often the one that maximizes the average silhouette coefficient.

Class Separability

If you have a perfectly balanced dataset (unlike [Imbalanced Datasets](#)) but still experience poor [classification accuracy](#), class separability might be an issue due to the following reasons:

1. **Overlapping Classes**: The features of different classes may overlap significantly, making it difficult for the model to distinguish between them. If the decision boundaries are not well-defined, the model may struggle to classify instances correctly.
 2. **Complex Decision Boundaries**: The underlying relationship between the features and the classes may be complex, requiring a more sophisticated model to capture the nuances. If the model is too simple, it may not be able to learn the necessary patterns.
-

3. **Noise in the Data:** If there is a significant amount of noise or irrelevant features in the dataset, it can obscure the true signal, leading to poor classification performance despite balanced class representation.
4. **Insufficient Feature Representation:** The features used for classification may not adequately represent the underlying characteristics that differentiate the classes. This can lead to a lack of separability, even in a balanced dataset.
5. **Model Overfitting or Underfitting:** If the model is overfitting, it may perform well on training data but poorly on unseen data. Conversely, if it is underfitting, it may not capture the complexity of the data, leading to poor accuracy.

Addressing these issues may require exploring different [feature engineering](#) techniques, selecting more appropriate models, or adjusting hyperparameters to improve class separability.

Classification Report

The `classification_report` function in `sklearn.metrics` is used to evaluate the performance of a classification model. It provides a summary of key metrics for each class, including precision, recall, F1-score, and support.

Function Signature

```
sklearn.metrics.classification_report(
    y_true,
    y_pred,
    ,
    labels=None,
    target_names=None,
    sample_weight=None,
    digits=2,
    output_dict=False,
    zero_division='warn'
)
```

Parameters:

- `y_true` : Array of true labels.
- `y_pred` : Array of predicted labels.
- `labels` : (Optional) List of label indices to include in the report.
- `target_names` : (Optional) List of string names for the labels.
- `sample_weight` : (Optional) Array of weights for each sample.
- `digits` : Number of decimal places for formatting output.
- `output_dict` : If `True`, return output as a dictionary.
- `zero_division` : Sets the behavior when there is a zero division (e.g., 'warn', 0, 1).

Metrics Explained

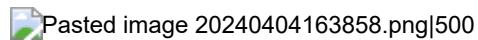
- **Precision:** The ratio of correctly predicted positive observations to the total predicted positives. It indicates the quality of the positive class predictions.

- **Recall** (Sensitivity): The ratio of correctly predicted positive observations to all actual positives. It measures the ability of a model to find all relevant cases.
- **F1 Score**: The weighted average of precision and recall. It is a better measure than accuracy for imbalanced classes.
- Support: The number of actual occurrences of the class in the specified dataset.

Resources

In [ML_Tools](#) see: [Evaluation_Metrics.py](#)

[official documentation](#).



Classification

Classification is a type of [Supervised Learning](#) in machine learning, where the algorithm learns from labeled data to predict which category or class a new, unlabeled data point belongs to. The goal is to assign the correct label to input data based on patterns learned from the training set.

Examples of Classifiers

Classifier: A model used for classification tasks, predicting discrete labels or categories. For example, determining whether an email is spam or not, or identifying the species of a flower based on its features. This contrasts with a Regressor ([Regression](#)), which predicts continuous values.

[Naive Bayes](#)

[Decision Tree](#)

[Support Vector Machines](#)

[K-nearest neighbours](#)

[Neural network](#)

[Model Ensemble](#)

Choosing a Classifier Algorithm

1. Data Characteristics: Some algorithms work better on structured data, while others perform better on unstructured data.
2. Problem Complexity: Simple classifiers for straightforward problems, complex models for intricate tasks.
3. Model Performance: Consider accuracy and speed requirements.
4. Model [Interpretability](#): Some models, like decision trees, are easier to interpret, while others, like neural networks, can be more challenging.
5. Model Scalability: Large datasets need scalable models like SVM or Naive Bayes.
6. Model Flexibility: Algorithms like KNN are flexible when the data distribution is unknown.

Use Cases of Classification

-
7. Object Recognition: Classifying objects in images (e.g., identifying a cat or a dog).
 8. Spam Filtering: Classifying emails as either spam or legitimate.
 9. Medical Diagnosis: Using patient symptoms and test results to classify diseases.

Claude

Claude is better for code and uses Artifact for tracking code changes.

Claude is crazy see: <https://youtu.be/RudrWy9uPZE?t=473>

Click_Implementation.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Click_Implementation.py

This script implements a command-line interface (CLI) tool using Python's `click` library. The CLI allows users to interact with a JSON file, enabling them to view keys, retrieve values, and update key-value pairs.

Functionality Overview

1. CLI Initialization (`cli` function)
 - Serves as the main command group.
 - Accepts a JSON file (`document`) as an argument.
 - Reads the JSON file and stores its content in `ctx.obj`, making it accessible to all subcommands.
2. Displaying Keys (`show_keys` command)
 - Lists all top-level keys in the JSON document.
3. Retrieving a Value (`get_value` command)
 - Accepts a key as an argument.
 - Prints the corresponding value if the key exists; otherwise, prints "Key not found".
4. Updating a Value (`update_value` command)
 - Requires `-k/--key` (key to update) and `-v/--value` (new value).
 - Updates the key's value in memory.
 - Saves the updated JSON data back to the file.

Example Usage

1. Viewing Keys

```
python script.py data.json show_keys
```

Example Output (if `data.json` contains `{"name": "Alice", "age": 30}`):

```
Keys: ['name', 'age']
```

2. Retrieving a Value

```
python script.py data.json get_value name
```

Output:

```
Alice
```

3. Updating a Value

```
python script.py data.json update_value -k name -v Bob
```

Modifies `data.json` to:

```
{
    "name": "Bob",
    "age": 30
}
```

Cloud Providers

Among the biggest cloud providers are [AWS](#), [Microsoft Azure](#), [Google Cloud](#).

Whereas [Databricks](#) ([Databrick](#)) and [Snowflake](#) provide dedicated [Data Warehouse](#) and [Data Lakehouse](#)|[Lakehouse](#) solutions

Features

[Scaling](#) [Server Load Balancing](#) [Memory Caching](#)

Clustering

Clustering involves grouping a set of data points into subsets or clusters based on inherent patterns or similarities. It is an [Unsupervised Learning](#) technique used for tasks like customer segmentation and [standardised/Outliers|anomalies](#) detection. The primary goal of clustering is to organize data by grouping similar items.

Applications

- Customer Segmentation: Group customers with similar purchasing behavior or demographics for targeted marketing.
- Image Segmentation: Group pixels in an image based on color or texture to identify objects or regions.
- [Anomaly Detection](#): Identify clusters of normal behavior to detect anomalies that deviate significantly from these clusters.

Methods

- K-means
- DBScan
- Hierarchical Clustering
- Gaussian Mixture Models

Interpretability

Feature Scaling: Essential for bringing features to the same scale, as clusters may appear distorted without it.

```
from sklearn.preprocessing import scale
from sklearn.preprocessing import MinMaxScaler
```

Use clustering to find Correlation between features. Utilize a Dendograms to visualize the relationship between features.

Clustering_Dashboard.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Clustering_Dashboard.py

Clustermap

Related to: Preprocessing|Preprocess

- Purpose: Identify which features are most similar using Dendograms.
- Visualization: Regions of color show clustering, similar to a heatmap.
- Functionality: Performs clustering on both rows and columns.

Requirements: Input should be numerical; data needs to be scaled.

```
import seaborn as sns
sns.clustermap(x_scaled, cmap='mako', standard_scale=0) # 0 for rows, 1 for columns
```

Resources

- [Video Explanation](#)
- [Seaborn Clustermap Documentation](#)

Code Diagrams

[Documentation & Meetings](#)

There are class diagrams showing the hierarchy of classes Classes (Object orientated).

Done in Mermaid.

Overall [Architecture Diagram](#): showing how software components interact.

[Sequence diagram](#) of components interact.

Sequence diagram: how the components interact Architecture diagram : main components fitting together

Columnar Storage

A database storage technique that stores **data by columns** rather than rows,

Useful for read-heavy operations and **large-scale data analytics**, as it enables the retrieval of specific columns without the need to access the entire row.

Columnar Storage Example (Analytical Workloads)**:

order_id	customer_id	order_date	order_amount
1	101	2024-10-01	\$100
2	102	2024-10-02	\$150
3	103	2024-10-03	\$200

In **columnar storage**, the data would be stored by columns, like:

- customer_id : [101, 102, 103]

If you're querying for the total sales (`order_amount`) in a specific period, only the `order_amount` and `order_date` columns are accessed.

Use case: **Data Analytics/OLAP (Online Analytical Processing)**

- Running a query to get the **total sales for October** only needs to scan the `order_amount` and `order_date` columns, rather than scanning entire rows, faster [Querying](#)

Command Line

The command line is a text-based interface used to interact with a computer's operating system or software. It allows users to execute commands, run scripts, and perform various tasks.

[PowerShell](#)

[Powershell vs Bash](#)

[Bash](#)

[Command Prompt](#)

Command Prompt

Command Prompt (cmd) is a command-line interpreter on Windows systems that allows users to execute commands to perform various basic tasks. Below are some common tasks that can be performed in cmd, along with examples:

Related to:

- [Bash](#)

1. Navigating the File System

- **Changing Directories:**

```
cd C:\path\to\directory
```

Changes the current directory to `C:\path\to\directory`.

- **Listing Files and Directories:**

```
dir
```

Lists the files and directories in the current directory.

2. Managing Files and Directories

- **Creating a Directory:**

```
mkdir newfolder
```

Creates a new directory named `newfolder`.

- **Deleting a Directory:**

```
rmdir /s /q newfolder
```

Deletes the directory `newfolder` and its contents. The `/s` flag removes all directories and files in the specified directory, and the `/q` flag runs the command quietly without asking for confirmation.

- **Copying Files:**

```
copy C:\source\file.txt D:\destination\
```

Copies `file.txt` from the `C:\source` directory to the `D:\destination` directory.

- **Renaming Files:**

```
ren oldfile.txt newfile.txt
```

Renames `oldfile.txt` to `newfile.txt`.

- **Deleting Files:**

```
del file.txt
```

Deletes `file.txt`.

3. Viewing and Managing System Information

- **Viewing IP Configuration:**

```
ipconfig
```

Displays the current network configuration.

- **Viewing System Information:**

```
systeminfo
```

Provides detailed system information including OS version, hardware details, and network configurations.

4. Managing Processes

- **Viewing Running Processes:**

```
tasklist
```

Lists all currently running processes.

- **Killing a Process:**

```
taskkill /F /PID 1234
```

Terminates the process with the Process ID (PID) `1234`. The `/F` flag forces the process to terminate.

5. Networking Commands

- **Pinging a Server:**

```
ping www.example.com
```

Sends ICMP Echo Request packets to the specified host and displays the response.

- **Tracing Route to a Server:**

```
tracert www.example.com
```

Traces the route packets take to the specified host.

6. Batch File Scripting

- **Creating and Running a Simple Batch File:**

- Create a file named `example.bat` with the following content:

```
@echo off  
echo Hello, World!  
pause
```

- o Run the batch file:

```
example.bat
```

This batch file prints "Hello, World!" to the console and waits for the user to press a key before closing.

7. Environment Variables

- **Viewing Environment Variables:**

```
set
```

Displays all current environment variables and their values.

- **Setting an Environment Variable:**

```
set MYVAR=Hello
```

Sets an environment variable `MYVAR` with the value `Hello`.

8. Disk Operations

- **Checking Disk Usage:**

```
chkdsk C:
```

Checks the file system and file system metadata of the C: drive for logical and physical errors.

- **Formatting a Disk:**

```
format D: /FS:NTFS
```

Formats the D: drive with the NTFS file system.

9. Echoing Messages

- **Displaying a Message:**

```
echo Hello, World!
```

Prints `Hello, World!` to the console.

10. Redirecting Output

- Redirecting Command Output to a File:

```
dir > output.txt
```

Redirects the output of the `dir` command to `output.txt`.

These examples illustrate some of the basic functionalities of Command Prompt. While cmd is less powerful compared to [PowerShell](#), it remains useful for simple file system navigation, file management, and running legacy scripts.

Common Security Vulnerabilities In Software Development

[Security](#) vulnerabilities can be encountered and mitigated in [Software Development Portal](#).

In this note describe potential security risks in their applications.

Useful Tools

- [tool.bandit](#)

Examples

Command Injection

General Description: Command injection is a security vulnerability that occurs when an attacker is able to execute arbitrary commands on the host operating system via a vulnerable application. This typically happens when user input is improperly handled and passed to a system shell.

Example: The `dangerous_subprocess` function uses `subprocess.call` with `shell=True`, which can lead to command injection if user input is not properly sanitized.

```
import subprocess
def dangerous_subprocess(user_input):
    subprocess.call(user_input, shell=True)
```

Mitigation:

- Avoid using `subprocess.call` with `shell=True`. Use `subprocess.run` or `subprocess.call` with a list of arguments.
- Validate and sanitize user inputs ([Input is Not Properly Sanitized](#)).

Hardcoded Password

General Description: Hardcoded passwords refer to credentials that are embedded directly in the source code. This practice is insecure as it exposes sensitive information and makes it difficult to change passwords without modifying the code.

Example: The `hardcoded_password` function contains a hardcoded password, which is a common security issue.

Introduction

```
def hardcoded_password():
    password = "123456"
    return password
```

Mitigation:

- Use environment variables or configuration files to store sensitive information.
- Consider using a secrets management tool.

Use of eval

General Description: The `eval` function in Python evaluates a string as a Python expression. If not properly controlled, it can execute arbitrary code, leading to security vulnerabilities.

Example:

- The `unsafe_eval` function uses `eval`, which can execute arbitrary code if the input is not controlled.

```
def unsafe_eval(user_input):
    return eval(user_input)
```

Mitigation:

- Avoid using `eval`. Use safer alternatives like `ast.literal_eval`.
- Ensure input is strictly controlled and sanitized.

Insecure Deserialization

General Description: Insecure deserialization occurs when untrusted data is used to reconstruct objects. This can lead to arbitrary code execution, data tampering, or other malicious activities.

Example:

- The `insecure_deserialization` function uses `pickle.loads`, which can be exploited if untrusted data is deserialized.

```
import pickle
def insecure_deserialization(data):
    return pickle.loads(data)
```

Mitigation:

- Avoid using `pickle` for untrusted data. Use safer formats like JSON ([Why JSON is Better than Pickle for Untrusted Data](#)).
- Ensure data is from a trusted source.

SQL Injection

Cross-Site Scripting (XSS)

General Description: Cross-Site Scripting (XSS) is a security vulnerability that allows an attacker to inject malicious scripts into content from otherwise trusted websites. It occurs when an application includes untrusted data in a web page without proper validation or escaping.

Example:

Introduction

- The `display_user_input` function directly inserts user input into HTML, which can lead to XSS if the input is not properly sanitized.

```
<div>
<%= user_input %>
</div>
```

Mitigation:

- Escape user input before rendering it in HTML.
- Use security libraries or frameworks that automatically handle escaping

Common Table Expression

A Common Table Expression (CTE) is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

The CTE can also be used in a [Views](#). Serve as temporary views for a single [Querying|Queries](#).

```
WITH cte_query AS
(SELECT ... subquery ...)
SELECT main query ... FROM/JOIN with cte_query ...
```

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/ExplorationsSQLite/Utilities/Common_Table_Expression.ipynb

Non-Recursive CTE

The non-recursive are simple where CTE is used to [avoid SQL duplication](#) by referencing a name instead of the actual SQL statement. See [Views](#) simplification usage.

```
WITH avg_per_store AS
(SELECT store, AVG(amount) AS average_order
 FROM orders
 GROUP BY store)

SELECT o.id, o.store, o.amount, avg.average_order AS avg_for_store
 FROM orders o
 JOIN avg_per_store avg
 ON o.store = avg.store;
```

Recursive CTE

CTEs can be used in [Recursive Algorithm](#). The recursive query calls itself until the query satisfied the condition. In a recursive CTE, we should provide a where condition to terminate the recursion.

A recursive CTE is useful in querying hierarchical data such as organization charts where one employee reports to a manager or multi-level bill of materials when a product consists of many components, and each component itself also consists of many other components.

```

WITH levels AS (
    SELECT
        id,
        first_name,
        last_name,
        superior_id,
        1 AS level
    FROM employees
    WHERE superior_id IS NULL
    UNION ALL
    SELECT
        employees.id,
        employees.first_name,
        employees.last_name,
        employees.superior_id,
        levels.level + 1
    FROM employees, levels
    WHERE employees.superior_id = levels.id
)
SELECT *
FROM levels;

```

Communication Techniques

Overview

Using these structured communication bridges can enhance clarity and engagement, especially in spontaneous or high-stakes discussions.

Tips for Using Communication Bridges

1. Start Small: Begin by integrating 2-3 bridges that feel natural to you.
2. Observe Reactions: Notice how listeners respond when you clarify changes, summarize key points, or highlight actions.
3. Practice Consistency: Make these bridges a regular part of your speaking style.

[Speak More Clearly: 8 Precise Steps to Improve Communication](#)

1. Context Bridge

- Purpose: Aligns everyone by setting the context before diving into details.
- How to Use: Start with phrases like:
 - "At a high level..."
 - "This is our goal..."
 - "**The main problem is...**"
- Effect: Helps to focus thoughts and prevents initial rambling.

2. Change Bridge

- Purpose: Emphasizes shifts, trends, or significant moments in the discussion.
- How to Use: Use phrases that highlight changes, such as:
 - "Here's the before, and here's the after..."
 - "We're shifting from X to Y..."
 - "We are at a tipping point..."
- Effect: Grabs attention by making the change clear.

3. Insight Bridge

- Purpose: Shares deeper insights or unique perspectives, creating "aha" moments.
- How to Use: Key phrases include:
 - "Counterintuitively, ..."
 - "Here's what most people miss..."
 - "The deeper insight is..."
 - "The key point here is..."
- Effect: Signals that you've thought deeply, which moves the conversation forward.

4. Analysis Bridge

- Purpose: Anchors discussion in evidence, keeping it grounded.
- How to Use: Reference specific data points or comparisons with:
 - "The evidence shows..."
 - "The data indicates..."
 - "When we compared X and Y..."
- Effect: Focuses on facts, minimizing loss of direction.

5. Logical Transition Bridge

- Purpose: Provides a clear flow in the conversation, avoiding confusion.
- How to Use: Classic transitions include:
 - "First, second, third..."
 - "This leads to..."
 - "On the other hand..."
- Effect: Helps listeners follow along without losing the thread.

6. Summary Bridge

- Purpose: Ensures that key messages stay clear, especially in long discussions.
- How to Use: Frequently summarize main points with phrases like:
 - "The bottom line is..."
 - "If you remember one thing, it's this..."
 - "To bring it back to the goal..."
- Effect: Reinforces the main message throughout the discussion.

7. Refinement Bridge

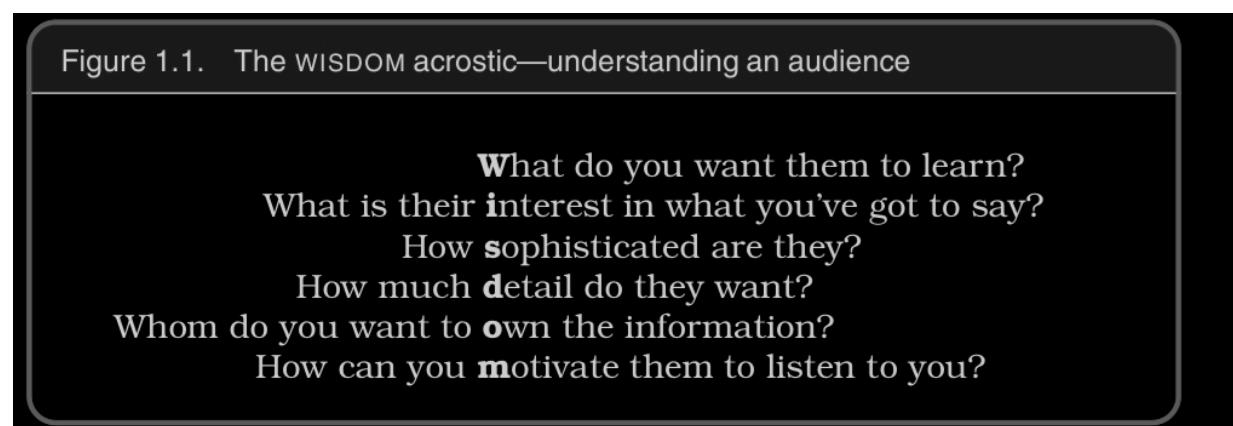
- Purpose: Allows for clarification or expansion of ideas as needed.
- How to Use: Rephrase or elaborate with:
 - "Let me break this down further..."

- "Another way of looking at it is..."
- "A useful analogy might be..."
- Effect: Clarifies complex points, helping everyone understand the core message.

8. Action Bridge

- Purpose: Concludes with actionable steps, defining the next moves.
- How to Use: Conclude with statements like:
 - "Our immediate priority is..."
 - "Here's what we'll do next..."
 - "The deliverables are..."
- Effect: Ends discussions with clear direction and accountability.

Communication Principles



We feel that the only way to develop software reliably, and to make our developments easier to understand and maintain, is to follow what we call the *DRY* principle:

EVERY PIECE OF KNOWLEDGE MUST HAVE A SINGLE, UNAMBIGUOUS, AUTHORITATIVE REPRESENTATION WITHIN A SYSTEM.

Why do we call it *DRY*?

Tip 11

DRY—Don't Repeat Yourself

Comparing LIm

Use Imarena.ai as a bench marking tool.

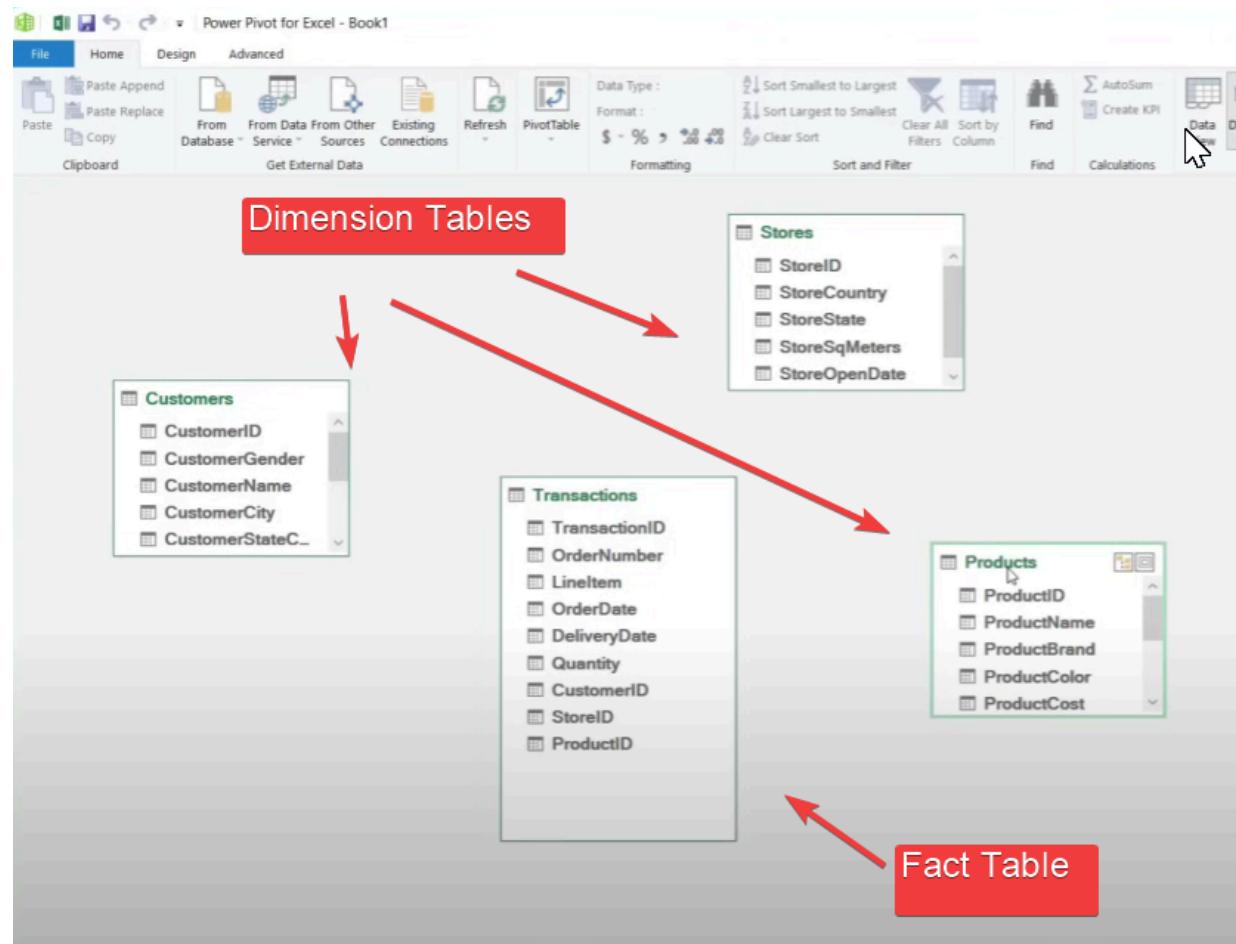
LLM

web dev arena

text to image leader board

Components Of The Database

Fact Table in main table that Dimension Table connect to them.



Computer Science

Algorithms

Concatenate

Conceptual Model

Conceptual Model

- Entities: Customer, Order, Book
- Relationships: Customers place Orders, Orders include Books

Conceptual Model

- Focuses on high-level business requirements.

- Defines important data entities and their relationships.
 - Tools: [ER Diagrams](#), ER Studio, DbSchema.
-

Concurrency

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLite/Transactions/Concurrency.ipynb

Confidence Interval

A confidence interval is a range of values, derived from sample data, that is likely to contain the true population parameter. It is associated with a confidence level, such as 95%, indicating the probability that the interval captures the true parameter.

Key Points

- **Confidence Level:** The likelihood that the interval includes the true parameter (e.g., 95%).
- **Purpose:** Quantifies the uncertainty of an estimate, providing a range rather than a single value.

Example

- A 95% confidence interval for a mean of (50, 60) suggests that, in repeated sampling, 95% of such intervals would contain the true mean.

Confusion Matrix

A Confusion Matrix is a table used to evaluate the performance of a [Classification](#) model. It provides a detailed breakdown of the model's predictions across different classes, showing the number of true positives, true negatives, false positives, and false negatives.

Purpose

- The confusion matrix helps identify where the classifier is making errors, indicating where it is "confused" in its predictions.

Structure

		Predicted	
		Negative (N)	Positive (P)
Actual	Negative -	True Negative (TN)	False Positive (FP) Type I Error
	Positive +	False Negative (FN) Type II Error	True Positive (TP)

Structure

- True Positives (TP): Correctly predicted positive instances.
- False Positives (FP): Incorrectly predicted positive instances (Type 1 error).
- True Negatives (TN): Correctly predicted negative instances.
- False Negatives (FN): Incorrectly predicted negative instances (Type 2 error).

Metrics

- **Accuracy**: The overall percentage of correct predictions. In this case, the accuracy is 78.3%.
- **Precision**: The ratio of true positives to all positive predictions (including both TPs and FPs). In this case, the precision for class 0 is 85.7% and the precision for class 1 is 66.4%.
- **Recall**: The ratio of true positives to all actual positive cases (including both TPs and FNs). In this case, the recall for class 0 is 80.6% and the recall for class 1 is 74.1%.
- **F1 Score**: A harmonic average of precision and recall. In this case, the F1-score for class 0 is 83.0% and the F1-score for class 1 is 70.0%.
- **Specificity**
- **Recall**

Further Examples



Example Code

```
from sklearn.metrics import confusion_matrix

# Assuming y_train and y_train_pred are your true and predicted labels
conf_matrix = confusion_matrix(y_train, y_train_pred)
print(conf_matrix)
```

Example Output:

```
array([[377, 63],
       [ 91, 180]], dtype=int64)
```

Continuous Delivery Deployment

Continuous Delivery

- Ensures that code changes are automatically prepared for a release to production.
- Builds, tests, and releases are automated, but the deployment is manual.

Continuous Deployment:

- Extends continuous delivery by automating the deployment process.
- Every change that passes the automated tests is deployed to production automatically.

[Model Deployment](#)

A continuous integration and continuous deployment (CI/CD) pipeline is **a series of steps that must be performed in order to deliver a new version of software**

Continuous Integration

- Developers frequently integrate code into a shared repository.
- Automated builds and tests are run to detect issues early.
- Encourages smaller, more manageable code changes.

Converting Categorical Variables To A Dummy Indicators

Convolutional Neural Networks

Convolutional networks, or CNNs, are specialized [Deep Learning](#) architectures designed for processing data with grid-like structures, such as images.

They use convolutional layers with learnable filters to extract spatial features from the input data. The convolutional operation involves sliding these filters across the input, performing element-wise multiplications and summations to create feature maps.

Introduction

CNNs are particularly effective for image classification, object detection, and image segmentation tasks.

Primarily used in image recognition and processing tasks. CNNs use convolutional layers to automatically detect spatial patterns in images, like edges and textures.

Pooling:

The idea of pooling in convolutional neural networks is to do two things:

- Reduce the number of parameters in your network (pooling is also called “down-sampling” for this reason)
- To make feature detection ([Feature Extraction](#)) more robust by making it more impervious to scale and orientation changes
- shrink multiple data to single points.



Correlation Vs Causation

What is the meaning of [Correlation](#) does not imply causation?

Correlation measures the statistical association between two variables, while causation implies a cause-and-effect relationship.

- **Correlation:** Indicates an association between variables but does not imply that changes in one variable cause changes in the other.
- **Causation:** Suggests a direct cause-and-effect relationship between variables, requiring experimentation to establish.

Correlation

Use in understanding relationships between variables in data analysis.

While it helps identify associations, it's important to remember that **correlation does not imply causation.**

Visualization tools like heatmaps and clustering can aid in identifying and interpreting these relationships effectively.

- What is Correlation?: A measure of the strength and direction of the relationship between two variables.

Description

- Correlation measures the relationship between two variables, indicating how they change together. It ranges from -1 to 1:
 - -1: Perfect negative correlation
 - 0: No correlation
 - 1: Perfect positive correlation

Key Points

- **Correlation vs Causation:** Correlation does not imply causation. While correlation highlights associations, causation establishes a direct influence.
- **Significance:** Correlation values < -0.5 or $> +0.5$ are considered significant.
- **Impact of Outliers:** [standardised/Outliers](#) can distort correlation results.

- Standardization: Correlation is a standardized version of [Covariance](#).

Model Preparation

Feature Selection:

- Identify features correlated with the target. If all are correlated, keep all.
- For features correlated with each other, consider dropping one to avoid redundancy.
- If two features are highly correlated with the target, both can be retained.

If two variables are strongly positively correlated, it often makes sense to drop one of them to simplify the model.

This is because [highly correlated variables can introduce redundancy](#), leading to [multicollinearity](#) in regression models.

By removing one of the correlated variables, you can:

1. Reduce Complexity: Simplifying the model by reducing the number of predictors can make it easier to interpret and manage.
2. Improve Stability: Reducing multicollinearity can lead to more stable and reliable coefficient estimates.
3. Enhance Performance: In some cases, removing redundant features can improve the model's predictive performance by reducing overfitting.

However, it's important to ensure that the variable you choose to keep is the one that is more relevant or has a stronger theoretical justification for inclusion in the model.

Viewing Correlations

- Use [Heatmap](#) or [Clustering](#) to visualize correlations between features.

Example Code

To find the correlation between two features:

```
df[['var1', 'target']]#var1-target).groupby(['var1'], as_index=False).mean().sort_values(by='target', ascending=False)
```

Cosine Similarity

Cosine similarity is a [Metric](#) used to measure how similar two vectors are by calculating the cosine of the angle between them. It ranges from -1 to 1, where 1 indicates identical orientation, 0 indicates orthogonality, and -1 indicates opposite orientation.

Cosine similarity is commonly used in

- text analysis,
- information retrieval,
- recommendation systems to compare document similarity, user preferences, or item features.

In [Binary Classification](#), cosine similarity can be used as a feature to help distinguish between two classes. For instance, in text classification tasks, you might represent documents as vectors using techniques like [TF-IDF](#).

Cost Function

The concept of a Cost Function is central to [Model Optimisation](#), particularly in training models.

A cost function, also known as a loss function or error function, is a mathematical function used in optimization and machine learning to measure the difference between predicted values and actual values. It quantifies the error or "cost" of a model's predictions. The goal of many machine learning algorithms is to minimize this cost function, thereby improving the accuracy of the model. Common examples of cost functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks.

1. Relation to [Loss Function](#): The cost function is related to the loss function. While the loss function measures the error for a single data point, the cost function typically aggregates these errors over the entire dataset, often by taking an average. See [Loss versus Cost function](#).
2. Parameter Space ([Model Parameters](#)) and Surface Plotting: By plotting the cost function over the parameter space, you can visualize how different parameter values affect the cost. This surface can have various peaks and valleys representing different levels of error.
3. [Gradient Descent](#): This is an optimization algorithm used to find the minimum of the cost function. By iteratively adjusting the parameters in the direction that reduces the cost, gradient descent helps in finding the optimal parameters for the model.
4. Caveats: The cost function is dependent on the dataset and may not always have an explicit formula. This means that the shape of the cost function surface can vary greatly depending on the data, and finding the global minimum can be challenging.



[Reward Function](#): Mentioned as the opposite of a cost function, typically used in [Reinforcement learning](#) to indicate the desirability of an outcome.

Covariance

In statistics, covariance is a measure of the degree to which two random variables change together. It indicates the direction of the linear relationship between the variables. Specifically, covariance can be defined as follows:

- **Positive Covariance**: If the covariance is positive, it means that as one variable increases, the other variable tends to also increase. Conversely, if one variable decreases, the other variable tends to decrease as well.
- **Negative Covariance**: If the covariance is negative, it indicates that as one variable increases, the other variable tends to decrease, and vice versa.
- **Zero Covariance**: A covariance close to zero suggests that there is no linear relationship between the two variables.

The formula for calculating the covariance between two random variables X and Y is given by:

$$\text{Cov}(X, Y) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})$$

where:

- X_i and Y_i are the individual sample points,
- \bar{X} and \bar{Y} are the means of X and Y respectively,

- n is the number of data points.

Covariance is used in:

- in the calculation of [correlation](#) coefficients
- and in multivariate statistics, such as in [Gaussian Mixture Models](#) where it helps describe the shape and orientation of the data distribution.

Covering Index

Like an [Database Index|Index](#) but for partial indexes?

Cron Jobs

Cross Entropy

Cross entropy is a [Loss function](#) used in [Classification](#) tasks, particularly for [categorical data](#). The cross entropy loss function is particularly effective for multi-class classification problems, where the goal is to assign an input to one of several categories.

Cross entropy measures confidence.

Cross entropy works by measuring the (difference/loss) [dissimilarity between two probability distributions](#): the true distribution (actual class labels) and the predicted distribution (model's output probabilities).

Fit of Predictions:

- A low cross entropy loss means the predicted probabilities are close to the true labels (e.g., assigning high probability to the correct class).
- A high loss indicates significant divergence, meaning the model's predictions are inaccurate or uncertain.

By minimizing cross entropy, the model learns to produce probability distributions that closely match the true class distributions, thereby improving its classification [accuracy](#).

1. Probability Distributions: In a classification task, the model outputs a probability distribution over the possible classes for each input. For example, in a three-class problem, the model might output probabilities like [0.7, 0.2, 0.1] for classes A, B, and C, respectively.
2. True Labels: The true class label is represented as a one-hot encoded vector. If the true class is A, the vector would be [1, 0, 0].
3. Cross Entropy Calculation calculates the loss by comparing the predicted probabilities with the true labels. The formula for cross entropy loss L for a single instance is:

$$L = - \sum_{i=1}^N y_i \log(p_i)$$

where:

- N is the number of classes.
- y_i is the true label (1 if the class is the true class, 0 otherwise).
- p_i is the predicted probability for class i .

4. Interpretation: The cross entropy loss increases as the predicted probability diverges from the actual label. If the model assigns a high probability to the correct class, the loss is low. Conversely, if the model assigns a low probability to the correct class, the loss is high.
5. Optimization: During training, the model's parameters are adjusted to minimize the cross entropy loss across all training examples. This process helps the model improve its predictions over time.

Where is it used

Cross entropy is widely used in classification for several reasons:

Probabilistic Modeling:

- It directly aligns with the goals of probabilistic classifiers, as it measures how well the predicted probability distribution matches the true distribution.

Focus on Confidence:

- Encourages the model to assign higher probabilities to the correct classes, improving not just accuracy but also confidence.

Optimization Efficiency:

- Cross entropy is smooth and convex for logistic regression-like models, enabling efficient gradient-based optimization.

Multi-Class Support:

- Works seamlessly in multi-class scenarios where the true labels are one-hot encoded and predictions are probability distributions.

Implementation

In [ML_Tools](#) see:

- [Cross_Entropy_Single.py](#)
- [Cross_Entropy.py](#)
- [Cross_Entropy_Net.py](#)

Cross Validation

Cross-validation is a statistical technique used in machine learning to [assess how well a model will generalize](#) to an independent dataset. It is a crucial step in the model-building process because it helps ensure that the model is not [overfitting](#) or underfitting the training data.

- Cross-validation is a technique used in machine learning and statistics to evaluate the performance ([Model Optimisation](#)) of a predictive model.
- It provides a robust evaluation by splitting the training data into smaller chunks and training the model multiple times.

Introduction

- K-Fold Cross-Validation: Involves dividing the dataset into (k) equal-sized subsets (called "folds") and using each fold as a validation set once, while the remaining ($k-1$) folds are used for training.
- The model's performance is averaged across all (k) folds to provide a more robust estimate of its generalization performance.

Common Variations

- K-Fold Cross-Validation: The most common method, where the data is split into (k) folds and the model is trained (k) times, each time using a different fold as the validation set.
- Stratified K-Fold: Ensures each fold has a similar proportion of class labels, important for imbalanced datasets.
- Repeated K-Fold: Repeats the process multiple times with different random splits for more robust results.
- Leave-One-Out Cross-Validation (LOOCV): Each data point is used once as a test set while the rest serve as the training set.

How Cross-Validation Fits into Building a Machine Learning Model

1. **Model Evaluation**: Used to evaluate the performance of different models or algorithms to choose the best one.
2. **Hyperparameter Tuning**: Provides a reliable performance metric for each set of hyperparameters.
3. **Model Validation**: Ensures consistent performance across different subsets of data.
4. **Bias and variance tradeoff**: Helps in understanding the tradeoff between bias and variance, guiding the choice of model complexity.

Advantages:

- Reduced Bias: Offers a more reliable performance estimate compared to using a single validation set.
- Efficient Data Use: All data is used for both training and validation.
- Prevents Overfitting: By evaluating on multiple folds, it can detect if the model is overfitting to the training data.

Choosing (k)

- Common values: 5 or 10
- Higher (k) leads to more accurate estimates but increases computation time.
- Consider dataset size and complexity when choosing (k).

Code Implementation

In [ML_Tools](#) see:

- [KFold_Cross_Validation.py](#)

Cross-Validation Strategy in Time Series

All notebooks use cross-validation based on `TimeSeriesSplit` to ensure proper evaluation of performance with no [Data Leakage](#). This method ensures that training and test data are split while maintaining the chronological order of the data.

Cross_Entropy.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Selection/Model_Evaluation/Classification/Cross_Entropy.py

Generalized Script Description:

1. **Dataset:** Uses the Iris dataset from `sklearn` to classify flower species.
2. **Preprocessing:** One-hot encodes the target labels and splits the data into training and testing sets.
3. **Model:** Trains a multinomial logistic regression model to predict probabilities for each class.
4. **Cross Entropy Calculation:** Computes cross entropy loss for all predictions in the test set.
5. **Visualization:** Plots a histogram to show the distribution of loss values across the test samples.
6. **Summary Statistics:** Outputs mean, median, maximum, and minimum loss values for analysis.

This approach provides insight into the model's performance by analyzing the spread and typical values of cross entropy loss over multiple predictions.

Strengths:

1. **Real-World Dataset:** The Iris dataset is well-known and intuitive, making it easier to follow and validate the results.
2. **Generalization:** The script calculates the cross entropy loss for multiple predictions, demonstrating the loss function in a real-world, multi-class classification scenario.
3. **Insights Through Visualization:** The histogram of losses provides a clear picture of how well the model performs across different test samples.
4. **Statistical Summary:** The inclusion of mean, median, max, and min loss values gives a quick overview of the model's performance.
5. **Numerical Stability:** The small epsilon value in the log computation ensures stability when dealing with probabilities close to zero.
6. **Reproducibility:** Using `sklearn`'s preprocessing and modeling tools ensures that the example is easy to replicate.

Possible Enhancements:

1. **Alternative Models:** Incorporating another model (e.g., a neural network) could showcase the versatility of cross entropy in various settings.
2. **Analysis of Misclassifications:** Add a breakdown of where the model performed poorly and why (e.g., confusion matrix analysis).
3. **Feature Exploration:** Include visualizations or explanations of feature importance to show how the model makes decisions.
4. **Comparative Losses:** Compare cross entropy loss with other loss functions (e.g., mean squared error) to highlight its advantages in classification.

Distribution Insights:

- The histogram of loss values shows how well the model performs across the test dataset.
 - A **narrow distribution** around a low value suggests consistent, accurate predictions.
 - A **wide or skewed distribution** indicates variability in the model's performance, with some instances being predicted poorly.

Mean Squared Error versus Cross Entropy

- **When Comparison Makes Sense:**
 - MSE can highlight how "far off" the predicted probabilities are in terms of magnitude but doesn't account for the probabilistic nature of classification tasks.
 - Comparing cross entropy with MSE can show:
 - How the model performs when considering confidence (cross entropy).

- How the model performs when focusing on numerical proximity (MSE).

- **Insights Gained:**

- If cross entropy is low but MSE is high, it might indicate that the model predicts probabilities close to the correct class but has poor numerical calibration for other classes.
- If both are high, the model is likely underperforming across the board.

Cross_Entropy_Single.py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Selection/Model_Evaluation/Classification/Cross_Entropy_Single.py

Example

Let's consider a three-class classification problem with classes A, B, and C. Suppose we have a single data point with the true class label being A. The true label in one-hot encoded form would be [1, 0, 0].

Assume the model predicts the following probabilities for this data point:

- Probability of class A: 0.7
- Probability of class B: 0.2
- Probability of class C: 0.1

The predicted probability vector is [0.7, 0.2, 0.1].

To calculate the cross entropy loss for this example, we use the formula:

$$L = -\sum_{i=1}^N y_i \log(p_i)$$

Substituting the values:

- For class A: $y_1 = 1$ and $p_1 = 0.7$
- For class B: $y_2 = 0$ and $p_2 = 0.2$
- For class C: $y_3 = 0$ and $p_3 = 0.1$

The cross entropy loss L is calculated as:

$$L = -(1 \cdot \log(0.7) + 0 \cdot \log(0.2) + 0 \cdot \log(0.1))$$

$$L = -(\log(0.7))$$

$$L \approx -0.3567 = 0.3567$$

So, the cross entropy loss for this example is approximately 0.3567. This value represents the penalty for the model's predicted probabilities not perfectly matching the true class distribution. The lower the loss, the better the model's predictions align with the true labels.

Script Description:

1. **Cross Entropy Function:** Computes the cross entropy loss given true labels and predicted probabilities.
2. **True and Predicted Probabilities Visualization:** Bar plots display the true one-hot encoded labels and the predicted probability distribution.
3. **Cross Entropy Loss Calculation:** Prints the loss value for a sample data point.
4. **Loss Curve:** A line graph shows how the loss changes as the predicted probability for the true class increases.

Crosstab

Used to compute a simple cross-tabulation of two (or more) factors. It is particularly useful for computing frequency tables. Here's an example:

```
# Sample DataFrame
df = pd.DataFrame({
    'Category': ['A', 'B', 'A', 'B', 'A'],
    'Subcategory': ['X', 'X', 'Y', 'Y', 'X']
})

# Cross-tabulation of 'Category' and 'Subcategory'
crosstab = pd.crosstab(df['Category'], df['Subcategory'])

print(crosstab)
```

Input

	Category	Subcategory
0	A	X
1	B	X
2	A	Y
3	B	Y
4	A	X

Output:

Category	Subcategory	X	Y
A	X	2	1
B	X	1	1

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/reshaping.ipynb

Cryptography

Cryptography is the foundation of digital [Security](#), enabling privacy and secure communication over the internet.

Examples are implemented in [Node.JS](#) (using `crypto` module) and are written in [JavaScript](#).

Resources:

- [7 Cryptography Concepts EVERY Developer Should Know](#)
- <https://fireship.io/lessons/node-crypto-examples/>

Hash (Chop and mix)

A hashing function takes an input of any length and outputs a fixed-length value, ensuring:

- The same input always produces the same output.
 - It is computationally expensive to reverse the hash.
 - It has a low probability of collisions.
-

Create a Hash in Node.js

```
const { createHash } = require('crypto');

function hash(str) {
    return createHash('sha256').update(str).digest('hex');
}

let password = 'hi-mom!';
const hash1 = hash(password);
console.log(hash1);

password = 'hi-mom';
const hash2 = hash(password);
console.log(hash1 === hash2 ? '✓ Good password' : '✗ Password does not match');
```

Salting

Salting strengthens hashes by appending a random string before hashing, preventing attacks using precomputed hash tables.

Password Salt with Scrypt in Node.js

```
const { scryptSync, randomBytes, timingSafeEqual } = require('crypto');

function signup(email, password) {
  const salt = randomBytes(16).toString('hex');
  const hashedPassword = scryptSync(password, salt, 64).toString('hex');
  users.push({ email, password: `${salt}:${hashedPassword}` });
}

function login(email, password) {
  const user = users.find(v => v.email === email);
  if (!user) return 'login fail!';

  const [salt, key] = user.password.split(':');
  const hashedBuffer = scryptSync(password, salt, 64);
  const match = timingSafeEqual(hashedBuffer, Buffer.from(key, 'hex'));
  return match ? 'login success!' : 'login fail!';
}

const users = [];
signup('foo@bar.com', 'pa$$word');
console.log(login('foo@bar.com', 'password'));
```

HMAC (Hash-based Message Authentication Code)

HMAC combines a hash with a secret key, ensuring authenticity and integrity.

HMAC in Node.js

```
const { createHmac } = require('crypto');

const password = 'super-secret!';
const message = '👋 hello jack';

const hmac = createHmac('sha256', password).update(message).digest('hex');
console.log(hmac);
```

Symmetric Encryption

Symmetric encryption uses the same key to encrypt and decrypt data.

Symmetric Encryption in Node.js

```
const { createCipheriv, randomBytes, createDecipheriv } = require('crypto');

const message = 'i like turtles';
const key = randomBytes(32);
const iv = randomBytes(16);
const cipher = createCipheriv('aes256', key, iv);
const encryptedMessage = cipher.update(message, 'utf8', 'hex') + cipher.final('hex');

const decipher = createDecipheriv('aes256', key, iv);
const decryptedMessage = decipher.update(encryptedMessage, 'hex', 'utf-8') + decipher.final('utf8');

console.log(`Decrypted: ${decryptedMessage}`);
```

Keypairs

Keypairs consist of a public key (shared) and a private key (kept secret) for secure communication.

Generate an RSA Keypair in Node.js

```
const { generateKeyPairSync } = require('crypto');

const { privateKey, publicKey } = generateKeyPairSync('rsa', {
  modulusLength: 2048,
  publicKeyEncoding: { type: 'spki', format: 'pem' },
  privateKeyEncoding: { type: 'pkcs8', format: 'pem' },
});

console.log(publicKey);
console.log(privateKey);
```

Asymmetric Encryption

Asymmetric encryption encrypts with a public key and decrypts with a private key, securing communication over networks.

RSA Encryption in Node.js

```
const { publicEncrypt, privateDecrypt } = require('crypto');
const { publicKey, privateKey } = require('../keypair');

const secretMessage = 'Confidential message';
const encryptedData = publicEncrypt(publicKey, Buffer.from(secretMessage));
console.log(encryptedData.toString('hex'));

const decryptedData = privateDecrypt(privateKey, encryptedData);
console.log(decryptedData.toString('utf-8'));
```

Signing

Sigining verifies the authenticity of a message by hashing it and encrypting the hash with a private key.

RSA Signing in Node.js

```
const { createSign, createVerify } = require('crypto');
const { publicKey, privateKey } = require('../keypair');

const data = 'this data must be signed';
const signer = createSign('rsa-sha256');
signer.update(data);
const signature = signer.sign(privateKey, 'hex');
console.log(signature);

const verifier = createVerify('rsa-sha256');
verifier.update(data);
const isVerified = verifier.verify(publicKey, signature, 'hex');
console.log(isVerified);
```

Current Challenges Within The Energy Sector

Current challenges within the energy sector related to reinforcement learning and that can be progressed with recent technological advances

Cleaning Terminal Path

<https://www.youtube.com/watch?v=18hUejOK0qk>

```
prompt $g
```

powershell

```
$profile

microsoft_Powershell_profile have

function prompt{
$p = -path
"$p> "
}
```

getting the script working

<https://stackoverflow.com/questions/41117421/ps1-cannot-be-loaded-because-running-scripts-is-disabled-on-this-system>

Conceptual Data Model

D

Table of Contents

- DBScan
- DS & ML Portal
- Dash
- Dashboarding
- Data AI Education at Work
- Data Analysis Portal
- Data Analysis
- Data Analyst
- Data Architect
- Data Archive Graph Analysis
- Data Cleansing
- Data Collection
- Data Contract
- Data Distribution
- Data Drift
- Data Engineer
- Data Engineering Portal
- Data Engineering Tools
- Data Engineering
- Data Governance
- Data Hierarchy of Needs
- Data Ingestion
- Data Integration
- Data Integrity
- Data Lake
- Data Lakehouse
- Data Leakage
- Data Lifecycle Management
- Data Management
- Data Modelling
- Data Observability
- Data Pipeline to Data Products
- Data Pipeline
- Data Principles
- Data Product
- Data Quality
- Data Reduction
- Data Roles
- Data Science
- Data Scientist
- Data Selection in ML
- Data Selection
- Data Steward
- Data Storage
- Data Streaming

Introduction

- Data Terms
- Data Transformation with Pandas
- Data Transformation
- Data Validation
- Data Virtualization
- Data Visualisation
- Data Warehouse
- Data transformation in Data Engineering
- Data transformation in Machine Learning
- Database Index
- Database Management System (DBMS)
- Database Schema
- Database Storage
- Database Techniques
- Database
- Databricks vs Snowflake
- Databricks
- Datasets
- Debugging ipynb
- Debugging
- Debugging.py
- Decision Tree
- Deep Learning Frameworks
- Deep Learning
- Deep Q-Learning
- DeepSeek
- Deleting rows or filling them with the mean is not always best
- Demand forecasting
- Dendograms
- Design Thinking Questions
- Determining Threshold Values
- DevOps
- Difference between Databricks vs. Snowflake
- Difference between snowflake to hadoop
- Differentiation
- Digital Transformation
- Digital twin
- Dimension Table
- Dimensional Modelling
- Dimensionality Reduction
- Dimensions
- Directed Acyclic Graph (DAG)
- Directory Structure
- Distillation
- Distributed Computing
- Distribution_Analysis.py
- Distributions
- Docker Image
- Docker
- Documentation & Meetings

- Dropout
- DuckDB in python
- DuckDB vs SQLite
- DuckDB
- Dummy variable trap
- dagster
- data asset
- data lineage
- data literacy
- dbt
- declarative
- dependency manager

Dbscan

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a [Clustering](#) algorithm that groups together data points **based on density**. It is particularly useful when K-means doesn't work well, such as in datasets with complex shapes or when there are outliers.

- **Used when K-means doesn't work:** DBSCAN handles datasets with **irregular cluster shapes** and is not sensitive to outliers like K-means.
- **When you have nesting of clusters:** It can identify clusters of varying shapes and sizes without needing to predefine the number of clusters, unlike K-means.
- **Groups core points to make clusters:** DBSCAN identifies core points, which have many nearby points, and groups them together.
- **Can identify standardised/Outliers:** It detects noise points (outliers) that don't belong to any cluster.

Python Example:

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Create sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
clusters = dbscan.fit_predict(X)

# Plot results
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='plasma')
plt.show()
```

This will cluster the data and visualize it, highlighting core points and marking outliers as separate clusters.

Sources

1. hex.tech - When and Why To Choose Density-Based Methods
2. newhorizons.com - DBSCAN vs. K-Means: A Guide in Python

Machine Learning Fundamentals

- ML_Tools
- Supervised Learning
- Unsupervised Learning
- Reinforcement learning
- Deep Learning

Model Training and Optimisation

- Learning rate
- Overfitting
- Regularisation
- Hyperparameter
- Hyperparameter Tuning
- Model Optimisation
- Model Selection
- Vanishing and exploding gradients problem

Feature Engineering and Data Handling

- Feature Selection
- Feature Engineering
- Imbalanced Datasets
- Outliers
- Anomaly Detection
- Multicollinearity
- Dimensionality Reduction
- Clustering

Machine Learning Models

Classification Models

- Classification
- Binary Classification
- Support Vector Machines
- Decision Tree
- Random Forests
- K-nearest neighbours
- Logistic Regression

Regression Models

- Regression
- Linear Regression

Boosting and Optimisation

Introduction

- Gradient Descent
 - Gradient Boosting
 - XGBoost
-

Deep Learning and Neural Networks

- BERT
- LSTM
- Recurrent Neural Networks
- Transformer
- Attention mechanism
- Neural network

Model Evaluation and Metrics

- Cost Function
- Loss function
- Cross Entropy
- Evaluation Metrics
- Model Evaluation
- Accuracy
- Precision
- Recall

Algorithms and Frameworks

- Machine Learning Algorithms
- Optimisation techniques
- Optimisation function
- Model Ensemble
- Batch Processing
- Apache Spark
- Sklearn

Statistical and Data Analysis Concepts

- Distributions
- Statistics
- Correlation
- Data Analysis
- Data Quality
- Principal Component Analysis

Misc

- Interpretability
 - RAG
-

Dash

Dash is an open-source framework for building interactive web applications using Python.

It is particularly well-suited for data visualization and dashboard creation.

Dash integrates with popular libraries such as Plotly, Pandas, and NumPy, making it ideal for creating dynamic and interactive visualizations.

In [ML_Tools](#) see [Clustering_Dashboard.py](#)

Key Components of Dash

1. **Dash App:** The main application instance, created using `dash.Dash(__name__)`.
2. **Dash HTML Components** (`dash_html_components`): Provides wrappers for standard HTML elements (e.g.,
`html.Div`, `html.H1`).
3. **Dash Core Components** (`dash_core_components`): Includes interactive UI components like graphs, dropdowns, sliders, and more (e.g., `dcc.Graph`, `dcc.Dropdown`).
4. **Callback Functions:** Used to make components interactive by linking inputs (user actions) to outputs (changes in the UI).
5. **Plotly Integration:** Dash apps leverage Plotly for creating interactive visualizations.

Dashboarding

[Dash Streamlit.io](#)

Data Ai Education At Work

Introduction

Organizations are increasingly recognizing the importance of integrating data and AI learning into their people strategies. This involves practical steps to ensure employees are equipped with the necessary skills to leverage these technologies effectively.

Integrating data and AI education into organizational strategies is essential for maintaining competitiveness and fostering a culture of continuous learning. By addressing these areas, organizations can better prepare their workforce for the evolving technological landscape.

Practical Steps for Integration

1. Access to Training:

- o Provide clear guidance on how to access training courses.
- o Offer details on accessing training funds and budgets.
- o Partner with training providers to offer relevant courses.

2. Learning Resources:

- o Collect and distribute clear and concise training materials.
- o Capture, document, and discuss use cases from staff experiences.
- o Encourage peer-to-peer learning and collaboration.

3. Organizational Support:

- o Align training with professional competencies.

Introduction

- Foster communication and collaboration with external partners.
- Encourage staff to experiment with AI tools to enhance efficiency.

4. Governance and Strategy:

- Establish governance and skill strategies before deploying AI.
- Develop acceptable use policies for AI tools.
- Recognize that adopting AI is a gradual process requiring leadership support.

Fostering a Culture of Continuous Learning

1. Leadership and Culture:

- Connect learning initiatives with employee incentives and pay.
- Allow time and space for learning by alleviating workloads.
- Protect training time and build networks to showcase use cases.

2. Mindset and Adaptability:

- Promote digital literacy and adaptability among employees.
- Encourage openness to new ideas and recognize skills beyond the technical team.

Business Risks of Not Upskilling

1. Competitive Disadvantage:

- Risk of losing competitive and productivity edges.
- Potential loss of market differentiation as competitors advance.

2. Staff Retention:

- Risk of losing skilled staff uncomfortable with new technologies.
- Employees may move to companies at the cutting edge of AI.

3. Operational Challenges:

- Inconsistencies in ways of working between partner organizations.
- Inappropriate use of AI tools by untrained staff.

4. Productivity and Trust:

- AI's potential to enhance productivity is yet to be fully realized.
- Trust and verifiability of AI systems (black boxes) are crucial for business benefits.

Data Analysis Portal

[Data Analyst](#)

[Data Visualisation](#)

Data Analysis

What is it? Usually done with a [Data Analyst](#). After processing, data is analyzed to extract meaningful insights and derive value from the data.

Types of analysis:

Exploration and understanding:

Introduction

- **EDA:** Involves exploring data sets to find patterns, anomalies, or relationships without having a specific hypothesis in mind. It is often used in the initial stages of data analysis to generate insights.
- **Descriptive:** Focuses on summarizing historical data to understand what has happened in the past. It often involves the use of **Statistics** measures and **Data Visualisation** tools to present data trends and patterns.
- **Diagnostic:** Seeks to understand why something happened. It involves examining data to identify causes and correlations, often using techniques like data mining and statistical analysis.

Forward looking:

- **Predictive:** Uses historical data and statistical algorithms to forecast future outcomes. It helps in identifying trends and making predictions about future events based on past data.
- **Prescriptive:** Goes a step further by recommending actions based on the predictions made. It uses optimization and simulation algorithms to suggest the best course of action for a given situation.
- **Inferential:** Makes inferences and predictions about a population based on a sample of data. It often involves **Hypothesis testing** and **Confidence Interval**.

Data Analyst

Summary:

- Gathers and processes data to generate reports.
- Communicates insights and findings to management
- Conducts **Data Analysis**.

Key responsibilities of a data analyst:

- Define Objectives: Clearly outline the goals of the analysis to guide the process.
- **Data Collection:** Gather relevant data from various sources, ensuring accuracy, completeness, and timeliness.
- **Data Cleansing:** Clean the data to remove errors, duplicates, and inconsistencies for reliable findings.
- Data Exploration: Perform exploratory data analysis (**EDA**) to understand data structure, **Distributions|distribution**, and relationships.
- Choose the Right Tools: Utilize appropriate tools and software for analysis, such as Excel, R, Python, SQL, or specialized platforms.
- **Statistics:** Apply various statistical methods and techniques, such as regression analysis, clustering, and **hypothesis testing**.
- **Data Visualisation:** Use visualization techniques to effectively present findings and communicate insights.
- Interpret Results: Analyze results in the context of objectives and consider implications for decision-making.
- Documentation: Maintain thorough **Documentation & Meetings** of the analysis process, including data sources, methodologies, and findings.
- Continuous Learning: Stay updated with the latest tools, techniques, and best practices in the evolving field of data analysis.

Data Architect

Data Architect

Introduction

- Designs and manages the data infrastructure.
 - Ensures data is stored, organized, and accessible for analysis.
-

Data Archive Graph Analysis

Use the following to

[Dataview Graph View](#)

Check out [Graph Analysis Plugin](#)

Convert Dataview to CSV

Data Cleansing

Data cleansing is the process of correcting or removing inaccurate, incomplete, or inconsistent data to improve its [Data Quality](#) for analysis. Involves:

- standardised/Outliers|Handling Outliers
- Handling Missing Data
- Handling Different Distributions

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Cleaning/Dataframe_Cleaning.ipynb
- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Cleaning

Related terms:

- [Data Selection](#)

Follow-up questions:

- [Deleting rows or filling them with the mean is not always best](#)

Data Collection

Determine the [Data Quality](#) and quantity of data required and get it.

[Imbalanced Datasets](#)

Data Contract

pattern to handle schema changes

Pattern to apply to organisation using tools they have.

Tooling:

- [dbt](#)

Data contracts help prevent preventable data issues while increasing collaboration and reducing costs.

Introduction

A data contract is an agreed interface between the generators of data and its consumers. It sets the expectations around that data, defines how it should be governed, and facilitates the explicit generation of quality data that meets the business requirements.

Interfaces:

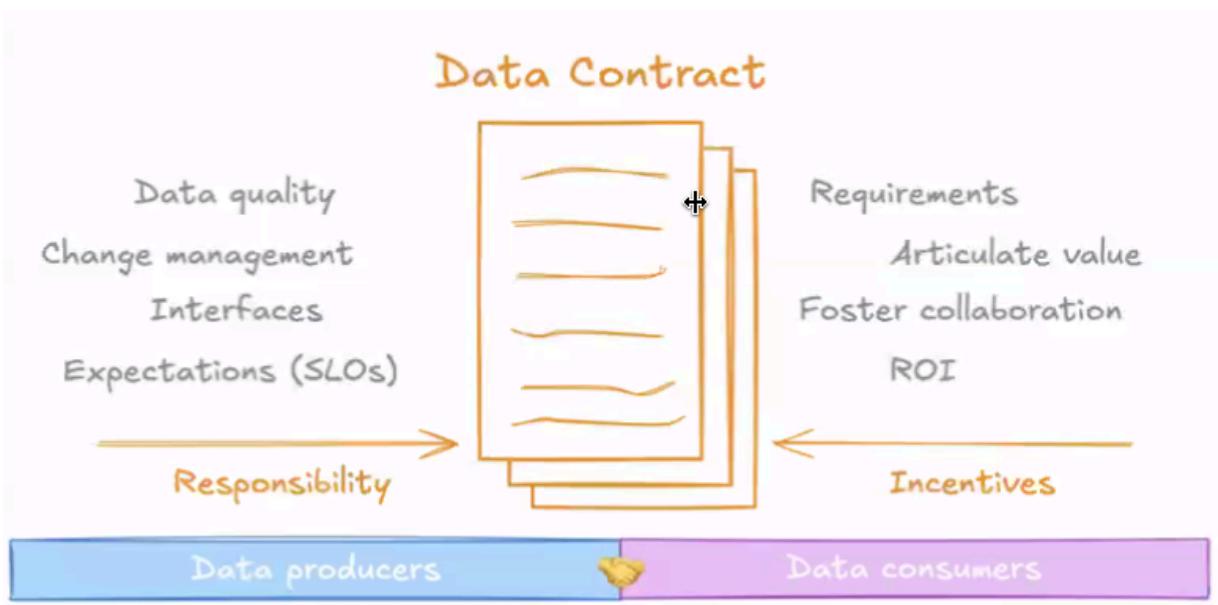
- API for data.

A document to codify what has been agreed.

Q: How does the Data Contract allow for contextual rules? Example the same schema can support multiple products in our org but the DQ rules can be different for different Products A: Data contracts are particular business. Could use **inheritance** of rules in data contracts - basic template. To get standardisation across products.

Data Contract By establishing a **data contract** and building interfaces based on it, organizations can improve data quality. Implementing structured agreements and automated change management processes can help business users, who may not be data experts, produce higher-quality data (**Data Quality**).

Images



Data Distribution

Data distribution refers to the process of making processed and analyzed data available for downstream applications and systems.

This can involve supplying data to

- business applications,
- reporting systems,
- or other data-driven processes,
- ensuring that stakeholders

have access to the information they need for decision-making and operations.

Data Drift

Data drift refers to changes in the statistical properties of input data that a machine learning (ML) model encounters during production. Such shifts can lead to decreased model performance, as the model may struggle to make accurate predictions on data that differ from its training set.

Regular monitoring and prompt response to data drift are essential to maintain the effectiveness of ML models in dynamic production environments.

Concepts:

- Data drift involves changes in input data distributions
- Concept drift pertains to alterations in the relationship between inputs and outputs.
- [Performance Drift](#) drift relates to changes in model outputs.

Training-Serving Skew: This refers to discrepancies between training data and production data, which can arise from data drift or other factors, leading to performance issues.

Detecting Data Drift:

Identifying data drift is crucial for maintaining model accuracy. Techniques include:

- **Statistical Hypothesis Testing:** Assessing whether differences between training and production data distributions are statistically significant.
- **Distance Metrics:** Quantifying the divergence between data distributions using measures like Kullback-Leibler divergence or Kolmogorov-Smirnov tests.
- **Monitoring Summary Statistics:** Regularly reviewing key statistical indicators (e.g., mean, variance) of input features to detect anomalies.

Addressing Data Drift:

Once detected, strategies to manage data drift include:

1. **Data Quality Checks:** Ensure that the drift isn't due to data quality issues, such as errors in data collection or processing.
2. **Investigate the Drift:** Analyze the source and nature of the drift to understand its implications.
3. **Model Retraining:** Update the model using recent data to help it adapt to new patterns.
4. **Model Rebuilding:** In cases of significant drift, it may be necessary to redesign the model architecture or feature engineering processes.
5. **Fallback Strategies:** Implement alternative decision-making processes, such as rule-based systems or human judgment, when the model's reliability is compromised.

Data Engineer

The primary responsibility of a data engineer is to take data from its source and make it available for analysis. They focus on

- automating the data collection,
- processing,
- and analysis workflows,
- solving how systems manage and handle the flow of data.

Introduction

Develops data pipelines and ensures data flow between systems.

Resources:

- [Link](#)

Key Responsibilities:

1. Infrastructure Design and Maintenance:

Data engineers design, build, and maintain the necessary infrastructure to collect, process, and store large amounts of data. This infrastructure is crucial for ensuring data is accessible and usable for analysis and reporting.

2. Data Pipeline:

3. Support Role:

Data engineers act as a bridge between **data producers and consumers**, ensuring smooth and reliable data flow. They support business operations through scalable and efficient **Data Management** solutions, contributing indirectly to product delivery and decision-making.

Core Activities:

What engineers do & interact with: see [Data Engineering Portal](#)

Stakeholders they interact with see [Data Roles](#)

Tools they use: [Data Engineering Tools](#)

Tasks They Are Usually Given

- Project Management: Tracking tasks, bugs, and progress through Azure Boards.
- Collaboration: Facilitating teamwork with shared repositories and [continuous integration](#) workflows.
- Continuous Learning: Keeping up-to-date with the latest technologies and updating pipelines due to obsolescence of tech
- [Documentation & Meetings](#) and Security: Creating documentation, implementing security measures, and exploring system upgrades for enhanced efficiency.

Data Engineering Portal

Databases manage large data volumes with scalability, speed, and flexibility. Key systems include:

- [MySQL](#)
- [PostgreSQL](#)

They facilitate efficient [CRUD.md](#) operations and transactional processing ([OLTP.md](#)), structured by a [Database Schema.md](#) that organizes data into tables and relationships.

Key Features

- **Structured Data:** Organized for efficient CRUD operations, allowing reliable access.
- **Relational Databases:** Use SQL to manage data in tables with relationships expressed through foreign keys and joins, minimizing redundancy.

Structure

- Data is organized into tables (like spreadsheets) with columns (fields) and rows (records), enabling efficient storage and retrieval.

Flexibility

- Databases have a flexible schema that adapts to evolving requirements, unlike static solutions like spreadsheets.

Related Ideas:

- [Spreadsheets vs Databases](#)
- [Database Management System \(DBMS\)](#)
- [Components of the database](#)
- [Relating Tables Together](#)
- [Turning a flat file into a database](#)
- [Database Techniques](#)

Data Engineering Tools

- **Snowflake:** Cloud-based data warehousing for scalable storage and processing.
- **Microsoft SQL Server:** SQL-based relational database management.
- **Azure SQL Database:** Managed relational database service on Azure.
- **Azure Data Lake Storage:** Scalable storage for big data analytics.
- **SQL and T-SQL:** Query languages for managing and querying relational databases.
- **AWS Amazon S3|S3:** Storage for data lakes.

Data Ingestion Tools and Technologies:

- [Apache Kafka](#)
- AWS Kinesis: A cloud service for real-time data processing, enabling the collection and analysis of streaming data.
- Google Pub/Sub: A messaging service that allows for asynchronous communication between applications, supporting real-time data ingestion.

Data Storage Tools:

Amazon S3, Google BigQuery, Snowflake.

[dbt](#)

Tags

- **Tags:** #data_tools, #data_management

Data Engineering

The definition from the [Fundamentals of Data Engineering](#), as it's one of the most recent and complete:

Introduction

Data engineering is the development, implementation, and maintenance of systems and processes that take in raw data and produce high-quality, consistent information that supports downstream use cases, such as analysis and machine learning. Data engineering intersects security, [Data Management](#), DataOps, data architecture, orchestration, and software engineering.

A [Data Engineer](#) today oversees the whole data engineering process, from collecting data from various sources to making it available for downstream processes. The role requires familiarity with the multiple stages of the [Data Engineering Lifecycle](#) and an aptitude for evaluating data tools for optimal performance across several dimensions, including price, speed, flexibility, scalability, simplicity, reusability, and interoperability.

Data Engineering helps also overcome the bottlenecks of [Business Intelligence](#):

- More transparency as tools are open-source mostly
- More frequent data loads
- Supporting [Machine Learning](#) capabilities

Compared to existing roles it would be a **software engineering plus business intelligence engineer including big data abilities** as the [Hadoop](#) ecosystem, streaming, and computation at scale. Business creates more reporting artifacts themselves but with more data that needs to be collected, cleaned, and updated near real-time and complexity is expanding every day.

With that said more programmatic skills are needed similar to software engineering. **The emerging language at the moment is Python** which is used in engineering with tools alike [Apache Airflow](#), [dagster](#), [Prefect](#) as well as data science with powerful libraries.

As a data engineer, you use mainly [SQL](#) for almost everything except when using external data from an API. Here you'd use [ELT](#) tools or write some [Data Pipeline](#) with the tools mentioned above.

Data Governance

Data governance is a collection of processes, roles, policies, standards, and metrics that ensure the effective and efficient use of information in enabling an organization to achieve its goals.

It establishes the processes and responsibilities that ensure the [Data Quality](#) and security of the data used across a business or organization. Data governance defines who can take what action, upon what data, in what situations, and using what methods.

Data Governance: Focuses on ensuring that data is managed consistently and adheres to policies, often working in tandem with [Data Observability](#) to enforce quality standards.

Data Hierarchy Of Needs



The **Data Hierarchy of Needs** is a framework that outlines the stages required to effectively use data in organizations. It resembles Maslow's hierarchy, progressing from basic data needs to advanced capabilities:

1. **Data Collection:** (bottom) Start by collecting raw data from various sources, ensuring it's stored securely and reliably.
2. **Data Storage and Access:**

Organize and store data so it's easily accessible for those who need it, using databases or data warehouses.

3. Data Cleaning and Preparation:

Clean, preprocess, and transform data to ensure it's accurate, consistent, and ready for analysis.

4. Data Analytics:

Analyze the prepared data to generate insights, identify patterns, and create reports.

5. Data-Driven Decision Making:

Use the insights from data analytics to inform and improve decision-making across the organization.

6. Advanced Data Capabilities (AI/ML):

(top) Once the foundation is in place, apply advanced techniques like machine learning and artificial intelligence for predictive and prescriptive insights.

Data Ingestion

Data ingestion is the process of collecting and importing raw data from various sources ([Database](#), [API](#), [Data Streaming](#) services) into a system for processing and analysis, and can be performed in batch and realtime ingestion. The goal is to gather raw data that can be processed and analyzed.

Used for building [Data Pipeline](#)

Challenges

- [Data Quality](#): Ensuring that the ingested data is accurate, complete, and consistent.
- [Scalability](#): Handling large volumes of data efficiently as the data sources grow.
- [Latency](#): Minimizing the delay between data generation and processing, especially in real-time scenarios.

Use Cases:

- Data ingestion is used in various applications, including: [business intelligence](#), [Machine Learning](#)

Related to:

- [Data Engineering Tools](#)

[Data Ingestion Tags](#): #data_collection, #data_management

Data Integration

Data integration is the process of combining data from disparate source systems into a single unified view, moving data to a [Single Source of Truth](#).

Manual Integration

Manual integration involves analysts manually logging into source systems, analyzing and/or exporting data, and creating reports.

Disadvantages of Manual Integration:

- **Time-consuming:** The process requires significant time investment.
- **Security Risks:** Analysts need access to multiple operational systems.
- **Performance Issues:** Running analytics on non-optimized systems can interfere with their functioning.
- **Outdated Reports:** Data changes frequently, leading to quickly outdated reports.

Data Virtualization

Data virtualization is a method that allows access to data without needing to replicate it, providing a unified view of data from multiple sources.

Application Integration

Application integration links multiple applications to move data directly between them.

Methods of Application Integration:

- **Point-to-Point Communications:** Direct connections between applications.
- **Middleware Layer:** Using tools like an Enterprise Service Bus (ESB).
- **Application Integration Tools:** Specialized tools for integrating applications.

Disadvantages of Application Integration:

- **Data Redundancy:** May result in multiple copies of the same data across systems.
- **Increased Costs:** Managing multiple copies can lead to higher costs.
- **Point-to-Point Traffic:** Can create excessive traffic between systems.
- **Performance Impact:** Executing analytics on operational systems may interfere with their functioning.

Data Integrity

Data integrity refers to the

- accuracy,
- consistency, and
- reliability of data

throughout its lifecycle. It ensures that data remains **unaltered** and **trustworthy**, whether it is being

- stored,
- processed,
- or transmitted.

Maintaining data integrity involves implementing measures to prevent unauthorized access, corruption, or loss of data.

In the context of [Database](#) and information systems, data integrity can be enforced through:

1. **Validation Rules:** Ensuring that data entered into a system meets certain criteria.
2. **Access Controls:** Limiting who can view or modify data.
3. **Backups:** Regularly saving copies of data to prevent loss.
4. **Error Checking:** Using [Checksum](#) or [Hash](#) to verify data integrity during transmission.

[Data Integrity Tags:](#) #data_quality, #data_management

Data Lake

A Data Lake is a storage system with vast amounts of [unstructured data](#) and [structured data](#), stored as-is, without a specific purpose in mind, that can be built on multiple technologies such as Hadoop, NoSQL, Amazon Simple Storage Service, a relational database, or various combinations and different formats (e.g. Excel, CSV, Text, Logs, etc.).

Definition: A repository that [stores diverse data types](#), including structured, semi-structured, and unstructured data. If it can't fit into a database.

Features:

- **Versatility:** Can accommodate various data formats, including videos, images, documents, and more.
- **Raw Data Storage:** Preserves data in its raw form, suitable for advanced analytics, particularly in machine learning and AI.
- **Data Usability:** Raw data [may require cleaning and transformation for analytical use](#), often transferred to databases or data warehouses.
- **Use Case:** Valuable for storing large volumes of raw data, especially in contexts requiring advanced analytics and experimentation.

[unstructured data](#) for predictive modeling and analysis. This leads to the creation of a **data lake**, which stores raw data without predefined schemas.

The data lake supports the following capabilities:

- To capture and store raw data at scale for a low cost
- To store many types of data in the same repository
- To perform [Data Transformation](#) on the data where the purpose may not be defined
- To perform new types of data processing
- To perform single-subject analytics based on particular use cases

Components of a data lake

- ```
1. [Storage Layer](term/storage%20layer%20object%20store.md)
2. [Data Lake File Format](term/data%20lake%20file%20format.md)
3. [Data Lake Table Format](term/data%20lake%20table%20format.md) with [Apache Parquet](term/apache%20parquet.md), [Ap
```



## Data Lakehouse

A Data Lakehouse open [Data Management](#) architecture that combines the flexibility, cost-efficiency, and scale of [Data Lake](#) with the data management and ACID transactions of [Data Warehouse](#) with Data Lake Table Formats ([Delta Lake](#), [Apache Iceberg](#) & [Apache Hudi](#)) that enable Business Intelligence (BI) and Machine Learning (ML) on

all data.

A **data lakehouse** is an emerging architectural approach that combines the best features of data lakes and data warehouses to provide a unified platform for storing, processing, and analyzing large volumes of structured and unstructured data. Here's a breakdown of its key characteristics and benefits:

The data lakehouse architecture represents a significant evolution in [Data Management](#), addressing the limitations of traditional data lakes and [Data Warehouse|Warehouse](#) by providing a unified platform for all data types.

## Key Characteristics

### 1. Unified Storage:

- Data lakehouses store data in a single repository, accommodating both structured data (like tables in a database) and unstructured data (like images, videos, and text). This eliminates the need for separate systems, simplifying data management.

### 2. Support for Multiple Data Types:

- They can handle various data formats, such as **CSV**, **JSON**, **Parquet**, and **Avro**, enabling flexibility in how data is ingested and stored.

### 3. ACID Transaction:

- Unlike traditional data lakes, data lakehouses provide [ACID Transaction](#) which ensure reliable data operations and integrity, even in concurrent processing environments.

### 4. Schema Enforcement:

- Data lakehouses can enforce [Database Schema|schema](#) at the time of data write, allowing users to define data structures while still benefiting from the flexibility of a data lake.

### 5. Performance Optimization:

- They incorporate various optimization techniques, such as indexing and caching, to improve query performance and provide faster access to data.

### 6. Integration with BI Tools:

- Data lakehouses are designed to work seamlessly with business intelligence (BI) tools and data analytics platforms, enabling users to derive insights without needing extensive data preparation.

## Benefits

### 1. Cost-Effectiveness:

- By merging the functionalities of data lakes and data warehouses, organizations can reduce the costs associated with maintaining separate systems for structured and unstructured data.

### 2. Scalability:

- Data lakehouses leverage cloud storage solutions, allowing for scalable data storage that can grow with the organization's needs.

### 3. Data Accessibility:

- With a unified architecture, data from different sources can be accessed and analyzed together, breaking down silos and fostering a more holistic view of the organization's data landscape.

### 4. Simplified Data Pipelines:

- Data lakehouses streamline the data ingestion process, enabling organizations to build more efficient data pipelines that accommodate a variety of data sources.

### 5. Support for Advanced Analytics:

## Introduction

- They provide a robust foundation for advanced analytics, including machine learning and real-time data processing, allowing organizations to extract actionable insights more effectively.
- 

Platforms that implement the data lakehouse architecture include:

- **Databricks Lakehouse Platform:** Combines data engineering, data science, and BI capabilities with a focus on collaboration.
- **Apache Iceberg:** A high-performance table format for large analytic datasets that supports ACID transactions and schema evolution.

## Data Leakage

**Data Leakage** refers to the unintentional inclusion of information in the training data that would not be available in a real-world scenario, leading to overly optimistic model performance. It occurs when the model has access to data it shouldn't during training, such as future information or test data, which can result in misleading evaluation metrics and poor generalization to new data.

## Data Lifecycle Management

This is the comprehensive process of managing data from its initial ingestion to its final use in downstream processes.

Used for maintaining [data integrity](#), optimizing performance, and ensuring that data-driven decisions are based on accurate and timely information.

Not the same as the [Software Development Life Cycle](#)

Key Stages of Full Lifecycle Management

1. Data Ingestion
2. Data Storage
3. Preprocessing
4. Data Analysis
5. Data Visualisation
6. Data Distribution

Data engineers must evaluate and select tools and technologies based on several [Performance Dimensions](#)

## Data Management

Data management involves overseeing processes to maintain data integrity and quality. It includes:

- **Responsibility:** Identifying accountable individuals or teams.
- **Issue Resolution:** Mechanisms for detecting and addressing data-related problems.

Data management ensures that a [Data Pipeline](#) operates efficiently, focusing on monitoring errors, performance issues, and [data quality](#).

**Tools:**

- [Apache Airflow](#)
  - [Prefect](#)
  - [Dagster](#)
-

Related Concepts:

- Database Management System (DBMS)
- Master Data Management
- Data Distribution

Data Management Tags: #data\_management, #data\_quality

## Data Modelling

Data modelling is the process of creating a visual representation of a system's data and the relationships between different data elements.

This helps in organizing and structuring the data so it can be efficiently managed and utilized.

Data modelling ensures that data is logically structured and organized, making it easier to store, retrieve, and manipulate in a database.

Workflow of Data Modeling: 1) [Conceptual Model](#) 2) [Logical Model](#) 3) [Physical Model](#)

Types of Modeling:

- Relational: Organizes data into tables.
- Object-Oriented: Focuses on objects and their state changes, e.g., robots in a car factory.
- Entity: Uses [ER Diagrams](#) to represent data entities and relationships.
- Network: An extension of hierarchical models.
- Hierarchical: Organizes data in a tree-like structure.

Data Modelling Tags: #data\_modeling, #database\_design

## Data Observability

Data observability refers to the continuous monitoring and collection of metrics about your data to ensure its [Data Quality](#), reliability, and availability.

It covers various aspects, such as data quality, pipeline health, metadata management, and infrastructure performance. By tracking key metrics and [standardised/Outliers|anomalies](#), it helps detect issues like data freshness problems, schema changes, or pipeline failures before they impact downstream processes or users.

## Categories of Observability

Auto-profiling Data:

Automatically tracks data attributes, such as row count, column types, data distributions, and schema changes.

- Bigeye: Provides ML-driven threshold tests and automatic alerts when data drifts beyond expected ranges.
- Datafold: Integrates with GitHub to run data diffs between environments, offering insights into differences between datasets during development.
- Monte Carlo: Enterprise-focused with data lake integrations for comprehensive observability.
- Metaplane: Offers a high level of configuration and both out-of-the-box and custom tests.

Pipeline Testing:

Ensures that data transformation pipelines are functioning correctly by verifying the quality and accuracy of data as it moves through different stages.

## Introduction

- Great Expectations: An open-source tool that allows you to define tests and automatically generate documentation for those tests, promoting transparency in data quality checks.
- Soda: Offers pipeline testing with the flexibility of a self-hosted option for more control over data quality monitoring.
- `dbt`tests: Integrated with `dbt` Core and `dbt` Cloud, allowing testing during the transformation process in a `dbt` project.

### Infrastructure Monitoring:

Monitors the health and performance of the underlying data infrastructure, such as databases, pipelines, and servers, to prevent failures and bottlenecks.

- DataDog: Provides deep monitoring capabilities, including for Airflow, containers, and custom metrics, allowing visibility at various layers of the data stack.

## Managing Metadata

Managing metadata is critical for observability, as it provides context and lineage for your data. Metadata can include:

- Technical Metadata: Information about the dataset's structure, such as table schema, data types, and column descriptions.
- Operational Metadata: Information about the dataset's freshness, when it was last updated, and the number of records processed.
- Business Metadata: Describes the meaning of data, such as field definitions and business rules, helping stakeholders understand the context and usage of the dataset.

### How to Manage Metadata:

- Manual Documentation: Teams may manually document metadata, but this can be prone to human error and inconsistency.
- Automated Metadata Management: Many modern data tools, such as data catalogs (e.g., Atlan, Alation), automatically track and manage metadata, offering insights into data lineage, schema changes, and data usage.
- Integration with Data Pipelines: Tools like `dbt` also generate metadata about transformations, which can be included in downstream monitoring systems to ensure consistency and traceability.

### Data Observability

- Tracking the issues.
- Alerting and ensuring data owners fix it.

## Data Pipeline To Data Products

The journey from [Data Pipeline](#) to [Data Product](#) involves transforming raw data into valuable insights or applications that can be used to drive business decisions. This process typically includes several stages, each with its own set of tasks and objectives.

Read more on [Data Orchestration Trends: The Shift From Data Pipelines to Data Products](#).

## Workflow

### 1. Define Objectives:

- Understand the business goals and what insights or products are needed.

**2. Design the Pipeline:**

- Plan the architecture and select appropriate tools for each stage of the pipeline.

**3. Implement and Test:**

- Build the pipeline, ensuring data flows smoothly from ingestion to product delivery.
- Test for accuracy, performance, and reliability.

**4. Deploy and Monitor:**

- Deploy the pipeline in a production environment.
- Continuously monitor for performance and make adjustments as needed.

**5. Iterate and Improve:**

- Gather feedback and refine the pipeline and products to better meet business needs.

## Example

Imagine a retail company wants to create a recommendation system for its online store:

1. **Data Ingestion:** Collect customer browsing and purchase data from the website.
2. **Data Processing:** Clean and transform the data to identify patterns in customer behavior.
3. **Data Storage:** Store the processed data in a data warehouse for easy access.
4. **Data Analysis:** Use machine learning algorithms to analyze the data and generate recommendations.
5. **Data Visualization:** Create dashboards to visualize customer trends and recommendation performance.
6. **Data Products:** Deploy the recommendation system on the website to enhance customer experience.

# Data Pipeline

A data pipeline is a series of processes that automate the movement and transformation of data from various sources to a destination where it can be stored, analyzed, and used to generate insights.

It ensures that data flows smoothly and efficiently through different stages, maintaining data quality and [Data Integrity](#).

By implementing a data pipeline, organizations can automate data workflows, reduce manual effort, and ensure timely and accurate data delivery for decision-making.

## Workflow

1. [Data Ingestion](#)
2. [Data Transformation](#)
3. [Data Storage](#)
4. [Preprocessing|Data Preprocessing](#)
5. [Data Management](#)

## Other steps:

Design:

- Define the objectives and requirements of the data pipeline.
- Choose appropriate tools and technologies.

Development:

- Build the pipeline components and integrate them into a cohesive system.

Testing:

- Validate the pipeline to ensure data accuracy and performance.

Deployment:

- Deploy the pipeline in a production environment.

Monitoring and Maintenance:

- Continuously monitor the pipeline and make necessary adjustments to improve performance and reliability.

## Related Notes

- [Data Pipeline to Data Products](#)

**Data Pipeline Tags:** #data\_workflow, #data\_management

# Data Principles

Data principles are essential for ensuring that data is managed, used, and maintained effectively and ethically.

1. **Data Quality**: Ensure data is accurate, complete, reliable, and up-to-date. High-quality data is crucial for making informed decisions.
2. **Data Governance**: Establish clear policies and procedures for data management, including roles and responsibilities, to ensure [data integrity](#) and compliance with regulations.
3. **Data Privacy**: Protect personal and sensitive information by adhering to privacy laws and regulations, such as GDPR or CCPA, and implementing appropriate security measures.
4. **Data Security**: Safeguard data against unauthorized access, breaches, and other security threats through encryption, access controls, and regular [security](#) audits.
5. **Data Accessibility**: Ensure that data is easily accessible to those who need it while maintaining appropriate security and privacy controls. This includes providing the necessary tools and training for data access.
6. **Data Transparency**: Maintain transparency about data collection, usage, and sharing practices. This helps build trust with stakeholders and ensures accountability.
7. **Data Consistency**: Standardize data formats and definitions across the organization to ensure consistency and interoperability.
8. **Data Stewardship**: Assign data stewards to oversee [data management](#) practices, ensuring data quality, compliance, and proper usage.
9. **Data Lifecycle Management**: Manage data throughout its lifecycle, from creation and storage to archiving and deletion, ensuring that data is retained only as long as necessary.
10. **Ethical Data Use**: Use data ethically and responsibly, considering the potential impact on individuals and society. Avoid biases and ensure fairness in data-driven decisions.
11. **Data Documentation & Meetings**: Maintain thorough documentation of data sources, definitions, and processes to facilitate understanding and reproducibility.
12. **Data Sharing and Collaboration**: Encourage data sharing and collaboration within and across organizations to maximize the value of data, while respecting privacy and security constraints.
13. **DRY**

Related:

- 
- [Performance Dimensions](#)

## Data Product

A data product is

"a product that facilitates an end goal through data".

Delivering the final output, which could be dashboards, reports, or machine learning models. For example Recommendation systems or predictive analytics dashboards.

It applies more product thinking, whereas the "Data Product" essentially is a dashboard, report, and table in a [Data Warehouse](#) or a Machine Learning model.

Sometimes Data Products are also called [data asset](#).

## Data Quality

Data quality is the process of ensuring that data meets established expectations. High-quality data is crucial for effective decision-making and analysis.

**Definition:** Data quality refers to the [accuracy, consistency, and reliability of data](#). It is essential for maintaining trust in data-driven processes and outcomes.

**Importance:** The principle of "garbage in, garbage out" highlights that poor-quality data leads to poor model performance.

Related terms:

- [Data Observability](#)
- [Change Management](#)
- [Prevention Is Better Than The Cure](#)

Related terms:

- [Data Observability](#)
- [Data Contract](#)
- [Change Management](#)

## Data Reduction

Reducing the volume of data through techniques:

[Dimensionality Reduction](#)

[Sampling](#): Use subsets of data for training to speed up the process and address issues like imbalanced data representation.

Remove features with zero or low [variance](#) and redundant features to improve model performance.

# Data Roles

A data team is a specialized group within an organization responsible for managing, analyzing, and leveraging data to drive business decisions and strategies.

The team collaborates across various functions to ensure data integrity, accessibility, and usability.

## Key Roles and Responsibilities

| Role                 | Focus Area                | Key Responsibilities                                                                           |
|----------------------|---------------------------|------------------------------------------------------------------------------------------------|
| Data Steward         | Data quality & governance | Enforces data policies, resolves data quality issues, manages metadata.                        |
| Data Governance Team | Policy & compliance       | Defines data management rules, ensures regulatory adherence.                                   |
| Data Engineer        | Data infrastructure       | Builds data pipelines, integrates data sources, and ensures data flow.                         |
| Data Scientist       | Data analysis & modeling  | Utilizes BI tools, analyzes data, develops and deploys ML models.                              |
| ML Engineer          | Machine learning          | Configures and optimizes ML models, monitors performance in production.                        |
| Data Architect       | Data architecture         | Designs and manages data infrastructure, ensures data accessibility.                           |
| Data Analyst         | Reporting & visualization | Gathers and processes data, generates reports, communicates insights using tools like Tableau. |

## Other Stakeholders

- **Business Analysts:** Ensure data is structured and accessible for analysis and reporting.
- **Senior Stakeholders and Business Ambassadors:** Communicate requirements, progress, and solutions to align with business goals.
- **Software Engineers and Data Teams:** Coordinate on data production and integration processes.

# Data Science

A field that uses the [Scientific Method](#), algorithms, and systems to [extract knowledge](#) and insights from structured and [unstructured data](#). It combines techniques from [statistics](#), computer science, and domain expertise to analyze and interpret complex data sets, enabling informed decision-making and predictive modeling.

Resources:

- [https://scikit-learn.org/stable/auto\\_examples/index.html](https://scikit-learn.org/stable/auto_examples/index.html)

# Data Scientist

---

## Data Scientist

- Utilizes [Business Intelligence](#) (BI) tools to analyze data.
- Works with data lakes to extract insights.
- Develops and deploys production Machine Learning (ML) models for predictions.

## Data Selection In MI

When selecting data for machine learning models, several important considerations can significantly impact the model's performance/[Model Optimisation](#) and the insights you can derive from it. Here are key factors to consider:

### 1. Relevance:

- Ensure that the features (input variables) you select are relevant to the problem you are trying to solve. Irrelevant features can introduce noise and reduce model accuracy.

### 2. Quality: [Data Quality](#)

- Assess the quality of the data, including checking for missing values, outliers, and errors. Poor quality data can lead to inaccurate models.

### 3. Quantity:

- Consider the size of your dataset. More data can lead to better models, but it also requires more computational resources. Ensure you have enough data to train your model effectively.

### 4. Balance: [Imbalanced Datasets](#)

- Check for [Imbalanced Datasets](#)|[class imbalance](#) in classification problems. An imbalanced dataset can bias the model towards the majority class. Techniques like resampling, synthetic data generation, or using different evaluation metrics can help address this.

### 5. Feature Distribution: [Distributions](#)

- Analyze the distribution of your features. Features with skewed [distributions](#) may need transformation ([Data Transformation](#)) (e.g., log transformation) to improve model performance.

### 6. Correlation:

- Examine the correlation between features. Highly correlated features can lead to [multicollinearity](#), which can affect model stability and interpretability. Consider removing or combining correlated features.

### 7. Dimensionality: [Dimensionality Reduction](#)

- High-dimensional data can lead to overfitting. Techniques like [feature selection](#), dimensionality reduction (e.g., PCA), or regularization can help manage this.

### 8. Temporal Considerations:

### 9. For time series data, ensure that the temporal order is maintained. Avoid data leakage by ensuring that future information is not used in training.

### 10. Domain Knowledge:

- Leverage domain expertise to select features that are known to be important for the problem. This can guide feature engineering and selection.

### 11. Data Leakage:

- Be cautious of [Data Leakage](#), where information from the test set is inadvertently used in training. This can lead to overly optimistic performance estimates.

12. Scalability:

13. Consider the scalability of your data selection process. As datasets grow, ensure that your methods can handle larger volumes efficiently.

## Data Selection

Data selection is a crucial part of data manipulation and analysis. Pandas provides several methods to select data from a DataFrame.

In [DE\\_Tools](#) we explore how to do Data Selection with Pandas

- [https://github.com/rhyslwells/DE\\_Tools/blob/main/Explorations/Transformation/selection.ipynb](https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/selection.ipynb)

Related:

- [Data Selection in ML](#)

## Examples

### Selecting Columns

You can select a single column from a DataFrame using either bracket notation or dot notation:

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
column_a = df['A'] # or df.A
```

### Selecting Rows by Index

To select rows by their index position, you can use slicing:

```
rows_0_to_2 = df[0:3] # Selects the first three rows
```

### Selecting Rows by Date Range

If your DataFrame has a DateTime index, you can select rows within a specific date range:

```
date_rng = pd.date_range(start='2013-01-01', end='2013-01-06', freq='D')
df = pd.DataFrame(date_rng, columns=['date'])
df.set_index('date', inplace=True)
selected_dates = df['2013-01-02':'2013-01-04']
```

### Label-based Selection

Use `.loc` or `.at` to select rows by label:

```
df = pd.DataFrame({'Weather': ['Sunny', 'Rain', 'Cloudy'], 'Temp': [30, 22, 25]})
df.set_index('Weather', inplace=True)
rain_row = df.loc['Rain'] # or df.at['Rain']
```

## Position-based Selection

Use `.iloc` or `.iat` to select rows by position:

```
third_row = df.iloc[2] # Selects the third row
specific_value = df.iat[1, 1] # Selects the value at row 1, column 1
```

## Conditional Selection

Create a new DataFrame based on a condition:

```
df_new = df[df['var1'] >= 999] # Selects rows where 'var1' is greater than or equal to 999
```

The condition `df["var1"] >= 999` creates a boolean Series that filters the rows of `df`.

# Data Steward

A **Data Steward** is responsible for ensuring the quality, integrity, and governance of an organization's data assets. They act as a bridge between business users, IT teams, and data governance policies, ensuring that data is well-defined, accurate, and used appropriately.

## Key Responsibilities of a Data Steward

1. **Data Quality Management** – Ensuring data accuracy, completeness, consistency, and reliability across systems.
2. **Metadata Management** – Documenting data definitions, relationships, and lineage.
3. **Data Governance Compliance** – Implementing policies, standards, and best practices for data handling.
4. **Master Data Management (MDM)** – Managing critical business data entities like customers, products, and suppliers.
5. **Collaboration with Stakeholders** – Acting as a liaison between business units, data engineers, and data governance teams.
6. **Issue Resolution** – Identifying and resolving data-related issues such as duplicates, missing values, and inconsistencies.
7. **Data Security & Privacy** – Ensuring compliance with regulations (e.g., GDPR, HIPAA) by monitoring access and usage.

## Why is a Data Steward Important?

- Enhances **data trustworthiness**, leading to better decision-making.
- Reduces **data inconsistencies** and errors in analytics and reporting.
- Supports **regulatory compliance** and risk management.
- Enables **efficient data integration** across systems and departments.

- Responsible for [data governance](#) and quality.
- Ensures that data policies and standards are adhered to across the organization.
- Acts as a liaison between data users and IT to facilitate [data management](#).

## Data Storage

Data storage is a fundamental aspect of [Data Engineering](#), influencing processes such as

- (occurring after [Data Ingestion](#))
- [Data Transformation](#)
- [Querying](#)
- [data management](#).

Storing the [Data Transformation](#) data in a database or [Data Warehouse](#) for easy access and analysis.

## Types of Storage

Data storage encompasses various methods and technologies for storing, retrieving, and managing data. The choice of storage method significantly impacts [data retrieval efficiency](#) and consistency

| Storage Type                 | Description                                                                                              |                                                                                                |
|------------------------------|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| [storage layer object store] | Object Store](#storage-layer-object-storeobject-store)                                                   | The gold standard for data lakes, ideal for unstructured data such as images, audio, and text. |
| Database                     | The most widely deployed database globally is <a href="#">SQLite</a> . Suited for transaction recording. |                                                                                                |
| NoSQL                        |                                                                                                          |                                                                                                |
| Data Warehouse               | Excels in analytics and reporting.                                                                       |                                                                                                |
| Data Lake                    | Offers versatility for storing raw data, particularly beneficial for advanced analytics applications.    |                                                                                                |

## Follow-Up Questions

- How do different data storage methods impact data retrieval speed in large datasets?
- What are the trade-offs between using relational versus [NoSQL](#) databases in specific applications?

## Related Resources

- [Cloud Providers](#)
- [Amazon S3](#)
- [Data Governance](#)
- [Data Engineering Tools](#)

# Data Streaming

---

Data Streaming is used for real-time data processing, allowing continuous flow and processing of data as it arrives. This is different from [batch processing](#), which handles data in chunks.

The key to data streaming is the [Publish and Subscribe](#)

[Apache Kafka](#)

Example:

- Companies like Netflix use Kafka to handle billions of messages daily, powering real-time recommendations, analytics, and user activity tracking.

[Alternatives to Batch Processing](#)

[Data Streaming Tags](#): #data\_workflow

## Data Terms

## Data Transformation With Pandas

Using [pandas](#) we can do the following:

- Merge
- Concatenate
- Joining Datasets
- Pandas join vs merge
- Multi-level index
- Aggregation
- Pandas Stack
- Crosstab

A summary of transformations steps can be helpful:

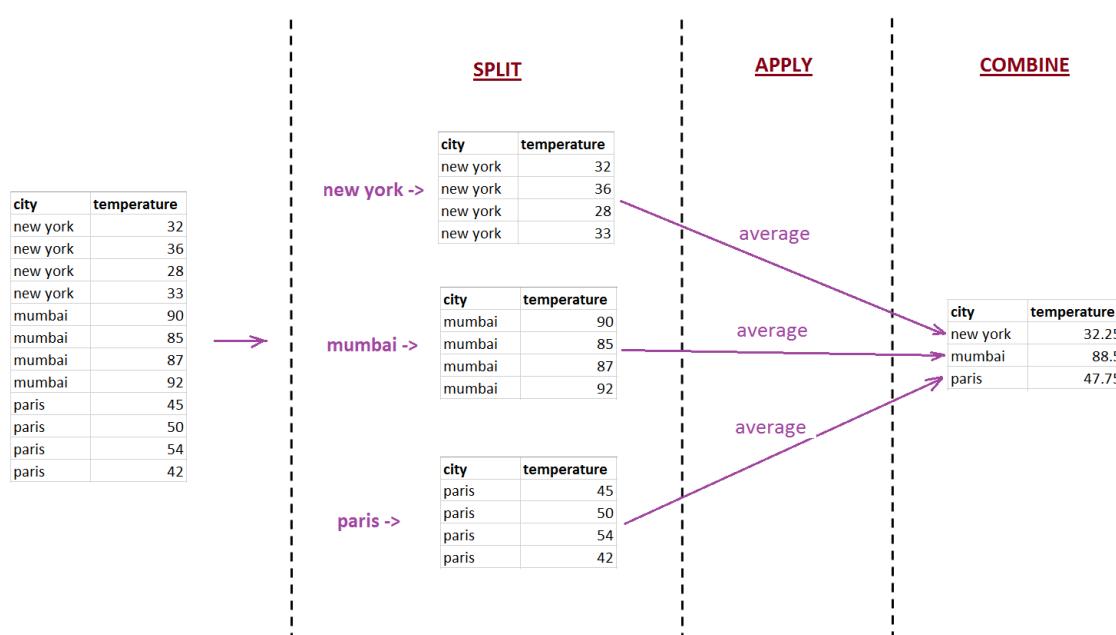
| Step | Operation                    | Result                              |
|------|------------------------------|-------------------------------------|
| 1    | <code>set_index</code>       | Rows get hierarchical keys          |
| 2    | <code>stack</code>           | Wide → long with 3-level row index  |
| 3    | <code>reset + extract</code> | Parse variable names into fields    |
| 4    | <code>pivot</code>           | Tidy format with metric columns     |
| 5    | <code>unstack</code>         | Wide format with MultiIndex columns |

In [DE\\_Tools](#) see:

- [https://github.com/rhyslwells/DE\\_Tools/blob/main/Explorations/Investigating/Transformation](https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Investigating/Transformation)

Related terms:

## Split-Apply-Combine



## Data Transformation

Data transformation is the process of converting data from one format to another.

Data transformation may involve:

- Data Cleansing
- Structuring and organizing data
- Aggregation
- Data Selection
- Joining Datasets
- Normalisation of data
- Normalised Schema

Others:

- Sorting: Arranging data in a logical order.
- Validating: Ensuring data integrity and accuracy.
- Data Type Conversion: Changing data types (e.g., converting strings to integers).
- Schema Normalization: Ensuring a consistent data structure for efficiency.

Related:

- [Data Transformation with Pandas](#)
- [Data transformation in Data Engineering](#)
- [Data transformation in Machine Learning](#)
- [Benefits of Data Transformation](#)

## Data Validation

Data Validation:

## Introduction

- **Error Prevention:** It ensures data accuracy by preventing incorrect or inappropriate data entries.
  - **Consistent Data Entry:** Helps maintain consistency across large datasets by controlling what users can input.
  - **Efficiency:** By providing drop-down lists or constraints, it reduces the chances of manual errors.
  - **Better Data Quality:** Validating input ensures that your data is clean and ready for analysis or reporting without requiring additional checks.
  - [type checking](#)
  - [TypeScript](#)
- 

## Pydantic

# Data Virtualization

Organizations may also consider adopting a data virtualization solution to integrate their data.

In this type of [data integration](#), data from multiple sources is left in place and is [accessed](#) via a virtualization layer so that it [appears](#) as a single data store.

This virtualization layer makes use of adapters that translate queries executed on the virtualization layer into a format that each connected source system can execute.

The virtualization layer then combines the responses from these source systems into a single result. This data integration strategy is sometimes used when a BI tool like Tableau needs to access data from multiple data sources.

One disadvantage of data virtualization is that analytics workloads are executed on operational systems, which could interfere with their functioning. Another disadvantage is that the virtualization layer may act as a bottleneck on the performance of analytics operations.

# Data Visualisation

Data visualization involves presenting data in a visual format, enabling stakeholders to quickly grasp insights and make informed decisions. Effective visualization tools include dashboards and reports.

Can generate reports using:

- [Tableau](#)
- [PowerBI](#)
- [Looker Studio](#)

# Data Warehouse

A Data Warehouse (DWH) is a centralized repository designed for [Querying](#) and analysis, storing large volumes of structured data from various sources within an organization. It supports reporting and decision-making by providing a consolidated view of data.

## Key Features

**Data Ingestion Integration:** Combines data from diverse sources (e.g., transactional databases, CRM systems) into a single repository, ensuring consistency.

## Introduction

Subject-Oriented: Organizes data around key business areas (e.g., sales, finance) rather than operational processes.

---

Non-Volatile: Data remains unchanged once entered, preserving historical data for long-term analysis.

Time-Variant: Stores data with a time dimension, enabling historical analysis and trend identification.

## Components

Data Sources: Internal (e.g., ERP systems) and external (e.g., market research data) origins of data.

### ETL

#### Data Storage

Metadata/Documentation & Meetings: Information about the data, including definitions and transformation rules, aiding in data management.

Access Tools: Tools for querying and analyzing data, such as SQL clients and business intelligence tools.

### Resources

- [Designing a Data Warehouse](#)
- [Why a Data Warehouse?](#)

## Data Transformation In Data Engineering

Data transformation in [Data Engineering](#) is a key step in data pipelines, often part of:

- [ETL \(Extract, Transform, Load\) ETL](#): Data is transformed before loading into the target system.
- [ELT \(Extract, Load, Transform\) ELT](#): Data is loaded first, then transformed for analysis.
- [EtLT \(Extract, “tweak”, Load, Transform\)](#): A hybrid approach combining elements of ETL and ELT.

Related:

- [ETL vs ELT](#) for a comparison.

## Data Transformation In Machine Learning

Transforming raw data into a meaningful format is necessary for building effective models.

- [Supervised Learning](#): Annotating datasets with correct labels (e.g., labeling images of apples vs. other fruits).
- Manual & Automated Labeling: Using human annotators or leveraging existing labeled datasets (e.g., Google reCAPTCHA).
- Feature Scaling & Encoding: Applying normalization and encoding to categorical variables.
- [Encoding Categorical Variables](#): Converting categorical data into numerical format for machine learning models.

## Database Index

In [DE\\_Tools](#) see:

- [https://github.com/rhyslwells/DE\\_Tools/blob/main/Explorations/SQLite/Indexing/Indexing.ipynb](https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLite/Indexing/Indexing.ipynb)
-

Related terms:

- [Covering Index](#)
- [Partial Index \(Index with where clause\)](#)

Indexing is a technique used to [speed up data retrieval](#) in [database](#). It achieves this by creating a separate structure, known as an index, that organizes specific columns of data for faster access. Better than scanning.

Commonly created on [primary keys](#) (unique for item) and foreign keys.

Indexes can also be created across multiple tables to enhance the performance of complex queries, especially those that involve joins. A special type of index, called a [covering index](#), includes all the necessary data within the index itself, further improving efficiency.

Example:

For instance, creating an index on the "title" column in the "movies" table can significantly reduce the time it takes to execute [Querying|queries](#) that search for movie titles.

## Using Indexes

Keep in mind that indexes consume additional storage space.

Creating Indexes: To improve search performance, create indexes on relevant columns. For example:

```
CREATE INDEX idx_title ON movies(title);
```

Analyzing Queries: Use the `EXPLAIN QUERY PLAN` command to check if a query is utilizing an index effectively.

Dropping Indexes: If an index is no longer needed, it can be removed using:

```
DROP INDEX idx_title;
```

## Space and Time Trade-offs

Space: Indexes require extra storage because they are built using B-Trees, which are hierarchical data structures.

Time: While indexes speed up data retrieval, creating and updating them can slow down data insertion and modification processes.

## How Indexes Work

- Data Structure: Indexes typically use a [B-tree](#) data structure, which allows for efficient searching.
- Node Structure: A B-tree organizes data into nodes, where each node contains links to the corresponding rows in the table. The data is sorted, enabling quick access.
- Search Mechanism: When searching, a binary search method is employed. This involves checking the middle of the data and deciding which side to search next, taking advantage of the ordered nature of B-trees for efficiency.

# Database Management System (Dbms)

A **Database Management System** (DBMS) is software that allows you to interact with and manage databases.

Easiest to use:

- [SQLite](#)
- [PostgreSQL](#)

Others:

- [MySQL](#)
- [MongoDB](#)
- [Oracle](#)

These systems enable users to perform [CRUD](#) operations while maintaining data integrity and providing tools for backup, security, and optimization.

Can be proprietary (paid, with support) or Open source (free, self-supported).

## Database Schema

A [Database Schema](#) is the structure that defines how data is organized in a [Database](#), used in [Data Management](#). It specifies the tables, columns, relationships, and constraints within the database. The schema is used for ensuring data is stored consistently and can be queried efficiently.

1. Definition and Components: A database schema represents the [structure](#) around the data, including tables, views, fields, relationships, and various other elements like indexes and triggers. It provides a framework for organizing and understanding data.
2. Importance of [Structure](#): Without a schema, data can be chaotic and difficult to [interpret](#). A well-defined schema organizes data, making it [manageable and meaningful](#).
3. Schema on Read vs. Schema on Write:
  - Schema on Read: Structure is applied when the data is read, useful for unstructured data stores.
  - Schema on Write: Structure is enforced when data is written, typical of traditional databases.
4. Design Influences: The design of a schema impacts database behavior. For example, schemas designed with tables connected by primary keys are optimized for transactional applications, while star schemas are designed for efficient read operations in data warehouses.
5. Performance Impact: A good schema can significantly [improve query performance](#), reducing processing [time](#) and [cost](#), and simplifying query complexity.
6. [Data Modelling](#): Despite being considered an old concept, data modeling remains crucial for creating effective schemas, particularly in the context of big data and analytics.
7. Iterative Process: Developing a data warehouse schema involves iterative refinement, starting with interviews to create a [conceptual data model](#), which is then tested and refined through multiple iterations before being implemented.
8. Strategic Importance: The strategic design and deployment of a database schema are vital for efficient data warehousing and analytics. Intracity specializes in this process, helping organizations define and execute their data strategies.

Related to:

## Introduction

- [Types of Database Schema](#)
  - [Implementing Database Schema](#)
- 

## Resources

[link](#)

# Database Storage

Methods and optimizations for storing, retrieving, and processing data in [database](#) systems.

[Columnar Storage](#)

[Row-based Storage](#)

[Vectorized Engine](#)

# Database Techniques

Techniques:

- [Soft Deletion](#)
- [Concurrency](#)
  - [Race Conditions](#)
- [Querying](#)
  - [SQL Joins](#)
  - [Stored Procedures](#)
  - Cleaning: Use **Levenshtein Distance** (if SQLite extension is available) to group similar entries.
- [Database Index|Indexing](#)
  - [Query Plan](#)
  - [Vacuum](#)

# Database

Databases manage large data volumes with scalability, speed, and flexibility. Key systems include:

- [MySQL](#)
- [PostgreSQL](#)
- [MongoDB](#)

They facilitate efficient [CRUD](#) operations and transactional processing ([OLTP](#)) structured by [Database Schema|schema](#) that organizes data into tables and relationships.

Key Features

- **Structured Data:** Organized for efficient CRUD operations, allowing reliable access.
- **Relational Databases:** Use SQL to manage data in tables with relationships expressed through foreign keys and joins, minimizing redundancy.

Structure

---

## Introduction

- Data is organized into tables (like spreadsheets) with columns (fields) and rows (records), enabling efficient storage and retrieval.
- 

## Flexibility

- Databases have a flexible schema that adapts to evolving requirements, unlike static solutions like spreadsheets.

## Related Ideas:

- Spreadsheets vs Databases
- Database Management System (DBMS)
- Components of the database
- Relating Tables Together
- Turning a flat file into a database
- Database Techniques

# Databricks Vs Snowflake

## Comparison between **Databricks** and **Snowflake**:

- **Databricks** is a versatile platform that emphasizes collaborative data science and engineering through interactive notebooks, making it suitable for advanced analytics and machine learning applications.
- **Snowflake**, on the other hand, focuses on **Data Warehouse** and offers a robust SQL interface for analytics, making it a preferred choice for organizations prioritizing data storage and reporting capabilities.

| Feature                         | Databricks                                                                                                                                       | Snowflake                                                                                                          |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>Primary Functionality</b>    | Unified analytics platform for big data processing and machine learning.                                                                         | Cloud-based data warehousing and analytics platform.                                                               |
| <b>Data Processing</b>          | Built on <b>Apache Spark</b> , optimized for large-scale data processing and machine learning workflows.                                         | Uses its own SQL-based engine for data warehousing; excels in querying structured data.                            |
| <b>Collaboration</b>            | Emphasizes collaboration through <b>notebooks</b> (e.g., Jupyter <code>.ipynb</code> files) that allow for interactive data analysis and coding. | Provides features for data sharing and collaboration but lacks the notebook interface.                             |
| <b>Data Structure</b>           | Supports both structured and unstructured data, integrating seamlessly with data lakes (e.g., Delta Lake).                                       | Primarily designed for structured data and semi-structured data (like JSON) stored in tables.                      |
| <b>Scalability</b>              | Uses clusters to scale up compute resources dynamically; suitable for big data workloads.                                                        | Offers automatic scaling of compute and storage resources, focusing on cost-effective scaling.                     |
| <b>Machine Learning Support</b> | Integrated support for ML libraries (e.g., MLlib, MLflow) to build and deploy machine learning models.                                           | Limited built-in support for machine learning, primarily used for data storage and querying.                       |
| <b>Query Language</b>           | Supports multiple programming languages (Python, R, Scala, SQL) within notebooks.                                                                | Primarily uses SQL for querying data, providing a familiar interface for data analysts.                            |
| <b>Deployment</b>               | Available on major cloud platforms (AWS, Azure, GCP); allows for more customization and flexibility in deployment.                               | Also cloud-native, designed for seamless deployment in the cloud, with less emphasis on infrastructure management. |
| <b>Use Cases</b>                | Ideal for big data analytics, data engineering, and data science projects requiring complex processing.                                          | Best suited for traditional data warehousing, business intelligence, and analytics use cases.                      |

# Databricks

## Databricks Overview

[!Summary]

Databricks is a cloud-based platform for **big data** processing built on **Apache Spark**. It provides an integrated workspace for collaboration among **data engineers**, data scientists, and analysts. Databricks on Azure simplifies Spark deployment by offering auto-scaling clusters, real-time analytics, and integration with various Azure services, such as Azure **Data Lake** for large-scale data storage.

### Cloud Platform Compatibility:

- Supports the big three cloud providers (AWS, Azure, GCP).
  - **Integration with Other Technologies:**
- Combines capabilities of:
  - **Apache Spark**
  - **Delta Lake**
  - **MLflow**
  - **Data Lakehouse Architecture:**
- Represents a combination of a data **Data Warehouse** and a **data lake**.

Core Components:

1. **Tables:**
  - Represents files and data sources.
2. **Clusters:**
  - Provides computing power for data processing.
3. **Notebooks:**
  - Similar to Jupyter notebooks; support multiple programming languages and allow for productionization of code.
4. **Workspaces:**
  - Collaborative environments for teams to work together.

Scalability

- Leverages the scalability of **Hadoop** while integrating advanced features for big data processing.

[Databricks vs Snowflake](#)

## Datasets

This note collects notes on datasets that are good examples for exploring various concepts.

## Heart Failure Prediction Dataset

- **Link:** [Heart Failure Prediction Dataset](#)
- **Useful for:** Exploring predictive modeling in healthcare.

## Time Series Exploration

- **Description:** There is a dataset with seasonality, bikes, which can be used to explore **Time Series** concepts.

## Numenta Anomaly Benchmark (NAB)

- **Link:** [Numenta Anomaly Benchmark \(NAB\)](#)
- **Columns:** timestamp, value
- **Description:** NAB is used to evaluate and compare the performance of different anomaly detection algorithms on a diverse set of time series data. It includes real-world and artificial time series data covering domains such as finance, transportation, and environmental monitoring.

## U.S. Census Bureau's International Data Base (IDB)

---

- **Link:** [International Data Base \(IDB\)](#)
- **Useful for:** Researchers, policymakers, and businesses studying population dynamics, forecasting future population growth, monitoring economic development, and comparing demographic and economic characteristics of different countries.

## Wikipedia Web Traffic Time Series Dataset

- **Link:** [Wikipedia Web Traffic Time Series Dataset](#)
- **Useful for:** Examining the dynamics of website traffic, understanding interactions with Wikipedia, and identifying patterns and trends in online behavior. It can be used to compare traffic across languages, analyze the popularity of articles, and track the evolution of articles over time.

## Debugging Ipynb

debugging jupyter cells

[https://www.youtube.com/watch?v=CY6uZloF\\_kQ](https://www.youtube.com/watch?v=CY6uZloF_kQ)

Sometimes disappears: <https://stackoverflow.com/questions/72671709/vs-code-debug-cell-disappears-arbitrarily-in-jupyter-notebook-view>

## Debugging

Debugging is the process of identifying, analyzing, and resolving bugs or defects in software while [Testing](#). It is a critical part of the [Software Development Life Cycle](#), ensuring that applications function correctly and efficiently. Debugging involves several techniques and tools to pinpoint the source of errors and fix them.

In [ML\\_Tools](#) see:

- [Debugging.py](#)
- [Testing\\_unittest.py](#)
- [Testing\\_Pytest.py](#)

### Key Concepts in Debugging

- **Types of Computational Bugs:** Understanding the types of bugs, such as cumulative rounding errors, integer overflow, and race conditions, is essential for effective debugging.
- **How to Manage/View Bugs:**
  - **Console Log/Dir:** Use console logging to output variable values and program states to the console, helping to trace the flow of execution.
  - **Availability in VSCode:** Visual Studio Code provides powerful debugging features like breakpoints, watch expressions, and call stacks to help developers inspect and modify code execution.
  - **Sample Script:** Creating a minimal script that reproduces the bug can simplify the debugging process by isolating the problem.

- **Logging in Python:** Python's logging module allows developers to record events, errors, and informational messages, which can be crucial for diagnosing issues.
- **Run and Debug:** Step through code execution using debugging tools to observe the program's behavior and identify where it deviates from expected results.
- **Log Point/Break Point:** Set breakpoints to pause execution at specific lines of code, allowing inspection of variables and program state at that moment.

## Solution Attempts

1. **Reproduce the Bug:** Simplifying the code to reproduce the bug helps in understanding its cause and facilitates easier sharing with others for collaborative debugging. Sharing on platforms like [StackBiz](#) can help others contribute to the solution.
2. **Automated Testing:** Implementing automated tests ensures that code changes do not introduce new bugs and that existing functionality remains intact.
3. **Test-Driven Development (TDD):** Writing tests before the actual code helps define expected behavior and ensures that the code meets these expectations.
4. **Static Analysis:** Tools like [TypeScript](#) and ESLint analyze code for potential errors without executing it, helping to catch issues early in the development process.

## Debugging Tools and Techniques

- **Integrated Development Environments (IDEs):** IDEs like Visual Studio Code, IntelliJ IDEA, and Eclipse offer built-in debugging tools that streamline the debugging process.
- **Version Control Systems:** Tools like [Git](#) allow developers to track changes and revert to previous versions if a bug is introduced.
- **Profilers:** These tools analyze program performance and help identify bottlenecks or inefficient code paths.
- **Memory Analyzers:** Tools like Valgrind help detect memory leaks and other memory-related issues.

## Debugging.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Utilities/Debugging.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Debugging.py)

This script includes examples of logging, using breakpoints, and reproducing a simple bug for practice

## Key Concepts Demonstrated in the Script

1. **Logging in Python:** The script uses Python's logging module to record debug, info, error, and warning messages. This helps track the flow of execution and diagnose issues.
2. **Reproduce the Bug:** The script intentionally includes a division by zero bug to demonstrate how to identify and fix it.
3. **Breakpoints:** You can set a breakpoint in your IDE at the line where `result = divide_numbers(num1, num2)` to inspect the values of `num1` and `num2`.
4. **Automated Testing:** The script includes a simple assertion to test the `divide_numbers` function, ensuring it behaves as expected.

5. **Static Analysis:** The commented line `unused_variable = 42` can be used to simulate a static analysis warning for an unused variable.

---

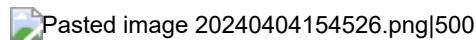
## Decision Tree

A Decision Tree is a type of [Supervised Learning](#) algorithm used to predict a target variable based on input features. It involves splitting data into subsets to create a tree-like model.

Decision Tree Structure are a flowchart-like model where each internal node represents a decision based on a feature, branches represent outcomes, and leaf nodes represent final predictions.

Splits data recursively based on feature importance, forming a tree-like structure.

The decision tree algorithm calculates the [Gini impurity](#) for each possible split and selects the one with the lowest impurity. Use to make predictions on new data, the algorithm traverses the decision tree from the root node to a leaf node, following decision rules based on input features. Once it reaches a leaf node, it assigns the corresponding class label or prediction.



## Key Concepts

1. Objective: Predict a target variable using input features.
2. Splitting: Identify the best feature to split the data into subsets, aiming for homogeneous groups.
3. Impurity Calculation: Use metrics like [Gini Impurity](#) or [Cross Entropy](#) ([Gini Impurity vs Cross Entropy](#)) to evaluate splits. Choose the split that minimizes impurity.
4. Purity: A node is pure if it perfectly classifies the data, requiring no further splits.
5. Leaf Node Output: Assigns the most common class label or average value in the node.
6. [Overfitting](#): Can occur if the tree is too complex. Mitigate with [pruning](#) and limiting tree depth.
7. [Cross Validation](#): Refine the model to better generalize to new data.

## Splitting Process

The splitting process in a Decision Tree involves dividing the dataset into subsets to create a tree-like structure. This process is crucial for building an effective model that can predict target variables accurately.

Splitting Criteria:

- The algorithm evaluates various features to determine the best split at each node.
- It selects the feature and split point that minimize impurity in the resulting child nodes.
- The split that most effectively reduces impurity is chosen, ensuring that each subset is as homogeneous as possible.

## Building Process

### 1. Initial Splitting:

- Begin at the root node and select the best feature to split the data. This selection is based on impurity measures such as Gini impurity or entropy for [classification](#) tasks, and variance reduction for regression tasks.
- The goal is to create subsets that are as homogeneous as possible with respect to the target variable.

### 2. Recursive Partitioning:

- After the initial split, each subset becomes a child node.

- The algorithm recursively applies the splitting process to each child node.
- Continue splitting until stopping criteria are met, such as reaching a maximum tree depth, having a minimum number of samples per node, or achieving insufficient improvement in purity.

### 3. Leaf Nodes:

- The process continues until reaching leaf nodes, which have no further splits.
- At each leaf node, assign a class label (for classification) or predict a continuous value (for regression) based on the majority class or average value of the samples in that node.

## Refinement

Pruning:

- Pre-pruning: Stop tree growth early based on criteria like maximum depth or minimum impurity improvement.
- Post-pruning: Allow the tree to grow fully, then prune back based on performance metrics.

## Hyperparameter

Can use [GridSearchCV](#) to pick the best parameters.

| Parameter                      | Purpose                           | Effect                  | Example                                                                         |
|--------------------------------|-----------------------------------|-------------------------|---------------------------------------------------------------------------------|
| <code>criterion</code>         | Splitting criteria                | Impacts decision logic. | <code>criterion='gini' OR criterion='entropy'</code>                            |
| <code>max_depth</code>         | Maximum tree depth                | Prevents overfitting.   | <code>max_depth=5</code> limits the tree depth to 5.                            |
| <code>min_samples_split</code> | Min samples to split a node       | Limits tree growth.     | <code>min_samples_split=10</code> requires at least 10 samples to split a node. |
| <code>min_samples_leaf</code>  | Min samples at leaf node          | Reduces overfitting.    | <code>min_samples_leaf=5</code> ensures every leaf has at least 5 samples.      |
| <code>max_features</code>      | Features considered for splitting | Adds randomness.        | <code>max_features='sqrt' OR max_features=3</code> .                            |
| <code>max_leaf_nodes</code>    | Max leaf nodes allowed            | Reduces overfitting.    | <code>max_leaf_nodes=20</code> caps the tree at 20 leaves.                      |
| <code>class_weight</code>      | Adjusts for imbalanced data       | Improves fairness.      | <code>class_weight='balanced' OR class_weight={0:1, 1:2}</code> .               |
| <code>ccp_alpha</code>         | Pruning parameter                 | Simplifies tree.        | <code>ccp_alpha=0.01</code> prunes weak splits based on complexity.             |

## Advantages and Disadvantages of Decision Trees

Advantages:

- Simple and [interpretability|interpretable](#) model.
- Minimal data preparation required.
- Transparent decision-making process.

Disadvantages:

- 
- Prone to overfitting, especially with complex datasets.
  - Sensitive to small changes in data.
  - Can become complex with many features.

[Decision Tree](#)

## Deep Learning Frameworks

[Watch Overview Video](#)

### TensorFlow

**Focus:** TensorFlow is a comprehensive open-source platform for machine learning. It provides a flexible and comprehensive ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and developers easily build and deploy ML-powered applications.

**Integration:** TensorFlow can implement a wide range of machine learning algorithms, including those available in [Sci-kit Learn](#), making it versatile for various applications.

**Modularity:** Its modular architecture allows users to deploy computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

**Parallelization:** TensorFlow is optimized for high-performance numerical computation, making it suitable for large-scale machine learning tasks that require parallel processing.

**Use Cases:** TensorFlow is widely used in both academic research and industry for tasks such as image and speech recognition, natural language processing, and more.

### Sci-kit Learn

**Focus:** Sci-kit Learn is a simple and efficient tool for data mining and data analysis, built on NumPy, SciPy, and matplotlib. It is primarily used for traditional machine learning techniques such as classification, regression, clustering, and dimensionality reduction.

**Limitations:** While excellent for classical machine learning tasks, Sci-kit Learn is not designed for deep learning or neural network architectures, which require more specialized frameworks like TensorFlow or PyTorch.

### Keras

**API Level:** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It allows for easy and fast prototyping through user-friendly, modular, and extensible code.

**Integration:** Keras is tightly integrated with TensorFlow 2.0, providing a simplified interface for building and training deep learning models.

**Purpose:** Designed to enable fast experimentation, Keras is ideal for beginners and researchers who need to quickly prototype and test new ideas.

**Performance:** While Keras simplifies model building, it may not be as performant as lower-level frameworks like TensorFlow when it comes to fine-tuning and optimizing models for production.

## Deep Learning

[!Summary] Deep learning is a subset of machine learning that uses neural networks to process large-scale data for tasks like image and speech recognition, natural language processing, and recommendation systems.

A neural network consists of layers of nodes where each node performs weighted sums of its inputs, applies activation functions like ReLU or sigmoid, and produces an output.

**Backpropagation** is the primary algorithm for training neural networks by minimizing error through **Gradient Descent**. Regularization techniques, such as dropout, prevent overfitting.

Popular frameworks like **PyTorch** and **TensorFlow** facilitate deep learning model development.

Questions:

- [What is the role of gradient-based optimization in training deep learning models.](#)
- [Explain different gradient descent algorithms, their advantages, and limitations.](#)

Areas of Deep Learning:

- [LLM](#)
- [Neural network|Neural Network](#)

[!Follow up questions]

- How does the choice of activation function affect the performance of deep learning models across different tasks?
- What are the trade-offs between different gradient descent algorithms (e.g., [Stochastic Gradient Descent|SGD](#) vs. Adam) in training neural networks? See [Optimisation techniques](#).

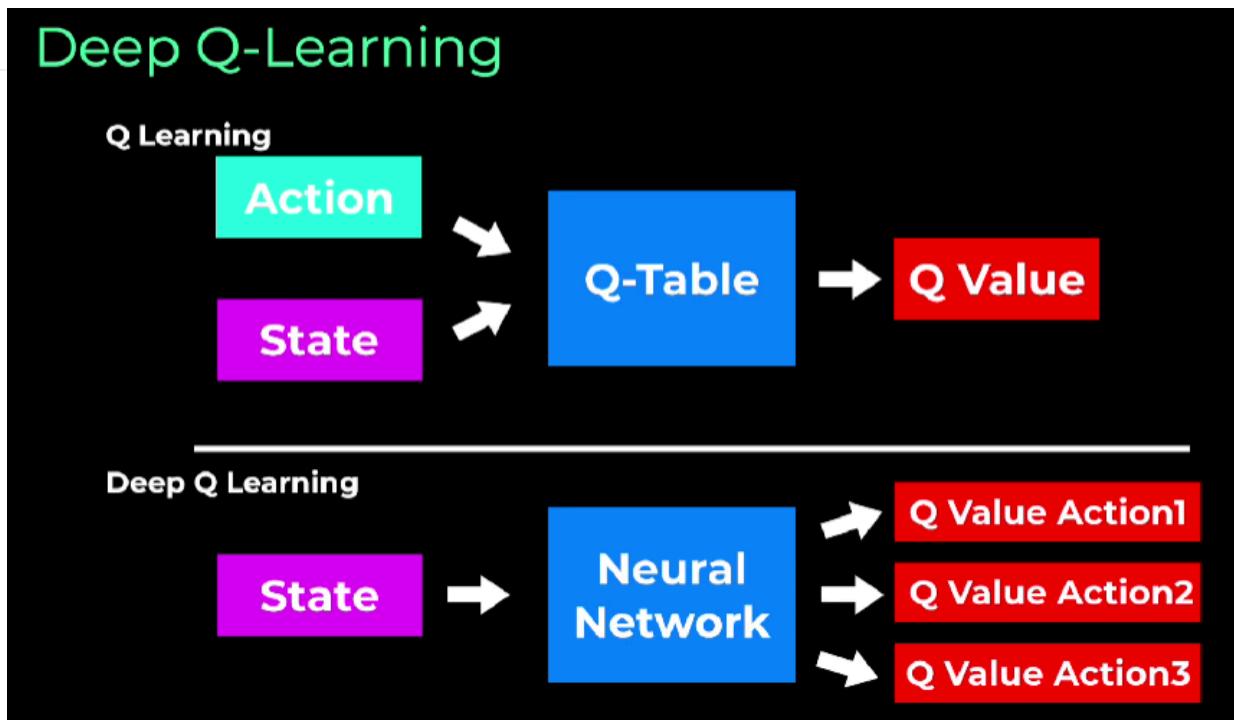
[!Related Topics]

- [Transfer Learning](#): Applying pre-trained models to new tasks.

## Deep Q Learning

Deep [Q-Learning](#) is a type of [reinforcement learning](#) algorithm that combines Q-Learning with [Neural network](#). Necessary when Q-Table grows too large.

Updates the weights in the model.



## Key Concepts

### Target Network

- Purpose:** The target network is used to stabilize the training process in Deep Q-Learning.
- When is it needed?:** It is needed when updating the Q-values to prevent oscillations and divergence during training.
- How it works:** The target network is a copy of the main Q-network and is used to generate target Q-values. It is updated less frequently than the main network, often using a technique called a "soft update," where the target network is slowly adjusted towards the main network over time.

### Experience Replay

- Purpose:** Experience replay is used to break the correlation between consecutive experiences, which can lead to inefficient learning and instability.
- Issue it resolves:** When an agent learns from sequential experiences, the strong correlations between them can cause problems such as oscillations and instability in learning.
- How it works:**
  - Experiences (state, action, reward, next state) are stored in a memory buffer.
  - During training, random mini-batches of experiences are sampled from this buffer to update the network.
  - This random sampling helps to generate uncorrelated experiences, improving stability and efficiency.
  - It also allows the agent to reuse experiences for multiple updates, increasing data efficiency.

## Deepseek

[LLM example](#)

open source

optimising for performance vs efficiency.

Introduction

Deepseek leading in efficiency

---

o3 mini

[Chain of thought](#) can see it - ui choice

[Distillation](#) - use gpt output and trains on it.

security

## drafting

[jevon paradox](#) - as cost decreases usage increases

edge inference [Edge Machine Learning Models](#)

[The Genius of DeepSeek's 57X Efficiency Boost](#) key value caching impacts attention block compute scaling: linear increased memory usage Solution: multi-query attention vs mutli - head attention Grouped-query attention Multi-head latent attention - deepseek uses uses compresses latent space linear algebra absorbed weights at training

- **Access to Advanced AI Features Without Payment:** You can now access powerful reasoning models like DeepSeek's R1 and ChatGPT's 03 Mini for free. These models are good at complex math, programming, and step-by-step reasoning.
- **Privacy Protection:** If you are concerned about privacy, you have options to protect your data by using platforms like Perplexity, Venice AI, or Cursor to access DeepSeek models, which keeps data in the US. For full privacy, you can run DeepSeek models locally using LM Studio or oLama, but this may limit access to more powerful models due to hardware constraints.
- **Smart Choices About Switching:** Consider if DeepSeek offers clear advantages for your specific needs before changing your current AI tools or workflows. DeepSeek is beneficial for developers focused on cost minimization. If you are an everyday user already paying for ChatGPT and concerned about data storage, switching may not be necessary unless DeepSeek significantly improves your workflow.

## Deleting Rows Or Filling Them With The Mean Is Not Always Best

## Demand Forecasting

- **Overview:** Demand response programs encourage consumers to adjust their energy usage during peak periods in response to time-based rates or other incentives. RL can optimize how these programs are implemented.
- **Applications:**
  - **Incentive Management:** Reinforcement learning|RL models can dynamically adjust incentives for consumers to reduce usage during peak times based on real-time grid conditions and consumer behavior.
  - **Behavioral Adaptation:** By learning from historical consumer response data, RL systems can predict how different consumers will react to incentives, allowing for more tailored and effective demand response strategies.

How can we model the effects of energy consumption patterns on demand forecasting

---

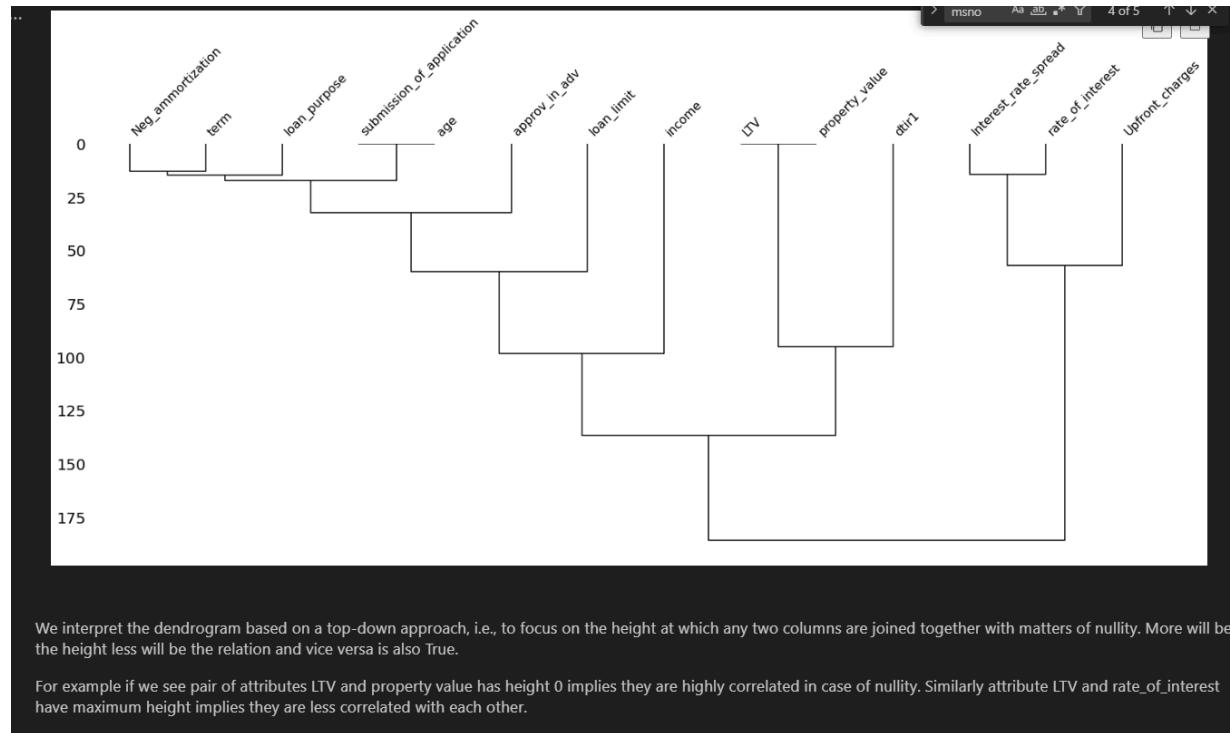
## Introduction

- **Dynamic Programming:** Useful in solving multi-stage decision problems, such as optimal scheduling of power plants.
- **Linear Programming:** Used for optimizing resource allocation in energy production and distribution, such as maximizing output while minimizing costs. -

## Dendograms

Dendograms show **close** vectors in the data where taken as a vector.

Can tell which **features are the most similar** with [Dendograms](#)



## Design Thinking Questions

- "What is the user's need?"
- "What constraints are at play?"
- "How might we...?" — a classic starter for idea generation.

## Determining Threshold Values

In [Binary Classification](#) problems, a threshold value is used to convert predicted probabilities into discrete class labels. The choice of threshold significantly impacts the model's performance, affecting [Evaluation Metrics](#).

Important Considerations:

- [Imbalanced Datasets|Class Imbalance:](#) If the classes are imbalanced, the choice of threshold can be significantly affected. Techniques like oversampling, undersampling, or using weighted loss functions can help mitigate the impact of class imbalance.

## Introduction

- **Data Quality:** The quality of the training data can also influence the choice of threshold. If the data is noisy or contains outliers, the chosen values may not be optimal.
- Choose **Evaluation Metrics** that are appropriate for the specific problem and the desired trade-off between different types of errors.

Here are common methods for determining the optimal threshold value:

- Receiver Operating Characteristic (ROC) Curve Analysis : [ROC \(Receiver Operating Characteristic\)](#)
- [Precision-Recall Curve](#) Analysis
- [Cost-Sensitive Analysis](#)

## Devops

DevOps refers to practices for collaboration and automation between [Software Development Portal \(Dev\)](#) and IT operations (Ops) teams, aiming for faster, more reliable software delivery.

**Integration:** It integrates the work of software development and operations teams by fostering a culture of collaboration and shared responsibility.

### Principles:

- Emerges from Agile principles.
- Emphasizes collaboration between development and operations teams.

### Approach:

- Utilizes continuous integration and continuous delivery ([CI-CD](#)) to ensure frequent code changes and quick feedback loops.
- Enables rapid and reliable updates with high levels of automation and efficiency.

### Goals:

- Ensures existing processes are optimized and streamlined.

Related to:

- [DataOps](#)

## Difference Between Databricks Vs. Snowflake

## Difference Between Snowflake To Hadoop

Snowflake and Hadoop are both [Data Management](#) systems, but they serve different purposes and have distinct architectures and functionalities.

In summary, Snowflake and Hadoop are both powerful tools for managing and analyzing data, but they are optimized for different types of workloads and use cases. Snowflake excels in [cloud-based data warehousing](#) and real-time analytics, while Hadoop is suited for [large-scale data processing](#) and storage in a distributed environment.

[Snowflake](#)

[Hadoop](#)

## Key Differences

- 
1. **Deployment:**
    - **Snowflake:** Cloud-based, requires no hardware or infrastructure management by users.
    - **Hadoop:** Can be deployed on-premises or in the cloud, but typically requires more hands-on management.
  2. **Ease of Use:**
    - **Snowflake:** User-friendly with a simple SQL interface, automated maintenance, and optimization.
    - **Hadoop:** Requires more technical expertise to set up, manage, and optimize.
  3. **Performance and Scalability:**
    - **Snowflake:** Excels in performance for analytical queries with the ability to scale compute resources independently.
    - **Hadoop:** Scales horizontally by adding more nodes, suitable for large-scale data processing but may have higher query latency.
  4. **Cost:**
    - **Snowflake:** Pay-as-you-go model based on compute and storage usage.
    - **Hadoop:** Costs depend on the infrastructure (hardware or cloud resources) and maintenance overhead.

## Example Use Case Scenarios

- **Snowflake:** A retail company wanting to perform real-time analytics and reporting on sales data would benefit from Snowflake's high performance and ease of use for SQL-based queries and dashboards.
- **Hadoop:** A tech company needing to process and analyze massive amounts of log data for machine learning models might use Hadoop due to its ability to handle large-scale data processing and diverse data types.

## Differentiation

## Forward Mode Automatic Differentiation

uses dual numbers

implemented in tensor flow

see also Reverse Mode Automatic Differentiation

Fast, Flexible, Exact

## Digital Transformation

"Digital transformation starts with data centralisation"

To digitally transform your department, you'll need to approach the process in a structured and strategic way that addresses both technological and organizational changes.

## Data Audit

- 
- **Understand Department Processes:** Identify current workflows, key processes, and technologies being used.

- **Identify Pain Points:** Collect feedback from employees on inefficiencies, bottlenecks, and areas for improvement.
  - **\*\*Data Collection and Analysis:\*\*** Review existing data and determine how it's being used (or underutilized) in decision-making processes.
- 

## 2. Define Clear Objectives

- **Set Transformation Goals:** Align transformation efforts with the broader goals of the organization. For example, improving efficiency, enhancing customer experience, or better data-driven decision-making.
- **Quantifiable Metrics:** Define success metrics (e.g., reduced processing time, increased customer satisfaction, or data accuracy).

## 3. Engage Stakeholders

- **Get Leadership Buy-In:** Ensure leadership is aligned with the transformation and committed to driving it forward.
- **Collaborate with Employees:** Involve employees in planning to understand their needs and gain their support for changes.
- **Map Stakeholders:** Identify key influencers, decision-makers, and implementers within the department.

## 4. Evaluate and Select Technologies

- **Research Tools and Platforms:** Evaluate technologies such as cloud platforms, automation tools, analytics solutions, and collaboration software.
- **Pilot New Technologies:** Test small-scale pilots to validate the value of proposed solutions before large-scale rollouts.
- **Interoperability:** Ensure that new technologies can integrate smoothly with existing systems and data.

## 5. Build a Transformation Roadmap

- **\*\*Prioritize Initiatives:\*\*** Focus on high-impact areas that align with the department's goals.
- **Create a Timeline:** Develop a step-by-step implementation timeline, including milestones and deadlines.
- **Allocate Resources:** Assign teams, budgets, and technology resources for each phase of the transformation.

## 6. Change Management and Training

- **Implement Change Management Strategies:** Proactively manage resistance to change by communicating the benefits of digital transformation.
- **Provide Training and Support:** Train staff on new technologies and provide ongoing support to ensure a smooth transition.
- **Foster a Digital Culture:** Encourage a mindset of innovation, experimentation, and continuous improvement.

## 7. Implement New Technologies and Processes

- **Roll Out in Phases:** Implement technology solutions gradually, allowing time for adjustments and feedback.
- **Monitor and Iterate:** Regularly check the implementation process and adjust based on feedback and performance metrics.

## 8. Measure and Optimize

- **Track Key Metrics:** Use the pre-defined success metrics to measure the impact of digital transformation.
-

- **Continuously Improve:** Adjust and optimize workflows, tools, and processes based on performance and evolving needs.
- 

## 9. Ensure Long-Term Sustainability

- **Encourage Innovation:** Promote continuous learning and adoption of new technologies to keep the department adaptable to future changes.
- **Monitor Industry Trends:** Stay updated on new digital trends and opportunities that can benefit the department.
- **Review and Update Goals:** Periodically revisit your transformation goals and adjust them based on organizational needs and market shifts.

### Example Focus Areas:

- **Automation:** Streamline repetitive processes using Robotic Process Automation (RPA) or AI.
- **Data-Driven Decision Making:** Introduce analytics tools and dashboards for real-time insights.
- **Cloud Adoption:** Shift to cloud-based platforms for better scalability, collaboration, and remote work capabilities.
- **Collaboration Tools:** Implement communication and project management platforms like Microsoft Teams or Slack to enhance collaboration.

### Digital Transformation

Businesses have data, but need to evaluate, organise, clean , and prepare before using.

Digital Transformation is a [Change Management|Change Program](#).

Where a business can benefit. Areas that can be improved:

- reporting and financial
- data quality / backend
- network management
- Customer services

What we want to move away from/towards

- Silos in our work
- No KPIs
- Multiple spreadsheets doing the same thing
- One-time reporting (get a standarised process in place)
- Systems that are not understood or replicatable
- Poor data quality
- Missed opportunities
- Small, reactive team to issues.

Client focus:

- What does the client want? A fuller understanding of the network. The ability to dive deeper if they wanted to. The ability to understand their bills (AccM). To be able to plan their projects better knowing the information we have?
  - What can the water usage tell us about a site/all sites, that could be beneficial to the client? Stress levels on the system - predictive maintenance? Does high usage result in higher leakage.
  - What are the reasons for leaks? - non error - what happened on the system that caused it?
- 
-

Digital transformation aims to provide innovation to business processes.

- save time
- do more,
- save money.

Digital transformation aims to:

- Automate repetitive tasks.
- Create systematic improvements to business sops, through strategies, methodologies, technologies.
- How can we prepare digital assets for further automation?

To consider when conducting:

- Conduct a Data audit company wide, what is the format of data and where is it? is the data accessible?
- Reporting mechanisms what can you do with the data
- How to handle data debt/minimise it.
- What does leadership want as a data/product pathway.
- How can we continuously improve the quality of our data?
- Need to track by design, i.e. what do we need to know to benefit in the future.

## Digital Twin

[!Summary] A **digital twin** is a virtual representation of a physical object, system, or process that mirrors its real-world counterpart in real-time. This digital model is used to simulate, monitor, analyze, and optimize the physical entity by continuously updating based on data collected from sensors, devices, or other inputs. The concept is widely applied in industries such as manufacturing, healthcare, **Energy**, smart cities, and more to improve decision-making, predictive maintenance, and efficiency. A digital twin is a powerful tool for enhancing real-time decision-making, optimizing processes, and predicting future performance by bridging the physical and digital worlds. Its applications continue to expand across various industries, helping organizations to reduce costs, improve efficiency, and innovate faster.

[!Example] Consider a **digital twin of a wind turbine**. Sensors installed on the turbine gather data on operational conditions such as wind speed, blade position, temperature, and vibration. This data is continuously transmitted to the digital twin, which mirrors the turbine's state in real-time. The digital twin runs simulations to predict when parts of the turbine might fail due to wear and tear. Maintenance teams can use this information to schedule repairs before a breakdown occurs, minimizing downtime and improving the turbine's efficiency.

### Key Components of a Digital Twin:

#### 1. Physical Object or Process:

- The real-world entity (such as a machine, a building, a production line, or even a human body) that the digital twin replicates.

#### 2. Digital Model:

- A virtual replica of the physical entity, designed using data models, physics-based simulations, and other analytical tools. The digital model reflects the structure, behavior, and function of the physical object or process.

#### 3. Data Integration:

- The digital twin **relies on real-time data from sensors**, IoT devices, or historical databases connected to the physical counterpart. This data flow enables the twin to reflect current operating conditions and states.

#### 4. Analytics and Simulation:

- Advanced analytics (e.g., machine learning, artificial intelligence) and simulations are applied to the digital twin to gain insights into the performance, predict future behavior, and test scenarios that would be difficult or expensive to replicate in the real world.

#### 5. Feedback Loop:

- A digital twin allows for continuous interaction between the physical and digital worlds. Insights or predictions from the digital twin can inform changes to the physical system, and any updates in the physical system feed back into the digital twin, maintaining accuracy and alignment.

## Types of Digital Twins:

### 1. Component/Asset Twin:

- Represents individual components or parts of a larger system (e.g., the digital twin of a jet engine or an electric motor).

### 2. System or Unit Twin:

- Models entire systems or units, such as a production line in a factory or the electrical system of a building.

### 3. Process Twin:

- Focuses on simulating and optimizing processes, such as a manufacturing workflow or supply chain operations.

### 4. Environment Twin:

- Used to simulate larger, more complex systems like cities, ecosystems, or large-scale infrastructure (e.g., smart city initiatives or environmental monitoring).

## Applications of Digital Twins:

### 1. Manufacturing:

- In smart factories, digital twins are used to simulate production processes, predict machine failures, optimize maintenance schedules, and improve product design by running real-time simulations of manufacturing conditions.

### 2. Healthcare:

- Digital twins of patients are being developed to model individual health profiles, allowing for personalized treatment plans and predictive diagnostics. A digital twin of a human organ, for example, could simulate medical treatments before they are applied to the patient.

### 3. Energy:

- In energy systems, digital twins help optimize the operation of power plants, monitor grid performance, and simulate the impacts of renewable energy integration, improving reliability and efficiency.

### 4. Smart Cities:

- Urban planners use digital twins to model traffic flow, infrastructure usage, or environmental conditions. This allows them to simulate different scenarios and optimize city operations, reduce congestion, and improve public services.

### 5. Aerospace and Automotive:

- Digital twins are used extensively in designing, testing, and maintaining complex systems like aircraft, satellites, and autonomous vehicles. Engineers can simulate operational conditions to identify potential problems before they occur in the physical system.

### 6. Building Management:

- Digital twins of buildings or infrastructure monitor and control systems like HVAC, lighting, and security, improving energy efficiency and safety. They are also used for simulating how a building will perform under different conditions (e.g., weather events or occupancy changes).

## Benefits of Digital Twins:

1. **Real-time Monitoring:**
  - Provides live feedback from the physical entity, which enables organizations to make faster, more informed decisions.
2. **Predictive Maintenance:**
  - Predicts when equipment or systems are likely to fail based on real-time data and simulations, reducing downtime and maintenance costs.
3. **Optimization:**
  - Enables the continuous improvement of processes by testing scenarios in a virtual environment without disrupting real-world operations.
4. **Improved Design and Innovation:**
  - Digital twins allow engineers and designers to experiment with different configurations, materials, or processes virtually, leading to faster, cheaper, and more innovative solutions.
5. **Reduced Risk:**
  - By simulating potential failures or dangerous scenarios in the digital world, organizations can assess risk and plan mitigation strategies without putting the physical system at risk.

## Challenges:

1. **Data Management:**
  - Digital twins require a large amount of real-time data to maintain accuracy. Collecting, managing, and processing this data efficiently can be complex and costly.
2. **Integration:**
  - Integrating the digital twin with physical systems, particularly in legacy environments, can be challenging due to compatibility issues and the need for IoT infrastructure.
3. **Security:**
  - Because digital twins rely on real-time data transmission, they are vulnerable to cyberattacks, which can lead to compromised systems or intellectual property theft.
4. **Scalability:**
  - Scaling digital twin models to encompass entire cities or large systems involves high computational and infrastructural requirements.

## Dimension Table

A dimension table is a key component of a [star schema](#) or snowflake schema in a data warehouse. It provides descriptive attributes (or dimensions) related to the [Facts](#) stored in a fact table.

They provide the context and descriptive information necessary for analyzing the quantitative data stored in fact tables (e.g., product names, customer demographics, time periods).

1. **Descriptive Attributes:** Dimension tables contain qualitative data that describe the entities involved in the business process. For example, a product dimension table might include attributes such as product name, category, brand, and manufacturer.
2. **Primary Key:** Each dimension table has a primary key that uniquely identifies each record in the table. This primary key is used as a foreign key in the [Fact Table](#) to establish relationships between the two.
3. **Hierarchies:** Dimension tables often include hierarchies that allow for data to be analyzed at different levels of granularity. For example, a time dimension might include attributes for year, quarter, month, and day, allowing users to drill down or roll up in their analysis.
4. **Smaller Size:** Compared to fact tables, dimension tables are typically smaller in size, as they contain descriptive data rather than large volumes of transactional data.
5. **Static Data:** Dimension tables usually contain relatively static data that does not change frequently, such as product details or customer information. However, they can be updated as needed to reflect changes in the business.
6. **Support for Filtering and Grouping:** Dimension tables enable users to filter and group data in reports and analyses. For example, users can analyze sales data by different dimensions such as time, geography, or product category.

#### Examples

- **TimeDimension:** Contains information about the time period.
  - Columns: DateKey , Year , Quarter , Month , Day
- **ProductDimension:** Contains product details.
  - Columns: ProductKey , ProductName , ProductCategory
- **RegionDimension:** Contains regional information.
  - Columns: RegionKey , RegionName , Country

Dimension Table Tags: #data\_modeling, #data\_warehouse

## Dimensional Modelling

Dimensional modeling is a design technique used in [Data Warehouse](#) used to structure data for efficient [retrieval](#) and analysis. It is particularly well-suited for organizing data in a way that supports complex [queries](#) and reporting, making it easier for business users to understand and interact with the data.

Dimensional modeling is a foundational technique in building data warehouses and is often associated with methodologies like the [Kimball](#) approach, which emphasizes the use of [Star Schema](#) and the importance of understanding business processes and user requirements.

#### Key Concepts in Dimensional Modeling

- **Fact Table & Facts**
  - [Dimension Table](#)
  - [Grain](#)

Benefits of Dimensional Modeling: [Performance Dimensions](#)

Dimensional Modelling Tags: #data\_modeling, #data\_warehouse

# Dimensionality Reduction

---

Dimensionality reduction is a step in the [Preprocessing](#) phase of machine learning that helps simplify models, enhance interpretability, and improve computational efficiency.

It's a technique used to reduce the number of input variables (features) in a dataset while retaining as much information as possible. This process is essential for several reasons:

1. **Improves Model Performance:** Reducing the number of features can help improve the performance of machine learning models by minimizing overfitting and reducing noise.
2. **Enhances Visualization:** It allows for easier [Data Visualisation](#) of high-dimensional data by projecting it into lower dimensions (e.g., 2D or 3D).
3. **Reduces Computational Cost:** Fewer features mean less computational power and time required for training models.

## Common Techniques

- **Principal Component Analysis (Principal Component Analysis):** A statistical method that transforms the data into a new coordinate system, where the greatest variance by any projection lies on the first coordinate ([principal component/orthogonal components](#)), the second greatest variance on the second coordinate, and so on.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE):** A technique particularly well-suited for visualizing high-dimensional data by reducing it to two or three dimensions while preserving the local structure of the data. t-SNE is a non-linear technique used for visualization and dimensionality reduction by preserving pairwise similarities between data points, making it suitable for exploring high-dimensional data.
- [Linear Discriminant Analysis](#) method used for both classification and dimensionality reduction, which finds a linear combination of features that best separates two or more classes.

## [Explain the curse of dimensionality](#)

## Dimensions

Dimensions are the categorical buckets that can be used to segment, filter, or group—such as sales amount region, city, product, color, and distribution channel.

Traditionally known from [OLAP \(online analytical processing\)](#)|[OLAPcubes](#) with Bus Matrixes, and [Dimensional Modeling](#).

They provide context to the [Facts](#).

## Directed Acyclic Graph (Dag)

DAG stands for [Directed Acyclic Graph](#).

A DAG is a graph where information must travel along with a finite set of nodes connected by vertices. There is no particular start or node and also no way for data to travel through the graph in a loop that circles back to the starting point.

---

It's a popular way of building data pipelines in tools like [Apache Airflow](#), [dagster](#), [Prefect](#). It clearly defines the data lineage. As well, it's made for a functional approach where you have the [idempotency](#) to restart pipelines without side-effects.

## Directory Structure

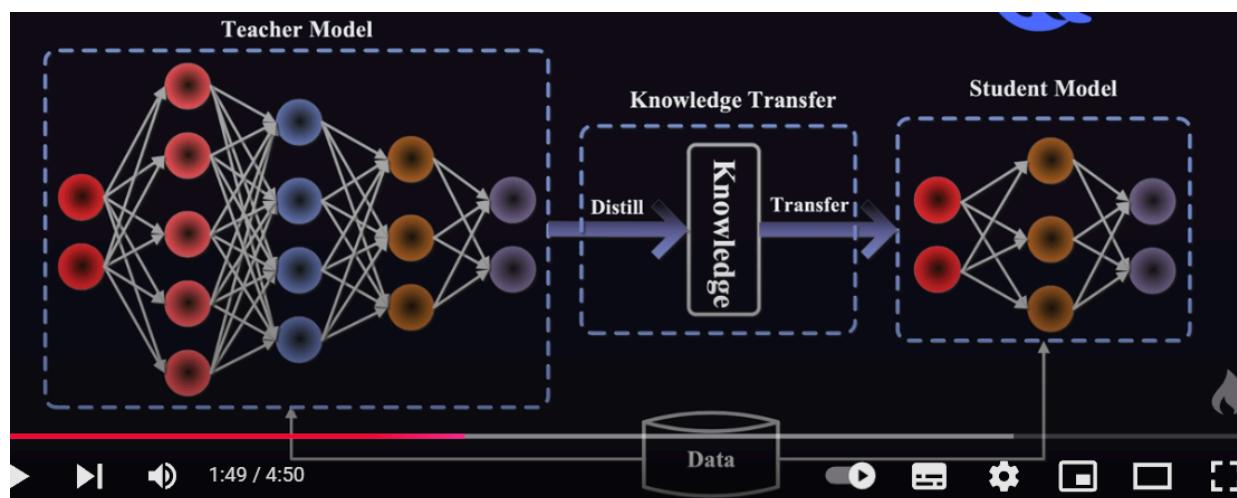
[To make a file tree](#)

```
|── README.md <- The top-level README for developers using this project. |── data |── external <- Data
from third party sources. |── interim <- Intermediate data that has been transformed. |── processed <- The
final, canonical data sets for modeling. |── raw <- The original, immutable data dump. |── models <- Trained
and serialized models, model predictions, or model summaries |── notebooks <- Jupyter notebooks. Naming
convention is a number (for ordering), the creator's initials, and a short - delimited description, e.g. |── 1.0-jqp-
initial-data-exploration. |── reports <- Generated analysis as HTML, PDF, LaTeX, etc. |── figures <-
Generated graphics and figures to be used in reporting |── requirements.txt <- The requirements file for
reproducing the analysis environment, e.g. |── generated with pip freeze > requirements.txt |── src <- Source
code for use in this project. |── data <- Scripts to download or generate data |── make_dataset.py |──
features <- Scripts to turn raw data into features for modeling |── build_features.py |── models <- Scripts to
train models and then use trained models to make |── predictions |── predict_model.py |── train_model.py
|── visualization <- Scripts to create exploratory and results oriented visualizations └── visualize.py
```

## Distillation

training smaller models with larger.

[Small Language Models](#)



## Distributed Computing

**Distributed Computing** is essential for managing **massive data volumes** by distributing tasks across multiple servers or machines. This enables scalability and efficient data processing.

[Hadoop](#) is a framework that handles both data storage and processing across **clusters of servers**:

- It ensures **scalability**: can easily grow as more data is added.
  - Provides **redundancy**: data is replicated across servers to prevent loss in case of failures.
- 

Tools like [Apache Spark](#) are built to process data in these **distributed environments**, allowing for fast, parallel processing across the cluster.

Distributed computing is central to modern data handling, driven by frameworks like Spark and Hadoop, supported by cloud infrastructure, and expanding into real-time and [edge computing](#).

## Distributed Computing: Current State

Distributed computing enables the processing of massive datasets and computational tasks by distributing them across multiple machines. This approach increases [scalability](#), parallelism, and fault tolerance, making it essential for modern data processing.

## Key Frameworks

- **Hadoop**: An early pioneer, Hadoop introduced distributed storage via **HDFS** and data processing with **MapReduce**, allowing tasks to be split across clusters of servers.
- **Apache Spark**: A faster alternative to MapReduce, Spark uses in-memory computing for real-time, iterative tasks, improving speed and efficiency. It has become the leading tool for distributed data processing.

## Distributed Storage

- **HDFS** and cloud storage systems like [Amazon S3](#) break data into smaller parts and distribute them across multiple servers. This setup provides high throughput, redundancy, and fault tolerance.
- **Distributed databases** such as [Cassandra](#) and [Bigtable](#) offer scalable storage for structured data, ensuring availability across nodes.

## Real-Time Processing (Data Streaming)

- Frameworks like [Apache Flink](#) and [Kafka Streams](#) are critical for real-time data processing, enabling continuous data handling as it is generated. They are commonly used in applications requiring instant processing, such as fraud detection or live analytics.

## Cloud-Native Computing

- **Cloud platforms** (e.g., AWS, Google Cloud, Azure) have made distributed computing accessible through services like [Amazon EMR](#) and [Google Dataproc](#). These services simplify the deployment and management of distributed applications.
- [Kubernetes](#) has become the standard for orchestrating distributed applications, managing containers and ensuring high availability across clusters.

## Trends and Challenges

- **Edge computing** is gaining momentum, enabling data to be processed closer to the source (e.g., IoT devices), reducing latency and bandwidth usage.
  - Challenges include **fault tolerance**, **network latency**, and **data consistency**. Innovations in consensus algorithms and fault-tolerant storage systems are working to mitigate these issues.
-

# Distribution\_Analysis.Py

---

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Preprocess/Distribution\\_Analysis.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Distribution_Analysis.py)

The goodness-of-fit results represent the **p-values** from the **Kolmogorov-Smirnov (KS) test**, which assesses how well the data fits each distribution. Here's how to interpret these values:

1. **Higher p-value** → The distribution is a better fit.
2. **Lower p-value** → The distribution is a poor fit (likely not the correct model for the data).
3. **Threshold**: A common significance level is **0.05**.
  - If  $p > 0.05$ , we do **not** reject the hypothesis that the data follows this distribution.
  - If  $p < 0.05$ , we reject the hypothesis, meaning the data **likely does not follow** that distribution.

Example using penguins.csv column "bill\_depth\_mm"

Goodness-of-fit results for bill\_depth\_mm Gaussian: 0.026308596409291618 T: 0.025906678848475195 Chi-squared: 1.4504381882536289e-15 Exponential: 2.8020502445188308e-14 Logistic: 0.05019989765502264

- **Gaussian (0.0263)** → **Poor fit** ( $p < 0.05$ ). The data likely does not follow a normal distribution.
- **T (0.0259)** → **Poor fit** ( $p < 0.05$ ). The data does not fit a t-distribution well.
- **Chi-squared (1.45e-15)** → **Very poor fit** (extremely low p-value). The data is **highly unlikely** to follow a chi-squared distribution.
- **Exponential (2.80e-14)** → **Very poor fit** (extremely low p-value). The data is **not** exponentially distributed.
- **Logistic (0.0502)** → **Acceptable fit** ( $p \approx 0.05$ ). The data could **potentially** follow a logistic distribution.

The **Logistic distribution** has the **highest p-value (0.0502)**, making it the **best candidate** among the tested distributions. However, since it's **borderline ( $\approx 0.05$ )**, you may want to visualize the distribution and compare the fits.

## Distributions

In [ML\\_Tools](#) see:

- [Distribution\\_Analysis.py](#)

### Discrete Distributions

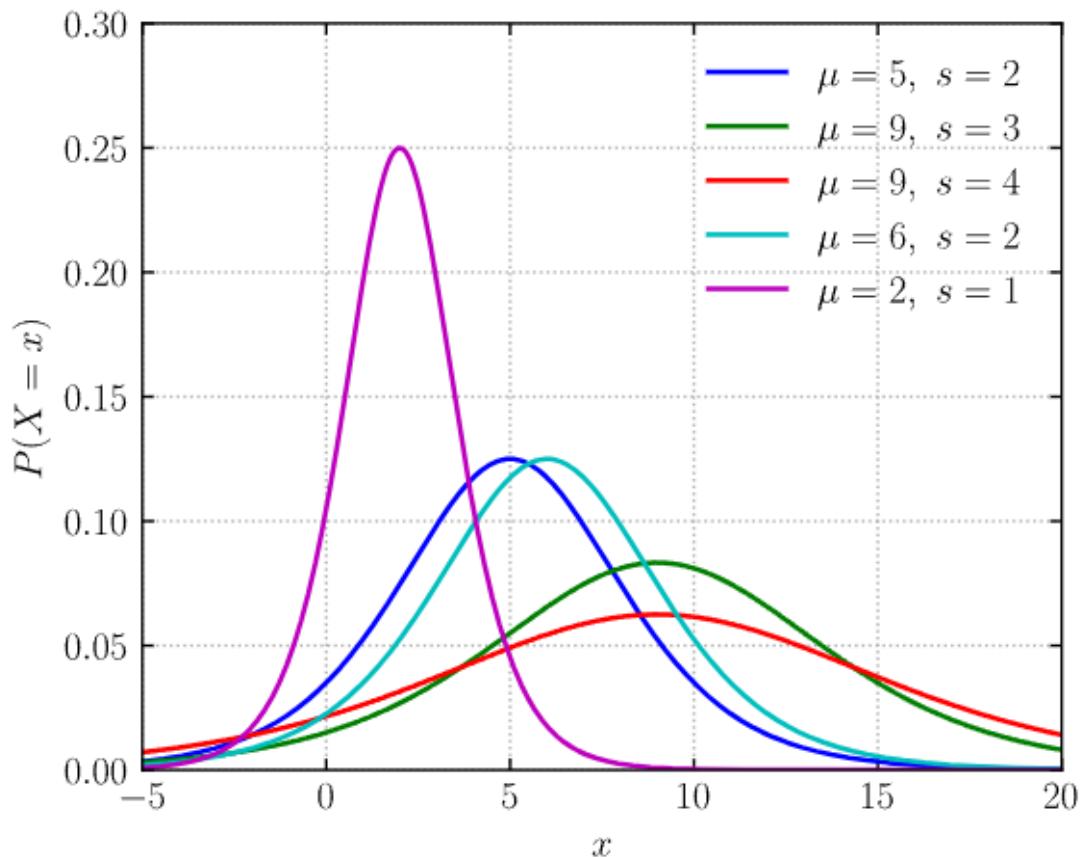
These distributions have probabilities concentrated on specific values.

- **Uniform** Distribution: All outcomes are equally likely. Example: Drawing a card from a shuffled deck. A boxplot can be meaningful if there's variation in the distribution. Since the values are discrete, the boxplot will show the range and quartiles effectively.
- **Bernoulli** Distribution: Represents two possible outcomes. Example: Coin flip (heads or tails), true/false scenarios. A bar chart or frequency plot would be better for visualizing the proportions. or rolling a dice.
- **Binomial** Distribution: Represents the number of successes in a sequence of Bernoulli trials. Example: Number of heads in 10 coin flips,
- **Poisson** Distribution: Models the frequency of events in a fixed interval. Example: Number of website visits per hour. A boxplot is suitable for this distribution, showing central tendencies, spread, and potential outliers.

### Continuous Distributions

These distributions have probabilities spread over a continuous range.

- **Gaussian Distribution:** Characterized by a bell-shaped curve, symmetric with thin tails. Example: Heights, exam scores.
- **T Distribution:** Similar to the normal distribution but with fatter tails, useful with limited data.
- **Chi-squared Distribution:** Asymmetric and non-negative, commonly used in [hypothesis testing](#).
- **Exponential Distribution:** Models the time between events. Example: Time between website traffic hits, radioactive decay.
- **Logistic Distribution:** S-shaped curve, often used in forecasting and modeling growth.



## Q-Q plots (Quantile-Quantile Plots)

A Q-Q (quantile-quantile) plot is a graphical tool used to compare the distribution of a dataset against a theoretical distribution (e.g., normal, logistic, exponential). It helps assess how well a given distribution fits the data.

How Q-Q Plots Work:

1. Sort your dataset → Compute the sample quantiles (percentiles).
2. Compute the theoretical quantiles → Take the same number of points from the theoretical distribution (e.g., normal, logistic).
3. Plot sample quantiles vs. theoretical quantiles:
  - If the points lie on a straight diagonal line, the data follows the theoretical distribution.
  - If the points deviate significantly, the data does not fit that distribution.

Interpreting a Q-Q Plot:

- Straight diagonal line → Data follows the chosen distribution.
- Curved S-shape → Data has skewness.
  - Upward curve (right tail high) → Right-skewed.
  - Downward curve (left tail high) → Left-skewed.
- Heavy tails (outliers) → Points at the ends deviate from the line.

- Light tails (thin-tailed distribution) → Points at the ends fall below the line.

## Practical Applications

Feature Distribution: Understanding the distribution of numerical/ categorical feature values across samples can provide insights into data characteristics.

- Observation: Analyze the spread and central tendency of data.
- Decision: Determine appropriate statistical methods or transformations.

## Related Notes

In [ML\\_Tools](#) see: [Feature\\_Distribution.py](#)

- [Violin plot](#)
- [Boxplot](#)

## Docker Image

A Docker image is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files. Docker images are used to create Docker containers, which are instances of these images running in an isolated environment.

Docker images are used for:

1. **Consistency:** They ensure that software runs the same way regardless of where it is deployed, whether on a developer's laptop, a test server, or in production.
2. **Portability:** Docker images can be easily shared and moved across different environments, making it easier to deploy applications.
3. **Version Control:** Images can be versioned, allowing developers to track changes and roll back to previous versions if necessary.
4. **Isolation:** Each Docker container runs in its own isolated environment, which helps in avoiding conflicts between different applications or services running on the same host.
5. **Scalability:** Docker images can be used to quickly scale applications by running multiple containers from the same image.

## Example: Running a Simple Web Server (dont do unless required)

6. **Install Docker:** First, ensure Docker is installed on your machine. You can download it from the [Docker website](#).
7. **Pull a Docker Image:** Use a pre-built Docker image from Docker Hub. For this example, we'll use the official Nginx image, which is a popular web server.

```
docker pull nginx
```

8. **Run a Docker Container:** Start a container using the Nginx image. This will run the web server.

```
docker run --name my-nginx -d -p 8080:80 nginx
```

- `--name my-nginx` : Names the container "my-nginx".
- `-d` : Runs the container in detached mode (in the background).
- `-p 8080:80` : Maps port 80 in the container to port 8080 on your host machine.

9. **Access the Web Server:** Open a web browser and go to `http://localhost:8080`. You should see the default Nginx welcome page, indicating that the web server is running inside the Docker container.

10. **List Running Containers:** Check which containers are running.

```
docker ps
```

11. **Stop the Container:** When you're done, you can stop the container.

```
docker stop my-nginx
```

12. **Remove the Container:** If you no longer need the container, you can remove it.

```
docker rm my-nginx
```

## Key Features Demonstrated:

- **Portability:** The Nginx server runs the same way on any machine with Docker installed.
- **Isolation:** The web server runs in its own environment, separate from other applications.
- **Ease of Use:** Starting and stopping services is straightforward with simple commands.
- **Resource Efficiency:** Docker containers are lightweight compared to virtual machines.

This example gives you a taste of how Docker can be used to quickly deploy and manage applications. As you become more familiar with Docker, you can explore building your own images, managing multi-container applications with Docker Compose, and more.

# Docker

Utilizes Docker images [Docker Image](#) to set up containers for consistent development and testing environments.

Containers can include necessary dependencies like Python and pip.

Tutorial:

- [The Only Docker Tutorial You Need To Get Started](#)

Docker Volumes - storing data

Docker Compose

Multi use containers

```
docker init command
```

- generated Dockerfile - image
- compose.yaml
- .dockerignore

## Tools

---

- [pdoc](#) – Auto-generate Python API documentation
- [Mermaid](#) – Create diagrams and flowcharts from text in a Markdown-like syntax

## Templates

### Project & Technical Meetings

- [Technical Design Doc Template](#) – Document system architecture, components, data flow
- [Pull Request Template](#) – Checklist and summary for PRs (code, tests, reviewers, notes)
- [Experiment Plan Template](#) – Hypothesis, variables, metrics, and analysis plan
- [Retrospective Template](#) – Guide for reviewing past sprint/project cycles

### Data & Analytics Meetings

- [Data Request Template](#) – Intake form for new analysis or data extract needs
- [One Pager Template](#) – Summarize a project, idea, or proposal on a single page
- [Meeting Notes Template](#) – Standard format for taking concise and structured notes

### Cross-functional / Stakeholder Meetings

- [One Pager Template](#) – Summarize a project, idea, or proposal on a single page
- [Meeting Notes Template](#) – Standard format for taking concise and structured notes
- [Data Request Template](#) – Intake form for new analysis or data extract needs

### Reporting & Strategic Meetings

- [Postmortem Template](#) – Structure for documenting incidents and learnings
- [Retrospective Template](#) – Guide for reviewing past sprint/project cycles
- [One Pager Template](#) – Summarize a project, idea, or proposal on a single page

### Collaborative & Feedback Sessions

- [1-on-1 Template](#) – Agenda for 1-on-1 check-ins with manager or reports
- [Feedback Template](#) – Structure for giving/receiving peer or performance feedback

## Meeting Types

### Project & Technical Meetings

- **Sprint Planning / Stand-ups (Agile):**

Define priorities, plan tasks, and report blockers on a daily/weekly basis

- **Design & Architecture Reviews:**

---

Evaluate technical designs—e.g., pipeline architecture, schemas, model design

- **Code Reviews / Pair Programming:**

Collaborative code sessions for learning and validation

- **Pipeline/Model Monitoring & Debugging:**

Diagnose failures, resolve data/model performance issues in production

## Data & Analytics Meetings

- **Exploratory Data Analysis (EDA) Discussions:**

Share insights, anomalies, or feature engineering results

- **Model Review & Evaluation:**

Present metrics, discuss trade-offs (e.g., ROC-AUC, fairness, overfitting)

- **Data Quality & Validation:**

Ensure schema consistency, missing data checks, rule-based validations

- **Experiment Review:**

Discuss A/B test results, statistical significance, and business impact

## Cross-functional / Stakeholder Meetings

- **Product or Domain Team Syncs:**

Discuss project goals, KPIs, data availability

- **Ad Hoc Analysis Requests:**

---

Clarify requirements for exploratory or one-off reporting

- **User Feedback / Data Product Reviews:**

---

Gather user input on dashboards, ML outputs, or data access tools

- **Requirements Grooming:**

Translate business requirements into data/technical specifications

## Reporting & Strategic Meetings

- **Quarterly Business Reviews / OKR Alignment:**

Evaluate progress against metrics and strategic initiatives

- **Dashboard Walkthroughs / Reporting Demos:**

Demonstrate key metrics, performance trends, and data tooling

- **Metrics Definition Meetings:**

Align teams on KPI definitions, calculation logic, and data sources

## Collaborative Initiatives

- **Knowledge Sharing / Lunch & Learns:**

Internal sessions to demo tools, share best practices, or teach concepts

- **Workshops (e.g., dbt, Airflow, MLOps):**

Hands-on learning sessions for technical upskilling

- **Data Governance / Compliance:**

---

Ensure responsible data use, privacy adherence, and lineage tracking

- **Hackathons / Innovation Days:**
- 

Time-boxed events for experimentation and cross-team collaboration

## Dropout

**Dropout** is a [Regularisation](#) technique used in [Neural network](#) training to prevent [overfitting](#). It works by randomly dropping units (neurons) during training, which helps the network to not rely too heavily on any single neuron.

### Purpose

- The main goal of dropout is to improve the generalization of the model by reducing over-reliance on specific neurons. This encourages the network to learn more robust features that are useful in different contexts.

### How It Works

- During each training iteration, a subset of neurons is randomly selected and ignored (dropped out). This means their contribution to the activation of downstream neurons is temporarily removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.

### Implementation

```
from tensorflow.keras.layers import Dropout

Add a dropout layer with a dropout rate of 0.5

The dropout rate (e.g., 0.5) specifies the fraction of neurons to drop during training. A rate of 0.5 means that half of
Dropout(0.5)
```

## Duckdb In Python

To use **DuckDB** in Python, you can follow these steps to install the DuckDB library and perform basic operations such as creating a database, running queries, and manipulating data. Here's a simple guide:

### Step 1: Install DuckDB

You can install DuckDB using pip. Open your terminal or command prompt and run:

```
pip install duckdb
```

### Example: Full Code

Here's a complete example that incorporates all the steps:

```

import duckdb

Step 1: Connect to an in-memory database
conn = duckdb.connect(database=':memory:')

Step 2: Create a table
conn.execute("""
CREATE TABLE users (
 id INTEGER,
 name VARCHAR,
 age INTEGER
)
""")

Step 3: Insert data
conn.execute("""
INSERT INTO users VALUES
(1, 'Alice', 30),
(2, 'Bob', 25),
(3, 'Charlie', 35)
""")

Step 4: Query data
result = conn.execute("SELECT * FROM users").fetchall()

Print the results
for row in result:
 print(row)

Step 5: Close the connection
conn.close()

```

## Additional Features

DuckDB also supports advanced features such as:

- **Reading from CSV and Parquet files:** You can load data directly from these formats using commands like `READ_CSV` or `READ_PARQUET`.
- **Integration with Pandas:** You can easily convert DuckDB query results to Pandas DataFrames for further analysis.

## Example of Reading from a CSV

```

Load data from a CSV file into a DuckDB table
conn.execute("CREATE TABLE my_data AS SELECT * FROM read_csv_auto('path/to/your/file.csv')")

```

# Duckdb Vs Sqlite

---

Choosing between **DuckDB** and **SQLite** for data processing in **Python** depends on your specific use case and requirements.

While **SQLite** is an excellent choice for lightweight applications, local data storage, and simple transactional workloads.

**DuckDB** shines in scenarios involving complex analytical queries, large datasets, and data science workflows.

If your primary focus is on data analysis and you need high performance for analytical tasks, DuckDB may be the better option. However, if you need a simple, lightweight database for small-scale applications, SQLite could be more appropriate.

## 1. Performance for Analytical Queries

- **DuckDB** is optimized for analytical workloads and can handle complex queries involving large datasets more efficiently than SQLite. It uses a [columnar storage](#) format, which is particularly beneficial for aggregation and analytical operations.
- **SQLite**, while fast for transactional workloads, may not perform as well with large-scale analytical queries.

## 2. In-Memory Processing

- DuckDB can operate entirely in-memory, allowing for faster data processing and query execution. This is especially useful for data science tasks where speed is critical.
- SQLite can also work in-memory, but its performance may not match that of DuckDB for analytical tasks.

## 3. Support for Complex Data Types

- DuckDB supports more complex data types and operations, such as nested data structures and advanced analytical functions, which can be advantageous for data analysis.
- SQLite has a more limited set of data types and may not support some advanced analytical functions.

## 4. Integration with Data Science Tools

- DuckDB is designed to integrate seamlessly with data science libraries like Pandas, making it easier to perform data analysis and manipulation directly within your Python workflows.
- While SQLite can also be used with Pandas, DuckDB's integration is often more straightforward for analytical tasks.

## 5. Cloud and Big Data Compatibility

- DuckDB is designed to work well with cloud-based data warehouses and big data environments, making it suitable for modern data workflows that involve large datasets stored in cloud storage.
- SQLite is more suited for lightweight applications and local data storage.

## 6. Columnar Storage Format

- DuckDB's columnar storage format allows for more efficient data compression and faster query performance on analytical workloads, as it reads only the necessary columns for a query.
- SQLite uses a row-based storage format, which can be less efficient for certain types of analytical queries.

## 7. Ease of Use for Data Transformation

- DuckDB simplifies data transformation processes, allowing you to perform transformations directly within the database after loading the data. This can streamline workflows and reduce the need for additional data processing steps.
- SQLite requires more manual handling for data transformations, especially when dealing with large datasets.

## Duckdb

**DuckDB** is an open-source analytical database management system designed for efficient data processing and analysis. It is optimized for running complex queries on large datasets and is particularly well-suited for data science and analytics tasks. Here are some key features and characteristics of DuckDB:

Resources:

- <https://duckdb.org/>
- DuckDB in python
- DuckDB vs SQLite

## Key Features

1. **In-Memory Processing:** DuckDB operates primarily in-memory, which allows for fast query execution and data manipulation.
2. **Columnar Storage** It uses a columnar storage format, which is efficient for analytical queries that often involve aggregations and scans over large datasets.
3. **SQL Support:** DuckDB supports SQL as its query language, making it accessible to users familiar with SQL syntax.
4. **Integration with Data Science Tools:** DuckDB can be easily integrated with popular data science tools and programming languages, such as Python and R, allowing for seamless data analysis workflows.
5. **Lightweight and Easy to Use:** It can be embedded in applications and does not require a separate server, making it lightweight and easy to deploy.
6. **Compatibility:** DuckDB can read from various data formats, including CSV, [Parquet](#)

## Use Cases

- Data Analysis
- Data Transformation
- Querying

## Key Takeaways:

- The dummy variable trap occurs due to [multicollinearity](#), where **one dummy variable can be perfectly predicted from others.**
- Dropping one dummy variable avoids this issue and ensures that the model has a reference category against which the other categories are compared.
- This approach leads to a well-conditioned model and allows for more interpretable regression coefficients.

## Dummy Variable Trap

The dummy variable trap refers to a scenario in which there is multicollinearity in your dataset when you create dummy variables for categorical features. Specifically, it occurs when one of the dummy variables in a set of dummy variables can be perfectly predicted by a linear combination of the others.

This situation arises when you create dummy variables for a categorical feature with  $n$  categories, leading to  $n$  binary columns. However, if you include all  $n$  dummy variables in your regression model, the model will face redundancy because knowing the values of  $n-1$  dummy variables will already give you the value of the last one (since all the categories must add up to 1 for each observation). This results in perfect multicollinearity.

## Why Do We Need to Drop One of the Dummy Variables?

In a regression model, multicollinearity can cause problems because it makes the estimation of coefficients unstable, leading to unreliable statistical inferences. Specifically, the model can't determine which of the correlated variables is truly responsible for explaining the variation in the target variable.

### Example:

Suppose you have a categorical feature `town` with three categories: `West Windsor`, `Robbinsville`, and `Princeton`. When you apply [one-hot encoding](#), you create three dummy variables:

| <code>town</code>         | <code>West Windsor</code> | <code>Robbinsville</code> | <code>Princeton</code> |
|---------------------------|---------------------------|---------------------------|------------------------|
| <code>West Windsor</code> | 1                         | 0                         | 0                      |
| <code>Robbinsville</code> | 0                         | 1                         | 0                      |
| <code>Princeton</code>    | 0                         | 0                         | 1                      |

Now, if you include all three dummy variables in a linear regression model, the columns `West Windsor`, `Robbinsville`, and `Princeton` will be perfectly correlated. For example, if the values of `West Windsor` and `Robbinsville` are both 0, then `Princeton` must be 1, and vice versa.

This creates multicollinearity because you can predict one dummy variable perfectly by knowing the others. Hence, you need to drop one of the dummy variables—usually, you drop one category, which becomes the reference group.

If you drop the `West Windsor` dummy column, your table would look like this:

| <code>town</code>         | <code>Robbinsville</code> | <code>Princeton</code> |
|---------------------------|---------------------------|------------------------|
| <code>West Windsor</code> | 0                         | 0                      |
| <code>Robbinsville</code> | 1                         | 0                      |
| <code>Princeton</code>    | 0                         | 1                      |

Now, your model will use the `West Windsor` category as the baseline. The coefficients of `Robbinsville` and `Princeton` in the regression model will indicate how much higher or lower their prices are compared to `West Windsor`.

## Dagster

---

Dagster is a [data orchestrator] focusing on data-aware scheduling that supports the whole development [Data Lifecycle Management](#) lifecycle, with integrated lineage and observability, a [declarative](#) programming model, and best-in-class testability.

Key features are:

- Manage your data assets with code
- A single pane of glass for your data platform

## Data Asset

## Data Lineage

Data lineage uncovers the [Data Lifecycle Management](#) life cycle of data. It aims to show the complete data flow from start to finish.

Data lineage is the process of understanding, recording, and visualizing data as it flows from data sources to consumption.

This includes all [Data Transformation](#) (what changed and why).

## Data Literacy

Data literacy is the ability to read, work with, analyze, and argue with data in order to extract meaningful information and make informed decisions. This skill set is crucial for employees across various levels of an organization, especially as data-driven decision-making becomes increasingly important.

Organizations should invest in data literacy training programs to empower their employees with the necessary skills to effectively engage with data. A data-literate employee can read charts, draw correct conclusions, recognize when data is being used inappropriately or misleadingly, and gain a deeper understanding of the business domain. This enables them to communicate more effectively using a common language of data, spot unexpected operational issues, identify root causes, and prevent poor decision-making due to data misinterpretation.

Examples of data literacy in action include:

- Implementing the Adoptive Framework to create a Data Literacy Program.
- Employees working with spreadsheets to understand the rationale behind data-driven decisions and advocating for alternative courses of action.
- Work teams identifying areas where data needs clarification for a project.

By nurturing a data-literate workforce, businesses can improve their ability to make informed decisions, drive innovation, and achieve better outcomes.

## Dbt

Data build tool is an open-source framework designed for [Data Transformation](#) within a modern data stack.

It enables analysts and engineers to transform, model, and manage data using [SQL](#) while [adhering to software engineering best practices](#) like version control, testing, and [Documentation & Meetings](#).

---

## Key Concepts of dbt:

1. **SQL-based Transformation:** dbt allows users to write SQL queries to define transformations and models, making it accessible for analysts who are already familiar with SQL. It doesn't handle extraction or loading of data, but focuses purely on transforming data that is already in a data warehouse.
2. **Modular and Reusable Models:** dbt encourages the creation of modular, [reusable SQL "models."](#) Each model represents a transformation, and these models can be built on top of each other. A model is essentially a SQL query stored as a `.sql` file that dbt uses to transform raw data into a refined dataset. Models are run in sequence, with dbt handling dependencies between models.
3. **Version Control and Collaboration:** dbt integrates with Git for version control, making it easy for teams to collaborate, track changes, and roll back to previous versions if needed. This promotes transparency and accountability within the data team.
4. **Testing:** dbt allows users to write and run tests to ensure data integrity and consistency. You can define tests for specific models or fields, like checking for non-null values or ensuring data uniqueness.
5. **Documentation:** dbt auto-generates documentation from your models, providing a clear overview of your data transformations, lineage, and dependencies. You can also add descriptions for models and fields to improve the clarity of your data pipelines.
6. **Data Lineage:** dbt automatically tracks the lineage of your data by mapping dependencies between models. This makes it easy to understand how data flows through the pipeline and where any upstream or downstream issues might originate.
7. **Extensibility:** dbt has a plugin architecture that allows users to extend functionality. For example, there are adapters for popular data warehouses like Snowflake, BigQuery, Redshift, and others, making dbt highly flexible in different data stack environments.
8. **Cloud and Core Versions:**
  - o **dbt Core** is the open-source version that you run locally or in your cloud infrastructure.
  - o **dbt Cloud** is a fully managed service that adds features like scheduling, logging, and a web-based IDE for dbt workflows.

## Workflow with dbt:

1. **Data Loading:** First, data is loaded into a data warehouse from various sources using ELT tools (e.g., Fivetran, Stitch).
  2. **Transform with dbt:** Using dbt, you write SQL models to clean, transform, and aggregate the raw data into useful, analytical datasets.
  3. **Build Data Models:** You organize your models into layers, often referred to as staging, intermediate, and final models.
  4. **Testing and Documentation:** Run tests to validate data, generate lineage diagrams, and create documentation.
  5. **Deploy:** Schedule or trigger dbt jobs to run in production environments, ensuring consistent and accurate data transformations.
-

## Example of a dbt Model:

```
-- models/staging_orders.sql
WITH raw_orders AS (
 SELECT * FROM {{ ref('raw_orders_data') }}
)
SELECT
 order_id,
 customer_id,
 order_date,
 amount
FROM raw_orders
WHERE order_status = 'completed';
```

In this model:

- `ref('raw_orders_data')` is referencing another model that contains raw order data.
- The model selects and transforms only the completed orders.

## Benefits of Using dbt:

- 1. Analyst Empowerment:** dbt empowers data analysts to own the transformation process using SQL, reducing dependency on data engineers for transformations.
- 2. Version Control and Testing:** Built-in version control and testing improve data reliability and reduce risks of errors in production.
- 3. Modularity and Scalability:** The modular nature of dbt models makes it easier to scale transformations and manage complex pipelines.
- 4. Transparency and Documentation:** dbt creates clear documentation and lineage automatically, improving visibility across teams.

## Tools Integrating with dbt:

- **Data Warehouses:** Redshift, Snowflake, BigQuery, Postgres.
- **ELT Tools:** Stitch, Fivetran, Airbyte (for the extraction and loading phase).
- **Version Control:** GitHub, GitLab, Bitbucket (for managing dbt code).

## Resources:

<https://www.getdbt.com/blog/what-exactly-is-dbt>

dbt Tags: #data\_transformation, #data\_tools

## Declarative

In a **declarative data pipeline**, the focus is on *what* needs to be achieved, not *how* it should be executed. You define the desired outcome or the data products, and the system takes care of the underlying execution details, such as the order in which tasks are performed. This is in contrast to an **imperative** pipeline, where the developer explicitly specifies the steps and the order in which they should be executed. Here's a breakdown of the key aspects:

## Declarative Programming:

- Focuses on **what** needs to be done.
- Describes the desired state or result without dictating the control flow or step-by-step process.
- In a data pipeline context, a declarative approach might involve specifying the desired data products and letting the system optimize how and when different parts of the pipeline are executed.
- Example: **SQL** is often considered declarative because you specify the result you want (e.g., the output of a query) without explicitly stating the steps for how the database engine should retrieve it.

## Imperative Programming:

- Focuses on **how** tasks should be done.
- Specifies the **control flow** explicitly, dictating the exact steps to be performed and the order of operations.
- In a data pipeline, this would involve writing scripts that detail each step in the transformation and loading process in the sequence they must be executed.
- Example: A series of Python scripts that process data in a specific sequence.

## Advantages of Declarative Pipelines:

1. **Easier to Debug:** Since the desired state is clearly defined, it is easier to identify discrepancies between the intended outcome and the current state. This can help pinpoint issues in the pipeline.
2. **Automation:** Declarative systems often enable better automation since the system has the flexibility to determine the most efficient way to achieve the defined goals.
3. **Simplicity and Intent:** Declarative approaches focus on the **intent** of the program, making it easier for others to understand what the program is supposed to do without having to dive into implementation details.
4. **Reactivity:** The pipeline can automatically adjust when inputs or dependencies change. For example, if certain data dependencies change, the system can rerun the necessary parts of the pipeline to maintain consistency.

## Example in Data Engineering:

A declarative approach to data engineering would involve **Functional Data Engineering** principles. This involves treating data as immutable and focusing on defining the desired transformations and outputs in a declarative manner. Instead of writing imperative scripts for each data transformation step, you'd define the desired outputs, and the system would optimize the execution.

## Use Cases:

Declarative pipelines are particularly useful in **data lineage**, **Data Observability**, and **Data Quality** monitoring\*. By defining what\* data products should exist and what their properties should be, it's easier to track changes and ensure the consistency and quality of data. It also makes systems more resilient to changes, as the declarative nature enables the system to adjust the execution order or method dynamically, based on current conditions.

## Dependency Manager

[Virtual environments](#)

[requirements.txt](#)

[TOML](#)



**E**

## Table of Contents

- EDA
- EDA\_Pandas.py
- ELT
- ETL Pipeline Example
- ETL vs ELT
- ETL
- Edge Machine Learning Models
- Education and Training
- Elastic Net
- Embedded Methods
- Encoding Categorical Variables
- Energy ABM
- Energy Storage
- Energy
- Environment Variables
- Epoch
- Epub
- Estimator
- EtLT
- Evaluating Language Models
- Evaluation Metrics
- Event Driven Events
- Event Driven Microservices
- Event Driven
- Event-Driven Architecture
- Everything
- Excel & Sheets
- Explain different gradient descent algorithms, their advantages, and limitations.
- Explain the curse of dimensionality
- Exploration vs. Exploitation
- Exploration
- embeddings for OOV words
- emergent behavior

## Eda

Exploratory Data Analysis (EDA) is an approach to analyzing datasets to summarize their main characteristics, often utilizing visual methods. EDA helps users to:

- Understand the Data's Structure: Gain insights into the organization and format of the data.
- Detect Patterns: Identify trends and patterns within the data.
- Decide on Statistical Techniques: Choose appropriate statistical methods by examining distributions and correlation.
- Select Variables: Determine which variables to include in further analysis.
- Handle Data Quality: Address issues related to data quality and integrity.

- Spot Anomalies and [standardised/Outliers](#): Identify unusual data points that may affect analysis.
  - Generate and Test Hypotheses: Formulate hypotheses and validate them using statistical methods.
  - Check Assumptions: Verify assumptions through statistical summaries and graphical representations.
- 

## Common Techniques Used in EDA

- Descriptive Statistics: Calculating measures such as mean, median, mode, [standard deviation](#), and percentiles to summarize data.
- [Data Visualisation](#): Using plots and charts like histograms, box plots, scatter plots, and bar charts to visually explore data.
- Correlation Analysis: Assessing relationships between variables using [correlation](#) coefficients and scatter plots.
- [Data Transformation](#): Applying transformations to data, such as normalization or log transformation, to better understand its characteristics.

## Implementation

In [ML\\_Tools](#) see:

- [EDA\\_Pandas.py](#)

## Eda\_Pandas.Py

## Elt

**ELT** (Extract, Load, Transform) is a data integration approach that involves three main steps:

1. **Extract (E)**: Data is extracted from a source system.
2. **Load (L)**: The raw data is loaded into a destination system, such as a data warehouse.
3. **Transform (T)**: Transformation of the data occurs within the destination system after the data has been loaded.

This approach contrasts with the traditional **ETL** (Extract, Transform, Load) method, where data is transformed before reaching the destination. For a detailed comparison, see [ETL vs ELT](#)

## Advantages of ELT

The shift from ETL to ELT has been facilitated by several factors:

- **Cost Efficiency**: The decreasing costs of cloud-based storage and computation have reduced the advantages of ETL's pre-loading data transformation.
- **Cloud-Based Data Warehouses**: The emergence of cloud-based data warehouses like Redshift, [BigQuery](#), and Snowflake has made the ELT approach more feasible and efficient.

## Historical Context

Historically, ETL was preferred for reasons that are now less relevant:

- **Cost Savings**: ETL was believed to save costs by filtering out unwanted data before loading. However, this is less significant with modern cloud solutions.
  - **Complexity Management**: ETL minimizes the complexity of post-loading transformations. Yet, contemporary tools like [dbt](#) simplify this process, making it easier to perform transformations after loading.
-

ELT Tags: #data\_transformation, #data\_integration

---

# Etl Pipeline Example

[Link](#)

[Github]([https://github.com/syalanuj/youtube/blob/main/de\\_fundamentals\\_python/etl.py](https://github.com/syalanuj/youtube/blob/main/de_fundamentals_python/etl.py))

## 1. Extract using a API

Get data via api or download.

## 2. Transform

Put into a pandas df.

## 3. Load

Save df as a database. Save SQLite database. Save as table.

Run functions

```

"""
Python Extract Transform Load Example
"""

%%

import requests
import pandas as pd
from sqlalchemy import create_engine

def extract()-> dict:
 """
 This API extracts data from
 http://universities.hipolabs.com
 """
 API_URL = "http://universities.hipolabs.com/search?country=United+States"
 data = requests.get(API_URL).json()
 return data

def transform(data:dict) -> pd.DataFrame:
 """
 Transforms the dataset into desired structure and filters"""
 df = pd.DataFrame(data)
 print(f"Total Number of universities from API {len(data)}")
 df = df[df["name"].str.contains("California")]
 print(f"Number of universities in California {len(df)}")
 df['domains'] = [','.join(map(str, l)) for l in df['domains']]
 df['web_pages'] = [','.join(map(str, l)) for l in df['web_pages']]
 df = df.reset_index(drop=True)
 return df[["domains", "country", "web_pages", "name"]](#domainscountryweb_pagesname)

def load(df:pd.DataFrame)-> None:
 """
 Loads data into a sqlite database"""
 disk_engine = create_engine('sqlite:///my_lite_store.db')
 df.to_sql('cal_uni', disk_engine, if_exists='replace')

%%
data = extract()
df = transform(data)
load(df)

%%

```

## Etl Vs Elt

**ETL** (Extract, Transform, and Load) and **ELT** (Extract, Load, and Transform) are two paradigms for moving data from one system to another.

**ELT is most friendly for analysts**

The main difference between them is that when an ETL approach is used, data is transformed before it is loaded into a destination system. On the other hand, in the case of ELT, any required transformations are done after the data has been written to the destination and are *then done inside* the destination -- often by executing SQL commands. The difference between these approaches is easier to understand by a visual comparison of the two approaches.

The image below demonstrates the ETL approach to [data integration](#):

While the following image demonstrates the ELT approach to data integration:

ETL was originally used for [Data Warehousing](#) and ELT for creating a [Data Lake](#).

## Disadvantages of ETL compared to ELT

ETL has several **disadvantages compared to ELT**, including the following:

- Generally, only transformed data is stored in the destination system, and so **analysts must know beforehand every way** they are going to use the data, and every report they are going to produce.
- Modifications to requirements can be costly, and often require re-ingesting data from source systems.
- Every transformation that is performed on the data may obscure some of the underlying information, and analysts only see what was kept during the transformation phase.
- Building an ETL-based data pipeline is often beyond the technical capabilities of analysts.

## Etl

**ETL** (Extract, Transform, Load) is a data integration process that involves moving data from one system to another. It consists of three main stages:

1. **Extract**: Collecting data from various sources, such as databases, APIs, or flat files. This may involve setting up API connections to pull data from multiple sources.
2. **Transform**: Cleaning and converting the data into a usable format. This includes filtering, aggregating, and joining data to create a unified dataset. See [Data Transformation](#).
3. **Load**: Inserting the transformed data into a destination system, such as a data warehouse (organized), database, or data lake (unorganized).

## Historical Context

The ETL paradigm emerged in the 1970s and was traditionally preferred for its ability to transform data before it reaches the destination. This approach ensures that data is standardized and passes quality checks, enhancing overall data quality.

## Transition to ELT

In recent years, the data movement paradigm has shifted towards **ELT** (Extract, Load, Transform). This approach emphasizes keeping raw data accessible in the destination system, allowing for more flexibility in data processing. For a comparison of ETL and ELT, see [ETL vs ELT](#).

Reasons for Change to see [ELT](#)

## Modern ETL Tools

Current ETL processes are often managed using tools like [Apache Airflow](#), [dagster](#), and [Temporal](#).

## Enhancing the ETL Process

To improve an ETL process, consider the following enhancements:

- **Error Handling:** Implement error handling to manage exceptions and prevent silent failures.
- **Logging:** Include logging to track the process flow and facilitate debugging.
- **Parameterization:** Make scripts flexible by parameterizing file paths and database connections.
- **Data Validation and Cleaning:** Incorporate steps to validate and clean the data.
- **Database Indexing and Constraints:** Optimize database tables with proper indexing and constraints for better performance.

### Related Notes

- [ETL Pipeline Example](#)
- [ETL vs ELT](#)

**ETL Tags:** #data\_transformation, #data\_integration

## Edge Machine Learning Models

**Edge ML** refers to deploying machine learning models directly on edge devices, such as IoT sensors, smartphones, or embedded systems, instead of relying on cloud-based processing. This is crucial in scenarios requiring low-latency, real-time decision-making, or environments with limited connectivity.

### Key Characteristics of Edge ML Models:

#### 1. Low Latency

- Models running at the edge can make real-time decisions with minimal delay. This is critical in applications like autonomous vehicles, industrial automation, or real-time health monitoring, where delays can have serious consequences.

#### 2. Reduced Bandwidth Usage

- By processing data locally, edge ML models reduce the need to send large amounts of data to the cloud for analysis. This is particularly valuable in environments with limited or expensive connectivity (e.g., remote locations or bandwidth-constrained networks).

#### 3. Privacy Preservation

- Processing sensitive data on-device, instead of sending it to the cloud, enhances privacy and reduces the risk of data breaches. This is important in healthcare, financial services, or any scenario involving personal data.

#### 4. Energy Efficiency

- Edge devices often have limited power resources. As a result, models deployed at the edge need to be optimized for low energy consumption, ensuring they can operate for extended periods without requiring frequent battery replacements or recharging.

### Common Applications of Edge ML:

#### 1. Autonomous Systems (e.g., Drones, Robots, Vehicles)

- Autonomous systems rely on real-time decision-making for navigation, obstacle detection, and control. Edge ML allows these systems to react instantaneously to their surroundings without depending on external servers.

## 2. Smart Cities and Industrial IoT

- Edge ML powers applications such as **traffic monitoring**, **environmental sensing**, and **predictive maintenance** in smart factories. For example, sensors in factories can use edge models to predict equipment failure before it occurs, ensuring smooth operations without cloud reliance.

## Challenges in Edge ML:

### 1. Model Compression

- Since edge devices often have limited storage and computational power, ML models need to be compressed or optimized (e.g., using techniques like quantization, pruning, or knowledge distillation) to run efficiently while maintaining accuracy.

### 2. On-Device Model Updates

- Keeping models updated without frequent cloud interactions is a challenge. Edge devices need mechanisms for **incremental learning** or efficient updates without disrupting normal operations.

## Popular Frameworks for Edge ML:

3. **TensorFlow Lite**: A lightweight version of TensorFlow, designed to run on mobile and embedded devices.
4. **PyTorch Mobile**: PyTorch's framework for deploying ML models on mobile devices.
5. **ONNX Runtime**: Optimized for running machine learning models on various platforms, including edge devices.
6. **Edge Impulse**: A platform specifically for building ML models for edge devices, particularly for IoT applications.

Edge ML is driving innovation in industries requiring decentralized, real-time intelligence, enabling devices to make smart decisions locally while minimizing reliance on cloud resources.

# Education And Training

## Adaptive Learning Systems

- **Overview**: Adaptive learning systems use technology to tailor educational experiences to individual student needs. RL is instrumental in personalizing these systems.
- **Applications**:
  - **Personalized Learning Paths**: RL algorithms can create customized learning paths for students based on their performance, preferences, and engagement levels, adapting content delivery in real-time.
  - **Feedback and Assessment**: Adaptive systems can provide immediate feedback based on student responses, reinforcing concepts through targeted exercises and adjusting difficulty levels as needed.
  - **Engagement Strategies**: By analyzing student interactions, RL can suggest motivational strategies, such as gamification elements or timely reminders, to keep students engaged and motivated.

# Elastic Net

This method combines both L1 ([Lasso](#)) and L2 ([Ridge](#)) regularization by adding both absolute and squared penalties to the loss function. It strikes a balance between Ridge and Lasso.

It is particularly useful when you have high-dimensional datasets with highly correlated features.

The Elastic Net loss function is:

$$\text{Loss} = \text{MSE} + \lambda_1 \sum_{i=1}^n |w_i| + \lambda_2 \sum_{i=1}^n w_i^2$$

where  $\lambda_1$  controls the L1 regularization and  $\lambda_2$  controls the L2 regularization.

## Code

```
from sklearn.linear_model import ElasticNet

Initialize an Elastic Net model
model = ElasticNet(alpha=0.1, l1_ratio=0.5) # l1_ratio controls the L1/L2 mix
model.fit(X_train, y_train)
```

# Embedded Methods

Embedded methods for [Feature Selection](#) integrate feature selection directly into the model training process.

Embedded methods provide a convenient and efficient approach to feature selection by seamlessly integrating it into the model training process, ultimately leading to models that are more parsimonious and potentially more interpretable.

1. **Incorporated into Model Training:** Unlike [Filter method](#) and [Wrapper Methods](#), which involve feature selection as a separate step from model training, embedded methods perform feature selection simultaneously with model training. This means that feature importance or relevance is determined within the context of the model itself.
2. **Regularization Techniques:** Embedded methods commonly use [Regularisation](#) techniques to penalize the inclusion of unnecessary features during model training.
3. **Automatic Feature Selection:** Embedded methods automatically select the most relevant features by learning feature importance during the training process. The model adjusts the importance of features iteratively based on their contribution to minimizing the [Loss function](#).

### 4. Examples of Embedded Methods:

- o [Lasso \(L1 Regularization\)](#):
- o [Elastic Net](#): Elastic Net combines L1 ([Lasso](#)) and L2 ([Ridge](#)) regularization .
- o **Tree-based Methods**: [Decision Tree](#) and ensemble methods like [Random Forests](#) and [Gradient Boosting](#) inherently perform feature selection during training by selecting the most informative features at each split node of the tree.

### 5. Advantages:

- o Simplicity: Embedded methods simplify the feature selection process by integrating it into model training, reducing the need for additional preprocessing steps.
- o Efficiency: Because feature selection is performed during model training, embedded methods can be more computationally efficient compared to wrapper methods, which require training multiple models.

### 6. Considerations:

- o Hyperparameter Tuning: Tuning regularization parameters or other model-specific parameters may be necessary to optimize feature selection performance.

- o Model **interpretability**: While embedded methods can automatically select features, interpreting the resulting model may be challenging, especially for complex models like ensemble methods.

# Encoding Categorical Variables

## Overview

Categorical variables need to be converted into numerical representations to be used in models, particularly in [Regression](#) analysis. This process is essential for transforming categorical results into a format that algorithms can interpret.

## Label Encoding

This method assigns a unique integer to each category in the variable.

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
var1_cat = df['var1'] # Replace df with your DataFrame
var1_encoded = label_encoder.fit_transform(var1_cat)
```

For example, if `df[col]` contains the categories `['apple', 'banana', 'orange']`, the `LabelEncoder` would transform them into `[0, 1, 2]`.

However, keep in mind that this encoding can imply an order or hierarchy in the data, which might not be intended. In some cases, you might want to use `OneHotEncoder` instead, which creates a binary vector for each category.{}{}

Given a term in the df you can transform it without needing to look up its value.

```
company="google"
company_n = LabelEncoder().transform([company])
```

## One-Hot Encoding

This technique creates a binary column for each category, allowing the model to treat each category as a separate feature.

```
from sklearn.preprocessing import OneHotEncoder

binary_encoder = OneHotEncoder(categories='auto')
var1_1hot = binary_encoder.fit_transform(var1_encoded.reshape(-1, 1))
var1_1hot_mat = var1_1hot.toarray()
var1_DF = pd.DataFrame(var1_1hot_mat, columns=['cat1', 'cat2', 'cat3']) # Adjust column names as needed
var1_DF.head()
```

Understanding OneHotEncoder:

The `OneHotEncoder` from `sklearn.preprocessing` is used to convert categorical integer values into a format that can be provided to machine learning algorithms to do a better job in prediction. It creates a binary column for each category and returns a sparse matrix or dense array.

## Converting All Categorical Variables to Dummies

To convert all categorical variables in a DataFrame to dummy variables, you can use the following loop:

```
for col in df.columns:
 if df[col].dtype == 'object':
 dummies = pd.get_dummies(df[col], drop_first=False)
 dummies = dummies.add_prefix(f'{col}_')
 df.drop(col, axis=1, inplace=True)
 df = df.join(dummies)
```

**Dummy Variable Trap:** When using one-hot encoding, it's important to avoid the **dummy variable trap**, which occurs when one category can be perfectly predicted from the others. To prevent this, you can drop one of the dummy variables, as one column is sufficient to represent a binary choice (0 or 1).

## Alternative Encoding Method

Another way to encode categorical variables is by mapping them directly to integers:

```
dataset['var1'] = dataset['var1'].map({'A': 0, 'B': 1, 'C': 2}).astype(int)
```

## Related Topics

- **Regression:** Understanding how regression models utilize encoded variables.
- **Feature Engineering:** Techniques to enhance model performance through better feature representation.

## Overview

- Categorical variables need to be converted into numerical representations for use in models. This is essential for transforming categorical data into a format that algorithms can interpret.

## Methods

- Label Encoding: Assigns a unique integer to each category.
- One-Hot Encoding: Creates a binary column for each category, allowing the model to treat each category as a separate feature.

## Example Code

```

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import pandas as pd

Label Encoding
label_encoder = LabelEncoder()
var1_encoded = label_encoder.fit_transform(df['var1'])

One-Hot Encoding
binary_encoder = OneHotEncoder(categories='auto')
var1_1hot = binary_encoder.fit_transform(var1_encoded.reshape(-1, 1))
var1_1hot_mat = var1_1hot.toarray()
var1_DF = pd.DataFrame(var1_1hot_mat, columns=['cat1', 'cat2', 'cat3'])

```

## Energy ABM

- **Complex Systems Understanding:** Energy systems involve numerous stakeholders (producers, consumers, regulators) with diverse interests and behaviors. [Agent-Based Modelling|ABM](#) helps capture this complexity, providing a clearer picture of system dynamics.
- **Adaptive Behavior:** Agents in ABM can adapt their behavior based on interactions, mirroring how consumers and producers might respond to incentives or changes in the market.
- **Scenario Analysis:** ABM allows for "what-if" analyses, enabling stakeholders to explore different scenarios, such as the impact of implementing new technologies or policies on energy systems.
- **Data-Driven Insights:** With the rise of smart meters and IoT devices, ABM can leverage real-time data to improve model accuracy and relevancy, enhancing decision-making processes.

## Energy Storage

### Energy Storage

Battery farms exist.

Stored energy can be traded.

Stored energy can be stored using distributed system such as EV cars.

## Energy

Areas of interest:

- [Smart Grids](#)
- [Energy Storage](#)
- [Demand forecasting](#)
- [Network Design](#)
- [Energy ABM](#)

Questions:

---

## Introduction

- [How to model to improve demand forecasting](#)
  - What patterns can be identified in consumer behavior data to inform energy pricing strategies?
  - How can predictive maintenance be implemented using data from smart sensors in energy infrastructure?
- 

### Techniques:

- **Differential Equations**: Used to model dynamic systems in energy generation and consumption. For example, they can describe the behavior of power systems over time or the thermal dynamics of energy storage systems.
- **Stochastic Modeling**: Involves random variables to model uncertainties in energy production (e.g., variability in solar or wind energy) and consumption.
- **Agent-Based Modelling**: Simulates interactions of agents (consumers, producers, regulators) to understand complex systems and emergent phenomena in energy markets.
- **Time Series Analysis**: Analyzing historical data to forecast future energy demand or production trends.
- **Regression Analysis**: Used to model relationships between different variables, such as energy prices and consumption patterns.
- **Neural network|Neural Network**: Particularly deep learning, is applied for complex pattern recognition in large datasets, such as detecting anomalies in energy consumption or predicting equipment failures.

Dynamic pricing, incentivised load management, local generation

Use green energy if on grid

## Environment Variables

Solution 1: Set Environment Variables Permanently (Recommended) This ensures that environment variables persist across sessions.

On Windows (Permanent) Open Control Panel → System → Advanced system settings → Environment Variables.

Under System Variables, click New.

Variable Name: PG\_USER

Variable Value: postgres

Click New again.

Variable Name: PG\_PASSWORD

Variable Value: your\_password

Click OK and restart your computer.

Once restarted, Jupyter Notebook should be able to access the variables.

## Epoch

An epoch in machine learning is a single pass through the entire training dataset. The number of epochs, denoted as  $N$ , determines how many times the data is applied to the model.

Why Use Multiple Epochs?

- **Repetition for Learning**: The data is applied to the model  $N$  times to improve learning and accuracy. For example, if  $N = 10$ , the model will see the entire dataset 10 times.
-

## Example



```
Epoch 1/10
6250/6250 [<mark>=</mark><mark>=</mark><mark>=</mark><mark>=</mark><mark>=</mark>] - 6s 910us/step - loss: 0
```

- **Epoch 1/10:** Indicates the model is currently on the first epoch out of a total of 10.
- **Batches:** For efficiency, the dataset is divided into smaller groups called 'batches'. In TensorFlow, the default batch size is 32. With 200,000 examples, this results in 6,250 batches.
- **Batch Execution:** The notation `6250/6250` shows the progress of batch execution within the current epoch.

# Epub

An **EPUB** (short for *electronic publication*) file is a widely used **open eBook format** that is designed for **reflowable content**, meaning it can adapt its layout to fit various screen sizes—unlike PDFs, which preserve a fixed layout.

## Key Features of EPUB

- **Reflowable Text:** The content adjusts to screen size, font preferences, and orientation. This is ideal for smartphones, tablets, and e-readers like Kobo or Apple Books.
- **HTML + CSS Based:** Internally, an EPUB file is a compressed archive (`.zip`) that contains HTML files, images, stylesheets, metadata, and a manifest.
- **Navigation:** It supports **table of contents**, **internal links**, and **chapters** for easy navigation.
- **Supports Rich Media:** EPUB 3 can include audio, video, interactive elements, and MathML.

## How EPUB Shows “Pages”

EPUB doesn't have fixed "pages" like PDF. Instead:

- The **reading software** (like Apple Books, Calibre, or Kobo) dynamically **splits content into pages** based on screen size, font size, and user settings.
- Pages can vary in number depending on:
  - Device screen resolution
  - Font size or style
  - Margin settings

Because of this, you can't refer to a fixed page number universally across devices.

## EPUB vs PDF

| Feature                    | EPUB                            | PDF                              |
|----------------------------|---------------------------------|----------------------------------|
| Layout                     | Reflowable                      | Fixed                            |
| Usability on small screens | Excellent                       | Poor                             |
| Internal format            | HTML + CSS + XML                | PostScript-based (binary)        |
| Navigation                 | Flexible (TOC, links, metadata) | Static (can have TOC, but fixed) |

## Estimator

Given a sample an estimator is a formula that approximates a population parameter i.e feature

## Etlt

[!important] EtlT refers to Extract, “tweak”, Load, [Transform](#), and can be thought of an extension to the [ELT](#) approach to [data integration](#).

When compared to ELT, the EtlT approach incorporates an additional light “tweak” (small “t”) transformation, which is done on the data after it is extracted from the source and before it is loaded into the destination.

## Evaluating Language Models

The [LMSYS](#) Chatbot Arena is a platform where various large language models ([LLMs](#)), including versions of GPT and other prominent models like LLaMA or Claude, are compared through side-by-side interactions. The performance of these models is evaluated using human feedback based on the quality of their generated responses.

The arena employs several techniques to rank and compare models:

1. [Elo Rating System](#): Adapted from chess, this system rates models based on their relative performance in head-to-head competitions. When one model's response is preferred over another's, the winning model gains points while the losing model loses points. The rating difference helps determine the strength of models in future predictions. The system adjusts ratings gradually to avoid bias towards more recent results, ensuring stability over time.
2. [Bradley-Terry Model](#): This model goes beyond simple win-loss records by taking into account the [difficulty of the task](#) and the models' relative strengths. It helps fine-tune the ranking, especially when one model consistently performs better against tougher tasks.

In addition to these ranking systems, users can directly compare LLMs by giving them [Prompting](#) to handle, such as writing articles, answering questions, or performing translations. Human voters then decide which model's output is better, or they can declare a tie if neither response is satisfactory.

These methods ensure continuous improvement of the rankings, providing a transparent and evolving leaderboard of the best generative models, including GPT versions

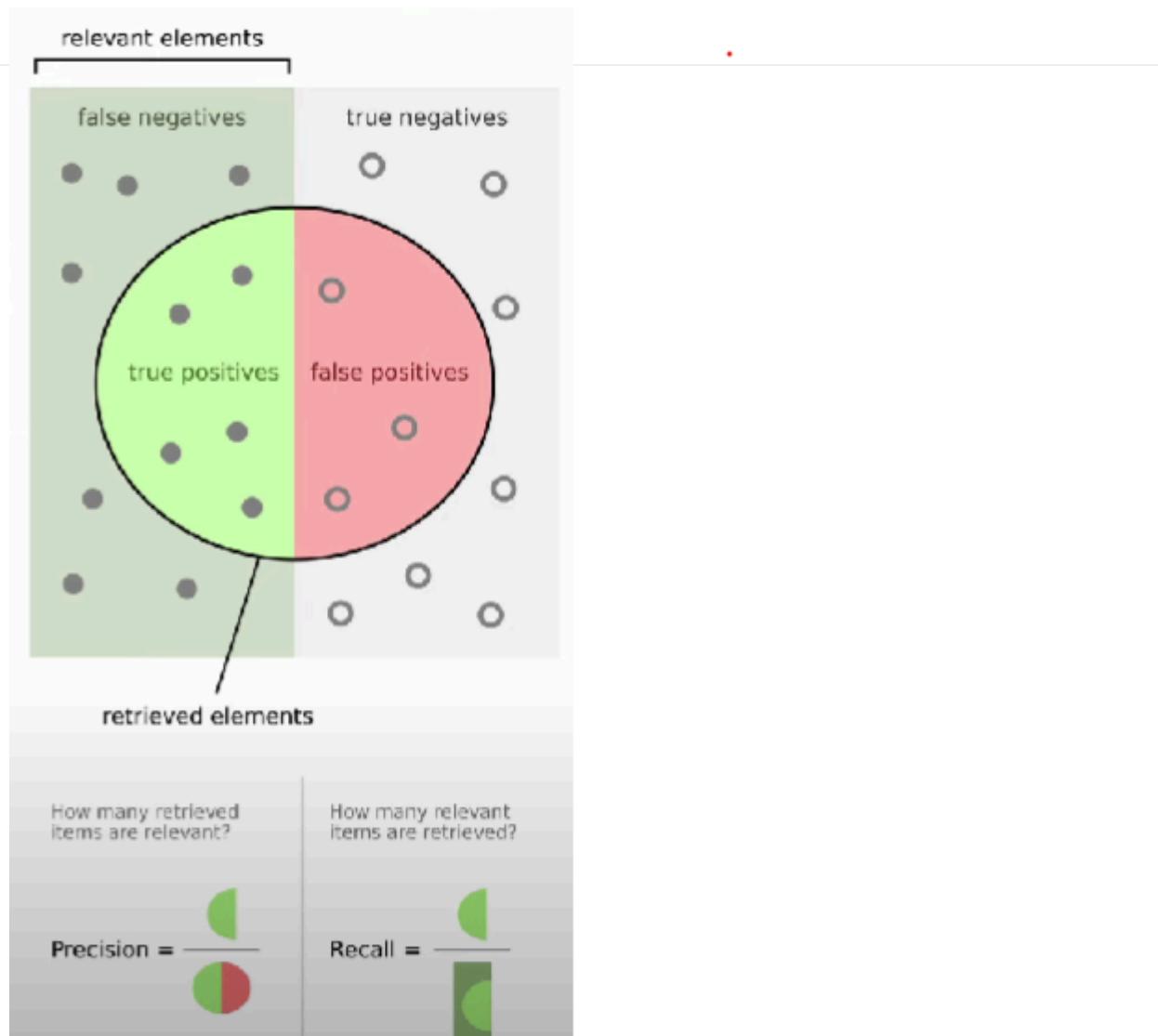
<https://openlm.ai/chatbot-area/>

<https://www.analyticsvidhya.com/blog/2024/05/from-gpt-4-to-llama-3-lmsys-chatbot-area-ranks-top-langs/>

## Evaluation Metrics

### Description

Confusion Matrix Accuracy Precision Recall Precision or Recall F1 Score Recall Specificity



## Resources:

[Link to good website describing these](#)

In `ML_Tools` see: [Evaluation\\_Metrics.py](#)

## Types of predictions

Types of predictions in evaluating models. Also see [Why Type 1 and Type 2 matter](#)

### True Positive (TP):

- This occurs when the model correctly predicts the positive class. For example, if the model predicts that an email is spam and it actually is spam, that's a true positive.

### False Positive (FP):

- Also known as a "Type I error," this occurs when the model incorrectly predicts the positive class. For example, if the model predicts that an email is spam but it is not, that's a false positive.

### True Negative (TN):

- This occurs when the model correctly predicts the negative class. For example, if the model predicts that an email is not spam and it actually is not spam, that's a true negative.

**False Negative (FN):**

- Also known as a "Type II error," this occurs when the model incorrectly predicts the negative class. For example, if the model predicts that an email is not spam but it actually is spam, that's a false negative.

## Evaluation metrics in practice

Having many evaluation metrics is hard to understand and optimise. Sometimes it is best to combine into one.

Use a single number i.e. accuracy or [F1 Score](#).

This speeds up development of ml projects.

In order to use metrics to evaluate a model we can:

- Can combine multiple metrics a formula, i.e. weighted average.
- If there is a metrics we are happy that the model passes a given level then we can have it "Satisfying". So the for the given metric it just needs to pass a given level.
- For metrics we are interested in we have it "Optimising", the one we want to be the best.
- Setup: Pick N-1 satisfying and 1 optimising.

## Another cat classification example

| Classifier | Accuracy | Running time |
|------------|----------|--------------|
| A          | 90%      | 80ms         |
| B          | 92%      | 95ms         |
| C          | 95%      | 1,500ms      |

$$\text{Cost} = \underline{\text{accuracy}} - 0.5 \times \underline{\text{runningTime}}$$

Maximize Accuracy  
Subject to RunningTime  $\leq$  100ms.

N metrics : 1 optimizing  
N-1 satisfying

## Event Driven Events

Events can be stored in a [Data Lake](#) and analysed to find patterns/predictions.

Event Driven Microservices allow for [Business observability](#)

[Monolith Architecture](#)

[Event Driven Microservices](#)

[API Driven Microservices](#)

## Event Driven Microservices

Event-driven microservices refer to a [software architecture](#) pattern where [microservices](#) communicate and coordinate their actions through the production, detection, consumption, and reaction to [Event Driven Events](#). This approach is designed to create a more decoupled and scalable system, where services can operate independently and react to changes in real-time.

Event-driven microservices are particularly useful for applications that require high scalability, real-time processing, and flexibility.

They are commonly used in domains like e-commerce, IoT, financial services, and any system where real-time data processing and responsiveness are critical.

However, they also introduce challenges in terms of event schema management, eventual consistency, and debugging, which need to be carefully addressed.

Key characteristics of event-driven microservices include:

1. **Event Producers and Consumers:** In this architecture, services can act as event producers, consumers, or both. An event producer generates events when something of interest happens (e.g., a new order is placed), and event consumers listen for these events to perform actions (e.g., processing the order).
2. **Asynchronous Communication:** Events are typically communicated asynchronously, meaning that the producer does not wait for the consumer to process the event. This allows services to operate independently and improves system responsiveness and scalability.
3. **Event Brokers:** An event broker or message broker (such as Apache Kafka, RabbitMQ, or AWS SNS/SQS) is often used to facilitate the distribution of events between services. The broker decouples producers from consumers, allowing them to evolve independently.
4. **Loose Coupling:** Services are loosely coupled because they do not need to know about each other directly. They only need to know about the events they produce or consume, which reduces dependencies and increases flexibility.
5. **Real-Time Processing:** Event-driven architectures are well-suited for real-time processing and can handle high volumes of events efficiently, making them ideal for applications that require immediate responses to changes.
6. **Scalability and Resilience:** The decoupled nature of event-driven microservices allows for independent scaling of services. If one service fails, it does not necessarily affect others, enhancing the system's resilience.
7. **Event Sourcing and CQRS:** Event-driven architectures often use patterns like event sourcing (storing the state of an entity as a sequence of events) and CQRS (Command Query Responsibility Segregation) to manage data consistency and separation of concerns.

## Event Driven

Event-driven refers to a [programming paradigm](#) or architectural style where the flow of the program is determined by events—changes in state or conditions that trigger specific actions or responses.

## Introduction

In this model, components of a system communicate through events, which can be generated by user interactions, system changes, or external sources.

---

## Key Concepts of Event-Driven Architecture:

1. Events: An event is a significant change in state or an occurrence that can trigger a response. For example, a user clicking a button, a file being uploaded, or a sensor detecting a change in temperature.
2. Event Producers: These are components or services that generate events. For instance, a web application might produce events when users perform actions like signing up or making a purchase.
3. Event Consumers: These are components or services that listen for and respond to events. They take action based on the events they receive, such as updating a database or sending a notification.
4. Event Channels: These are the pathways through which events are transmitted from producers to consumers. This can include message queues, event buses, or streaming platforms.
5. Loose Coupling: In an event-driven system, components are often loosely coupled, meaning that producers and consumers do not need to know about each other directly. This allows for greater flexibility and scalability.

## Benefits of Event-Driven Architecture:

- Scalability
- Responsiveness
- Flexibility

## Related topics

- [Event Driven Events](#)
- [Event-Driven Architecture](#)
- [Event Driven Microservices](#)
- **Tags:** #event\_driven, #data\_processing

# Event Driven Architecture

## Everything

Can we search with descriptions ?

## Tips

use \ to match in paths i.e \playground

can copy file

new window crl+ n

Use | to get or search

search syntax

# Excel & Sheets

## Links

[Google sheets example folder](#)

see [standardised/GSheets|GSheets](#)

Excel Example folder: Desktop/Desktop/Example\_Examples

## Tools common to Excel and Sheets

### Vlookup

Table

| Product ID | Product Name | Price  |
|------------|--------------|--------|
| 1001       | Apple        | \$1.00 |
| 1002       | Banana       | \$0.50 |
| 1003       | Orange       | \$0.80 |

$\$0.50 = VLOOKUP(1002, \text{Table}, 3, \text{FALSE})$

### Pivot table

## Excel specific

index-match

index-match-match

### Excel: Evaluate Formula

**Evaluate Formula** is a feature in Excel that allows you to step through complex formulas to see how Excel calculates the result. This tool is helpful for debugging or understanding how nested formulas work.

**Example:** Suppose you have a formula like this:

```
=IF(SUM(A1:A3) > 10, A4 * 2, A5 + 5)
```

To understand how Excel processes this formula, you can use **Evaluate Formula**. It will break down the formula into steps, showing how the `SUM(A1:A3)` is calculated, followed by the evaluation of the `IF` condition, and finally either the multiplication or addition operation based on the result.

### How to use it:

1. Select the cell containing the formula.
2. Go to the "Formulas" tab on the ribbon.
3. Click on "Evaluate Formula".
4. Click "Evaluate" to step through each part of the formula.

## Excel: What-If Analysis

**What-If Analysis** in Excel allows you to explore different scenarios by changing the inputs to your formulas. The three main tools within What-If Analysis are **Scenario Manager**, **Goal Seek**, and **Data Tables**.

- **Scenario Manager:** Lets you save different sets of input values and switch between them to see the impact on the result.
- **Goal Seek:** Finds the required input value to achieve a specific output.
- **Data Tables:** Shows how changing one or two variables affects your formulas.

**Example (Goal Seek):** Imagine you have a loan payment formula:

```
=PMT(interest_rate, number_of_periods, loan_amount)
```

You want to find out what interest rate would result in a monthly payment of \$500.

Steps:

1. Enter the formula with an initial guess for the interest rate.
2. Go to "Data" > "What-If Analysis" > "Goal Seek".
3. Set the cell with the formula to a value of 500.
4. Set the interest rate cell as the one to change.
5. Click "OK" and Excel will adjust the interest rate to meet the target.

## Excel: Forecast Sheets

**Forecast Sheets** use historical data to predict future values. Excel creates a forecast chart based on the pattern in the data, using linear or exponential smoothing models. This is particularly useful for time series data, such as sales or financial data over time.

**Example:** Suppose you have monthly sales data in a column:

| Month | Sales |
|-------|-------|
| Jan   | 1000  |
| Feb   | 1100  |
| Mar   | 1200  |
| Apr   | 1300  |

You want to forecast future sales for the next 6 months.

Steps:

1. Select the range of historical data (both months and sales).
2. Go to the "Data" tab on the ribbon.
3. Click "Forecast Sheet".
4. Excel will automatically create a forecast for the future months based on the trend in the data.
5. You can adjust the forecast options (e.g., forecast length, confidence intervals) before creating the sheet.

## Excel: Consolidate

In Excel, the **Consolidate** feature (found under the **Data** tab) allows you to combine data from multiple ranges or worksheets into a single summary. It is particularly useful when you have data spread across different locations and want to summarize it, such as calculating totals, averages, or other aggregate functions.

Key Features of Consolidate:

- **Multiple Ranges:** You can consolidate data from different ranges, even if they are in separate worksheets or workbooks.
- **Functions:** It provides several functions such as SUM, AVERAGE, COUNT, MIN, MAX, and more to aggregate the data.
- **Labels:** You can use row and column labels to match and consolidate the data correctly.

How to Use Consolidate:

1. **Prepare your data:** Ensure that your data is organized in a table format with similar structures (e.g., same columns and rows across different sheets or ranges).
2. **Navigate to Consolidate:** Go to the **Data** tab and click on **Consolidate** in the **Data Tools** group.
3. **Select Function:** In the Consolidate dialog box, select the function you want to use (e.g., **Sum**, **Average**, etc.).
4. **Add References:** Add the ranges you want to consolidate by clicking on **Add** after selecting the range. You can select ranges from different worksheets or workbooks.
5. **Use Labels:** If your data contains row or column labels, check the boxes for "Use labels in top row" or "Use labels in left column" to consolidate the data correctly based on these labels.
6. **Create links:** If you want the consolidated data to update automatically when the source data changes, check the box for **Create links to source data**.

Benefits:

- Saves time by avoiding manual data entry or copy-pasting from multiple sheets.
- Helps in summarizing large amounts of data quickly.
- Ensures accuracy in consolidation, especially with functions like **Sum** or **Average**.

Example: Suppose you have sales data in two worksheets as follows:

**Sheet1 (Region A):**

| Product | Sales |
|---------|-------|
| A       | 100   |
| B       | 200   |
| C       | 300   |

**Sheet2 (Region B):**

| Product | Sales |
|---------|-------|
| A       | 150   |
| B       | 250   |
| C       | 350   |

You can use the **Consolidate** feature to combine the sales from both regions into a summary table:

**Consolidated Sheet:**

| Product | Sales |
|---------|-------|
| A       | 250   |
| B       | 450   |
| C       | 650   |

In this case, you would use the **Sum** function in the Consolidate dialog to add the sales from the two regions.

### Excel: Text to Columns

The **Text to Columns** feature in Excel is used to split the data in one column into multiple columns, based on a delimiter (like commas, spaces, or tabs) or a fixed width. This is particularly helpful when you have data combined in a single column and you need to separate it into distinct parts.

Benefits:

- Efficiently splits combined data without manual editing.
- Helps with data cleaning when importing text-based data from sources like CSV files.
- Reduces errors by automating the process of separating values.

Key Features of Text to Columns:

- **Delimiters:** You can split text based on specific delimiters such as commas, spaces, semicolons, or custom characters.
- **Fixed Width:** If the data is in a consistent format, you can split the text based on fixed-width segments.

How to Use Text to Columns:

1. **Select the Data:** Highlight the column or cells that contain the text you want to split.
2. **Navigate to Text to Columns:** Go to the **Data** tab on the ribbon, then click **Text to Columns** in the **Data Tools** group.
3. **Choose the Split Type:**
  - **Delimited:** Choose this option if your data is separated by characters like commas, tabs, or spaces.
  - **Fixed Width:** Choose this if the data is aligned into specific columns with consistent spacing.
4. **Select Delimiters or Set Width:**
  - For **Delimited**, choose the character that separates your data (e.g., comma, space, semicolon).
  - For **Fixed Width**, manually set where the splits should occur by clicking in the preview window.
5. **Select Destination:** Choose where you want the split data to appear (by default, it will overwrite the original data).
6. **Finish:** Click **Finish** to apply the split.

Example 1: Delimited (Comma-Separated Values) Imagine you have a list of full names in one column:

| Full Name      |
|----------------|
| John,Smith     |
| Jane,Doe       |
| Robert,Johnson |

You want to split these names into two separate columns: First Name and Last Name. Here's how you would use **Text to Columns**:

1. Select the column with the full names.

2. Go to **Data > Text to Columns**.
3. Choose **Delimited**, then select **Comma** as the delimiter.
4. Click **Finish**. The data will be split into two columns:

| First Name | Last Name |
|------------|-----------|
| John       | Smith     |
| Jane       | Doe       |
| Robert     | Johnson   |

Example 2: Fixed Width Suppose you have a column with product codes where each section of the code has a fixed length:

| Product Code  |
|---------------|
| 12345ABC67890 |
| 67890DEF12345 |

If the first 5 characters are the product number, the next 3 characters are the product category, and the last 5 characters are the batch number, you can split them based on fixed widths:

1. Select the column with the product codes.
2. Go to **Data > Text to Columns**.
3. Choose **Fixed Width**.
4. In the preview window, set the breaks where you want to split the text (after the 5th and 8th characters).
5. Click **Finish**. The data will be split into separate columns like this:

| Product Number | Category | Batch Number |
|----------------|----------|--------------|
| 12345          | ABC      | 67890        |
| 67890          | DEF      | 12345        |

## Excel: Data Validation

- **Restrict Data Types:** You can limit entries to specific data types, such as whole numbers, decimal numbers, dates, or times.
- **Set Input Rules:** You can set conditions (e.g., numbers between 1 and 100, dates in a certain range, or specific text lengths).
- **Create Drop-Down Lists:** You can provide users with a predefined list of options to select from.
- **Custom Validation:** You can use formulas for advanced validation rules.
- **Error Alerts:** You can display custom messages to users when they try to enter invalid data.

How to Use Data Validation:

1. **Select the Cell(s):** Select the range of cells where you want to apply data validation.
2. **Go to Data Validation:** Navigate to the **Data** tab on the ribbon, and in the **Data Tools** group, click **Data Validation**.
3. **Set Validation Criteria:**
  - o In the **Settings** tab, choose the type of data you want to allow (Whole Number, Decimal, List, Date, Time, Text Length, or Custom).
  - o Specify the condition (e.g., a range of values or a list of items).

4. **Input Message (Optional):** In the **Input Message** tab, you can set a message that will appear when the user selects the cell, providing guidance on what they should enter.
5. **Error Alert:** In the **Error Alert** tab, define what happens if invalid data is entered. You can show an error message and choose whether to stop the entry, give a warning, or provide information.

**Restricting Input to a Range of Numbers** You want to restrict the values in a certain column to only allow whole numbers between 10 and 100.

Steps:

1. Select the column or cells where you want to apply the rule.
2. Go to **Data > Data Validation**.
3. In the **Settings** tab, choose **Whole Number** from the **Allow** drop-down.
4. Set the **Minimum** to 10 and the **Maximum** to 100.
5. Optionally, create an input message or error alert to inform the user of the valid range.

**Creating a Drop-Down List** You want users to select from a list of predefined options, such as product categories (e.g., "Electronics", "Furniture", "Clothing"). Steps:

1. Select the cells where the drop-down list should appear.
2. Go to **Data > Data Validation**.
3. In the **Settings** tab, choose **List** from the **Allow** drop-down.
4. In the **Source** field, type the list items, separated by commas: `Electronics,Furniture,Clothing`.
5. Click **OK**. Now users will see a drop-down arrow in the cells, allowing them to choose from the options.

**Validating Dates** You want to ensure that users can only enter dates within a specific range, such as between January 1, 2023, and December 31, 2023. Steps:

1. Select the cells where the dates will be entered.
2. Go to **Data > Data Validation**.
3. In the **Settings** tab, choose **Date** from the **Allow** drop-down.
4. Set the **Start Date** to 1/1/2023 and the **End Date** to 12/31/2023.
5. Click **OK**. Now users will only be able to enter dates within the specified range.

**Custom Validation with Formulas** You want to ensure that users can only enter text starting with the letter "A".

Steps:

1. Select the cells where the validation should apply.
2. Go to **Data > Data Validation**.
3. In the **Settings** tab, choose **Custom** from the **Allow** drop-down.
4. In the **Formula** field, enter:

```
=LEFT(A1, 1)="A"
```

5. Click **OK**. Now users will only be able to enter text starting with the letter "A".

## Google specific

Xlookup

# Explain Different Gradient Descent Algorithms, Their Advantages, And Limitations.

## Explain The Curse Of Dimensionality

The **curse of dimensionality** refers to the various phenomena that arise when working with data in high-dimensional spaces.

- **Increased Data Sparsity:** As the number of dimensions grows, the available data becomes increasingly sparse, making it difficult for algorithms to find **meaningful patterns**. This sparsity can lead to poor generalization performance, as the algorithm might not have enough data points in each region of the input space to learn a robust model.
- **Distance Metric Issues:** In high-dimensional spaces, traditional distance metrics like Euclidean distance become less effective, as the relative difference between the nearest and farthest points diminishes. This can make it difficult for algorithms like k-nearest neighbours to identify meaningful neighbours.
- **Difficulty in Visualization:** Visualizing data beyond three dimensions becomes incredibly challenging, making it difficult to gain insights from the data and understand the behaviour of machine learning models.

### Examples of the Curse of Dimensionality

#### Vulnerability of Ngrams Language Models:

Classical n-gram language models in **NLP**, which rely on counting the occurrences of word sequences, are particularly vulnerable to the curse of dimensionality. As the vocabulary size and the value of 'n' increase, the number of possible n-grams grows exponentially, making it impossible to observe most of them in even a massive training set.

### Addressing the Curse of Dimensionality

While the curse of dimensionality presents significant challenges, there are techniques to mitigate its effects:

- **Dimensionality Reduction:** Techniques like Principal Components Analysis (PCA), Factor Analysis, and Multidimensional Scaling (MDS) can reduce the number of features while retaining essential information, making it easier to visualize and analyze data and train machine learning models.
- **Feature Selection:** Identifying and selecting the most relevant features for a given task can improve model performance and reduce computational complexity.
- **Distributed Representations:** Using distributed representations, where information is encoded across multiple features rather than a single one, can help overcome the limitations of one-hot encodings in high-dimensional spaces.
- **Regularisation:** Techniques like weight decay in neural networks can help prevent overfitting and improve generalization performance, particularly in high-dimensional settings
- **Manifold Learning:** Manifold learning methods assume that the data lies on a lower-dimensional manifold embedded in a high-dimensional space. By learning this manifold structure, these methods can reduce dimensionality while preserving nonlinear relationships in the data.

# Exploration Vs. Exploitation

One of the major challenges in [Reinforcement learning](#) is balancing exploration (trying new actions) and exploitation (choosing the best-known actions).

The [epsilon-greedy strategy](#) is commonly used, where a small probability (`epsilon`) allows for exploration while primarily exploiting the best-known actions.

## Exploration

### Embeddings For Oov Words

Can you find words in a [Vector Embedding](#)/[word embedding](#) that were not used to create the embedding?

Yes, but with important caveats. If a word is not in the [spaCy](#) model's vocabulary with a vector, then:

#### What you can do

#### Option 1: Filter out words without vectors (what you're doing now)

This is the cleanest option:

```
if token.has_vector:
 embeddings.append(token.vector)
 valid_words.append(word)
```

#### Option 2: Fallback to character-level embeddings (optional)

If you're using `en_core_web_lg`, spaCy sometimes provides approximate vectors for out-of-vocabulary (OOV) words using subword features. But with `en_core_web_md`, OOV words truly lack vector meaning.

#### Option 3: Use a different embedding model

Use FastText or transformer-based models (e.g., Sentence Transformers), which can produce [embeddings for OOV words](#) based on subword information or context.

Example with [FastText](#) (using gensim):

```
from gensim.models import KeyedVectors

model = KeyedVectors.load_word2vec_format("cc.en.300.vec") # or download from FastText
embedding = model.get_vector("unseenword") # FastText will synthesize it
```

 **Summary**

| Approach                          | Handles OOV? | Notes                                       |
|-----------------------------------|--------------|---------------------------------------------|
| spaCy <code>en_core_web_md</code> | ✗            | Skips words without vectors (recommended)   |
| spaCy <code>en_core_web_lg</code> | ⚠ Sometimes  | May infer vectors using subword info        |
| FastText / GloVe                  | ✓            | Good for unseen words                       |
| Sentence Transformers (BERT)      | ✓            | Contextualized, ideal for phrases/sentences |

## NLP #ml\_process #ml\_optimisation

### Emergent Behavior

## F

---

# Table of Contents

- FAISS
- Fabric
- Fact Table
- Factor Analysis
- Factor\_Analysis.py
- Facts
- FastAPI
- FastAPI\_Example.py
- Feature Engineering
- Feature Evaluation
- Feature Extraction
- Feature Importance
- Feature Scaling
- Feature Selection vs Feature Importance
- Feature Selection
- Feature selection and creation
- Feature\_Distribution.py
- Feed Forward Neural Network
- Feedback Template
- Filter method
- Firebase
- Fishbone diagram
- Fitting weights and biases of a neural network
- Flask
- Folder Tree Diagram
- Forecasting\_AutoArima.py
- Forecasting\_Baseline.py
- Forecasting\_Exponential\_Smoothing.py
- Foreign Key
- Forward Propagation
- Fuzzywuzzy
- filter methods
- functional programming

## Faiss

FAISS (Facebook AI [Similarity Search](#)) is a library developed by Facebook AI Research that enables efficient similarity search and [clustering](#) of dense vectors. It is especially well-suited for applications involving high-dimensional vector data, such as [NLP](#)

Related terms:

- [Vector Embedding](#)

## Overview

FAISS is optimized for:

- **Fast retrieval** from large collections of vectors (millions or more).
- **Approximate nearest neighbor (ANN)** search, which trades off accuracy for speed.
- **Exact search**, depending on the chosen index type.
- **GPU acceleration** for very large-scale search tasks.

## Core Concept

At its core, FAISS takes a large number of **high-dimensional vectors** (e.g., sentence or document embeddings), and enables fast **similarity search** to retrieve the most similar vectors to a given [Querying|query](#).

For example, in an NLP [Memory|context](#):

- Documents or notes are embedded into vector space using a model like SBERT.
- These embeddings are stored in a FAISS index.
- Given a query, its embedding is computed, and FAISS returns the nearest neighbors (i.e., most semantically similar notes).

## Index Types

FAISS offers different types of indices depending on use case:

- `IndexFlatL2` : exact search using L2 (Euclidean) distance.
- `IndexIVFFlat` : approximate search using inverted files.
- `IndexHNSW` : Hierarchical Navigable Small World graph-based index (good for high recall).
- `IndexPQ` : product quantization for memory-efficient indexing.

## Fabric

Fabric is a unified analytics platform that operates in the cloud, eliminating the need for local installations. It provides an integrated environment for data analysis, similar to how [Microsoft Office](#) serves as a suite for productivity tasks.

## Key Features

- Unified Platform: Combines various data tools and services into a single platform, streamlining data analysis and management.
- Cloud-Based: Operates entirely in the cloud, ensuring accessibility and scalability without the need for local installations.
- Integrated Environment: Offers a cohesive environment for data analysis, integrating tools like [Data Factory](#), [Synapse](#), and [PowerBI](#).

## Components

- Data Factory: Facilitates data integration and transformation.
- Synapse: Acts as a [Relational Database](#) and [Data Warehouse](#), supporting [T-SQL](#) for data management.
- Data Lake: Fabric provides open data storage solutions, allowing for efficient data management and analysis.

## Introduction

- OneLake: Offers workspaces for different departments, enabling data sharing and referencing without duplication.
- 

## Technologies

- Programming Languages: Supports [Scala](#) and [PySpark](#) for data processing within the [Data Lake](#).
- PowerBI Integration: Enhances data visualization and querying capabilities within Fabric, offering faster insights.

## Advantages

- Data as Fuel: Recognizes data as the essential component powering AI and analytics.
- No ETL Required: Mirrors data sources, eliminating the need for [ETL](#) processes when source data is edited.
- Cross-Workspace Shortcuts: Allows departments to reference data across workspaces without creating copies.
- Copilot with PowerBI: Integrates AI-driven insights and automation within PowerBI for enhanced data analysis.

## Fact Table

A fact table is a central component of a star [Database Schema|schema](#) or snowflake schema in a [data warehouse](#), it stores [Facts](#).

Essential for [data analysis](#) in a data warehouse, providing the numerical data that can be analyzed in conjunction with the descriptive data stored in dimension tables.

1. **Measures:** Fact tables contain measurable, quantitative data known as "facts" or "measures." Examples include sales revenue, quantity sold, profit, or any other numeric data that can be aggregated.
2. **Foreign Keys:** Fact tables include foreign keys that reference [Dimension Tables](#). These keys link the fact table to related dimensions, allowing for contextual analysis. For example, a sales fact table might include foreign keys for dimensions such as time, product, and customer.
3. **Granularity:** The [granularity](#) of a fact table refers to the level of detail stored in the table. For example, a sales fact table might store data at the transaction level (each sale) or at a higher level (daily sales totals).
4. **Large Size:** Fact tables can become quite large, as they often store a significant amount of transactional data over time. This is in contrast to dimension tables, which are usually smaller and contain descriptive attributes.
5. **Aggregation:** Fact tables are often used for aggregation and analysis, allowing users to perform operations such as summing, averaging, or counting the measures.

Example:

- Columns: `DateKey` , `ProductKey` , `RegionKey` , `SalesAmount` , `UnitsSold`

Fact Table Tags: #data\_modeling, #data\_warehouse

## Factor Analysis

Factor Analysis (FA) is a statistical method used for:

- [dimensionality reduction](#),
  - [EDA](#)
-

- or latent variable detection

It identifies underlying relationships between observed variables by modeling them as linear combinations of a smaller number of **unobserved latent factors**.

In simpler terms, it helps reduce a large number of variables into fewer factors while retaining the core information and structure of the data. It assumes that observed variables are influenced by some common latent factors and unique errors.

## Key Features of Factor Analysis:

1. Latent Factors: These are unobserved variables that capture the shared variance among observed variables.
2. Variance Decomposition: FA splits the total variance of observed variables into:
  - Common variance: Shared by latent factors.
  - Unique variance: Specific to each observed variable.

In [ML\\_Tools](#) see: [Factor\\_Analysis.py](#)

## Next Steps:

1. Would you like to visualize the factors to understand how the data clusters in the new latent space?
2. Should we explore the relationships between the factors and target classes (e.g., species in the Iris dataset)?

# Factor\_Analysis.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Preprocess/Factor\\_Analysis.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Factor_Analysis.py)

## 1. Factor Loadings Table

This table shows how strongly each feature (e.g., `sepal length (cm)`) is correlated with the two extracted factors (Factor 1 and Factor 2).

- **Rows:** Represent the extracted factors (`Factor 1` and `Factor 2`).
- **Columns:** Represent the original features of the Iris dataset.
- **Values:** Represent the "loading" or contribution of each feature to the factor. Higher absolute values indicate a stronger relationship between a feature and a factor.

## Interpretation:

- **Factor 1 (Row 0):**
  - Strongly influenced by `petal length (cm)` (loading = 1.757902) and `petal width (cm)` (loading = 0.731005).
  - Moderately influenced by `sepal length (cm)` (loading = 0.727461).
  - Weak negative contribution from `sepal width (cm)` (loading = -0.180852).
  - This factor might represent the size of petals and sepals.
- **Factor 2 (Row 1):**
  - Weak contributions from all features, with slightly negative contributions from `sepal length (cm)` and `sepal width (cm)`.
  - This factor might capture a subtle relationship or orthogonal variance not well-defined by the dataset.

## 2. Explained Variance

The explained variance values indicate how much of the dataset's total variance is captured by each factor.

- **Factor 1 (0.9988):** This factor explains ~99.88% of the variance among the features.
- **Factor 2 (0.9039):** This factor explains ~90.39% of the variance among the features.

### Combined Variance:

The two factors together capture a large portion of the total variance in the dataset. This suggests that most of the information in the dataset can be reduced to two latent factors, simplifying its structure while retaining the core relationships.

### Overall Interpretation

1. **Dimensionality Reduction:** The dataset with four features can effectively be reduced to two latent factors while retaining most of its variance.
2. **Factor 1 Dominates:** Factor 1 has strong contributions from `petal length`, `petal width`, and `sepal length`. This factor likely represents size-related characteristics.
3. **Factor 2 is Subtle:** Factor 2 shows weaker relationships with the features, potentially capturing noise or orthogonal variance.

### Next Steps:

1. Would you like to visualize the factors to understand how the data clusters in the new latent space?
2. Should we explore the relationships between the factors and target classes (e.g., species in the Iris dataset)?

### Breakdown of Extensions:

#### 1. Visualization:

After performing factor analysis, we visualize how the data clusters in the new latent space (Factor 1 vs. Factor 2). The points are colored based on the species (target classes), which helps us see if the factors capture any clustering patterns related to species.

##### o Plot Details:

- The x-axis represents **Factor 1**.
- The y-axis represents **Factor 2**.
- Each species is plotted in different colors to visualize possible separations.

#### 2. Exploring Factor-Target Relationships:

We compute the **average factor values** for each species. This shows how the latent factors (Factor 1 and Factor 2) relate to the different species in the dataset.

##### o Interpretation:

- If any species tends to cluster around specific values of Factor 1 and Factor 2, it suggests that the extracted factors capture some species-specific variance.

## Next Steps:

- The plot should give a clear idea of whether the latent factors allow for a meaningful separation of species.
- The summary table of average factor values will help understand how the factors relate to the target variable.

## Facts

Facts are quantitative data points that are typically stored in the [Fact Table](#).

They represent measurable events or metrics, such as sales revenue or quantities sold.

## Fastapi

**FastAPI** is a modern web framework for building APIs with Python. It is designed to be fast and easy to use, leveraging Python's type hints to provide features like:

1. Automatic generation of OpenAPI documentation.
2. Input data validation based on Python's type annotations.
3. Asynchronous request handling with native support for `asyncio`.
4. High performance, as it is built on Starlette and [Pydantic](#).

### Key Features

5. **Automatic validation:** Based on type hints and Pydantic models.
6. **Interactive API docs:** Automatically generated Swagger UI and ReDoc.
7. **Asynchronous support:** Full support for async functions.
8. **Dependency injection:** Built-in support for dependencies.

## How to Run

In [ML\\_Tools](#) see: [FastAPI\\_Example.py](#) <- see this

1. Save the script as `main.py`.
2. Install FastAPI and Uvicorn: `pip install fastapi uvicorn`
3. Run the server:

```
ren FastAPI_Example.py main.py # possibly change to
uvicorn FastAPI_Example:app --reload
```

4. Open the browser and navigate to:
  - **API documentation (Swagger UI):** <http://127.0.0.1:8000/docs>
  - **ReDoc documentation:** <http://127.0.0.1:8000/redoc>

## Fastapi\_Example.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Deployment/FastAPI\\_Example.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Deployment/FastAPI_Example.py)

## Explanation of New Features

1. **Path and Query Parameter Metadata:** Added descriptions and constraints for better validation and autogenerated documentation.
2. **Nested Models:** Demonstrated hierarchical data validation with the `User` model that includes a list of `Item` instances.
3. **Partial Updates:** Introduced a `PATCH` endpoint to allow partial updates of fields using the `Body` method.
4. **Static Data:** Provided a status endpoint that returns static information about the API.
5. **Returning Key-Value Data:** Added a summary endpoint to showcase mock data.

## Main

### What the Script Does

#### 1. Starts the FastAPI Application

When you run the script, it starts a web server using Uvicorn. This makes your FastAPI app accessible via the browser or API clients like Postman.

#### 2. Default Endpoint (`GET /`)

The browser sends a `GET` request to the root path (`http://127.0.0.1:8000/`).

The root route (`@app.get("/")`) is defined in the script to return:

```
{"message": "Welcome to the expanded FastAPI example!"}
```

This is why you see that message in the browser or the API response.

#### 3. 404 for `/favicon.ico`

The browser automatically tries to load a favicon (a small icon displayed in the browser tab). Since no favicon is defined in the script, a `404 Not Found` is returned, which is normal behavior.

### What the Script Offers Beyond the Root Endpoint

The script defines several API endpoints that are not automatically accessed unless you explicitly call them. Here's a summary of the key routes:

| Endpoint         | HTTP Method | Description                                                                                        |
|------------------|-------------|----------------------------------------------------------------------------------------------------|
| /                | GET         | Returns a welcome message.                                                                         |
| /items/{item_id} | GET         | Fetches an item by its <code>item_id</code> , With an optional query parameter.                    |
| /search/         | GET         | Searches items using <code>limit</code> and <code>offset</code> query parameters for pagination.   |
| /items/          | POST        | Creates a new item using the <code>Item</code> model for validation.                               |
| /items/{item_id} | PUT         | Updates an item by its <code>item_id</code> with a new <code>Item</code> object.                   |
| /items/{item_id} | DELETE      | Deletes an item by its <code>item_id</code> .                                                      |
| /users/          | POST        | Creates a new user with optional nested items using the <code>User</code> model.                   |
| /items/{item_id} | PATCH       | Partially updates fields (like <code>price</code> or <code>on_offer</code> ) of an item by its ID. |
| /status/         | GET         | Returns static data, such as the API's status and version.                                         |
| /summary/        | GET         | Returns a dictionary with some mock data, such as total items and users.                           |

## Testing the Script

You need to explicitly visit or call other endpoints to explore more features of the script. For example:

1. **Accessing the Swagger UI** Go to <http://127.0.0.1:8000/docs> in your browser.

This interactive interface shows all the available endpoints and lets you test them directly.

2. **Calling Specific Endpoints** You can call the endpoints via:

- o **Browser** (e.g., <http://127.0.0.1:8000/items/123?q=test>).
- o **API Clients** like Postman or Curl.

## Why You See Only the Welcome Message

You've accessed only the root endpoint (`/`). The other features of the script (like creating, updating, or searching for items) require you to call the respective endpoints explicitly.

## Calling endpoints

Here's how to call the respective endpoints of the script using various tools like **browser**, **Curl**, or **Postman**. Each example demonstrates an endpoint and its purpose.

## 1. Welcome Endpoint ( GET / )

---

- **Purpose:** Displays a welcome message.
- **Call:**

- **Browser:** Open `http://127.0.0.1:8000/`.
- **Curl:**

```
curl -X GET http://127.0.0.1:8000/
```

- **Response:**

```
{"message": "Welcome to the expanded FastAPI example!"}
```

## 2. Retrieve an Item ( GET /items/{item\_id} )

- **Purpose:** Fetches item details by its `item_id` with an optional query parameter `q`.
- **Example:** Retrieve item `3` with query `test`.
- **Call:**

- **Browser:** Open `http://127.0.0.1:8000/items/3?q=test`.
- **Curl:**

```
curl -X GET "http://127.0.0.1:8000/items/3?q=test"
```

- **Response:**

```
{"item_id": 3, "query": "test"}
```

---

## 3. Search Items ( GET /search/ )

- **Purpose:** Uses `limit` and `offset` query parameters for pagination.
- **Example:** Limit results to 5, skip the first 2.
- **Call:**

- **Browser:** Open `http://127.0.0.1:8000/search/?limit=5&offset=2`.
- **Curl:**

```
curl -X GET "http://127.0.0.1:8000/search/?limit=5&offset=2"
```

- **Response:**

```
{"limit": 5, "offset": 2}
```

## 4. Create an Item ( POST /items/ )

- **Purpose:** Adds a new item.
- **Example:** Create an item named "Laptop" priced at 999.99.
- **Call:**

- o **Curl:**

```
curl -X POST "http://127.0.0.1:8000/items/" -H "Content-Type: application/json" -d "{\"name\": \"Laptop\", \"pri
```

- o **Response:**

```
{
 "message": "Item created successfully",
 "item": {
 "name": "Laptop",
 "price": 999.99,
 "description": "High-end laptop",
 "on_offer": true
 }
}
```

## 5. Update an Item ( `PUT /items/{item_id}` )

- **Purpose:** Updates item details by its `item_id`.

- **Example:** Update item `3` with new data.

- **Call:**

- o **Curl:**

```
curl -X PUT "http://127.0.0.1:8000/items/3" \
-H "Content-Type: application/json" \
-d '{"name": "Smartphone", "price": 499.99, "description": "Updated phone", "on_offer": false}'
```

- o **Response:**

```
{
 "item_id": 3,
 "updated_item": {
 "name": "Smartphone",
 "price": 499.99,
 "description": "Updated phone",
 "on_offer": false
 }
}
```

## 6. Delete an Item ( `DELETE /items/{item_id}` )

- **Purpose:** Deletes an item by its `item_id`.

- **Example:** Delete item `2`.

- **Call:**

- o **Curl:**

```
curl -X DELETE http://127.0.0.1:8000/items/2
```

- **Response:**

```
{"message": "Item 2 deleted successfully"}
```

## 7. Create a User ( POST /users/ )

- **Purpose:** Creates a user, optionally with nested items.
- **Example:** Create a user with items.
- **Call:**

- **Curl:**

```
curl -X POST "http://127.0.0.1:8000/users/" -H "Content-Type: application/json" -d "{\"username\": \"john_doe\"}
```

- **Response:**

```
{
 "message": "User created successfully",
 "user": {
 "username": "john_doe",
 "email": "john@example.com",
 "full_name": "John Doe",
 "items": [
 {
 "name": "Tablet",
 "price": 299.99,
 "description": "Portable tablet",
 "on_offer": null
 }
]
 }
}
```

## 8. Partially Update an Item ( PATCH /items/{item\_id} )

- **Purpose:** Partially updates an item using optional fields.
- **Example:** Update the price of item 5.
- **Call:**

- **Curl:**

```
curl -X PATCH "http://127.0.0.1:8000/items/5" \
-H "Content-Type: application/json" \
-d '{"price": 79.99}'
```

- **Response:**

```
{
 "item_id": 5,
 "updates": {
 "price": 79.99
 }
}
```

## 9. Check API Status ( GET /status/ )

- **Purpose:** Returns static information like API status.

- **Call:**

- **Browser:** Open `http://127.0.0.1:8000/status/`.
- **Curl:**

```
curl -X GET http://127.0.0.1:8000/status/
```

- **Response:**

```
{"status": "API is running", "version": "1.0.0"}
```

## 10. Retrieve a Summary ( GET /summary/ )

- **Purpose:** Returns a mock summary of items and users.

- **Call:**

- **Browser:** Open `http://127.0.0.1:8000/summary/`.
- **Curl:**

```
curl -X GET http://127.0.0.1:8000/summary/
```

- **Response:**

```
{"total_items": 42, "total_users": 5, "recent_activity": "Item purchase"}
```

## Testing Tips

- Use **Swagger UI** at `http://127.0.0.1:8000/docs` for interactive testing of all endpoints.
- Use **Postman** for testing more complex requests with nested data.

## New

### Explanation of Changes:

- `created_items` : Items that have been created using the `/items/` POST endpoint.
- `updated_items` : Items that have been updated using the `/items/{item_id}` PUT endpoint.
- `deleted_items` : Items that have been deleted using the `/items/{item_id}` DELETE endpoint.

- **Summary Endpoint:**

- Returns counts of items created (`created_items_count`), updated (`updated_items_count`), and deleted (`deleted_items_count`).

---

Here's the formatted content for use in `cmd`:

**TEST LATER**

## Testing with `curl`:

### 1. Create an item:

```
curl -X POST "http://127.0.0.1:8000/items/" -H "Content-Type: application/json" -d '{"name": "Tablet", "price": 299.99}'
```

### 2. Update an item:

```
curl -X PUT "http://127.0.0.1:8000/items/1" \
-H "Content-Type: application/json" \
-d '{"name": "Tablet", "price": 250.00, "description": "Updated portable tablet"}'
```

### 3. Delete an item:

```
curl -X DELETE "http://127.0.0.1:8000/items/1"
```

### 4. Get the summary:

```
curl -X GET "http://127.0.0.1:8000/summary/"
```

## Expected Response:

```
{
 "total_items": 0,
 "total_users": 5,
 "recent_activity": "Item purchase",
 "created_items_count": 1,
 "updated_items_count": 1,
 "deleted_items_count": 1
}
```

You can copy and paste these commands directly into `cmd` to test the API.

## Note:

Make sure your FastAPI server is running on `http://127.0.0.1:8000` before executing these commands.

# Feature Engineering

It's the term given to the iterative process of building good features for a better model. It's the process that makes relevant features (using formulas and relations between others).

We use it when we have a refined and optimised model.

What does it involve

- Create new features from existing ones (e.g., ratios, interactions).
- Transform features to better capture non-linear relationships.
- **Dimensionality Reduction** if necessary.

The main techniques of feature engineering:

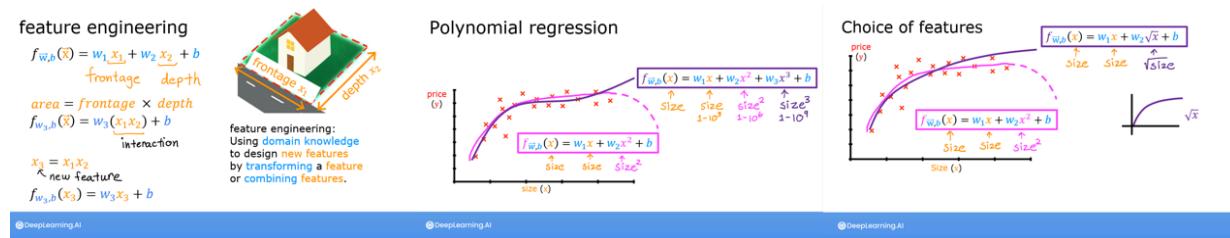
- are selection (picking subset),
- learning (picking the best),
- extraction and combination(combining).

Example: Predicting house prices. Raw features might be square footage, number of bedrooms, and location.

Feature engineering could involve: Combining square footage and bedrooms into a "living space" feature.

**Example:**

- Decompose datetime information into separate features for date and time to capture their respective predictive powers.



# Feature Evaluation

## Note

Garbage in garbage out. It is the features that

## What is involved:

Want to assess the **usefulness** of chosen features

### Measuring feature importance:

Quantifying the contribution of each feature to the model's predictions. This can be done through various methods like statistical tests, model-specific importance scores, or permutation importance.

## Analysing feature relationships:

Investigating correlations and redundancy,

## Assessing feature impact on model performance:

## Example:

## When are we done:

- \*\*Reaching a stable and satisfactory model performance
- \*\*No further room for improvement
- \*\*Understanding the model's decision-making process

# Feature Extraction

## Summary:

In machine learning, **Feature extraction** is the process of transforming raw data into a set of useful features that can be effectively used by algorithms. It involves identifying and selecting key attributes or characteristics of the data that are most relevant to the problem at hand. Feature extraction helps improve both the performance and efficiency of machine learning models.

Feature extraction simplifies complex data by transforming it into a smaller set of **informative features**. Techniques like **Dimensionality Reduction** help retain the most significant information, improving model performance and **interpretability**. It is conceptually similar to the way the **Attention mechanism** in **LLMs** captures relationships between concepts, or how **Activation atlases** visualize key patterns in deep learning models.

## Key Concepts of Feature Extraction:

### 1. Similarity and Relations:

- Feature extraction can be compared to how the **Attention mechanism** works in large language models (LLMs), which identify relationships and similarities between concepts. For example, the analogy "King - Queen ~ Man - Woman" highlights how certain features (gender, royalty) can be extracted to understand the underlying relationship between words.

Similarly, in feature extraction, relationships between data points can be used to capture important aspects of the data, such as patterns or correlations, and transform them into features that a model can learn from.

### 2. Dimensionality Reduction:

- One of the key techniques in feature extraction is **Dimensionality Reduction**, which is used to reduce the number of features while still preserving the important information in the data. This involves compressing the data into a smaller set of features that capture most of its variance. By doing this, you improve the efficiency of the machine learning model and make the analysis more interpretable. Allowing to focus on the most important features while **reducing noise** and redundancy.

### 3. Visual Feature Extraction:

- In the case of complex data like images, techniques such as **Activation atlases** can be used to visualize and understand the features that are being extracted and activated within a neural network. These atlases

show how different neurons in a neural network respond to specific features or patterns within the data, giving insights into what the model "sees" as important.

---

## Feature Importance

Feature importance refers to techniques that assign scores to input features (predictors) in a machine learning model to indicate their relative impact on the model's predictions.

Feature importance is typically assessed after Model Building. It involves analyzing the trained model to determine the impact of each feature on the predictions.

Feature importance helps in:

- improving model interpretability,
- identifying key predictors,
- and possibly performing Feature Selection to reduce dimensionality, and refining performance

The outcome is a ranking or scoring of features based on their importance.

By understanding which features contribute the most to the predictions, you can focus on the most relevant information in your data and potentially reduce model complexity without sacrificing performance.

## Types of Feature Importance Methods

### 1. Model-Specific Methods:

- Tree-based models: Models like Random Forests, Gradient Boosted Trees, and Decision Trees have built-in mechanisms for calculating feature importance. They do so based on the decrease in impurity (e.g., Gini Impurity in classification tasks or variance in regression tasks) or based on the reduction in error when the feature is used for splitting.
- Linear models: In models like linear regression or logistic regression, feature importance can be derived from the absolute values of the model coefficients, assuming features are standardized.

### 2. Model-Agnostic Methods:

- Permutation importance: This method measures the importance of a feature by randomly shuffling its values and observing the impact on the model's performance. The larger the decrease in performance, the more important the feature is.
- SHapley Additive exPlanations
- Local Interpretable Model-agnostic Explanations

## Code snippets for conducting Feature Importance

[SHapley Additive exPlanations](#)

[Local Interpretable Model-agnostic Explanations](#)

Tree-based algorithms like Random Forests or XGBoost automatically calculate feature importance.

In Python, for example, after training a Random Forest model, you can access the feature importance scores using:

```
from sklearn.ensemble import RandomForestClassifier

Train a RandomForest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

Get feature importance scores
importances = model.feature_importances_
```

This method uses the decrease in node impurity as a measure of feature importance.

## Feature Scaling

Used in preparing data for machine learning models.

Feature Scaling is a [preprocessing](#) step in machine learning that involves adjusting the range and distribution of feature values.

This ensures that all features contribute equally to the model's performance, especially when they are measured on different scales, which is particularly important for distance-based algorithms, [Principal Component Analysis](#), and optimization techniques like [Gradient Descent](#).

By using methods like normalization and standardization, you can enhance the performance and accuracy of your models.

See `sklearn.preprocessing`

Examples of algorithms not affected by feature scaling are [Naive Bayes](#), [Decision Tree](#), and [Linear Discriminant Analysis](#).

## Why Use Feature Scaling?

Feature scaling is important for several reasons:

1. Distance-Based Algorithms: Algorithms like k-nearest neighbors (KNN) rely on distance measures (e.g., Euclidean distance). If features are on different scales, those with larger magnitudes will disproportionately influence the distance calculations. Scaling ensures that all features weigh equally.
2. Principal Component Analysis (PCA): PCA aims to identify the directions (principal components) that maximize variance in the data. If features have high magnitudes, they will dominate the variance calculation, skewing the results. Scaling helps to mitigate this issue.
3. Gradient Descent Optimization: In optimization algorithms like gradient descent, features with larger ranges can cause inefficient convergence. Scaling ensures that all features are on a similar scale, allowing for faster and more stable convergence to the optimal solution.

## Common Scaling Methods

[Normalisation](#)

[Standardisation](#)

Min-Max Scaling: Scales features to a fixed range (e.g.,  $[0, 1]$ ), preserving relative distances.

## Example of Scaling

Here's how you can scale a DataFrame using the `scale` function from `sklearn`:

```
from sklearn import preprocessing
df_scaled = preprocessing.scale(df) # Scales each variable (column) with respect to itself
```

This returns an array where each feature is standardized.

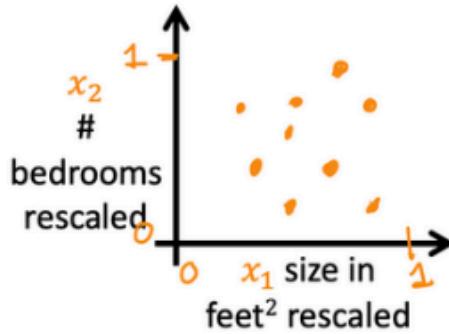
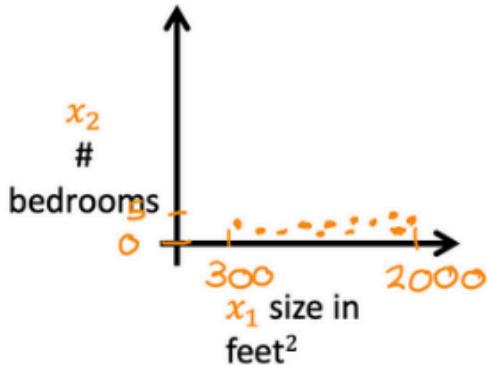
**Note:**

- Scaling is done when one feature is at a significantly different scale.
- For each data point, subtract the mean and divide by the range (max-min).

### Practice quiz: Gradient descent in practice

Total points 4

1.



1 point

Which of the following is a valid step used during feature scaling?

- Subtract the mean (average) from each value and then divide by the (max - min).
- Add the mean (average) from each value and then divide by the (max - min).

## Feature Selection Vs Feature Importance

### Summary

- **Feature Selection** is about choosing which features to include in the model **before training**, aiming to improve model performance and efficiency.
- **Feature Importance** is about understanding the role and impact of each feature **after the model has been trained**, providing insights into the model's decision-making process.

Use for **interpretability** of the model, but they are applied at different stages and serve different purposes.

# Feature Selection

---

Purpose: The primary goal of feature selection is to identify and retain the most relevant features for model training while **removing irrelevant or redundant ones**. This helps in simplifying the model, reducing overfitting, and improving computational efficiency.

Process: Feature selection is typically performed before model training. It involves evaluating features based on certain criteria or algorithms to decide which features to keep or discard.

Through an iterative process, feature selection experiments with different methods, adjusts parameters, and evaluates model performance until an optimal set of features is found.

See [Feature Selection vs Feature Importance](#)

## Methods

The choice of feature selection method depends on factors like the size of your dataset, the number of features, and the complexity of your model. It's often a balance between computational cost and performance improvement.

- **Filter Methods:** Select features based on statistical properties, independent of any machine learning algorithm. (Separate stage to training)
- **Wrapper Methods:** Involve training multiple models with different subsets of features and selecting the subset that yields the best performance. (Separate stage to training)
- **Embedded Methods:** Perform feature selection as part of the model training process. (Part of training)

After selecting features, it's essential to evaluate your model's performance ([Model Evaluation](#)) with the chosen subset. Sometimes, feature selection can inadvertently remove important information.

## Detecting Noisy or Redundant Features

- **Correlation Analysis:** Use a [Heatmap](#) or [Clustering](#). Features with low correlation to the target or high correlation with other features may be candidates for removal.
- **Dimensionality Reduction Techniques:** Techniques like [Principal Component Analysis](#) or [Singular Value Decomposition \(SVD\)](#) can transform the features into a lower-dimensional space while preserving as much variance as possible. Features with low contribution to the principal components can be considered for removal.
- **Visualizations:** Plotting pairwise scatter plots or [Heatmap](#) of feature [Correlation](#) can provide visual insights into redundant features. Clusters of highly correlated features or scatter plots showing no discernible pattern with the target variable can indicate noisy or redundant features.

## Investigating Features

- **Variance Thresholding:** Check the [Variance & Distributions](#) of each feature. Features with very low variance (close to zero) contribute little information and may be considered noisy. Removing such features can help simplify the model without sacrificing much predictive power.
- **Univariate Feature Selection:** Use statistical tests like chi-square for categorical variables or [ANOVA](#) for numerical variables to assess the relationship between each feature and the target variable. Features with low test scores or high p-values may be less relevant and can be pruned.

# Feature Selection And Creation

---

[Feature Selection](#)

[Feature Engineering](#)

After the data is ready.

Which features have the best value, which play the biggest role.

Combining features to simplify the

How to select features.

- Correlation between each two (poor)
- Stepwise regression
- Lasso and ridge regression

When selecting features we ask:

- Can we control it/select it?
- Can we control it easily what do we gain from it
- is it a sensible variable?

## Feature\_Distribution.Py

## Feed Forward Neural Network

A **Feedforward Neural Network (FFNN)** is the simplest type of [Neural network](#). In this model, connections between neurons do not form a cycle, allowing data to flow in one direction—from the input layer, through the hidden layers, to the output layer—without any loops or backward connections. This straightforward design is primarily used for [supervised learning](#) tasks.

### Structure

- Information flows in one direction: input → hidden layers → output.
- During [forward propagation](#), input data is passed through the network, processed by each layer, and an output is produced.
- Unlike [recurrent neural networks](#) (RNNs), FFNNs do not share information or weights between layers, meaning the model does not maintain memory of past inputs.

### Learning

- FFNNs learn by adjusting weights and biases during training to minimize the [Loss function](#).

### Limitations

- **Shallow vs. Deep Networks:** Simple feedforward networks with only a few hidden layers (shallow networks) may struggle to learn complex, hierarchical representations of data. Deeper networks (deep feedforward networks) with many layers can model more complex patterns but require more data and computational resources to train.

- **Overfitting:** FFNNs can overfit on the training data, especially if they have many parameters and not enough regularization (e.g., dropout, [Ridge|L2](#) regularization).
  - **No Temporal Understanding:** Unlike [Recurrent Neural Networks](#) or transformers, FFNNs cannot model sequential dependencies in data. They are better suited for static, non-sequential tasks.
- 

## Feedback Template

- Praise: I really appreciate your work on this
  - *add here*
- FYI: It's really not a big deal, but I'm letting you know just in case.
  - *add here*
- Suggestion: I'm fairly confident this would help, but I can live without it
  - *add here*
- Recommendation: This could be holding you back
  - *add here*
- Plea: It's almost at the breaking point if it's not already there.
  - *add here*

## Filter Method

## Firebase

Googles version of [AWS](#)

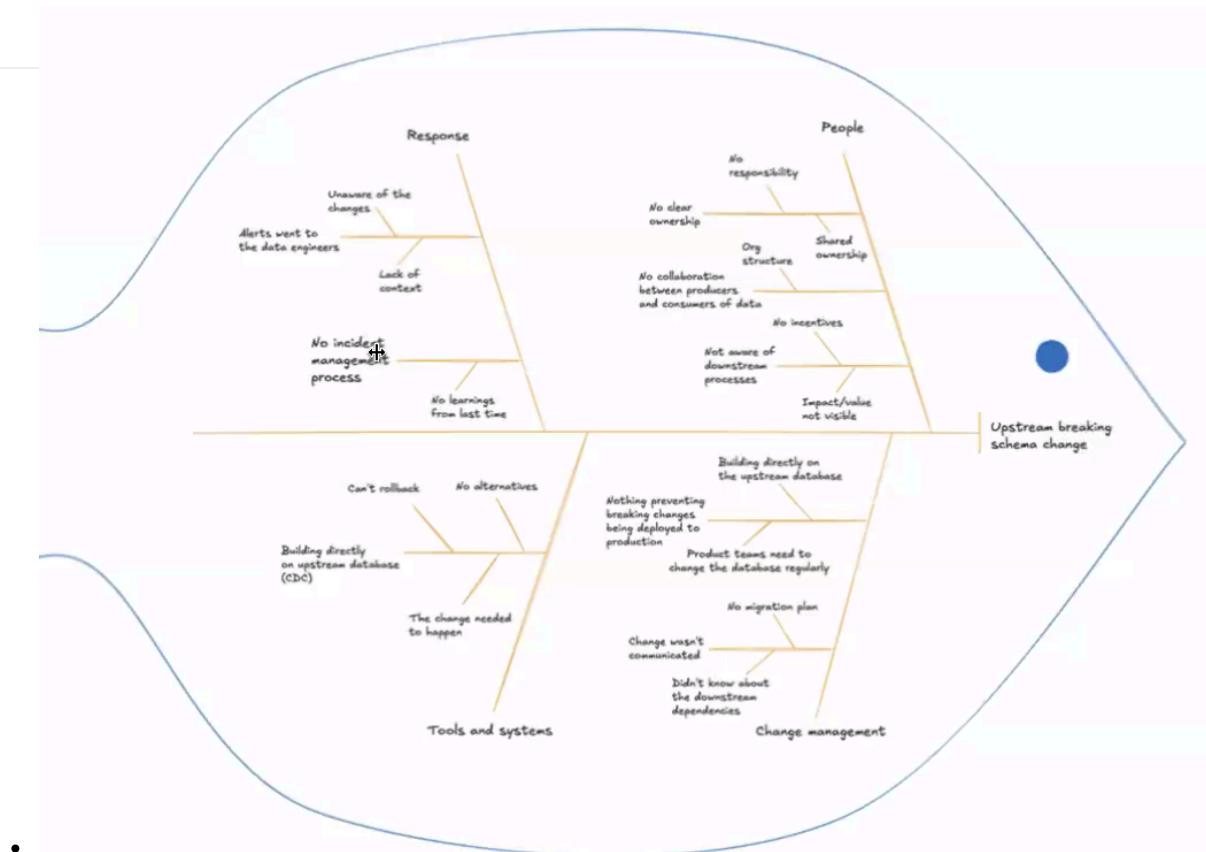
[Setup basics](#)

Project idea: Set up a basic emailer app.

## Fishbone Diagram

Fishbone diagram [Documentation & Meetings](#) Root cause analysis: [Documentation & Meetings](#)

- 5 Y's
  - Fishbone diagram: start at issue at head
-



- People and ownership: Who is entering the data: the source data

## Fitting Weights And Biases Of A Neural Network

For a neural network model, fitting weights and biases involves optimizing these [Model Parameters](#) so the model learns to map input features ( $X$ ) to target outputs ( $y$ ) effectively. This is achieved through the training process, which minimizes the error between predictions and actual values.

### Best Practices

- Use appropriate weight initializations like He or Xavier.
- Choose a suitable [loss function](#) for the task.
- Optimize using advanced optimizers like Adam or RMSprop.
- Experiment with batch sizes, epochs, and learning rates.
- Apply regularization (L2, [Dropout](#)) to prevent overfitting.
- Monitor validation metrics and use early stopping.

In [ML\\_Tools](#) see: Neural\_Net\_Weights\_Biases.py

## Initialization of Weights and Biases

Initializing all weights randomly. The weights are assigned randomly by initializing them very close to 0. It gives better accuracy to the model since every neuron performs different computations.

Proper initialization is critical for training to converge efficiently. Poor initialization can lead to slow convergence or getting stuck in local minima. By starting with well-chosen initial values, the network can learn more effectively and avoid issues like vanishing or exploding gradients.

Weights:

- Use small random values (e.g., drawn from Gaussian or uniform [distributions](#)) to break symmetry and ensure that neurons learn different features.
- Initialization techniques like He initialization (for ReLU activations) or Xavier initialization (for sigmoid/tanh activations) are commonly used because they help maintain the scale of gradients across layers, promoting stable and faster convergence.

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.initializers import HeNormal

Example of He initialization for ReLU activation
Dense(25, activation="relu", kernel_initializer=HeNormal())
```

Biases:

- Start with zeros (`0`) to ensure symmetry-breaking during optimization. This allows the network to learn offsets for the activations without introducing bias in the initial learning phase.

## Forward Propagation

During forward propagation, the network computes activations using the current weights and biases, and passes these activations to subsequent layers to generate predictions. This step is crucial as it determines how well the network can map inputs to outputs based on its current parameters.

## Loss Function

The loss function quantifies the difference between predicted outputs and true labels. It serves as the objective function that the network aims to minimize during training. Choosing the right loss function is essential as it directly impacts the learning process and the network's ability to generalize.

- Binary Cross-Entropy: For [Binary Classification](#).
- Categorical Cross-Entropy: For multi-class classification.
- Mean Squared Error (MSE): For regression tasks.

Example:

```
from tensorflow.keras.losses import BinaryCrossentropy
loss_fn = BinaryCrossentropy()
```

## Backpropagation

Backpropagation computes the gradients of the loss function with respect to weights and biases using the chain rule. This process is fundamental for learning, as it provides the necessary information to update the parameters in a way that reduces the loss.

## Gradient Descent Optimization

Gradients from backpropagation are used to update weights and biases iteratively. Optimization algorithms like Adam, RMSprop, and [Stochastic Gradient Descent|SGD](#) with momentum are crucial as they determine the efficiency and speed of convergence, especially in large datasets and complex models.

Example:

```
from tensorflow.keras.optimizers import Adam
optimizer = Adam(learning_rate=0.001)
```

## Batch Training

Weights and biases are updated after processing a batch of data. Batch training helps in stabilizing the learning process and can lead to faster convergence compared to updating after each sample. The choice of batch size and number of epochs affects the trade-off between computational efficiency and the quality of the learned model.

## Regularisation Techniques

Prevent overfitting by penalizing large weights. Regularization is essential for improving the generalization of the model, ensuring it performs well on unseen data.

[Ridge Dropout](#)

## Learning Rate Tuning

Learning rate impacts convergence. It is a [hyperparameter](#) that determines the step size during optimization. A poorly chosen learning rate can lead to divergence or slow convergence.

Techniques:

- [Learning Rate](#) Scheduling: Reduce learning rate as training progresses to fine-tune the learning process.
- Adaptive Learning Rates: Optimizers [Optimisation techniques](#)

## software

web app framework for writing web pages

uses decorators

```
app.py
import flask

app = flask.Flask(__name__)

@app.route("/hello/")
def hello():
 return "Hello World!"

if __name__ == "__main__":
 app.run(debug=True)
```

## Flask

### Flask app example

<https://www.youtube.com/watch?v=wBCEDCiQh3Q&list=PLcWfeUsAys2my8yUIOa6jEWB1-QbkNSU>

You can run a flask app in google colab and then share it publicly with ngrok.

flask app saved on github.

## Folder Tree Diagram

### Links

Simple method <https://www.digitalcitizen.life/how-export-directory-tree-folder-windows/>

More general

<https://superuser.com/questions/272699/how-do-i-draw-a-tree-file-structure>

[Treeviz Graphviz](#)

tree /a /f >output.doc

## Forecasting\_Autoarima.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/TimeSeries/Forecasting\\_AutoArima.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/TimeSeries/Forecasting_AutoArima.py)

## Tools and Resources

- AutoARIMA: Automatically selects the best ARIMA model parameters. Available in the statsforecast library by Nixtla.
- Implementation Example: See `TS_AutoArima.py` in `ML_Tools` for practical implementation.

## Performance Insights

- **Evaluation Metrics:** Marginal differences in evaluation metrics across ARIMA models may occur due to the volatile nature of the data.

## Forecasting\_Baseline.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/TimeSeries/Forecasting\\_Baseline.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/TimeSeries/Forecasting_Baseline.py)

Baseline methods are essential for establishing a performance benchmark. They provide insights into the data's underlying patterns and help in assessing the effectiveness of more sophisticated forecasting models. By comparing advanced models against these baselines, you can determine if the added complexity is justified by improved accuracy.

### Methods Implemented:

- ```
- **Mean Forecasting:** Uses the average of all past values as the forecast for future periods.
- **Naive Forecasting:** The last observed value is used as the forecast for all future periods.
- **Seasonal Naive Forecasting:** Uses the value from the previous seasonal period to forecast the future.
- **Drift Method:** Predicts future values based on the trend between the first and last observations in the training data
```



Forecasting_Exponential_Smoothing.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/TimeSeries/Forecasting_Exponential_Smoothing.py

Exponential smoothing models are a set of Time Series Forecasting techniques that apply weighted averages of past observations, with the weights decaying exponentially over time. These methods are useful for capturing different components of time series data, such as level, trend, and seasonality.

However, their effectiveness depends on the nature of the data. For Datasets with simple patterns, these models can be quite effective, but for more complex series, alternative methods may be necessary.

Methods Implemented

Simple Exponential Smoothing (SES):

- Description: Suitable for forecasting data without trends or seasonality. It applies a constant smoothing factor to past observations.
- Use Case: Best for stationary series where the data fluctuates around a constant mean.

Double Exponential Smoothing (Holt's Linear Trend):

Introduction

- Description: Extends SES by adding a trend component, allowing it to model data with a linear trend.
 - Use Case: Ideal for series with a consistent upward or downward trend but no seasonality.
-

Triple Exponential Smoothing (Holt-Winters):

- Description: Incorporates both trend and seasonal components, making it suitable for data with both linear trends and seasonal patterns.
- Use Case: Effective for series with regular seasonal fluctuations.

Advanced Alternatives: For complex datasets like stock prices, advanced models such as [Forecasting_AutoArima.py](#) may be more appropriate to capture the intricacies of the data.

Foreign Key

A foreign key is a field in one table that uniquely identifies a row in another table, linking to the primary key of that table.

For example, `DepartmentID` in the [Employees](#) table links to `DepartmentID` in the [Departments](#) table.

Foreign keys establish relationships between tables and maintain referential integrity by ensuring valid connections between records.

Departments Table

DepartmentID	DepartmentName
1	Human Resources
2	IT
3	Marketing

Employees Table

EmployeeID	EmployeeName	DepartmentID
101	Alice	1
102	Bob	2
103	Charlie	1
104	Dana	3

Forward Propagation

[!Summary]

Forward propagation is the process by which input data moves through a neural network, layer by layer, to produce an output. During this process, each layer's weights and biases are applied to the input data, and an activation function is used to transform the data at each layer.

Mathematically, for each layer, the input x is transformed into an output y through the equation $y = f(Wx + b)$, where W represents the weights, b is the bias, and f is the activation function (e.g., ReLU, sigmoid). The output from one layer becomes the input to the next, and this continues until the final layer produces the predicted output.

This process does not involve learning; it only **computes the prediction based on current weights**.

[!Breakdown]

Key Components:

- Input data: Initial values fed into the network.
- Weights (W) and biases (b): Parameters adjusted during training.
- Activation function: Non-linear transformation, e.g., ReLU or sigmoid.
- Output: Prediction made by the network.

[!important]

- Forward propagation calculates predictions **by applying current model parameters** to inputs.
- It is the first step before backpropagation, where the error is used to adjust weights.

[!attention]

- Forward propagation does not involve **learning or updating weights**.
- The accuracy of forward propagation depends entirely on the current values of weights and biases.

[!Example]

In a simple neural network with one hidden layer, forward propagation can be described as:

$$z_1 = W_1x + b_1$$

$$a_1 = \text{ReLU}(z_1)$$

$$z_2 = W_2 a_1 + b_2$$

$$y = \text{sigmoid}(z_2)$$

Here, x is the input, and y is the output prediction.

[!Follow up questions]

- How does the choice of **activation function** impact the forward propagation process?
- In deep networks, how can **vanishing and exploding gradients problem** during forward propagation affect training?

[!Related Topics]

- **Backpropagation** in neural networks
- Activation functions in deep learning

Fuzzywuzzy

Tool used for correcting spelling with pandas.

[Data Cleansing](#)

Filter Methods

For Feature Selection

1. Pearson Correlation Coefficient:

- Measures the linear correlation between two continuous variables.
- Features with low correlation with the target variable are considered less relevant.
- Features with high correlation among themselves might be redundant.

2. Mutual Information:

- Measures the amount of information obtained about one variable through another variable.
- High mutual information indicates strong dependency between features and the target variable.
- Can handle both continuous and categorical variables.
- used to rank or score features based on their relevance to the target variable.
- joint probability distribution
- information theory,

3. ANOVA (Analysis of Variance):

- Assesses the differences in means among groups of a categorical variable.
- Calculates the F-statistic and p-value to determine if there are significant differences in the means of the target variable across different levels of a categorical feature.
- Useful for selecting features with significant impact on the target variable in classification tasks.

4. Chi-Squared Test:

- Tests the independence between two [categorical](#) variables.
- Calculates the chi-squared statistic and p-value to determine if the observed frequency distribution differs from the expected distribution.
- Helpful for [Feature Selection](#) in classification tasks with categorical variables.

Functional Programming

Functional Programming is a style of building functions that threaten computation as a mathematical function that avoids changing state and mutable data. It is a declarative programming paradigm, which means programming expressive and [declarative](#) as opposed to imperative. It's getting more popular with the rise of [Functional Data Engineering](#).

See also [Programming Languages](#).

G

Table of Contents

- GIS
- GRU
- GSheets
- Gaussian Distribution
- Gaussian Mixture Models
- Gaussian Model
- Gaussian_Mixture_Model_Implementation.py
- General Linear Regression
- Generative AI From Theory to Practice
- Generative AI
- Generative Adversarial Networks
- Get data
- Gini Impurity vs Cross Entropy
- Gini Impurity
- Git
- Gitlab
- Google Cloud Platform
- Google My Maps Data Extraction
- Gradient Boosting Regressor
- Gradient Boosting
- Gradient Descent
- Gradio
- Grain
- Grammar method
- Graph Analysis Plugin
- Graph Neural Network
- Graph Theory Community
- Graph Theory
- GraphRAG
- Grep
- GridSearchCV
- Groupby vs Crosstab
- Groupby
- Grouped plots
- Guardrails
- gitlab-ci.yml
- granularity

Gis

Geographic information system.

File formats:

Introduction

The Web Map Tile Service (WMTS) and Web Feature Server (WFS) are both specifications used in the field of Geographic Information Systems (GIS) to serve different types of geographic data over the web. The primary differences between them lie in the type of data they serve and how they serve it.

```
[Web Map Tile Service (WMTS)](#web-map-tile-service-wmts)
[Web Feature Server (WFS)](#web-feature-server-wfs)
[Key Differences of Web Feature Server (WFS) and Web Feature Server (WFS)](#key-differences-of-web-feature-server-wfs-and-
```

[shapefile](#)

There are free GIS softwares

Gru

Gsheets

Useful functions:

- [QUERY GSheets](#)
- [ARRAYFORMULA](#)
- Indirect

Accessing google sheets from a script: <https://www.youtube.com/watch?v=zCEJurlGFRk>

Gaussian Distribution

Common assumption for a [Distributions](#).

Gaussian Mixture Models

Gaussian Mixture Models (GMMs) represent data as a mixture of multiple Gaussian [distributions](#), with each cluster corresponding to a different Gaussian component. GMMs are more effective than [K-means](#) because they consider the distributions of the data rather than relying solely on distance metrics.

Soft Clustering technique.

In [ML_Tools](#) see: [Gaussian_Mixture_Model_Implementation.py](#)

[Kmeans vs GMM](#)

GMMs can have difference [Covariance Structures](#)

Key Concepts

- **Gaussian Components:** Each Gaussian distribution is characterized by its mean and [Covariance](#).
- **Likelihood:** The likelihood of a data point belonging to a cluster is given by the formula:

$$P(X|C_k) = \pi_k \cdot \mathcal{N}(X|\mu_k, \Sigma_k)$$

where $P(X | C_k)$ is the probability of data point X given cluster C_k , π_k is the prior probability of cluster C_k , and \mathcal{N} is the Gaussian distribution.

- **Expectation-Maximization (EM) Algorithm:** GMMs utilize the EM algorithm to iteratively optimize the parameters of the Gaussian components.

Advantages of GMMs

- **Complex Data Distributions:** GMMs can capture complex data distributions, unlike [K-means](#), which only considers distance metrics.
- **Probabilistic Framework:** GMMs provide a probabilistic framework for clustering, allowing for soft assignments of data points to clusters.
- **Modeling Elliptical Clusters:** The use of covariance matrices enables GMMs to model elliptical clusters, enhancing clustering performance.

Applications

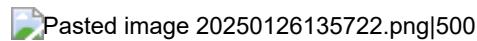
- [Anomaly Detection](#): GMMs are widely used in various applications, including anomaly detection.

Important Considerations

- **Covariance Types:** The choice of covariance types (full, tied, diagonal, spherical) can significantly impact the performance of GMMs.

Follow-up Questions

- How do GMMs compare to other clustering algorithms in terms of scalability and computational efficiency?
- What are the implications of choosing different covariance types in GMMs?



Gaussian Model

(Univariate)

- **Formula:**

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- **Steps:**

- Estimate μ and σ^2 from the data.
- Compute the probability density for each data point.
- Points with low probabilities (below a threshold ϵ) are considered anomalies.



5. Multivariate Gaussian Distribution

- **Steps:**

Introduction

- Extend the Gaussian model to include covariance across features.
 - Fit the multivariate Gaussian model:
-

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

- μ : Mean vector
- Σ : Covariance matrix

- Threshold low-probability examples to identify anomalies.

Gaussian_Mixture_Model_Implementation.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/Clustering/Gaussian_Mixture_Model_Implementation.py

Follow-Up Questions

- How do GMMs compare to other clustering algorithms in terms of scalability and computational efficiency?
- What are the implications of choosing different covariance types in GMMs?

General Linear Regression

[Linear Regression](#)

[t-test](#) - to compare means between two populations.

[ANOVA](#) - tests

Objective:

How do LLMs work and operate.

Enabling [LLM](#)'s at scale: Explore recent AI and Generative AI language models

Steps

Math on words: Turn words into coordinates. Statistics on words: Given context what is the probability of what's next. Vectors on words. Cosine similarity. How train: Use [Markov chain](#) for prediction of the next [Tokenisation](#)

Tokeniser: map from token to number

1. Pre-training: tokenise input using [NLP](#) techniques
2. [LLM](#) looks at context: nearby tokens, in order to predict

different implementations for different languages. Different tokenisers or translating after.

Journey to scale:

1. Demos, POC (plan to scale): understand limitations
 2. Beyond experiments and before production:
-

3. Enterprise level: translate terms so they can use governess techniques.

Building:

Buy, Build or Boost?

	Buy – API-Based Consumption	Boost – Fine-tuning or Open-source	Build – Creating our own LLM?
What?	<ul style="list-style-type: none"> OpenAI, Google, AWS consumption of LLMs via an endpoint. Send prompt + parameters via HTTP or other method, receive output. 	<ul style="list-style-type: none"> Use of either a fine-tuning API or a self-hosted model and using a smaller dataset, alongside techniques like LoRA, qLoRA to produce a model more suitable for a specific task. 	<ul style="list-style-type: none"> Fully building your own model based on your own training pipeline and input data stack. What OpenAI, Anthropic, Google, Meta have all done and provide via the various services.
Pros	<ul style="list-style-type: none"> Fast and stable Does not require infrastructure handling Can be launched in private cloud Predictable cost based on dollars/1k token 	<ul style="list-style-type: none"> Can tailor and train model to give better outputs for a given task, cheaper than full training. If self-hosted, creates a model asset no other competitor would have. Can form part of an RLHF training pipeline to further improve (Reinforcement Learning with Human Feedback) 	<ul style="list-style-type: none"> Fully tailorabile. Powerful if done properly. Creates an asset which you own (though the input data provenance may also determine ownership – still an open legal question in the world!)
Cons	<ul style="list-style-type: none"> Minimal ability to adapt or retrain models Do not own model asset May require client trust in 3rd party (even though data does not leave VPC) Costs can spiral in scaled systems 	<ul style="list-style-type: none"> Requires GPU compute resources and a technical knowledge on how to train. Results not guaranteed (depends on input data) If using an API-based provider like OpenAI, typically only older models can be fine-tuned. 	<ul style="list-style-type: none"> Highly expensive (\$10s of millions of dollars +) Highly detailed data science knowledge required. If it goes wrong, wasted cost. Your competition is OpenAI, Meta, Google et al...
Examples	<ul style="list-style-type: none"> OpenAI's ChatGPT is an example of an implementation of this API (sits on top of GPT-3.5 / GPT-4). 	<ul style="list-style-type: none"> ChatGPT deploys a type of RLHF training pipeline based on user input, but only OpenAI is familiar with this. Code Assistant models such as Github Copilot, TabNine and Cursor are fine-tuned on coding-specific sources such as public Github repos and StackOverflow posts. 	<ul style="list-style-type: none"> Only really recommended for companies that actually have a reason or an advantage, as well as the capital and skillsets to do so. Think Apple because of their unique hardware advantage in the mobile market, or Bloomberg due to their wealth of financial data.

Software Development Life Cycle

For GenAI: Building an applicaiton with GenAi features

1. Plan: use case: prompts : architecture: cloud or on site
2. Build: vector database
3. Test: Quality and responsible ai.

call summarisation

take transcript -> summariser -> summarise

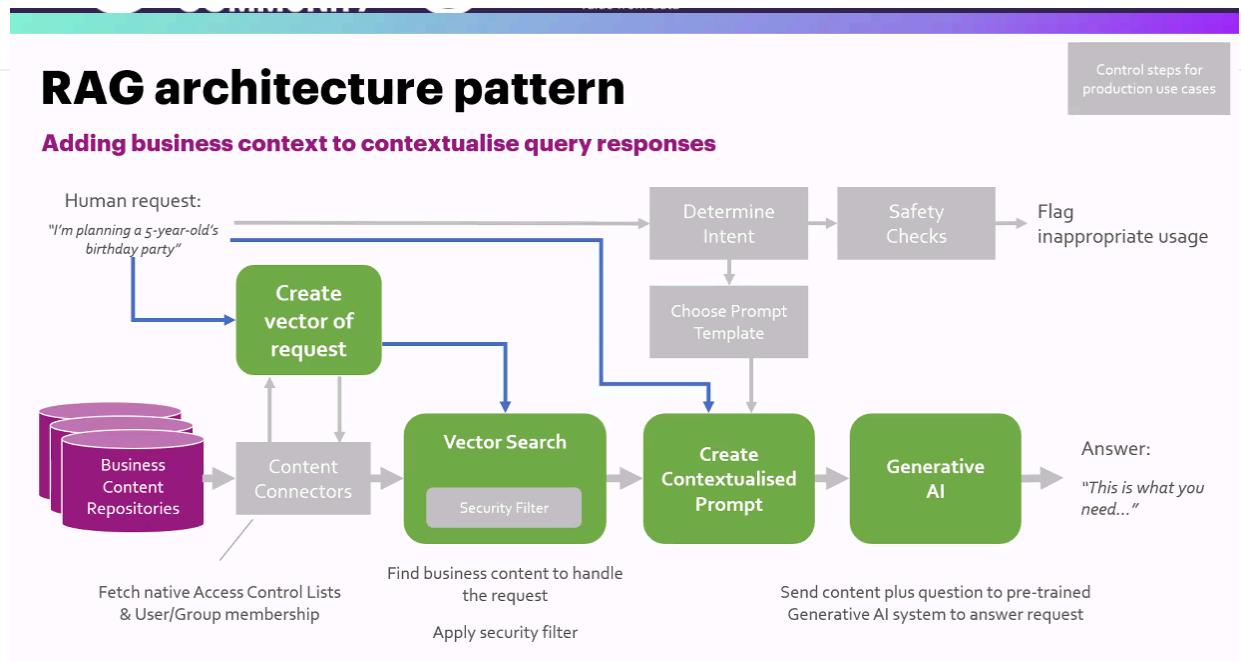
Source: human labeled transcripts to check summariser.



Ngrams analysis - when specific words realy matter

RAG

Use relvant data to make response better:



GAN

For image models.

Examples: midjourney,stable diffusion,dall-e 3

image model techniques:

- text to image
 - image to image

Notes:

Use LLM's to get short info, then cluster. Going round training data : called a Epochs

Generative Ai

Generative Adversarial Networks

Composed of two neural networks, a generator, and a discriminator, that compete against each other. GANs are used for tasks like generating realistic images or videos.

Get Data

What is involved:

```
df = pd.read_csv('Categorical.csv')
```

Introduction

- Gather relevant data from appropriate sources, addressing any quality or privacy concerns.

```
## Get textbook data using for example:

import re

def read_file(filename):
    with open(filename, "r", encoding='UTF-8') as file:
        contents = file.read().replace('\n\n', ' ').replace('[edit]', '').replace('\ufe0f', '').replace('\n', ' ')
    return contents

text = read_file('Data/Various/Monte_Cristo.txt')

text_start = [m.start() for m in re.finditer('VOLUME ONE', text)]
text_end = [m.start() for m in re.finditer('End of Project Gutenberg', text)]
text = text[text_start[1]:text_end[0]]
```

How would you approach a colleague who is hesitant to share their data? ?

- explain the purpose and benefits
- ensure confidentiality (GDPR) with data masking.
- and finding common ground to address any concerns or objections.
- build trust.
- make agreements of terms of use/ownership/document the data accessing process.

How would you go about obtaining the necessary permissions for a dataset? ?

- establishing clear communication channels within the organisation.
- obtaining necessary approvals
- emphasizing the value of collaboration.

How would you gather sensitive data?; Get consent. Ensure anonyminity (follow regularions)

How to you ensure data is unbiased and representative. ?

- Stratified sampling, (group then randomly sample).
- Examine the data sources.

Gini Impurity Vs Cross Entropy

When working with decision trees, both [Gini Impurity](#) and [Cross Entropy](#) are metrics used to evaluate the quality of a split. They help determine how well a feature separates the classes in a dataset.

Gini Impurity

- **Definition:** Gini impurity measures the probability of incorrectly classifying a randomly chosen element if it was randomly labeled according to the distribution of labels in the node.
- **Computation:** Generally faster to compute than cross-entropy because it does not involve logarithms.
- **Use Case:** Often used in the CART (Classification and Regression Trees) algorithm. It is a good default choice for classification tasks due to its simplicity and efficiency.

Cross Entropy (More refined than impurity)

- **Definition:** Cross-entropy measures the amount of information needed to encode the class distribution of the node. It quantifies the expected amount of information required to classify a new instance.
- **Computation:** Involves logarithmic calculations, which can be computationally more intensive than Gini impurity.
- **Use Case:** Often used in algorithms like ID3 and C4.5. It can be more informative in cases where the class [Distributions](#) is skewed or when you need a more nuanced measure of impurity.

Choosing Between Gini Impurity and Cross Entropy

- **Performance:** In practice, both metrics often lead to similar results in terms of the structure and performance of the decision tree. The choice between them may not significantly affect the final model.
- **Efficiency:** If computational efficiency is a concern, Gini impurity might be preferred due to its simpler calculation.
- **Interpretability:** Cross-entropy provides a more information-theoretic perspective, which might be preferred if you are interested in the information gain aspect of the splits.

Gini Impurity

Gini impurity is a metric used in decision trees to measure the degree or probability of misclassification in a dataset. It is associated with the leaves of a [Decision Tree](#) and helps determine the best split at each node.

Calculation

- Mathematical Formula: Gini impurity is calculated as the probability of incorrectly classifying a randomly chosen element if it were randomly labelled according to the distribution of labels in the subset.
- Formula:

$$\text{Gini Impurity} = 1 - \sum_{i=1}^n p_i^2 \text{ where } p_i \text{ is the probability of an element being classified into a particular class.}$$

Usage

- **Decision Trees:** Gini impurity is commonly used in decision trees to evaluate splits. A lower Gini impurity indicates a better split, as it means the data is more homogeneously classified.
- **Classification Tasks:** It is particularly useful in classification tasks where the goal is to minimize misclassification.

Relationship to Other Metrics

- Gini impurity is one of several [Regression Metrics](#) used to evaluate the performance of decision trees, alongside others like entropy.

Example

Suppose you have a dataset with a binary classification problem, where the target variable can be either "Yes" or "No". You have a node in your decision tree with the following distribution of classes:

- 10 samples labeled "Yes"
 - 5 samples labeled "No"
-

Gini Impurity Calculation

The formula for Gini impurity is:

$$\text{Gini impurity} = 1 - \sum(p_i^2)$$

where p_i is the proportion of class i in the node.

Step-by-Step Calculation

1. Calculate the proportion of each class:

- Total samples = 10 (Yes) + 5 (No) = 15
- Proportion of "Yes" = $\frac{10}{15} = 0.67$
- Proportion of "No" = $\frac{5}{15} = 0.33$

2. Calculate the squared proportions:

- $(0.67)^2 = 0.4489$
- $(0.33)^2 = 0.1089$

3. Sum the squared proportions:

- Sum = $0.4489 + 0.1089 = 0.5578$

4. Calculate the Gini impurity:

- Gini impurity = $1 - 0.5578 = 0.4422$

A Gini impurity of 0.4422 indicates the level of impurity in this node. A Gini impurity of 0 would mean the node is pure (all samples belong to one class), while a higher value indicates more impurity or mixed classes.

This calculation helps in deciding whether to split the node further or not. The goal is to choose splits that minimize the Gini impurity, leading to more homogeneous branches.

Git

tags:

- software

Do git bash here.

git status

git add . (adds all)

git status

git commit -m ""

git push

Notes

<https://www.youtube.com/watch?v=xnR0dlOqNVE>

Examples

Git: Common Issues and Fixes

Git can be frustrating, especially when things go wrong. This guide provides practical solutions to common Git mistakes, explained in simple terms.

<https://ohshitgit.com/>

how to remove something from a git history, if i forgot to add it to the .gitignore, but now have

1. Remove the file from the Git index This tells Git to stop tracking the file.

git rm --cached path/to/file For a folder: git rm -r --cached path/to/folder

1. Commit this change This saves the removal from the index.

bash Copy code git commit -m "Stop tracking path/to/file and add to .gitignore"

Undoing Mistakes

I messed up badly! Can I go back in time?

Yes! Use Git's reflog to find a previous state:

```
git reflog  
# Find the index of the state before things broke  
git reset HEAD@{index}
```

This is useful for recovering deleted commits, undoing bad merges, or rolling back to a working state.

Commit Fixes

I committed but forgot a small change!

```
# Make the change  
git add .  
git commit --amend --no-edit
```

⚠ Warning: Never amend a commit that has already been pushed!

I need to change the last commit message!

```
git commit --amend
```

This will open an editor where you can modify the commit message.

🔗 Branching Issues

I committed to `master` but wanted a new branch!

```
# Create a new branch from the current state  
git branch new-branch  
# Remove the commit from master  
git reset HEAD~ --hard  
git checkout new-branch
```

⚠️ Warning: If you've already pushed the commit, additional steps are needed.

I committed to the wrong branch!

```
# Undo the last commit but keep the changes  
git reset HEAD~ --soft  
git stash  
git checkout correct-branch  
git stash pop  
git add .  
git commit -m "Moved commit to correct branch"
```

Alternative:

```
git checkout correct-branch  
git cherry-pick master # Moves last commit to correct branch  
git checkout master  
git reset HEAD~ --hard # Removes the commit from master
```

🔍 Diff and Reset

I ran `git diff`, but it showed nothing!

If your changes are staged, use:

```
git diff --staged
```

This shows differences between the last commit and staged files.

I need to undo a commit from 5 commits ago!

```
git log # Find the commit hash  
git revert [commit-hash]
```

This creates a new commit that undoes the changes.

Undoing Changes

I need to undo changes to a file!

```
git log # Find a commit before the changes  
git checkout [commit-hash] -- path/to/file  
git commit -m "Reverted file to previous version"
```

I want to reset my repo to match the remote!

 *Destructive action—this cannot be undone!*

```
git fetch origin  
git checkout master  
git reset --hard origin/master  
git clean -d --force # Removes untracked files
```

Last Resort

If everything is completely broken, nuke the repo and reclone:

```
cd ..  
sudo rm -r repo-folder  
git clone https://github.com/user/repo.git  
cd repo-folder
```

Gitlab

[GitLab CI CD Tutorial for Beginners Crash Course](#)

- Provides managed runners to execute CI-CD pipelines.
- Integrates with version control systems to automate the CI/CD process.

Google Cloud Platform

Google Cloud Platform is a suite of cloud computing services offered by Google. It provides a range of services including computing, storage, and application development that run on Google hardware.

Resources: [Introduction to Google Cloud](#)

Compute Engine

Description: GCP's Infrastructure as a Service (IaaS) offering, allowing users to run virtual machines on Google's infrastructure.

Features: Custom Machine Types: Create VMs with custom configurations. Preemptible VMs: Costeffective, shortlived instances for batch jobs and faulttolerant workloads. Sustained Use Discounts: Automatic discounts for prolonged usage. Persecond Billing: Charges calculated per second for cost savings.

Use Cases: Suitable for web hosting, data processing, and largescale applications. Integration: Works seamlessly with other GCP services like Google [Kubernetes Engine](#), [Cloud Storage](#), and [BigQuery](#).

Bigtable

A scalable [NoSQL](#) database service for large analytical and operational workloads.

App Engine

A platform for building scalable web applications and mobile backends.

BigQuery

A fullymanaged, serverless data warehouse for largescale data analytics.

Cloud Storage

Object storage service for storing and accessing data on Google's infrastructure.

Cloud SQL

Managed relational database service for [MySQL](#), PostgreSQL, and SQL Server.

CI/CD

Tools and services for continuous integration and continuous delivery.

standardised/Firebase

A platform for building mobile and web applications with realtime databases, authentication, and more.

Notes:

Consider setting up a personal GCP example for handson experience. Explore the generic repository for additional resources and examples.

Google My Maps Data Extraction

Summary:

This guide covers the key workflows and tools for managing and processing location data in Google Sheets and Google My Maps. Suppose we have marks on Google My Maps. In order to extract the location of markers to a google sheet.

1. **Export Marker Data** from Google My Maps as KML/CSV.
2. Convert KML to CSV
3. **Use Apps Script in Google Sheets** to extract data like addresses or postal codes from coordinates.

Extract Data from Google My Maps

- **Export Custom Markers:**
 1. Open Google My Maps.
 2. Use the menu (three dots) to select **Export to KML**.
 3. The exported file will contain marker names, descriptions.

Extract KML data to google sheets

- Rename KML file to XML.
- Open XML in excel.
- Extract marker and coordinate data.
- Paste data into google sheets.

Extracting Information from Coordinates in Google Sheets

Use [Google Apps Script](#) to extract additional information like addresses or postal codes from geographic coordinates.

- **Get Address from Coordinates:**

```
function getAddress(lat, lng) {
  var response = Maps.newGeocoder().reverseGeocode(lat, lng);
  var result = response.results[0];
  if (result) {
    return result.formatted_address;
  } else {
    return 'No address found';
  }
}
```

- **Get Postal Code from Coordinates:**

```

function getPostalCode(lat, lng) {
  var response = Maps.newGeocoder().reverseGeocode(lat, lng);
  var result = response.results[0];
  if (result) {
    for (var i = 0; i < result.address_components.length; i++) {
      var component = result.address_components[i];
      if (component.types.indexOf('postal_code') !== -1) {
        return component.long_name;
      }
    }
    return 'Postal code not found';
  } else {
    return 'No results found';
  }
}

```

- Use `getAddress()` with `getPostalCode()` .
- `=getPostalCode(56.033139, -3.4182519)`

Gradient Boosting Regressor

<https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor>

Boosting

The `GradientBoostingRegressor` from the `sklearn.ensemble` module is a model used for regression tasks. It builds an **Model Ensemble** of **Decision Tree** in a sequential manner, where each tree tries to correct the errors made **by the previous ones**. Here's a breakdown of the key parameters:

1. **loss**: Specifies the loss function to optimize. Default is `'squared_error'`, which is the least-squares loss function. Other options like `'absolute_error'` can be used for robustness against outliers.
2. **learning_rate**: Controls the contribution of each tree to the final prediction. A smaller value (e.g., 0.01) makes the model learn more slowly, but it can lead to better generalization. Default is 0.1.
3. **n_estimators**: The number of boosting stages (i.e., trees). More trees can improve performance but also increase the risk of overfitting. Default is 100.
4. **subsample**: The fraction of samples to be used for fitting each tree. Setting this to a value less than 1.0 can help reduce overfitting, at the cost of a slight increase in bias. Default is 1.0 (use all samples).
5. **criterion**: The function used to measure the quality of a split. `'friedman_mse'` is the default, which is an improved version of mean squared error for decision trees. Other options include `'mse'` and `'mae'`.
6. **max_depth**: The maximum depth of the individual trees. This parameter controls the complexity of each tree. Default is 3, which typically works well for most tasks.
7. **min_samples_split**: The minimum number of samples required to split an internal node. Default is 2, meaning any node can be split as long as there are at least 2 samples.
8. **min_samples_leaf**: The minimum number of samples required to be at a leaf node. This helps control overfitting by requiring more data points at each leaf. Default is 1.

-
- 9. **alpha**: The quantile used for the loss function in cases of robust regression. This is useful when dealing with data that includes outliers. Default is 0.9.
 - 10. **validation_fraction**: The fraction of training data to set aside for validation to monitor performance during training. Default is 0.1.
 - 11. **n_iter_no_change**: The number of iterations with no improvement on the validation score to wait before stopping the training early. Default is `None`, meaning no early stopping.
 - 12. **ccp_alpha**: Complexity parameter used for pruning the trees. A larger value leads to more pruning (simplifying the model), which can help prevent overfitting.

Gradient Boosting

Gradient Boosting is a technique used for building predictive models [Model Building](#), particularly in tasks like regression and classification. It combines the concepts of [Boosting](#) and [Gradient Descent](#) to create strong models by sequentially combining multiple [Weak Learners](#) ([Decision Tree](#)).

Key Idea: Instead of fitting a single strong model, Gradient Boosting builds multiple weak learners sequentially. Each new model focuses on [correcting the mistakes made by the previous ones](#) by fitting to the residuals (differences between observed and predicted values).

Gradient Boosting builds an ensemble of [Weak Learners](#) (usually [Decision Tree](#)) sequentially. Each new model focuses on the errors of the previous ones, aiming to minimize the residual errors.

Final Prediction: The final prediction is made by aggregating the predictions of all the weak models, usually through a weighted sum.

High Performance: Known for its high performance and efficiency in terms of speed and memory usage.

[Watch Video Explanation](#)

[Model Ensemble](#)

Key Components

- [Weak Learners](#): Typically decision trees used in the ensemble.
- [Loss Function](#): Measures how well the model fits the data.
- [learning rate](#): Controls the contribution of each weak learner to the final model.

Examples

- [LightGBM](#)
- [XGBoost](#)
- [CatBoost](#)

Benefits

- **Predictive Accuracy:** Often outperforms other [Machine Learning Algorithms](#).
- **Feature Handling:** Effectively manages [heterogeneous features](#) and automatically selects relevant ones.
- [Overfitting](#): Less prone to overfitting compared to other complex models.

Gradient Descent

Gradient descent is an [Optimisation function](#) used to minimize errors in a model by adjusting its parameters iteratively. It works by moving in the direction of the steepest decrease of the [Loss function](#).

Uses the difference quotient.

The step size is important between derivatives (small then slow) (if large then might miss minimum).

With Stochastic method we can don't need to the entire data set again, we can just add the new information to get improvement.

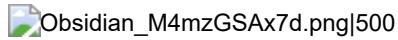
Gradient descent uses the entire data set.

Used to find the min/max of [Cost Function](#).

Given any point on the cost function surfaces. Then ask, "In what direction should I go to make the biggest change downhill or up hill, i.e. gradient descent"



How do you implement Gradient descent? You update the (direction) parameter by the small step by the [learning rate](#).



Stochastic Gradient Descent

Stochastic uses random entries to get derivative instead of the full dataset Why do we use [Stochastic Gradient Descent](#)?;; To find the derivative of discrete data so we can determine a straight line with the Least Square Error (LSE). What is [Stochastic Gradient Descent](#)?;; updates the model parameters based on the gradient of a single randomly chosen data point.

Batch gradient descent

What is [Batch gradient descent](#)?;; computes the gradient of the entire dataset,

Mini-batch gradient descent

Stochastic Mini-batched descent is the fastest way (groups then does randomly). What is [Mini-batch gradient descent](#)?;; Is a compromise of [Batch gradient descent](#) and [Stochastic Gradient Descent](#).

What is the difference between batch gradient descent and stochastic gradient descent?;; Batch gradient descent computes the gradient of the cost function using the entire training dataset in each iteration, while stochastic gradient descent updates the model's parameters based on the gradient of the cost function with respect to one training example at a time. Mini-batch gradient descent is a compromise, using a subset of the training data in each iteration.

Gradient Descent

Gradient descent is commonly used in:

- **Deep Learning:** Frameworks like TensorFlow and PyTorch use variations of gradient descent for training.

- **Custom Implementations:** If you write logistic regression from scratch, gradient descent is a straightforward optimization method.

How Gradient Descent Works

Gradient Descent, a common [Optimisation techniques](#), iteratively updates the [Model Parameters](#) by computing the gradient of the [loss function](#) with respect to the parameters. The update formula is:

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta)$$

Where:

- θ are the parameters (intercept and coefficients).
- α is the learning rate (step size for updates).
- $\nabla_{\theta} J(\theta)$ is the gradient of the [cost function](#) with respect to the parameters θ .

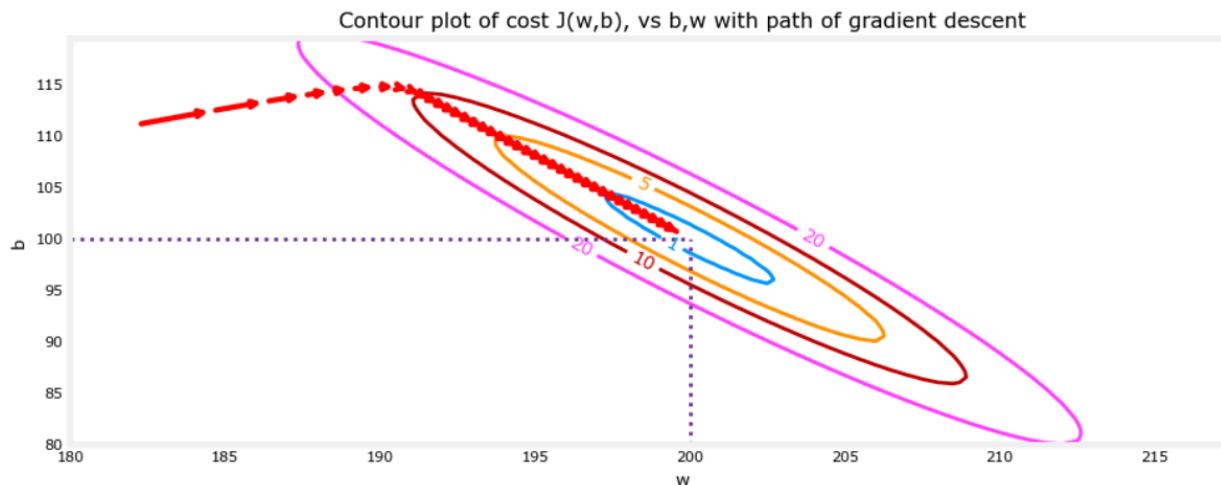
Process:

1. Calculate the gradient of the loss function.
2. Adjust the parameters in the direction of the negative gradient (to reduce loss).
3. Repeat until either:
 - o The loss function converges (minimal change between updates), or
 - o The maximum number of iterations is reached.

Gradient Descent

[Cost Function](#) value versus number of iterations of [Gradient Descent](#) should decrease

Can use contour plots to show [Gradient Descent#](#) moving towards minima.



Gradio

Gradio is an open-source platform that simplifies the process of [creating user interfaces](#) for machine learning models.

It allows users to quickly build interactive demos and applications for their models without extensive front-end development knowledge.

Main uses:

- **Interactive Interfaces:** Gradio provides a simple way to create web-based interfaces where users can interact with machine learning models by uploading files, entering text, or adjusting sliders.
- **Rapid Prototyping:** It enables quick prototyping and sharing of machine learning models, making it easier to demonstrate model capabilities to stakeholders or gather user feedback.
- **Ease of Integration:** Gradio can be easily integrated with popular machine learning frameworks like TensorFlow, PyTorch, and Hugging Face Transformers, allowing seamless deployment of models.

Related content

Video Link <https://www.gradio.app/>

Overview

Grain

Grain

- Definition: The level of detail or [granularity](#) of the data stored in the fact table.
- Importance: Defining the grain is crucial as it determines what each record in the fact table represents (e.g., individual transactions, daily summaries).

Grammar Method

can understand the Grammar as a method for acceptable sentences.

Graph Analysis Plugin

Graph Neural Network

Resources:

- [How Graph Neural Networks Are Transforming Industries](#)

Use cases:

- [Recommender systems](#) i.e. Uber, Pinterest (PinSage)
- Traffic Prediction - Deepmind in google maps
- Weather forecasting - GraphCast - Deepmind
- Data Mining - Relational Deep Learning
- Material Science - Deepmind - GNome - Density function theory.
- Drug Discovery - MIT - antibiotic activities

Graph Theory Community

In graph theory, a community (also known as a cluster or module) is a group of nodes that are more densely connected to each other than to the rest of the network.

graph_analysis #clustering

https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.louvain.louvain_communities.html

Intuition

Communities often represent:

- Functional units in biological networks (e.g., protein complexes)
- Groups of friends or followers in social networks
- Topical clusters in knowledge graphs or citation networks

They capture meso-scale structure—between the local (node/edge) and global (graph-level) scale.

Formal Definition

There is no single universal definition, but communities typically exhibit:

- High intra-community density: lots of edges within the group
- Low inter-community density: few edges connecting to other groups

Mathematically, a common goal is to maximize modularity, a measure that quantifies the density of links inside communities compared to links between them.

Community Detection Algorithms

Some widely used algorithms:

Algorithm	Description
Louvain	Fast and widely used; optimizes modularity
Girvan–Newman	Based on removing high-betweenness edges
Label Propagation	Propagates labels through the network
Leiden	Improved version of Louvain for better quality and performance

Graph Theory

[Graph Theory Community](#)

[Page Rank](#)

[PyGraphviz](#)

[networkx](#)

[Plotly](#) for graphs <https://plotly.com/python/network-graphs/>

Graphrag

[GraphRAG](#) is a [RAG](#) framework that utilizes [Knowledge Graphs](#) to enhance information retrieval and processing. A significant aspect of this framework is the use of large language models (LLMs) for [Named Entity Recognition](#) (NER) within [Neo4j](#).

[Graph Neural Network](#)

Related Terms

- [How to search within a graph](#)
- **Text2Cypher:** This feature allows users to interact with the graph in a user-friendly manner, converting natural language queries into Cypher queries.
- How to move datasets into a graph database.
- Graphrag patterns.
- The role of [interpretability](#) in understanding graph-based retrieval.

Implementation

I discovered an insightful LinkedIn post discussing the potential of knowledge graphs: This specific graph is called a "Lexical Graph with Extracted Entities". [LinkedIn Post In ML_Tools](#) see: [Wikipedia_API.py](#)

Resources

- [GraphRAG Site](#)
- [Neo4j: Building Better GenAI: Your Intro to RAG & Graphs](#)

Grep

```

bash - refactorai_app
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG (refactoring_prep) $ cd notebooks/
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks (refactoring_prep) $ cd refactorai_app/
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks/refactorai_app (refactoring_prep) $ grep -r 'save_to_file'
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks/refactorai_app (refactoring_prep) $ grep -r 'call_api'
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks/refactorai_app (refactoring_prep) $ grep -r 'refactor'
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks/refactorai_app (refactoring_prep) $ cd refactoring/
(venv) @hyslwells →.../S2DS-Spring24-TeamCGG/notebooks/refactorai_app/refactoring (refactoring_prep) $ grep -r 'refactor'
(venv) @hyslwells →.../S2DS-Spring24-TeamCGG/notebooks/refactorai_app/refactoring (refactoring_prep) $ grep -r 'call_api'
(venv) @hyslwells →.../S2DS-Spring24-TeamCGG/notebooks/refactorai_app/refactoring (refactoring_prep) $ grep -r "call_api"
refactoring_cycle.py:    response = API.call_api_contents(file, template, code_smells)
refactoring_cycle.py:    #TODO! Add Report_summary_content element to (output of call_api function).
grep: __pycache__/_refactoring_cycle.cpython-311.pyc: binary file matches
(venv) @hyslwells →.../S2DS-Spring24-TeamCGG/notebooks/refactorai_app/refactoring (refactoring_prep) $ cd..
bash: cd..: command not found
(venv) @hyslwells →.../S2DS-Spring24-TeamCGG/notebooks/refactorai_app/refactoring (refactoring_prep) $ cd ..
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks/refactorai_app (refactoring_prep) $ grep -r "save_to_file"
src/api_handling.py:# def save_to_file(content,name):
src/api_handling.py:#     save_to_file(output_string, "test")
grep: src/__pycache__/api_handling.cpython-311.pyc: binary file matches
(venv) @hyslwells →/workspaces/S2DS-Spring24-TeamCGG/notebooks/refactorai_app (refactoring_prep) $ 

```

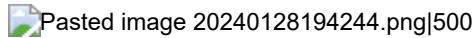
Gridsearchcv

Used `GridSearchCV` to search through the `Hyperparameter` space

```
rf_regressor = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf_regressor, param_grid=param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_

# Model Training with best hyperparameters
rf_regressor = RandomForestRegressor(**best_params, random_state=42)
rf_regressor.fit(X_train, y_train)
```

Given a parameter grid of `Hyperparameter`, a model, then you model it on the hypers, then gives you the best hypers, that gives the highest cross validation performance.



Groupby Vs Crosstab

In pandas, `Groupby` and `Crosstab` serve related but distinct purposes for data `aggregation` and summarization.

- `groupby` is more flexible for aggregation and transformations,
- whereas `crosstab` is specifically designed for creating frequency tables and exploring the relationship between categorical variables.

In `DE_Tools` see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/reshaping.ipynb

Key Differences

- **Purpose:**
 - `groupby` : Used for performing aggregate functions (sum, mean, count, etc.) on grouped data.
 - `crosstab` : Used for generating frequency tables or contingency tables.
- **Output:**
 - `groupby` : Returns a DataFrame with aggregated values.
 - `crosstab` : Returns a DataFrame with counts or specified aggregation functions applied across two or more columns.
- **Usage:**
 - `groupby` : Can be used with multiple aggregation functions and complex groupings.
 - `crosstab` : Typically used for counting occurrences and exploring the relationship between two categorical variables.

Groupby

`Groupby` is a versatile method in pandas used to group data based on one or more columns, and then perform aggregate functions on the grouped data.

Introduction

Related:

- Groupby vs Crosstab

In DE_Tools see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Investigating/Transformation/group_by.ipynb

Implementation

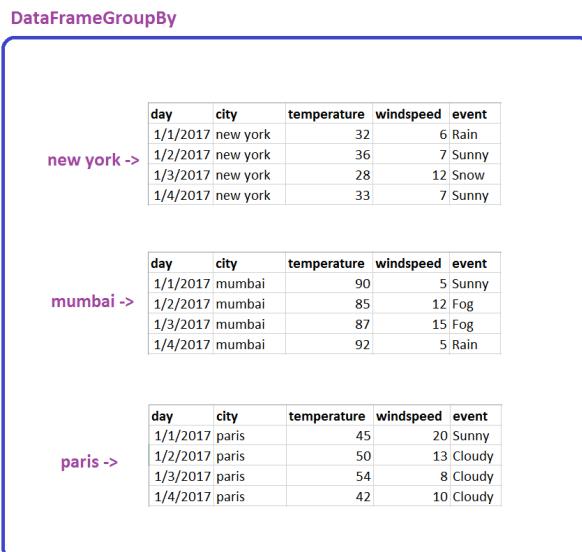
```
# Sample DataFrame  
df = pd.DataFrame({'Category': ['A', 'B', 'A', 'B', 'A'], 'Values': [10, 20, 30, 40, 50]})  
  
# Group by 'Category' and calculate the sum of 'Values'  
grouped = df.groupby('Category').sum()  
  
print(grouped)
```

Output:

```
Values  
Category  
A      90  
B      60
```

day	city	temperature	windspeed	event
1/1/2017	new york	32	6	Rain
1/2/2017	new york	36	7	Sunny
1/3/2017	new york	28	12	Snow
1/4/2017	new york	33	7	Sunny
1/1/2017	mumbai	90	5	Sunny
1/2/2017	mumbai	85	12	Fog
1/3/2017	mumbai	87	15	Fog
1/4/2017	mumbai	92	5	Rain
1/1/2017	paris	45	20	Sunny
1/2/2017	paris	50	13	Cloudy
1/3/2017	paris	54	8	Cloudy
1/4/2017	paris	42	10	Cloudy

df.groupby('city') →



Grouped Plots

Related:

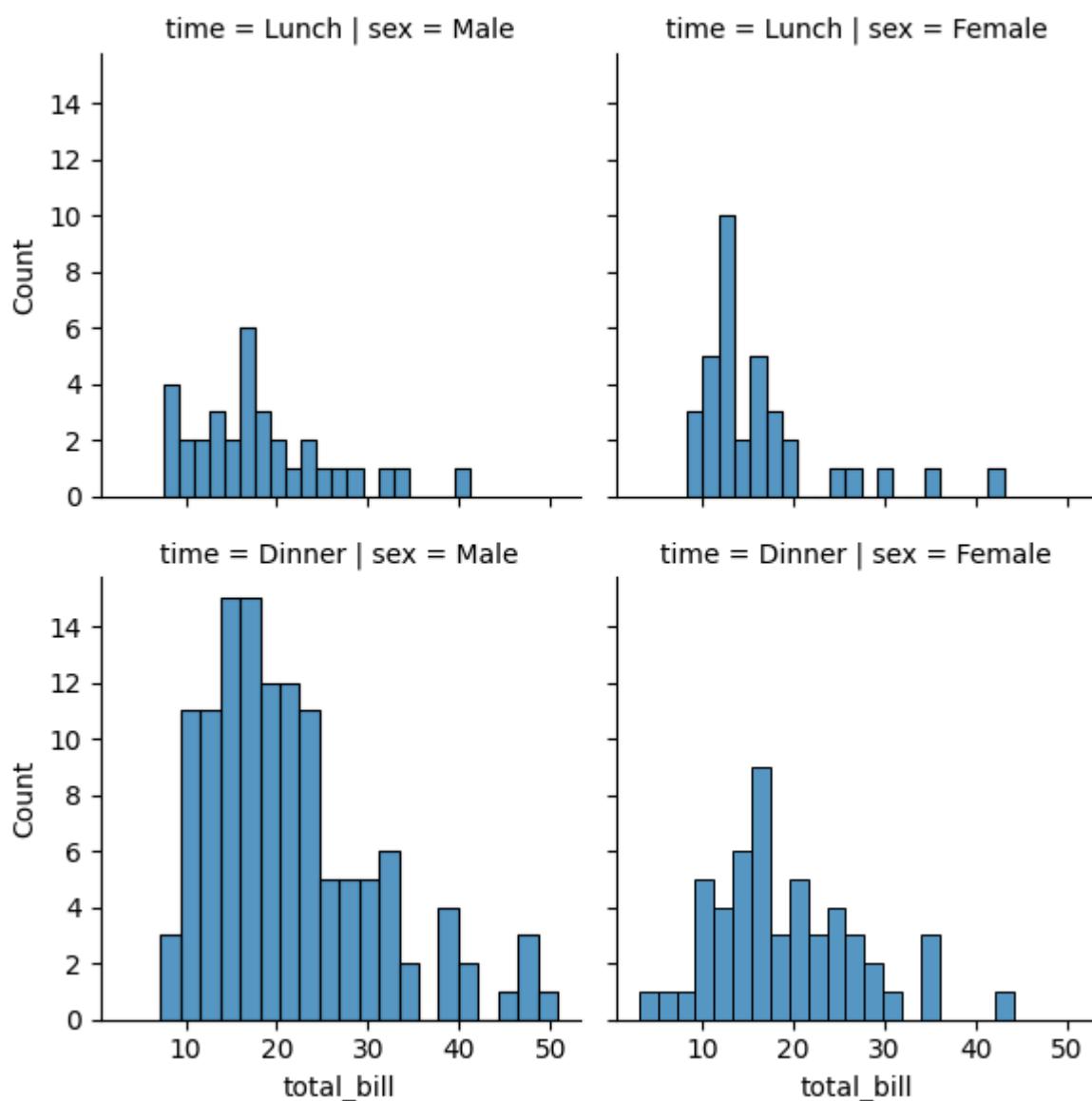
- Data Visualisation
- pairplots

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load example dataset
tips = sns.load_dataset("tips")

# Facet Grid Example
g = sns.FacetGrid(tips, col="sex", row="time")
g.map_dataframe(sns.histplot, x="total_bill", bins=20)

plt.show()
```



Guardrails

Controlling a [Generative AI](#) in business through the use of [Guardrails](#) ensures that the AI remains aligned with specific business goals and avoids unintended or harmful outputs. Guardrails are essential for maintaining security, compliance, and reliability in AI systems. Here's an outline based on your notes:

1. Input Guardrails

- Prompt Injection Control: [Prompting](#) To prevent users from prompting the AI in ways that could result in harmful or inappropriate responses, filtering or validating inputs can be essential. This reduces the risk of the model being "jailbroken" (i.e., forced to generate outputs outside its intended use case).
- Topic Restriction: Limit the AI's inputs to specific business-relevant topics. For instance, if the AI is designed for customer support, it should ignore inputs about unrelated topics (e.g., entertainment or politics).
- User Authentication: Depending on business needs, certain input guardrails can restrict access to specific features or sensitive information based on user credentials or roles.

2. Output Guardrails

- Content Moderation: Post-processing can be applied to outputs to ensure they align with business values, compliance regulations, or safety standards. For example, any harmful or offensive language can be filtered out.
- Pre-defined Boundaries: Limit the AI's responses to fall within specific domains. For instance, when the AI is asked questions outside its scope, it can respond with a predefined message, such as "I am not programmed to handle that topic."
- Compliance and Ethical Constraints: Outputs can be regulated to ensure the model adheres to legal, ethical, and regulatory constraints, which is especially important in industries like finance or healthcare.

3. Jailbreaking Concerns

- Jailbreaking occurs when a user manipulates the system to bypass these guardrails, leading to undesirable outputs. This depends on the business context—some may tolerate more flexible AI behavior, while others, like legal or healthcare firms, need strict controls.

4. Business-Specific Use Cases

- Tailor the AI to address specific business needs. For example, a generative AI for a legal firm should stick to legal advice and documentation, whereas a customer service chatbot should handle predefined topics like returns and product support.
- [Data Observability|monitoring](#) / Monitoring and Logging: Keep track of input and output interactions to ensure that the AI's performance remains within its intended boundaries.

Gitlab Ci.Yml

The purpose of a `gitlab-ci.yml` file is to define and configure the **GitLab CI/CD pipeline** for automating tasks such as building, testing, and deploying your code. It is the core configuration file that GitLab uses to orchestrate and execute CI/CD workflows in a repository.

Key Purposes:

-
- 1. Automation of Workflows:

- Automates repetitive tasks like running tests, building applications, linting code, and deploying updates.

2. Pipeline Definition:

- Specifies the **stages** (e.g., `build`, `test`, `deploy`) and their sequence.
- Defines the **jobs** within each stage and their respective commands.

3. Consistency and Reliability:

- Ensures consistent execution of tasks across environments, reducing errors caused by manual intervention.

4. Integration with GitLab:

- Automatically triggers pipelines in response to events such as code pushes, merge requests, or scheduled runs.

5. Environment Management:

- Manages deployments to various environments (e.g., development, staging, production) with variables, conditions, and manual approvals.

6. Feedback and Reporting:

- Provides immediate feedback on the status of tasks (e.g., whether tests passed) directly in the GitLab interface.
- Supports artifact generation and uploads (e.g., logs, reports, or compiled binaries).

Benefits:

- Improves development velocity by automating workflows.
- Increases code quality through consistent testing and linting.
- Simplifies deployments to various environments.
- Enables team collaboration with clear and visible pipeline progress.

Example

```

# Define the stages of the pipeline in the order they will be executed

stages:
  - build      # The stage where the application is built
  - test       # The stage where tests are executed
  - deploy     # The stage where the application is deployed

# Job to build the project

build_job:
  stage: build          # Assign this job to the 'build' stage
  script:                # Commands to execute during this job
    - echo "Building the project" # Example build command (replace with actual build steps)
  artifacts:             # Files or directories to save for use in subsequent jobs
  paths:
    - build/            # Save the 'build' directory as an artifact for later stages

# Job to test the project

test_job:
  stage: test           # Assign this job to the 'test' stage
  script:                # Commands to execute during this job
    - echo "Running tests" # Example test command (replace with actual test steps)

# Job to deploy the project

deploy_job:
  stage: deploy          # Assign this job to the 'deploy' stage
  script:                # Commands to execute during this job
    - echo "Deploying the application" # Example deployment command (replace with actual deployment steps)
  only:
    - main              # Specify when this job should run
      # Only run this job for commits to the 'main' branch

```

Granularity

Definition of Grain in Dimensional Modelling

- The grain of a **Fact Table** defines what a single row in the table represents. It is the level of detail captured by the fact table.
- Declaring the grain is essential because it sets the foundation for the entire dimensional model. It determines how detailed the data will be.

Importance of Grain Declaration:

- The grain must be established before selecting **Dimensions** and **Facts** because all dimensions and facts must align with the grain.
- This alignment ensures consistency across the data model, which is critical for the performance and usability of **business intelligence** applications.

Balancing Granularity:

- In the transformation layer, you need to decide the level of aggregation. For instance, you might aggregate hourly data into daily data to save storage space.

- Adding dimensions increases the number of rows exponentially, so it's important to carefully choose which dimensions to include.
-

Semantic Layer:

- A [semantic layer](#) sits on top of transformed data in a data warehouse, providing flexibility and enabling ad-hoc analysis without needing to store every possible data representation.
- This is akin to [OLAP](#) cubes, where you can perform complex queries (slice-and-dice) on large datasets without pre-storing all combinations.

Choosing the level of granularity

Granularity, or grain, refers to the [level of detail](#) represented by a single row in a fact table within a data warehouse.

The choice of granularity depends on the business requirements and the types of analyses you want to support. Finer granularity (e.g., transaction-level) provides more detailed insights but requires more storage and processing power. Coarser granularity (e.g., monthly product-level) reduces storage needs and can improve query performance but may limit the depth of analysis.

By clearly defining the grain, you ensure that all dimensions and facts in the data model are consistent and aligned with the intended analytical use cases.

Example: Retail Sales Data

Imagine you are designing a data warehouse for a retail company that tracks sales transactions. You need to decide the granularity of the sales fact table. Here are a few possible options:

1. Transaction-Level Granularity:

- Grain: Each row represents a single sales transaction.
- Example: A row might include details such as transaction ID, date and time of sale, store location, product sold, quantity, and total sale amount.
- Use Case: This level of granularity is useful for detailed analysis, such as examining individual customer purchases or identifying specific transaction patterns.

2. Daily Store-Level Granularity:

- Grain: Each row represents the total sales for a specific store on a specific day.
- Example: A row might include the store ID, date, total sales amount, and total number of transactions for that day.
- Use Case: This granularity is suitable for analyzing daily sales trends across different stores, comparing store performance, or identifying peak sales days.

3. Monthly Product-Level Granularity:

- Grain: Each row represents the total sales for a specific product across all stores for a specific month.
- Example: A row might include the product ID, month, total sales amount, and total units sold.
- Use Case: This level is ideal for tracking product performance over time, identifying best-selling products, or planning inventory and supply chain logistics.

H

Table of Contents

- [Hadoop](#)
- [Handling Different Distributions](#)
- [Handling_Missing_Data.ipynb](#)
- [Handling_Missing_Data_Basic.ipynb](#)
- [Handwritten Digit Classification](#)
- [Hash](#)
- [Heatmap](#)
- [Heatmaps_Dendrograms.py](#)
- [Hierarchical Clustering](#)
- [High cross validation accuracy is not directly proportional to performance on unseen test data](#)
- [Honkit](#)
- [Hosting](#)
- [How LLMs store facts](#)
- [How businesses use Gen AI](#)
- [How do we evaluate of LLM Outputs](#)
- [How is reinforcement learning being combined with deep learning](#)
- [How is schema evolution done in practice with SQL](#)
- [How to do git commit messages properly](#)
- [How to model to improve demand forecasting](#)
- [How to normalise a merged table](#)
- [How to reduce the need for Gen AI responses](#)
- [How to search within a graph](#)
- [How to use Sklearn Pipeline](#)
- [How would you decide between using TF-IDF and Word2Vec for text vectorization](#)
- [Hugging Face](#)
- [Hyperparameter Tuning](#)
- [Hyperparameter](#)
- [Hypothesis testing](#)
- [heterogeneous features](#)
- [how do you do the data selection](#)

Hadoop

Hadoop provides the backbone for distributed storage and computation. It uses HDFS (Hadoop Distributed File System) to split large datasets across clusters of servers, while MapReduce enables parallel processing. It's well-suited for [Batch Processing](#) tasks, though newer tools like [Apache Spark](#)|[Spark](#) often outperform Hadoop in terms of speed and ease of use.

1. Architecture:

- **Open-Source Framework:** Hadoop is an open-source framework for distributed storage and processing of large datasets using clusters of commodity hardware.
 - **Distributed File System:** The Hadoop Distributed File System (HDFS) stores data across multiple machines, providing high throughput access to data.
-

- **MapReduce:** Originally designed for [Batch Processing](#) using the MapReduce programming model, though newer frameworks like Apache Spark are often used now.

2. Data Storage:

- **Unstructured, Semi-Structured, and Structured Data:** Hadoop can handle a wide variety of data formats, including unstructured, semi-structured, and structured data.
- **Scalable Storage:** HDFS can store vast amounts of data by adding more nodes to the cluster.

3. Management:

- **Complex Management:** Requires more administrative effort to manage and maintain the infrastructure, including handling failures, load balancing, and tuning.

4. Performance:

- **Batch Processing:** Hadoop is optimized for batch processing of large datasets, though it can be less efficient for real-time processing compared to other systems.
- **Latency:** Higher latency for query processing compared to Snowflake, particularly for complex analytical queries.

5. Use Cases:

- **Big Data Processing:** Ideal for large-scale data processing tasks, including ETL (Extract, Transform, Load), data mining, and large-scale machine learning.
- **Data Lake:** Commonly used as a data lake to store vast amounts of raw data.

Handling Different Distributions

Handling different [distributions](#) is needed for developing robust, fair, and accurate machine learning models that can adapt to a wide range of data environments.

Importance of Handling Different Distributions

1. **Model Robustness:** Ensures models generalize well to new, unseen data.
2. Bias Mitigation: Prevents bias in predictions by accommodating diverse data types.
3. Improved [Accuracy](#): Fine-tunes models for better accuracy across varied [Datasets](#).
4. Maintains model effectiveness across different data sources.
5. Decision Making: Informs [Preprocessing](#), [model selection](#), and evaluation strategies.

Resources

Video: [Training and Testing on Different Distributions](#)

Example Scenario

High-resolution photos (many) vs. amateur photos (small number) exhibit different distributions.

Strategy for Handling Distributions

Code Example: See `Handling_Different_Distributions.py` in [ML_Tools](#)

In this script:

- **Data Generation:** Creates two mock datasets with different distributions.
- **Data Splitting:** Combines and splits the data into train, dev, and test sets.
- **Model Tuning:** Uses `GridSearchCV` to find the best hyperparameters for a RandomForest model.
- **Model Training and Evaluation:** Trains the model on the training set and evaluates it on the dev and test sets.
- **Visualization:** Uses `matplotlib` to plot the distribution of a feature from both datasets and the model's accuracy on the dev and test sets.

Follow up questions

How best to combine the datasets? How should we shuffle and split based on the distributions? How do we pick the dev set?

1. **Combining Datasets:**
 - The script combines two datasets (`dataset1` and `dataset2`) that may have different distributions. This step ensures that the model is exposed to a variety of data during training.
2. **Random Shuffling and Splitting:**
 - By shuffling and splitting the combined dataset into train, dev, and test sets, the script ensures that each set contains a mix of data from both distributions. This helps the model learn from the diversity in the data.
3. **Model Tuning with Diverse Data:**
 - The model tuning process uses the dev set, which contains data from both distributions. This helps in finding hyperparameters that work well across different data characteristics.

Related Topics

- Preprocessing

Handling_Missing_Data.ipynb

https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Investigating/Cleaning/Handling_Missing_Data.ipynb

Handling_Missing_Data_Basic.ipynb

https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Investigating/Cleaning/Handling_Missing_Data_Basic.ipynb

Handwritten Digit Classification

 Pasted image 20241006124356.png|800

Hash

A hash is a fixed-size string of characters that is generated from input data of any size using a hash function.

Hashes are used to ensure [data integrity](#) by providing a unique representation of the data, making it easy to detect any changes or alterations.

Key Characteristics of Hashes:

1. **Deterministic:** The same input will always produce the same hash output.
2. **Fixed Size:** Regardless of the size of the input data, the output hash will always be of a fixed length (e.g., SHA-256 produces a 256-bit hash).
3. **Fast Computation:** Hash functions are designed to compute the hash value quickly.
4. **Pre-image Resistance:** It should be computationally infeasible to reverse-engineer the original input from its hash.
5. **Collision Resistance:** It should be difficult to find two different inputs that produce the same hash output.

How Hashes are Used in Data Integrity:

1. **Data Verification:** When data is stored or transmitted, a hash of the data is generated and stored or sent along with it. When the data is later accessed, the hash is recalculated and compared to the original hash. If they match, the data is considered intact; if not, it indicates potential corruption or tampering.
2. **Digital Signatures:** Hashes are often used in digital signatures to ensure the authenticity and integrity of a message or document.
3. **Password Storage:** Instead of storing passwords in plain text, systems often store the hash of the password. When a user logs in, the system hashes the entered password and compares it to the stored hash.

Example of Hashing:

For example, if we take the string "Hello, World!" and apply a hash function like SHA-256, it will produce a unique hash value:

- Input: "Hello, World!"
- Hash (SHA-256): a591a6d40bf420404a011733cfb7b190d62c65bf0bcda190f4b6c3f0f3c3b8a

If the input data changes even slightly (e.g., "Hello, World"), the hash will be completely different, making it easy to detect any alterations.

Heatmap

Description

A **heatmap** is a two-dimensional graphical representation of data where individual values are represented by colors. It is particularly useful for visualizing numerical data organized in a table-like format.

A heatmap is a graphical representation of data where individual values are represented by colors. It is useful for visualizing numerical data and analyzing the correlation between features.

A heatmap is a visualization tool for analyzing the [Correlation](#) between features in a dataset. In the context of correlation analysis, a heatmap can display the correlation coefficients between different features in a dataset.

By using a heatmap, you can easily identify [Multicollinearity](#) and make informed decisions about which features to retain or remove, ultimately enhancing the performance and interpretability of your machine learning models.

Correlation Coefficients

The correlation coefficients range from -1 to 1:

- **-1:** Indicates a perfect negative correlation; if one attribute is present, the other is almost certainly absent.

- **0:** Indicates no correlation; there is no dependence between the attributes.
 - **1:** Indicates a perfect positive correlation; if one attribute is present, the other is also certainly present.
-

Implementation in Python

In [ML_Tools](#) see: [Heatmaps_Dendograms.py](#)

Heatmaps_Dendograms.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations\Preprocess\Correlation\Heatmaps_Dendograms.py

See:

- [Heatmap](#)
- [Dendograms](#)

Hierarchical Clustering

Hierarchical clustering builds a treelike structure of clusters, with similar clusters merged together at higher levels.

Hierarchical clustering builds a tree-like structure of clusters, with similar clusters merged together at higher levels.

High Cross Validation Accuracy Is Not Directly Proportional To Performance On Unseen Test Data

Reasons a Model with High [Cross Validation](#) Accuracy May Perform Poorly on Unseen Test Data

[Data Leakage:](#)

- Information from test folds leaks into training, inflating CV accuracy.
- Solution: Apply [preprocessing](#) independently within each CV fold.

[Overfitting:](#)

- Model captures noise in training data, leading to high CV but low test accuracy.
- Solution: Use simpler models, regularization, and evaluate test performance during [hyperparameter tuning](#).

[Insufficient Cross-Validation Folds:](#)

- Too few folds lead to high variance in performance estimates.
- Solution: Use more folds (e.g., 5- or 10-fold CV) for reliable estimates.

[Over-Optimized Hyperparameters:](#)

- Excessive tuning results in models that fail to generalize.
- Solution: Reserve a separate validation set for tuning and use nested cross-validation.

[Small Dataset Size:](#)

- Small datasets may lead to unreliable accuracy estimates.
- Solution: Use bootstrapping or collect more data if possible.

[Inappropriate Performance Metric:](#)

- CV accuracy may not align with the true objective (e.g., [imbalanced datasets](#)).
-

Introduction

- Solution: Choose appropriate [Evaluation Metrics](#) based on the problem context.

Practical Recommendations

- Evaluate the model on a completely independent test set after cross-validation.
- Check for [Distributions](#)|[distribution](#) differences between training and test data.
- Avoid data leakage by ensuring strict separation of preprocessing in CV folds.

Honkit

<https://honkit.netlify.app/examples>

<https://flaviocopes.com/how-to-create-ebooks-markdown/#:~:text=honkit%20works%20great.,and%20let%20CloudFlare%20distribute%20it.>

https://github.com/rhyslwells/Note_Compiler

Hosting

Using Plotly

Once you have a simple Dash application.

What You Can Do With It

Free Public Hosting Options

a) Render (<https://render.com>)

1. Create a free account.
2. Create a new "Web Service."
3. Connect your GitHub repo (with `app.py` and `requirements.txt`).
4. Set the start command: `python app.py`.

Pros:

- Simple and free, supports Dash

Cons:

- Cold start delay on the free tier

b) Railway (<https://railway.app>)

1. Sign up for a free plan.
2. Link your GitHub project.
3. Define your start command and Python version.

Pros:

- Very fast to set up

Cons:

- ✖ May require a credit card to unlock some features
-

How LMs Store Facts

[How might LLMs store facts](#)

Not solved

How do [Multilayer Perceptrons](#) store facts?

Different directions encode information in [Vector Embedding](#) space.

MLP's are blocks of vectors, these are acted on by the context matrix

[Johnson–Lindenstrauss lemma](#)

Sparse Autoencoder - used in [interpretability](#) of [LLM](#) responses

See [Anthropic](#) posts

- https://transformer-circuits.pub/2022/toy_model/index.html#adversarial
- <https://transformer-circuits.pub/2023/monosemantic-features>

How Businesses Use Gen Ai

Businesses leverage generative AI to transform various operations, using models like OpenAI, Gemini (Google Cloud), Anthropic, and Meta models. These models provide services through cloud providers, making them accessible via APIs. Key use cases include:

1. **Content Creation:** Generative AI can produce text, images, code, and even videos, enhancing marketing, design, and communication efforts.
2. **Customer Support:** AI chatbots and assistants automate customer interactions, reducing response times and improving service quality.
3. **Data Analysis & Insights:** Models help businesses analyze large datasets, enabling predictive analytics and trend forecasting.
4. **Customization:** Personalization of products and services, such as tailored recommendations or [transactional journeys](#)/customer experiences, is powered by generative AI.
5. **Multi-Model Access:** Enterprises use AI gateways to integrate multiple generative models, allowing them to choose the best model for specific tasks based on performance or cost efficiency.

Cloud providers like **Google Cloud (Gemini)** or **Microsoft Azure (OpenAI)** offer easy integration of these models into business workflows through APIs, streamlining deployment for large-scale applications

AI Gateway?

An AI Gateway is a middleware platform that simplifies and secures interactions between AI models and applications. In this context, businesses use AI gateways to streamline the integration, management, and deployment of generative AI models like those provided by OpenAI, Google (Gemini), and Anthropic. AI gateways provide the following key benefits:

1. **Model Access and Management:** They centralize access to multiple AI models via APIs, making it easier for businesses to switch between or utilize multiple AI models for different tasks.
-

-
- 2. **Security and Governance:** AI gateways add layers of security, enabling compliance with regulations and protecting proprietary data when using external AI services [1] . [2]
 - 3. **Performance Optimization:** By handling the AI model interactions efficiently, gateways can reduce latency and improve [model performance](#) in business applications [3]

Sources

- 4. [konghq.com](#) - What is an AI Gateway? Concepts and Examples
- 5. [ibm.com](#) - How an AI Gateway provides leaders with greater control
- 6. [traefik.io](#) - AI Gateway: What Is It? How Is It Different From API Gateway?

How Do We Evaluate Of LLM Outputs

Methods for assessing the quality and relevance of LLM-generated outputs, critical for improving model performance.

The evaluation of [LLM](#) outputs involves various methodologies to assess their quality and relevance.

Important

- Evaluating LLM outputs requires both quantitative metrics ([LLM Evaluation Metrics](#)) and qualitative assessments (human judgment).
- The iterative feedback loop from evaluations informs model improvements and prompt engineering strategies.

Follow up questions

- How does the inclusion of diverse datasets impact the robustness of LLM evaluations
- [What are the best practices for evaluating the effectiveness of different prompts](#)

Related Topics

- [Prompt engineering](#) in natural language processing

How Is Reinforcement Learning Being Combined With Deep Learning

The sources touch upon reinforcement learning as an area beyond the scope of their discussion. However, the combination of [Reinforcement learning](#) with [Deep Learning](#) has shown remarkable results in recent years, particularly in areas like game playing and robotics.

Exploring the potential of this combination in other domains and developing new algorithms that effectively integrate deep learning representations with reinforcement learning principles could lead to significant advancements in artificial intelligence.

How Is Schema Evolution Done In Practice With Sql

Structure of a goof Git Commit Message

1. Subject Line

- o Keep it short (50 characters or less).
- o Use the imperative mood (e.g., "Fix bug" instead of "Fixed bug").
- o Capitalize the first letter.
- o Do not end with a period.

2. Body (Optional)

- o Separate from the subject line with a blank line.
- o Explain the "what" and "why" of the changes, not the "how".
- o Wrap text at 72 characters.

3. Footer (Optional)

- o Include references to issues or pull requests (e.g., "Closes #123").
- o Add any additional notes or metadata.

Good Examples:

1. Fix a Bug

```
Fix incorrect login redirection  
The login redirection was leading to an unauthorized page after successful login.  
This fix ensures users are redirected to their dashboard upon successful authentication.
```

2. Add a New Feature

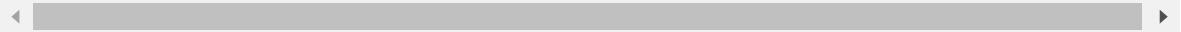
```
Add search functionality to the user dashboard  
Introduced a search bar in the user dashboard, allowing users to quickly find relevant information within their profi  

```

3. Update Documentation

```
Update README to include new API endpoints  
Added details about the newly added API endpoints for user registration and password recovery in the README file.
```

4. Refactor Code

```
Refactor data fetching logic in the dashboard  
The data fetching logic was consolidated into a reusable service to improve maintainability and reduce duplication.  

```

5. Add Unit Tests

```
Add unit tests for the authentication service  
Implemented unit tests for the login and registration methods to ensure robust coverage of authentication functionali
```



Bad Examples:

1. Too Vague

```
Update
```

This commit message doesn't provide any meaningful context.

2. Incomplete Explanation

```
Fix bug
```

This doesn't explain the what or why, making it unclear to someone reviewing the code.

3. Too General

```
Changed stuff
```

"Changed stuff" is not informative and doesn't provide clear insight into what was actually modified.

4. No Context

```
Fixed issue
```

No context is provided about what the issue was, making it difficult for others to understand the change.

5. Overly Short

```
Remove unused variable.
```

The message could be expanded to include more context about why the variable was removed and what impact it had.

Tips Expanded:

- **Be Descriptive:** Commit messages should give a clear understanding of the changes. Describe what was changed, *why* it was changed, and *how* (if necessary).
- **Focus on One Change:** Avoid including unrelated changes in a single commit. Each commit should represent one logical unit of work.
- **Use Active Voice:** Avoid passive voice. Focus on the subject doing something (`Add` , `Fix` , `Implement`).
- **Wrap Text Appropriately:** Ensure lines don't exceed 72 characters for better readability in Git log and discussions.
- **Be Concise:** Subject lines should be short (preferably under 50 characters), clear, and to the point without unnecessary detail.

Common Pitfalls to Avoid:

- **Too Vague:** Commit messages like "Update" or "Fix bug" provide no actionable information.
- **Too Long:** Descriptive, yes, but avoid overly lengthy messages that become difficult to scan quickly.
- **No Context:** A good commit message should allow anyone reviewing it to understand the change without needing additional context.

How To Model To Improve Demand Forecasting

How To Normalise A Merged Table

See: [Normalised Schema](#)

Splitting out tables.

Resource: [Database Normalization for Beginners | How to Normalize Data w/ Power Query \(full tutorial!\)](#)

How To Reduce The Need For Gen Ai Responses

Reducing the need for frequent [Generative AI](#) (Gen AI) responses can be done by leveraging techniques such as [caching](#) and setting up predefined [transactional journeys](#). Here's a breakdown:

1. **Caching AI Responses:** Caching allows storing frequently requested AI responses and reusing them. This reduces the number of queries to the AI model, thus lowering both response time and cost. For example, common queries like "How do I reset my password?" can be cached for quick reuse without engaging the AI model each time (1).
2. **Predefined Transactional Journeys:** For repetitive tasks (e.g., "I want to close my account"), predefined [workflows](#) or "journeys" can be set up. These automate processes without requiring AI interaction. This is ideal for tasks like bill payments, account management, or order cancellations, where responses can be scripted or handled by traditional logic, bypassing AI.

Examples of User Journeys:

- **Account Closure:** Guiding users through the steps to close an account without involving AI.
- **Password Reset:** Automating the reset process with predefined steps.
- **Order Tracking:** Providing real-time updates using existing tracking systems.

Sources

- [medium.com - How Cache Helps in Generative AI Response and Cost Optimization](#)
- [medium.com - Slash Your AI Costs by 80%](#)
- [botpress.com - How to Optimize AI Spend Cost in Botpress](#)

Vector Search with Graph Context

[Vector Embedding](#) plays a crucial role in enhancing search capabilities:

Comparison of Vector-Only vs. Graph-RAG:

Introduction

- Vector-only searches may lack context, while Graph-RAG utilizes graph traversal to provide richer, multi-step context.
- This leads to more complex and informative responses.

Contextual Prompts:

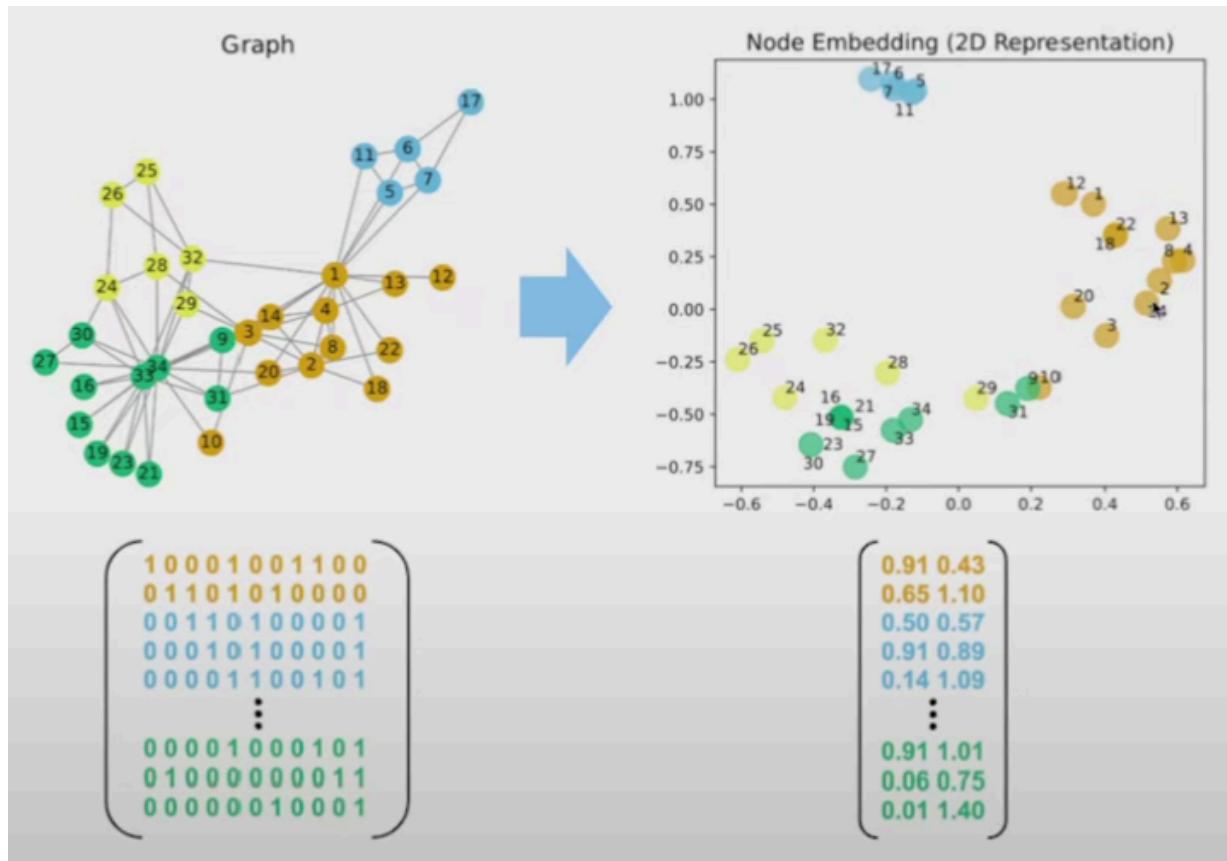
- Context is used to answer prompts (in JSON format). With graph traversal, this context involves more steps, allowing for more elaborate retrieval queries.

Text2Cypher

How to search within a graph

Node Embedding

Useful in [GraphRAG](#) is understanding the relationships of nodes in a [Knowledge Graph](#) using node embeddings.



How To Use Sklearn Pipeline

How Would You Decide Between Using Tf Idf And Word2Vec For Text Vectorization

Hugging Face

Hugging Face is open-source platform known for its contributions to natural language processing (NLP) and machine learning.

Introduction

It provides a comprehensive library called [Transformer](#), which includes pre-trained models for tasks such as text classification, translation, summarization, and question answering.

Hugging Face is widely used for:

- **Access to Pre-trained Models:** Offers a vast collection of state-of-the-art models that can be easily fine-tuned for specific NLP tasks.
- **Ease of Use:** Simplifies the implementation of complex NLP models with user-friendly APIs.
- **Community and Collaboration:** Hosts a vibrant community where researchers and developers share models and datasets, fostering collaboration and innovation in AI.

Transformer library

Resources:

- [Getting Started With Hugging Face in 15 Minutes | Transformers, Pipeline, Tokenizer, Models](#)

Hyperparameter Tuning

Objective:

- Tune the model's hyperparameters to improve performance. For example, in regularized linear regression, the main hyperparameter to tune is the regularization strength (e.g., `alpha` in Ridge or Lasso).
- Use [Cross Validation](#) to evaluate the model's performance with different hyperparameters.

Optimization Techniques:

- [GridSearchCV](#): Exhaustively searches through a specified subset of hyperparameters.
- Random Search: Randomly samples from the hyperparameter space, often more efficient than grid search.
- [standardised/Optuna](#)
- [Regularisation](#): Often part of the hyperparameter tuning process, especially in models prone to overfitting.

Key Considerations

- Balance Between Exploration and Exploitation: Ensure a good balance between exploring the hyperparameter space and exploiting known good configurations.
- [Cross Validation](#): Use cross-validation to ensure that the hyperparameter tuning process is robust and not overfitting to a particular train-test split.

Order matters: See [Interpretable Decision Trees](#). Example when tuning hyperparameters for [Random Forests](#) try the following:

1. High Train Accuracy, Low Test Accuracy (Overfitting)
2. Objective: Reduce model complexity to prevent overfitting.
3. Parameters to Adjust:
 - o `max_depth` : Limit the depth of each tree.
 - o `min_samples_split` : Increase the minimum number of samples required to split a node.
 - o `max_features` : Reduce the number of features considered for splitting.
 - o `n_estimators` : Decrease the number of trees in the forest.
4. Low Train Accuracy, Low Test Accuracy (Underfitting)
5. Objective: Increase model complexity to improve learning capacity.
6. Parameters to Adjust:
 - o `n_estimators` : Increase the number of trees.

Introduction

- o `max_depth` : Allow deeper trees.
 - o `min_samples_split` : Decrease the minimum number of samples required to split a node.
-

7. Moderate Train Accuracy, Moderate Test Accuracy (Balanced but Low Performance)

8. Objective: Fine-tune the model for better performance.

9. Parameters to Adjust:

- o `max_features` : Experiment with different numbers of features.
- o `max_depth` : Adjust the depth of trees.
- o `n_estimators` : Fine-tune the number of trees.
- o `min_samples_split` : Adjust the minimum samples for splitting.

10. High Train Accuracy, High Test Accuracy (Optimal)

11. Objective: Make minor adjustments for incremental improvements.

12. Parameters to Adjust:

- o `n_estimators` : Slightly adjust the number of trees.
- o `max_features` : Fine-tune the number of features.
- o `min_samples_split` : Make small adjustments to the minimum samples for splitting.

Links

See [ML_Tools: Hyperparameter_tuning_GridSearchCV.py](#)

Hyperparameter_tuning_RF.py Video link: <https://youtu.be/jUxhUgkKAjE?list=PLtqF5YXg7GLtQSLKSTnwCcHqTZASedbO&t=765>

Hyperparameter

Hyperparameters are parameters set before training that control the learning process, such as:

- the number of nodes in a [Neural network](#)
- or k in [K-nearest neighbours|KNN](#).

The best ones are found with [Hyperparameter Tuning](#).

Also see:

- [Model Parameters](#)
- [Model parameters vs hyperparameters](#)

Hypothesis Testing

Used to draw inferences about population parameters based on sample data. The process involves the formulation of two competing hypotheses: the null hypothesis (H_0) and the alternative hypothesis (H_1).

Key Concepts

- Null Hypothesis (H_0): The hypothesis that there is no effect or no difference, which we seek to test.
 - Alternative Hypothesis (H_1): The hypothesis that indicates the presence of an effect or a difference.
 - P-value: A measure that helps determine the strength of the evidence against H_0 . A small p-value (typically < 0.05) suggests that we reject H_0 , indicating that the observed effect is statistically significant.
-

Decision Making

- Accepting H_0 : This means there is insufficient evidence to support the alternative hypothesis, suggesting that any observed effect could be due to random chance.
- Rejecting H_0 : This indicates that there is enough statistical evidence to conclude that the status quo does not represent the truth.

Limitations

Hypothesis testing is subject to Type I errors (false positives) and Type II errors (false negatives). A small p-value does not guarantee practical significance or causation, and results can be influenced by sample variability.

Example

An example of hypothesis testing is conducting a t-test to compare the means of two groups. The null hypothesis states that the means are equal ($H_0: \mu_1 = \mu_2$), while the alternative hypothesis states they are not equal ($H_1: \mu_1 \neq \mu_2$).

Important Notes

- Hypothesis testing relies on the formulation of H_0 and H_1 , and the decision to accept or reject H_0 is based on the [p values](#).
- A small p-value indicates statistical significance but does not imply practical relevance or causation.

Follow-up Questions and Answers

In hypothesis testing, why might a very small p-value still lead to incorrect conclusions?

A very small p-value might lead to incorrect conclusions due to several factors, including:

- Sample Size: With large sample sizes, even trivial effects can yield small p-values, leading to the rejection of H_0 for effects that are not practically significant.
- Multiple Comparisons: Conducting multiple tests increases the risk of Type I errors, where we incorrectly reject H_0 .
- Misinterpretation: A small p-value does not imply that the effect is large or important; it merely [indicates that the observed data is unlikely under \$H_0\$](#) .

How does the inclusion of effect size metrics improve the interpretation of hypothesis testing results?

Including [effect size metrics](#) provides a quantitative measure of the magnitude of the observed effect, allowing researchers to assess the practical significance of their findings. While p-values indicate whether an effect exists, [effect sizes help determine how meaningful that effect is](#) in real-world terms.

What are the implications of multiple testing on the validity of p-values in hypothesis testing?

Multiple testing increases the likelihood of encountering false positives (Type I errors). When multiple hypotheses are tested simultaneously, the probability of incorrectly rejecting at least one true null hypothesis rises. This necessitates adjustments to p-values (e.g., Bonferroni correction) to maintain the overall error rate.

Related Topics

- Bayesian statistics and its approach to hypothesis testing
- The role of confidence intervals in statistical [inference](#)

- [Statistics](#)
 - [Testing](#)
-

Heterogeneous Features

Description

In machine learning, heterogeneous features refer to a situation where the input data contains a variety of different types of features. Let's break it down:

1. Features:

- Features are the individual measurable properties or characteristics of the data used for making predictions in a machine learning model.
- For example, in a dataset about houses, features could include the number of bedrooms, square footage, location, and whether it has a garden.

2. Homogeneous vs. Heterogeneous:

- **Homogeneous Features:** In some datasets, all features are of the same type, such as numerical or categorical. For instance, a dataset containing only numerical features like age, income, and temperature is homogeneous.
- **Heterogeneous Features:** In contrast, heterogeneous features refer to datasets where features are of different types. This means the dataset may contain a mix of numerical, categorical, text, image, or other types of data.

3. Examples of Heterogeneous Features:

- **Numerical Features:** Represented by continuous values like age, income, or temperature.
- **Categorical Features:** Represented by discrete values such as gender, city, or type of car.
- **Text Features:** Textual data like product descriptions, customer reviews, or email content.
- **Image Features:** Visual data represented by pixels in an image, used in tasks like image recognition or object detection.

4. Challenges and Considerations:

- Handling heterogeneous features requires specialized techniques in [Preprocessing](#) and model building.
- Different types of features may need different preprocessing steps, such as encoding categorical variables, scaling numerical features, or extracting features from text or images.
- Models need to be capable of handling diverse data types, either through feature engineering or using algorithms specifically designed for heterogeneous data.

5. Applications:

- Heterogeneous features are common in many real-world applications, such as e-commerce (combining text descriptions with numerical features), healthcare (integrating medical records with images or text), and social media analysis (analyzing text, images, and user profiles).

6. Resources for Further Learning:

- Feature Engineering for Machine Learning: <https://www.datacamp.com/community/tutorials/feature-engineering-kaggle>
- Handling Text Data in Machine Learning: <https://towardsdatascience.com/handling-text-data-in-machine-learning-projects-b52bbc9531d7>
- Image Feature Extraction Techniques: <https://towardsdatascience.com/image-feature-extraction-techniques-91e8625616f1>

Understanding how to work with heterogeneous features is essential for building effective machine learning models that can handle diverse types of data and extract meaningful insights from them.

How Do You Do The Data Selection

When you sample a dataset, [how do you do the data selection?](#) Data Selection A: By randomly sampling, by time period (use a feature)..



Table of Contents

- [Imbalanced Datasets](#)
- [Imbalanced_Datasets_SMOTE.py](#)
- [Immutable vs mutable](#)
- [Impact of multicollinearity on model parameters](#)
- [Implementing Database Schema](#)
- [In NER how would you handle ambiguous entities](#)
- [Industries of interest](#)
- [Input is Not Properly Sanitized](#)
- [Interpreting logistic regression model parameters](#)
- [Interquartile Range \(IQR\) Detection](#)
- [Isolated Forest](#)
- [imperative](#)
- [in-memory format](#)
- [incremental synchronization](#)
- [inference versus prediction](#)
- [inference](#)
- [information theory](#)
- [interoperable](#)
- [interpretability](#)
- [interview notepad](#)
- [ipynb](#)

Imbalanced Datasets

Handling imbalanced datasets to ensure robustness of models is a common challenge in machine learning, particularly in classification tasks where one class significantly outnumbers the other(s).

In [Classification](#) tasks, an imbalanced dataset can lead to a model that [performs well on the majority class but poorly on the minority class](#). This is because the model may learn to predict the majority class more often due to its prevalence.

For [Regression](#) tasks, handling outliers or data skewness might be necessary.

In [ML_Tools](#) see:

- [Imbalanced_Datasets_SMOTE.py](#)

Examples

Consider a scenario where you have an imbalanced dataset of resumes, with a majority of male resumes and a minority of female resumes. You want to build a model to predict gender based on resume features.

Strategies to address imbalances

Data-Level Approaches

Resampling Techniques:

- Oversampling: Increase the number of instances in the minority class by duplicating existing samples or generating new ones using techniques like [SMOTE \(Synthetic Minority Over-sampling Technique\)](#).
- Undersampling: Reduce the number of instances in the majority class by randomly removing samples. This can help balance the dataset but may lead to loss of important information.
- Data Augmentation: Apply transformations to existing data to create new samples, which is particularly useful in image data. Techniques include rotation, flipping, scaling, and cropping.

Algorithm-Level Approaches

- [Cost-Sensitive Analysis](#) / Cost-Sensitive Learning: Modify the learning algorithm to give more importance to the minority class. This can be done by assigning higher misclassification costs to the minority class during training.1. You have a perfectly balanced dataset but still experience poor classification accuracy. Why might the class separability be the issue?
- Ensemble Methods: [Bagging](#) and Boosting: Use ensemble techniques like [Random Forests](#) or AdaBoost, which can be adapted to handle class imbalance by adjusting the sample weights or using balanced bootstrap samples.

Evaluation Metrics & Others

- [Model Evaluation/Evaluation Metrics](#) Use Appropriate Metrics: Instead of accuracy, use metrics that are more informative for [imbalanced datasets](#), such as precision, recall, F1-score, and the area under the ROC curve (AUC-ROC).
- [Anomaly Detection](#) Models: Treat the minority class as anomalies and use [anomaly detection](#) techniques to identify them.
- [Transfer Learning](#): Use pre-trained models that have learned features from a balanced dataset, which can be fine-tuned on the imbalanced dataset.

Imbalanced_Datasets_Smote.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Imbalanced_Datasets_SMOTE.py

Demonstrating the Value of Resampling in Imbalanced Classification

This example highlights the effectiveness of resampling techniques, such as [SMOTE \(Synthetic Minority Over-sampling Technique\)|SMOTE](#), in addressing [Imbalanced Datasets|class imbalance](#) issues in classification tasks. By implementing the following strategies, the setup ensures a measurable improvement in model performance:

1. **Severe Imbalance and Dataset Size:**
 - Utilizing a larger dataset with a severe imbalance ratio (e.g., 99:1) makes the impact of resampling more apparent. This imbalance necessitates resampling for the model to predict the minority class accurately.
2. **Choice of Classifier:**
 - Switching from robust classifiers like [Random Forests](#) to more sensitive ones like [Logistic Regression](#) or Support Vector Machine ([Support Vector Machines|SVM](#)) highlights the benefits of resampling. These

simpler models struggle with imbalance, providing a clear contrast between resampling and non-resampling scenarios.

3. Feature Overlap:

- Ensuring overlap in the feature space between minority and majority classes enhances the effectiveness of synthetic resampling techniques, such as SMOTE.

4. Focus on Minority Class Metrics:

- Emphasizing evaluation metrics like [recall](#) and F1-score for the minority class explicitly measures the model's ability to capture minority class instances, demonstrating the value of resampling in improving these metrics.

Results:

Without Resampling:

Class	Precision	Recall	F1-Score	Support
0	0.99	1.00	1.00	990
1	0.67	0.20	0.31	10
Accuracy			0.99	1000
Macro Avg	0.83	0.60	0.65	1000
Weighted Avg	0.99	0.99	0.99	1000

- The minority class recall will likely be very low (close to 0), as the classifier may predict the majority class almost exclusively.
- Overall [accuracy](#) will be high because the majority class dominates.

With SMOTE Resampling:

Class	Precision	Recall	F1-Score	Support
0	1.00	0.82	0.90	990
1	0.04	0.70	0.07	10
Accuracy			0.82	1000
Macro Avg	0.52	0.76	0.49	1000
Weighted Avg	0.99	0.82	0.89	1000

- Minority class recall and F1-score should improve significantly, as SMOTE provides synthetic samples to balance the training set.
- Accuracy might decrease slightly due to more emphasis on minority class performance.

Immutable Vs Mutable

[Python](#)

list being mutable

Side effect

```
def get_largest_numbers(numbers, n):
    numbers.sort()

    return numbers[-n:]

nums = [2, 3, 4, 1, 34, 123, 321, 1]

print(nums)
largest = get_largest_numbers(nums, 2)
print(nums)
```

Impact Of Multicollinearity On Model Parameters

See <https://youtu.be/StSAJIZuqws?t=655>

```
)  
  
# Monte Carlo Simulation: Multicollinearity & Harm  
results = expand_grid(  
  rho = seq(0, 0.95, 0.05),  
  rep = 1:1000  
) %>%  
  mutate(  
    sim = map(rho, function(p) {  
  
      set.seed(runif(1, 1, 10000) %>% ceiling)  
      R = matrix(c(1, p, p, 1), nrow = 2, ncol = 2, byrow = TRUE)  
      Sigma = cor2cov(R, c(1, 1))  
  
      data = MASS:: mvrnorm(n = 30, mu = c(0, 0), Sigma = Sigma) %>%  
        as_tibble %>%  
        mutate( Y=1+0.5*V1+0.5*V2+ rnorm(30) )  
  
      model = lm(Y ~ V1 + V2, data = data)  
  
      summary(model)$coefficients %>% as_tibble  
    })
```

Implementing Database Schema

To manage and create a database schema in SQLite, you can use the following commands:

- To view all commands used to create a database, execute:

```
.schema
```

- To view the schema for a specific table, use:

```
.schema table
```

- To run a schema from a file, use:

```
.read schema.sql
```

Creating a Database Schema

When creating a database schema, follow these steps:

1. **Identify the Tables:** Determine which tables are necessary for your data.
2. **Define Columns:** Specify the columns for each table.
3. **Choose Data Types:** Select appropriate data types for each column.
4. **Establish Keys:** Define primary and foreign keys to maintain data integrity.
5. **Set Column Constraints:** Ensure that values adhere to specified conditions.

Note that constraints do not need to apply to primary and foreign keys. Common constraints include:

- **CHECK** : Ensures values meet certain criteria (e.g., amount must be greater than 0).
- **DEFAULT** : Sets a default value for a column.
- **NOT NULL** : Ensures a column cannot have a NULL value.
- **UNIQUE** : Ensures all values in a column are distinct.

Example Schema Creation

Here's an example of how to create tables for a database:

```

CREATE TABLE cards (
    "id" INTEGER PRIMARY KEY
);

CREATE TABLE stations (
    "id" INTEGER PRIMARY KEY,
    "name" TEXT UNIQUE NOT NULL,
    "line" TEXT NOT NULL
);

CREATE TABLE swipes (
    "id" INTEGER PRIMARY KEY,
    "card_id" INTEGER,
    "station_id" INTEGER,
    "type" TEXT NOT NULL CHECK("type" IN ('enter', 'exit', 'deposit')),
    "datetime" NUMERIC NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "amount" NUMERIC NOT NULL CHECK("amount" != 0),
    FOREIGN KEY("card_id") REFERENCES "cards"("id"),
    FOREIGN KEY("station_id") REFERENCES "stations"("id")
);

```

Modifying the Schema

To change an existing schema, you can use commands such as `RENAME`, `ADD COLUMN`, and `DROP COLUMN`.

```

ALTER TABLE visits
RENAME TO swipes;

ALTER TABLE swipes
ADD COLUMN "swipetype" TEXT;

DROP TABLE "riders";

```

Relating Entities

Done using foreign keys.

```

CREATE TABLE visits (
    "rider_id" INTEGER,
    "station_id" INTEGER,
    FOREIGN KEY("rider_id") REFERENCES "riders"("id"),
    FOREIGN KEY("station_id") REFERENCES "stations"("id")
);

```

Also See:

- [Many-to-Many Relationships](#)

- [ER Diagrams](#)
-

In NER How Would You Handle Ambiguous Entities

Handling ambiguous entities in Named Entity Recognition (NER) can be quite challenging. Here are some strategies that can be employed:

1. **Contextual Analysis:** Utilize the surrounding **context** of the ambiguous entity to determine its correct classification. For example, the word "Apple" could refer to the fruit or the company, but the context in which it appears can help disambiguate its meaning.
2. **Disambiguation Models:** Implement additional models specifically designed for entity disambiguation. These models can leverage knowledge bases or ontologies to determine the most likely entity based on context.
3. **Multi-label Classification:** Instead of forcing a single label, allow for multiple possible labels for ambiguous entities. This can be useful in cases where an entity might belong to more than one category.
4. **Training Data:** Ensure that the training dataset includes examples of ambiguous entities in various contexts. This can help the model learn to recognize and differentiate between them.
5. **User Feedback:** Incorporate user feedback mechanisms to refine the model's predictions. If users can correct or confirm entity classifications, this can improve the model over time.

Industries Of Interest

Industries to investigate related to my background & interests:

- Energy
- Telecommunications
- Education and Training

Both Reinforcement Learning and Explainable AI offer exciting opportunities for mathematicians to contribute significantly. Your deep mathematical understanding allows you to tackle complex problems, develop new methodologies, and provide theoretical foundations for emerging techniques.

Exploratory Questions

- [What algorithms or models are used within the energy sector](#)
- [What algorithms or models are used within the telecommunication sector](#)

Reinforcement learning

- **Stochastic Processes:** Your background will allow you to delve into the mathematical properties of [Markov Decision Processes](#) MDPs, optimizing transition dynamics, and improving algorithms based on theoretical insights.
- **Theoretical Analysis:** You can contribute to the development of new algorithms by providing theoretical proofs of convergence and performance guarantees, applying concepts from real analysis and optimization.
- **Complexity Analysis:** Understanding the computational complexity of various RL algorithms and contributing to the design of more efficient algorithms will leverage your mathematical skills.

Input Is Not Properly Sanitized

Input is Not Properly Sanitized

When we say that "input is not properly sanitized," it means that the input data from users or external sources is not being adequately checked or cleaned before being processed by the application. Proper sanitization involves validating and filtering input to ensure it is safe and expected, preventing malicious data from causing harm. Without proper sanitization, applications can be vulnerable to various attacks, such as:

- **Command Injection:** Malicious commands can be executed on the server.
- **SQL Injection:** Malicious SQL queries can be executed against a database.
- **Cross-Site Scripting (XSS):** Malicious scripts can be injected into web pages.

Sanitization typically involves:

- Validating input against expected formats or values.
- Escaping special characters that could be interpreted as code.
- Removing or encoding potentially harmful content.

Interpreting Logistic Regression Model Parameters

How do this in terms of odds, probabilities ,odds ratio.

[Logistic Regression](#)

Interquartile Range (Iqr) Detection

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Outliers/outliers_IQR.py

Context:

The IQR method is a robust and widely used statistical technique for identifying outliers, especially in [univariate data](#). It is based on the distribution of data and is less sensitive to extreme values compared to methods reliant on mean and standard deviation.

Steps:

- Compute the IQR:
 - The IQR is the range within which the central 50% of the data lies.
 - Formula:

$$\text{IQR} = Q3 - Q1$$

where:

- $Q1$: The first quartile (25th percentile)
- $Q3$: The third quartile (75th percentile).

- Determine the bounds:

- Define lower and upper bounds to detect potential outliers:
-

$$\text{Lower Bound} = Q1 - 1.5 \cdot \text{IQR}$$

$$\text{Upper Bound} = Q3 + 1.5 \cdot \text{IQR}$$

- Identify anomalies:

- Any data point outside the lower or upper bounds is flagged as an anomaly.

Applications:

- Best suited for non-Gaussian distributions.
- Commonly used in boxplots for visualizing outliers.

Isolated Forest

Isolation Forest (iForest) is an [Model Ensemble](#)-based method used for anomaly detection. It operates by isolating data points using a series of random binary splits.

The key idea is that [standardised/Outliers|anomalies](#), being rare and different, are easier to isolate and thus require fewer splits.

Mathematically, the isolation of a point is captured by the path length in a decision tree, where shorter paths indicate anomalies. The algorithm constructs multiple isolation trees, and the [anomaly score of a point](#) is determined by the average path length across all trees.

Isolation Forest is highly efficient for large datasets and is particularly useful when the assumption is that anomalies are rare and distinct from normal instances.

Steps:

- Randomly select a feature and a split value between the maximum and minimum values of that feature.
- Repeat this process to create a tree structure.
- Anomalies are isolated faster than normal points, leading to shorter path lengths in the tree.
- The average path length across multiple trees is used to compute an anomaly score.

Key Components:

- **Isolation Trees (iTrees)**: Binary trees where the goal is to isolate observations based on randomly chosen features and split values.
- **Anomaly Score**: Calculated based on the average path length across all isolation trees.
- **Path Length**: Anomalies tend to have shorter path lengths as they are easier to isolate.
- **Random Splitting**: Random feature selection and splitting result in the separation of instances, with fewer splits isolating anomalies.

Important

- Anomalies are identified based on shorter average path lengths in the isolation forest, [indicating that fewer splits are needed to isolate them](#).
 - The method scales well with large datasets because it relies on randomly generated trees, avoiding complex distance or density computations.
-

Introduction

- Isolation Forest assumes that anomalies are few and distinct; it may perform poorly when anomalies are not easily distinguishable.
- The method is sensitive to the [Hyperparameter](#) such as the number of trees and sample size.

Follow up questions

- How does the isolation forest compare to density-based methods like [DBSCAN](#) in terms of detecting complex anomalies? [Anomaly Detection with Clustering](#)
- What impact does the choice of sample size have on the performance and accuracy of isolation forests in high-dimensional data?

Related Topics

- [Random Forests](#) for classification and regression
- One-Class [Support Vector Machines|SVM](#) for anomaly detection

Imperative

An **imperative** pipeline tells [how to proceed](#) at each step in a procedural manner.

In contrast, a **declarative** data pipeline does not tell the order it needs to be executed but instead [allows each step/task to find the best time and way to run](#).

The *how* should be taken care of by the tool, framework, or platform running on.

For example, update an asset when upstream data has changed.

Both approaches result in the same output.

However, the declarative approach benefits from **leveraging compile-time query planners** and **considering runtime statistics** to choose the best way to compute and find patterns to reduce the amount of transformed data.

In Memory Format

The term "in-memory format" refers to the way data is stored and managed directly in a [computer's RAM](#) (Random Access Memory) rather than on disk storage like a hard drive or SSD. This approach is used to optimize performance, as accessing data in RAM is significantly faster than accessing data on disk.

In-memory formats are often used in applications that require high-speed data processing, such as real-time analytics, caching systems, and certain types of databases (e.g., in-memory databases like Redis or SAP HANA). By keeping data in memory, these systems can reduce latency and improve throughput, enabling faster data retrieval and processing.

In-memory formats may involve [specific data structures](#) or serialization methods that are optimized for quick access and manipulation in RAM. These formats are designed to make efficient use of memory resources while ensuring that data can be quickly read and written by the application.

In-memory formats are optimized to:

- hit fast instruction sets
- be cache friendly
- be parallelizable

Formats:

- [Apache Arrow](#)

- [Apache Spark](#)
 - [NumPy](#)
 - [Pandas](#)
-

The opposed to in-memory formats are [Data Lake File Formats](#) which save space, be cross-language and serve as long-term storage.

Incremental Synchronization

Inference Versus Prediction

[inference](#) is similar to [prediction](#), but in the context of [Generative AI](#), it is more specific to the application of a pre-trained model to produce an output from new input data.

While [prediction](#) often refers to tasks like classification or regression, inferencing in [Gen AI](#) refers to generating novel outputs, such as text, images, or audio, based on learned patterns.

The key distinction is that in [generative models](#), inferencing not only predicts but creates new data (like text or images) rather than assigning categories or predicting numerical values, as in traditional machine learning models. For example:

- In a [language model LLM](#), inferencing is generating the next word or sentence in a text.
- In a [text-to-image model](#), inferencing produces an image based on a textual description.

Inference

Inferencing involves prediction, but the output is more generative and creative in nature.

[inference versus prediction](#)

Information Theory

Information theory is a mathematical framework for quantifying the transmission, processing, and storage of information.

Information theory has profound implications and applications across various domains, providing the theoretical foundation for understanding and optimizing how information is communicated and processed.

1. **Entropy:** Often referred to as Shannon entropy, it measures the average amount of uncertainty or surprise associated with random variables. In essence, it quantifies the amount of information contained in a message or dataset.
 2. **Information:** In information theory, information is defined as the reduction in uncertainty. When you receive a message, the amount of information it provides is related to how much it reduces your uncertainty about the subject.
 3. **Mutual Information:** This measures the amount of information that two random variables share. It quantifies the reduction in uncertainty about one variable given knowledge of the other.
 4. **Channel Capacity:** This is the maximum rate at which information can be reliably transmitted over a communication channel. It is determined by the channel's bandwidth and noise characteristics.
-

-
- 5. **Data Compression:** Information theory provides the basis for data compression techniques, which aim to reduce the size of data without losing essential information. Lossless compression (e.g., ZIP) and lossy compression (e.g., JPEG) are two types of compression.
 - 6. **Error Detection and Correction:** Information theory also deals with methods for detecting and correcting errors in data transmission, ensuring that information can be accurately received even in the presence of noise.
 - 7. **Rate-Distortion Theory:** This aspect of information theory deals with the trade-offs between the fidelity of data representation and the amount of compression, which is crucial in applications like audio and video compression.

Interoperable

Interpretability

Links

- 1. Interpretability Importance
- 2. <https://christophm.github.io/interpretable-ml-book/index.html>

Interpretability

Interpretability in machine learning (ML) is about understanding the reasoning behind a model's predictions. It involves making the model's decision-making process comprehensible to humans, which is crucial for trust, debugging, and ensuring fairness and reliability.

Importance of Interpretability

- **Trust:** Stakeholders are more likely to trust models they understand.
- **Debugging:** Easier to identify and fix issues in interpretable models.
- **Bias Detection:** Helps identify biases in data and model predictions.
- **Social Acceptance:** Models that can explain their decisions are more socially acceptable.
- **Fairness and Reliability:** Ensures models are fair and reliable, especially in high-impact areas.

Levels of Interpretability

- **Global, Holistic Model Interpretability:** Involves comprehending the entire model at once, including feature importance and interactions. This level of interpretability is challenging, especially for models with many parameters.
- **Global Model Interpretability on a Modular Level:** Focuses on understanding parts of the model (e.g., weights in linear models or splits in decision trees). While individual parameters may be interpretable, their interdependence complicates interpretation.
- **Local Interpretability for a Single Prediction:** Allows for detailed examination of why a model made a specific prediction for an individual instance. This can provide clearer insights as local predictions may exhibit simpler relationships than the global model.

Challenges in Achieving Interpretability

- Effective interpretation requires **context**; for instance, understanding the significance of weights in linear models is often conditional on other feature values.

- **Trade-offs:** Users must weigh the need for predictions against the need for understanding the rationale, particularly in contexts where decisions have significant consequences.
 - **Human Learning:** Interpretability supports human curiosity, facilitating updates to mental models based on new information.
 - **Safety and Bias Detection:** Essential for high-risk applications (e.g., self-driving cars) and for identifying biases in decision-making.
 - **Social Acceptance:** Machines that explain their decisions tend to be more accepted.
-

Properties of Explanations

These properties provide a framework for evaluating and comparing explanation methods in interpretable machine learning, ensuring they are effective and useful for understanding model predictions.

Properties of Explanation Methods

1. **Expressive Power:** Refers to the types of explanations generated (e.g., rules, decision trees, natural language).
2. **Translucency:** Measures the extent to which an explanation method examines the model's internal parameters. High translucency allows for more informative explanations, while low translucency enhances portability.
3. **Portability:** Indicates the range of models compatible with the explanation method. Methods that treat models as black boxes (e.g., surrogate models) are more portable.
4. **Algorithmic Complexity:** Reflects the computational demands of generating explanations, which is crucial when processing time is a concern.

Properties of Explanations

1. **Accuracy:** Assesses how well the explanation predicts unseen data. High accuracy is vital if the explanation is used in place of the model.
 2. **Fidelity:** Evaluates how closely the explanation matches the black box model's predictions. High fidelity is essential; otherwise, the explanation is ineffective.
 3. **Consistency:** Measures how similar explanations are across models trained on the same task. High consistency is desirable when models rely on similar relationships.
 4. **Stability:** Examines how consistent explanations are for similar instances. High stability is preferred to avoid erratic changes due to minor variations in input features.
 5. **Comprehensibility:** Assesses how easily humans understand the explanations. This property is challenging to define but is critical for effective communication of model behavior.
 6. **Certainty:** Indicates whether the explanation reflects the model's confidence in its predictions, adding value by clarifying prediction reliability.
 7. **Degree of Importance:** Evaluates how well the explanation identifies the importance of features involved in a decision.
 8. **Novelty:** Addresses whether the instance to be explained lies outside the training data distribution, affecting prediction accuracy.
-

-
- 9. **Representativeness:** Measures how many instances an explanation covers, ranging from individual predictions to broader model interpretations.
-

Understanding an Explanation

Here are the key takeaways on human-friendly explanations in interpretable machine learning:

Need comprehensibility and accuracy in explanations to enhance user understanding and trust in machine learning models.

Importance of Human-Friendly Explanations

- 1. **Preference for Short Explanations:** Humans favor concise explanations (1-2 causes) that contrast current situations with hypothetical scenarios where the event did not occur.
- 2. **Nature of Explanations:** An explanation answers "why" questions, focusing primarily on everyday situations rather than general scientific queries.

Characteristics of Good Explanations

- 1. **Contrastive Nature:** Good explanations highlight differences between predicted outcomes, aiding comprehension. For instance, explaining why a loan was rejected by comparing it to a hypothetical accepted application is more effective.
- 2. **Selective Focus:** People tend to prefer explanations that identify one or two key causes rather than exhaustive lists. This selective approach aligns with the "Rashomon Effect," where multiple valid explanations can exist for the same event.
- 3. **Social Context:** Explanations are influenced by the social context and audience. Tailoring explanations to the audience's knowledge level enhances understanding.
- 4. **Emphasis on Abnormal Causes:** Humans focus on rare or abnormal causes to explain events. Including these in explanations can significantly enhance clarity.
- 5. **Truthfulness vs. Selectivity:** While truthfulness is important, selectivity often takes precedence. A concise, selective explanation is more impactful than a comprehensive but complex one.
- 6. **Consistency with Prior Beliefs:** Explanations that align with the explainee's existing beliefs are more readily accepted, highlighting the challenge of integrating complex model behaviors that contradict common intuitions.
- 7. **Generality:** Good explanations should be generalizable, but abnormal causes can sometimes provide more compelling insights.

Implications for Interpretable Machine Learning

- **Design Considerations:** Create explanations that are short, contrastive, and tailored to the audience's background.
 - **Methodology:** Incorporate techniques that can produce contrastive explanations while maintaining accuracy and fidelity to the model's predictions.
 - **Audience Awareness:** Understanding the audience's social context and prior beliefs is crucial for effective communication of model outcomes.
 - Understand how the model makes predictions.
 - Use techniques like feature importance scores or LIME to explain individual predictions.
-

- **How can we design machine learning models that are both accurate and interpretable?** While deep learning models often achieve high accuracy, their complexity can make them difficult to interpret. This raises questions about how to balance accuracy and interpretability. Exploring techniques for visualizing and understanding the internal representations learned by deep networks, or developing inherently interpretable models that still achieve high performance, could lead to greater trust and adoption of machine learning in critical applications like healthcare and finance.

Interview Notepad

Tell about a recent project of yours;; Collaborating in the image matching kaggle competition. Obviously S2DS project too.

What are some areas in this business you are interested in?;; Technical consulting, energy projects. Any area with the room to problem solve, to apply the scientific method to business problems. Areas where data can be turned into decisions. Building technical systems too for operations (feasibility projects).

How do you approach prioritizing tasks in a data science project?;; Current project objectives, complexity, dependencies, and client impact.

How do you handle conflicts or disagreements within a data science team?;; I promote open communication, facilitate discussions, and seek win-win solutions to resolve conflicts constructively.

How do you stay updated with the latest developments and trends in data science?;; I would like to attend events, talk to peers, stay connect to colleagues online. Watch updates on youtube.

How do you ensure the quality and reliability of data used in your data science projects? ;; I implement data validation and cleaning procedures, conduct exploratory data analysis, and would collaborate with domain experts and engineers to verify data accuracy and relevance.

Can you describe a time when you had to make a decision under tight deadlines in a data science project? ;; In S2DS the client wanted further features to be added, so we said no nicely.

What strategies do you employ to ensure alignment between data science initiatives and business objectives? ;; I collaborate closely with stakeholders to understand business goals, prioritize projects based on their strategic importance, and regularly communicate progress and results to ensure alignment and maximize impact.

How do you stay organized and manage deadlines in your data science projects? ;; I utilize project management tools, break down tasks into manageable components, set realistic timelines, and regularly reassess priorities to ensure timely delivery.

Can you discuss a challenging problem you encountered in a data science project and how you resolved it? ;; In alice in wonderland people issue,, I analyzed the root cause, consulted domain experts or literature, experimented with alternative approaches, and iteratively refined solutions until achieving satisfactory results.

What do you think is the most important thing in a team?;; Buy in, communication, also initiative.

What do you think is a no-go in a team?;; Lack of accountability/blaming people, just own your mistakes and learn from them its more productive

Business and situational questions

Tell about a recent project of yours;; Collaborating in the image matching kaggle competition. Obviously S2DS project too.

Introduction

What are some areas in this business you are interested in?;; Technical consulting, [Energy](#) projects. Any area with the room to problem solve, to apply the scientific method to business problems. Areas where data can be turned into decisions. Building technical systems too for operations (feasibility projects).

How do you approach prioritizing tasks in a data science project?;; Current project objectives, complexity, dependencies, and client impact.

How do you handle conflicts or disagreements within a data science team?;; I promote open communication, facilitate discussions, and seek win-win solutions to resolve conflicts constructively.

How do you stay updated with the latest developments and trends in data science?;; I would like to attend events, talk to peers, stay connect to colleagues online. Watch updates on youtube.

How do you ensure the quality and reliability of data used in your data science projects?;; I implement data validation and cleaning procedures, conduct exploratory data analysis, and would collaborate with domain experts and engineers to verify data accuracy and relevance.

Can you describe a time when you had to make a decision under tight deadlines in a data science project?;; In S2DS the client wanted further features to be added, so we said no nicely.

What strategies do you employ to ensure alignment between data science initiatives and business objectives?;; I collaborate closely with stakeholders to understand business goals, prioritize projects based on their strategic importance, and regularly communicate progress and results to ensure alignment and maximize impact.

How do you stay organized and manage deadlines in your data science projects?;; I utilize project management tools, break down tasks into manageable components, set realistic timelines, and regularly reassess priorities to ensure timely delivery.

Can you discuss a challenging problem you encountered in a data science project and how you resolved it?;; In alice in wonderland people issue,, I analyzed the root cause, consulted domain experts or literature, experimented with alternative approaches, and iteratively refined solutions until achieving satisfactory results.

Team work questions

What do you think is the most important thing in a team?;; Buy in, communication, also initiative.

What do you think is a no-go in a team?;; Lack of accountability/blaming people, just own your mistakes and learn from them its more productive

General questions

What are some areas of the DS field you are interested in?;; NLP (the techniques machines use to understand complex concepts), time series analysis (very real, forecasting).

Why are you interested in data science?;; Problem solving aspect, with tools that are technically interesting. Work with technical minded people. I enjoy the scientific viewpoint.

How would you interact with the data science community?;; Participate in Datafest, Kaggle projects, and engage with colleagues.

```
#interview_questions
```

What is data normalization and why do we need it?;; Data [Normalised Schema](#) is used in [Preprocessing](#) for preprocessing as it rescales values to fit within a specific range.

Introduction

Explain [Dimensionality Reduction](#), where it's used, and its benefits? ;; Dimensionality Reduction involves reducing the number of feature variables by obtaining a set of principal variables, reducing storage space, speeding up computation, removing redundant features, and enabling data visualization to identify patterns.

How do you handle missing or corrupted data in a dataset? ;; Missing or corrupted data can be handled by dropping affected rows or columns, replacing them with another value, or filling them with a placeholder value using methods like `isnull()`, `dropna()`, or `fillna()` in Pandas.

How would you go about doing an exploratory data analysis (EDA)? ;; EDA involves gaining insights from data before applying predictive models, starting with high-level global insights, dropping unnecessary columns, filling missing values, and creating basic visualizations such as bar plots and scatter plots to understand feature relationships.

How do you know which machine learning model you should use? ;; [Model Selection](#) depends on factors such as the nature of the problem, data characteristics, and desired outcomes, often involving trial-and-error.

Explain your phd and its outcomes. ;; Looking for a counter examples, what are FCJ. Built algorithms to compute, then computed them

Q2. What is the difference between Type I vs Type II error? ;; Type I error occurs when the null hypothesis is true, but it is rejected. Type II error occurs when the null hypothesis is false, but it is not rejected.

Q3. What is [Linear Regression](#)? ;; Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables.

What do the terms P-value, coefficient, R-Squared value mean? ;; P-value indicates the significance of the coefficient, coefficient represents the strength and direction of the relationship, and R-Squared value measures the proportion of variance explained by the model.

What are the assumptions required for [Linear Regression](#)? ;; The assumptions include a linear relationship between dependent and independent variables, normally distributed and independent errors, minimal multicollinearity among explanatory variables.

What is a statistical interaction? ;; Statistical interaction occurs when the effect of one variable on a dependent variable is dependent on the value of another variable.

What is selection bias? ;; Selection bias refers to a systematic error in sampling that results in a sample that is **not representative of the population**, leading to incorrect conclusions about the population.

Ipynb

[Printing without code : Documentation & Meetings/ nbconvert](#)

<https://stackoverflow.com/questions/49907455/hide-code-when-exporting-jupyter-notebook-to-html>

```
jupyter nbconvert stock_analysis.ipynb --no-input --to pdf
```

```
jupyter nbconvert --to html --no-input --no-prompt phi_analysis.ipynb
```

```
--clear -output
```

```
jupyter nbconvert --to html --TemplateExporter.exclude_input=True Querying.ipynb
```

J

Table of Contents

- [Java vs JavaScript](#)
- [JavaScript](#)
- [Johnson–Lindenstrauss lemma](#)
- [Joining Datasets](#)
- [Json to Yaml](#)
- [Json](#)
- [Junction Tables](#)
- [jinja template](#)

Java Vs Javascript

Difference Between Java and JavaScript

Although their names are similar, **Java** and **JavaScript** are fundamentally different languages designed for different purposes. Below is a comparison between the two:

Aspect	Java	JavaScript
Type	Object-Oriented Programming Language	Scripting Language
Use	General-purpose, used for desktop, mobile, and enterprise applications	Primarily used for web development (front-end and back-end)
Execution	Runs on the Java Virtual Machine (JVM)	Runs in the browser or on server-side (Node.js)
Compiled or Interpreted	Compiled to bytecode, then executed by the JVM	Interpreted directly by the browser or Node.js
Syntax	Strongly typed; requires defining data types	Loosely typed; variables can change types
Concurrency	Supports multithreading	Single-threaded, but supports asynchronous programming (e.g., with callbacks, promises)
Platform Dependency	Platform-independent (write once, run anywhere)	Platform-independent, mainly within the context of the web
Main Use Case	Enterprise applications, Android development, large systems	Dynamic web pages, front-end and server-side scripting for web applications
Libraries/Frameworks	Spring, Hibernate, JavaFX, Android SDK	React, Angular, Vue.js (front-end), Node.js (back-end)
Syntax Example	<code>System.out.println("Hello, World!");</code>	<code>console.log("Hello, World!");</code>

Key Points:

- **Java** is used for building **large-scale applications**, including desktop apps and Android apps. It is strongly typed, compiled, and can handle multithreading.
- **JavaScript** is mainly used for **web development**, both for the front-end (managing user interfaces) and back-end (using Node.js), and is more flexible with dynamic typing and asynchronous behavior.

Javascript

Johnson–Lindenstrauss lemma

math

https://youtu.be/9-Jl0dxWQs8?list=PLZx_FHIHR8AwKD9csfl6SI_pgCXX19eer&t=1125

The number of vectors that can be fit into a spaces grows exponentially.

Useful for **LLM** in storing ideas.

Introduction

Plotting M>N almost orthogonal vectors in N-dim space

Optimisation process that nudges them towards being perpendicular between 89-91 degrees

Introduction

```
import torch
import matplotlib.pyplot as plt
from tqdm import tqdm

# List of vectors in some dimension, with many
# more vectors than there are dimensions
num_vectors = 10000
vector_len = 100
big_matrix = torch.randn(num_vectors, vector_len)
big_matrix /= big_matrix.norm(p=2, dim=1, keepdim=True)
big_matrix.requires_grad_(True)

# Set up an optimization loop to create nearly-perpendicular vectors
optimizer = torch.optim.Adam([big_matrix], lr=0.01)
num_steps = 250

losses = []

dot_diff_cutoff = 0.01
big_id = torch.eye(num_vectors, num_vectors)

for step_num in tqdm(range(num_steps)):
    optimizer.zero_grad()

    dot_products = big_matrix @ big_matrix.T
    # Punish deviation from orthogonality
    diff = dot_products - big_id
    loss = (diff.abs() - dot_diff_cutoff).relu().sum()

    # Extra incentive to keep rows normalized
    loss += num_vectors * diff.diag().pow(2).sum()

    loss.backward()
    optimizer.step()
    losses.append(loss.item())

# Plot loss curve
plt.plot(losses)
plt.grid(True)
plt.show()

# Compute angle distribution
dot_products = big_matrix @ big_matrix.T
norms = torch.sqrt(torch.diag(dot_products))
normed_dot_products = dot_products / torch.outer(norms, norms)
angles_degrees = torch.rad2deg(torch.acos(normed_dot_products.detach()))

# Use this to ignore self-orthogonality
self_orthogonality_mask = ~(torch.eye(num_vectors, num_vectors).bool())
plt.hist(angles_degrees[self_orthogonality_mask].numpy().ravel(), bins=1000, range=(0, 180))
```

```
plt.grid(True)  
plt.show()
```

Joining Datasets

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/Joining.ipynb

```
# Merge  
df1 = pd.DataFrame({'key': ['A', 'B'], 'value': [1, 2]})  
df2 = pd.DataFrame({'key': ['A', 'B'], 'value': [3, 4]})  
merged_df = pd.merge(df1, df2, on='key')  
  
# Concat  
concat_df = pd.concat([df1, df2])  
  
# Join  
df1.set_index('key', inplace=True)  
df2.set_index('key', inplace=True)  
joined_df = df1.join(df2, lsuffix='_left', rsuffix='_right')
```

Merging datasets for completeness (also see [SQL Joins](#)).

Json To Yaml

[Json](#)

[yaml](#)

```
{  
    "json": [  
        "rigid",  
        "better for data interchange"  
    ],  
    "yaml": [  
        "slim and flexible",  
        "better for configuration"  
    ],  
    "object": {  
        "key": "value",  
        "array": [  
            {  
                "null_value": null  
            },  
            {  
                "boolean": true  
            },  
            {  
                "integer": 1  
            },  
            {  
                "alias": "aliases are like variables"  
            },  
            {  
                "alias": "aliases are like variables"  
            }  
        ]  
    },  
    "paragraph": "Blank lines denote\\nparagraph breaks\\n",  
    "content": "Or we\\ncan auto\\nconvert line breaks\\nto save space",  
    "alias": {  
        "bar": "baz"  
    },  
    "alias_reuse": {  
        "bar": "baz"  
    }  
}
```

```

---  

# <- yaml supports comments, json does not  

# did you know you can embed json in yaml?  

# try uncommenting the next line  

# { foo: 'bar' }  
  

json:  

  - rigid  

  - better for data interchange  

yaml:  

  - slim and flexible  

  - better for configuration  

object:  

  key: value  

array:  

  - null_value: null  

  - boolean: true  

  - integer: 1  

  - alias: aliases are like variables  

  - alias: aliases are like variables  

paragraph: |  

  Blank lines denote  

  paragraph breaks  

content: |-  

  Or we  

  can auto  

  convert line breaks  

  to save space  

alias:  

  bar: baz  

alias_reuse:  

  bar: baz

```

Json

Stands for [javascript object notation](#)

- records separated by commas
- keys & strings wrapped by double quotes
- good choice for data transport

JSON data embedded inside of a string, is an example of semi-structured data. The string contains all the information required to understand the structure of the data, but is still for the moment just a string -- it hasn't been structured yet. The Raw JSON stored by Airbyte during ELT is an example of semi-structured data. This looks as follows:

	<u>_airbyte_data</u>
Record 1	\\"{"id": 1, "name": "Mary X"}\\"
Record 2	\\"{"id": 2, "name": "John D"}\\"

Junction Tables

Jinja Template

Resources

[LINK](#)

Practical

jinja2 works with python 3.

```
>>> import jinja2
>>> env = jinja2.Environment()
>>> template = env.from_string("Hello {{ name }}!")
>>> template.render(name="World")
'Hello World!'
```

Renders templates with variable substitutions

You can use tags too.

```
{# templates/all.txt #-}
Hello {{ name }},

{% if score > 80 %}
I'm happy to inform you that you did very well on today's {{ test_name }}.
{% else %}
Your score on today's test {{ test_name }} could have been better. More
studying, less vampire hunting!
{% endif %}
You achieved {{ score }} out of {{ max_score }} points!

See you tomorrow,

Ms. Calendar
```

Get gpt to generate example if necessary.

can get a csv to export the data.

context dictionaries are used can do html and flask.

jinja used to manage web pages

Flask

makes me think about how [Quartz](#) is constructed.

About

Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to [Python](#) syntax. Then the template is passed data to render the final document.

Most popularized by [dbt](#). Read more on the [Jinja Documentation](#).

integrates with [Flask](#).

K

Table of Contents

- K-means
- K-nearest neighbours
- K_Means.py
- Kaggle Abalone regression example
- Kernelling
- Key Differences of Web Feature Server (WFS) and Web Feature Server (WFS)
- Kmeans vs GMM
- Knowledge Graph
- Knowledge Graphs with Obsidian
- Knowledge Work
- Knowledge graph vs RAG setup
- kubernetes

K Means

K-means clustering is an [Unsupervised Learning](#) algorithm that partitions data into (k) clusters. Each data point is assigned to the cluster with the nearest centroid.

The algorithm partitions a dataset into k clusters by assigning data points to the closest cluster mean. The means are updated iteratively until convergence is achieved.

In [ML_Tools](#) see: [K_Means.py](#)

Key Features

- Unsupervised Learning: K-means organizes unlabeled data into meaningful groups without prior knowledge of the categories.
- [Hyperparameter](#) k: The number of clusters must be specified beforehand. The optimal number of clusters can be determined using [WCSS](#) and [elbow method](#).

Algorithm Process:

1. Randomly choose k initial centroids.
2. Assign each data point to the nearest centroid.
3. [Recalculate](#) the centroids based on the current cluster assignments.
4. Repeat steps 2 and 3 until convergence (i.e., centroids no longer change significantly).

Visualization: Scatterplots can be used to visualize clusters and their centroids.

Adaptability: K-means can be updated with new data and allows for comparison of changes in centroids over time.

The initial centroids can effect the end results.

To correct this the algo is run multiple times with varying starting positions.

The centroids are updated after each iteration.

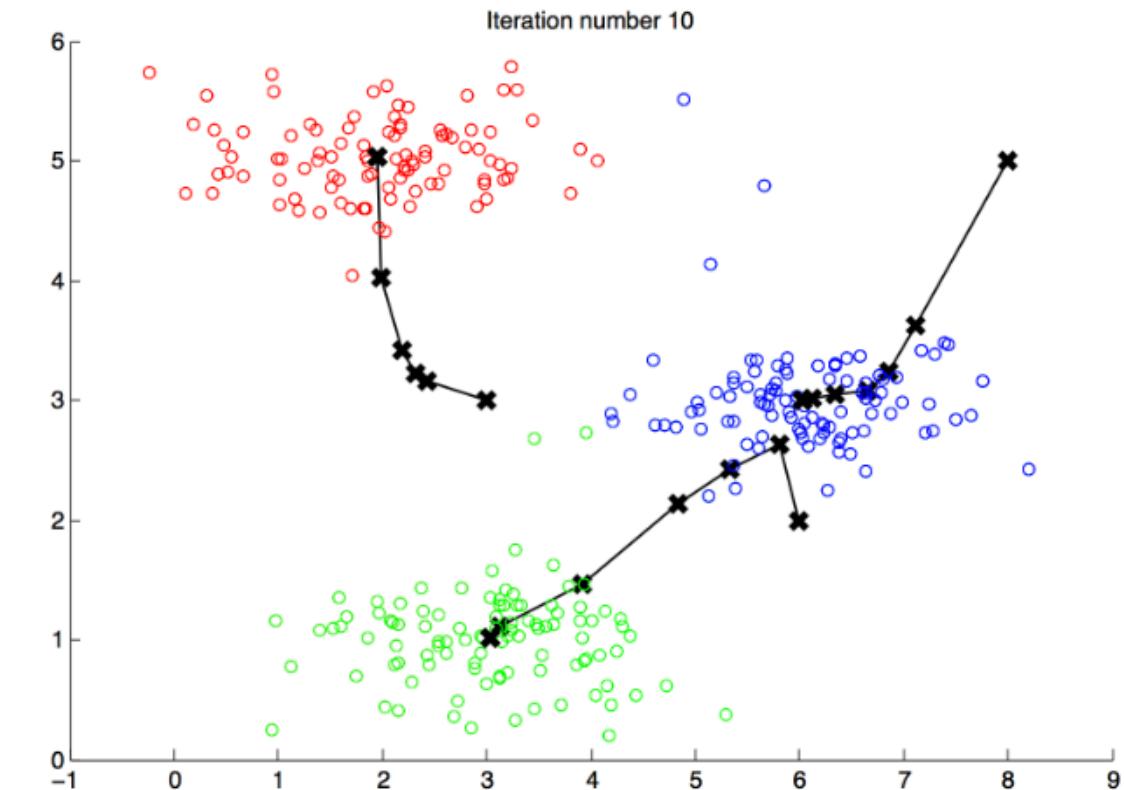


Figure 1: The expected output.

Limitations

- Sensitivity to Initialization: The algorithm is sensitive to the initial placement of centroids, which can affect the final clustering outcome.
- Predefined Number of Clusters: The number of clusters k must be specified in advance, which may not always be straightforward.

Resources

- [Statquest Video on K-means](#)

K Nearest Neighbours

K-nearest Neighbors is a non-parametric method used for both [classification](#) and [regression](#) tasks. It classifies a sample by a majority vote of its neighbors, assigning the sample to the class most common among its (k) nearest neighbors, where (k) is a small positive integer.

How It Works

- **Classification:** When a new data point needs classification, KNN identifies its (k) nearest neighbors in the training data based on feature similarity. The class label most common among these neighbors is assigned to the new data point.

- **Regression:** For regression tasks, KNN predicts the average value of the (k) nearest neighbors.

Applications

- [Recommender systems](#)
- Pattern recognition

Key Points

- **Non-parametric:** KNN does not make any assumptions about the underlying data distribution.
- **Supervised Learning:** Despite the note's mention of unsupervised learning, KNN is actually a supervised learning algorithm because it requires labeled training data.

Use Cases

- KNN is useful for tasks where the decision boundary is irregular and not easily captured by parametric models ([parametric vs non-parametric models](#)). It is simple to implement and understand but can be computationally expensive with large datasets.

Understanding K-nearest Neighbors

KNN is a straightforward algorithm that relies on the proximity of data points to make predictions. It is particularly effective in scenarios where the relationship between features and the target variable is complex and non-linear.

- **Choice of (k):** The value of (k) is crucial. A small (k) can be sensitive to noise, while a large (k) can smooth out the decision boundary too much.
- **Distance Metric:** The choice of distance metric (e.g., Euclidean, Manhattan) affects how neighbors are determined and can impact the algorithm's performance.

KNN is best suited for smaller datasets due to its computational intensity, as it requires calculating the distance between the new data point and all existing data points.

K-nearest neighbours

- Classifies a new data point based on the majority class of its k nearest neighbors.
- Simple but computationally expensive for large datasets.

K_Means.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/Clustering/KMeans/K_Means.py

Key Concepts Used in the Script

1. Data Loading:

- The script reads data from a CSV file (`penguins.csv`) and uses a sample dataset with random features for demonstration purposes.

2. Data Preprocessing:

- **Standardization:** Features are standardized using `sklearn.preprocessing.scale` and `StandardScaler` to ensure that all features contribute equally to the clustering process.

3. Feature Selection:

- Specific features, such as `bill_length_mm` and `bill_depth_mm`, are selected for clustering.

4. K-Means Clustering:

- The core clustering algorithm is applied with `n_clusters=3`.
- Outputs include cluster centroids and labels for each data point.

5. Visualization:

- Scatter plots are used to display the clustering results, highlighting the cluster centroids.

6. Evaluation of Optimal Clusters:

- **Elbow Method:** This method iterates through different numbers of clusters to determine the optimal number based on the within-cluster sum of squares (WCSS).

7. Cluster Assignment:

- Labels are assigned to data points, and the results are visualized to show the clustering outcome.

8. Exploratory Analysis:

- The script examines the impact of different numbers of clusters using an example function (`scatter_elbow`).

Kaggle Abalone Regression Example

Task: For each model as we tune the hyperparameters what happens to the (RMSLE) metric (scatter metric against hyperparameter).

Using **Root Mean Squared Logarithmic Error** RMSLE to evaluate.

Practice with model and feature engineering ideas, create visualizations

- [] create eda for blog post
- [] create model training optuna for blog post

Questions

[Hyperparameter](#) [Hyperparameter](#) tuning can be done with [Hyperparameter](#)

[Cross Validation](#) Both (sklearn) `StratifiedKFold` and `RepeatedStratifiedKFold` can be very effective when used on classification problems with a severe class imbalance. They both *stratify* the sampling by the class label; that is, they split the dataset in such a way that preserves approximately the same class distribution (i.e., the same percentage of samples of each class) in each subset/fold as in the original dataset. However, a single run of `StratifiedKFold` might result in a noisy estimate of the model's performance, as different splits of the data might result in very different results. That is where `RepeatedStratifiedKFold` comes into play.

`RepeatedStratifiedKFold` allows improving the estimated performance of a machine learning model, by simply repeating the [cross-validation](#) procedure multiple times (according to the `n_repeats` value), and reporting the *mean* result across all folds from all runs. This *mean* result is expected to be a more accurate estimate of the model's performance

Kernelling

[Kernelling](#) is a technique where the [Support Vector Machines|SVM](#) uses a kernel function to map the dataset into a higher-dimensional space, making it easier to identify separable clusters that may not be apparent in the original low-dimensional space.

Kernel Trick:

- When the data cannot be separated by a straight line or plane in its original (low-dimensional) space, SVM uses a technique called kernelling to project the data into a higher dimension where it becomes easier to separate.
- The Kernel Trick allows the transformation of data into a higher dimension without explicitly computing the transformation. There are different types of kernels, with common examples being:
 - Polynomial kernel
 - Radial Basis Function (RBF) or exponential kernel

Key Differences of Web Feature Server (WFS) and Web Feature Server (WFS)

1. Data Type:

- **Web Map Tile Service (WMTS)**: Serves image tiles (raster data).
- **Web Feature Server (WFS)**: Serves geographic features (vector data).

2. Use Case:

- **WMTS**: Ideal for applications needing fast rendering of static maps, such as online map viewers.
- **WFS**: Suitable for applications requiring access to and manipulation of raw geographic data, such as spatial analysis and GIS applications.

3. Performance:

- **WMTS**: High performance due to pre-rendered and cached tiles, optimized for rapid delivery.
- **WFS**: Performance depends on the complexity of the data and queries, typically slower than WMTS due to on-the-fly data retrieval and processing.

4. Interactivity:

- **WMTS**: Limited interactivity, primarily for viewing maps.
- **WFS**: High interactivity, supporting complex queries and data manipulation.

Example Scenarios

5. **WMTS**: A web application displaying a world map with zoom and pan functionality. The map is made up of pre-rendered image tiles that load quickly as the user navigates.
6. **WFS**: An environmental monitoring system that allows users to query and retrieve data about specific geographic features, such as the location and attributes of water bodies, for analysis and reporting.

In summary, WMTS is focused on efficiently serving map images for fast visualization, while WFS provides access to detailed, manipulable geographic feature data for more in-depth spatial analysis and querying.

Key Differences Between k-Means and GMM

Cluster Shape

- **k-Means**: Assumes clusters are spherical and equidistant from their centroids.
- **GMM**: Models clusters as Gaussian distributions, allowing for **different shapes** (e.g., ellipses) by incorporating mean and covariance matrices. GMM can model clusters of varying **shapes and sizes** by adjusting the **Covariance Structures** (e.g., full, diagonal, spherical).

Probability-Based Assignments

- **k-Means:** Assigns each point deterministically to the nearest cluster centroid.
- **GMM:** Provides a [probability Distributions|distribution](#) for cluster membership, making it a soft clustering method.
- GMM handles overlapping clusters effectively by assigning **probabilities** to data points for each cluster, instead of enforcing hard boundaries like k-means.

Flexibility

- **k-Means:** Performs well when clusters are **spherical** and well-separated.
- **GMM:** Handles **overlapping clusters** and clusters with **different shapes**, leveraging its covariance modeling capability.

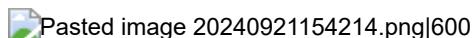
Knowledge Graph

[!Summary]

Knowledge graphs (KGs) enable large language models (LLMs) to generate more accurate, trustworthy AI outputs. Neo4j is leader in this space and make use of KG through such as generative AI techniques like GraphRAG.

- Knowledge graphs are critical for managing complex data relationships and making strategic AI-driven decisions.
- The combination of KGs and LLMs improves AI accuracy, diversity of viewpoints, and [explainability](#).

A **knowledge graph** is a structured representation of knowledge that captures entities (e.g., people, places, concepts) and the relationships between them. Knowledge graphs are often used to represent and store factual information in a way that machines can easily query and understand. They use a **graph structure** where: **Nodes** represent entities (like "Company A" or "Person B"). **Edges** represent relationships between those entities (like "works at" or "founded").



[!important]

KGs act as a control for Large Language Models (LLMs) by enabling knowledge-based reasoning based on the connections in the data

[!attention]

- No significant limitations or concerns were highlighted, but the implementation of KGs may require technical expertise and resources.

[!Follow up questions]

- [] How does the integration of knowledge graphs and LLMs improve explainability in AI-generated responses?

[!Link] <https://neo4j.com/blog/genai-knowledge-graph-deep-understanding/>

Key characteristics of knowledge graphs:

- **Structured data:** Information is represented in a highly structured form (triples: subject, predicate, object) that allows efficient querying and reasoning.
- **Semantic relationships:** Entities are connected by meaningful relationships, often using ontologies and taxonomies to organize the knowledge.
- **Reasoning and inference:** Some knowledge graphs can support reasoning capabilities, where new information can be inferred based on the existing relationships and rules (e.g., if "Person A works at Company B," and "Company B is in Industry C," it can infer that "Person A works in Industry C").

Example of a Knowledge Graph:

- **Nodes:** Barack Obama , United States , President .
- **Edges:** Barack Obama --> President of --> United States .

Knowledge Graphs With Obsidian

[!Summary]

Llama Index is a python package that can be used to create Knowledge graphs (KGs). There exists a method to integrate with Obsidian. This is a tutorial on how to set this up a **RAG** system with obsidian. Visualisation is done using the `pyvis` library.

Requires API to **LLM**.

[!Important]

- RAG improves **LLM** performance by utilizing external databases.
- Llama Index facilitates the transformation of Obsidian notes into a structured Knowledge Graph.
- The tutorial includes steps for setup, dependencies, and visualization.

[!attention]

- **LLMs** can still produce errors known as "hallucinations."
- Requires familiarity with Python and Jupyter Notebook.

[!Code set up]

[!Example]

The tutorial provides a code snippet for querying the Knowledge Graph about the assumptions of the Black-Scholes model, yielding detailed contextually relevant insights.

[!Follow up questions]

- [] How can RAG be applied to other data sources beyond Obsidian?
- [] What specific challenges might arise when integrating large datasets into a Knowledge Graph?
- [] Can you also set up with PDF's within Obsidian?

[!Link]

https://medium.com/@haiyangli_38602/make-knowledge-graph-rag-with-llamaindex-from-own-obsidian-notes-b20a350fa354 Knowledge Graph

Knowledge Work

Knowledge work refers to tasks that primarily involve handling or using information and require cognitive skills rather than manual labor. It is characterized by problem-solving, critical thinking, and the application of specialized knowledge.

Key Characteristics

- Problem Solving: Knowledge work often involves identifying, analyzing, and solving complex problems. This requires creativity, analytical skills, and the ability to synthesize information from various sources.
- Use of the [Scientific Method](#): Many knowledge work tasks, especially in research and development, rely on the scientific method. This involves forming hypotheses, conducting experiments, analyzing data, and drawing conclusions.
- Information Management: Knowledge workers must efficiently gather, process, and apply information to make informed decisions.

Examples of Knowledge Work

- Research and development
- Software development
- Data analysis
- Strategic planning
- Writing and content creation

Knowledge Graph Vs RAG Setup

Comparison: Knowledge Graph vs. RAG Setup

- **Knowledge Graphs** are structured representations of entities and their relationships, designed primarily for querying, reasoning, and storing factual information.
- **RAG setups** enhance generative models by retrieving external knowledge (from unstructured or semi-structured data) and integrating it into the generation process.

While not the same, these two concepts can be used together to build systems that combine structured knowledge retrieval with the natural language generation capabilities of RAG models.

While **knowledge graphs** and **RAG** are distinct, they can be integrated to improve certain systems:

- A **RAG model** could use a **knowledge graph** as the retrieval source. Instead of retrieving unstructured text documents, the RAG model could retrieve structured, factual triples from a knowledge graph and incorporate this into the generation process. This would improve the accuracy of fact-based questions and answers.

A [Knowledge Graph](#) and a [Retrieval-Augmented Generation \(RAG\)](#) setup are related but distinct concepts, particularly in how they handle knowledge representation and retrieval. While they can complement each other in certain applications, they serve different purposes and operate in different ways.

Aspect	Knowledge Graph	RAG Setup
Purpose	Stores and organizes knowledge for querying and reasoning	Combines retrieval of external information with text generation
Data Structure	Highly structured (graph with nodes and edges)	Unstructured or semi-structured (documents, text snippets)
Retrieval Mechanism	Queries are made through graph traversal or SPARQL-like languages	Information is retrieved via search mechanisms (e.g., dense embeddings)
Usage	Often used for querying factual data, answering structured queries, Semantic Relationships	Used to enhance the factual accuracy of generative models by retrieving external data
Reasoning and Inference	Capable of logical reasoning based on relationships	Does not perform reasoning; it retrieves and integrates relevant text
Scalability	Requires careful design to manage large, complex graphs	Can handle large text corpora, but retrieval quality affects the final generation
Generative Capabilities	Not generative (focused on querying existing knowledge)	Generative (synthesizes and generates natural language responses)

Kubernetes

It's a platform that allows you to run and orchestrate container workloads. **Kubernetes has become the de-facto standard** for your cloud-native apps to (auto-) **scale-out** and deploy your open-source zoo fast, cloud-provider-independent. No lock-in here. Kubernetes is the **move from infrastructure as code** towards **infrastructure as data**, specifically as [YAML](#). With Kubernetes, developers can quickly write applications that run across multiple operating environments. Costs can be reduced by scaling down.

L

Table of Contents

- [LBFGS](#)
- [LLM Evaluation Metrics](#)
- [LLM](#)
- [LSTM](#)
- [Label encoding](#)
- [Labelling data](#)
- [Langchain](#)
- [Language Model Output Optimisation](#)
- [Language Models Large \(LLMs\) vs Small \(SLMs\)](#)
- [Language Models](#)
- [Lasso](#)
- [Latency](#)
- [Latent Dirichlet Allocation](#)
- [Learning Styles](#)
- [LightGBM vs XGBoost vs CatBoost](#)
- [LightGBM](#)
- [Linear Discriminant Analysis](#)
- [Linear Regression](#)
- [Linked List](#)
- [Load Balancing](#)
- [Local Interpretable Model-agnostic Explanations](#)
- [Logical Model](#)
- [Logistic Regression Statsmodel Summary table](#)
- [Logistic Regression does not predict probabilities](#)
- [Logistic Regression](#)
- [Logistic regression in sklearn & Gradient Descent](#)
- [Looker Studio](#)
- [Loss function](#)
- [Loss versus Cost function](#)
- [lambda architecture](#)
- [learning rate](#)
- [lemmatization](#)

Lbfgs

LBFGS stands for Limited-memory Broyden-Fletcher-Goldfarb-Shanno, which is an [Optimisation function](#) optimization algorithm used to find the minimum of a function. In the context of [logistic regression](#), LBFGS is a method for optimizing the cost function to find the optimal [model parameters](#) (such as the intercept and coefficients).

Here's a breakdown of the key features of LBFGS:

1. Quasi-Newton Method: LBFGS is a type of Quasi-Newton method, which approximates the inverse of the Hessian matrix (second-order derivatives of the cost function). Instead of computing the full Hessian matrix, it uses an approximation, which makes it more efficient for large datasets.

-
2. Limited Memory: The "limited-memory" part refers to the fact that LBFGS does not store the entire Hessian matrix, which is computationally expensive and memory-intensive. Instead, it keeps a limited amount of information from previous iterations, making it well-suited for large-scale problems where full memory-based methods might not be feasible.
 3. Optimization for Smooth, Differentiable Functions: It is designed to optimize smooth, differentiable functions like the [cost function](#) in logistic regression.

In the context of logistic regression with sklearn, LBFGS is used as a solver for optimization. When you set `solver='lbfgs'`, [Sklearn's](#) logistic regression uses this algorithm to iteratively adjust the model parameters (the intercept and coefficients) to minimize the logistic loss (the cost function) while possibly incorporating regularization.

LBFGS is often preferred for its efficiency and ability to converge quickly without needing a lot of iterations, especially when the number of features is large.

Llm Evaluation Metrics

[LLM Evaluation Metrics](#)

- BLEU,
- ROUGE,
- perplexity which quantify the similarity between generated text and reference outputs.

[LLM](#)

A Large Language Model (LLM) is a type of language model designed for language understanding and generation. They can perform a variety of tasks, including:

- Text generation
- Machine translation
- Summary writing
- Image generation from text
- Machine coding
- Chatbots or Conversational AI

Questions

- [How do we evaluate of LLM Outputs](#)
- [Memory|What is LLM memory](#)
- [Relationships in memory|Managing LLM memory](#)
- [Mixture of Experts](#): having multiple experts instead of one big model.
- [Distillation](#)
- Mathematics on the parameter usage [Attention mechanism](#)
- Use of [Reinforcement learning](#) in training [Chain of thought](#) methods in LLM's ([deepseek](#))

How do Large Language Models (LLMs) Work?

Large [Language Models](#) (LLMs) are a type of artificial intelligence model that is designed to understand and generate human language. Key aspects of how they work include:

- Word Vectors: LLMs represent words as long lists of numbers, known as word vectors ([standardised/Vector Embedding|word embedding](#)).
- Neural Network Architecture: They are built on a neural network architecture known as the [Transformer](#). This architecture enables the model to identify relationships between words in a sentence, irrespective of their position in the sequence.
- [Transfer Learning](#): LLMs are trained using a technique known as transfer learning, where a pre-trained model is adapted to a specific task.

Characteristics of LLMs

- **Non-Deterministic:** LLMs are non-deterministic, meaning the types of problems they can be applied to are of a probabilistic nature ([temperature](#)).
- Data Dependency: The performance and behaviour of LLMs are heavily influenced by the data they are trained on.

Lstm

What is LSTM

LSTM (Long Short-Term Memory) networks are a specialized type of Recurrent Neural Network (RNN) designed to overcome the [vanishing and exploding gradients problem](#) that affects traditional [Recurrent Neural Networks](#).

LSTMs address this challenge through their unique architecture.

Used for tasks that require the retention of information over time, and problems involving [sequential data](#).

The key strength of LSTMs is their ability to manage [long-term dependencies](#) using their [gating mechanisms](#).

Key Components of LSTM Networks:

Resources: [Understanding LSTM Networks](#)

$\$x_t$ input, $\$h_t$ output, cell state $\$C_t$, conveyer belt



Memory Cell:

- The core of an LSTM network is the memory cell, which maintains information over long time intervals. This cell helps store, forget, or pass on information from previous time steps.

Gates:

- Input Gate: Controls how much of the input should be allowed into the memory cell.
- Forget Gate: Determines which information should be discarded from the memory cell.
- Output Gate: Controls what part of the cell's memory should be output as the hidden state for the current time step.

Introduction

These gates are regulated by **sigmoid** activation, which output values between 0 and 1, acting like a filter to determine the amount of information that should pass through. This gate mechanism allows the LSTM network to maintain a balance between retaining relevant data and discarding unnecessary information over time.

Why is LSTM less favourable over using transformers

[!Summary]

Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network ([RNN](#)), are less favorable than [Transformer](#) for many modern tasks, especially in Natural Language Processing ([NLP](#)). LSTMs process sequences of data one step at a time, making them inherently sequential and difficult to parallelize.

Transformers, on the other hand, leverage a self-attention mechanism that allows them to process entire sequences simultaneously, leading to faster training and the ability to capture long-range dependencies more effectively.

Mathematically, LSTM's sequential nature leads to slower computations, while the Transformer's attention mechanism computes relationships between all tokens in a sequence, allowing better scalability and performance for tasks like translation, summarization, and language modeling.

[!Breakdown]

Key Components:

- Sequential Processing in LSTM: Each time step relies on the previous one, creating a bottleneck for long sequences.
- Self-Attention Mechanism in Transformers: Allows simultaneous processing of all elements in a sequence.
- Parallelization: Transformers leverage parallel computing more effectively due to non-sequential data processing.
- Positional Encoding: Used by Transformers to retain the order of the sequence, overcoming the need for explicit recurrence.

[!important]

- LSTMs are slower in training due to their sequential nature, as calculations depend on previous states.
- Transformers efficiently handle long-range dependencies using self-attention, which calculates the relationships between tokens directly without needing previous time steps.

[!attention]

- LSTMs suffer from vanishing/exploding gradient issues, especially in long sequences, limiting their effectiveness for long-term dependencies.
- Transformers require more data and computational power to train, which can be a limitation in resource-constrained environments.

[!Example]

In a language translation task, LSTMs process words sequentially, making them less efficient in handling long sentences. In contrast, a Transformer can analyze the entire sentence at once, using self-attention to determine relationships between all words, leading to faster and more accurate translations.

[!Follow up questions]

Introduction

- How does the attention mechanism in Transformers improve the model's ability to capture long-range dependencies compared to LSTM's **cell structure?**
- In what cases might LSTM still be a better option over Transformers, despite their limitations?

[!Related Topics]

- [Attention mechanism](#) in deep learning
- [BERT](#) (Bidirectional Encoder Representations from Transformers)
- [GRU](#) (Gated Recurrent Unit) as an alternative to LSTM

Example Workflow in Python using Keras:

In this example, we define a simple LSTM model in [Keras](#) for a time series forecasting task. The model processes sequences with 1000 time steps, and the LSTM layer has 50 units, followed by a fully connected (Dense) layer for the final prediction.

```
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense

# Sample data: time series with 1000 timesteps and 1 feature
time_steps = 1000
features = 1
X_train = np.random.rand(1000, time_steps, features)
y_train = np.random.rand(1000)

# Define LSTM model
model = Sequential()
model.add(LSTM(50, activation='tanh', return_sequences=False, input_shape=(time_steps, features)))
model.add(Dense(1)) # Output layer for regression tasks

model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=64)
```

notes

[LSTM](#) How to implement [LSTM](#) with [PyTorch](#)? <https://lightning.ai/lightning-ai/studios/statquest-long-short-term-memory-lstm-with-pytorch-lightning?view=public§ion=all> without lightning - there is a script there

[LSTM](#)

Label Encoding

Labelling Data

Possible missing labelling or bias in the data, or under-represented data. Construction of the data set comes from the group collecting it.

Examples:

- ImageNet

Langchain

[Python](#) framework

For building apps with [LLM](#) and interaction with them and combining models.

Its end to end, through composability

Example: [Pandas Dataframe Agent](#)

Modules

models

interface

prompts

chains

sequences of calls to a LLM

Memory

Indexes

Agents and tools

set up [Agentic Solutions](#)

Language Model Output Optimisation

What techniques from [information theory](#) can be used to measure and optimize the amount of information conveyed by a language model?

In information theory, several techniques can be applied to measure and optimize the amount of information conveyed by a [Language Model](#).

1. Entropy: Entropy measures the uncertainty or unpredictability of a random variable. In the context of language models, it can be used to quantify the uncertainty in predicting the next word in a sequence. Lower entropy indicates more predictable and informative outputs.
2. Cross Entropy: This measures the difference between two probability distributions. For language models, cross-entropy can be used to evaluate how well the predicted distribution of words matches the actual distribution in the data. Minimizing cross-entropy during training helps optimize the model's predictions.
3. Perplexity: Perplexity is a common metric for evaluating language models. It is the exponentiation of the cross-entropy and represents the model's uncertainty in predicting the next word. Lower perplexity indicates a better-performing model.
4. Mutual Information: This measures the amount of information shared between two variables. In language models, it can be used to assess how much information about the input is retained in the output, helping to optimize the model's ability to convey relevant information.
5. KL Divergence: Kullback-Leibler divergence measures how one probability distribution diverges from a second, expected probability distribution. It can be used to optimize language models by minimizing the divergence between the predicted and true distributions.
6. Information Bottleneck: This technique involves finding a balance between compressing the input data and preserving relevant information for the task. It can be used to optimize models by focusing on the most informative features.
7. Rate-Distortion Theory: This involves finding the trade-off between the fidelity of the information representation and the amount of compression. It can be applied to optimize language models by balancing the complexity of the model with the quality of the information conveyed.
8. Attention Mechanism: While not strictly an information theory concept, attention mechanisms in neural networks can be seen as a way to dynamically allocate information processing resources, focusing on the most informative parts of the input.

Overview

Language models can be categorized into **large language models (LLM)** and **small language models (SLM)**. While LLMs boast extensive general-purpose knowledge and capabilities, SLMs offer distinct advantages in certain scenarios, particularly when it comes to efficiency, resource constraints, and task-specific environments.

Key Differences

Aspect	LLMs	SLMs
Accuracy	Higher accuracy across broad tasks due to large datasets and extensive training.	Comparable performance in domain-specific tasks after fine-tuning.
Efficiency	Computationally expensive; requires significant resources for training and inference.	More resource-efficient; suited for edge devices and real-time applications.
Interpretability	Often a "black box"; difficult to explain decision-making.	More interpretable due to simpler architecture.
Generality	General-purpose; capable of handling a wide range of tasks.	Task-specific; excels in specific domains and structured data.

Language Models

A language model is a machine learning model that is designed to understand, generate, and predict human language.

It does this by analyzing large amounts of text data to learn the patterns, structures, and relationships between words and phrases.

They work by assigning probabilities to sequences of words, allowing them to predict the next word in a sentence or generate coherent text based on a given prompt.

Related to: [LLM Small Language Models|SLM](#)

Lasso

1. L1 Regularization (Lasso Regression):

- o In L1 regularization, a penalty proportional to the absolute value of the coefficients is added to the loss function.
- o The L1 penalty tends to shrink some coefficients to exactly zero, effectively performing **feature selection** by eliminating irrelevant features.
- o **Lasso regression** (Least Absolute Shrinkage and Selection Operator) is an example of a model that uses L1 regularization.

The L1-regularized loss function is: $\text{Loss} = \text{MSE} + \lambda \sum_{i=1}^n |w_i|$ where λ is the regularization parameter, w_i are the model weights, and MSE is the mean squared error.

For **Lasso Regression (L1)**:

```
from sklearn.linear_model import Lasso

# Initialize a Lasso model
model = Lasso(alpha=0.1) # alpha is the regularization strength
model.fit(X_train, y_train)
```

[LINK](#)

In L1 regularization, a penalty term proportional to the absolute value of the weights is added to the loss function. This encourages sparsity in the model by driving some weights to exactly zero, effectively performing feature selection.

- Adds a penalty term proportional to the **absolute value of the model's coefficients**.
- Encourages sparsity by **driving some coefficients to exactly zero**.
- Useful for **Feature Selection**, as it tends to eliminate less important features by setting their corresponding coefficients to zero.
- Can result in a sparse model with fewer features retained.
- Lasso regression adds a penalty term to the loss function proportional to the absolute value of the coefficients of the features. This encourages sparsity in the coefficient vector, effectively setting some coefficients to zero and performing feature selection.

Latency

Latent Dirichlet Allocation

Related terms:

- [topic modeling](#)
- [Semantic Relationships](#)
- [NLP](#)

Latent Dirichlet Allocation (LDA) is a generative probabilistic model used in Natural Language Processing (NLP) and machine learning for topic modeling. It assumes that documents are mixtures of topics, and topics are mixtures of words. The goal of LDA is to uncover the latent topics in a collection of text documents by identifying groups of words that frequently appear together in the same documents.

Libraries like gensim in Python are highlighted as being particularly suitable for topic modeling

Key Concepts:

- **Topic:** A distribution over a fixed vocabulary of words. A topic might represent a certain theme, like "sports" or "politics."
- **Document:** A mixture of topics. A document is modeled as a combination of topics with different proportions.
- **Words:** Each word in the document is associated with a topic based on the topic proportions of that document.

How LDA Works:

1. **Assume each document has a mixture of topics:** Each document is made up of a certain proportion of topics. For example, a document could be 70% about "sports" and 30% about "politics."
2. **Assume each topic is a distribution of words:** Each topic has a specific distribution over words. For example, the "sports" topic might have a high probability for words like "football," "game," and "score."
3. **Infer the topics from the documents:** LDA tries to discover these hidden topics from the words in the documents, without knowing the topics in advance.

Example:

Imagine you have a set of three simple documents:

1. **Document 1:** "Football is a great sport"
2. **Document 2:** "Basketball is also fun to play"
3. **Document 3:** "Soccer and football are similar sports"

LDA might identify two topics:

- **Topic 1:** Words related to "sports" (e.g., "football," "basketball," "soccer").
- **Topic 2:** Words related to "competition" or "play" (e.g., "game," "score," "fun").

The algorithm would assign a mixture of these topics to each document based on the words in the documents.

Example Code:

Here's a simple implementation of LDA using `sklearn` in Python:

```

import numpy as np
import pandas as pd
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
docs = [
    "Football is a great sport",
    "Basketball is also fun to play",
    "Soccer and football are similar sports"
]

# Step 1: Convert the documents into a document-term matrix (DTM)
vectorizer = CountVectorizer(stop_words='english')
dtm = vectorizer.fit_transform(docs)

# Step 2: Apply LDA to discover topics
lda = LatentDirichletAllocation(n_components=2, random_state=42)
lda.fit(dtm)

# Step 3: Display topics
terms = vectorizer.get_feature_names_out()

for index, topic in enumerate(lda.components_):
    print(f"Topic #{index + 1}:")
    print([terms[i] for i in topic.argsort()[-5:]]) # Print the top 5 words in the topic
    print()

```

Example Output:

```

# Topic #1:
# ['football', 'sports', 'game', 'score', 'play']
# Topic #2:
# ['basketball', 'soccer', 'sport', 'fun', 'great']

```

Output Explanation:

- **Topic #1** might have words like **football**, **sports**, **game**, because these terms appear frequently in the documents related to sports.
- **Topic #2** could have terms like **basketball**, **soccer**, **fun**, because these are associated with the activities discussed in the documents.

How to Interpret:

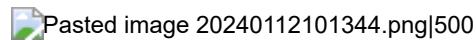
- **Document 1 ("Football is a great sport")**: LDA might classify this document as being 60% about Topic #1 (sports-related) and 40% about Topic #2 (competitive play).
- **Document 2 ("Basketball is also fun to play")**: LDA might classify this document as being 80% about Topic #2 (competitive play) and 20% about Topic #1 (sports-related).

Why Use LDA?

- **Topic Discovery:** It helps discover hidden themes in a large corpus of text data.
- **Dimensionality Reduction:** LDA reduces the complexity of the text data by modeling it with a smaller number of topics instead of many individual words.

Learning Styles

What does the data look like [continuous](#) or [categorical](#)?



[Unsupervised Learning Regression Classification](#)

[Unsupervised Learning Dimensionality Reduction Clustering](#)

Lightgbm Vs Xgboost Vs Catboost

This table summarizes the key differences and strengths of each [Gradient Boosting](#) framework.

Feature/Aspect	LightGBM (LGBM)	XGBoost	CatBoost
Tree Growth Strategy	Leaf-wise growth, leading to deeper trees and potentially better accuracy.	Level-wise growth, resulting in more balanced trees.	Ordered boosting, reducing overfitting and improving generalization.
Speed and Memory	High speed and low memory usage, especially with large datasets.	Balanced speed and accuracy with robust regularization options.	Competitive performance with minimal hyperparameter tuning.
Handling Categorical Features	Requires preprocessing (e.g., label encoding).	Requires preprocessing of categorical features.	Natively handles categorical features without preprocessing.
Regularisation	Supports regularization but not as robust as XGBoost.	Strong regularization options (L1 and L2) to prevent overfitting.	Utilizes techniques like ordered boosting to mitigate overfitting.
Use Cases	Ideal for large datasets and when computational efficiency is a priority.	Suitable for structured data and tabular datasets; widely used in competitions.	Useful for datasets with many categorical features and missing values.
Performance	Fast training and efficient on large datasets.	Accurate and flexible, often used in competitions.	Provides competitive performance, especially with categorical data.

Lightgbm

LightGBM is a gradient boosting framework that is designed for speed and efficiency. It is particularly well-suited for handling large datasets and high-dimensional data.

- **Tree Growth:** Splits the tree leaf-wise, which can lead to faster convergence compared to level-wise growth.
- **Learning Rate:** Similar to [Gradient Descent](#), LightGBM uses a learning rate to control the contribution of each tree.
- **DART:** A variant of LightGBM known for its performance.
- **Parameter Definition:** Requires parameters to be defined in a dictionary for model configuration.

[Watch Video Explanation](#)

Key Parameters

- **Learning Rate:** Controls the step size at each iteration while moving toward a minimum of the loss function.
- **Number of Leaves:** Determines the complexity of the tree model.

Advantages

- **Speed:** Renowned for its speed, often outperforming other gradient boosting implementations.
- **Memory Usage:** Optimizes memory usage, enabling efficient handling of large datasets.
- **Leaf-Wise Growth:** Grows trees leaf-wise, leading to faster convergence.
- **Parallel and GPU Learning:** Supports parallel and GPU learning for further speedup.

Use Cases

- **Large Datasets:** Ideal for applications where speed is crucial.
- **High-Dimensional Data:** Efficient when dealing with high-dimensional data and categorical features.

Linear Discriminant Analysis

Linear Regression

Description

Linear regression assumes [linearity](#) between the input features and the target variable. Assumes that the relationship between the independent variable(s) and the dependent variable is linear.

During the training phase, the algorithm adjusts the slope (m) and the intercept (b) of the line to minimize the [Loss function](#).

The linear regression model is represented as:

$$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

- y is the dependent variable (the variable we want to predict).
- x_1, x_2, \dots, x_n are the independent variables (features or predictors).
- $b_0, b_1, b_2, \dots, b_n$ are the coefficients (weights) associated with each independent variable.
- b_0 is the intercept term.

You evaluate the performance of your model by comparing its predictions to the actual values in a separate test dataset. Common metrics for evaluating regression models are:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R squared.

The goal of linear regression is to find the values of coefficients $b_0, b_1, b_2, \dots, b_n$ that minimize the sum of squared errors (SSE), also known as the residual sum of squares (RSS) or (MSE - mean square error).

Mathematically, SSE is a Loss function given by:

$$SSE = \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

where N is the number of observations, y_i is the actual value of the dependent variable for observation i , and \hat{y}_i is the predicted value based on the linear regression model.

The formula for the Regression Sum of Squares (SSR) in the context of linear regression is:

$$SSR = \sum_{i=1}^N (\hat{y}_i - \bar{y})^2$$

Where:

- \hat{y}_i is the predicted value of the dependent variable for observation i based on the linear regression model.
- \bar{y} is the mean of the observed values of the dependent variable.
- N is the total number of observations.

SSR measures the amount of variability in the dependent variable that is explained by the independent variables in the model. It reflects how well the regression model captures the relationship between the independent and dependent variables.

Total Sum of Squares (SST) represents the total variability in the dependent variable y . The relationship between SST, SSR, and SSE is given by:

$$SST = SSR + SSE$$

This equation reflects the decomposition of total variability into explained variability (SSR) and unexplained variability (SSE) due to errors.

Ordinary Least Squares

The Ordinary Least Squares method is used to minimize SSE. It achieves this by finding the values of that minimize the sum of squared differences between the observed and predicted values. The formulas for are derived by setting partial derivatives of SSE with respect to each coefficient to zero.

OLS is an analytical method

Gradient Descent

It iteratively updates coefficients to minimize error.

Gradient descent is an optimization algorithm used to minimize the cost function in linear regression by iteratively adjusting the model parameters (coefficients). Here's how it works with linear regression:

1. **Initialize Parameters:** Start with initial guesses for the coefficients (weights), typically set to zero or small random values.

2. **Compute Predictions:** Use the current coefficients to make predictions for the dependent variable \hat{y} using the linear regression model: $\hat{y} = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$
3. **Calculate the Cost Function:** Compute the loss function, SSE.
4. **Compute the Gradient:** Calculate the gradient of SSE function with respect to each coefficient. The gradient is a vector of partial derivatives indicating the direction and rate of change of the cost function:
$$\frac{\partial J}{\partial b_j} = -\frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i)x_{ij}$$
 Here, x_{ij} is the value of the j -th feature for the i -th observation.
5. **Update the Coefficients:** Adjust the coefficients in the opposite direction of the gradient to reduce the cost function. This is done using a learning rate α , which controls the size of the steps taken: $b_j = b_j - \alpha \frac{\partial J}{\partial b_j}$
6. **Iterate & Converge:** Repeat steps 2 to 5 until the cost function converges to a minimum or a predefined number of iterations is reached. The algorithm converges when the changes in the cost function or the coefficients become very small, indicating that the minimum has been reached.

Model Evaluation

- R squared
- Adjusted R squared takes into account the number of variables.
- F-statistic overall significance of model (lower worse).
- Feature Selection Use P>|t| column to decide whether to keep variable less than 0.05
- p-values in linear regression in sklearn

| 500 | 500

Impact of Extra Variables on Intercept:

When additional variables are introduced, it can impact the intercept (b_0) in the linear regression model. The intercept is the value of y when all independent variables (x_1, x_2, \dots, x_n) are zero. The presence of extra variables can affect the baseline value of the dependent variable.

Linked List

A **linked list** is a linear data structure in which elements (called **nodes**) are linked together using pointers. Unlike arrays, linked lists do not store elements in contiguous memory locations; instead, each node contains:

1. **Data** – The actual value stored in the node.
2. **Pointer (or Reference)** – A reference to the next node in the sequence.

Types of Linked Lists:

1. **Singly Linked List** – Each node has a pointer to the next node only.
2. **Doubly Linked List** – Each node has pointers to both the previous and next nodes.
3. **Circular Linked List** – The last node points back to the first node, forming a circular structure.
 - Circular **singly** linked list: Last node points to the first node.
 - Circular **doubly** linked list: Last node points to the first node, and first node points back to the last.

Advantages of Linked Lists:

- **Dynamic size** – Unlike arrays, they do not have a fixed size.
- **Efficient insertions/deletions** – Adding or removing elements does not require shifting (unlike arrays).

- **Efficient memory utilization** – Memory is allocated as needed.
-

Disadvantages of Linked Lists:

- **Extra memory usage** – Each node requires additional storage for pointers.
- **Slower access time** – No direct access like arrays; traversal is required.

Introduction

```
class Node:
    """A node in a singly linked list."""
    def __init__(self, data):
        self.data = data # Store data
        self.next = None # Pointer to the next node (initially None)

class LinkedList:
    """A simple singly linked list."""
    def __init__(self):
        self.head = None # Initialize the list as empty

    def append(self, data):
        """Add a new node at the end of the list."""
        new_node = Node(data)
        if not self.head: # If the list is empty
            self.head = new_node
        return
        last = self.head
        while last.next: # Traverse to the last node
            last = last.next
        last.next = new_node # Link the last node to the new node

    def display(self):
        """Print the linked list elements."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None") # Indicate the end of the list

    def delete(self, key):
        """Delete a node by value."""
        current = self.head

        # If the node to be deleted is the head
        if current and current.data == key:
            self.head = current.next
            current = None
            return

        prev = None
        while current and current.data != key:
            prev = current
            current = current.next

        if current is None: # If the key was not found
            return

        prev.next = current.next
        current = None
```

```

# Example usage
ll = LinkedList()
ll.append(10)
ll.append(20)
ll.append(30)
ll.display() # Output: 10 -> 20 -> 30 -> None

ll.delete(20)
ll.display() # Output: 10 -> 30 -> None

```

Load Balancing

Load balancing is a technique used to distribute incoming network traffic across multiple servers. This helps ensure both reliability and performance by preventing any single server from becoming overwhelmed with too much traffic.

Local Interpretable Model Agnostic Explanations

LIME explains individual predictions **by approximating the model locally** with an interpretable model and calculating the feature importance based on the surrogate model.

Key Points

- **Purpose:** LIME focuses on explaining individual predictions by approximating the model locally using a simpler, interpretable model (like linear regression).
- **How it Works:**
 - For a given prediction, LIME generates perturbed samples (e.g., by modifying input features).
 - It observes how the predictions change, thus inferring feature importance for that specific instance.
- **Use Cases:** Useful for understanding why a specific decision was made in complex black-box models.
- **Advantage:**
 - LIME can work with any model type.
 - It is relatively easy to apply to tabular, text, and image data.
- **Scenario:**
 - A healthcare provider uses a deep learning model to classify whether patients have a high or low risk of heart disease based on several health metrics, such as cholesterol levels, age, and blood pressure.
 - **LIME Explanation:** LIME is used to explain why the model flagged a specific patient as high-risk. By perturbing the input data (e.g., altering cholesterol levels and re-running the prediction), LIME shows that the patient's high cholesterol level and advanced age are the main reasons for the high-risk classification. This makes it easier for the healthcare provider to justify the decision to recommend lifestyle changes or further medical tests.

Example Code

To use LIME for feature importance, you can use the LIME package:

```
import lime
import lime.lime_tabular

# Create a LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(X_train, feature_names=feature_names)

# Explain a single prediction
explanation = explainer.explain_instance(X_test[0], model.predict_proba)
explanation.show_in_notebook()
```

Logical Model

Logical Model

- Customer: CustomerID, Name, Email
- Order: OrderID, OrderDate, CustomerID
- Book: BookID, Title, Author
- Order-Book Relationship: OrderID, BookID

Logical Model

- Details the attributes of each data entity.
- Specifies relationships without depending on a specific database management system.

Logistic Regression Statsmodel Summary Table

Statsmodel has this summary table unlike [Sklearn](#)

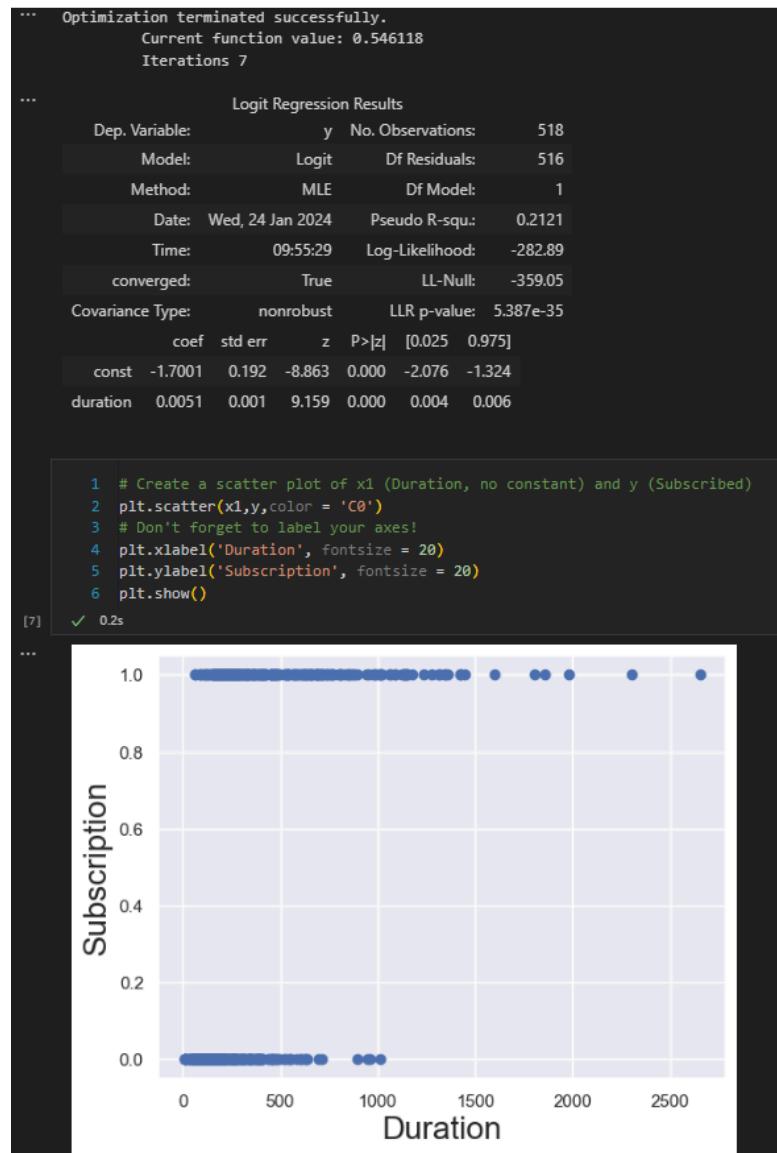
Explanation of summary

The dependent variable is 'duration'. The model used is a Logit regression (logistic in common lingo), while the method

- Maximum Likelihood Estimation ([MLE](#)). It has clearly converged after classifying 518 observations.
- The Pseudo R-squared is 0.21 which is within the 'acceptable region'.
- The duration variable is significant and its coefficient is 0.0051.
- The constant is also significant and equals: -1.70 (p value close to 0)
- High p value, suggests to remove from model, drop one by one, ie [Feature Selection](#).

Introduction

Specifically a graph such as,



N

Logistic Regression Does Not Predict Probabilities

In logistic regression, the model predicts the odds of an event happening rather than directly predicting probabilities. The odds are defined as:

$$\text{Odds} = \frac{P(\text{success})}{P(\text{failure})} = \frac{p}{1-p}$$

where \$p\$ is the probability of success. The log-odds (or logit function) is the natural logarithm of the odds:

Log-Odds = $\ln\left(\frac{p}{1-p}\right) = b_0 + b_1x$ This transformation makes the relationship between the independent variables and the dependent variable linear, allowing logistic regression to estimate the parameters b_0 and b_1 .

Resources:

- [Explanation of log odds](#)
- [Explanation of log odd function](#)

[What is the difference between odds and probability](#)

Logistic Regression

Logistic regression models the log-odds of the probability as a linear function of the input features.

It models the probability of an input belonging to a particular class using a logistic (sigmoid) function.

The model establishes a decision boundary (threshold) in the feature space.

Logistic regression is best suited for cases where the decision boundary is approximately linear in the feature space.

Logistic [Regression](#) can be used for [Binary Classification](#) tasks.

Related Notes:

- [Logistic Regression Statsmodel Summary table](#)
- [Logistic Regression does not predict probabilities](#)
- [Interpreting logistic regression model parameters](#)
- [Model Evaluation](#)
- To get [Model Parameters](#) use [Maximum Likelihood Estimation](#)

In [ML_Tools](#), see:

- [Regression_Logistic_Metrics.ipynb](#)

Key Concepts of Logistic Regression

Logistic Function (Sigmoid Function)

Logistic regression models the probability that an input belongs to a particular class using the logistic (sigmoid) function. This function maps any real-valued input into the range (0,1), representing the probability of belonging to the positive class (usually class 1).

The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

where

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

Thus, the logistic regression model is given by:

$$P(y = 1 \mid \mathbf{x}) = \sigma(z)$$

Log odds: Transforming from continuous to 0-1

Logistic regression is based on the **log-odds** (logit) transformation, which expresses probability in terms of odds:

$$\text{Odds} = \frac{P(y=1|\mathbf{x})}{1-P(y=1|\mathbf{x})}$$

Taking the natural logarithm of both sides gives the logit function:

$$\log\left(\frac{P(y=1|\mathbf{x})}{1-P(y=1|\mathbf{x})}\right) = \mathbf{w} \cdot \mathbf{x} + b$$

This equation shows that **logistic regression models the log-odds of the probability as a linear function of the input features.**

Decision Boundary

- Similar to [Support Vector Machines](#), logistic regression defines a decision boundary that separates the two classes.
- The logistic function determines the probability of a data point belonging to a specific class. If this probability exceeds a given **threshold** (typically 0.5), the model assigns the point to the positive class; otherwise, it is classified as negative.

Binary Classification

- Logistic regression is primarily used for binary classification tasks, where the target variable has only two possible values (e.g., "0" and "1").
- It can handle multiple independent variables (features) and assigns probabilities to the target classes based on the feature values.
- Examples include:

No Residuals

- Unlike [Linear Regression](#), logistic regression does not compute standard residuals.
- Instead, [Model Evaluation](#) is performed by comparing predicted probabilities with actual class labels using metrics such as accuracy, precision, recall, and the [Confusion Matrix](#).

Also see:

Related terms:

- Cost function for logistic regression
- Gradient computation in logistic regression
- Regularized logistic regression
- Cost function for regularized logistic regression

Logistic regression can be extended to handle non-linear decision boundaries through:

- Polynomial features to capture more complex relationships.
- Regularization techniques to improve generalization.

Logistic regression in sklearn & Gradient Descent

sklearn's Logistic Regression implementation does not use [Gradient Descent](#) by default. Instead, it uses more sophisticated optimization techniques depending on the solver specified. These solvers are more efficient and robust for finding the optimal parameters for logistic regression. Here's a summary:

Optimisation function: Solvers in sklearn's Logistic Regression**

1. `lbfgs` (**default in many cases**):
 - o Stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno.
 - o It's a quasi-Newton method, which approximates the second derivatives (Hessian matrix) to find the minimum of the cost function efficiently.
2. `liblinear` :
 - o Uses the coordinate descent method for optimization.
 - o Ideal for small datasets or when `penalty='l1'` is used.
3. `sag` (**Stochastic Average Gradient**):
 - o An iterative solver similar to stochastic gradient descent (SGD) but averages gradients over all samples.
 - o Efficient for large datasets.
4. `saga` :
 - o An improved version of `sag`, supporting both `l1` and `l2` penalties.
 - o Suitable for sparse and large datasets.
5. `newton-cg` :
 - o Uses the Newton method with conjugate gradients.
 - o Efficient for datasets with many features.

Looker Studio

Looker studio is [Google](#) version of [PowerBI](#), but its free.

Connectors to data

Can connect to data sources i.e:

- [standardised/GSheets|GSheets](#)
- [PostgreSQL](#)

Data Modelling

Data models are called Blends

[Dashboard with Relational Database in Looker Studio Data Blending and Modeling](#)

Loss Function

Loss functions are used in training machine learning models. Also known as a [cost function](#), error function, or objective function. Serves as a metric for [model evaluation](#).

Purpose: **Measure predictive accuracy**: Measures the difference between predicted and actual values. That is they measure how well a model's predictions match the actual target values by quantifying the error between the predicted output and the true output.

Goal: **To be minimized**: The primary goal during model training is to minimize this loss, improving accuracy of predictions on unseen data.

Used during training to adjust [model parameters](#) and during evaluation to assess model performance.

Examples

- **Mean Squared Error (MSE)**: Commonly used for [regression](#) tasks.
- **Cross Entropy**: Often used for [classification](#) tasks.

Resources

- [Video Explanation](#)
- [Loss versus Cost function](#)

Loss Versus Cost Function

In machine learning, the terms "loss function" and "cost function" are often used interchangeably, but they can have slightly different meanings depending on the context:

1. **Loss function**: This typically refers to the function that measures the error for a single training example. It quantifies how well or poorly the model is performing on that specific example - data point. Common examples include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks.
2. **Cost Function**: This is generally used to refer to the average of the loss function over the entire training dataset. It provides a measure of how well the model is performing overall.

The cost function is what is minimized during the training process to improve the model's performance, see [Model Optimisation|Optimisation](#). Used with the parameters to determine the best ones.

[Model parameters vs hyperparameters](#)

Lambda Architecture

Lambda architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods. [Data Streaming](#)

This approach to architecture attempts to balance **latency, throughput, and fault tolerance** using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data.

The two view outputs may be joined before the presentation. The rise of lambda architecture is correlated with the growth of big data, real-time analytics, and the drive to mitigate the latencies of [MapReduce](#).

Lambda architecture is a [design pattern](#) for processing large volumes of data by combining both batch and stream processing methods. It aims to provide a comprehensive and efficient way to handle big data by addressing the needs for low-latency data processing, high throughput, and fault tolerance.

Batch Processing

Lambda architecture is particularly useful in scenarios where **both historical** and **real-time data insights** are crucial, such as in financial services, telecommunications, and online retail. It allows organizations to leverage the strengths of both batch and stream processing to meet diverse data processing needs.

Components of Lambda Architecture

1. Batch Layer:

- **Purpose:** To process large sets of historical data in batches.
- **Functionality:** It computes results from all available data, ensuring accuracy and completeness.
- **Tools:** Often uses distributed processing frameworks like Hadoop or Spark for batch processing.
- **Output:** Produces a batch view, which is a complete and accurate dataset that can be queried.

2. Speed Layer:

- **Purpose:** To process real-time data streams with low latency.
- **Functionality:** It provides immediate insights by processing data as it arrives.
- **Tools:** Utilizes stream processing frameworks like Apache Storm, Apache Flink, or Spark Streaming.
- **Output:** Generates a real-time view that reflects the most recent data.

3. Serving Layer:

- **Purpose:** To merge and serve the results from both the batch and speed layers.
- **Functionality:** It combines the batch view and real-time view to provide a unified, queryable dataset.
- **Tools:** Databases or data stores optimized for fast reads, such as Cassandra or HBase, are often used.

How Lambda Architecture Works

- **Data Ingestion:** Data is ingested into both the batch and speed layers simultaneously.
- **Batch Processing:** The batch layer processes data in large volumes, typically with higher latency, to ensure accuracy and completeness.
- **Stream Processing:** The speed layer processes data in real-time, providing low-latency updates.
- **Data Serving:** The serving layer combines outputs from both layers, allowing users to query the most up-to-date and accurate data.

Benefits of Lambda Architecture

- **Fault Tolerance:** By separating batch and real-time processing, the architecture can handle failures more gracefully.
- **Scalability:** It can scale to handle large volumes of data by leveraging distributed processing frameworks.
- **Flexibility:** Supports both historical and real-time data processing, making it suitable for a wide range of applications.

Challenges

- **Complexity:** Maintaining two separate processing paths (batch and speed) can increase system complexity.
- **Data Consistency:** Ensuring consistency between batch and real-time views can be challenging.
- **Maintenance:** Requires more effort to maintain and update due to its dual-layer nature.

Learning Rate

Description

The learning rate is a [Hyperparameter](#) in machine learning that [determines the step size at which a model's parameters are updated during training](#). It plays a significant role in the optimization process, particularly in algorithms like [Gradient Descent](#) which are used to minimize the [Loss function](#).

Key Points about Learning Rate:

1. Parameter Updates:

- o During training, the model's parameters (such as weights and biases in neural networks) are adjusted iteratively to minimize the loss function.
- o The learning rate controls how much the parameters are changed in response to the estimated error each time the model weights are updated.

2. Impact on Training/ Convergence

- o A high learning rate can lead to faster convergence but [risks overshooting](#) the optimal solution, potentially causing the model to diverge.
- o A low learning rate ensures more stable and precise convergence but may result in slow training and can get stuck in local minima. A lower learning rate makes the model more robust but requires more iterations to converge.
- o

3. Tuning:

- o The learning rate is a hyperparameter that needs careful tuning. It can be adjusted manually or through automated hyperparameter optimization techniques like [standardised/Optuna](#).
- o The optimal learning rate depends on various factors, including the dataset, model complexity, and the specific optimization algorithm used.

4. Practical Considerations:

- o It's common to start with a moderate learning rate and adjust based on the model's performance during training.
- o Techniques like learning rate schedules or adaptive learning rate methods (e.g., [Adam Optimizer](#)) can dynamically adjust the learning rate during training to improve convergence.

This impacts the efficiency of [Gradient Descent](#)

Effects occur if too small (takes long), or too large (over shoots missing the minima).

What happens if you are at a local minima? Then no change.

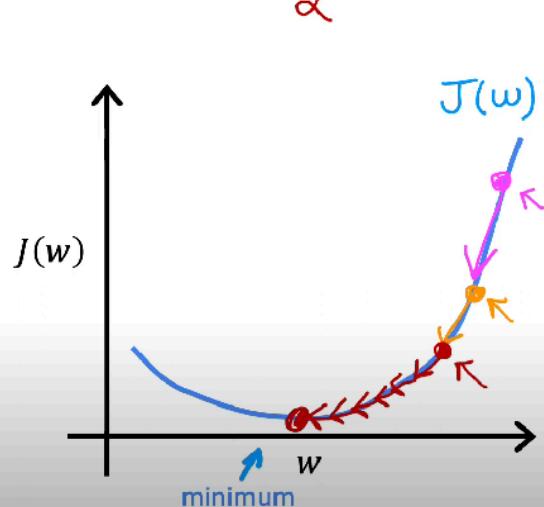
Can reach local minimum with fixed learning rate

$$w = w - \alpha \frac{d}{dw} J(w)$$

smaller
 not as large
 large

- Near a local minimum,
- Derivative becomes smaller
 - Update steps become smaller

Can reach minimum without decreasing learning rate α



Lemmatization

Lemmatization is the process of reducing a word to its base or root form, known as the "lemma."

Unlike stemming, which simply cuts off word endings, lemmatization considers the context and morphological analysis of the words.

It ensures that the root word is a valid word in the language. For example, the words "running," "ran," and "runs" would all be lemmatized to "run."

This process helps in normalizing text data for natural language processing tasks by grouping together different forms of a word.

M

Table of Contents

- ML Engineer
 - MNIST
 - Machine Learning Algorithms
 - Machine Learning Operations
 - Machine Learning
 - Maintainable Code
 - Makefile
 - Manifold Learning
 - Many-to-Many Relationships
 - Markov Decision Processes
 - Markov chain
 - Master Observability Datadog
 - Mathematical Reasoning in Transformers
 - Mathematics
 - Maximum Likelihood Estimation
 - Mean Squared Error
 - Melt
 - Memory Caching
 - Memory
 - Merge
 - Metadata Handling
 - Methods for Handling Outliers
 - Metric
 - Microsoft Access
 - Mini-batch gradient descent
 - Mixture of Experts
 - Model Building
 - Model Cascading
 - Model Deployment
 - Model Ensemble
 - Model Evaluation vs Model Optimisation
 - Model Evaluation
 - Model Interpretability
 - Model Observability
 - Model Optimisation
 - Model Parameters Tuning
 - Model Parameters
 - Model Selection
 - Model Validation
 - Model parameters vs hyperparameters
 - Model preparation
 - Momentum
 - Momentum.py
 - MongoDB
 - Monolith Architecture
-

Introduction

- Monte Carlo Simulation
- Multi-Agent Reinforcement Learning
- Multi-head attention
- Multi-level index
- Multicollinearity
- Multinomial Naive bayes
- MySql
- maintainability
- map reduce
- master data management
- mean absolute error

MI Engineer

ML Engineer

- Configures and optimizes production ML models.
- Monitors the performance and accuracy of ML models in production environments.

Mnist

Datasets

Machine Learning Algorithms

Machine learning [Algorithms](#) are used to automate tasks, extract insights, and make more informed decisions.

Choosing the right algorithm for a specific problem involves understanding the task, the characteristics of the data, and the strengths and limitations of different algorithms.

Supervised Learning

Common [Classification](#) algorithms include:

- Logistic Regression
- Support Vector Machines
- Naive Bayes
- Decision Tree
- Random Forests

Common [Regression](#) algorithms include:

- Linear Regression
- Support Vector Regression
- Random Forest Regression

Unsupervised Learning

Common [Clustering](#) algorithms include:

- K-means
- Gaussian Mixture Models
- Clustering
- Dimensionality Reduction

Common [Dimensionality Reduction](#) algorithms include:

- Principal Component Analysis
- Manifold Learning

Strengths and Limitations of Machine Learning Algorithms

Strengths:

Automation: Machine learning algorithms can automate complex tasks, freeing up human resources for other activities.

Adaptability: Machine learning algorithms can adapt to changing data patterns, making them suitable for dynamic environments.

Scalability: Machine learning algorithms can handle large datasets efficiently, making them applicable to big data problems.

Knowledge Discovery: Machine learning algorithms can help discover hidden patterns and relationships in data, leading to new insights and knowledge.

Limitations:

Data Dependence: The performance of machine learning algorithms heavily depends on the [Data Quality](#) and quantity of the training data.

Overfitting occurs when the model learns the training data too well and fails to generalise to new, unseen data.

Bias and variance: Machine learning algorithms can be biased, reflecting the biases present in the training data.

Interpretability: Some machine learning algorithms, especially deep learning models, can be complex and difficult to interpret, making it challenging to understand the reasoning behind their predictions.

Machine Learning Operations

Machine Learning Operations (MLOps) is a set of practices and tools designed to streamline the entire lifecycle of machine learning models, from development to deployment and maintenance. It aims to integrate machine learning with [DevOps](#) principles to ensure that models are reliable, scalable, and efficient in production environments.

1. Development: MLOps focuses on creating a seamless workflow for developing machine learning models. This includes data preprocessing, feature engineering, model building, and training. The goal is to ensure that models can be developed quickly and efficiently. See [DS & ML Portal](#).

2. Deployment: Once a model is developed and evaluated, MLOps facilitates its deployment into a production environment. This involves setting up the necessary infrastructure to serve the model and ensuring that it can handle real-world data and workloads.
3. Maintenance: MLOps emphasizes the importance of monitoring and maintaining models over time. This includes tracking model performance, detecting data drift, and retraining models as needed to ensure they remain accurate and relevant.
4. Generalization and Robustness: MLOps aims to create models that generalize well to new, unseen data, especially in dynamic environments. It also focuses on ensuring models remain robust to noisy or unexpected data inputs.
5. Collaboration and Automation: MLOps encourages collaboration between data scientists, engineers, and operations teams. It also leverages automation to streamline repetitive tasks, such as model training, evaluation, and deployment.
6. [Model Observability](#) and Retraining: Continuous monitoring of model performance is crucial in MLOps. Observability tools help track metrics and identify when a model needs retraining due to changes in data patterns or performance degradation.

Machine Learning

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning [Machine Learning Algorithms](#) use historical data as input to predict new output values.

Maintainable Code

[Pydantic](#) : routine analysis

[Pyright](#): static analysis

[Testing](#)

Want robust and reliable Python applications.

Makefile

A Makefile is a special file used by the `make` build automation tool to manage the build process of a project. It defines a set of tasks to be executed, typically to compile and link a program. Here are some key functions of a Makefile:

1. **Compilation Instructions:** It specifies how to compile and link the program. This includes defining the source files, the compiler to use, and any necessary flags or options.
2. **Dependencies:** Makefiles list dependencies between files, ensuring that changes in source files trigger recompilation of only the necessary parts of the program.
3. **Automation:** It automates repetitive tasks, such as cleaning up build artifacts, running tests, or deploying software.

- 4. **Targets and Rules:** A Makefile consists of targets, dependencies, and rules. A target is usually a file to be generated, dependencies are files that the target depends on, and rules are the commands to create the target from the dependencies.
- 5. **Variables:** Makefiles can use variables to simplify and manage complex build processes, making it easier to maintain and modify.

Example

```

# Compiler
CXX = g++

# Compiler flags
CXXFLAGS = -Wall -g

# Target executable
TARGET = myprogram

# Source files
SRCS = main.cpp utils.cpp

# Object files
OBJS = $(SRCS:.cpp=.o)

# Default target
all: $(TARGET)

# Rule to build the target executable
$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJS)

# Rule to build object files
%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $< -o $@

# Clean up build artifacts
clean:
    rm -f $(OBJS) $(TARGET)

# Phony targets
.PHONY: all clean

```

Explanation:

- **CXX and CXXFLAGS:** These variables define the compiler and the flags used during compilation.
- **TARGET:** The name of the final executable.
- **SRCS and OBJS:** Lists of source and object files. The `OBJS` variable is automatically generated by replacing `.cpp` with `.o` in the `SRCS` list.
- **all:** The default target that builds the executable.

- **`$(TARGET)`**: This rule specifies how to link object files into the final executable.
- **`%.o: %.cpp`**: A pattern rule to compile each `.cpp` file into a `.o` object file.
- **`clean`**: A target to remove all object files and the executable, useful for cleaning up the build directory.
- **`.PHONY`**: Declares `all` and `clean` as phony targets, meaning they are not actual files but just names for commands to run.

Running it

If you run this Makefile using the `make` command in a terminal, here's what would happen:

1. **Compilation**: The `make` tool will look for a file named `Makefile` in the current directory. It will then execute the default target, which is `all` in this case.
2. **Building the Executable**:
 - `make` will check if the target executable `myprogram` needs to be built. It does this by comparing the timestamps of the source files (`main.cpp`, `utils.cpp`) and the corresponding object files (`main.o`, `utils.o`).
 - If any of the source files are newer than their corresponding object files, or if the object files do not exist, `make` will compile the source files into object files using the rule `%.o: %.cpp`.
 - Once the object files are up-to-date, `make` will link them together to create the `myprogram` executable using the rule for `$(TARGET)`.
3. **Output**: During this process, you'll see the compilation and linking commands being executed in the terminal. If there are any errors in the source code, the compiler will output error messages.
4. **Clean Up**: If you run `make clean`, it will execute the `clean` target, which removes the object files and the executable, cleaning up the build directory.

Manifold Learning

Manifold learning is a powerful approach for high-dimensional data exploration, focusing on uncovering the lower-dimensional manifold that the data resides on. These algorithms aim to identify and map the underlying low-dimensional structure, or **manifold**, that the data is assumed to lie on, within the high-dimensional space. This is particularly useful for reducing dimensionality while preserving the intrinsic properties of the data.

Methods like **Isomap** aim to preserve the geodesic distances between points, which better represent the data's true structure than straight-line distances in high-dimensional space. This enables effective **Dimensionality Reduction** for non-linear data while preserving important relationships between data points.

Key Concepts of Manifold Learning:

1. **High-Dimensional Data**:
 - In many machine learning problems, data can have a high number of dimensions (features), making it challenging to analyze directly. However, often this high-dimensional data lies on a much simpler, lower-dimensional structure, or **manifold**, embedded in the high-dimensional space. Manifold learning seeks to find and represent this lower-dimensional structure, simplifying the analysis and visualization of complex datasets.
2. **Manifold Assumption**:
 - Manifold learning assumes that although the data may appear high-dimensional, the **true degrees of freedom are much fewer**. This means the data can be represented in a lower-dimensional space without losing important information about its structure.

3. Geodesic Distances:

- o In manifold learning, the goal is often to **preserve certain distances or relationships between data points**. **Isomap**, for example, is a popular manifold learning algorithm that aims to preserve **geodesic distances** —the shortest paths between points along the manifold. These distances represent the true relationships between points in the underlying lower-dimensional space, even though they may seem far apart in the high-dimensional space.

4. Dimensionality Reduction

- o Like other dimensionality reduction techniques (such as PCA), manifold learning helps reduce the number of features in the data. However, manifold learning is particularly effective when the data is non-linear, meaning traditional linear techniques like PCA might not capture the true underlying structure. Algorithms like **Isomap**, **Locally Linear Embedding (LLE)**, and **t-SNE** are examples of manifold learning methods that handle such **non-linear structures**.

Example: Isomap

5. **Isomap** is a manifold learning algorithm that tries to preserve the **geodesic distances** between all pairs of data points. It computes these distances by constructing a graph in which the edges represent the shortest path along the manifold (not the straight-line distance in high-dimensional space). Then, it uses these distances to map the data to a lower-dimensional space, retaining the structure of the original data.



Many-to-Many Relationships

Occurs when multiple records in one table are associated with multiple records in another table.

Need to use a **junction table** (also known as a bridge table or associative entity). This table will contain foreign keys that reference the primary keys of the two tables involved in the relationship.

Steps to Implement Many-to-Many Relationships

1. **Identify the Entities:** Determine the two entities that will participate in the many-to-many relationship. For example, consider `students` and `courses`.
2. **Create the Junction Table:** Create a new table that will serve as the junction table. This table will hold the foreign keys from both entities. In our example, we can create a table called `enrollments`.
3. **Define Foreign Keys:** In the junction table, define foreign keys that reference the primary keys of the two related tables. This establishes the relationship between the entities.
4. **Add Additional Attributes (if necessary):** If needed, you can also include additional attributes in the junction table that are relevant to the relationship itself. For instance, you might want to track the enrollment date.

Example Schema

```
-- Create the students table
CREATE TABLE students (
    "id" INTEGER PRIMARY KEY,
    "name" TEXT NOT NULL,
    "email" TEXT UNIQUE NOT NULL
);

-- Create the courses table
CREATE TABLE courses (
    "id" INTEGER PRIMARY KEY,
    "title" TEXT NOT NULL,
    "description" TEXT
);

-- Create the junction table for the many-to-many relationship
CREATE TABLE enrollments (
    "student_id" INTEGER,
    "course_id" INTEGER,
    "enrollment_date" DATE NOT NULL,
    PRIMARY KEY("student_id", "course_id"), -- Composite primary key
    FOREIGN KEY("student_id") REFERENCES "students"("id"),
    FOREIGN KEY("course_id") REFERENCES "courses"("id")
);
```

A composite primary key (`student_id`, `course_id`) ensures that each student can enroll in a course only once.

To retrieve data from a many-to-many relationship, you can use SQL JOIN statements. For example, to find all courses a specific student is enrolled in, you can run:

```
SELECT courses.title
FROM courses
JOIN enrollments ON courses.id = enrollments.course_id
WHERE enrollments.student_id = 1; -- Replace 1 with the desired student ID
```

Many-to-Many Relationships

- Records in Table A relate to multiple records in Table B, and vice versa.
- Requires a junction table to manage the relationships.
- Example: Students and Courses tables with a junction table Enrollments.

Markov Decision Processes

Markov Decision Process ([Markov Decision Processes|MDP](#)) is a formal framework for decision-making where outcomes depend solely on the current state (Markov property).

architecture

Markov Decision Processes ([Markov Decision Processes|MDPs](#)): The mathematical framework for modelling decision-making, characterized by states, actions, transition probabilities, and rewards. Your understanding of probability theory and stochastic processes will be crucial here.

Markov Chain

Is a stochastic model that describes a sequence of events in which the probability of each event depends only on the state attained in the previous event.

Master Observability Datadog

what happens in prod, pre prod.

monitoring web frontend.

how is infrastructure working in prod

Datadog

agents

tagging

profile how it was working versus other dates.

dashboards

Lambdas

logging

Mathematical Reasoning In Transformers

transformer-based models that address mathematical reasoning either through pretraining, hybrid systems, or fine-tuning on specific mathematical tasks

- **Challenges:** General-purpose transformers [Transformer](#) are trained primarily on large corpora of text, which include mathematical problems but lack systematic and rigorous math-specific training. This results in limited capabilities for handling complex calculations or abstract algebraic problems.
- **Grokking in Mathematical Reasoning:** This is an area of research where models are trained on small datasets of synthetic math problems to encourage **grokking**, a phenomenon where the model suddenly achieves near-perfect performance after extended training. Researchers are interested in how transformers might be able to "**grok**" math concepts after seeing many examples.

math data sets: MATH dataset, Aristo

Pretrained transformers on math specific data.

[GPT-f](#) represents a significant advancement in the use of **transformer-based models for mathematical reasoning**,

Mathematics

[Johnson–Lindenstrauss lemma](#)

[Big O Notation](#)

[Directed Acyclic Graph \(DAG\)](#)

Maximum Likelihood Estimation

Resource:

- <https://www.youtube.com/watch?v=YevSE6bRhTo>

Used to infer **Model Parameters** from collected data for example in **Linear Regression** (β_0, β_1).

Definition: Likelihood

Why is it a good tool for guessing parameter values?

The likelihoods plot a distribution, the max gives the most likely.

This is called the MLE.

Properties of a MLE:

- As more data comes in the **Estimator** should approach a true value
- MLE is a **consistent Estimator**, i.e it gets closer to the true parameter value as the sample size grows.
- Asymptotical Normal
- Asymptotic Efficiency

Assumptions for MLE:

- Regularity

parametric vs non-parametric models

Likelihood is a function of a parameter

Mean Squared Error

Measures numerical proximity.

Melt

In pandas, the `melt` function is used to **transform (Data Transformation)** a DataFrame from a wide format to a long format. This is especially useful for data analysis and visualization tasks where long-format data is preferred or required. The wide format typically has multiple columns for different variables, whereas the long format has a single column for variable names and a single column for values.

Related:

- Database Techniques
- Turning a flat file into a database

In **DE_Tools** see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/reshaping.ipynb
-

Key Reasons to Use `melt` :

- [Normalisation](#)
 - Wide to Long Transformation: `melt` helps in converting data with many columns (wide format) into a more normalized form with fewer columns (long format). This is useful for many statistical and visualization libraries that prefer long-format data.
- Easier Analysis and [Data Visualisation](#):
 - Compatibility with Plotting Libraries: Many plotting libraries like `seaborn` and `ggplot` require data in a long format for creating certain types of plots, such as [Grouped plots](#).
- Simplifying Complex Data Structures:
 - Handling [Multi-level index](#): If a DataFrame has multiple levels of columns, `melt` can help flatten this structure, making it easier to work with.
- Preparation for Aggregation:
 - Facilitating [Groupby](#) Operations: Long-format data is often more suitable for these.

Parameters of `melt` :

- `id_vars` : Columns to use as identifier variables. These columns are kept as-is in the output.
- `value_vars` : Columns to unpivot. These columns are transformed into a single column.
- `var_name` : Name to use for the `variable` column in the output.
- `value_name` : Name to use for the `value` column in the output.

Example Usage:

Consider a DataFrame in wide format:

```
import pandas as pd

# Sample wide format data
data = {
    'id': [1, 2, 3],
    'math_score': [88, 92, 95],
    'science_score': [85, 90, 89],
    'english_score': [78, 85, 88]
}
df_wide = pd.DataFrame(data)
print(df_wide)
```

Output:

	<code>id</code>	<code>math_score</code>	<code>science_score</code>	<code>english_score</code>
0	1	88	85	78
1	2	92	90	85
2	3	95	89	88

To convert this wide-format DataFrame into a long-format DataFrame using `melt` :

Introduction

```
# Melt the DataFrame
df_long = pd.melt(df_wide, id_vars=['id'],
                   value_vars=['math_score', 'science_score', 'english_score'],
                   var_name='subject', value_name='score')
print(df_long)
```

Output:

```
    id      subject  score
0   1      math_score    88
1   2      math_score    92
2   3      math_score    95
3   1  science_score    85
4   2  science_score    90
5   3  science_score    89
6   1  english_score    78
7   2  english_score    85
8   3  english_score    88
```

Memory Caching

Memory Caching

- Use in-memory caches to store frequently accessed data closer to the user, reducing latency.

Memory

Memory in large [language models](#) (LLMs) involves managing context windows to enhance reasoning capabilities without the high costs associated with traditional training methods. The goal of [Memory](#) is to address challenges like "forgetting," where LLMs struggle to retain context across interactions.

Key Concepts:

Forgetting Context:

Understanding how and why LLMs lose context, especially in multi-turn dialogues, and its impact on response accuracy. Forgetting occurs due to the limitations of fixed **context windows**, manifesting differently in single-turn (immediate forgetting) versus multi-turn interactions (progressive loss of context).

Prioritization of Context: Techniques for determining which parts of the context are most relevant and need to be retained, optimizing memory usage.

Time Length of Memory: Balancing how long memory should be maintained to ensure it remains useful and relevant over time.

Dynamic Memory Management: Adapting memory structures in real-time to accommodate evolving knowledge and interactions.

In-Context Memory: Memory tied to specific interactions, making it more relevant and easier to apply in particular scenarios.

Multi-turn Interactions: Addressing context retention across multiple interactions, emphasizing the importance of maintaining coherence over extended conversations.

Types of Memory:

Semantic Memory: Focuses on the meaning and [Semantic Relationships](#) between concepts, which is crucial for improving LLM reasoning and context understanding.

Hierarchical Memory: Balances immediate retrieval with long-term storage of information, enabling better performance in various applications.

Supports evolving and persistent memory systems tailored to specific tasks.

Merge

Metadata Handling

Trimming

- Description: Removing data points identified as outliers based on criteria such as being beyond a certain number of standard deviations from the mean or outside a specified percentile range.
- Implementation Example:

```
lower_quantile = df["var1"].quantile(0.01)
upper_quantile = df["var1"].quantile(0.99)
df_no_outliers = df[(df["var1"] >= lower_quantile) & (df["var1"] <= upper_quantile)]
```

Capping or Flooring

- Description: Setting a maximum or minimum threshold beyond which data points are considered outliers and replacing them with the threshold value.

Winsorizing

- Description: Similar to capping and flooring, winsorizing replaces extreme values with less extreme values within a specified range, typically using percentiles.

Metric

Metrics in Machine Learning

[Evaluation Metrics](#) [Regression Metrics](#)

Metrics in business

A metric, also called [KPI.md](#) or (calculated) measure, are terms that serve as the building blocks for how business performance is both measured and defined, as knowledge of how to define an organization's KPIs. It is fundamental to have a common understanding of them. Metrics usually surface as business reports and dashboards with direct access to the entire organization.

For example, think of [operational metrics](#) that represent your company's performance and service level or financial metrics that describe its financial health.

Calculated measures are part of metrics and apply to specific [Dimensions](#) traditionally mapped inside a [Bus Matrix](#).

Microsoft Access

Tasks

- [] How to update (or more so insert) into multiple related tables. How to insert existing data into a [Database](#).
SQL triggers?
- [] Investigate: Helper table to gather many small tables into one.
- [] What are typical types of databases.
- [] Questions: Make a form that accesses multiple tables.

Resources

[Tutorial](#)

[Best Practices](#)

[LINK](#)

[TIME](#)

Notes

Why use access: Handles lots of data better than excel. Understand relationships between sources of data.

Querying: Can do querying. Which might be hard to do in excel. Graphical way to make queries.

Forms: Access can make it easier for user interfaces forms- opening other forms, drop downs, user interface secure fields on forms so users can only see so much

Features: Has user security. Has control over the types of data input. User control feature user friendly.

Issues: Possible limitations when scale increases. Next steps, can upscale to SQL server.

Mini Batch Gradient Descent

Mixture Of Experts

Different parts of the network focusing on parts of the questions

Routing, distribution

Model Building

The Model Building phase follows the [Preprocessing](#) phase, where data is organized and prepared for analysis. This phase focuses on selecting and setting up the appropriate machine learning models to solve the problem at hand.

Key Steps

Types of Models:

- Choose a model to apply based on the problem requirements and data characteristics.
- Explore different [Machine Learning Algorithms](#) to find the best fit for your data.
- Consider the tradeoffs between [parametric vs nonparametric models](#).

Setting Up a Model:

- Divide the data into [Train-Dev-Test Sets](#) to ensure robust evaluation and tuning.
- Optimize [Model Parameters](#) and configurations for best performance.

Model Selection:

- Evaluate the appropriateness of models in the [Model Selection](#) phase.

Model Cascading

Model Deployment

Deploying a machine learning model involves moving it from a development environment to a production environment where it can make predictions on new data.

Steps for Model Deployment

Model Exporting

- Use tools like `joblib` or `pickle` to serialize the model.

```
import joblib
joblib.dump(model, 'linear_regression_model.pkl')
```

Deployment Options

- Application Integration: Embed the model into an application for real-time predictions.
- API Deployment: Use frameworks like [Flask](#) or [FastAPI](#) to create an API endpoint for the model.
- Automated Workflows: Integrate the model into automated data processing pipelines.

Tools and Platforms

- **Sklearn Pipeline:** Streamline the deployment process by integrating [Preprocessing](#) and model steps.
- **Gradio:** Create user-friendly interfaces for model interaction.
- **Streamlit.io**

Considerations

- **Scalability:** Ensure the deployment solution can handle the expected load.
- **Model Observability:** Implement monitoring to track model performance and detect issues.

Deploying using PyCaret

AWS: When deploying model on AWS S3, environment variables must be configured using the command-line interface. To configure AWS environment variables, type `aws configure` in terminal. The following information is required which can be generated using the Identity and Access Management (IAM) portal of your amazon console account:

- AWS Access Key ID
- AWS Secret Key Access
- Default Region Name (can be seen under Global settings on your AWS console)
- Default output format (must be left blank)

GCP: To deploy a model on Google Cloud Platform ('gcp'), the project must be created using the command-line or GCP console. Once the project is created, you must create a service account and download the service account key as a JSON file to set environment variables in your local environment. Learn more about it:

<https://cloud.google.com/docs/authentication/production>

Azure: To deploy a model on Microsoft Azure ('azure'), environment variables for the connection string must be set in your local environment. Go to settings of storage account on Azure portal to access the connection string required. AZURE_STORAGE_CONNECTION_STRING (required as environment variable) Learn more about it:
[https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python?
toc=%2Fpython%2Fazure%2FTOC.json](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python?toc=%2Fpython%2Fazure%2FTOC.json)

Model Ensemble

Ensemble models in machine learning are techniques that **combine the predictions of multiple individual models** to improve overall performance. Ensemble methods can achieve better accuracy and robustness than any single model alone.

Key Concepts of Ensemble Models:

1. **Diversity:** The strength of ensemble models lies in the **diversity** of the base models. Different models may capture different patterns or errors in the data, and combining them can lead to more accurate predictions.
2. **Combination:** Ensemble methods aggregate the predictions of individual models using **techniques like averaging, voting, or weighted sums** to produce a final prediction.

Main Ensemble Techniques:

- **Bagging**
- **Boosting**

- Stacking
 - Isolated Forest
-

In [ML_Tools](#) see: [Comparing_Ensembles.py](#)

Further Understanding

Analogy:

- Ensemble methods can be likened to consulting multiple doctors for a diagnosis. Each doctor (model) may have a different opinion, but by considering all opinions, the final diagnosis (prediction) is more accurate than relying on a single doctor's opinion.

Advantages of Ensemble Models:

- **Increased Accuracy:** By combining multiple models, ensemble methods often achieve higher accuracy than individual models.
- **Robustness:** They are less sensitive to overfitting, especially when using techniques like bagging.
- **Flexibility:** Ensemble methods can be applied to various types of base models and are not limited to a specific algorithm.

Challenges:

- **Complexity:** Ensemble models can be more complex and computationally intensive than single models.
- **Interpretability:** The final model may be harder to interpret compared to simpler models like decision trees.

Model Evaluation Vs Model Optimisation

[Model Evaluation](#) focuses on assessing a model's performance, while [Model Optimisation](#) aims to improve that performance through various techniques.

Iterative Process: Model evaluation and optimization are often iterative. After evaluating a model, insights gained can guide further optimization. Conversely, after optimizing a model, it needs to be re-evaluated to ensure improvements.

Feedback Loop: Evaluation provides feedback on the effectiveness of optimization efforts, helping refine the model further.

Model Evaluation

Assess the model's performance using various metrics to ensure it meets the desired accuracy and reliability.

Appropriate evaluation metrics are used based on the problem type (classification vs. regression), to assess how well the model predicts.

The aim is to improve accuracy but also to generalize and avoid biases and [Overfitting](#).

- **Performance Assessment:** Models are evaluated on a testing set using metrics relevant to the problem type.
 - **Generalization and Bias:** Evaluation includes assessing how well the model generalizes to new data and identifying any biases.
-

Introduction

For categorical classifiers: **Evaluation Metrics**: Use metrics such as accuracy, precision, recall, F1-score, and confusion matrix to evaluate performance.

For regression tasks: **Regression Metrics**: Metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) are used.

Cross Validation is a technique used to assess the performance of a model by splitting the data into multiple subsets for training and testing to assesses performance and generalization. It helps detect **Overfitting**, provides reliable performance estimates.

Feature Importance: After training, analyze which features have the most significant impact on the model's predictions.

Model Interpretability

Model **interpretability** tools are crucial in ensuring that machine learning models are transparent, explainable, and understandable to stakeholders, particularly in industries where decisions need to be justifiable (e.g., finance, healthcare).

These tools are becoming standard for ensuring trustworthiness and transparency in ML models, enabling organizations to defend model predictions in regulated industries and maintain user trust.

p values and **Confidence Interval**: If statistical significance is needed, interpret these values to determine which features significantly contribute to the model.

SHapley Additive exPlanations

Local Interpretable Model-agnostic Explanations

Counterfactual Explanations:

- Purpose: Counterfactual explanations aim to provide insight into **how small changes in the input features could lead to different outcomes**, helping users understand model behavior.
- How it works: It identifies the minimal changes needed to alter a prediction. For example, in a credit scoring model, it might show how an individual could change their features (e.g., increasing income) to get approved for a loan.
- Use cases: Particularly useful in sensitive fields like credit scoring, hiring, and medical diagnosis, where actionable explanations are critical.
- Advantage: Provides intuitive and actionable feedback on predictions.

Global Surrogate Models

- Purpose: A global surrogate is an interpretable model that is trained to approximate the predictions of a black-box model.
- How it works: It uses simpler models (like decision trees) to mimic the behavior of a complex model and provide a global, easy-to-understand representation of how the model makes decisions.
- Use cases: Provides **insight into overall model behavior**, though not as accurate as local explanations for specific predictions. -
- Advantage: Simplicity and clarity for non-technical stakeholders.
- Scenario: An e-commerce company uses a neural network to predict customer churn based on features like purchase history, browsing behavior, and customer support interactions.

- Surrogate Model: To explain the overall decision-making process of the complex neural network, the data science team trains a decision tree as a global surrogate model. This decision tree offers a simplified view, showing that customers with a decline in recent purchases and frequent negative support interactions are most likely to churn.

Model Observability

Monitor the model's performance over time (in production). Similar to [Model Validation](#).

In the context of machine learning (ML), Observability refers to the ability to [monitor, understand, and diagnose the performance and behaviour of ML models](#) in production.

It encompasses the processes, tools, and techniques that help practitioners ensure models are functioning as expected and identify when they deviate from desired outcomes.

[Master Observability Datadog](#)

Key aspects of observability in machine learning include:

Observability is a process in ML, and is usually achieved through logging, metrics collection, real-time monitoring, and advanced diagnostic tools integrated into the ML pipeline.

1. Monitoring Model Performance ([monitoring metrics](#)):

- Tracking key metrics such as [Accuracy](#), [Precision](#), [Recall](#), [F1 Score](#), [ROC \(Receiver Operating Characteristic\)](#) and other relevant KPIs over time to identify performance degradation or improvements.
- Monitoring [Performance Drift](#) in model inputs (features) and outputs (predictions) to detect when the model no longer performs well due to changes in data distribution ([data drift](#)) or changes in relationships between variables ([Performance Drift](#)).

2. Error and [Isolated Forest](#):

- Identifying when predictions are out of the expected range or when the model behaves abnormally, such as high error rates on specific subsets of data or excessive latency in prediction generation.

3. [Interpretability](#):

- Ensuring that the internal workings of the model (e.g., feature importance, decision pathways) are visible, interpretable, and explainable to humans. This allows for easier debugging and accountability, especially in critical applications such as finance, healthcare, or autonomous systems.

4. [data lineage](#) and Provenance:

- Tracking the data sources, transformations, and processes that influence the model's input data. This provides visibility into how data flows through the pipeline and helps in reproducing results or addressing data-related issues.

5. Pipeline Monitoring:

- Observing the entire ML pipeline from data ingestion and preprocessing to model training, [validation](#), and deployment. This includes identifying bottlenecks, delays, and system failures that may affect the model's ability to make predictions in real-time.

6. Alerts and Automation:

- Setting up [automated alerts](#) when certain thresholds are breached, such as a sudden drop in accuracy or an increase in response time. This allows for prompt interventions, whether retraining the model, adjusting the pipeline, or tuning hyperparameters.

Why Observability Matters in Machine Learning:

- Ensures Reliability: Observability provides insights into how models behave under different conditions, ensuring that they remain reliable and consistent in their performance.
- Prevents Model Drift ([Performance Drift](#)): With observability, teams can detect model drift early, enabling them to retrain or recalibrate the model before performance deteriorates.
- Improves Accountability: Particularly in high-stakes applications, having observability in place allows organizations to understand and justify the model's decisions.
- Supports Continuous Monitoring: Observability is critical in ML systems that operate continuously in production, ensuring they are making accurate and meaningful predictions over time.

Monitor the model's performance over time. If the data distribution changes (concept drift), or the model's accuracy declines, retraining or updating the model may be necessary.

Related to:

- [Data Observability](#)
- [Model Validation](#)

Model Optimisation

Model optimization is a step in the machine learning workflow aimed at enhancing a model's performance by fine-tuning its parameters and hyperparameters. The goal is to improve the model's accuracy, efficiency, and ability to generalize to new data.

Purpose:

- Accuracy: Improve the model's predictive performance.
- Efficiency: Ensure the model runs efficiently in terms of computation and resource usage.
- Generalization: Enhance the model's ability to perform well on unseen data, avoiding overfitting.

Process:

1. [Model Parameters tuning](#)
2. [Hyperparameter|Hyperparameter tuning](#)
 - Adjust hyperparameters such as learning rate, number of layers in a neural network, and regularization strength to find the optimal configuration.
 - Techniques like grid search, random search, or Bayesian optimization can be used for this purpose.
3. [Feature Engineering](#)
 - Involves selecting, transforming, or creating new features that can improve model performance.
 - This step can significantly impact the model's ability to learn patterns from the data.
4. [Model Evaluation](#)
 - Evaluate the model using appropriate metrics based on the problem type (e.g., classification or regression).
 - Metrics for classification include accuracy, precision, recall, F1-score, and confusion matrix.
 - Metrics for regression include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2).

5. Cross Validation

- A technique to assess the model's performance by splitting the data into multiple subsets for training and testing.
- Helps in detecting overfitting and provides reliable performance estimates.

6. Model Ensemble: Combining models to get better performance

Model Parameters Tuning

To find optimal [Model Parameters](#).

Finding Optimal Model Parameters

1. Parameter Space Exploration:

- It's useful to visualize slices of the parameter space by selecting two parameters at a time. This helps in understanding how different parameter values affect the model's performance.

2. Cost Function

- The cost function is used to find the minimum error in predictions. It measures the difference between predicted and actual values, and the goal is to minimize this function to improve model accuracy.

3. Optimisation function

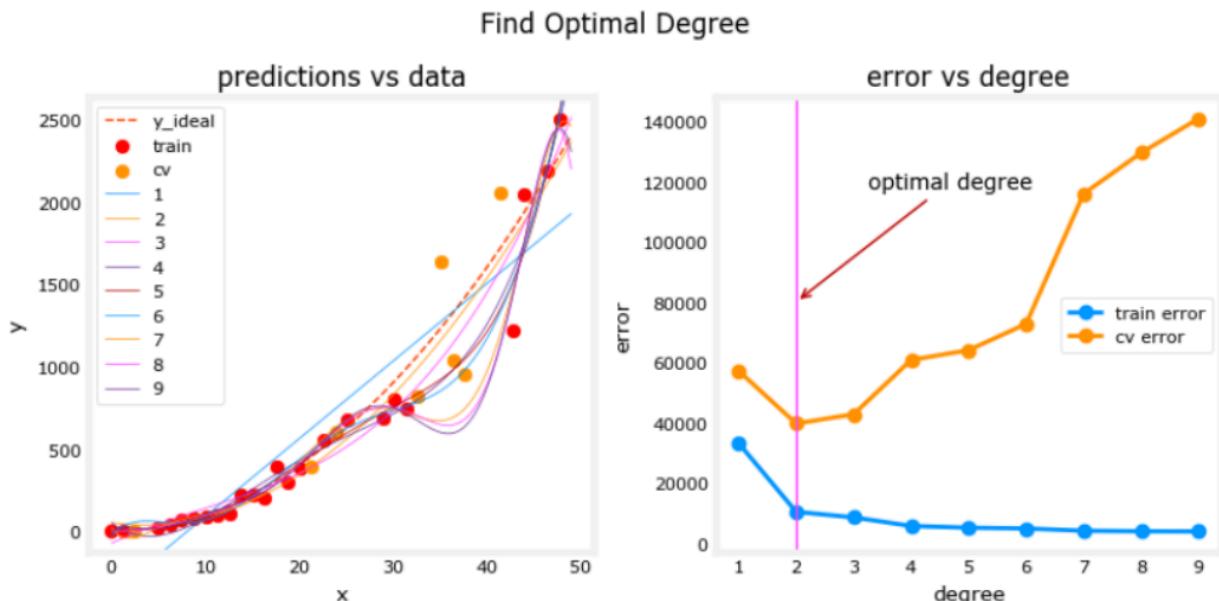
- Ideal parameters are found using optimization functions, which adjust the model parameters to minimize the loss function. Common optimization algorithms include Gradient Descent, Adam Optimizer, and Stochastic Gradient Descent.

4. Data Splitting:

- Split the data into training and cross-validation sets to evaluate model performance. Plot the parameter of interest on the x-axis and accuracy on the y-axis to visualize performance.

[Optimisation techniques](#)

Example

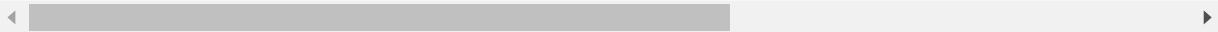


To find optimal model parameters, graph the parameter against error of the model.

On the left plot, the solid lines represent the predictions from these models. A polynomial model with degree 1 produces a straight line that intersects very few data points, while the maximum degree hews very closely to every data point.

On the right:

- the error on the trained data (blue) decreases as the model complexity increases as expected
- the error of the cross-validation data decreases initially as the model starts to conform to the data, but then increase



Model Parameters

Model parameters are also called weights and biases.

These parameters are adjusted during the training process to optimize the model's performance on the given task.

See also:

- [Model Parameters Tuning](#)
- [Optimisation techniques](#)

Examples

- [Linear Regression](#):
 - Coefficients (weights) for each feature in the input data.
 - Intercept term (bias).
- [Logistic Regression](#):
 - Similar to [linear regression](#), it has coefficients for each feature and an intercept term, but it models the probability of a binary outcome.
- [Deep Learning|Neural Networks](#):
 - Weights: The connections between neurons in different layers.
 - Biases: Additional parameters added to the weighted sum of inputs to a neuron.
- [Support Vector Machines \(SVM\)](#):
 - Support vectors: Data points that define the decision boundary.
 - Coefficients for the hyperplane equation.
- [Decision Trees](#): [Decision Tree](#)
 - Splitting thresholds for each node.
 - Structure of the tree (which features are used at each split).
- [K-Means Clustering](#):
 - Centroids: The center points of each cluster.

Model Selection

Model selection is an integral part of building a [Machine Learning Operations](#) to ensure that the best performing model is chosen for a given task, avoiding issues like overfitting or underfitting.

This is a crucial step because the model's ability to **generalize** to unseen data depends on selecting the right one.

Model selection typically involves the following steps:

1. Define candidate models: These can be models of different types (e.g., decision trees, support vector machines, neural networks) or the same model type but with varying hyperparameters.
2. Train each model: Train all the candidate models on the training set using different algorithms or parameter settings.
3. Evaluate performance ([Model Evaluation](#)): Use a validation set or cross-validation to evaluate the performance of each model. Common evaluation metrics include accuracy, precision, recall, F1 score, and mean squared error, depending on the type of problem (classification or regression).
4. Select the best model: Based on the evaluation metrics, choose the model that performs best on the validation set. The aim is to balance bias and variance to achieve good generalization to unseen data.
5. Test on unseen data: Finally, test the selected model on a test set to ensure that it generalizes well and has not been overfitted to the validation data.

Common approaches for model selection include:

- [GridSearchCV](#) and [Random Search](#) for hyperparameter tuning.
- [Cross Validation](#) to ensure robustness by evaluating model performance on different subsets of the data.
- Bayesian Optimization, which can be used to efficiently search the hyperparameter space.
- Choose the best-performing model based on [Evaluation Metrics](#) and optimization results.
- [Cross Validation](#): Evaluate the model more robustly by splitting the training data into smaller chunks and training the model multiple times.
- [Model Interpretability](#): Utilize tools to understand and interpret the model's predictions, ensuring transparency and trustworthiness.

Model Validation

Model Validation refers to the process of evaluating a machine learning model's performance on a separate dataset (often called the validation set) to ensure it generalizes well to new, unseen data. This step is crucial for tuning [model parameters](#), selecting the best model, and preventing overfitting. Validation helps in assessing how well the model will perform in real-world scenarios.

[Model Observability](#), on the other hand, involves monitoring and understanding the model's performance and behavior in production over time. It includes tracking metrics, detecting [Performance Drift](#), and ensuring the model continues to function as expected in dynamic environments.

While model validation is a step in the model development process, model observability is an ongoing practice once the model is deployed. Both are related in that they aim to ensure the model's reliability and effectiveness, but they occur at different stages of the model lifecycle. Validation is about initial performance assessment, whereas observability is about continuous monitoring and maintenance.

Model Parameters Vs Hyperparameters

Model parameters and hyperparameters serve different roles:

[Model Parameters](#)

Introduction

- These are the internal variables of the model that are learned from the training data. They define the model's structure and are adjusted during the training process to minimize the [Loss function](#).
- Examples include:
 - the weights and biases in a neural network,
 - the coefficients in a linear regression model,
 - or the support vectors in a support vector machine.
- Model parameters are directly influenced by the data and are optimized through algorithms like [Gradient Descent](#).

Hyperparameter

- These are external configurations set before the training process begins. They are not learned from the data but are used for controlling the learning process and the model's architecture.
- Examples include the:
 - [learning rate](#),
 - the number of hidden layers in a [Neural network](#),
 - the number of trees in a random forest,
 - or the regularization parameter in a regression model.
- Hyperparameters are typically tuned through methods like grid search or random search to find the best configuration that results in optimal model performance.

Model Preparation

```
model =  
model.fit(X_train, y_train)  
print(model)  
y_pred = model.predict(X_test)  
print(accuracy_score(y_expect, y_pred))
```

Momentum

Momentum is an [Model Optimisation|Optimisation](#) technique used to accelerate the [Gradient Descent](#) algorithm by incorporating the concept of inertia. It helps in reducing oscillations and speeding up convergence, especially in scenarios where the [cost function](#) has a complex landscape (surface). Momentum helps in dampening oscillations and achieving faster convergence. Momentum is a technique that helps accelerate gradient descent by adding a fraction of the previous update to the current update. Formula:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla_{\theta} J(\theta)$$

$$\theta_{t+1} = \theta_t - \alpha v_{t+1}$$

Where:

- v_t is the velocity (the accumulated gradient).
- β is the momentum factor.
- $\nabla_{\theta} J(\theta)$ is the gradient of the cost function with respect to the parameters θ .
- α is the learning rate.

In [ML_Tools](#) see: [Momentum.py](#)

Key Features of Momentum

Inertia Effect: Momentum uses the past gradients to smooth out the updates, which helps in navigating the parameter space more effectively.

Parameter Update Rule: The update rule for momentum involves maintaining a velocity vector that accumulates the gradients. The parameters are then updated using this velocity, which is a combination of the current gradient and the previous velocity.

Hyperparameter

- **Learning Rate (α):** Controls the size of the steps taken towards the minimum.
- **Momentum Coefficient (β):** Determines the contribution of the previous gradients to the current update. A typical value is 0.9.

Momentum.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Optimisation/Momentum.py

Mongodb

Monolith Architecture

A monolith, in the context of [software architecture](#), refers to a **single, unified application where all components and functionalities are interconnected and interdependent**. In a monolithic architecture, the entire application is typically built as a **single codebase**, and all functions and modules are tightly coupled.

While monolithic architectures can be simpler to develop and deploy initially, they can become cumbersome as the application grows in complexity. Many organizations eventually transition to [microservices](#) or other modular architectures to improve scalability, flexibility, and maintainability. However, monoliths can still be effective for smaller applications or teams with limited resources.

When we talk about a "function call-driven" monolith, we are referring to the way in which different parts of the application interact with each other. In such a system:

1. **Tightly Coupled Components:** All components of the application are part of a single codebase and often share the same resources, such as databases and libraries.
2. **Function Calls:** Communication between different parts of the application is primarily done through direct function or method calls. This means that one part of the application can directly invoke functions or methods in another part.
3. **Single Deployment Unit:** The entire application is deployed as a single unit. Any changes to one part of the application require redeploying the whole application.
4. **Shared Memory Space:** Since all components are part of the same application, they often share the same memory space, which can simplify data sharing but also lead to issues with scalability and fault isolation.
5. **Challenges with Scalability and Flexibility:** As the application grows, a monolithic architecture can become difficult to manage, scale, and update. Changes in one part of the system can have unintended consequences elsewhere, making it challenging to innovate quickly.

Monte Carlo Simulation

Resources:

- <https://www.youtube.com/watch?v=r7cn3WS5x9c>

Algorithms that use repeated random sampling.

Monte Carlo: random

How does the randomness in data generation impact the randomness of the parameter calculation.

Simulation study: 1) FIGURE OUT A WAY TO APPROXIMATE A PROCESS WITH A RANDOM NUMBER GENERATOR 2) GENERATE THE DATA AND CALCULATE A VALUE OF INTEREST(I.E.MEAN, BIAS, COVERAGE) 3) REPEAT STEPS 1& 2 MANY TIMES TO LEARN ABOUT THE UNCERTAINTY IN THIS VALUE

Simulation studies:

Multi Agent Reinforcement Learning

Multi Head Attention

Summary

Aggregates different perspectives

This approach allows the model to attend to different parts of the input sequence simultaneously, capturing various aspects of the context more effectively.

Like a hydra, it focuses on different aspects of the context. Getting a finer understanding.

Multi-head attention captures more context by dividing the input processing into multiple independent attention heads. Each head focuses on different parts of the input and captures diverse types of relationships, both local and global. This parallelism allows the model to learn multiple perspectives simultaneously, enriching its understanding of the input sequence and improving performance on complex tasks like language modeling, machine translation, and more.

Related to the [Attention mechanism](#).

Multi-Head Attention

In **multi-head attention**, the idea is to split the input into multiple subsets of attention heads. Instead of computing a single attention score for each token pair, multiple attention "heads" are used, with each head attending to different parts of the input. This provides several benefits:

a) Diverse Attention Patterns

Each head in multi-head attention can focus on different aspects of the input sequence, allowing the model to capture multiple relationships between tokens. For instance:

- One head may focus on syntactic relationships (like word order or structure).
- Another head may focus on semantic relationships (like the meaning or context of words).

By having multiple attention heads, the model can learn to capture **different types of context simultaneously**.

b) Different Projection Spaces

Each attention head has its own set of parameters, which project the input into a different subspace (i.e., they use different weight matrices to transform queries, keys, and values). This allows each head to learn different relationships between tokens in various representational subspaces, thus increasing the model's capacity to understand complex dependencies.

For example, one head might focus on short-range dependencies (like adjacent words), while another head could capture long-range dependencies (like relationships between words at opposite ends of the sentence).

c) Improved Expressiveness

By combining the outputs of multiple heads, the model gains a richer representation of the context. Each attention head contributes unique insights about how tokens in the sequence relate to each other, and by concatenating these different perspectives, the overall attention mechanism becomes more expressive.

Multi-Head Attention Process

The process for multi-head attention involves the following steps:

1. **Linear Transformations:** The input vectors (representing words or tokens) are linearly transformed into **queries (Q)**, **keys (K)**, and **values (V)** for each head. These transformations are different for each head, allowing each head to capture different relationships in the data.
2. **Attention Calculation for Each Head:** For each head, scaled dot-product attention is calculated independently. The attention mechanism computes scores by comparing queries and keys, and these scores are used to weigh the values. This results in a unique context vector for each head.
3. **Concatenation:** The outputs from all the attention heads are concatenated into a single vector. This step combines the different perspectives learned by each head.
4. **Final Linear Transformation:** After concatenation, a final linear transformation is applied to combine the information from all heads into a single vector that can be used in the next layer of the model.

How Multi-Head Attention Captures More Context

Multi-head attention captures more context than single-head attention for several reasons:

- **Multiple Focus Areas:** By using multiple heads, the model can simultaneously focus on different parts of the sequence. Some heads might **attend to local dependencies**, while others might capture more distant relationships. This gives the model a **broader understanding** of the entire sequence.
- **Handling Ambiguity:** In natural language, the meaning of words can depend heavily on context. Multi-head attention allows the model to disambiguate meanings by attending to different context clues in parallel. For instance, the word "bank" in "I went to the bank" can mean different things, and different heads can **capture clues** from the surrounding words to determine whether "bank" refers to a financial institution or a riverbank.
- **Diverse Representations:** Each head transforms the input into **different representational subspaces**, meaning the model learns diverse representations of the same input. This diversity enhances the model's ability to generalize and capture complex relationships in the data.

Application Example: Language Translation

Consider translating a sentence from one language to another. The multi-head attention mechanism in the Transformer model helps capture different linguistic structures:

- One head might focus on aligning **subject-verb** pairs between the two languages.
- Another head might capture **longer dependencies**, like how nouns and pronouns refer to each other across a long sentence.
- A third head might capture **grammatical structure** differences between the source and target languages.

By **aggregating these different perspectives**, multi-head attention ensures that the model understands both the local and global context, leading to better translation quality.

Multi Level Index

Multi-level indexing in pandas—also called hierarchical indexing—enables you to work with higher-dimensional data in a 2D DataFrame. It's particularly useful for working with grouped or nested data structures.

Why use multi-level index:

- MultiIndex makes your data **interoperable**
- Enables systematic slicing and aggregation
- Logical grouping of variables

Operations like `.stack()` and `.unstack()` rely on MultiIndex to move between long and wide formats.

- In a flat DataFrame, reshaping often requires column renaming or pivoting.
- With MultiIndex, it's structured and reversible.
- Stack can be used to make a multi index from a flat dataframe.

If you need frequent slicing/aggregation across multiple levels, MultiIndex saves effort and code.

When *not* to use it

- If your data is simple or small.
- If you're just loading, cleaning, and exporting CSVs.
- If you don't need `.groupby(level=...)`, `.stack()`, or `.xs()` operations.

Similar to:

- SQL composite keys
- Python nested dictionaries
- **JSON** hierarchical structures

Related:

- [Groupby](#)
- [Pandas Stack](#)

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/multi_level_index.ipynb

How this mimics a 3D array:

- You can think of each (Product, Store) pair as defining a "slice" of a 2D array.
- The columns (Jan, Feb) represent time-like progression (3rd axis).
- Visually, it's like you've flattened a cube into a matrix while retaining the ability to slice along all original axes.

Month	Jan	Feb
Product	Store	
Product A	Store X	100 110
	Store Y	120 115
Product B	Store X	90 105
	Store Y	95 100

Multicollinearity

When two or more regressors are in [Correlation](#)

Multicollinearity refers to the **instability** of a model due to **highly correlated independent variables**.

It occurs when two or more independent variables in a regression model are highly correlated, which can make it difficult to determine the individual effect of each variable on the dependent variable.

Multicollinearity affects regression models primarily because it leads to instability in the estimated coefficients of the independent variables.

Also see:

- [Addressing Multicollinearity](#)
- [Impact of multicollinearity on model parameters](#)

Related:

- Multicollinearity hurts your hypothesis test
 - Correlation increases bias in the estimated parameters
 - Decreases power via exploded standard errors

Results of Multicollinearity:

1. **Difficulty in Estimating Coefficients:** When independent variables are highly correlated, it becomes challenging to isolate the individual effect of each variable on the dependent variable. This can result in large standard errors for the coefficients, making them unreliable.
2. **Inflated Variance:** The presence of multicollinearity inflates the variance of the coefficient estimates, which can lead to less precise estimates. This means that small changes in the data can lead to large changes in the estimated coefficients.
3. **Misleading Significance Tests:** Multicollinearity can cause some variables to appear statistically insignificant when they might actually be significant. This can lead to incorrect conclusions about the importance of predictors in the model.
4. **Model Interpretation:** The interpretation of the coefficients becomes complicated, as the effect of one variable may be confounded with the effect of another correlated variable.

Key Points

5. Assumption: The multicollinearity assumption suggests that **independent variables should not be collinear**.
6. Detection: Use tools like [Heatmap](#) or [Clustering](#) to visualize [Correlation](#) and identify multicollinearity.
7. Variance Inflation Factor (VIF): High VIF values (typically greater than 10) indicate a high degree of multicollinearity. **Features with high VIF should be dropped to improve model stability.**

Multinomial Naive Bayes

Mysql

MySQL has more **granularity** with types than SQLite. For example, an integer could be `TINYINT` , `SMALLINT` , `MEDIUMINT` , `INT` OR `BIGINT` based on the size of the number we want to store.

The following table shows us the size and range of numbers we can store in each of the integer types.

Data Type	Size (in Bytes)	Minimum Value (Signed)	Maximum Value (Signed)
TINYINT	1	-128	127
SMALLINT	2	-32,768	32,767
MEDIUMINT	3	-8,388,608	8,388,607
INT	4	-2,147,483,648	2,147,483,647
BIGINT	8	-2^{63}	$2^{63} - 1$

Tags

- **Tags:** #relational_database, #data_management

Maintainability

Map Reduce

MapReduce is a programming model and processing technique used for processing and generating large data sets with a parallel, distributed algorithm on a cluster. [Distributed Computing](#)

It is a core component of the Apache Hadoop [Hadoop](#) framework, which is designed to handle vast amounts of data across many servers. The MapReduce model simplifies data processing across large clusters by breaking down the task into two main functions: **Map** and **Reduce**.

MapReduce is particularly effective for [Batch Processing](#) tasks where the data can be processed independently and aggregated later. However, it may not be the best choice for [real-time processing](#) or tasks that require low-latency responses, where other frameworks like [Apache Spark](#) might be more suitable.

Key Components of MapReduce

1. Map Function:

- **Purpose:** To process and transform input data into a set of intermediate key-value pairs.
- **Functionality:** Each input data element is processed independently, and the output is a collection of key-value pairs.
- **Example:** In a word count application, the map function reads a document and emits each word as a key with a count of one as the value.

2. Shuffle and Sort:

- **Purpose:** To organize the intermediate data by keys.
- **Functionality:** The framework sorts the output of the map function and groups all values associated with the same key together. This step is crucial for the reduce function to process data efficiently.

3. Reduce Function:

- **Purpose:** To aggregate and summarize the intermediate data.
- **Functionality:** The reduce function takes the grouped key-value pairs and processes them to produce a smaller set of output values.
- **Example:** Continuing with the word count example, the reduce function sums up the counts for each word, resulting in the total count for each word across all documents.

Why MapReduce is Used

- **Scalability:** MapReduce can process petabytes of data by distributing the workload across a large number of servers in a cluster.
- **Fault Tolerance:** The framework automatically handles failures by reassigning tasks to other nodes, ensuring that the processing continues without data loss.
- **Simplicity:** It abstracts the complexity of parallel processing, allowing developers to focus on the map and reduce logic without worrying about the underlying infrastructure.
- **Flexibility:** MapReduce can be used for a wide range of applications, including data mining, log analysis, and machine learning, among others.
- **Cost-Effectiveness:** By using commodity hardware and open-source software, organizations can process large data sets without significant investment in specialized hardware.

Master Data Management

Master data management is a method to **centralize** master data.

It's the bridge between the business that maintain the data and know them best and the data folks, and it's a tool of choice. It helps with uniformity, accuracy, stewardship, semantic consistency, and accountability of mostly enterprise master data assets.

Master [Data Management](#)(MDM) refers to the processes, technologies, and tools used to define, manage, and maintain an organization's critical data entities, such as customers, products, employees, suppliers, and locations, **ensuring that this data is accurate, consistent, and up-to-date across all systems and departments**. The goal of MDM is to create a single, authoritative [source of truth](#) for master data, which is shared and synchronized across the organization to improve decision-making, reduce duplication, and maintain data integrity.

MDM is especially important in large organizations where data is often siloed across various departments and systems, leading to **inconsistencies, duplication, and errors**. By centralizing the management of key data, MDM helps improve operational efficiency, regulatory compliance, and the overall effectiveness of business processes.

Key aspects of MDM include:

-
1. **Data Governance:** Establishing policies, rules, and standards for how master data is managed, who is responsible for it, and how data quality is monitored.
 2. **Data Integration:** Consolidating and harmonizing data from various sources (e.g., databases, applications) to create a unified, consistent view of master data.
 3. **Data Quality:** Ensuring that the data is complete, accurate, valid, and consistent across the organization.
 4. **Data Stewardship:** Assigning roles and responsibilities for managing the master data and ensuring that it complies with the established governance policies.
 5. **Metadata Management:** Maintaining a consistent definition of data entities, relationships, and attributes, helping stakeholders understand the meaning and usage of the data.
 6. **Data Synchronization:** Ensuring that any updates or changes to master data in one system are reflected across all relevant systems.

Mean Absolute Error

N

Table of Contents

- NLP
- Naive Bayes
- Named Entity Recognition
- Network Design
- Neural Network Classification
- Neural Scaling Laws
- Neural network in Practice
- Neural network
- Ngrams
- NoSQL
- Node.JS
- Non-parametric tests
- Normalisation of Text
- Normalisation of data
- Normalisation vs Standardisation
- Normalisation
- Normalised Schema
- NotebookLM
- nbconvert
- neo4j
- nltk
- npy Files A NumPy Array storage

Nlp

Natural Language Processing (NLP) involves the interaction between computers and humans using natural language. It encompasses various techniques and models to process and analyze large amounts of natural language data.

Key Concepts

Preprocessing

- **Normalisation of Text:** The process of converting text into a standard format, which may include lowercasing, removing punctuation, and stemming or [lemmatization](#).
- **Part of speech tagging:** Assigning a specific part-of-speech category (such as noun, verb, adjective, etc.) to each word in a text.

Models

- **Bag of words:** Represents text data by counting the occurrence of each word in a document, ignoring grammar and word order. It takes key terms of a text in normalized **unordered** form.
 - **TF-IDF:** Stands for Term Frequency-Inverse Document Frequency. It improves on Bag of Words by considering the importance of a word in a document relative to its frequency across multiple documents.
-

- **Vectorization:** Converting text into numerical vectors. Techniques like Bag of Words, TF-IDF, or standardised/Vector Embedding (e.g., Word2Vec, GloVe) are used to represent text data numerically.

Analysis

- **One Hot Encoding:** Converts categorical data into a binary vector representation, indicating the presence or absence of a word from a list in the given text.

Methods

- **Ngrams:** Creates tokens from groupings of words, not just single words. Useful for capturing context and meaning in text data.
- **Grammar method:** Involves analyzing the grammatical structure of sentences to extract meaning and relationships between words.

Actions

- **Summarisation:** The process of distilling the most important information from a text to produce a concise version.

Tools and Libraries

General Imports

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer, PorterStemmer

porter_stemmer = PorterStemmer()
wordnet_lemmatizer = WordNetLemmatizer()
```

- **nltk:** A leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources.
 - **punkt:** An unsupervised trainable model for tokenizing text into sentences and words.
 - **stopwords:** Commonly used words (such as "the", "is", "in") that are often removed from text data because they do not carry significant meaning.
 - **wordnet:** A lexical database for the English language that groups words into sets of synonyms and provides short definitions and usage examples.
 - **re:** Regular expressions for pattern matching and text manipulation.

Naive Bayes

Can values for X,y be categorical ? [Encoding Categorical Variables](#)

BernoulliNB()

Introduction

Why Naive Bayes? Order doesn't matter, features are independent. Treated it as a [Bag of words](#). Which **simplifies** the above equation.

Want to use this in classifiers for ML Want to understand: [Multinomial Naive bayes](#) classifier There is also: [Gaussian Naive Bayes](#)

Issues

To avoid having 0 probability sometimes they add **counts** α to do this.

Links:

<https://youtu.be/PPeaRc-r1OI?t=169>

Formula

$$P(A|B) = P(A) \times \frac{P(B|A)}{P(B)}$$
 Think of the line as "given".

Examples

Example 1



Example 2

In the formula above $P(A)$ is $P(+)$, $P(B)=P(\text{NEW})$

$$P(B|A) = P(A=0|+) \dots P(C=0|+)$$

$P(A=0, B=1, C=0)$ is the same for both + and - class so remove.



Example Car accidents

What's the probability of car having an accident given that driver is driving in summer, there is no rain, it's a night and it's an urban area?

Mock data:

Season	Weather	Daytime	Area	Did Accident Occur?
Summer	No-Raining	Night	Urban	No
Summer	No-Raining	Day	Urban	No
Summer	Raining	Night	Rural	No
Summer	Raining	Night	Urban	Yes
Summer	Raining	Day	Urban	No
Summer	Raining	Night	Rural	No
Winter	Raining	Night	Urban	Yes
Winter	Raining	Night	Urban	Yes
Winter	Raining	Night	Rural	Yes
Winter	No-Raining	Night	Rural	No
Winter	No-Raining	Night	Urban	No
Winter	No-Raining	Day	Urban	Yes
Spring	No-Raining	Night	Rural	Yes
Spring	No-Raining	Day	Rural	Yes
Spring	Raining	Night	Urban	No
Spring	Raining	Day	No	No
Spring	No-Raining	Night	Urban	No
Autumn	Raining	Night	Urban	Yes
Autumn	Raining	Day	Rural	Yes
Autumn	No-Raining	Night	Urban	No
Autumn	No-Raining	Day	Rural	No
Autumn	No-Raining	Day	Urban	No
Autumn	Raining	Day	Yes	No
Autumn	Raining	Night	Yes	No
Autumn	No-Raining	Night	No	No

To handle data like this it is possible to calculate frequencies for each case:

0. Accident probability

$$P(\text{Accident}) = \frac{9}{25} = 0.36$$

$$P(\text{No-Accident}) = \frac{16}{25} = 0.64$$

1. Season probability

Frequency table:

Season	Accident	No Accident	
Spring	2/9	3/16	5/25
Summer	1/9	5/16	6/25
Autumn	2/9	6/16	8/25
Winter	4/9	2/16	6/25
	9/25	16/25	

Probabilities based on table:

$$P(\text{Spring}) = \frac{5}{25} = 0.20$$

$$P(\text{Summer}) = \frac{6}{25} = 0.24$$

$$P(\text{Autumn}) = \frac{8}{25} = 0.32$$

$$P(\text{Winter}) = \frac{6}{25} = 0.24$$

$$P(\text{Spring} \mid \text{Accident}) = \frac{2}{9} = 0.22$$

$$P(\text{Summer} \mid \text{Accident}) = \frac{1}{9} = 0.11$$

$$P(\text{Autumn} \mid \text{Accident}) = \frac{2}{9} = 0.22$$

$$P(\text{Winter} \mid \text{Accident}) = \frac{4}{9} = 0.44$$

2. Weather probability

Frequency table:

	Accident	No Accident	
Raining	6/9	7/16	13/25
No-Raining	3/9	9/16	12/25
	9/25	16/25	

Probabilities based on table:

$$P(\text{Raining}) = \frac{13}{25} = 0.52$$

$$P(\text{No-Raining}) = \frac{12}{25} = 0.48$$

$$P(\text{Raining} \mid \text{Accident}) = \frac{6}{9} = 0.667$$

$$P(\text{No-Raining}|\text{Accident}) = \frac{12}{25} = 0.333$$

3. Daytime probability

Frequency table:

	Accident	No Accident	
Day	3/9	6/16	9/25
Night	6/9	10/16	16/25
	9/25	16/25	

Probabilities based on table:

$$P(\text{Day}) = \frac{9}{25} = 0.36$$

$$P(\text{Night}) = \frac{16}{25} = 0.64$$

$$P(\text{Day}|\text{Accident}) = \frac{3}{9} = 0.333$$

$$P(\text{Night}|\text{Accident}) = \frac{6}{9} = 0.667$$

4. Area probability

Frequency table:

	Accident	No Accident	
Urban Area	5/9	8/16	13/25
Rural Area	4/9	8/16	12/25
	9/25	16/25	

Probabilities based on table:

$$P(\text{Urban}) = \frac{13}{25} = 0.52$$

$$P(\text{Rural}) = \frac{12}{25} = 0.48$$

$$P(\text{Urban}|\text{Accident}) = \frac{5}{9} = 0.556$$

$$P(\text{Rural}|\text{Accident}) = \frac{4}{9} = 0.444$$

Assemble:

Calculating probability of car accident occurring in summer, when there is no rain and during night, in urban area.

Where B equals to:

- Season: Summer
- Weather: No-Raining
- Daytime: Night
- Area: Urban

Where A equals to:

- Accident

Using Naive Bayes:

$$P(A|B) = P(\text{Accident} | \text{Season} = \text{Summer}, \text{Weather} = \text{No-Raining}, \text{Daytime} = \text{Night}, \text{Area} = \text{Urban})$$

$$P(A|B) = \frac{P(\text{Summer}|\text{Accident})P(\text{No-Raining}|\text{Accident})P(\text{Night}|\text{Accident})P(\text{Urban}|\text{Accident})P(\text{Accident})}{\{P(\text{Summer})P(\text{No-Raining})P(\text{Night})P(\text{Urban})\}}$$

$$P(A|B) = \frac{\frac{1}{9}\frac{6}{9}\frac{6}{9}\frac{5}{9}\frac{9}{25}}{\{\frac{6}{25}\frac{12}{25}\frac{16}{25}\frac{13}{25}\}} = \frac{0.111 \cdot 0.667 \cdot 0.667 \cdot 0.556 \cdot 0.36}{0.24 \cdot 0.48 \cdot 0.64 \cdot 0.52} = \frac{0.0099}{0.038} = 0.26$$

$$P(A)=P(\text{Accident})$$

$$P(B)=P(\text{Summer})P(\text{No-Raining})P(\text{Night})P(\text{Urban})$$

$$P(B|A)=P(\text{Summer}|\text{Accident})P(\text{No-Raining}|\text{Accident})P(\text{Night}|\text{Accident})P(\text{Urban}|\text{Accident})$$

What is the Bayes theorem?; The formula is $P(A|B) = P(B|A) * P(A) / P(B)$.

What are the main advantages of Naive Bayes, and when is it commonly used?; simplicity, quick implementation, and scalability, used in text classification.

**When using Naive Bayes with numerical variables, what condition is assumed on the data?; Naive Bayes assumes that numerical variables follow a normal distribution.

How does Naive Bayes perform with categorical variables? makes no assumptions about the data distribution.

What is Naive Bayes, and why is it called "naive"?; Algo which uses Bayes theorem, used for classification problems. It is "naive" because it assumes that predictor variables are independent, which may not be the case in reality. The algorithm calculates the probability of an item belonging to each possible class and chooses the class with the highest probability as the output.

Naive Bayes Naive Bayes classifiers are based on Bayes' theorem and assume that the features are conditionally independent given the class label.

Naive Bayes

- A probabilistic classifier based on Bayes' theorem.
- Simple and fast, especially effective for text classification.

Named Entity Recognition

Named Entity Recognition (NER) is a subtask of [NLP|Natural Language Processing](#) (NLP) that involves identifying and classifying key entities in text into predefined categories such as names, organizations, locations.

The process typically employs algorithms like Conditional Random Fields (CRFs) or deep learning models such as Bi-directional [LSTM](#) (Long Short-Term Memory) networks.

Mathematically, NER can be framed as a sequence labeling problem where the goal is to assign a label y_i to each token x_i in a sentence. The model learns from annotated datasets, optimizing parameters to maximize the likelihood $P(y|x)$ using techniques like [backpropagation](#).

NER has significant implications in information extraction, search engines, and automated customer support systems.

Important

- NER transforms unstructured text into [structured data](#) for analysis.
- The choice of model significantly impacts the accuracy of entity recognition.

Example

An example of NER is identifying "Apple Inc." as an organization in the sentence: "Apple Inc. released a new product."

Follow up questions

- How does the choice of training data affect the performance of NER models
- What are the challenges of NER in multilingual contexts
- Why is named entity recognition (NER) a challenging task
- In NER how would you handle ambiguous entities

Related Topics

- Text classification in [NLP](#)
- Information extraction techniques

Network Design

Mixed-Integer Programming: Handles problems where some variables must be integers, commonly used in optimizing network design and capacity planning.

How systems interact.

Neural Network Classification

Choosing Thresholds/Clusters in [Neural network Classification](#)

When working with [Deep Learning|neural networks](#), the output is often a probability distribution across different classes. To make a final classification decision, we need to convert these probabilities into discrete class labels. This is typically done by comparing the probabilities against a threshold or by clustering them.

Threshold-Based Classification

In threshold-based classification, we set a specific probability value as the threshold. If the probability of a class exceeds this threshold, the input is classified as belonging to that class. Otherwise, it's classified as belonging to another class or as "unknown."

[Choosing a Threshold](#)

Clustering-Based Classification

In [clustering](#)-based classification, we group the probability distributions into clusters. Each cluster represents a class. This approach is useful when the class boundaries are not well-defined or when there are multiple overlapping classes.

[Choosing the Number of Clusters](#)

Additional Considerations:

- [Imbalanced Datasets](#): If the classes are imbalanced, the choice of threshold or number of clusters can be significantly affected. Techniques like oversampling, undersampling, or using weighted loss functions can help mitigate the impact of class imbalance.
- [Data Quality](#): The quality of the training data can also influence the choice of threshold or number of clusters. If the data is noisy or contains outliers, the chosen values may not be optimal.
- [Evaluation Metrics](#): Choose evaluation metrics that are appropriate for the specific problem and the desired trade-off between different types of errors.

Neural Scaling Laws

Even scaled model cannot cross the [compute efficient frontier](#)



[validation loss](#)

Neural scaling laws. That is error rates scale with compute, model size and dataset size, independent of model architecture. Can we drive to 0?

Same laws appear in video and image models.

LLMs are auto progressive models.

Theoretical results guiding experimental - saving compute time.

How LLM|LLMs work



During training we know next value, hence we have a [Loss function](#) to help learning.

L1 - loss functions

[Cross Entropy](#)-loss function (uses negative log of probability). Why is cross entropy used over L1?

unambiguous next words. [Entropy of natural language](#) due to this will LLMs cannot drive [Cross entropy loss](#) to zero.

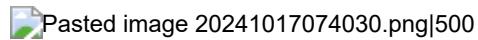
Manifolds?

Example [MNIST](#) data set images of numbers, has high dimensional dataset space.



[16:03](#) Similar concepts group together.

density of manifold average distance between point. or size of neighbourhoods s



$S=L D^{-1/d}$

Manifold hypothesis (data points in high dim space) and scaling laws

Knowing the manifold will help scaling. This is called

[Resolution limited scaling](#)

$$LOSS < D^{-4/d}$$

[Cross entropy loss](#) should scale wrt manifold.

19:24

[intrinsic dimension of natural language](#)

Neural Network In Practice

This guide provides practical insights into building and using [Neural network](#).

Refer to [ML_Tools](#) for more details: `Neural_Net_Build.py`

Softmax Placement at the End

Numerical stability is crucial. One way to enhance stability is by grouping the softmax function with the loss function rather than placing it at the output layer.

Building the Model

Final Dense Layer:

- Use a 'linear' activation function, which means no activation is applied. This setup allows the model to output raw logits.

Model Compilation:

- When compiling the model, specify `from_logits=True` in the loss function to indicate that the outputs are raw logits.

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Target Form:

- The target format remains unchanged. For instance, with SparseCategoricalCrossentropy, the target is the expected class label (e.g., digits 0-9 in the MNIST dataset).
- See [SparseCategoricalCrossentropy](#) or [CategoricalCrossEntropy](#)

Output Probabilities:

- Since the model outputs logits, apply a softmax function to convert these logits into probabilities if needed for interpretation or further processing.

TensorFlow History Loss (Cost)

Monitoring the cost, or loss, during training is essential for understanding how well the model is learning.

Monitoring Progress:

- Track the progress of gradient descent by observing the cost, referred to as `loss` in TensorFlow. Ideally, the loss should decrease as training progresses. **Loss Display**:
- The loss is displayed at each epoch during the execution of `model.fit`, providing real-time feedback on training performance. **History Variable**:
- The `.fit` method returns a `history` object containing various metrics, including the loss. This object is stored in the `history` variable for further analysis.

The `history` object can capture additional metrics such as accuracy, validation loss, and other performance indicators, depending on what was specified during model compilation and fitting. This information is valuable for evaluating the model's performance [Model Evaluation](#).

Neural Network

A [Neural network|Neural Network](#) is a computational model inspired by biological neural networks in the human brain. It consists of layers of interconnected nodes (neurons) that process and transmit information. Neural networks are fundamental to [Deep Learning](#).

Resources:

- [Keras Guide](#)

Also see:

- [Types of Neural Networks](#)
- [Neural network in Practice](#)

Key Components

The number of starting nodes depends on the input parameter, similar for output. The width and depth of the net are called [Hyperparameter](#).

Neurons (Nodes):

- The basic units of a neural network. Each neuron receives input, processes it, and passes it to the next layer. A neuron's output is typically computed using a mathematical function known as the activation function.

Layers:

- Neural networks are structured in layers of neurons:
 - Input Layer: Receives the raw input data (features) that are fed into the network.
 - Hidden Layers: Process the inputs received from the previous layer. There can be multiple hidden layers, making a neural network "deep." These layers transform the data to learn complex relationships and patterns.
 - Output Layer: Produces the final prediction or result, such as a class label in classification tasks or a continuous value in regression.

Weights and Biases: [Model Parameters](#)

- There are weights and biases at each layer. The shape of the weights is determined by the number of units in the previous layer and the number of units in the current layer.

Introduction

- Each connection between neurons has a weight that determines how much influence one neuron has on another. Weights are adjusted during the learning process to minimize prediction errors.
- Biases allow the network to shift the output of the [activation function](#) and help it better fit the data.

Training: See [Fitting weights and biases of a neural network](#)

Optimization: See [Optimisation techniques](#)

- The optimization process (often gradient descent) updates the network's weights to minimize the loss function, ensuring the model improves with training and generalizes well to new, unseen data.

Inputs:

- We need [Normalisation](#) of values (inputs) here for feeding the network to have balanced weights at the nodes.

Context

Example of Neural Network:

A neural network can be used for a task like image classification. For instance:

- The input layer receives the pixel values of the image.
- Hidden layers transform these pixel values through a series of mathematical operations, learning important features such as edges, shapes, and textures.
- The output layer classifies the image into one of the predefined categories (e.g., "cat" or "dog").

Pros:

- Flexibility: Can model complex, non-linear relationships.
- Adaptability: Can be applied to a wide range of problems like image recognition, speech processing, and game playing.
- Automatic Feature Extraction: Neural networks, especially CNNs, can automatically learn important features from raw data without manual intervention.

Cons:

- Data-hungry: Neural networks typically require large datasets to perform well.
- Computationally Intensive: Training deep networks can require substantial computational resources.
- Black Box Nature: The internal decision-making process is often difficult to interpret, although research into interpretability is addressing this.

Ngrams

N-grams are used in NLP that allow for the analysis of text data by breaking it down into smaller, manageable sequences.

An **N-gram** is a contiguous sequence of n items (or tokens) from a given sample of text or speech. In the context of natural language processing (NLP) and text analysis, these items are typically words or characters.

N-grams are used to analyze and [model the structure of language](#), and they can help in various tasks such as [text classification](#).

Types of N-grams

- **Unigram:** An N-gram where $n = 1$. It represents individual words or tokens. For example, in the sentence "I love AI", the unigrams are ["I", "love", "AI"].
- **Bigram:** An N-gram where $n = 2$. It represents pairs of consecutive words. For the same sentence, the bigrams would be ["I love", "love AI"].
- **Higher-order N-grams:** These can go beyond three words, such as 4-grams (quadgrams) or 5-grams, and so on.

Code implementations:

This can be done through kwargs in CountVectorizer.

Nosql

(Not Only SQL):** **Non-relational** database management systems offering flexibility and scalability for unstructured or document-based data.

NoSQL Databases: Accommodate unstructured data and can be represented through graph theory or document-based structures, allowing for flexible data models.

Node.Js

Non Parametric Tests

Normalisation Of Text

[Preprocessing](#) in NLP tasks is called Normalization involves reducing words to their base or root form, converting them to lowercase, and removing stop words.

Processes

What are some steps involved in the pre-processing of a text?. These include making the text lower case, removal of punctuation, tokenize the text (split up the words in a sentence), remove stop words as they convey grammar rather than meaning, word stemming (reduce words to their stems).

[Tokenisation:](#) Used to separate words or sentences.

[Stemming:](#) returns part of a words that doesn't change ie breaks, breakthrough gives break. Use

```
from nltk.stem.porter import PorterStemmer
temp=text #decomposed
porter_stemmer = PorterStemmer()
stemmed_tokens = [porter_stemmer.stem(token) for token in temp]

print(stemmed_tokens)
```

lemmatization: reducing word to it's base form e.g. words "is", "was", "were" will turn into "be".

```
from nltk.stem.wordnet import WordNetLemmatizer
temp=text #decomposed
wordnet_lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [wordnet_lemmatizer.lemmatize(token, pos="v") for token in temp]
print(lemmatized_tokens)
```

Code

main normaliser

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

def normalize_document(document, stemmer=porter_stemmer, lemmatizer=wodnet_lemmatizer):
    """Normalizes data by performing following steps:
    1. Changing each word in corpus to lowercase.
    2. Removing special characters and interpunction.
    3. Dividing text into tokens.
    4. Removing english stopwords.
    5. Stemming words.
    6. Lemmatizing words.
    """
    temp = document.lower()
    temp = re.sub(r"[^a-zA-Z0-9]", " ", temp)
    temp = word_tokenize(temp)
    temp = [t for t in temp if t not in stopwords.words("english")]
    temp = [porter_stemmer.stem(token) for token in temp]
    temp = [lemmatizer.lemmatize(token) for token in temp]
    return temp
```

Normalisation Of Data

Normalization is the process of structuring data from the source into a format appropriate for consumption in the destination.

Introduction

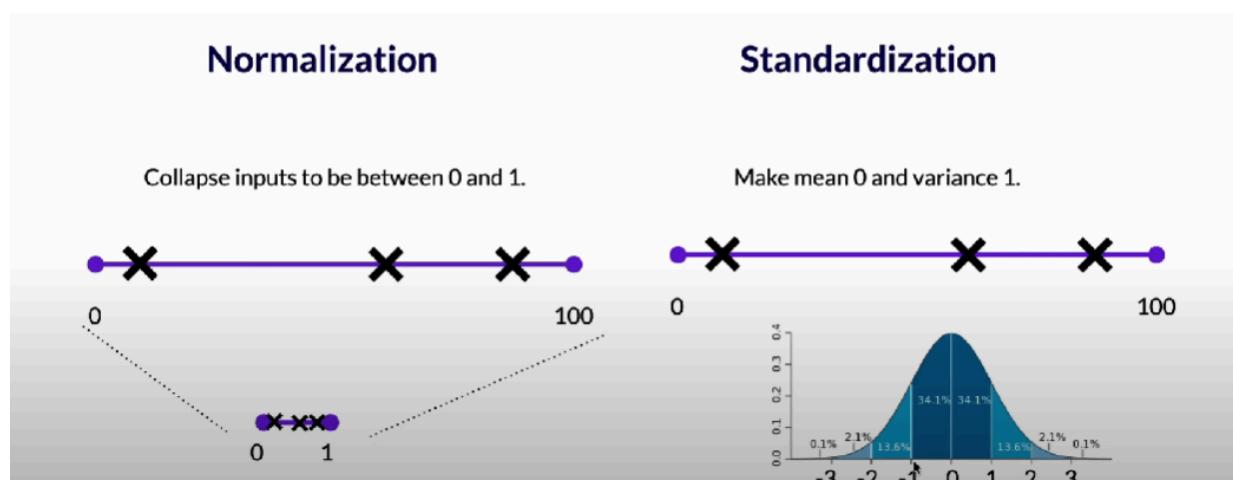
For example, when writing data from a nested, dynamically typed source like a [JSON API](#) to a relational destination like [PostgreSQL](#), normalization is the process that un-nests JSON from the source into a relational table format that uses the appropriate column types in the destination.

Normalisation Vs Standardisation

Key Differences:

[Normalisation](#) changes the range of the data, while standardisation changes the data distribution.

Normalisation is preferred when the data does not follow a Gaussian distribution, whereas [standardisation](#) is used when the data is normally distributed.



Normalisation

Standardizing data distributions for consistency.

In ML:

- Z-Normalisation
- Standardisation
- Normalisation vs Standardisation
- Batch Normalisation

In [Data Engineering](#):

- Normalisation of data
- Normalised Schema
- How to normalise a merged table

In [NLP](#):

- Normalisation of Text

```
# --- 15. GroupBy with Transformation (Using transform to align with original dataframe)
df['Value_transformed'] = df.groupby('Category')['Value'].transform(lambda x: x - x.mean())

# get the mean value for each category
print(df.groupby('Category')['Value'].mean())
print("\nTransformed Values with mean subtracted (transform()):")
print(df.sort_values('Category'))
```

Normalised Schema

In a normalized schema, data is organized into multiple related tables to minimize redundancy and dependency, and improve data integrity.

This approach is often used in transactional databases (OLTP) to ensure data integrity. However, it can lead to complex queries and slower performance for analytical queries.

Normalization involves organizing the columns (attributes) and tables (relations) in a database to ensure proper enforcement of dependencies through database integrity constraints.

This is achieved by applying formal rules during the creation of a new database design or **decomposition** (improvement of an existing database design) process.

1. First Normal Form (1NF):

- o Eliminate duplicate data by ensuring each attribute contains only atomic values and each table has a unique primary key.

2. Second Normal Form (2NF):

- o Meet all requirements of 1NF and **remove partial dependencies** by ensuring that **every non-prime attribute (attribute not part of any candidate key)** entirely depends on the primary key.

3. Third Normal Form (3NF):

- o Meet all requirements of 2NF and remove **transitive dependencies** by ensuring that no non-prime attribute is transitively dependent on the primary key.

[See here for an example](#)

[How to normalise a merged table](#)

Denormalization

Denormalization, on the other hand, is the process of intentionally introducing redundancy into a database design by combining tables or adding redundant data, aiming to improve query performance or simplify the database structure. Denormalization is the **opposite of normalization**. Please consider the trade-offs between data integrity and query performance. This technique is used with [Dimensional Modeling](#) in [OLAP](#).md cubes, for example.

Related to:

[Normalisation of data](#)

Notebooklm

<https://www.youtube.com/watch?v=EOmgC3-hznM>

key topics

chat interface takes into account resources.

save to note- to dave.

how to select and folders - from obsidian [Data Archive](#) for this ? A getter of some kind

can convert multiple notes into a single note.

Can add website as source.

project context - similar projects notes

Focus knowledge retrieval

- get info from sources (folders)

FAQ

Note: #portal can help with file extraction rem utils function (for [NotebookLM](#))

Nbconvert

Neo4J

Nltk

Npy Files A Numpy Array Storage

A .npy file is a binary file format specifically designed to store a single NumPy array. NumPy, or Numerical Python, is a powerful library in Python used for numerical computing and data analysis.

Why Use .npy Files?

- Efficiency: Storing data in binary format is generally more efficient than storing it in text-based formats like CSV or JSON. This means faster read/write operations and less disk space usage.
- Preserves Data Structure: .npy files maintain the exact structure and data type of the NumPy array, ensuring that the data can be loaded back into memory without any loss of information.
- Simple Format: The .npy format is relatively straightforward, making it easy to read and write using NumPy's built-in functions.

How to Create and Load .npy Files

Creating an .npy File:

Introduction

```
#1. Import NumPy:  
import numpy as np  
  
#2. Create a NumPy Array:  
my_array = np.array([[1, 2, 3], [4, 5, 6]])  
  
#3. Save the Array to a .npy File:  
np.save('my_array.npy', my_array)  
  
#Loading a .npy File:  
#1. Load the Array from the .npy File:  
  
loaded_array = np.load('my_array.npy')
```

O

Table of Contents

- [OLAP \(online analytical processing\)](#)
- [OLAP](#)
- [OLTP](#)
- [One Pager Template](#)
- [One-hot encoding](#)
- [One_hot_encoding.py](#)
- [Optimisation function](#)
- [Optimisation techniques](#)
- [Optimising Neural Networks](#)
- [Optimising a Logistic Regression Model](#)
- [Optuna](#)
- [Ordinary Least Squares](#)
- [Orthogonalization](#)
- [Outliers](#)
- [Over parameterised models](#)
- [Overfitting](#)
- [oltp \(online transactional processing\)](#)

Olap (Online Analytical Processing)

OLAP, or Online Analytical Processing, is a category of database technology.

OLAP systems allow organizations to gain insights by examining data across various dimensions, such as time, product, and region.

[Excel pivot table](#)

Key Features of OLAP & Operations

Query Performance Aggregation and Summarization across dimensions.

- **Slicing:** Extracting a single layer of data from the cube by selecting a specific dimension (e.g., sales for Q1).
- **Dicing:** Selecting a subcube by specifying values for multiple dimensions.
- **Drill Down:** Moving from a summary level to a more detailed level (e.g., from yearly to monthly sales).
- **Roll Up:** Aggregating data to a higher level (e.g., from daily to monthly sales).
- **Pivoting:** Rotating the data to view it from a different perspective (e.g., switching rows and columns).

Use Cases of OLAP

- **Business Intelligence (BI):** OLAP tools are integral to BI solutions, allowing for the analysis of financial data, sales performance, and other key metrics.
- **Data Warehousing:** OLAP is commonly used with data warehouses, where large volumes of historical data are stored for reporting and analysis.

Visualization Tools

To interact with the OLAP cube, users typically utilize tools such as:

- **Microsoft Power BI:** For creating dashboards and visualizations.
- **Excel with Pivot Tables:** For slicing, dicing, and reporting.
- **Tableau:** For visual analysis.

Olap

Oltlp

In online transaction processing (**OLTP**), information systems typically facilitate and manage **transaction-oriented** applications. It's the opposite of [OLAP \(Online Analytical Processing\).md](#).

Proposal: [Project name]

About this doc

Metadata about this document. Describe the scope and current status.

This doc is a proposal for [feature or change]. Upon approval, we will look to have this prioritized as a project and do a full Technical Design Document.

Sign off deadline	<i>Date</i>
Status	<i>Draft</i>
Author(s)	<i>Name 1, Name 2</i>

Sign offs

- *Name 1*
- *Name 2*
- Add your name here to sign off

Problem

What is the problem being solved? What are the pain points? What is the current solution and why is not good enough?

High level goal

Why should we do this? Answer this in metrics ideally but otherwise a sentence or two is okay.

What will happen if we don't solve this?

Make it clear the downsides of what will happen if we don't invest the time into this.

Proposed solution: [Option name]

State the option you suggest and explain your reasoning. What benefits will we get from this approach? Time, money, risk, convenience, etc.

Alternatives

A table or summary of the other options to achieve the goal. Also, consider adding this to an Appendix to keep the doc focused too.

- Option 1: ...
 - Pros: ...
 - Cons: ...
- Option 2: ...
 - Pros: ...
 - Cons: ...
- ...

Risks

What can go wrong with the proposed approach? How are you mitigating that?

- Risk 1
- Risk 2
- ...

Open Questions [optional]

Anything still being figured out that you could use some additional eyes or thoughts on.

One Hot Encoding

Related terms:

- Why do we need to drop one of the dummy columns? [Dummy variable trap](#):

[Dummy variables & One-hot encoding are fundamentally different from Label encoding](#)

[Why does label encoding give different predictions from one-hot encoding](#)

One-hot encoding is a technique used to convert categorical data into a numerical format that can be used by machine learning algorithms. It is particularly useful when dealing with categorical variables that have no ordinal relationship.

In one-hot encoding, each category is transformed into a binary vector. If there are (n) unique categories, each category is represented by a vector of length (n) where one element is "hot" (set to 1) and the rest are "cold" (set to 0). For example, if you have a categorical variable with three categories: "red," "green," and "blue," one-hot encoding would represent them as:

- "red" -> [1, 0, 0]
- "green" -> [0, 1, 0]
- "blue" -> [0, 0, 1]

One-hot encoding is used because many machine learning algorithms cannot work directly with categorical data. By converting categories into a numerical format, one-hot encoding allows these algorithms to process the data effectively. It is commonly used in preprocessing steps for machine learning models, especially in neural networks and other algorithms that require numerical input.

Implementation

In [ML_Tools](#) see: [One_hot_encoding.py](#)

```
cat_variables = ['Sex',
'ChestPainType',
'RestingECG',
'ExerciseAngina',
'ST_Slope'
]

# This will replace the columns with the one-hot encoded ones and keep the columns outside 'columns' argument as it is.
df = pd.get_dummies(data = df, prefix = cat_variables, columns = cat_variables)
```

One_Hot_Encoding.Py

Explorations\Preprocess\One_hot_encoding\One_hot_encoding.py

This script demonstrates how to preprocess categorical variables and apply linear regression for house price prediction. Key steps include:

1. **Data Loading:** It loads a dataset of house prices.
2. **Dummy Variables:** It creates dummy variables for the 'town' column using `pd.get_dummies()` and merges them with the original dataframe.
3. **Dummy Variable Trap:** It drops one dummy variable to avoid multicollinearity (dummy variable trap).
4. **Feature and Target Split:** It separates the dataset into features (X) and the target variable (price).
5. **Model Training:** A Linear Regression model is trained on the data.
6. **Predictions:** It predicts house prices based on various features and evaluates the model's accuracy.
7. **Label Encoding and One-Hot Encoding:** It applies `LabelEncoder` to convert 'town' names into numbers and uses `OneHotEncoder` to create dummy variables for categorical columns.
8. **Final Predictions:** It predicts prices using the transformed features and evaluates the model's performance.

Optimisation Function

Optimization functions adjust the [Model Parameters](#) to minimize the [Loss function](#), which measures how well the model performs. This is a fundamental step in training machine learning models.

General Optimization Process:

The [Optimisation function](#) (e.g., LBFGS, Newton-CG) iteratively updates the [Model Parameters](#) by:

1. Calculating the gradient of the loss function with respect to the parameters.
2. Updating the parameters in the direction of the negative gradient (as described in [Gradient Descent](#)).

This process is repeated until:

- The cost function converges (i.e., the change in the loss function becomes negligible), or
- The maximum number of iterations is reached.

See [Optimisation techniques](#).

Optimisation Techniques

[Optimisation techniques](#)

- [Adam Optimizer](#)
- [RMSprop](#)
- [Stochastic Gradient Descent](#)
- [standardised/Optuna](#)

[Gradient Descent](#)

- Iteratively updates parameters using the gradient of the [Cost Function](#) with respect to the parameters.
- Requires careful tuning of the [Learning Rate](#) (α), which controls the size of each update.

Optimization Solvers in [Sklearn](#) : Scikit-learn solvers improve on Gradient Descent by leveraging advanced techniques:

- Use second-order information, such as approximations to the Hessian matrix.
- Achieve faster and more reliable convergence compared to Gradient Descent.
- Automatically adapt step sizes [Adaptive Learning Rates](#), eliminating the need for manual tuning.

Optimising Neural Networks

[Deep Learning](#)

Ways to improve in using a [Neural network](#) more data, bigger network, diverse training set, try dropout, change network architecture.

Need strategies that will point towards what's the best methods to try.

Optimising a Logistic Regression Model

In `sklearn`, the logistic regression model uses an optimization algorithm to find the best parameters (intercept and coefficients) that minimize a loss function, typically the logistic loss (cross-entropy loss). Here's an overview of how it works:

Optimization process: `sklearn` optimizes the logistic regression parameters using iterative solvers (like LBFGS, newton-cg, or liblinear) that minimize the logistic loss function. `sklearn` is using one of these optimization algorithms (likely `lbfgs` or `liblinear` by default) to minimize the logistic loss function.

Objective Function (Loss Function)

The objective of logistic regression is to minimize the logistic [loss function](#) (also called cross-entropy loss) given by:

$$\text{Cost}(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(h(\theta)(x^{(i)})) + (1 - y^{(i)}) \log(1 - h(\theta)(x^{(i)})) \right]$$

Where:

- $h(\theta)(x) = \frac{1}{1 + \exp(-\theta^T x)}$ is the sigmoid function that predicts probabilities.
- $y^{(i)}$ is the actual label for the i -th sample (either 0 or 1).
- m is the number of samples.
- θ is the vector of parameters (intercept and coefficients).

The goal is to minimize this [cost function](#) by finding the optimal θ values (intercept and coefficients).

Optimization Algorithm (Solvers)

`sklearn` provides different [Optimisation function](#) to find the optimal [Model Parameters](#) for logistic regression. These solvers use optimization techniques like Gradient Descent or more advanced methods like Newton's Method or Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS). Here are the main solvers used in `sklearn`:

- 'liblinear': This solver uses coordinate descent or regularized Newton's method and is a good choice for smaller datasets. It supports both L1 (`Lasso`) and L2 (`Ridge`) regularization.
- 'lbfgs': This is a quasi-Newton method (Limited-memory BFGS), which approximates the Newton's method and is more efficient for larger datasets. It's an iterative solver that converges faster and requires fewer iterations than simple gradient descent.
- 'newton-cg': This solver is based on Newton's method, which computes the second-order derivative (Hessian matrix) and updates parameters using the inverse of this matrix. It works well for logistic regression with large datasets and is efficient when the number of features is large.
- 'saga': An efficient solver that can handle L1 regularization (for sparse solutions) and large datasets. It's a variant of Stochastic Gradient Descent (SGD) and supports L1, L2, and ElasticNet regularization.
- 'sgd': This is a solver for stochastic [Gradient Descent](#), which updates parameters iteratively using only a single data point (or a small batch) at a time. It can be slower but is useful for very large datasets.

5. Convergence Criteria

`sklearn`'s logistic regression solvers have specific criteria for when to stop the optimization process:

- Convergence tolerance (`tol`): This is the threshold for when the optimization stops. When the improvement in the cost function between iterations becomes smaller than `tol`, the process stops.

- Maximum iterations (`max_iter`): The maximum number of iterations allowed before the solver stops. If convergence is not achieved within the allowed iterations, the algorithm will stop, and a warning will be issued.

Optuna

Optuna is a [hyperparameter](#) optimization framework used to automatically tune hyperparameters for machine learning models.

Optuna automates the process of tuning hyperparameters by defining an objective function, testing different hyperparameter combinations, training the model, and evaluating its performance. The best set of hyperparameters is chosen based on the performance metric (e.g., test accuracy) returned by the objective function.

[Hyperparameter Tuning](#)

Benefits of Using Optuna

1. Efficient Search: Utilizes algorithms like TPE (Tree-structured Parzen Estimator) to search the hyperparameter space more efficiently than grid search.
2. Dynamic Search Space: Can explore continuous, categorical, and discrete spaces.
3. Automatic Pruning: Supports pruning of unpromising trials during training, improving computational efficiency.
4. Visualization: Offers built-in tools for visualizing the optimization process, aiding in understanding the impact of hyperparameters.

Steps to Use Optuna

1. Define Objective Functions:

- For each model (e.g., [LightGBM](#), [XGBoost](#), [CatBoost](#)), define an objective function.
- The objective function takes trial parameters as input and returns a score to optimize.
- Specify hyperparameters to tune within each function, such as:
 - LightGBM: learning rate, number of leaves
 - XGBoost: eta, max depth
 - CatBoost: learning rate, depth

2. Running Hyperparameter Optimization:

- Create a study object for each model using `optuna.create_study()`.
- Run the optimization process using the `.optimize()` method, specifying the objective function and the number of trials.
- Retrieve the best hyperparameters from each study object using `.best_params`.

3. Comparison and Evaluation:

- Compare the best hyperparameters obtained for each model.
- Evaluate the performance of the tuned models on a validation dataset.

Differences between Models with Optuna

Hyperparameters:

- The specific hyperparameters to tune may vary between models.

Introduction

- Example: LightGBM involves tuning parameters like learning rate and number of leaves, while XGBoost involves parameters like eta and max depth.

Objective Function:

- Tailor the objective function for each model to its respective API and requirements.
- Ensure the objective function properly trains and evaluates the model using the specified hyperparameters.

Optimization Strategy:

- Optuna provides different optimization algorithms (e.g., TPE, CMA-ES) that may behave differently depending on the model and hyperparameter space.

Implementation

```
import optuna

# 1. Creating an Optimization Study
# Initialize a study to track and manage the optimization process.
# The 'direction' parameter specifies whether to maximize or minimize the objective function.
study = optuna.create_study(direction='maximize')

# 2. Defining the Objective Function
def objective(trial):
    # 3. Suggesting Hyperparameters
    # Use trial.suggest_* methods to specify the hyperparameters to optimize.
    # Each method defines the type and range of values to explore.
    learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e-2)
    batch_size = trial.suggest_categorical('batch_size', [32, 64, 128])
    dropout = trial.suggest_uniform('dropout', 0.2, 0.5)
    units_1 = trial.suggest_int('units_1', 64, 128)

    # 4. Training the Model
    # Train the model using the suggested hyperparameters.
    # Evaluate the model's performance on the validation dataset.
    model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=config['epochs'], batch_size=batch_size, callbacks=)

    # Returning the Result
    # Return the model's test accuracy as the result of the trial.
    return test_accuracy

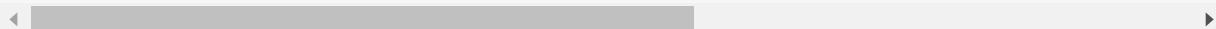
# Optimization
# Run the optimization process, executing the objective function multiple times.
# The 'n_trials' parameter specifies the number of trials to run.
study.optimize(objective, n_trials=15)

# Best Trial
# After optimization, retrieve the best trial with the highest test accuracy.
# The best hyperparameters are stored in 'best_trial.params'.
best_trial = study.best_trial
```

Ordinary Least Squares

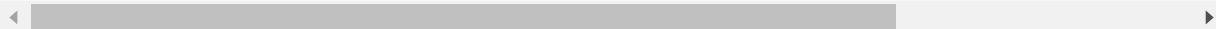
Derivation of Coefficients:

- OLS derives the coefficients by setting the partial derivatives of the SSE with respect to each coefficient to zero. This leads to a system of linear equations.
- In matrix form, the solution can be expressed as:



$$\hat{b} = (X^T X)^{-1} X^T y$$

- Here, X is the matrix of input features (including a column of ones for the intercept), y is the vector of observed values.



Orthogonalization

[link](#)

When training ML model need Orthogonalization in order to Determine what to tune, and observe the effect it has.

Each button does one thing.

Example: Car with Accelerator and angle of steering wheel.

Assumptions (controls for tuning):

- model works well with cost functions
 - Try [Adam Optimizer](#), bigger network
- work on training set
 - [Regularisation](#)
 - Try bigger training set
- works on test set of data
 - Try bigger training set.
- works well in real life.
 - Change training set
 - Change cost function.

Avoid early stopping as effects network size, and training set size.

Related links:

- [Optimising Neural Networks](#)
- [DS & ML Portal](#)

Outliers

Outliers are data points that differ significantly from other observations in the dataset. They can skew and mislead the training of machine learning models, especially those sensitive to the scale of data, such as [Linear Regression](#). They can sway the generality of the model, skewing predictions and increasing the standard deviation.

Related Concepts:

- Handling outliers in similar to [Handling Missing Data](#)

- Methods for Handling Outliers
 - Anomaly Detection
-

Why Removing Outliers May Improve Regression but Harm Classification

Impact on Regression Model:

Regression models, particularly linear regression, are sensitive to outliers because they attempt to minimize the sum of squared errors. By removing outliers, the model can better capture the underlying trend of the data, leading to improved performance metrics such as R-squared and reduced mean squared error.

Impact on Classification Models

- Class Boundary Distortion: Classification models, such as decision trees or support vector machines, rely on the distribution of data points to define class boundaries. **Outliers can provide valuable information about the variability within classes.**
- Loss of Information: Removing outliers may lead to the loss of important data points that could help in distinguishing between classes, potentially resulting in a less accurate model. For example, an outlier might represent a rare but important class that the model needs to learn from.

Over Parameterised Models

Neural network

Universal approximation theory

Overfitting

[!Summary]

Overfitting in machine learning occurs when a model captures not only the underlying patterns in the training data **but also the noise**, leading to poor performance on unseen data, and is unable to generalise.

Mathematically, overfitting results in a model with low bias but high variance, meaning it adapts too closely to the training data and fails to generalize well.

Key methods to address overfitting include **Regularisation** (such as L_1 and L_2 regularization), **Cross Validation**, and simpler models.

In statistical terms, it indicates a model with high complexity and too many parameters relative to the amount of training data, which results in $f(x)$ poorly representing the population distribution.

[!Breakdown]

Key Components:

- Regularization (Lasso: L_1 , Ridge: L_2) to penalize model complexity.
- **Cross Validation** to ensure model generalization.
- Early stopping in training to avoid learning noise.
- Simplification of models to prevent fitting irrelevant patterns.

[!important]

- Overfitting indicates high variance in the model's performance, which can be **identified by a significant drop in accuracy between training and test datasets**.
- Regularization adds penalty terms to the cost function, reducing model complexity and mitigating overfitting.

[!attention]

- Overfitting is more common in models with high-dimensional datasets.
- Excessive model tuning (hyperparameter optimization) may inadvertently increase overfitting.

[!Follow up questions]

- How does the choice of regularization type (e.g., L_1 vs. L_2) affect model generalization in overfitting scenarios?
- What role does the size of the training dataset play in mitigating overfitting?

[!Related Topics]

- **Cross Validation** techniques (e.g., k -fold, Leave-One-Out cross-validation)
- **Bias and variance** tradeoff in machine learning models

Oltp (Online Transactional Processing)

P

Table of Contents

- PCA Explained Variance Ratio
 - PCA Principal Components
 - PCA-Based Anomaly Detection
 - PCA_Analysis.ipynb
 - PCA_Based_Anomaly_Detection.py
 - PDF++
 - PDP and ICE
 - Page Rank
 - Pandas Dataframe Agent
 - Pandas Pivot Table
 - Pandas Stack
 - Pandas join vs merge
 - Pandas
 - Pandas_Common.py
 - Pandas_Stack.py
 - Parametric tests
 - Parquet
 - Part of speech tagging
 - Percentile Detection
 - Performance Dimensions
 - Performance Drift
 - Physical Model
 - Plotly
 - Poetry
 - Policy
 - Positional Encoding
 - PostgreSQL
 - PowerBI
 - PowerShell
 - Powerquery
 - Powershell versus cmd
 - Powershell vs Bash
 - Precision or Recall
 - Precision-Recall Curve
 - Precision
 - Prediction Intervals
 - Preprocessing
 - Prevention Is Better Than The Cure
 - Primary Key
 - Principal Component Analysis
 - Probability in other fields
 - Problem Definition
 - Prompt Engineering
 - Proportion Test
 - Publish and Subscribe
-

- Pull Request Template
- Push-Down
- PyCaret
- PyGraphviz
- PySpark
- PyTorch
- Pycaret_Anomaly.ipynb
- Pycaret_Example.py
- Pydantic
- Pydantic.py
- Pydantic_More.py
- Pyright vs Pydantic
- Pyright
- Pytest
- Python Click
- Python
- Pytorch vs Tensorflow
- p values
- p-values in linear regression in sklearn
- parametric vs non-parametric models
- parametric vs non-parametric tests
- parsimonious
- pd.Grouper
- pdoc
- pmdarima
- programming languages

Pca Explained Variance Ratio

PCA Explained Variance Ratio

- The variance explained by each principal component is printed using `pca.explained_variance_ratio_`.
- The sum of the explained variances is calculated and printed, which should ideally equal 1 (indicating that all variance in the data is accounted for by the principal components).

The **explained variance** refers to how much of the total variance in the original dataset is captured or explained by each principal component (PC) in Principal Component Analysis (PCA).

In the context of PCA:

1. **Explained Variance:** After performing PCA, each principal component corresponds to a certain amount of variance in the original data. The **explained variance** of a principal component is the proportion of the total variance in the original dataset that is captured by that component.
2. **Explained Variance Ratio:** This is the proportion of the total variance explained by each principal component. It is calculated as:

$$\text{Explained Variance Ratio of PC}_i = \frac{\text{Variance explained by PC}_i}{\text{Total variance of the dataset}}$$

It tells us how much of the total variance in the data is accounted for by each component. For example, if the first principal component explains 70% of the total variance, then the explained variance ratio for that component would be 0.70.

3. **Cumulative Explained Variance:** If we sum up the explained variance ratios of the first few principal components, we can assess how many components are needed to explain a significant portion of the total variance. For example, if the first two components explain 90% of the variance, it means that we can reduce the dataset's dimensionality by keeping just those two components without losing much information.

Example:

In PCA applied to the Iris dataset:

```
pca.explained_variance_ratio_
```

This might return something like:

```
[0.924, 0.053, 0.018, 0.005]
```

This means:

- The first principal component explains 92.4% of the total variance.
- The second principal component explains 5.3% of the variance.
- The third and fourth components explain very little (1.8% and 0.5%, respectively).

Summing these ratios gives the total variance explained by the first few components. In this case, the first two components explain over 97% of the variance in the dataset, meaning the remaining components contribute very little additional information.

Why it matters:

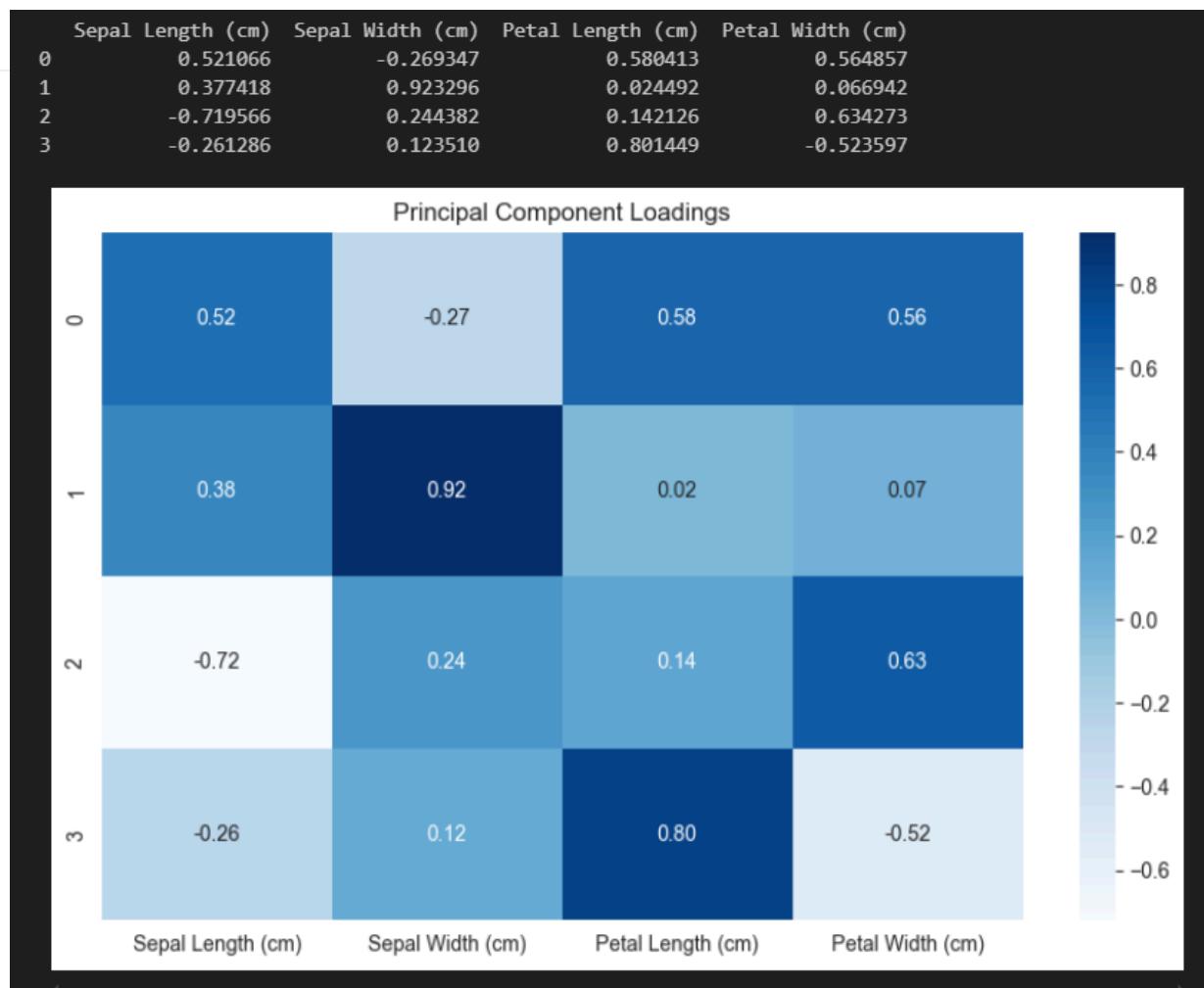
- **Dimensionality Reduction:** PCA is used to reduce the number of dimensions (features) in the dataset. The goal is to retain as much variance as possible while reducing the number of dimensions. By selecting the components with the highest explained variance, we can reduce the dataset's complexity without sacrificing much information.
- **Data Interpretability:** The explained variance helps us understand how important each component is in representing the dataset. If the first few components explain most of the variance, we can focus on them for analysis and modeling.

Pca Principal Components

The principal components (or the new axes that explain the most variance) are stored in `pca.components_` and displayed as a DataFrame for easier reading

Interpreting

See [PCA_Analysis.ipynb](#)



How to Interpret the PCA Heatmap

This heatmap represents the principal component loadings, which show how strongly each original feature contributes to each principal component (PC).

Understanding the Heatmap Content

- Rows = Principal Components (PCs)
 - Each row corresponds to a principal component (e.g., PC1, PC2, PC3, etc.).
 - PC1 captures the most variance, PC2 captures the second-most, and so on.
- Columns = Original Features
 - Each column represents an original feature from the dataset (e.g., `sepal length`, `sepal width`, etc.).
- Cell Values = Loadings
 - Each cell contains a loading coefficient, which tells us how much that feature contributes to the corresponding principal component.
 - The values range from -1 to 1: Close to 1 → The feature strongly contributes to that PC in the positive direction, etc.

Key Insights from the Heatmap

- Which features are most important for each PC?

Introduction

- Look for the largest absolute values in each row.
- Example: If `sepal length` has a high positive value in PC1, it means PC1 is largely influenced by `sepal length`.
- Feature Groupings & Correlations
 - Features with similar values in a PC vary together in the data.
 - Example: If `sepal length` and `sepal width` have similar values in PC1, they might be [Correlation](#) correlated in the dataset.
- Interpreting the First Few Principal Components
 - PC1 often represents the main pattern in the data (e.g., overall size of the iris flowers).
 - PC2 might represent a different pattern (e.g., a contrast between petal and sepal size).
 - Together, PC1 and PC2 often explain the majority of variance in the dataset.

Example Interpretation (Hypothetical Output)

Sepal Length	Sepal Width	Petal Length	Petal Width	
PC1	0.70	-0.40	0.85	0.75
PC2	-0.60	0.80	-0.35	-0.45

PC1 Interpretation: `Petal length` and `sepal length` have high positive loadings, meaning PC1 mainly captures flower size. `Sepal width` has a negative loading, meaning flowers with large sepals tend to have smaller widths.

PC2 Interpretation: `Sepal width` has the highest positive loading, while `sepal length` has a negative loading, suggesting that PC2 contrasts width vs. length.

How to Use This Information?

[Feature Selection](#): If one PC captures most of the variance, we can reduce dimensionality and keep only the most important PCs.

Pca Based Anomaly Detection

For implementation, see: [ML_Tools](#):

- [PCA_Based_Anomaly_Detection.py](#)

Pca_Analysis.ipynb

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/PCA/PCA_Analysis.ipynb

This script performs Principal Component Analysis (PCA) on the Iris dataset to reduce its dimensionality while preserving key variance.

See also [Principal Component Analysis|PCA](#)

Summary:

1. Load and Preprocess Data

- Loads the Iris dataset and extracts features and target labels.

Introduction

- Scales the data to standardize feature ranges.

2. Apply PCA (3 Components)

- Fits PCA to the scaled data and transforms it into three principal components.
- Stores the transformed data in a DataFrame with species labels.

3. Analyze PCA Loadings & Variance

- Computes and stores PCA loadings (weights of original features in principal components).
- Computes explained variance and cumulative variance to assess PCA effectiveness.

4. Visualizations

- Explained variance: Bar plot of individual and cumulative variance contributions.
- PCA Scores: 3D scatter plots of transformed data, colored by species.
- PCA Loadings: 3D scatter plot showing feature contributions to principal components.
- Heatmap: Displays PCA component weights for feature importance analysis.

5. Additional Full PCA Analysis

- Computes and prints explained variance for all components.
- Uses Seaborn to generate a heatmap of PCA component contributions.

Pca_Based_Anomaly_Detection.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/PCA/PCA_Based_Anomaly_Detection.py

Pdf++

<https://www.youtube.com/watch?v=4dU6WXULSqq>

Pdp And Ice

link:

https://scikit-learn.org/1.5/modules/partial_dependence.html#h2009

[interpretability|interpretable](#)

Regression example:

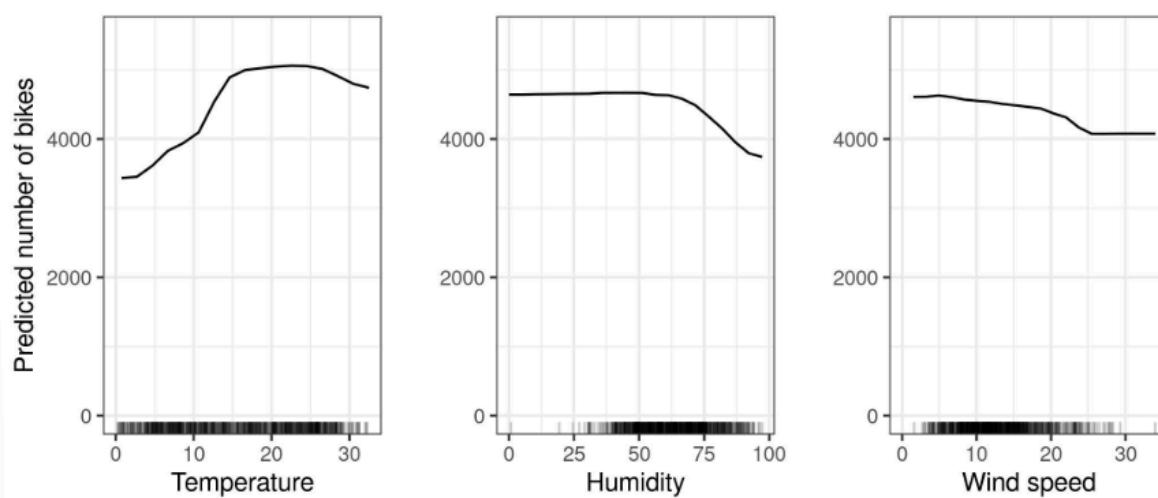


FIGURE 8.1: PDPs for the bicycle count prediction model and temperature, humidity and wind speed. The largest differences can be seen in the temperature. The hotter, the more bikes are rented. This trend goes up to 20 degrees Celsius, then flattens and drops slightly at 30. Marks on the x-axis indicate the data distribution.

Categorical Example

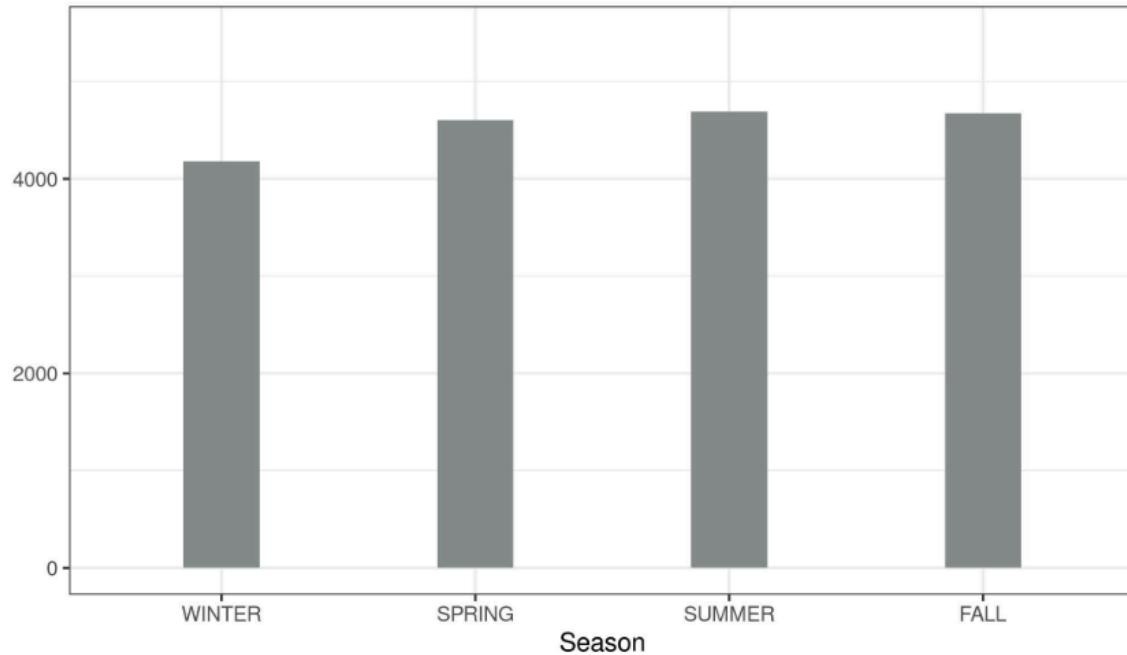


FIGURE 8.2: PDPs for the bike count prediction model and the season. Unexpectedly all seasons show similar effect on the model predictions, only for winter the model predicts fewer bicycle rentals.

Page Rank

PageRank is an algorithm originally developed by Larry Page and Sergey Brin (founders of Google) to rank web pages in search engine results. It measures the relative importance of each node (e.g., webpage) in a directed graph based on the structure of incoming links.

Intuition

The core idea is:

A node is important if many other important nodes link to it.

PageRank simulates a “random surfer” who clicks on links at random:

- With probability d (typically 0.85), the surfer follows a link from the current page.
- With probability $1 - d$, the surfer jumps to a random page.

Mathematical Formulation

Given a graph with N nodes, the PageRank of node i is defined recursively as:

$$PR(i) = \frac{1-d}{N} + d \sum_{j \in \text{In}(i)} \frac{PR(j)}{L(j)}$$

Where:

- d is the damping factor (usually 0.85),
- $\text{In}(i)$ is the set of nodes linking to i ,
- $L(j)$ is the number of outbound links from node j .

Implementation (using NetworkX)

```
import networkx as nx

G = nx.DiGraph()
G.add_edges_from([
    ('A', 'B'), ('B', 'C'), ('C', 'A'), ('A', 'D'), ('D', 'C')
])

pagerank_scores = nx.pagerank(G, alpha=0.85)
for node, score in pagerank_scores.items():
    print(f"{node}: {score:.4f}")
```

Use Cases

- Graph-based NLP: Keyword extraction (e.g., TextRank).

graph #data_visualization

Pandas Dataframe Agent

Example: <https://github.com/AssemblyAI/youtube-tutorials/tree/main/pandas-dataframe-agent>

Follow:

https://www.youtube.com/watch?v=ZlfzpmO8MdA&list=PLcWfeUsAys2kC31F4_ED1JXIkdmu6tlrm&index=7

Can ask pandas questions to a dataframe.

Types of questions:

- what is the max value of "col1"

Pandas Pivot Table

Pivot Table: Summarize Data

```
df = pd.DataFrame({'A': ['foo', 'foo', 'bar'], 'B': ['one', 'two', 'one'], 'C': [1, 2, 3]})  
pivot_table = df.pivot_table(values='C', index='A', columns='B', aggfunc='sum')
```

Relevant links:

- https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/pivot_table.ipynb

Pandas Stack

Tool for reshaping data, particularly when you need to pivot a DataFrame ([Pandas Pivot Table](#)) from a wide format to a long format.

See:

- [Pandas_Common.py](#)
- [Pandas_Stack.py](#)

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/reshaping.ipynb

Why Use stack ? ([Data Transformation](#))

Data Reshaping:

- **Wide to Long Format:** Convert a DataFrame from a wide format to a long format, which is often preferred for statistical models and visualizations.

Handling Multi-Index DataFrames:

Introduction

- Simplifying Structure: Move the inner level of a column MultiIndex to the row index, simplifying the DataFrame's structure.

Data Cleansing:

- Aggregation and Operations: Facilitate data cleaning by allowing aggregation or operations across columns in a more manageable long format.

Preparing Data for Grouping or Aggregation ([Groupby](#)):

- Ease of Grouping: Simplify group-by operations and aggregations on data with columns representing different categories or time periods.

Example of Using `stack`

Consider the following example to illustrate how `stack` works:

```
import pandas as pd

# Sample DataFrame
data = {
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

# Using stack
stacked_df = df.stack()
print("\nStacked DataFrame:")
print(stacked_df)
```

Output:

```
Original DataFrame:
```

```
   A  B  C  
0  1  4  7  
1  2  5  8  
2  3  6  9
```

```
Stacked DataFrame:
```

```
  0  A    1  
    B    4  
    C    7  
  1  A    2  
    B    5  
    C    8  
  2  A    3  
    B    6  
    C    9  
dtype: int64
```

In this example:

- The original DataFrame has three columns ('A', 'B', 'C') and three rows.
- After stacking, the DataFrame is transformed into a Series with a MultiIndex/[Multi-level index](#). The outer level of the index corresponds to the original DataFrame's row index, and the inner level corresponds to the original column labels.

When Not to Use `stack`

- Wide Format Requirements: If your analysis or processing requires a wide format, such as some [Machine Learning Algorithms](#), stacking may not be appropriate.
- Complexity: If stacking makes the data too complex to manage or understand, it might be better to keep the original structure.
- Simplicity: When the current structure of your DataFrame already suits your analysis needs, stacking may be unnecessary.

Pandas Join Vs Merge

In pandas, both `.join()` and `pd.merge()` are used to combine DataFrames, but they differ in **syntax**, **defaults**, and **use cases**.

[Merge](#) is better than Join.

Feature	<code>df.join()</code>	<code>pd.merge()</code>
Default key	Uses index of caller and index/column of other	Requires explicit column(s) to merge on
Syntax style	Method on a DataFrame	Function (<code>pd.merge(left, right)</code>)
Column join	Must specify <code>on=</code> and use one column from each	Can merge on multiple columns
Index join	Default behavior (index-to-index)	Requires <code>left_index=True</code> , <code>right_index=True</code>
Suffixes	<code>lsuffix</code> , <code>rsuffix</code>	<code>suffixes=('_x', '_y')</code>
Complex joins	Not well-suited	Supports full SQL-style joins
Use case	Simple joins on index or one column	Complex joins with control over join behavior

data_cleaning #data_integration

Pandas

In [ML_Tools](#) see:

- [Pandas_Common.py](#)

Areas:

- [Handling Missing Data](#)
- [Data Selection](#)
- [Data Transformation](#)

Pandas_Common.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Pandas_Common.py

Pandas_Stack.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Pandas_Stack.py

Parametric Tests

Parquet

A **Parquet file** is a **columnar storage file format** specifically designed for storing large amounts of data efficiently. It is commonly used in [Big Data](#) ecosystems due to its optimised performance for both storage and querying.

[Data Storage](#)

Key Features of Parquet:

1. Columnar Storage Format:

- Data is stored **column by column** instead of row by row.
- This design is highly efficient for analytical queries that access only specific columns, reducing the amount of data read.

2. Optimised for Big Data:

- Parquet is designed for distributed systems like Apache [Hadoop](#), [Apache Spark](#), and other big data processing tools.

3. Compression:

- It supports multiple compression algorithms (e.g., Snappy, GZIP) for reducing file size while maintaining fast read and write performance.

4. Schema Evolution:

- Parquet supports flexible schemas, allowing fields to be added or modified without breaking compatibility with older data.

5. Efficient Metadata Handling:

- Metadata is stored along with the data, making it easier to retrieve and query information about the dataset without scanning the entire file.

Advantages of Parquet:

1. Improved Query Performance:

- Since data is stored column-wise, queries that require only a few columns read less data compared to row-based formats like CSV.

2. Lower Storage Costs:

- Built-in compression and columnar storage **significantly reduce file size**.

3. Compatibility:

- Parquet is compatible with most big data processing tools, such as Hadoop, Spark, Hive, Presto, and AWS Athena.

4. Efficient I/O:

- Parquet's columnar format minimises disk I/O, making it faster to process large datasets.

When to Use Parquet:

- **Analytical Workloads:** Ideal for scenarios where you need to perform aggregations, filtering, or processing large datasets.
- **Big Data Processing:** Frequently used with tools like Spark, Hive, and Presto in data lakes.
- **Cloud Storage:** Supported by [Cloud Providers](#) platforms.

Example Use Case:

Imagine a dataset with 10 million rows and 100 columns.

- If you query just 3 columns in a row-based format (e.g., CSV), you must read all 100 columns for every row.
- In Parquet, only the 3 relevant columns are read, significantly improving performance.

File Structure:

- **Row Group:** Data is divided into chunks called row groups, enabling efficient data retrieval.
- **Columns:** Each column in a row group is stored together for fast access.
- **Footer:** Contains metadata, such as schema definitions and row group locations, allowing quick navigation of the file.

How to Work with Parquet in Python:

You can use libraries like **pandas** or **pyarrow**:

```
import pandas as pd

# Write a DataFrame to a Parquet file
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df.to_parquet('example.parquet', engine='pyarrow')

# Read a Parquet file into a DataFrame
df_read = pd.read_parquet('example.parquet')
print(df_read)
```

In summary, Parquet is an efficient, compact, and scalable file format ideal for big data analytics and storage, providing faster performance and reduced costs.

Part Of Speech Tagging

Part of speech tagging : assigning a specific part-of-speech category (such as noun, verb, adjective, etc.) to each word in a text

Part-of-speech tagging involves assigning a specific part-of-speech category (such as noun, verb, adjective, etc.) to each word in a text

```
from nltk import pos_tag  
pos_tag(temp[:20])
```

will get outputs such as [('history', 'NN'), ('poland', 'NN'), ('roots', 'NNS'), ('early', 'JJ').

Percentile Detection

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Outliers/outliers_percentile.py

Performance Dimensions

Efficiency & Performance

- **Cost-efficiency:** Ensuring that the solutions used are cost-effective and provide value for money.
- **Speed:** The ability to process and analyze data quickly to meet business needs.
- **Performance:** Optimizing query performance by organizing data in a way that supports efficient retrieval.

Flexibility & Scalability

- **Flexibility:** The capability to adapt to changing requirements and data sources.
- **Scalability:** Ensuring the system can handle increasing volumes of data without performance degradation. Can handle large volumes of data, making it suitable for enterprise-level data warehousing.
- **Reusability:** Designing components that can be reused across different projects or processes.
- **Latency:** Minimizing delays in data processing and retrieval.

Usability & Accessibility

- **Simplicity:** Keeping the system easy to manage and understand, reducing complexity.
- **Usability:** Providing a clear and intuitive structure that is easy for business users to understand and navigate.
- **Accessibility:** Ensuring that data is accessible when needed by authorized users.

Data Quality & Integrity

- **Data Integrity:** Ensuring data accuracy and consistency.
- **Data Quality:** Maintaining high-quality data for reliable insights.
- **Availability:** Ensuring that data is available when required by authorized users.

Interoperability

- **Interoperability:** Ensuring that different systems and tools can work together seamlessly.
- **Data Compatibility:** Ensuring data is compatible with different systems.

Performance Drift

Not Data Drift

TL;DR. Data drift is a change in the input data. Concept drift is a change in input-output relationships. Both often happen simultaneously.

Performance drift refers to the **gradual decline in a machine learning model's accuracy** or effectiveness over time as the underlying data distribution changes.

Introduction

This phenomenon occurs when the real-world data that the model is applied to differs from the data it was trained on. Mathematically, this is often represented by a shift in the joint distribution $P(X, Y)$ of the features X and target variable Y .

Performance drift can occur due to **concept drift** (when the relationship between inputs and outputs changes) or **covariate shift** (when the distribution of the inputs changes). The model's prediction error increases, leading to suboptimal decisions or predictions.

Key Components:

- **Concept drift:** Changes in the relationship between inputs and outputs, $P(Y|X)$.
- **Covariate shift:** Change in the input data distribution, $P(X)$.
- **Model monitoring Data Observability/monitoring:** Continuous assessment of a model's accuracy over time to detect drift.
- **Retraining:** Updating the model with new data to restore performance.

Important

- Performance drift results from data distribution shifts, leading to increased prediction errors.
- Monitoring and retraining are key strategies to address performance drift in real-world applications.
- A lack of continuous monitoring can result in undetected **model performance degradation**.
- Overfitting a model to the original data without considering future data can accelerate drift.

Example In a credit scoring model, performance drift may occur if consumer spending habits change due to an economic recession. The model trained on pre-recession data will perform poorly on post-recession data as the input patterns ($P(X)$) and the relationship between inputs and outputs ($P(Y|X)$) shift.

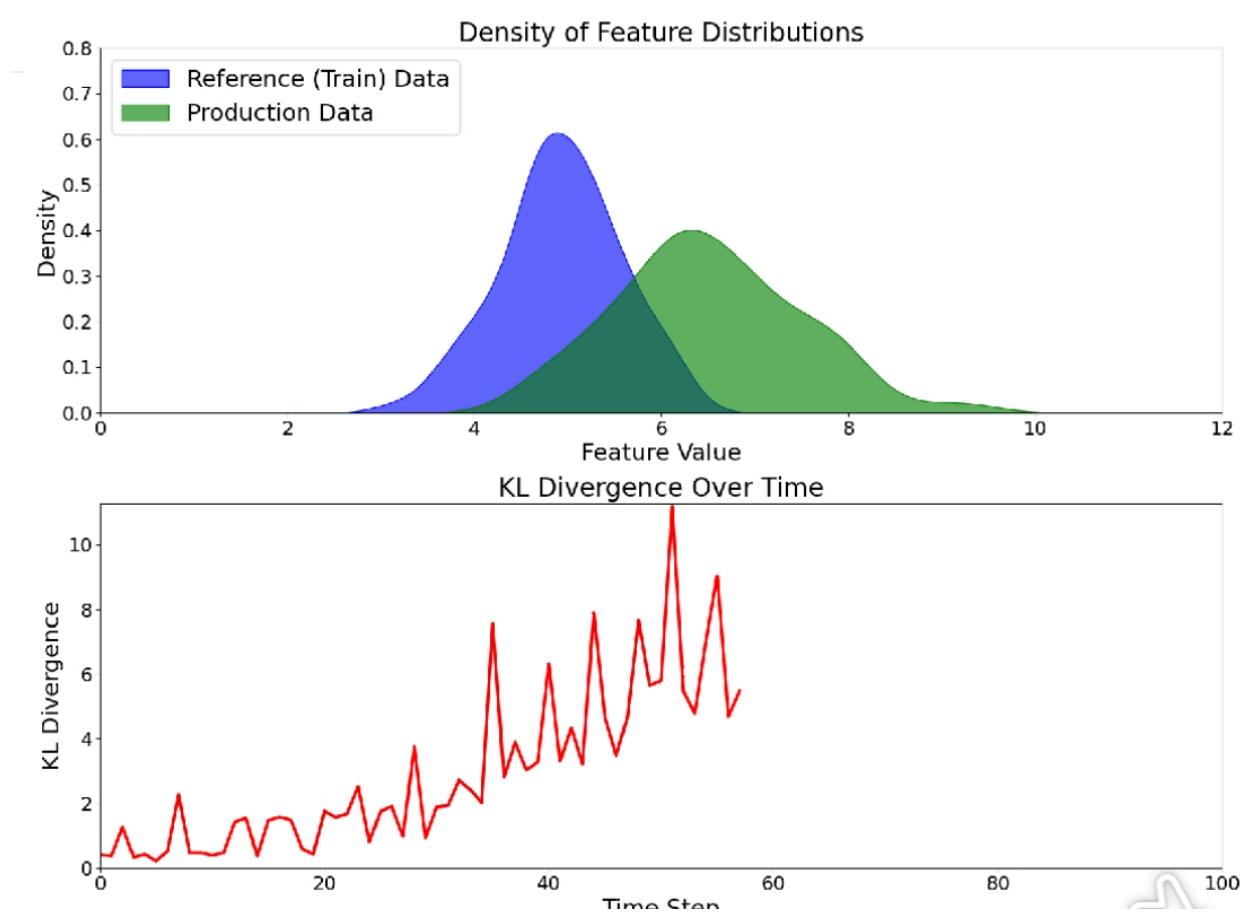
Questions

- How can adaptive learning techniques help mitigate the effects of performance drift?
- What statistical methods can be used to detect early signs of concept drift in production models?

Related Topics

- Model retraining strategies

Images



Physical Model

Physical Model

(for a SQL database):

```

CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);

CREATE TABLE Order (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);

CREATE TABLE Book (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(100)
);

CREATE TABLE OrderBook (
    OrderID INT,
    BookID INT,
    PRIMARY KEY (OrderID, BookID),
    FOREIGN KEY (OrderID) REFERENCES Order(OrderID),
    FOREIGN KEY (BookID) REFERENCES Book(BookID)
);

```

Physical Model

- Implements the logical model in a specific database system.
- Includes table structures, columns, data types, and constraints.

Plotly

Poetry

Modern version of setting up dependencies instead of requirements.txt ([dependency manager](#))

Primary Purpose: Poetry is a [dependency manager](#) and packaging tool for Python projects.

Main Features:

- Dependency management: Allows you to declare, install, and lock dependencies in the `pyproject.toml` file.
- Package management: Can help package your Python project for distribution (e.g., publishing to PyPI).
- Virtual environment management: Poetry automatically creates and manages a virtual environment for your project.
- Version management: Ensures that all your project dependencies use compatible versions through its `poetry.lock` file, s
- Built-in publishing: Simplifies the process of publishing your Python package to PyPI.

Ideal Use Case:

- If you need to manage dependencies for a Python project, create virtual environments, and ensure reproducibility (using Poetry).
- If you're developing a Python package that you want to distribute or manage versions for, Poetry is a great choice.

Use:

```
pip install poetry
poetry init
poetry add numpy
```

Policy

In [reinforcement learning \(RL\)](#), a **policy** is a strategy or a rule that defines the actions an agent takes in a given state to achieve its goals. It essentially [maps states of the environment to actions that the agent should take when in those states](#).

Policies are used in RL as they determine the behavior of the agent.

The goal of many RL algorithms is to find an optimal policy that maximizes the cumulative reward the agent receives over time. This involves balancing exploration (trying new actions) and exploitation (using known actions that yield high rewards).

Key Concepts

On-Policy vs. Off-Policy:

- **On-Policy:** The agent learns the value of the policy it is currently following. An example is the [SARSA](#) algorithm, which updates its policy based on the actual actions taken by the agent.
- **Off-Policy:** The agent learns the value of the optimal policy, regardless of the actions it actually takes. [Q-Learning](#) is an example of an off-policy algorithm, as it updates its policy based on the best possible action in the next state, not necessarily the action taken.

Conservatism:

- Some policies, like those in SARSA, are more conservative in their updates. This means they are more cautious and adapt to uncertainties in the environment, making them suitable for environments where [exploration](#) and [exploitation](#) need to be balanced carefully.

Example of a Policy

Consider a simple grid world where an agent can move up, down, left, or right to reach a goal. A policy in this context could be a set of rules that tells the agent to always move towards the goal if it is visible, or to explore randomly if the goal is not visible.

For instance, in a 3x3 grid where the goal is at position (2, 2), a simple policy might be:

- If the agent is at (0, 0), move right.
- If the agent is at (0, 1), move right.
- If the agent is at (0, 2), move down.
- Continue this pattern until the agent reaches the goal at (2, 2).

Introduction

This policy can be represented as a table or a function that maps each state (grid position) to an action (move direction).

State (Position)	Action
(0, 0)	Move Right
(0, 1)	Move Right
(0, 2)	Move Down
(1, 0)	Move Right
(1, 1)	Move Right
(1, 2)	Move Down
(2, 0)	Move Right
(2, 1)	Move Right
(2, 2)	Goal Reached

Positional Encoding

Postgresql

Installation

[How to set up a Postgres database on your Windows 10 PC](#)

In [Tableau](#) can connect to a database here.

There are plugins.

Spatial objects?

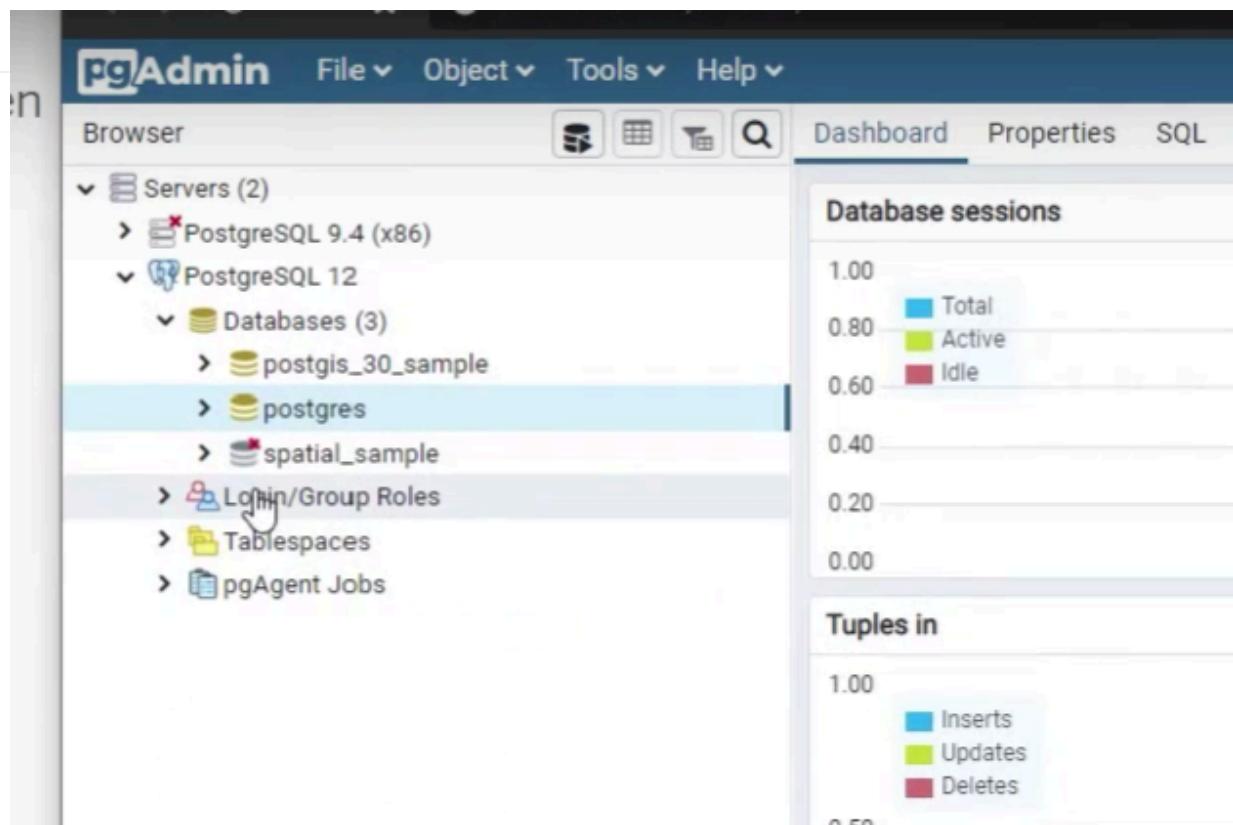
Connections to database tables: hosted

Connecting

[Adding a database to PostgreSQL](#)

PGAdmin

pgadmin tools: to check system information



Powerbi

[tutorial](#)

Business analytics tool for data visualization and reporting.

Powershell

Why is Powershell better than [Command Prompt|cmd](#)?

PowerShell is often considered better than Command Prompt (cmd) for several reasons:

1. **Object-Oriented:** PowerShell is built on the .NET framework and works with objects rather than plain text. This allows for more complex data manipulation and easier handling of outputs.
2. **Powerful Scripting Capabilities:** PowerShell supports advanced scripting features, including functions, loops, and error handling, making it more suitable for automation and complex tasks.
3. **Access to .NET Framework:** PowerShell can leverage the full power of the .NET framework, allowing users to utilize a vast array of libraries and functionalities.
4. **Cmdlets:** PowerShell uses cmdlets, which are specialized .NET classes designed to perform specific functions. This makes it easier to perform tasks compared to the simpler commands in cmd.
5. **Remote Management:** PowerShell has built-in capabilities for remote management, allowing users to manage multiple systems from a single console.
6. **Pipeline Support:** PowerShell allows for the use of pipelines to pass objects between cmdlets, enabling more efficient and powerful command chaining.

7. **Integrated Help System:** PowerShell includes a robust help system that can be accessed directly from the command line, making it easier to learn and use.
8. **Cross-Platform:** PowerShell Core (now known as PowerShell 7) is cross-platform, meaning it can run on Windows, macOS, and Linux, unlike cmd, which is Windows-only.

Scripts

PowerShell interacts with several types of scripts and scripting languages, including:

1. **PowerShell Scripts (.ps1):** These are the primary script files used in PowerShell. They contain a series of PowerShell commands and can automate tasks.
2. **Batch Files (or .cmd):** PowerShell can execute traditional Windows batch files, allowing for integration with legacy scripts.
3. **VBScript (.vbs):** PowerShell can run VBScript files, which can be useful for interacting with older systems or applications that rely on VBScript.
4. **Windows Management Instrumentation (WMI):** PowerShell can interact with WMI scripts to manage and monitor system resources.
5. **.NET Scripts:** Since PowerShell is built on the .NET framework, it can execute .NET code and interact with .NET assemblies.
6. **Python and Other Scripting Languages:** PowerShell can call scripts written in other languages (like Python) using the appropriate command-line interfaces.
7. **JSON and XML:** PowerShell can parse and manipulate JSON and XML data, which are often used in configuration files and data exchange.

Powerquery

[How to normalise a merged table](#)

Powershell Versus Cmd

PowerShell and Command Prompt (cmd) are both command-line interfaces available on Windows systems, but they differ significantly in their capabilities, syntax, and scripting abilities. Here are the key differences and examples that highlight their distinct features:

PowerShell:

1. **Object-Oriented Shell:**
 - o PowerShell is designed around the concept of objects rather than text streams like cmd. This makes it more powerful for scripting and automation tasks.
 - o **Example:** Getting detailed information about files:

```
Get-ChildItem | Select-Object Name, Length, LastWriteTime
```

This command retrieves file objects and selects specific properties (Name, Length, LastWriteTime).

2. Extensive Commandlets (Cmdlets):

- PowerShell includes a wide range of cmdlets for performing specific tasks, such as managing Active Directory, working with files, or interacting with web services.
- **Example:** Restarting a service:

```
Restart-Service -Name "serviceName"
```

This cmdlet restarts a service named "serviceName".

3. Advanced Scripting and Automation:

- PowerShell supports advanced scripting features, including loops, conditional statements, functions, and error handling.
- **Example:** Checking if a file exists:

```
if (Test-Path -Path "C:\path\to\file.txt") {  
    Write-Output "File exists."  
} else {  
    Write-Output "File does not exist."  
}
```

This script uses `Test-Path` cmdlet to check if a file exists and then outputs a message accordingly.

4. Integration with .NET Framework:

- PowerShell can leverage .NET Framework libraries and assemblies directly within scripts.
- **Example:** Using .NET Framework classes: `powershell

```
[System.IO.File]::ReadAllText("C:\path\to\file.txt")  
...  
This line reads the entire content of a text file using .NET Framework's `File` class.
```

Command Prompt (cmd):

1. Text-Based Command Line:

- Cmd operates primarily with text-based commands and outputs, lacking PowerShell's object-oriented approach.
- **Example:** Listing files in a directory:

```
dir /b
```

This command lists all files in the current directory in a bare format.

2. Limited Built-in Commands:

- Cmd has a more limited set of built-in commands (compared to PowerShell's cmdlets), focusing on basic system commands.
- **Example:** Copying files:

```
copy C:\source\file.txt D:\destination\
```

This command copies a file from `C:\source` to `D:\destination`.

3. Batch Scripting:

- Cmd uses batch files (.bat) for scripting, which are simpler and less flexible compared to PowerShell scripts.
- **Example:** Simple batch file to copy files:

```
@echo off  
copy C:\source\file.txt D:\destination\
```

This batch script copies a file without displaying command prompt output.

4. Direct Command Execution:

- Commands in cmd are executed directly without the pipeline and object manipulation features of PowerShell.
- **Example:** Renaming a file:

```
ren oldfile.txt newfile.txt
```

This command renames a file from `oldfile.txt` to `newfile.txt`.

Choosing Between PowerShell and cmd:

- **Use PowerShell When:**
 - You need to work with complex data structures or objects.
 - Automation and scripting tasks require advanced features like loops, conditions, and error handling.
 - Integration with .NET Framework or other external libraries is necessary.
- **Use Command Prompt (cmd) When:**
 - Performing basic system tasks or operations.
 - Working with simple text-based outputs.
 - Using legacy batch scripts or when PowerShell is unavailable.

In summary, PowerShell offers a more versatile and powerful environment for scripting, automation, and administrative tasks on Windows systems, while cmd remains useful for straightforward commands and basic system interactions.

Powershell Vs Bash

The choice between [PowerShell](#) and [Bash](#) largely depends on the user's needs and the environment in which they are working. Here are some considerations for each:

PowerShell

- **Windows Integration:** PowerShell is deeply integrated with Windows and is ideal for managing Windows systems and applications.
- **Object-Oriented:** As mentioned earlier, PowerShell works with objects, making it easier to manipulate data and interact with .NET applications.
- **Remote Management:** It has strong capabilities for remote management of Windows systems.
- **Scripting:** PowerShell's scripting capabilities are robust, making it suitable for complex automation tasks.

Bash

- **Unix/Linux Environment:** Bash is the default shell for many Unix and Linux systems, making it the go-to choice for system administrators and developers in those environments.
- **Simplicity:** Bash scripts are often simpler and more straightforward for basic tasks, especially for file manipulation and text processing.
- **Community and Resources:** There is a vast amount of community support, tutorials, and resources available for Bash, especially in the open-source community.
- **Cross-Platform:** While traditionally associated with Unix/Linux, Bash can also be used on Windows through [Windows Subsystem for Linux \(WSL\)](#) or [Git Bash](#).

Precision Or Recall

[Precision](#) and [Recall](#) are two fundamental metrics used to evaluate the performance of a [Classification](#) model, particularly in binary classification tasks. They are related through a trade-off: improving one often comes at the expense of the other.

Key Differences:

- [Precision](#) focuses on the quality of positive predictions.
- [Recall](#) focuses on the ability to identify all relevant positive instances

Trade-off:

- It is challenging to optimize both precision and recall simultaneously. Improving precision by reducing false positives may lead to an increase in false negatives, thereby reducing recall, and vice versa.
- However, it's important to balance recall with precision. A model with high recall might also have a higher rate of false positives (non-spam emails incorrectly marked as spam), which can lead to important emails being missed.

Task Dependency Example:

- The choice between prioritizing precision or recall is task-dependent. In a spam classification task, it might be more important to avoid moving important emails to the spam folder (high precision) than to occasionally allow spam emails into the inbox (lower recall). Thus, precision is prioritized over recall in this context.

Scenario A High Precision, Low Recall	Scenario B High Recall, Low Precision																														
<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2"></th> <th style="text-align: center;">Prediction</th> <th></th> </tr> <tr> <th colspan="2"></th> <th style="text-align: center;">0</th> <th style="text-align: center;">1</th> </tr> <tr> <th rowspan="2" style="writing-mode: vertical-rl; transform: rotate(180deg);">Actual</th> <th style="writing-mode: vertical-rl; transform: rotate(180deg);">0</th> <td style="text-align: center;">TN 14</td> <td style="text-align: center;">FP 1</td> </tr> </thead> <tbody> <tr> <th style="writing-mode: vertical-rl; transform: rotate(180deg);">1</th> <td style="text-align: center;">FN 4</td> <td style="text-align: center;">TP 1</td> </tr> </tbody> </table> <p>More spam emails went undetected → 4</p>			Prediction				0	1	Actual	0	TN 14	FP 1	1	FN 4	TP 1	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2"></th> <th style="text-align: center;">Prediction</th> <th></th> </tr> <tr> <th colspan="2"></th> <th style="text-align: center;">0</th> <th style="text-align: center;">1</th> </tr> <tr> <th rowspan="2" style="writing-mode: vertical-rl; transform: rotate(180deg);">Actual</th> <th style="writing-mode: vertical-rl; transform: rotate(180deg);">0</th> <td style="text-align: center;">TN 14</td> <td style="text-align: center;">FP 4</td> </tr> </thead> <tbody> <tr> <th style="writing-mode: vertical-rl; transform: rotate(180deg);">1</th> <td style="text-align: center;">FN 1</td> <td style="text-align: center;">TP 1</td> </tr> </tbody> </table> <p>More non-spam emails flagged as spam ← 4</p>			Prediction				0	1	Actual	0	TN 14	FP 4	1	FN 1	TP 1
		Prediction																													
		0	1																												
Actual	0	TN 14	FP 1																												
	1	FN 4	TP 1																												
		Prediction																													
		0	1																												
Actual	0	TN 14	FP 4																												
	1	FN 1	TP 1																												
Precision: 50% Recall: 20%	Precision: 20% Recall: 50%																														

Precision Recall Curve

A [precision-recall](#) curve is a graphical representation used to evaluate the performance of a [Binary Classification](#) model, particularly in scenarios where the classes are imbalanced. It plots [precision](#) (the positive predictive value) against [recall](#) (the true positive rate) for different threshold values.

Overall, precision-recall curves are a valuable tool for assessing the tradeoffs between precision and recall, helping to choose the optimal threshold for classification based on the specific requirements of the task.

Resources

In [ML_Tools](#) see: [ROC_PR_Example.py](#)

[Sklearn Link](#)

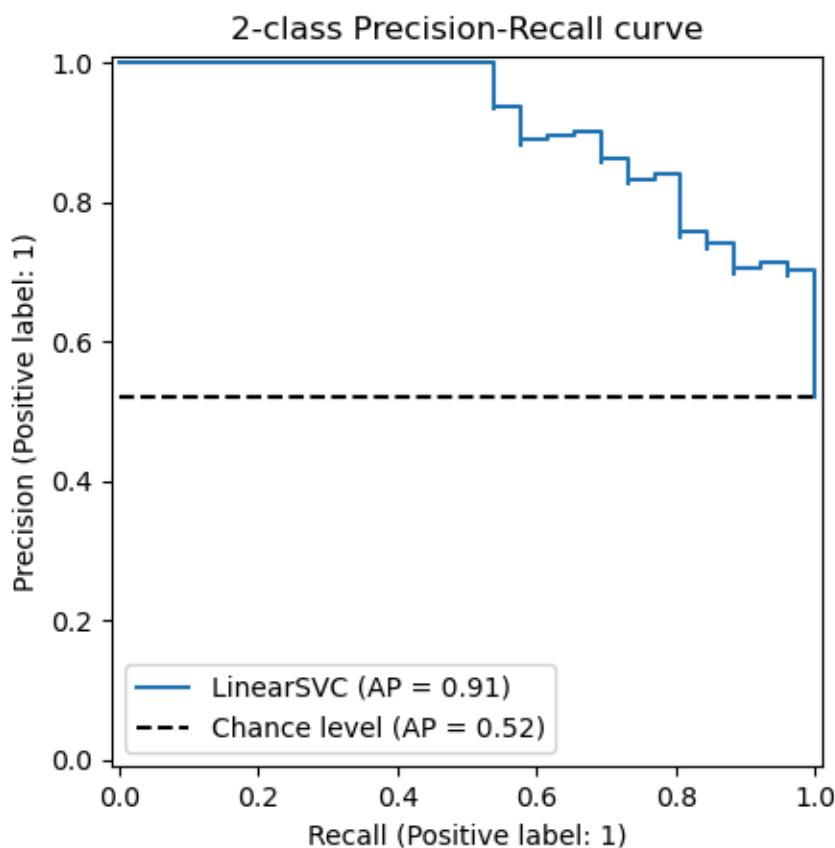
Precision Recall Curve:

Plot: The curve is generated by varying the threshold for classifying a positive instance and plotting the corresponding precision and recall values. Each point on the curve represents a precision recall pair at a specific threshold.

Interpretation:

- Be cautious of the class [Distributions](#) impact on the curve's shape. A curve that appears favorable might still represent poor performance if the positive class is very rare.
- A steep precision-recall curve indicates that the model maintains high precision across a range of recall values, which is desirable. Conversely, a more gradual curve might suggest a trade-off between precision and recall.
- A model with high precision and high recall is considered to perform well. However, there is often a tradeoff between [Precision or Recall](#).
- The area under the precision-recall curve (not the same as [AUC](#)) is a single scalar value that summarizes the performance of the model. A higher AUCPR indicates better model performance.

Use Cases: precision-recall curves are particularly useful in situations where the positive class is rare or when the cost of false positives and false negatives is different. They provide more insight than [ROC \(Receiver Operating Characteristic\)](#) curves in these scenarios because they focus on the performance of the positive class.



Other features

Multi-Class Scenarios

- **Adapting for Multi-Class Problems:** Precision-recall curves can be extended to multi-class classification problems by using strategies like one-vs-rest (OvR). In this approach, a separate precision-recall curve is computed for each class, treating it as the positive class while considering all other classes as negative.

Comparison with [ROC \(Receiver Operating Characteristic\)](#)

- **When to Use Precision-Recall Curves:** Precision-recall curves are particularly useful over ROC curves in scenarios with highly [imbalanced datasets](#). They provide a more informative picture of a model's performance by focusing on the positive class, which is often the minority class in imbalanced datasets.

Precision

Precision Score is a metric used to evaluate the [Accuracy](#) of a [Classification](#) model, specifically focusing on the positive class.

How many retrieved items are relevant?

This metric indicates the accuracy of positive predictions. The formula for precision is:

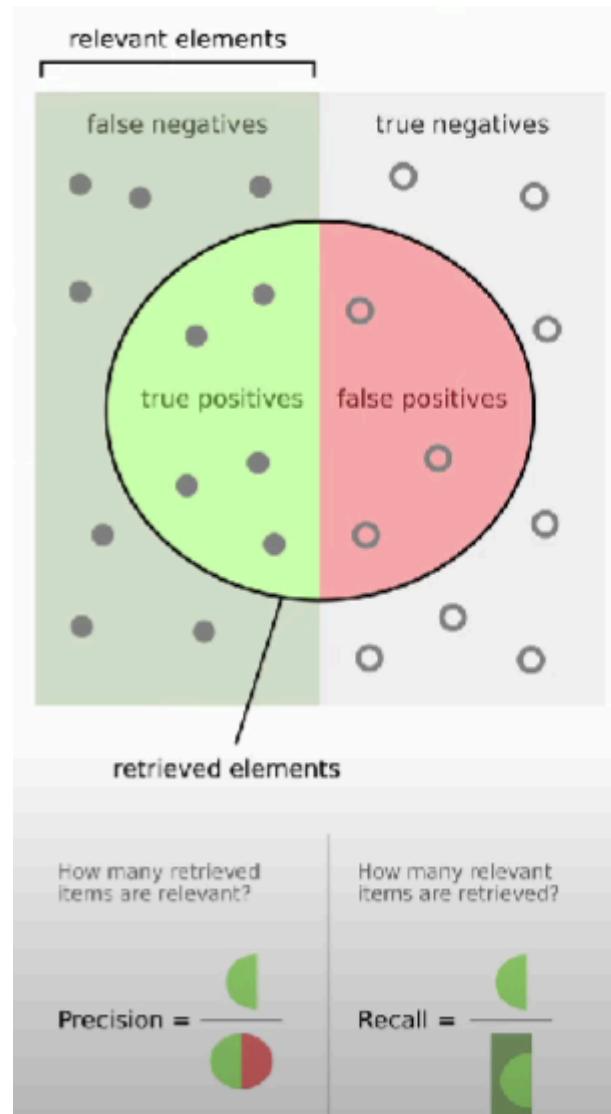
$$\text{Precision} = \frac{TP}{TP+FP}$$

where:

- **TP (True Positives):** The number of correctly predicted positive instances.
- **FP (False Positives):** The number of instances incorrectly predicted as positive.

Importance:

- Precision is crucial in scenarios where the **cost of false positives is high**, such as in spam detection or medical diagnosis. It helps in understanding how many of the predicted positive instances are actually positive.



Related Concepts

- Classification Report & Precision or Recall

Prediction Intervals

Prediction intervals estimate the range within which a future observation from the same distribution is likely to fall, with a specified confidence level.

Formula:

$$\bar{x} \pm t_{\alpha/2, n-1} \cdot s \cdot \sqrt{1 + \frac{1}{n}}$$

Where:

- \bar{x} : Sample mean
- s : Sample standard deviation
- n : Sample size
- $t_{\alpha/2, n-1}$: t-critical value for the chosen confidence level

Notes:

- Prediction intervals are always **wider** than a confidence interval for the mean.
- They use the t-distribution due to sample uncertainty.
- The interval is centered around \bar{x} but accounts for:
 - **Estimation error** of the mean
 - **Natural variability** of new values

Use Cases:

- Forecasting where a new measurement is likely to fall.
- Risk assessment and operational thresholds.

Preprocessing

Data Preprocessing

Data Preprocessing refers to the overall process of cleaning and transforming raw data into a format that is suitable for analysis and modelling. This includes a variety of tasks, such as:

Useful:

- Data Cleansing
- Data Transformation

Others:

- Data Collection
- Data Reduction

End goal:

- EDA

Feature Preprocessing

Feature preprocessing refers to the process of transforming raw data into a clean data set for learning models, after Data Preprocessing. This step is crucial for improving model performance and ensuring accurate predictions

1. **Feature Scaling:** Normalizing or standardizing features to ensure they are on a similar scale. Normalization and Scaling: Adjusting the range of features, often using techniques like min-max scaling or z-score normalization, to ensure that all features contribute equally to the model.
2. **Feature Selection:** Identifying and retaining the most relevant features that contribute to the predictive power of the model, often using statistical tests or model-based approaches.

-
3. **Dimensionality Reduction:** Reducing the number of features while preserving important information, using techniques like Principal Component Analysis (PCA).
 4. **Feature Engineering:** Creating new features from existing data to improve model performance, often based on domain knowledge.

Prevention Is Better Than The Cure

To ensure data products are effective essential to prioritize prevention over remediation of [Data Quality](#)

Prevention

Preventing data quality issues is the most effective strategy. This involves identifying and addressing potential problems at the source, ensuring that data is accurately entered from the beginning.

Remediation

When data quality issues do arise, organizations should implement remediation strategies, including:

- **Data Observability Tools:** Monitor data quality continuously to detect issues early.
- **Alerting Systems:** Notify stakeholders when data quality problems are identified.
- **Complex ETL Processes:** Manage data effectively to minimize errors.
- **Trust Building:** Address the erosion of trust that can result from poor data quality.

Consequences of Poor Data Quality

Failing to address data quality issues can lead to significant opportunity costs and hinder the ability to meet business goals. The sooner these issues are resolved, the cheaper and easier it is to manage them.

Motivating and Maintaining Data-Driven Value

To foster a culture of data quality, it is essential to motivate data producers by demonstrating the value of high-quality data.

Effective [Change Management](#) is vital for maintaining data quality. This includes: Clear Communication to ensure all stakeholders are informed about data quality standards.

Addressing Data Quality Issues

To effectively handle data quality issues, organizations should focus on:

1. **Detecting:** Identify issues as they arise through user reports, failed tests, or monitoring alerts.
2. **Understanding:** Analyze the root causes of data quality problems.
3. **Fixing:** Implement solutions to correct identified issues.
4. **Reducing:** Minimize the occurrence of future data quality problems.

Questions for Consideration

Q: What if data producers are not part of the data team but are business users (e.g., entering data into Google Sheets or Excel) with naming convention issues?

A: Encourage these users to apply the same data quality patterns by establishing agreements on data structure and implementing alerting and automated change management processes.

Primary Key

A primary key (PK) is a unique identifier for each record in a database table.

- **Uniqueness:** No two records can have the same primary key value.
- **Non-null:** A primary key cannot contain null values; every record must have a valid primary key.
- **Immutability:** Ideally, the primary key should not change over time, as it serves as a stable reference for the record

For example, an ISBN serves as a primary key for books, uniquely identifying each book in the database

Principal Component Analysis

PCA is a tool for [Dimensionality Reduction](#) in [Unsupervised Learning](#) to reduce the dimensionality of data.

It transforms the original data into a new coordinate system defined by the principal components, which are [orthogonal vectors](#) that capture the most [Variance](#) in the data.

It helps simplify models, enhances [interpretability](#), and improves computational efficiency by transforming data into a lower-dimensional space while [retaining the most significant variance](#), and reducing noise.

How PCA Works

- [Linear Technique:](#) PCA is a linear technique, meaning it assumes that the relationships between features are linear. This distinguishes it from methods like [Manifold Learning](#) which can capture non-linear relationships.
- [Principal Components:](#) PCA identifies principal components, which are linear combinations of the original features. These components are ordered by the amount of variance they capture from the data, with the first principal component capturing the most variance, the second capturing the second most, and so on.

Comparison with Other Techniques

- [t-SNE:](#) Unlike PCA, t-SNE is a non-linear technique used for visualization, preserving local structures in high-dimensional data.
- [Manifold Learning:](#) Techniques like Isomap and Locally Linear Embedding (LLE) are designed to capture non-linear structures, which PCA might miss due to its linear nature

Code Implementation

In [ML_Tools](#) see:

- [PCA-Based Anomaly Detection](#)
- [PCA_Analysis.ipynb](#)

Related terms

- [PCA Explained Variance Ratio](#)

- PCA Principal Components
-

Probability In Other Fields

Problem Definition

What is involved:

Clearly articulate the problem you're trying to solve and the outcomes you expect.

Follow up questions

What assumption can we make based on the problem?

What kind of questions are good to ask?

Business Context:

- What are the desired outcomes and how would success be measured?
- What are the limitations and feasibility of using machine learning in this context?

2. Data Availability and Quality:

- What data is available in quantity and quality and relevant to the problem?
- What is the format and structure of the data?

3. Feature Engineering and Model Selection:

- What are the key features or variables that might be predictive of the desired outcome?
- What type of machine learning model might be best suited for this problem (e.g., classification, regression, Clustering)?

4. Evaluation and Deployment:

- How will we evaluate the performance of the machine learning model?
- What metrics will be used to measure success?

Prompt Engineering

Prompt engineering is a technique in the field of natural language processing (NLP), particularly when working with large language models (LLMs).

It involves designing and optimizing input prompts to get the most relevant and accurate responses from these models.

Techniques like [prompt retrievers](#), which include systems like UPRISE and DaSLaM, enhance the ability to retrieve and generate contextually appropriate prompts.

Prompt engineering aims to guide LLMs toward producing desired outputs while minimizing ambiguity.

Key Takeaways

- Prompt engineering optimizes input to improve LLM responses.
- Techniques like prompt retrievers (e.g., UPRISE, DaSLaM) enhance prompt effectiveness.
- Quality prompts reduce ambiguity and guide model outputs.
- Applications span multiple industries, enhancing user interaction and content generation.

Key Components Breakdown

Methods:

- **Prompt Design:** Crafting specific, clear prompts to guide model responses.
- **Prompt Retrieval:** Utilizing systems like UPRISE and DaSLaM to find effective prompts based on context.

Concepts:

- **Contextualization:** Understanding the context in which prompts are used to improve relevance.
- **Iterative Testing:** Continuously refining prompts based on model performance.

Algorithms:

- **Retrieval-Augmented Generation (RAG):** Combines retrieval of relevant documents with generative responses.
- **Few-Shot Learning:** Providing examples within prompts to guide model behavior.

Concerns, Limitations, or Challenges

- **Ambiguity:** Poorly designed prompts can lead to vague or irrelevant responses.
- **Dependence on Training Data:** LLMs may produce biased or inaccurate outputs based on their training data.
- **Complexity:** Designing effective prompts requires a deep understanding of both the model and the task.

Example

For instance, if a user wants to generate a summary of a scientific article, a poorly constructed prompt like "Summarize this" may yield unsatisfactory results. In contrast, a well-engineered prompt such as "Provide a concise summary of the key findings and implications of the following article on climate change" is likely to produce a more relevant and informative response.

Follow-Up Questions

1. What are the best practices for evaluating the effectiveness of different prompts
2. How can prompt engineering be integrated into existing NLP workflows to enhance performance

Proportion Test

The proportion test is used to compare proportions between groups. It can be categorized into:

- **One-Sample Proportion Test:** Compares the proportion of successes in a single sample to a known population proportion.
- **Two-Sample Proportion Test:** Compares the proportions of successes between two independent samples.

Publish And Subscribe

The **Publish-Subscribe (Pub-Sub) model** is a messaging pattern that enables real-time data distribution by decoupling message producers from consumers. This architecture is widely used in [data streaming](#), [Event-Driven Architecture](#), and [Distributed Computing](#).

It can help in designing more efficient and [Scalability](#) and data processing architectures.

Core Components

This model ensures that multiple consumers can receive the same data without requiring direct connections between producers and consumers, allowing for a more scalable and flexible system.

- **Producers:** Entities or applications that generate data and publish messages to specific channels known as **Topics**. Each Topic represents a category or stream of data.
- **Consumers:** Applications or services that subscribe to Topics to receive messages. They process incoming data in real-time, enabling immediate action or analysis.

Importance in Data Streaming

Ensures continuous data flow, in contrast to [batch processing](#), which collects and processes data in groups at scheduled intervals. Streaming applications benefit from Pub-Sub by:

- Enabling real-time analytics and monitoring
- Supporting event-driven architectures
- Improving scalability by decoupling message producers from consumers

Questions for Consideration

- If you're working with a streaming dataset, why might [batch processing](#) not be suitable, and what alternatives would you consider?
- How does the decoupling of producers and consumers improve scalability in large-scale data systems?
- What are the trade-offs between a Pub-Sub model and a point-to-point messaging system?

Example: Apache Kafka

1. **Producers:** In a Kafka setup, a producer could be a web application that generates user activity events, such as clicks, page views, or purchases. This application publishes these events to a specific topic, for example, "user-activity".
2. **Topics:** The "user-activity" topic acts as a channel where all user activity events are sent. Multiple producers can publish messages to this topic without needing to know about the consumers.
3. **Consumers:** Various applications or services can subscribe to the "user-activity" topic to receive real-time updates. For instance:
 - An analytics service that processes user activity data to generate insights.
 - A notification service that sends alerts based on specific user actions (e.g., sending a welcome email after a user signs up).
 - A monitoring service that tracks user engagement metrics.

Workflow:

-
- When a user interacts with the web application, the producer generates an event and publishes it to the "user-activity" topic.
 - All subscribed consumers receive this event simultaneously, allowing them to process the data in real-time.
 - This decoupling means that the producer does not need to know how many consumers are listening or what they are doing with the data.

TL;dr

1-liner if the context of the change is long

Context

A few sentences on the high level context for the change. Link to relevant design docs or discussion.

This Change

What this change does in the larger context. Specific details to highlight for review. Include UI change screenshots or videos if applicable:

- Callout 1
- Callout 2
- Callout 3

Test Plan

Go over how you plan to test it. Your test plan should be more thorough the riskier the change is. For major changes, I like to describe how I E2E tested it and will monitor the rollout.

Links

- *link to ticket*
- *link to design doc*
- *link to design*

Checklist

- Pull request title is succinct with [tiny] if it's extra small
- Describes the problem
- Describes the solution (screenshots included if UI changes)
- Has a test plan
- Contains links to any context (Slack, Figma, JIRA ticket, etc.)

- Code is self reviewed for readability, approach, and edge cases
 - Lines changed that may require additional explanation are annotated with an explanation
 - Change is ideally < 500 lines if possible. < 150 is ideal.
-

Push Down

Query pushdown aims to execute as much work as possible in the source databases.

Push-downs or query pushdowns push transformation logic to the source database. This reduces to store data physically and transfers them over the network.

For example, a [semantic layer](#) or [Data Virtualization](#) translates the transformation logic into [SQL Querying|queries](#) and sends the SQL queries to the database. The source database runs the SQL queries to process the transformations.

Pushdown optimization increases mapping performance when the source database can process transformation logic faster than the semantic layer itself.

Pycaret

PyCaret is an open-source, low-code Python library designed to simplify machine learning workflows.

It allows users to build, evaluate, and deploy machine learning models with minimal coding and effort.

PyCaret provides an end-to-end solution for automating repetitive tasks in machine learning, such as

- [Preprocessing](#),
- model training,
- [hyperparameter tuning](#)
- [Model Deployment|Deployment](#)

Implementation

See: <https://pycaret.gitbook.io/docs/get-started/quickstart>

Resources: <https://github.com/pycaret/pycaret/tree/master>

In [ML_Tools](#) see:

- [Pycaret_Example.py](#)

Key Features of PyCaret

- Ease of Use: PyCaret is designed to be beginner-friendly, enabling users to build models without deep expertise in coding.
 - Modular Design: PyCaret supports various machine learning tasks through its modular APIs:
 - Classification: `pycaret.classification`
 - Regression: `pycaret.regression`
 - Clustering: `pycaret.clustering`
 - Anomaly Detection: `pycaret.anomaly`
 - NLP: `pycaret.nlp`
 - Time Series Forecasting: `pycaret.time_series`
-

Introduction

- Automated Machine Learning (AutoML): PyCaret automates data preprocessing, feature engineering, model selection, and [hyperparameter tuning](#).
- Integration: PyCaret integrates well with other Python libraries, such as Pandas, NumPy, and Plotly.
- Model Evaluation and Comparison: [Model Selection](#): It provides an easy way to compare multiple models and their performance metrics in a single function call.
- Deployment [Model Deployment](#): Facilitates the deployment of trained models using tools like Flask, FastAPI, or Microsoft Power BI.

Notes

Object or functional APIs

Advantages of PyCaret

- Time-Saving: Reduces the coding and time required to build machine learning pipelines.
- Quick prototyping of machine learning models.
- Educational purposes for teaching machine learning concepts.
- Rapid development of machine learning solutions for business problems.

Pygraphviz

PyGraphviz Interface: Thin wrapper around the C Graphviz API. Better integration with NetworkX, especially with graphviz_layout.

Advantages:

Native Graphviz object model (AGraph).

Seamless conversion between NetworkX graphs and Graphviz objects.

Supports advanced Graphviz features and layout options.

Limitations:

Requires Graphviz development libraries to be installed (can be hard to set up on Windows).

Slightly more complex installation due to C bindings.

Example with NetworkX:

```
python from networkx.drawing.nx_agraph import graphviz_layout pos = graphviz_layout(G, prog="dot")
```

Pyspark

[Python API for Apache Spark](#), a [distributed processing framework](#) for big data analysis and machine learning on clusters.

Part of [Apache Spark](#)

[Directed Acyclic Graph \(DAG\)](#)

Interlinked with [SQL](#) queriers

[Tutorial](#)

Can run local.

Similar to [Pandas](#)

Pytorch

[Text Generation With LSTM Recurrent Neural Networks in Python with Keras want for PyTorch](#)

Open-source [Deep Learning](#) framework with dynamic computational graphs, emphasizing flexibility and research.

Similar to [Tensorflow](#).

Framework pieces:

- torch: a general purpose array library similar to [Numpy](#) that can do computations on GPU when the tensor type is cast to (torch.cuda.TensorFloat)
- torch.autograd: a package for building a computational graph and automatically obtaining gradients
- torch.nn: a [Neural network|Neural Network](#) library with common layers and [Loss function](#)
- torch.optim: an optimization package with common optimization algorithms like [Stochastic Gradient Descent](#)

Basics of PyTorch

Tensors arrays

PyTorch uses tensors, which are similar to NumPy arrays, but with GPU acceleration.

```
import torch

# Creating a tensor
x = torch.tensor([2.0, 3.0, 4.0])

# Tensor operations
y = x + 2
z = x * 3

print(y) # Output: tensor([4., 5., 6.])
print(z) # Output: tensor([ 6.,  9., 12.])
```

Automatic Differentiation

PyTorch can compute gradients automatically with `autograd`.

```
# Create tensor with gradient tracking (input value)
x = torch.tensor([1.0, 2.0, 3.0], requires_grad=True) #`requires_grad=True`, which tells PyTorch to track all operations p

# Perform some operations
y = x ** 2 #formula
z = y.sum() #14

# Compute gradients
z.backward() #backpropagation (the chain rule of calculus). Since `z` is a scalar, PyTorch can compute the gradients of `z` with respect to all the inputs.

# Print gradients (dy/dx)
print(x.grad) # Output: tensor([2., 4., 6.])
```

The gradient is calculated by the chain rule:

- For $y = x^2$, the derivative of y with respect to x is $\frac{dy}{dx} = 2x$.
- Since z is the sum of the elements of y , the gradient of z with respect to each element in x is $\frac{\partial z}{\partial x_i} = 2x_i$ for each element in x .

`</mark>z` is a formula but calculates the derivative wrt x , so the derivative of y with x ==

So, the gradients for each element in x are: $\frac{\partial z}{\partial x} = [2 \times 1.0, 2 \times 2.0, 2 \times 3.0] = [2.0, 4.0, 6.0]$

The gradients are stored in `x.grad`. After calling `z.backward()`, `x.grad` contains the derivative of z with respect to each element of x , which is `[2.0, 4.0, 6.0]`

Gradient Computation (Summary):

- The gradient $\frac{\partial z}{\partial x}$ represents how much the output z changes for a small change in x . In our case, if we slightly increase x_1 , x_2 , or x_3 , the change in z can be predicted using these gradients.
- This gradient information is used to update weights in neural networks during training. For example, in optimization algorithms like ([Stochastic Gradient Descent](#)), these gradients are used to adjust model parameters in the direction that minimizes the loss function.

Confusion: total derivative is not the partial derivatives

The confusion here comes from the distinction between **partial derivatives** (for each element in a tensor) and **total derivatives**. Let me clarify this.

In the context of:

```
x = torch.tensor([1.0, 2.0, 3.0], requires_grad=True)
y = x ** 2
z = y.sum()
z.backward()
```

- `x` is a tensor: $x = [1.0, 2.0, 3.0]$
- `y = x^2` computes element-wise squares: $y = [1.0^2, 2.0^2, 3.0^2] = [1.0, 4.0, 9.0]$
- `z = y.sum()` adds the elements in `y`: $z = 1.0 + 4.0 + 9.0 = 14.0$

Now, you're asking about the derivative of z with respect to x , specifically whether $\frac{dz}{dx}$ **should be the sum of the derivatives** and equal to 4.

Derivative with Respect to Each Element (Partial Derivative)

When you calculate the gradient of z with respect to x , you're computing the **partial derivatives** of z with respect to each element in x . The function z depends on each element of x individually.

For each element x_i in x , we are calculating:

$$\frac{\partial z}{\partial x_i} = \frac{\partial (x_1^2 + x_2^2 + x_3^2)}{\partial x_i} = 2x_i$$

This gives:

$$\frac{\partial z}{\partial x} = [2x_1, 2x_2, 2x_3] = [2 \times 1.0, 2 \times 2.0, 2 \times 3.0] = [2.0, 4.0, 6.0]$$

These are the gradients stored in `x.grad` after calling `z.backward()`.

Total Derivative vs Partial Derivatives

If we are talking about **partial derivatives**, we get a gradient for each individual component of x :

Introduction

- $\frac{\partial z}{\partial x_1} = 2.0$
- $\frac{\partial z}{\partial x_2} = 4.0$
- $\frac{\partial z}{\partial x_3} = 6.0$

These partial derivatives form the gradient vector: [2.0, 4.0, 6.0].

Why Is It Not Just 4?

If you're thinking of the total derivative, that would be different from what we are calculating here. **The sum of the derivatives of z with respect to all components of x is:**

$$\sum_{i=1}^3 \frac{\partial z}{\partial x_i} = 2 + 4 + 6 = 12$$

However, this total derivative is not what we are computing here. **We are computing the partial derivatives for each element of x separately**, which results in the gradient vector [2.0, 4.0, 6.0].

In Summary:

- **Gradient ($x.grad$):** A vector of partial derivatives of z with respect to each element in x , giving us [2.0, 4.0, 6.0].
- **Sum of Gradients:** The sum of the elements of the gradient vector is 12, but that's not the gradient of z with respect to x as a whole—it's just a summation of the partial derivatives.

Basic Neural network|Neural Network Implementation

A simple feedforward network using PyTorch.

```
import torch
import torch.nn as nn

# Define a simple neural network
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.fc = nn.Linear(2, 1)

    def forward(self, x):
        #- **`forward()` method**: This defines the forward pass, i.e., how the input data is transformed as it moves through the network.
        return self.fc(x)

# Create the network
net = SimpleNet()

# Create input tensor
x = torch.tensor([1.0, 2.0])(#10-20)

# Forward pass
output = net(x)

print(output) # Output: a tensor from the linear layer
```

Input Tensor and Forward Pass

```
x = torch.tensor([1.0, 2.0](#10-20))
output = net(x)
```

- `torch.tensor([1.0, 2.0](#10-20))` : This defines a 2D input tensor with two features, which corresponds to the two input nodes in the network.
- `net(x)` : This performs a **forward pass**, feeding the input tensor `x` into the network. The linear layer applies the learned weights and bias to compute the output.

Output The output of the network is a tensor from the linear layer, which corresponds to the result of the operation $y = w_1 \cdot x_1 + w_2 \cdot x_2 + b$, where:

- w_1 and w_2 are the learned weights for each input feature.
- b is the learned bias.

[Use Cases for a Simple Neural Network Like](#)

Training a Simple Model

An example of training a linear model with

- [Gradient Descent](#):
- [Loss function: Ordinary Least Squares](#)
- [Stochastic Gradient Descent|SGD](#):

Training of [Linear Regression](#) model. This model find best w,b in $y=wx+b$.

```

import torch
import torch.optim as optim

# Data
x = torch.tensor([[1.0], [2.0], [3.0], [4.0]])
y = torch.tensor([[2.0], [4.0], [6.0], [8.0]])

# Model
model = nn.Linear(1, 1) #**`nn.Linear(1, 1)`** defines a simple linear model with **one input feature** and **one output**

# Loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01) # stochastic gradient descent

# Training loop
for epoch in range(100):
    # Forward pass #- **Forward pass**: For each epoch, the model makes predictions (`y_pred`) by passing the input `x` through the model
    y_pred = model(x)

    loss = criterion(y_pred, y) # - **Loss calculation**: The loss is computed by comparing `y_pred` with the true values `y`

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward() #- **Backward pass**: The gradients of the loss with respect to the model's parameters are computed using backpropagation
    optimizer.step() #- **Optimization step**: The optimizer (SGD) updates the model's parameters (`w` and `b`) based on the gradients

    if epoch % 20 == 0:
        print(f'Epoch {epoch}, Loss: {loss.item()}')

# Output the trained model's parameters
print(list(model.parameters()))

```

Moving Tensors to GPU

Summary:

- speed up training e.g. of [Neural network|Neural Network](#)
- use [Parallelism](#) for simultaneous calculations
- GPU can do larger batches of computations, better on the memory, better for [Gradient Descent](#) estimations
-

PyTorch makes it easy to move computations to a GPU.

Introduction

```
# Check for GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Create a tensor and move it to the GPU
x = torch.tensor([1.0, 2.0, 3.0]).to(device)

print(x) # Output: tensor([1., 2., 3.], device='cuda:0') (if GPU is available)
```

Both the **model** and the **data** are moved to the GPU (`device='cuda'`). All computations, including the forward pass, loss calculation, backward pass, and optimizer step, happen on the GPU.

```

import torch
import torch.nn as nn
import torch.optim as optim

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Initialize the model, loss function, and optimizer
model = SimpleNN().to(device) # Move model to GPU
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Dummy input data and labels (move them to GPU)
inputs = torch.randn(64, 1, 28, 28).to(device) # 64 images of 28x28 pixels
labels = torch.randint(0, 10, (64,)).to(device) # Random labels for 64 images

# Forward pass (computation happens on the GPU)
outputs = model(inputs)
loss = criterion(outputs, labels)

# Backward pass and optimization
optimizer.zero_grad()
loss.backward() # Compute gradients on the GPU
optimizer.step() # Update model weights

print("Training step completed on:", device)

```

Pycaret_Anomaly.ipynb

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Outliers/Pycaret_Anomaly.ipynb

Pycaret_Example.py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Deployment/Pycaret/Pycaret_Example.py

Pydantic

Pydantic is a Python library used for [data validation](#) and settings management using Python type annotations.

It provides a way to define data models with type hints, and it automatically validates and parses input data to ensure it matches the specified types.

Pydantic is often used in applications where data integrity is crucial, such as in web APIs, configuration management, and data processing pipelines. It helps developers catch errors early by enforcing type constraints and providing clear error messages when data does not conform to the expected format.

In [ML_Tools](#) see:

- [Pydantic.py](#)
- [Pydantic_More.py](#)

What Pydantic Does:

- **Data Validation and Parsing**

Pydantic takes raw input data (e.g., dictionaries, JSON) and validates it against the [type checking](#) types and constraints defined in a `BaseModel`. If the input data doesn't match the requirements (e.g., wrong type, missing fields, invalid values), it raises a `ValidationError`.

Example:

```
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name: str
    user = User(id="1", name="Alice") # Automatically parses "1" to integer.
```

- **Automatic Type Conversion**

Pydantic can coerce compatible types into the expected type. For example, if a field expects an `int` but receives a string `"123"`, it will try to convert it to an integer.

- **Error Messaging**

If validation fails, Pydantic provides detailed error messages explaining what went wrong, making debugging easier.

- **Nested and Complex Data Models**

Pydantic supports nested models and complex data structures, enabling you to handle hierarchical data easily.

- **Settings Management**

Pydantic can load configuration from environment variables or other sources using its `BaseSettings` class, making it handy for managing application settings.

- **Serialization and Deserialization**

Pydantic models support converting data into [JSON](#) or dictionaries, making it easy to work with web APIs or store validated data.

Key Advantages of Pydantic:

1. Type-Safe Programming:

By relying on Python's type hints, Pydantic promotes better coding practices and helps prevent runtime errors.

2. Ease of Use:

Pydantic abstracts a lot of the boilerplate code you'd write manually for validating and parsing data.

3. Error Reporting:

Pydantic provides clear and structured error messages, making [debugging](#) simpler.

4. Interoperability:

It works well with libraries like [FastAPI](#), where it powers request/response validation and serialization.

Use Cases:

1. Web APIs:

Validating incoming HTTP requests and outgoing responses (e.g., with FastAPI).

2. Data Processing:

Ensuring raw input data from files or APIs meets requirements before processing.

3. Configuration Management:

Validating and loading application settings from environment variables or files.

4. Data Pipelines:

Verifying the integrity of data as it moves through pipeline stages.

Analogy to Summarize:

Think of Pydantic as a **data traffic cop**. It stands at the intersection where raw data enters your application, ensuring that:

- The data is well-formed.
- It complies with rules you've set (type, format, constraints).
- It's transformed into the expected structure (if possible).

By using Pydantic, you focus on defining the rules, and it ensures the data fits them—saving you from writing repetitive validation code.

Is Pydantic Object-Oriented Programming (OOP)?

While Pydantic uses classes and inheritance (features of OOP), it is **not purely OOP in intent or design**. Instead, it is:

- **Data-centric**: Focused on defining and validating data structures rather than encapsulating behavior like traditional OOP.
- **Declarative**: Pydantic models are **declarative** in nature. You define the "shape" of your data (fields and their types) and rely on Pydantic to handle validation, parsing, and serialization.

Differences from Typical OOP:

- **Behavior vs. Structure**:

Traditional OOP often centers on defining behavior (methods) alongside data. Pydantic, on the other hand, prioritizes defining and validating data.

- **State Management**:

In OOP, objects encapsulate their state and methods for interacting with it. Pydantic models are more lightweight and focused on validation, not managing stateful objects.

Similarities to OOP:

- **Class-Based Models:**

Pydantic models are Python classes, and you can use inheritance, encapsulation, and even add methods to your models.

- **Reusability:**

You can define base models and extend them, similar to class inheritance in OOP.

Pydantic.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Pydantic.py

Explanation:

- **BaseModel:** This is the base class for creating data models in Pydantic. You define your model by subclassing `BaseModel` and specifying fields with type annotations.
- **Optional:** Used to indicate that a field is optional.
- **List:** Used to specify a list of items, in this case, a list of strings for friends.
- **Validator:** A custom validator is used to enforce additional constraints, such as ensuring the age is positive.
- **ValidationError:** This exception is raised when the input data does not conform to the model's constraints.

This script demonstrates how Pydantic can be used to validate and parse data, ensuring it meets the specified types and constraints.

Pydantic_More.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Pydantic_More.py

Key Features Demonstrated in the Script:

1. **Nested Models:** Use of the `Friend` model inside the `User` model.
2. **Custom Validators:** Validating `age` and `email` fields with specific logic.
3. **Dynamic Defaults:** Using `datetime.now` for `created_at`.
4. **Field Aliases:** Supporting different key names during parsing and serialization.
5. **Configuration Options:** Stripping whitespace and enabling strict typing.
6. **Model Inheritance:** Extending the `User` model to create an `AdminUser` model.
7. **Parsing Raw Data:** Demonstrating `parse_raw` for JSON strings.

Pyright Vs Pydantic

While [Pyright](#) and [Pydantic](#) serve different roles in Python development, they complement each other well.

Pyright helps ensure that the code adheres to type constraints before execution, while Pydantic ensures that the data being processed adheres to the expected types and formats during runtime.

Key Differences

1. Purpose:

- **Pyright** is aimed at improving code quality through static analysis and type checking.
- **Pydantic** is focused on runtime [data validation](#), ensuring that the data conforms to specified types and constraints.

2. Functionality:

- **Pyright** checks for type errors and enforces type hints during development, preventing potential issues before the code is executed.
- **Pydantic** validates and parses data at runtime, providing clear error messages when data does not conform to the expected format.

3. Use Cases:

- **Pyright** is beneficial in any Python project where type safety is desired, especially in large codebases.
- **Pydantic** is particularly useful in applications that require data validation, such as web frameworks (e.g., [FastAPI](#)) and data processing pipelines.

Key Similarities

1. Type Annotations:

- Both utilize Python's type hints to define and enforce types, promoting better coding practices and reducing runtime errors.

2. Error Handling:

- Both tools provide mechanisms for error reporting, although they do so at different stages (compile-time for Pyright and runtime for Pydantic).

3. Improving Code Quality:

- Both contribute to overall code quality and maintainability, albeit through different approaches—Pyright through static analysis and Pydantic through runtime validation.

Pyright

Pyright is a [static type checker](#) for Python that enhances code reliability by enforcing type constraints [at compile-time](#).

It utilizes type hints to identify potential errors, such as type mismatches, before runtime, thereby improving code robustness.

Pyright significantly reduces runtime errors by enforcing type constraints at compile-time.

The use of type hints in Pyright improves code readability and maintainability, serving as [Documentation & Meetings](#) for function signatures.

Related Topics

- Type inference in programming languages
- The role of type systems in [functional programming](#)

- Debugging
 - Maintainable Code
 - type checking
-

Follow up questions

- How does the inclusion of Pyright impact the performance of large-scale Python applications?
- What are the trade-offs between using Pyright and other static type checkers in terms of accuracy and speed?

@pytest.fixture Explanation

`@pytest.fixture` is a decorator in `pytest` used to define reusable test setup functions. It allows tests to use shared resources without redundant code.

Example & Usage

python

Copy code

```
import pytest
@pytest.fixture
def sample_data():
    return {"name": "John Doe", "age": 30}
def test_example(sample_data):
    assert sample_data["name"] <mark> "John Doe"
    assert sample_data["age"] </mark> 30
```

◆ How it works:

- The function `sample_data()` is decorated with `@pytest.fixture`, making it a fixture.
- The test function `test_example()` receives `sample_data` as an argument.
- `pytest` automatically provides the fixture data when running the test.

Why use fixtures?

- Avoids repetitive setup code.
- Ensures clean test environments.
- Can handle resource management (e.g., opening/closing database connections, creating temporary files).

Python Click

Python Click, or "Command Line Interface Creation Kit," is a library for building command-line interfaces (CLIs). It supports arbitrary nesting of commands, automatic help page generation, and lazy loading of subcommands.

In [ML_Tools](#) see: [Click_Implementation.py](#)

Installation

To install Click, use pip:

```
pip install click
```

Creating a Command Group

Click uses groups to organize related commands. A group serves as a container for multiple commands.

```
import click
import json

@click.group("cli")
@click.pass_context
@click.argument("document")
def cli(ctx, document):
    """An example CLI for interfacing with a document"""
    with open(document) as _stream:
        _dict = json.load(_stream)
    ctx.obj = _dict

def main():
    cli(prog_name="cli")

if __name__ == '__main__':
    main()
```

Running `python script.py --help` generates an automatic help page.

Adding Commands

Commands can be added to a Click group using the `@<group>.command` decorator.

Checking Context Object

```
import pprint

@cli.command("check_context_object")
@click.pass_context
def check_context(ctx):
    pprint.pprint(type(ctx.obj))
```

Custom Pass Decorator

A pass decorator allows passing specific objects through context.

```
pass_dict = click.make_pass_decorator(dict)
```

Retrieving Keys from Context

```
@cli.command("get_keys")
@pass_dict
def get_keys(_dict):
    keys = list(_dict.keys())
    click.echo("The keys in our dictionary are", fg="green")
    click.echo(click.style(str(keys), fg="blue"))
```

Retrieving a Specific Key

```
@cli.command("get_key")
@click.argument("key")
@click.pass_context
def get_key(ctx, key):
    if key in ctx.obj:
        pprint.pprint(ctx.obj[key])
    else:
        click.echo(f"Key '{key}' not found in document.", err=True)
```

Arbitrary Nesting of Commands

```
@cli.command("get_summary")
@click.pass_context
def get_summary(ctx):
    ctx.invoke(get_key, key="summary")
```

Adding Optional Parameters

Optional parameters can be defined using the `@click.option` decorator.

```

@cli.command("get_results")
@click.option("-d", "--download", is_flag=True, help="Download the result to a JSON file")
@click.option("-k", "--key", help="Specify a key from the results")
@click.pass_context

def get_results(ctx, download, key):
    results = ctx.obj.get('results', [])
    if key:
        results = {key: sum(entry.get(key, 0) for entry in results)}
    if download:
        filename = f"{key or 'results'}.json"
        with open(filename, 'w') as w:
            json.dump(results, w)
        click.echo(f"File saved to {filename}")
    else:
        pprint.pprint(results)

```

Using `@click.pass_obj`

`@click.pass_obj` passes only `ctx.obj` instead of the full context.

```

@cli.command("get_text")
@click.option("-s", "--sentences", is_flag=True, help="Return sentences")
@click.option("-p", "--paragraphs", is_flag=True, help="Return paragraphs")
@click.option("-d", "--download", is_flag=True, help="Download as JSON file")
@click.pass_obj

def get_text(_dict, sentences, paragraphs, download):
    results = _dict.get('results', [])
    text = {} if paragraphs else {'text': ''}
    for idx, entry in enumerate(results):
        if paragraphs:
            text[idx] = entry.get('text', '')
        else:
            text['text'] += entry.get('text', '')
    if sentences:
        text = {i: s for i, s in enumerate(text.get('text', '').split('.')) if s}
    pprint.pprint(text)
    if download:
        filename = "paragraphs.json" if paragraphs else "text.json"
        with open(filename, 'w') as w:
            json.dump(text, w)
        click.echo(f"File saved to {filename}")

```

Handling User Input

Click provides `@click.prompt` to interact with users.

```
@cli.command("prompt_user")
@click.pass_context
def prompt_user(ctx):
    name = click.prompt("Enter your name")
    age = click.prompt("Enter your age", type=int)
    click.echo(f"Hello {name}, you are {age} years old!")
```

Handling Confirmation

Use `@click.confirm` to get user confirmation before proceeding.

```
@cli.command("confirm_action")
@click.pass_context
def confirm_action(ctx):
    if click.confirm("Do you want to proceed?"):
        click.echo("Proceeding with action...")
    else:
        click.echo("Action canceled.")
```

Conclusion

Python Click simplifies CLI creation with its decorators and built-in features like automatic help generation and context passing. This guide provides a foundation for building more advanced command-line applications. Additionally, handling user input and confirmations enhances the interactivity of CLI applications.

Python

dynamic language lower learning, support object orientated

[Immutable vs mutable](#)

Pytorch Vs Tensorflow

- [Tensorflow](#) is widely adopted but pytorch picking up
- Dynamic vs static graph
- Tensorboard is better than [pytorch](#) visualization
- Plain tensorflow looks pretty much like a library
- Abstraction is better in pytorch, even data parallelism
- Tf.contrib, [keras](#) to rescue

P Values

A p-value is a measure of the evidence against a null hypothesis. p-values indicate whether an effect exists
[Used in Feature Selection](#)

P Values In Linear Regression In Sklearn

Question How to include p values in sklearn for a Linear Regression?

```
import scipy.stats as stat.
```

You can modify the class of LinearRegression() from sklearn to include them

C:\Users\RhysL\Desktop\DSObs\1_Inbox\Work\Udemy\Part_5_Advanced_Statistical_Methods(Machine_Learning)\multiple_linear_regression\sklearn - How to properly include p-values.ipynb

What is f_regression and why can it compute p values?

```
from sklearn.feature_selection import f_regression p_values = f_regression(x,y)[1] p_values
```

[link](#)

We will look into: f_regression finds the F-statistics for the *simple* regressions created with each of the independent variables. In our case, this would mean running a simple linear regression on GPA where SAT is the independent variable and a simple linear regression on GPA where Rand 1,2,3 is the independent variable. The limitation of this approach is that it does not take into account the mutual effect of the two features

```
f_regression(x,y)
```

There are two output arrays. The first one contains the F-statistics for each of the regressions. The second one contains the p-values of these F-statistics

```
outputs: (array([56.04804786, 0.17558437]), array([7.19951844e-11, 6.76291372e-01]))
```

Parametric Vs Non Parametric Models

Parametric Models

In [Statistics](#)

Definition: Models that summarize data with a set of parameters of fixed size, regardless of the number of data points.

Characteristics:

- Assumes a specific form for the function mapping inputs to outputs (e.g., linear regression assumes a linear relationship).
- Requires estimation of a finite number of parameters.
- Generally faster to train and predict due to their simplicity.
- Risk of underfitting if the model assumptions do not align well with the data.

Examples:

- Linear [regression](#), logistic regression, neural networks (with a fixed architecture), Bernoulli

Non-parametric Models

- **Definition:** Models that do not assume a fixed form for the function mapping inputs to outputs and can grow in complexity with more data.
- **Characteristics:**
 - Do not make strong assumptions about the underlying data distribution.
 - Can adapt to the data's complexity, potentially capturing more intricate patterns.
 - Generally require more data to make accurate predictions.
 - Risk of overfitting, especially with small datasets, as they can model noise in the data.
- **Examples:** K-nearest neighbors, decision trees, [support vector machines](#) (with certain kernels).

Key Differences

- **Flexibility:** Non-parametric models are more flexible and can model complex relationships, while parametric models are simpler and rely on assumptions about the data.
- **Data Requirements:** Non-parametric models typically require more data to achieve good performance compared to parametric models.
- **Computation:** Parametric models are usually computationally less intensive than non-parametric models.

Parametric Vs Non Parametric Tests

[Parametric tests](#) are statistical tests that make [assumptions about the distribution](#) of the data. For example, a t-test assumes that the data is normally distributed. Non-parametric tests do not make assumptions about the distribution of the data. Parametric tests are generally more powerful than non-parametric tests, but they are only valid if the data meets the [Statistical Assumptions](#) of the test.

[Non-parametric tests](#) are less powerful than parametric tests, but they can be used on any type of data, regardless of the distribution.

Parsimonious

Parsimonious refers to a principle in [Model Selection](#) and statistical modeling that emphasizes [simplicity](#). In the context of regression and other statistical models, a parsimonious model is one that explains the data with the fewest possible parameters or predictors while still providing a good fit.

A parsimonious model is one that achieves a good balance between simplicity and explanatory power.

Key Points about Parsimonious Models:

1. **Simplicity:** A parsimonious model avoids unnecessary complexity. It uses only the essential variables that contribute meaningfully to the prediction or explanation of the outcome.
2. **Avoiding Overfitting:** By keeping the model simple, a parsimonious approach helps prevent overfitting, where a model learns the noise in the training data rather than the underlying pattern. Overfitting can lead to poor generalization to new, unseen data.
3. **Interpretability:** Simpler models are often easier to interpret and understand. This is particularly important in fields where explaining the model's decisions is crucial, such as healthcare or finance.
4. **Balance:** The goal is to strike a balance between model accuracy and complexity. A parsimonious model should provide a good fit to the data without being overly complicated.

Pd.Grouper

`pd.Grouper` is a utility in pandas used with `.groupby()` to flexibly group data by a specific column, often useful for time-based grouping, multi-index grouping, or applying custom frequency aggregation.

See:

- [Groupby](#)
- [Multi-level index](#)

Why Use `pd.Grouper` ?

- Allows more readable and declarative code when working with time-indexed data.
- Supports multi-index groupings without restructuring your data.
- Enables resampling-like grouping without setting the index.

Syntax

```
pd.Grouper(key=None, level=None, freq=None, axis=0, sort=False)
```

Parameters

- `key` : The column name to group by.
- `level` : For MultiIndex, the level to group by.
- `freq` : Used to group time-series data (e.g., `'D'` for daily, `'M'` for monthly).
- `axis` : Default is 0 (rows).
- `sort` : Whether to sort the result.

Pdoc

[PDOC](#) is a documentation generator specifically designed for Python projects. Here are some key features and details:

1. **Automatic Documentation:** It scans your Python code and automatically generates documentation based on the docstrings you include in your code. This means that as long as you write clear comments and descriptions in your code, pdoc can create documentation without much extra work.
2. **Modern and Clean Design:** The output documentation is visually appealing and easy to navigate. It uses a modern design that enhances readability, making it user-friendly for anyone who needs to understand your code.
3. **Customization Options:** While pdoc generates documentation automatically, it also allows for some customization. You can configure settings to adjust how the documentation looks and what content is included.
4. **Markdown Support:** pdoc supports Markdown, which means you can use Markdown syntax in your docstrings to format your documentation with headings, lists, links, and more.
5. **Easy Integration:** It can be easily integrated into your development workflow, allowing you to generate documentation as part of your build process or whenever you need it.
6. **No Manual Guides Required:** With pdoc, you can avoid the tedious task of writing extensive documentation manually. Instead, you can focus on writing code, and pdoc will handle the documentation generation for you.

Once you generate the HTML files using `pdoc`, you have several options for what to do with them:

1. **Local Viewing:** You can open the generated HTML files directly in your web browser to view the documentation locally. This is useful for personal reference or for sharing with a small team.
2. **Hosting on a Web Server:** You can upload the generated HTML files to a web server to make the documentation accessible to a wider audience. This is a common practice for open-source projects or any project where you want to share documentation with users or collaborators.
3. **Integrating with Project Repositories:** If you're using version control systems like Git, you can include the generated documentation in your repository. This way, anyone who clones the repository can access the documentation easily.
4. **Publishing to Documentation Platforms:** You can publish the HTML files to documentation hosting platforms like Read the Docs, GitHub Pages, or similar services. These platforms often provide additional features like versioning, search functionality, and easy navigation.
5. **Archiving:** You can keep the generated HTML files as part of your project archive for future reference. This is useful for maintaining a history of your documentation as your project evolves.
6. **Sharing with Stakeholders:** If you are working on a project with stakeholders or clients, you can share the HTML documentation with them to provide insights into the project's structure and functionality.

To explicitly tell `pdoc` to document the local `scripts` directory, you need to prepend `./` to the directory name.

Here's how to do it:

1. **Open your terminal.**
2. **Navigate to the directory** where your `scripts` folder is located (which seems to be `C:\Users\RhysL\Desktop\Auto_YAML`).
3. **Run the `pdoc` command** with the `-o` option and prepend `./` to the `scripts` directory name:

```
pdoc -o docs ./scripts
```

This command tells `pdoc` to generate documentation for the local `scripts` directory and save the output in the `docs` folder.

After running this command, you should find the generated documentation in the `docs` folder, which you can then open in your web browser.

what to do next

1. **View Locally:** Open the HTML files in your web browser to view the documentation. You can start by opening the `index.html` file in the `docs` folder. This file typically serves as the main entry point for your documentation.
2. **Host on a Web Server:** If you want to make the documentation accessible online, you can upload the `docs` folder to a web server. This could be a personal website, a cloud storage service that supports HTML hosting, or a documentation hosting platform like GitHub Pages or Read the Docs.
3. **Integrate into a Project Repository:** If you're using version control (like Git), you can include the `docs` folder in your repository. This way, anyone who clones the repository can easily access the documentation.

Pmdarima

Helps find [Model Parameters](#) for [ARIMA](#) models

[Forecasting_AutoArima.py](#)

Programming Languages

Q

Table of Contents

- Q-Learning
- QUERY GSheets
- Quartz
- Query Optimisation
- Query Plan
- Querying
- QuickSort

Q Learning

Q-learning is a value-based, model-free [Reinforcement learning](#) algorithm where the agent learns the optimal [policy](#) by updating Q-values based on the rewards received. It is particularly useful in discrete environments like grids.

Uses a Q-Table which is populated by Q-values which are the maximum expected future reward for the given state and action. We improve the Q-Table in an iterative approach

Resources:

- [Q-Learning Explained - Reinforcement Learning Tutorial](#)

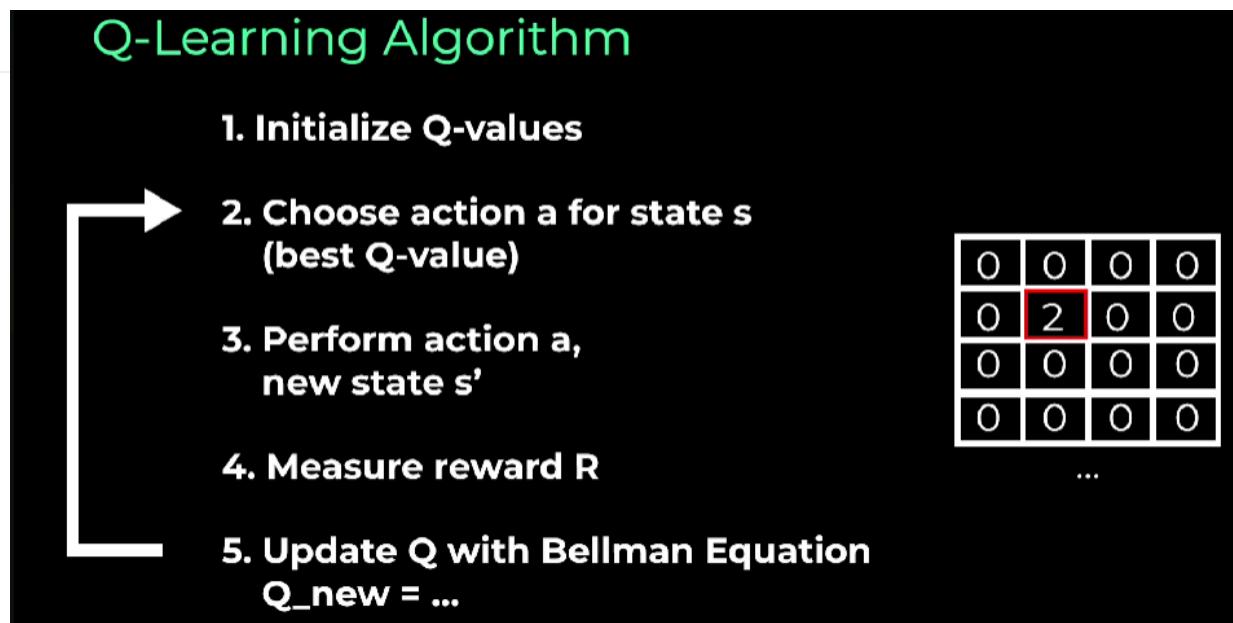
Q-learning update rule:

The left hand side gets updated ([Bellman Equations](#))

$$Q_{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Explanation:

- $Q(s_t, a_t)$: The Q-value of the current state s_t and action a_t .
- α : The learning rate, determining how much new information overrides old information.
- r_{t+1} : The reward received after taking action a_t from state s_t .
- γ : The discount factor, balancing immediate and future rewards.
- $\max_{a'} Q(s_{t+1}, a')$: The maximum Q-value for the next state s_{t+1} across all possible actions a' .

**Notes:**

- Q-learning is well-suited for environments where the state and action spaces are discrete and manageable in size.
- The algorithm is designed to converge to the optimal policy, even in non-deterministic environments, as long as each state-action pair is sufficiently explored.
- Exploration vs. Exploitation

Query Gsheets

In `standardised/GSheets` I want to use query, but I also want to remove certain rows based on a range of keys , can I do this ?

1. Use `FILTER` Inside `QUERY` (ArrayFormula Workaround)

Since `QUERY` does not support dynamic `NOT IN` , you can first filter out the excluded keys using `FILTER` , then pass the result to `QUERY` :

```
=QUERY(FILTER(A1:D, ISNA(MATCH(A1:A, X1:X10, 0))), "SELECT Col1, Col2, Col3, Col4", 1)
```

- `FILTER(A1:D, ISNA(MATCH(A1:A, X1:X10, 0)))` : Removes rows where column A matches any value in `X1:X10` .
- `QUERY(..., "SELECT Col1, Col2, Col3, Col4", 1)` : Runs a query on the filtered data.

Quartz

Vim: telescope? Search preview feature?

<https://www.youtube.com/watch?v=v5LGaczJaf0>

How does quartz work of a software level:

- Transforming text. Think [jinja template](#).
- Manipulating markdown notes

Introduction

- There is a diagram showing how markdown goes to html.
 - [JavaScript](#) for static site generators already existed.
-

Query Optimisation

[Querying](#) can be optimised for time, [space efficiency](#), and concurrency of queries.

Optimizing SQL [Querying](#):

- Timing queries
- [Database Index|Indexing](#)
- Managing [Transaction](#)
- and vacuuming, and handling concurrency with transactions and locks ensures efficient and reliable database performance.

Timing [Queries](#):

- Use `.timer on` to measure query execution time and identify slow queries.

[Database Index|Index](#) Search

Creating an index on specific columns can speed up searches:

- A covering index includes all the columns required by a query, eliminating the need to access the table data.
- Partial indexes cover a subset of rows, saving space while maintaining query performance for frequently accessed data i.e more likely to search movies that are recent.

Trade-offs when using indexes

- Indexes improve query speed but [consume additional space](#) and can slow down data insertion and updates.

To remove redundancy use [Transaction|Transactions](#)

Vacuum

SQLite's "VACUUM" command reclaims unused space after data deletion, reducing database size.

[Query Optimisation Tags](#): #performance_tuning, #querying

Query Plan

What is expected to happen to the query plan if there is [Database Index|Indexing](#)?

Querying

Querying is the process of asking questions of data. Querying makes use of keys primary and foreign within tables.

Useful Links

- [CS50 SQL Course](#)

In [DE_Tools](#) see:

Introduction

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLite/Querying/Querying.ipynb

SQL Commands and Examples

- SELECT
- LIMIT
- ORDER
- WHERE
- NOT
- LIKE
- WITH
- INSERT, UPDATE, or DELETE

You can have parameterised queries so that you can pass in variables to it:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLite/Querying/Parameterised_Questions.ipynb

Related terms:

- [SQL Joins](#)
- [SQL Injection](#):=Why we should not use f-strings in queries

Quicksort

Recursive Algorithm

Quicksort Algorithm in Five Lines of Code! - Computerphile

Fast algorithm (compared to say Insertion Sort)

1) Pick pivot value 2) Divide remaining numbers into two parts 3) sort sublists in some way <- apply alog again 4) merge

Recursion stops when nothing to pick for pivot value.

Introduction

```
def quick_sort(arr, depth=0):
    indent = " " * depth # Indentation for better readability in recursion
    print(f"{indent}Sorting: {arr}")

    if len(arr) <= 1:
        print(f"{indent}Returning sorted: {arr}")
        return arr # Base case: already sorted

    pivot = arr[len(arr) // 2] # Choosing pivot (middle element)
    left = [x for x in arr if x < pivot] # Elements smaller than pivot
    middle = [x for x in arr if x == pivot] # Elements equal to pivot
    right = [x for x in arr if x > pivot] # Elements greater than pivot

    print(f"{indent}Pivot: {pivot}")
    print(f"{indent}Left: {left}")
    print(f"{indent}Middle: {middle}")
    print(f"{indent}Right: {right}")

    sorted_left = quick_sort(left, depth + 1)
    sorted_right = quick_sort(right, depth + 1)

    sorted_array = sorted_left + middle + sorted_right
    print(f"{indent}Merged: {sorted_array}")

    return sorted_array # Recursively sort and merge

# Example usage:
arr = [10, 7, 8, 9, 1, 5]
print("Starting QuickSort...\n")
sorted_arr = quick_sort(arr)
print("\nFinal sorted array:", sorted_arr)
```

R

Table of Contents

- [R squared](#)
- [R-squared metric not always a good indicator of model performance in regression](#)
- [R](#)
- [RAG](#)
- [REST API](#)
- [ROC \(Receiver Operating Characteristic\)](#)
- [ROC_Curve.py](#)
- [Race Conditions](#)
- [Random Forest Regression](#)
- [Random Forests](#)
- [React](#)
- [Reasoning tokens](#)
- [Recall](#)
- [Recommender systems](#)
- [Recurrent Neural Networks](#)
- [Recursive Algorithm](#)
- [Regression Metrics](#)
- [Regression](#)
- [Regression_Logistic_Metrics.ipynb](#)
- [Regularisation of Tree based models](#)
- [Regularisation](#)
- [Regularisation.py](#)
- [Reinforcement learning](#)
- [Relating Tables Together](#)
- [Relational Database](#)
- [Relationships in memory](#)
- [Reward Function](#)
- [Ridge](#)
- [Row-based Storage](#)
- [requirements.txt](#)
- [reverse etl](#)
- [rollup](#)

R Squared

R^2 , or the coefficient of determination, measures the proportion of variance in the dependent variable that is explained by the independent variables in a regression model.

Interpretation:

- R^2 values range from 0 to 1.
- A value of 1 indicates perfect predictions, meaning the model explains all the variability of the response data around its mean.
- Higher R^2 values signify a better fit of the model to the data. However, it can be misleading when adding more predictors, as R^2 will never decrease when more variables are added to a model. See [Adjusted R squared](#).

Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Where:

- y_i = actual values
- \hat{y}_i = predicted values
- \bar{y} = mean of the actual values

Example:

An R^2 of 0.60 indicates that 60% of the variability observed in the target variable is explained by the regression model.

Follow up Questions

- R-squared metric not always a good indicator of model performance in regression

R Squared Metric Not Always A Good Indicator Of Model Performance In Regression

R-squared (R^2) is a commonly used metric for assessing the performance of regression models, but it is not always a reliable indicator of model quality. It should not be the sole criterion for evaluating model performance. It is essential to consider other metrics, such as [Adjusted R squared](#), [Cross Validation](#) results, and the overall context of the analysis.

1. **Increased Complexity:** R^2 will never decrease when more predictors are added to a model, even if those predictors do not have a meaningful relationship with the dependent variable. This can lead to overfitting, where the model captures noise rather than the underlying data pattern.
2. **Lack of Context:** A high R^2 value does not necessarily imply that the model is appropriate for prediction. It only indicates the proportion of variance explained. A model with a high R^2 might still have poor predictive performance if it does not generalize well to new data.
3. **Non-linearity:** R^2 assumes a linear relationship between the independent and dependent variables. If the true relationship is non-linear, R^2 may provide a false sense of model adequacy.
4. **Ignoring Model Assumptions:** R^2 does not account for whether the assumptions of the regression model (such as homoscedasticity, independence, and normality of residuals) are met. A model may have a high R^2 but still violate these assumptions, leading to unreliable results.
5. **Adjusted R²:** To address some of these issues, Adjusted R^2 is often used, which adjusts the R^2 value based on the number of predictors in the model. It provides a more accurate measure of model performance when comparing models with different numbers of predictors.

R

Rag

Rag is a framework the help [LLM](#) be more up to date.

RAG grounds the Gen AI in external data.

[!Summary] Given a question sometimes the answer given is wrong, issue with [LLM](#) is no source of data and is out of date. **RAG** is a specific architecture used in natural language processing ([NLP](#)), where a **retrieval mechanism** is combined with a **generative model** ([Generative](#)) (often a [Transformer](#) like GPT). RAG systems are designed to **enhance the ability of a generative model to answer questions or generate content by incorporating factual knowledge retrieved from external data sources** (such as documents, databases, or knowledge repositories). RAG is the connection of [LLM](#)'s with external databases.

[!Example] **Example of a RAG System:**

- A user asks: "*What is the capital of France?*"
- The **retrieval module** fetches a relevant document (e.g., from Wikipedia) that contains the information about France's capital.
- The **generation module** synthesizes the response: "The capital of France is Paris."

[LLM](#) Challenges

- Responses are sometimes no sources and out of date.
- LLM's are trained on some store of data (static). We want this store to be updated.

Key characteristics of RAG:



Based on a [Prompting](#).

1. Retrieval Component:

- This module fetches relevant documents (and up to date) or information from an external corpus based on the query or input. It may use traditional search methods like dense vector retrieval (e.g., using embeddings) or keyword-based retrieval.
- Retriever should be good enough to give the most truthful information based on the store

2. Generative Component:

- After retrieving relevant documents, the [Generative](#) model (such as GPT or [BERT](#)-based models) synthesizes the final response, integrating both the input query and the retrieved information to generate more accurate and contextually informed outputs.

3. Augmentation with External Knowledge:

- Instead of solely relying on pre-trained internal knowledge (as in traditional language models), RAG setups use the external knowledge source for augmenting generation, thus improving factual accuracy and reducing the risk of hallucinations (incorrect or fabricated responses).

Model should be able to say "I don't know" instead of [hallucinating](#)

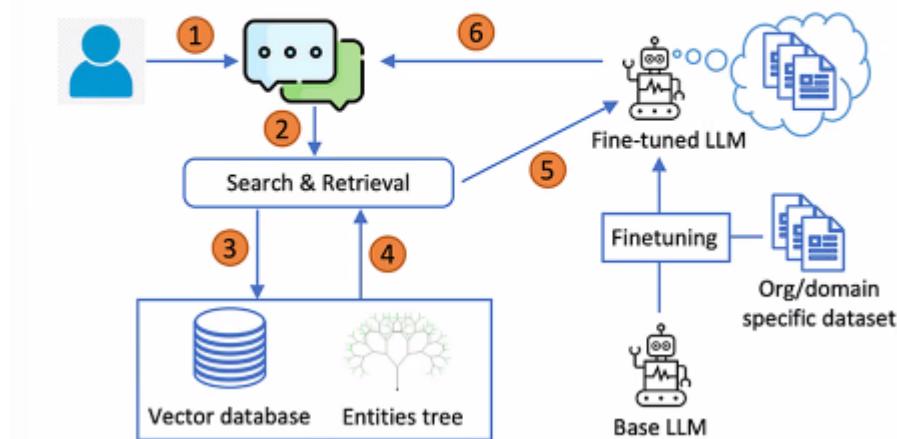
Resources

RAG Problems

1. LLMs struggle with memorization > "LLMs may struggle with tasks that require domain-specific expertise or up-to-date information."
2. LLMs struggle with generating factually inaccurate content (hallucinations) Solution · A lightweight retriever (SM) to extract relevant document fragments from external knowledge bases, document collections, or other tools

Different RAG techniques: Sparse retrievers, BM25, dense retrievers. Use Bert for similarity matching:

T-RAG (Pan et al., 2022): Inclusion of an entities tree in addition to the vector database for context retrieval.



REST API

- REST stands for Representational State Transfer.
- It is a **standardized** software architecture style used for API communication between a client and a server.

Benefits of REST APIs:

1. **Simplicity and Standardization:**
 - Data formatting and request structuring are standardized and widely adopted.
2. **Scalability and Statelessness:**
 - Easily modifiable as service complexity grows without tracking data states across client and server.
3. **High Performance:**
 - Supports **caching**, maintaining high performance even as complexity increases.

Main Building Blocks:

1. **Request:**
 - Actions (**CRUD**): Create (POST), Read (GET), Update (PUT), Delete (DELETE).
 - Components: Operation (**HTTP method**), Endpoint, Parameters/Body, Headers.
2. **Response:**
 - Typically in **JSON** format.

REST API Example: ice cream shop interacting with cloud database.

- Endpoint example: "icecream.com/api/flavors"
 - "api" indicates the API portion.

- "flavors" refers to the **resource** being accessed or modified.

Real-world Examples:

1. Get Flavors:

- Operation: **GET**
- Endpoint: "/api/flavors"
- Response: Array of flavor resources.

2. Update Flavor:

- Operation: **PUT**
- Endpoint: "/api/flavors/1"
- Body: New flavor data.
- Response: Confirmation of update.

3. Create New Flavor:

- Operation: **POST**
- Endpoint: "/api/flavors"
- Body: New flavor data.
- Response: Confirmation of creation.

Roc (Receiver Operating Characteristic)

ROC (Receiver Operating Characteristic) is a graphical representation of a classifier's performance across different thresholds, showing the trade-off between sensitivity (true positive rate) and specificity (1 - false positive rate).

A graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

It plots the true positive rate (TPR) against the false positive rate (FPR) at different threshold settings.

In [ML_Tools](#) see: [ROC_Curve.py](#)

Why Use Predicted Probabilities?

In ROC analysis, predicted probabilities (`y_probs`) are used instead of predicted classes (`y_pred`) because the ROC curve evaluates the model's performance across different threshold levels. Probabilities allow you to adjust the threshold to see how it affects sensitivity and specificity.

Threshold Level

The threshold level is the probability value above which an instance is classified as the positive class. Adjusting the threshold affects [Recall](#) and [Specificity](#)

- Lower Threshold: Increases sensitivity but may decrease specificity.
- Higher Threshold: Increases specificity but may decrease sensitivity.

Example Code

```

from sklearn.metrics import roc_curve, RocCurveDisplay
import matplotlib.pyplot as plt

# Actual and predicted values
y_act = [1, 0, 1, 1, 0]
y_pred = [1, 1, 0, 1, 0]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_act, y_pred)

# Display ROC curve
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.show()

```

Logistic Regression Example

```

from sklearn.linear_model import LogisticRegression

# Train a logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predict probabilities for the positive class
y_probs = logreg.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--', lw=2, label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend()
plt.show()

```

Roc_Curve.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Selection/ROC_Curve.py

Overview

This script demonstrates how to compute and interpret Receiver Operating Characteristic (ROC) curves and Area Under the ROC Curve (AUROC) scores using Random Forest and Naive Bayes classifiers. Below is the step-by-step breakdown:

Script Flow

- **Generate Synthetic Dataset**
 - Creates a binary classification dataset with 2,000 samples and 10 features.
 - Simulates a realistic classification problem.
- **Add Noisy Features**
 - Appends random, irrelevant features to increase the dataset's complexity.
 - Mimics challenging real-world scenarios where not all features are informative.
- **Split the Data**
 - Divides the dataset into training (80%) and testing (20%) subsets.
 - Ensures unbiased model evaluation on unseen data.
- **Train Classification Models**
 - Builds two models:
 - **Random Forest**: A robust ensemble-based classifier.
 - **Naive Bayes**: A simple probabilistic model based on Bayes' theorem.
- **Generate Prediction Probabilities**
 - Computes predicted probabilities for each class.
 - Retains probabilities for the positive class to construct the ROC curve.
- **Compute AUROC and ROC Curve Values**
 - Calculates:
 - **AUROC**: Measures model performance (higher is better).
 - **ROC Values**: False Positive Rate (FPR) and True Positive Rate (TPR) across thresholds.
- **Visualize ROC Curve**
 - Plots FPR (x-axis) against TPR (y-axis) for each model.
 - Includes AUROC scores in the legend for comparison.

Key Outputs

- **AUROC Scores**
 - Evaluates the overall discriminative power of the classifiers.
 - **ROC Plot**
 - Visualizes how well each model distinguishes between positive and negative classes across thresholds.
 - A random prediction baseline is included for reference.
-

Conclusion

This script illustrates the process of building, evaluating, and visualizing classification models using ROC curves. It highlights the strengths and weaknesses of different models in distinguishing classes.

Output

Interpretation of the Script Output

- **Random (Chance) Prediction: AUROC = 0.500**
 - An AUROC score of **0.500** represents a random guessing model with no predictive power.
 - The model's True Positive Rate (TPR) is equal to its False Positive Rate (FPR) across all thresholds, resulting in a diagonal line on the ROC curve.
- **Random Forest: AUROC = 0.922**
 - An AUROC score of **0.922** indicates excellent model performance.
 - The Random Forest classifier has a high ability to distinguish between positive and negative classes, with a much higher TPR than FPR across thresholds.
-
- **Naive Bayes: AUROC = 0.993**
 - An AUROC score of **0.993** suggests near-perfect model performance.
 - The Naive Bayes classifier has an extremely high discriminative power, with TPR approaching 1 and FPR close to 0 for most thresholds.

Summary

- The **Naive Bayes classifier** outperforms the **Random Forest classifier** in this specific setup.
- Both models significantly outperform random guessing (baseline AUROC = 0.500), indicating their utility for this classification task.
- However, such high performance (especially for Naive Bayes) may suggest that the dataset or features are particularly well-suited to the model, or there may be minimal noise in the classification task. Further evaluation (e.g., on new datasets) is recommended to confirm robustness.

Race Conditions

Random Forest Regression

Random Forest Regression : Like random forests for classification, random forest regression combines multiple regression trees to improve prediction accuracy.

Random Forests

A random forest is an [Model Ensemble](#) of [Decision Trees](#). Take many decision trees decisions to get better result.

What is the Random Forest method;; an ensemble learning method based on constructing multiple decision trees during training and combining their predictions through averaging. Random Forests are flexibility, robustness, and ability to handle high-dimensional data, as well as their resistance to overfitting.

Introduction

What is an issue with [Random Forests](#); susceptible to overfitting, especially when dealing with noisy or high-dimensional data. Proper tuning of hyperparameters like the number of trees and maximum depth is crucial to mitigate this.

Random forests combine multiple decision trees to improve accuracy and generalisation.

What is Random Forest, and how does it work?; Random Forest is a method that can perform regression, classification, dimensionality reduction, and handle missing values. It builds multiple decision trees and combines their outputs. Each tree is grown using a subset of the data and features, and the final output is determined by aggregating the predictions of individual trees.

Remember that for a Random Forest, we randomly choose a subset of the features AND randomly choose a subset of the training examples to train each individual tree.

if n is the number of features, we will randomly select \sqrt{n} of these features to train each individual tree.

- Note that you can modify this by setting the `max_features` parameter.

You can also speed up your training jobs with another parameter, `n_jobs`.

- Since the fitting of each tree is independent of each other, it is possible fit more than one tree in parallel.
- So setting `n_jobs` higher will increase how many CPU cores it will use. Note that the numbers very close to the maximum cores of your CPU may impact on the overall performance of your PC and even lead to freezes.
- Changing this parameter does not impact on the final result but can reduce the training time.



What is an issue with [Random Forests](#); susceptible to overfitting, especially when dealing with noisy or high-dimensional data. Proper tuning of hyperparameters like the number of trees and maximum depth is crucial to mitigate this.

[Decision Tree](#) are not the best - need to make flexible for new data. They work well with the data set they are defined on.

How to proceed with random forest: (build tree's randomly) i.e solve the issue with decision trees. Process is called [Bagging](#) 1) randomly select a dataset (bootstrap) 2) randomly select two (or multiple) features for each branch and proceed like in decision tree.

variety makes trees better.

To make a prediction , run data through trees in forest, and get prediction, conclude with majority prediction.

How to know if random forest is good ?

Use data that was not in bootstrap data set - measure the accuracy based on these classifications.

Refine the random forest by tweaking the [Hyperparameter](#) of number of features used per step.

React

React is a [JavaScript](#) library developed by Meta for building user interfaces, particularly in web development.

Related to:

- [Dashboarding](#)

Core Concepts

React's component-based architecture allows for reusable UI elements, enhancing maintenance and testing. It uses a Virtual DOM for efficient updates, minimizing direct DOM manipulation. Data flows unidirectionally from parent to child components.

Main Use Cases

React is ideal for Single Page Applications (SPAs) that load once and update dynamically, as well as for complex, interactive user interfaces and real-time applications like dashboards.

Common Tools

Popular UI libraries include Tailwind CSS and shadcn/ui.

Reasoning Tokens

Transformers rely on pattern recognition and language-based reasoning.

Thus, **reasoning tokens serve as a mechanism for token-based logical progression**, allowing models like ChatGPT to simulate math insights by leveraging pattern recognition, token relationships, and sequential reasoning, even without explicit symbolic or mathematical processing built into the model itself.

[Mathematical Reasoning in Transformers](#)

In the context of models like ChatGPT, **reasoning tokens** refer to the individual pieces of language that contribute to the step-by-step logical process used by the model to solve problems, including mathematical ones.

5. Logical Continuity and Error Correction

- Reasoning tokens enable the model to maintain **logical continuity**, allowing it to backtrack or adjust outputs based on the sequence of previously generated tokens. For example, if the model makes a mistake in an earlier step (like a miscalculation), it can revise its response as it generates subsequent tokens that recognize the inconsistency.

Recall

Recall Score is a [Evaluation Metrics](#) used to evaluate the performance of a [Classification](#) model, focusing on the model's ability to **identify all relevant instances of the positive class**.

It answers the question: **How many relevant items are retrieved?**

High recall means that the model is effective at identifying most of the actual spam emails. This is useful in environments where missing a spam email could lead to security risks such as in corporate email systems.

The formula for recall is:

$$\text{Recall} = \frac{TP}{TP+FN}$$

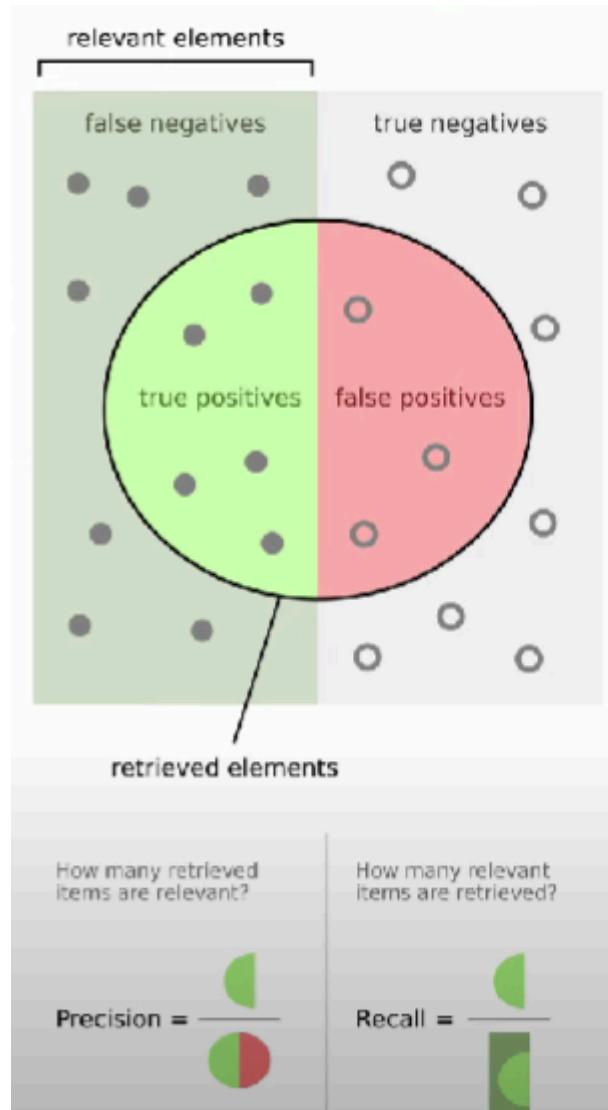
Importance

- Recall is crucial in scenarios where **missing a positive instance is costly**, such as in disease screening or fraud detection.

- It helps in understanding how well the model captures all the actual positive instances.

Related Concepts

- **Sensitivity** (also known as recall or the true positive rate) measures the proportion of actual positives that are correctly identified by the model. It indicates how well the model is at identifying positive instances.



Recommender Systems

Crab on Python

A recommender system, or recommendation system, is a type of information filtering system that aims to predict the preferences or interests of users by analyzing their behavior and the behavior of similar users or items. These systems are widely used in various applications, such as e-commerce, streaming services, social media, and content platforms, to provide personalized recommendations to users.

Key Components of Recommender Systems:

1. **User Data:** Information about users, such as their preferences, ratings, purchase history, and interactions with items.

-
- 2. **Item Data:** Information about the items being recommended, which can include attributes, descriptions, and metadata.
 - 3. **Recommendation Algorithms:** The methods used to generate recommendations. Common approaches include:
 - **Collaborative Filtering:** This technique relies on the behavior and preferences of similar users. It can be user-based (finding similar users) or item-based (finding similar items).
 - **Content-Based Filtering:** This approach recommends items based on the features of the items and the preferences of the user. For example, if a user likes action movies, the system will recommend other action movies based on their attributes.
 - **Hybrid Methods:** Combining collaborative and content-based filtering to improve recommendation accuracy and overcome limitations of each method.
 - 4. **Evaluation Metrics:** Metrics used to assess the performance of the recommender system, such as precision, recall, F1 score, and mean average precision.

Applications of Recommender Systems:

- **E-commerce:**
- **Streaming Services:**
- **Social Media:**
- **News and Content Platforms:**

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of [neural network](#) designed to process sequential data by maintaining a memory of previous inputs through hidden states. This makes them suitable for tasks where the order of data is needed, such as:

- [Time Series](#) prediction,
- speech recognition,
- and [NLP|natural language processing](#) (NLP).

RNNs have loops in their architecture, [allowing information to persist across sequence steps](#). However, they face challenges with long sequences due to [vanishing and exploding gradients problem](#). To address these issues, variants like Long Short-Term Memory ([LSTM](#)) and [Gated Recurrent Unit](#) (GRU) have been developed.

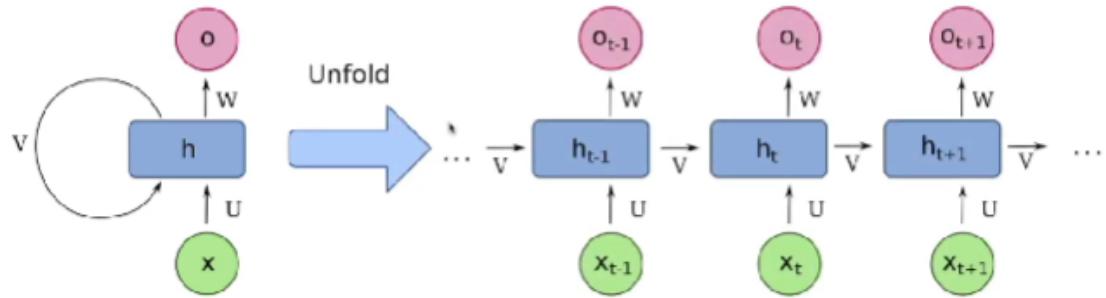
Resources:

Video link <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>

Key Concepts of RNNs

Sequential Data Handling

- RNNs maintain a hidden state that acts as memory, enabling them to model temporal dependencies. This is essential for tasks where the current output depends on both current and previous inputs.
- At each time step, RNNs process an input, combine it with the previous hidden state, and produce an output along with an updated hidden state.
- The hidden state carries forward information influenced by all previous inputs, theoretically allowing RNNs to remember long-term dependencies.



Backpropagation Through Time (BPTT)

- RNNs are trained using BPTT, a variant of backpropagation. The network unrolls over time, treating each time step as a layer in a deep network.
- Gradients are computed for each time step, and weights are updated based on cumulative error across the sequence. This allows learning of long-term dependencies but can lead to vanishing and exploding gradients in long sequences.

Variants of RNNs

- **LSTM**: Introduces gates (input, forget, output) to control information flow, addressing vanishing gradients and improving long-sequence handling.
- **GRU**: A simpler variant of LSTM with fewer parameters, offering efficiency and ease of training while maintaining performance on sequence tasks.

Example Code (RNN in Python with PyTorch)

See `RNN_Pytorch.py`

Problem of Long Term Dependencies

The more time steps we include the less data we keep from the past.

Solutions: **LSTM** and **GRU**: use gates: but are costly in computation.

Problem of Long-Term Dependencies



Other areas

Use of RNNs in energy sector

RNNs and Transformer|Transformers

Why have Transformer|Transformer's have replaced traditional RNN.

Inductive Reasoning, Memories and Attention.

How to address the limitations of vanilla recurrent networks.

Issues:

- RNNs are not inductive: They memorize sequences extremely well, but don't generalise well.
- They couple their representation size to the amount of computation per step.

Recursive Algorithm

Regression Metrics

These metrics provide various ways to evaluate the performance of regression models.

Evaluating Regression Models

These metric provide:

1. **Comprehensive Evaluation:** Each metric provides a different perspective on model performance. For example, while MSE and RMSE give insights into the average error magnitude, MAE provides a straightforward average error measure, and R-squared indicates how well the model explains the variance in the data.

2. **Sensitivity to standardised/Outliers:** Metrics like MSE and RMSE are sensitive to outliers due to the squaring of errors, which can be useful if you want to emphasize larger errors. In contrast, MAE and Median Absolute Error are more robust to outliers.
3. **Interpretability:** RMSE is in the same units as the target variable, making it easier to interpret in the context of the data. This can be particularly useful for stakeholders who need to understand the model's performance in practical terms.
4. **Model Comparison:** These metrics allow you to compare different models or configurations to determine which one performs best on your data.
5. **Variance Explanation:** R-squared and Explained Variance Score provide insights into how much of the variability in the target variable is captured by the model, which is crucial for understanding the model's effectiveness.

Common Regression Metrics

Mean Absolute Error (MAE):

- Definition: MAE measures the average absolute differences between predicted and actual values.
- Interpretation: Lower values indicate better model performance, as it reflects fewer errors in predictions.
- Formula: $MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$
- Where:
 - n = number of observations
 - y_i = actual value
 - \hat{y}_i = predicted value

Mean Squared Error (MSE):

- Definition: MSE calculates the average of the squares of the errors (the differences between predicted and actual values).
- Interpretation: Like MAE, lower values are better. However, MSE is more sensitive to outliers due to the squaring of errors, which can disproportionately affect the metric. Greater error values are exaggerated.
- Formula: $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
- Where:
 - n = number of observations
 - y_i = actual value
 - \hat{y}_i = predicted value

Root Mean Squared Error (RMSE):

- Definition: RMSE is the square root of MSE, providing an error metric in the same units as the target variable.
- Interpretation: Lower RMSE values indicate better model performance, and it also emphasizes larger errors due to the squaring process. Easier to interpret ([interpretability](#)), back to the same scale as the input.
- Formula: $RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$

R squared

Adjusted R squared

Median Absolute Error:

- Definition: This metric measures the median of the absolute errors between predicted and actual values.
- Interpretation: It provides a robust measure of prediction accuracy, especially in the presence of standardised/Outliers.
- Formula: $\text{MedAE} = \text{median}(|y_i - \hat{y}_i|)$

Explained Variance Score:

- Definition: This metric measures the proportion of variance in the target variable that is predictable from the features.
- Interpretation: Higher values indicate that the model explains a greater proportion of the **Variance** in the target variable.
- Formula: $\text{Explained Variance} = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}$
- Where:
 - $\text{Var}(y)$ = variance of the actual values
 - $\text{Var}(y - \hat{y})$ = variance of the prediction errors

Implementation

See Regression_Metrics.py

Regression

[!Summary]

Regression analysis is a statistical method used to **predict** a continuous variable based on one or more predictor variables. The most common form, **Linear Regression**, assumes a linear relationship between the dependent variable y and independent variables x_1, x_2, \dots, x_n . The goal is to minimize the residual sum of squares (RSS) between observed and predicted values. Other forms, such as **Logistic Regression**, handle classification problems.

Regression models can incorporate techniques like regularization (L_1 , L_2) to improve performance and prevent overfitting, especially with high-dimensional data. Advanced methods like **Polynomial Regression** address non-linearity, while generalized linear models (GLMs) extend regression to non-normal response variables.

Regressor: This is a type of model used for regression tasks, where the goal is to predict continuous values. For example, a regressor might be used to predict the price of a house based on its features, or to forecast future sales figures.

[!Breakdown]

Key Components:

- **Linear Regression:** Predicts $y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \epsilon$, where ϵ is the error term.
- Regularization: Adds L_1 (**Lasso**) or L_2 (**Ridge**) penalty to prevent overfitting in high-dimensional data.
- Feature transformation: **Polynomial Regression** and logarithmic transformations adjust for non-linearity in data.
- Regression is a type of **Supervised Learning**.

[!important]

- R^2 is a key metric, showing how much of the variance in y is explained by x .
- **Multicollinearity** can inflate variances of coefficient estimates, harming model reliability.

[!attention]

- Regression assumes **linearity**, so improper application to non-linear data can lead to biased predictions.
- Overfitting can occur with too many predictors, especially in small datasets.

[!Example]

In predicting insurance claims, a linear regression model could take input variables like age and driving history to estimate the expected claim amount. A transformation, such as logarithmic scaling, could address any non-linear patterns between variables.

[!Follow up questions]

- How can [Polynomial Regression](#) improve predictions in non-linear datasets?
 - What are the benefits of combining [Linear Regression](#) with [Feature Engineering](#) for complex datasets?
-

Regression_Logistic_Metrics.ipynb

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Selection/Model_Evaluation/Classification/Regression_Logistic_Metrics.ipynb

Regularisation Of Tree Based Models

Tree models, such as Random Forests and Gradient Boosting, can also be regularized, although they don't use L1 or L2 regularization directly. Instead, they are regularized through hyperparameters like max depth, min samples split, and learning rate to control the complexity of the trees.

In tree-based models, regularization is not applied in the same way as it is in linear models (i.e., using L1 or L2 penalties).

In tree models, [Regularisation](#) is done by controlling the growth of the trees using [hyperparameters](#) like

- `max_depth`,
- `min_samples_split`,
- `min_samples_leaf`.

These hyperparameters restrict the growth of the tree, preventing it from becoming too complex.

For [Model Ensemble](#) methods like Random Forests and Gradient Boosting, additional regularization techniques like

- subsampling,
- bootstrap sampling,
- and learning rate control

to help prevent overfitting. These techniques effectively restrict the model complexity, leading to better generalization .

Below are the common regularization techniques used in tree models such as [Decision Tree](#), [Random Forests](#).

Regularization in Different Tree Models

- Decision Trees: Prone to overfitting when not regularized, since they tend to grow large and complex trees. Regularization through pruning, limiting tree depth, and controlling minimum samples per split or leaf is critical.
- Random Forests: Regularization is mainly achieved through the use of multiple decision trees, random feature selection (`max_features`), and bootstrapping (`bootstrap`). Each tree learns a different part of the data, which reduces overfitting.
- Gradient Boosting Models (GBMs): Regularized by tuning the `learning_rate`, `subsample`, and controlling the tree depth and other tree-based hyperparameters like `min_samples_split`. The slower learning process with a smaller learning rate combined with these hyperparameters helps prevent overfitting.

Regularization Techniques for Tree Models

- Limiting Tree Depth:
-

- Max Depth (`max_depth`): This parameter restricts the maximum depth of the tree. Trees that are too deep can model complex patterns, but they are prone to overfitting. By limiting the depth, you constrain the tree's ability to learn highly specific patterns in the training data.
- Example: In scikit-learn, setting `max_depth` for a Decision Tree or Random Forest.

```
from sklearn.tree import DecisionTreeClassifier

# Limit tree depth to regularize the model
model = DecisionTreeClassifier(max_depth=5)
model.fit(X_train, y_train)
```

- Minimum Samples for Splitting:

- Min Samples Split (`min_samples_split`): This parameter specifies the minimum number of samples required to split an internal node. Increasing this value makes the tree more conservative and prevents it from splitting when there are too few samples, thus controlling its complexity.
- This helps avoid creating splits based on noise, which could lead to overfitting.

```
model = DecisionTreeClassifier(min_samples_split=10)
model.fit(X_train, y_train)
```

- Minimum Samples per Leaf:

- Min Samples Leaf (`min_samples_leaf`): This parameter sets the minimum number of samples a node must have after a split to be a leaf. Higher values result in fewer splits, producing simpler trees that are less likely to overfit.
- This also encourages broader splits that require more data, reducing the sensitivity to outliers.

```
model = DecisionTreeClassifier(min_samples_leaf=4)
model.fit(X_train, y_train)
```

- Max Number of Features:

- Max Features (`max_features`): This controls the number of features to consider when looking for the best split. Reducing the number of features makes the model less likely to overfit, as it limits the search space for splits. For Random Forests, this also introduces randomness that can improve generalization.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(max_features='sqrt') # Uses sqrt of total features
model.fit(X_train, y_train)
```

- Max Leaf Nodes:

- Max Leaf Nodes (`max_leaf_nodes`): This parameter limits the total number of leaf nodes the tree can have. Fewer leaf nodes result in simpler trees that are less likely to overfit the training data.

```
model = DecisionTreeClassifier(max_leaf_nodes=10)
model.fit(X_train, y_train)
```

- Subsampling (for Ensemble Methods like Random Forests and Gradient Boosting):

- Bootstrap Sampling (`bootstrap`): For Random Forests, regularization is achieved through bootstrapping (random sampling with replacement) during training. This introduces variability and helps prevent overfitting.
- Subsample (`subsample`): For Gradient Boosting, the `subsample` parameter controls the fraction of the training data used for fitting each individual tree. A value less than 1 introduces randomness and reduces the chance of overfitting, similar to how dropout works in neural networks.

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier(subsample=0.8) # Use 80% of data for each tree
model.fit(X_train, y_train)
```

- Learning Rate (for Gradient Boosting Models):

- Learning Rate (`learning_rate`): This parameter controls how much each tree contributes to the overall model in Gradient Boosting. A lower learning rate slows down the learning process, requiring more trees but helping to avoid overfitting by making small adjustments at each step.

```
model = GradientBoostingClassifier(learning_rate=0.1)
model.fit(X_train, y_train)
```

- Pruning:

- For Decision Trees, pruning is a post-processing regularization technique where branches that contribute little to the overall performance of the model are removed. This prevents the tree from learning noise in the data.
- In scikit-learn, Cost Complexity Pruning (`ccp_alpha`) is used for pruning. A larger value of `ccp_alpha` leads to more aggressive pruning, simplifying the tree.

```
model = DecisionTreeClassifier(ccp_alpha=0.01)
model.fit(X_train, y_train)
```

Regularisation

Regularization is a technique in machine learning that reduces the risk of overfitting by adding a penalty to the [Loss function](#) during model training. This penalty term restricts the magnitude of the model's parameters, thereby controlling the complexity of the model. It is especially useful in linear models but can also be applied to more complex models like neural networks.

Key Concepts

- **\$L_1\$ Regularization ([Lasso](#)):** Adds the absolute value of the coefficients to the loss function, encouraging sparsity by driving some coefficients to zero, effectively selecting a subset of features.
- **\$L_2\$ Regularization ([Ridge](#)):** Adds the square of the coefficients to the loss function, shrinking them toward zero. It encourages smaller coefficients but does not push them exactly to zero, helping reduce overfitting by penalizing large weights.
- **Elastic Net:** Combines both Lasso and Ridge regularization.

Benefits

- **Prevents Overfitting:** Regularization adds a penalty term to the loss function to avoid overfitting.
- **Feature Sparsity:** L_1 encourages feature sparsity, while L_2 reduces coefficient magnitudes.
- **Enhanced Generalization:** Dropout enhances generalization by preventing unit co-adaptation in neural networks.

Considerations

- **Underfitting Risk:** Over-penalizing parameters can lead to underfitting, where the model becomes too simplistic.
- **Tuning λ :** Choosing the right penalty term (i.e., λ) is crucial for balancing bias and variance.

[When and why not to us regularisation](#)

Questions

- How does the balance between L_1 and L_2 regularization impact model performance in large feature spaces?
- What are the best practices for tuning the λ parameter in regularization? [Model Parameters Tuning](#).

Example

Consider a linear regression model with L_2 regularization (Ridge). The [loss function](#) would be:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2$$

Here, λ controls the strength of the regularization. Higher λ values shrink the coefficients more.

Related Topics

- [Feature Selection](#): L1 regularization can zero out irrelevant features, improving model [interpretability](#) and reducing computational costs.
- [Model Selection](#) techniques for high-dimensional data.

Applications

Regularization is widely used in linear models but is also applied in other machine learning models, particularly those prone to overfitting:

- [Neural network](#)
- [Regularisation of Tree based models](#)

Implementation

In [ML_Tools](#) see: [Regularisation.py](#)

Regularisation.Py

https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Optimisation/Regularisation/Regularisation.py

Reinforcement Learning

Reinforcement Learning ([Reinforcement learning|RL](#)) is a branch of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives rewards or penalties based on its actions, and its goal is to [maximize cumulative reward](#).

Current Research Problems

1. Sample Efficiency: Many RL algorithms require a large number of interactions with the environment to learn effectively. Research is focused on developing methods to improve sample efficiency, such as model-based approaches and transfer learning.
2. [Exploration vs. Exploitation](#): Balancing exploration (trying new actions) and exploitation (choosing known rewarding actions) remains a challenge. New strategies, such as curiosity-driven learning and bandit algorithms, are being investigated.
3. [Multi-Agent Reinforcement Learning](#): Extending RL to environments with multiple interacting agents introduces complexity in learning optimal strategies. Research includes coordination, competition, and communication between agents.
4. Robustness and Stability: Ensuring that RL agents perform reliably in changing or adversarial environments is a key area of study. Techniques for robust control and stability analysis are being explored.

Algorithms in Reinforcement Learning

[Q-Learning](#) [Deep Q-Learning](#) [Sarsa](#)

Components

- Agent: An entity that interacts with the environment and learns to optimize its actions based on rewards.
- State (\$s\$): The current situation in the environment, often defined by the positions and attributes relevant to the agent's decision-making.
- Action (\$a\$): The available moves or decisions an agent can take.
- Reward (\$r\$): A scalar value received after taking an action, representing feedback from the environment.
- [Policy](#) (\$\pi\$): A strategy that the agent follows, mapping states to actions.
- Q-Value (\$Q(s, a)\$): The expected cumulative reward for taking a particular action in a given state and following the policy thereafter. The Q-values guide the agent in making decisions that maximize long-term rewards. [Q-Learning](#)

Mathematical Foundations

- [Markov Decision Processes](#)
- Dynamic Programming: Techniques such as [Bellman Equations](#) equations are central to RL, as they provide a way to break down complex decision-making problems into simpler subproblems.
- Optimization Techniques: Finding optimal [Policy|polices](#) often involves advanced optimization methods, including gradient ascent and policy iteration.

Reinforcement Learning Implementation

Use import gym

Introduction

Action Space: What actions are available to the agent? Observation Space: What information is available to the agent? Reward Environment: What rewards can be given to the agent?

Loads the environment for examples env = gym.make('LunarLander-v2')

Can step through the environment dynamics next_state, reward, done, info = env.step(action)

```
# Select an action
action = 0

# Run a single time step of the environment's dynamics with the given action.
next_state, reward, done, info = env.step(action)

with np.printoptions(formatter={'float': '{:.3f}'.format}):
    print("Initial State:", initial_state)
    print("Action:", action)
    print("Next State:", next_state)
    print("Reward Received:", reward)
    print("Episode Terminated:", done)
    print("Info:", info)
```

Relating Tables Together

Implementing these concepts, database tables can be effectively related, ensuring [Data Integrity](#), efficient retrieval, and easy maintenance.

Resources:

- [LINK](#)

Notes on Relating Database Tables

[Primary Key](#)

[Foreign Key](#)

One-to-One Relationships:

- Each record in Table A relates to one record in Table B and vice versa.
- Example: Each employee has one unique profile.

One-to-Many Relationships:

- A single record in Table A can relate to multiple records in Table B.
- Example: One department has many employees.

[Many-to-Many Relationships](#)

[Junction Tables](#)

- Used to manage many-to-many relationships.
- Contains foreign keys from both tables it connects.
- Example: Enrollments table with StudentID and CourseID as foreign keys.

[Referential Integrity](#)

Introduction

- Ensures that relationships between tables remain consistent.
 - Example: If an employee is assigned a department, the DepartmentID must exist in the Departments table.
-

Cascading Actions:

- Cascade Update: Updates related records automatically when a primary key is updated.
- Cascade Delete: Deletes related records automatically when a primary key is deleted.

ER Diagrams

Relational Database

Relationships In Memory

In managing the memory of a large language model (LLM), several key concepts and techniques play a crucial role in forming and maintaining relationships between data points:

RAG (Retrieval-Augmented Generation):

This technique enhances LLMs by integrating external data retrieval with generative capabilities. By employing chunking and reranking, RAG refines outputs, ensuring that the model can access and utilize relevant information efficiently. This process strengthens the model's ability to form and maintain relationships between different pieces of information, improving its memory and response accuracy.

Ontology and Correlating Data Points:

Ontologies establish [semantic relationships](#) between data points by defining a structured framework of concepts and their interrelations. This structured understanding aids in memory management by providing a clear map of how different pieces of information are related. Similarly, correlating data points involves understanding and forming connections, which is essential for enhancing memory management. Together, these approaches help LLMs organize and interpret information more effectively.

Vector Store and Metadata Management:

Utilizing [Vector Database](#) vector databases allows for efficient storage and retrieval of memory representations. These databases preserve the relationships between data points, enabling LLMs to access and utilize memory more effectively. Alongside this, managing metadata is crucial for organizing, retrieving, and correlating data points. Effective metadata management helps LLMs understand the context and relationships between different pieces of information, enhancing their memory capabilities.

Structure Memory Graph: [GraphRAG](#)

Organizing memory in graph structures allows LLMs to improve relational understanding and connection-making. Graphs provide a visual and structural representation of how information is interconnected, aiding the model's ability to form and maintain complex relationships in memory.

Cognitive Sciences:

Insights from cognitive science inform memory design and improve human-AI interaction. By integrating these insights, LLMs can mimic human-like memory processes, enhancing their ability to form, maintain, and retrieve relationships in memory, leading to more natural and effective interactions.

Reward Function

Recurrent Neural Networks|RNN

Ridge

L2 Regularization, also known as Ridge Regularization, adds a penalty term proportional to the square of the weights to the [loss function](#).

This technique enhances the robustness of linear regression models (and [Logistic Regression](#)) by penalizing large coefficients, encouraging smaller weights overall, and [distributing weight values more evenly across all features](#).

Key Points

- **Overfitting Mitigation:** Ridge helps mitigate [overfitting](#), especially in high-dimensional datasets, and is effective in managing [Multicollinearity](#) among predictors.
- **Coefficient Shrinkage:** Unlike Lasso regularization (L1), which can eliminate some features entirely by driving their coefficients to zero, Ridge reduces the magnitudes of coefficients but retains all features.
- **Multicollinearity Handling:** Particularly useful when predictors are highly correlated, as it stabilizes estimates by shrinking the coefficients of correlated features.
- **Feature Retention:** Ridge retains all features in the model, unlike Lasso, which can perform [Feature Selection](#) by setting some coefficients to zero.

Understanding Ridge Regularization

1. Purpose of Ridge Regularization

- **Penalty Addition:** Adds a penalty term to the loss function, proportional to the square of the coefficients (weights), discouraging overly complex models by shrinking the coefficients.

2. Mathematical Formulation

- The loss function for Ridge regression can be expressed as: $\text{Loss} = \text{SSE} + \lambda \sum_{j=1}^p b_j^2$
 - Where:
 - SSE (Sum of Squared Errors) is the original loss function for [linear regression](#)
 - λ is the regularization parameter (penalty term) that controls the strength of the penalty.
 - b_j are the coefficients of the model.
 - p is the number of predictors.

3. Effect of the Regularization Parameter (λ)

- **Range:** λ can take values from 0 to infinity.
- **Impact:**
 - A small λ (close to 0) means the model behaves similarly to ordinary least squares (OLS) regression, with minimal regularization.
 - A large λ increases the penalty, leading to smaller coefficients and a simpler model.

4. Finding the Best λ

- Use techniques like cross-validation to determine the optimal value of λ . By testing various values and evaluating model performance, select the one that minimizes prediction error (or variance).

Example Code

```
from sklearn.linear_model import Ridge

# Initialize a Ridge model
model = Ridge(alpha=0.1) # alpha is the regularization strength
model.fit(X_train, y_train)
```

Resources

- [Understanding Ridge Regularization](#)

Understanding the Content

- **L2 Regularization (Ridge):** This technique is crucial for improving model generalization by penalizing large coefficients, which helps in reducing overfitting and handling multicollinearity. The regularization parameter λ controls the trade-off between fitting the training data well and keeping the model coefficients small.

[Ridge](#)

L2 Regularization (Ridge Regression): for [Neural network](#)

Adds a penalty term to the loss: ($L_{\text{regularized}} = L + \lambda \cdot \|W\|^2$). This discourages overly complex models by penalizing large weights.

Example:

```
from tensorflow.keras.regularizers import l2
Dense(25, activation="relu", kernel_regularizer=l2(0.01))
```

Row Based Storage

Data is stored in consecutive rows, allows [CRUD](#)

Row-based storage is well-suited for transactional systems ([OLTP](#))

Less efficient than [Columnar Storage](#) in largedatasets.

Row-based Storage Example (Transactional Workloads): **For the same table, if the goal is to efficiently handle transactions like inserting or updating an order**, row-based storage** organizes data row by row.

order_id	customer_id	order_date	order_amount
1	101	2024-10-01	\$100
2	102	2024-10-02	\$150
3	103	2024-10-03	\$200

In **row-based storage**, entire records (rows) are stored together. For example:

- Row 1: [1, 101, 2024-10-01, \$100]

When performing an **insert** or **update**, the entire row can be read and written back quickly, making this method ideal for transactional operations where complete records need to be processed together.

Use case **OLTP**. For instance, inserting a new order or updating an existing one (like modifying `order_amount` or `customer_id`) is efficient because all the data for a single record is stored together in a row.

Requirements.Txt

Reverse Etl

Reverse **ETL** is the flip side of the **ETL/ELT**. With Reverse ETL, the **data warehouse becomes the source rather than the destination**. Data is taken from the warehouse, transformed to match the destination's data formatting requirements, and loaded into an application – for example, a CRM like Salesforce – to enable action.

In a way, the Reverse ETL concept is not new to data engineers, who have been enabling data movement warehouses to business applications for a long time.

As [Maxime Beauchemin](#) mentions in [his article](#), Reverse ETL “appears to be a modern new means of addressing a subset of what was formerly known as [Master Data Management \(MDM\)](#).”

Read more about in [Reverse ETL Explained](#).

Rollup

Rollup refers to aggregating data to a higher level of [granularity](#), such as summarizing hourly data into daily totals.

[Database](#)

S

Table of Contents

- SHapley Additive exPlanations
 - SMOTE (Synthetic Minority Over-sampling Technique)
 - SMSS
 - SQL Groupby
 - SQL Injection
 - SQL Joins
 - SQL Window functions
 - SQL vs NoSQL
 - SQL
 - SQLAlchemy vs. sqlite3
 - SQLAlchemy
 - SQLite Studio
 - SQLite
 - SVM_Example.py
 - Sarsa
 - Scala
 - Scalability
 - Scaling Agentic Systems
 - Scaling Server
 - Scheduled Tasks
 - Schema Evolution
 - Scientific Method
 - Seaborn
 - Search
 - Security
 - Semantic Relationships
 - Sentence Similarity
 - Sharepoint
 - Silhouette Analysis
 - Similarity Search
 - Single Source of Truth
 - Sklearn Pipeline
 - Sklearn
 - Slowly Changing Dimension
 - Small Language Models
 - Smart Grids
 - Snowflake Schema
 - Snowflake
 - Soft Deletion
 - Software Design Patterns
 - Software Development Life Cycle
 - SparseCategoricalCrossentropy or CategoricalCrossEntropy
 - Specificity
 - Spreadsheets vs Databases
 - Stacking
-

- Standard deviation
- Standardisation
- Star Schema
- Statistical Assumptions
- Statistical Tests
- Statistics
- Stemming
- Stochastic Gradient Descent
- Stored Procedures
- Strongly vs Weakly typed language
- Structuring and organizing data
- Summarisation
- Supervised Learning
- Support Vector Classifier (SVC)
- Support Vector Machines
- Support Vector Regression
- Symbolic computation
- Sympy
- semantic layer
- semi-structured data
- shapefile
- sklearn datasets
- spaCy
- storage layer object store
- structured data
- syntactic relationships

Shapley Additive Explanations

SHAP provides a unified approach to measure [Feature Importance](#) by computing the contribution of each feature to each prediction, based on game theory.

Key Points

- **Purpose:** SHAP provides consistent and locally accurate explanations by assigning each feature [an importance value based](#) on Shapley values from cooperative game theory.
- **How it Works:**
 - It calculates how each feature contributes to the model's output by comparing predictions with and without the feature, across various feature value combinations.
- **Use Cases:** Suitable for complex models like neural networks, random forests, or gradient boosting machines, where internal logic is difficult to understand.
- **Advantage:**
 - Provides global explanations (model-wide feature importance) and local explanations (individual prediction reasons).
- **Scenario:**
 - A financial institution uses a black-box XGBoost model to predict whether a loan applicant should be approved. The model takes several factors into account, such as credit score, income, employment history,

and outstanding debts.

- **SHAP Explanation:** For a specific loan rejection case, SHAP values reveal that the applicant's high debt-to-income ratio and short employment history contributed the most to the rejection decision. These factors had the highest negative SHAP values for this prediction, providing actionable insights to both the applicant and the loan officers.

Example Code

To compute SHAP values, you can use the SHAP library to interpret feature importance for any machine learning model:

```
import shap

# Explain the model's predictions using SHAP
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

# Plot the summary plot of feature importance
shap.summary_plot(shap_values, X_test)
```

Smote (Synthetic Minority Over Sampling Technique)

SMOTE (Synthetic Minority Over-sampling Technique)

Generate synthetic samples for the minority class by interpolating between existing samples.

SMOTE: This technique generates synthetic samples for the minority class (female resumes) by creating new instances that are interpolations of existing ones.

Sms

microsoft sql server management.

Sql Groupby

The **SQL GROUP BY** clause is used to group rows that have the same values in specified columns into summary rows, like "total sales per region" or "average age per department."

It is often used in conjunction with aggregate functions such as `COUNT()`, `SUM()`, `AVG()`, `MAX()`, and `MIN()` to perform calculations on each group.

Basic Syntax

```
SELECT column1, aggregate_function(column2)
FROM table_name
WHERE condition
GROUP BY column1;
```

Example Usage

Let's say you have a table called `sales` with the following columns:

- `id` : Unique identifier for each sale
- `product` : The name of the product sold
- `amount` : The sale amount
- `region` : The region where the sale occurred

1. Count the Number of Sales per Product

To count how many sales were made for each product, you would use:

```
SELECT product, COUNT(*) AS total_sales
FROM sales
GROUP BY product;
```

2. Calculate Total Sales Amount per Region

To calculate the total sales amount for each region, you would use:

```
SELECT region, SUM(amount) AS total_sales_amount
FROM sales
GROUP BY region;
```

Using `HAVING` with `GROUP BY`

You can also filter the results of a `GROUP BY` [Querying|query](#) using the `HAVING` clause. This is useful when you want to filter groups based on aggregate values.

Example: Filter Groups

For example, to find products with total sales greater than \$1000:

```
SELECT product, SUM(amount) AS total_sales_amount
FROM sales
GROUP BY product
HAVING SUM(amount) > 1000;
```

Important Points

- **Columns in SELECT:** When using `GROUP BY`, all columns in the `SELECT` statement must either be included in the `GROUP BY` clause or be used in an aggregate function.
- **Order of Execution:** The `GROUP BY` clause is processed after the `WHERE` clause but before the `ORDER BY` clause in the SQL execution order.

[SQL Groupby Tags](#): #data_transformation #querying

Sql Injection

SQL injection is a code injection technique that targets applications using SQL databases. It occurs when a malicious user injects harmful SQL code into a query, potentially compromising the security of the database.

How SQL Injection Works

Consider a scenario where a website prompts users to log in with their username and password. The application might execute a query like this:

```
SELECT `id` FROM `users`
WHERE `user` = 'Carter' AND `password` = 'password';
```

If the user named Carter enters their credentials correctly, the query functions as intended. However, a malicious user could input a different string, such as:

```
'password' OR '1' = '1'
```

This alters the query to:

```
SELECT `id` FROM `users`
WHERE `user` = 'Carter' AND `password` = 'password' OR '1' = '1';
```

As a result, the attacker could gain unauthorized access to the database.

Example of Vulnerable Code

The following Python function demonstrates how SQL injection can occur due to unsafe query construction:

```
import sqlite3

def unsafe_query(user_input):
    query = "SELECT * FROM users WHERE name = " + user_input + ""
    conn = sqlite3.connect('example.db')
    conn.execute(query)
```

In this example, the `unsafe_query` function constructs SQL queries using string concatenation, making it vulnerable if user input is not properly sanitized.

Preventing SQL Injection

To mitigate the risk of SQL injection, it is essential to use prepared statements or parameterized queries. For example, consider an SQL injection attack that aims to display all user accounts from the `accounts` table:

```
SELECT * FROM `accounts`
WHERE `id` = 1 UNION SELECT * FROM `accounts`;
```

Using a prepared statement, we can safeguard against such attacks:

```
PREPARE `balance_check`  
FROM 'SELECT * FROM `accounts`  
WHERE `id` = ?';
```

In this statement, the question mark acts as a placeholder for user input, preventing the execution of unintended SQL code.

Executing the Prepared Statement

To execute the prepared statement and check a user's balance, we can set a variable for the user ID:

```
SET @id = 1;  
EXECUTE `balance_check` USING @id;
```

Here, the `SET` statement simulates obtaining the user's ID through the application, with the `@` symbol denoting a variable in MySQL.

Testing with Malicious Input

If we attempt to run the same statements with a malicious ID:

```
SET @id = '1 UNION SELECT * FROM `accounts`';  
EXECUTE `balance_check` USING @id;
```

The output will still reflect the balance of the user with ID 1, without exposing any additional data. This demonstrates that prepared statements effectively prevent SQL injection attacks.

Mitigation Strategies

- **Use Parameterized Queries:** Always use parameterized queries or prepared statements to prevent SQL injection.
- **Validate and Sanitize Inputs:** Ensure user inputs are validated and sanitized before being processed.

Sql Joins

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/Transformation/Joining.ipynb



Sql Window Functions

SQL Window Functions are a feature in SQL that allow you to perform calculations across a set of table rows that are somehow related to the current row.

Unlike regular aggregate functions, which return a single value for a group of rows, window functions return a value for each row in the result set while still allowing access to the individual row data.

Key Characteristics of Window Functions

1. Non-Aggregating: Window functions do not collapse rows into a single output row. Instead, they perform calculations across a defined "window" of rows related to the current row.
2. OVER Clause: Window functions are defined using the `OVER` clause, which specifies the window of rows to be considered for the function.
3. Partitioning: You can partition the result set into groups using the `PARTITION BY` clause within the `OVER` clause. Each partition is processed independently.
4. Ordering: You can specify the order of rows within each partition using the `ORDER BY` clause within the `OVER` clause.

Example Use Case

Suppose you have a table `sales` with columns `salesperson`, `region`, and `amount`. You can use window functions to calculate the total sales for each salesperson while still displaying individual sales records:

Initial Table: `employees`

id	name	department	salary
1	Alice	Sales	50000
2	Bob	Sales	60000
3	Charlie	HR	55000
4	David	HR	70000
5	Eve	IT	80000
6	Frank	IT	75000

Example **Querying|Queries** Using SQL Window Functions

1. Using `ROW_NUMBER()`

The `ROW_NUMBER()` function assigns a unique rank to each employee within their department based on their salary.

```
SELECT
    id,
    name,
    department,
    salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;
```

Resulting Table:

id	name	department	salary	rank
2	Bob	Sales	60000	1
1	Alice	Sales	50000	2
4	David	HR	70000	1
3	Charlie	HR	55000	2
5	Eve	IT	80000	1
6	Frank	IT	75000	2

2. Using `SUM()` similar AVG

The `SUM()` function calculates the total salary for each department, showing the same total for each employee in that department.

```
SELECT
    id,
    name,
    department,
    salary,
    SUM(salary) OVER (PARTITION BY department) AS total_department_salary
FROM employees;
```

Resulting Table:

id	name	department	salary	total_department_salary
1	Alice	Sales	50000	110000
2	Bob	Sales	60000	110000
3	Charlie	HR	55000	125000
4	David	HR	70000	125000
5	Eve	IT	80000	155000
6	Frank	IT	75000	155000

[SQL Window functions](#) Tags: #data_analysis, #querying

Sql Vs Nosql

[NoSQL](#)

Sql

Structured Query Language (SQL) is the standard language for interacting with relational databases, enabling efficient data [Querying](#) and manipulation. It serves as a common interface for [Database](#)s and data lakes.

Features:

- Declarative language for storing and querying structured data.
- Transactional properties enhance speed and efficiency.

Good Practices

Capitalization:

- Use uppercase for SQL keywords for better readability.
- Use lowercase for table and column names.

Quotes:

- Use double quotes for SQL identifiers (table and column names).
- Use single quotes for string values.

Related terms

[Database Techniques](#)

SQLAlchemy vs. sqlite3: Which One Should You Use?

The choice between [SQLAlchemy](#) and [SQLite](#) depends on your specific needs. Here's a comparison based on key factors:

1. Abstraction and Ease of Use

Feature	SQLAlchemy	sqlite3
Abstraction	High-level ORM (Object Relational Mapping)	Low-level, direct SQL execution
Ease of Use	Pythonic API for working with databases	Requires writing raw SQL queries
Best for	Large projects, scalable applications	Simple scripts, small projects

Use SQLAlchemy if you want to work with database tables as Python objects (ORM).

Use sqlite3 if you are comfortable writing SQL queries directly.

2. Supported Databases

Feature	SQLAlchemy	sqlite3
Database Support	Works with MySQL, PostgreSQL, SQLite, MSSQL, etc.	Only works with SQLite
Portability	Can switch databases easily	Tied to SQLite only

Introduction

- Use SQLAlchemy if you need flexibility to work with different databases.

- Use sqlite3 if you are only working with SQLite.

3. Performance and Scalability

Feature	SQLAlchemy	sqlite3
Performance	Slightly slower due to ORM overhead	Faster for simple queries
Scalability	Supports connection pooling, transactions, and large-scale applications	Best for local, single-user applications

- Use SQLAlchemy for large applications with complex relationships.

- Use sqlite3 if you just need a simple, fast database for local use.

4. Querying and Data Manipulation

Feature	SQLAlchemy	sqlite3
Querying	Can use both ORM and raw SQL queries	Only supports raw SQL queries
Ease of Data Manipulation	Object-oriented approach (e.g., <code>session.add(obj)</code>)	SQL execution via <code>cursor.execute(query)</code>

- Use SQLAlchemy if you prefer writing queries in a Pythonic way (ORM).

- Use sqlite3 if you are fine with executing raw SQL statements.

5. Transaction Handling

Feature	SQLAlchemy	sqlite3
Transaction Control	Automatic transaction management	Manual transaction handling (<code>conn.commit()</code>)
Rollback Support	Easier and more reliable	Must be explicitly handled

- Use SQLAlchemy for better transaction control in complex applications.

- Use sqlite3 if you want manual control over transactions.

6. Learning Curve

Feature	SQLAlchemy	sqlite3
Difficulty Level	Higher due to ORM concepts	Easier to get started

Use sqlite3 if you want a simple database solution with SQL queries.

Use SQLAlchemy if you are comfortable with an ORM and want a scalable approach.

When to Use SQLAlchemy?

- You are building a large, scalable application.
- You need database flexibility (MySQL, PostgreSQL, etc.).
- You prefer Pythonic ORM instead of writing raw SQL.
- You want better transaction handling and connection management.

When to Use sqlite3?

- You need a lightweight, single-file database.
- You are working on a small project or script.
- You are comfortable writing raw SQL queries.
- You do not need an ORM or multiple database support.

Final Recommendation

- For simple SQLite-based projects: Use `sqlite3` (faster, simpler).
- For larger applications needing scalability and maintainability: Use `SQLAlchemy`.

Sqlalchemy

SQLAlchemy is a Python SQL toolkit and **Object Relational Mapper** (ORM) that provides tools to interact with databases in a more Pythonic way. It allows you to work with relational databases such as MySQL, PostgreSQL, SQLite, and others without writing raw `SQL` queries manually.

Related:

- [SQLAlchemy vs. sqlite3](#)

In `DE_Tools` see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLAlchemy/sql_alchemy.ipynb

Why Use SQLAlchemy?

- Reduces SQL complexity: Write Python code instead of SQL queries.
- Prevents `SQL Injection`: ORM prevents unsafe queries.
- Improves `maintainability`: Easier to refactor code.
- Handles connection pooling: Manages database connections efficiently.

- Works with Pandas: Can load and save data directly to databases.

Key Features of SQLAlchemy

- Database Connectivity
 - Provides a unified interface to connect to different databases.
- SQL Query Execution
 - Allows execution of raw SQL queries using [Pandas](#)
- ORM (Object Relational Mapping)
 - Converts database tables into Python objects (classes).
 - Eliminates the need to write SQL [Querying|Queries](#) manually.
 - Example:

```
from sqlalchemy.orm import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class Customer(Base):
    __tablename__ = 'customers'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    phone_number = Column(String)
```

- Transaction Management
 - Provides robust control over commit and rollback operations.
 - Ensures data integrity by handling failures safely.
- Efficient Query Building
 - Allows writing Pythonic queries instead of raw SQL.
 - Example:

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
session = Session()

customers = session.query(Customer).filter_by(name="John Doe").all()
```

- Supports Multiple Databases
 - Works with [[MySql],[PostgreSQL](#),[SQLite](#), etc.
 - Easily switch databases without changing the core logic.

Sqlite Studio

Sqlite

Lightweight [Database Management System \(DBMS\)](#) used in various applications (phone apps, desktop apps, websites).

Note [SQLite Studio](#) exists

To get in terminal enter:

```
sqlite3 database.db
```

Related notes:

- [Querying](#)
- [Concurrency](#)
- [SQL](#)

Svm_Example.Py

https://github.com/rhyslwells/ML_Tools/blob/main\Explorations/Build/Classifiers/SVM/SVM_Example.py

Overview

- **Objective:** To classify Iris flowers using SVM and explore various hyperparameters like kernel type, regularization (C), and gamma.
- **Dataset:** The Iris dataset contains information about sepal and petal dimensions for three flower species.
- To explore the effect of **soft boundaries** in SVMs, you can adjust the regularization parameter CCC. A smaller CCC allows a **softer boundary** (more margin violations), prioritizing generalization. A larger CCC enforces a **harder boundary** with fewer margin violations, but may lead to overfitting. Here's an extended version of the script to include this exploration:

Steps in the Script

1. Data Loading and Preparation

- The Iris dataset is loaded using `sklearn.datasets.load_iris`.
- A DataFrame is created with:
 - Features: Sepal and petal dimensions.
 - Target: Numerical representation of flower species.
 - Flower name: Categorical species name derived from the target.

2. Data Visualization

- The data is visualized to explore relationships between features:
 - **Sepal Length vs. Sepal Width** for two species (Setosa vs. Versicolor).
 - **Petal Length vs. Petal Width** for the same species.
- Scatter plots are used to identify separable patterns.

3. Model Training

- The data is split into training and testing sets (80%-20%).
- An **SVM classifier** (`sklearn.svm.SVC`) is trained on the training set.
- The model's performance is evaluated using the `.score()` method.

4. Hyperparameter Tuning

- **Regularization (C):**
 - Adjusting `C` controls the trade-off between achieving a large margin and minimizing classification errors.
 - Lower values of `C` allow a larger margin but can tolerate misclassified points.
 - Higher values of `C` prioritize correct classification over a larger margin.
- **Gamma:**
 - Controls the influence of individual data points. A high value means data points closer to the hyperplane have more influence.
- **Kernel:**
 - Different kernels (e.g., `linear`, `rbf`) are tested to find the best mapping of data into higher dimensions for better separation.

5. Prediction and Accuracy

- The model is used to predict flower species for new samples.
- The accuracy of the model is reported for each combination of hyperparameters.

Sarsa

SARSA stands for State-Action-Reward-State-Action

SARSA is another value-based [Reinforcement learning](#) algorithm, differing from Q-learning in that it updates the Q-values based on the action actually taken by the policy.

SARSA update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Explanation:

- $Q(s_t, a_t)$: The Q-value of the current state s_t and action a_t .
- α : The learning rate, determining how much new information overrides old information.
- r_{t+1} : The reward received after taking action a_t from state s_t .
- γ : The discount factor, balancing immediate and future rewards.
- $Q(s_{t+1}, a_{t+1})$: The Q-value for the next state s_{t+1} and the action a_{t+1} actually taken according to the policy.

Notes:

- SARSA's on-policy nature ensures that it learns a policy that aligns with its exploration strategy, leading to more stable behavior in environments with randomness or noise.
- The learning process may be slower compared to Q-learning, but it can be more robust in environments where the agent's behavior is expected to align closely with the policy it follows.

Scala

[!Summary] Scala is a functional programming language primarily used for **big data processing**, particularly with frameworks like **Apache Spark**. It is known for its **concise syntax** and its ability to integrate seamlessly with the **Java ecosystem**, running on the **JVM** (Java Virtual Machine).

While Scala has a **smaller user base** and is considered **hard to learn**, it is highly **expressive** and offers strong support for managing **distributed systems** and building large-scale data pipelines. Its robust features make it a top choice for **big data engineers**.

Key Features of Scala

1. Functional Programming:

- Scala is built around functional programming principles, offering key features such as:
 - **Lambdas** (anonymous functions)
 - **Pattern matching**
 - **Functions as first-class citizens**
 - **Data classes** for concise data modeling.

2. Immutability:

- **Immutability** is a core principle in Scala. By default, data structures are **immutable**, which promotes **thread safety** and makes code easier to reason about. This feature aligns well with building reliable and scalable distributed systems.

3. Advanced Language Features:

- **Type inference** allows the compiler to deduce types, leading to shorter and cleaner code.
- **Higher-order types** and **meta-programming** support advanced abstractions and code expressiveness.
- **Meta programming** allows compile-time code generation, improving type correctness and reducing runtime errors.

4. Expressive Data Manipulation:

- Scala is renowned for its **concise and readable code** when it comes to data manipulation. Its type-safe methods provide **powerful tools** for working with data models efficiently and expressively.

5. Type System:

- Scala has an **advanced type system** that enforces **strong typing** at compile time. This system helps prevent illegal states, reducing the need for runtime tests and making the code more robust and reliable.

6. Library Over Framework:

- Scala promotes the use of **libraries over frameworks**, which provides developers with more flexibility in how they design and structure their applications.

Additional Notes

- **Integration with Java:** Scala can be seamlessly mixed with **Java**, allowing developers to use existing Java libraries and tools.
- **Used with Apache Spark:** Scala is the most common language used for **Apache Spark**, a leading big data processing framework.
- **Niche and Learning Curve:** While Scala's adoption is smaller compared to languages like Java or Python, its **expressiveness** and **power** make it a popular choice for niche applications, especially in big data environments.

Scalability

Scalability refers to the capability of a system, network, or process to handle a growing amount of work or its potential to accommodate growth.

Key Benefits of Scalability:

- Performance Improvement: As demand increases, scalable systems can maintain or improve performance levels.
- Cost Efficiency: Organizations can manage costs by scaling resources according to demand, avoiding over-provisioning.
- Flexibility: Scalable systems can adapt to changing workloads and business needs, making it easier to accommodate growth.
- Reliability: Distributing workloads across multiple nodes can enhance system reliability and reduce the risk of a single point of failure.

Vertical Scalability (Scaling Up):

This involves adding more resources to a single node or server to increase its capacity. For example, upgrading a server's CPU, adding more RAM, or increasing storage space.

Vertical scaling can improve performance but has limitations, as there is a maximum capacity that a single machine can reach.

Horizontal Scalability (Scaling Out):

This involves adding more nodes or servers to a system to distribute the load. For example, adding more servers to a web application to handle increased traffic.

Horizontal scaling allows for greater flexibility and can often be more cost-effective, as it enables the use of multiple lower-cost machines rather than relying on a single powerful machine.

Tags

- Tags: #data_management

Scaling Agentic Systems

Agentic solutions propose an improvement over traditional Large Language Model (LLM) usage by employing networks of Small Language Models (SLMs). These systems aim to strike a balance between scalability, control, and performance, addressing specific tasks with precision while maintaining overall system adaptability.

Ideas from MLOPs talk by MaltedAI.

Agentic solutions represent a pragmatic approach to AI systems by focusing on modularity, task-specific efficiency, and the thoughtful integration of human expertise. These architectures show promise for enhancing scalability, control, and adaptability in real-world applications.

Contrasting SLMs and LLMs

Small Language Models|SLM (Small Language Models):

Introduction

- Intent-based conversations and decision trees.
- Controlled systems, harder to build features but easier to execute.
- Task-specific and efficient in offline environments.

LLMs (Large Language Models):

- Flexible and natural in handling diverse queries.
- Suitable for general-purpose scenarios and exploratory tasks.
- High computational and scaling costs.

Combined Approach:

- Use [Small Language Models|SLM](#) for inference and LLMs for training.
- Shift the focus from making models larger to solving real-world problems effectively.

Key Concepts in Agentic Solutions

- Neural Decision Parser:
 - Acts as the "brain" of the system, determining the appropriate action given user input.
 - SLMs interpret user utterances to express code aligned with system intent.
- Phased Policy:
 - Distinguishes between contextual and general-purpose questions.
 - Ensures deliberate task execution in stages for clarity and efficiency.
- Knowledge Graphs and Interaction Models:
 - Complex graph structures enable intelligent conversations between models.
 - RAG setups leverage teacher-student frameworks for effective task distribution.
- Distillation Networks of SLMs:
 - SMEs (Subject Matter Experts) guide teacher models that distill their expertise into student SLMs.
 - Enhances task scalability while controlling costs.
- Scaling with Distillation:
 - Leverage teacher-student frameworks for high-quality, scalable data.
 - Allow teacher models to handle hard-to-scale aspects.
- Knowledge Discovery:
 - Extract SME knowledge effectively while filtering irrelevant data.
 - Build high-quality datasets for task-specific applications.

Applications of SLM Networks

1. Task-Specific Systems:
 - Offline processing, task search, and targeted QA.
 - Optimized embedding spaces for domain-specific applications.
2. Swarm Intelligence:
 - Decision-making through deliberation among SLMs.
 - Aggregates diverse inputs (HR, tech, CEO) for robust conclusions.

3. Business Process Models:

-
- Search and page ranking systems.
 - Smaller, focused systems tailored to specific business needs.

Designing Agentic Solutions

1. Role of SMEs:

- Define tasks and input structures.
- Guide model development with domain knowledge.

2. Data Preparation:

- Comprehensive sampling of the problem space ensures generalization.
- Data variability is critical for robust models.

3. Evaluation and Responsiveness:

- Measure system performance to enable continuous improvement.
- Focus on responsive, real-time processing.

4. Tool Integration:

- Use LLMs with Python engines or computational tools like Wolfram for data analysis and complex interactions.

Advantages of SLM Networks

- Precision: Models perform only what they are designed for.
- Efficiency: Smaller models are scalable and cost-effective.
- Focused Applications: Avoids the complexity of embedding spaces for entire businesses.

Future Directions

Scaling Server

Scaling Server

- Horizontal Scaling: Adding more servers, preferred for scalability.
- Vertical Scaling: Adding more resources (memory, CPU) to existing servers.

Scheduled Tasks

Similar to [Cron jobs](#) in Unix

Using `schtasks` (Command-Line)

Windows provides `schtasks`, a command-line tool to schedule tasks.

Example Commands

- Run a Python script every 5 minutes:

cmd

CopyEdit

```
schtasks /create /tn "MyPythonScript" /tr "C:\Python\python.exe C:\path\to\script.py" /sc minute /mo 5 /f
```

- Run a batch file daily at 3 AM:

cmd

CopyEdit

```
schtasks /create /tn "DailyBackup" /tr "C:\path\to\backup.bat" /sc daily /st 03:00 /f
```

- Delete a scheduled task:

cmd

CopyEdit

```
schtasks /delete /tn "DailyBackup" /f
```

Schema Evolution

[Database Schema](#)|[Schema](#) Evolution means adding new columns without breaking anything or even enlarging some types.

You can even rename or reorder columns, although that might break backward compatibilities. Still, we can change one table, and the table format takes care of switching it on all distributed files. Best of all does not require rewrite of your table and underlying files.

How is schema evolution done in practice

In [DE_Tools](#) see:

- https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLite/Writing/Schema_Evolution.ipynb

See also:

- [ACID Transaction](#)

Scientific Method

Step 1: Start with Data

- **Collect Data:** Gather all relevant data sources that might be useful for your analysis.
- **Understand Data:** Familiarize yourself with the data types, structures, and any existing metadata.
- **Clean Data:** Perform data cleaning to handle missing values, outliers, and inconsistencies.

Step 2: Develop Intuitions

- **Explore Data:** Use exploratory data analysis ([EDA](#)) techniques to visualize and summarize the data.

- **Identify Patterns:** Look for trends, correlations, and anomalies that might inform your understanding.
 - **Ask Preliminary Questions:** Consider what initial questions the data might help answer.
-

Step 3: Formulate Your Question

- **Problem Definition:** Clearly articulate the problem you are trying to solve.
- **Set Objectives:** Determine what you aim to achieve with your analysis.
- **Consider Stakeholders:** Ensure the question aligns with business goals and stakeholder interests.

Step 4: Validate the Question

- **Test Feasibility:** Use the current data to assess whether the question is answerable.
- **Iterate:** Refine the question based on initial findings and feedback.
- **Formulate Hypothesis:** Develop a testable hypothesis that can guide your analysis.

Step 5: Create a Testing Framework

- **Design Experiments:** Plan how you will test your hypothesis, including control and experimental groups if applicable.
- **Select Methods:** Choose appropriate statistical or machine learning methods for analysis.
- **Prepare Tools:** Set up the necessary tools and environments for running experiments.

Step 6: Analyze Results

- **Run Experiments:** Execute your tests and collect results.
- **Interpret Data:** Use quantitative metrics to analyze the outcomes.
- **Draw Insights:** Identify key insights and patterns that answer your question.

Step 7: Assess Impact

- **Define Success Metrics:** Determine how you will measure success (e.g., accuracy, ROI, user engagement).
- **Evaluate Impact:** Assess the potential impact of your solution on the business.
- **Communicate Findings:** Present your results and recommendations to stakeholders.

Additional Considerations

- **Iterative Process:** Be prepared to revisit and refine each step as new insights emerge.
- **Documentation:** Keep thorough documentation of your process, findings, and decisions.

Seaborn

<https://seaborn.pydata.org/tutorial.html>

Related:

- [Data Visualisation](#)

Search

Security

[Common Security Vulnerabilities in Software Development](#)

Semantic Relationships

Semantic relationships refer to the connections and associations between words and concepts based on their meanings.

Understanding these relationships can enhance various natural language processing tasks, such as information retrieval, text analysis, and sentiment analysis.

Leveraging Lexical Resources like [WordNet](#)

One of the key resources for exploring semantic relationships is **WordNet**, a lexical database that groups words into sets of cognitive synonyms called **synsets**. These synsets are linked together in a hierarchy based on semantic relations, including:

- **Hypernymy:** Represents an "is-a" relationship (e.g., "dog" is a hypernym of "beagle").
- **Hyponymy:** Represents a more specific type (e.g., "beagle" is a hyponym of "dog").

You can use WordNet to find synonyms or related concepts for important words (those with high [TF-IDF](#) scores) in your documents. If different documents contain synonyms or words related in the WordNet hierarchy, this may indicate a semantic relationship between them, even if the exact words differ.

WordNet also provides measures of semantic similarity between synsets based on their paths in the hypernym hierarchy. These measures can be explored to quantify the semantic relatedness of key terms in your documents. The Natural Language Toolkit ([NLTK](#)) offers an interface to access WordNet.

Sentiment Analysis with [SentiWordNet](#)

Another valuable resource is **SentiWordNet**, which extends WordNet by assigning sentiment scores (positive, negative, objective) to different senses of words. While your primary goal may be to explore semantic relationships, analyzing the sentiment expressed in your documents based on important words can provide an additional layer of understanding.

Documents discussing similar topics might also share similar sentiments, strengthening the case for a semantic link. NLTK provides access to SentiWordNet, allowing you to incorporate sentiment analysis into your exploration of semantic relationships.

Sentence Similarity

Sentence similarity refers to the degree to which two sentences are alike in meaning. It is a crucial concept in natural language processing ([NLP](#)) tasks such as information retrieval, text summarization, and paraphrase detection. Measuring sentence similarity involves comparing the semantic content of sentences to determine how closely they relate to each other.

There are several methods to measure sentence similarity:

1. **Lexical Similarity:** This involves comparing the words in the sentences directly. Common techniques include:
 - **Jaccard Similarity:** Measures the overlap of words between two sentences.
 - **Cosine Similarity:** Represents sentences as vectors (e.g., using TF-IDF) and measures the cosine of the angle between them.
2. **Syntactic Similarity:** This considers the structure of the sentences, using techniques like:
 - **Parse Trees:** Comparing the syntactic trees of sentences to see how similar their structures are.
3. **Semantic Similarity:** This goes beyond surface-level word matching to understand the meaning of sentences:
 - **Word Embeddings:** Using models like Word2Vec, GloVe, or FastText to represent words in a continuous vector space, then averaging these vectors to represent sentences.
 - **Sentence Embeddings:** Using models like Universal Sentence Encoder, BERT, or Sentence-BERT to directly obtain embeddings for entire sentences, which can then be compared using cosine similarity or other distance metrics.
4. **Neural Network Models:** Advanced models like BERT, RoBERTa, or GPT can be fine-tuned on specific tasks to directly predict similarity scores between sentence pairs.
5. **Hybrid Approaches:** Combining multiple methods to leverage both lexical and semantic information for a more robust similarity measure.

Each method has its strengths and weaknesses, and the choice of method often depends on the specific requirements of the task and the available computational resources.

Sharepoint

SharePoint is a web-based collaboration platform developed by [Microsoft](#). It is primarily used for creating intranet sites, document management, and team collaboration, providing a centralized platform for managing content and communication.

SharePoint integrates with Microsoft Office and is highly customizable, making it a versatile tool for organizations of all sizes.

Key Features

1. **Intranet Sites:**
 - Create internal websites for team collaboration and communication.
 - Share news, updates, and resources within an organization.
2. **Document Repository:**
 - Store, organize, and manage documents in a centralized location.
 - Version control and access permissions ensure document integrity and security.
3. **Lists and Libraries:**
 - Create lists to manage data and tasks.
 - Libraries for storing and organizing documents and other files.
4. **Team and Communication Sites:**
 - Team Sites: Facilitate collaboration within specific teams or projects.
 - Communication Sites: Share information broadly across an organization.
5. **Integration with Microsoft Teams:**
 - SharePoint sites can be integrated with Microsoft Teams channels, providing a seamless collaboration experience.

Use Cases

- **Document Management:** Centralize document storage and enable easy sharing and collaboration.
- **Project Management:** Use lists and libraries to track project tasks and resources.
- **Internal Communication:** Share company news, updates, and announcements through intranet sites.

Silhouette Analysis

[Sklearn link](#)

Silhouette analysis is a technique used to evaluate the quality of clustering results. It provides a measure of how similar an object is to its own cluster compared to other clusters. This analysis helps in determining the appropriateness of the number of clusters and the consistency within clusters.

Overall, silhouette analysis is a valuable tool for assessing the quality of [clustering](#) results and making informed decisions about the number of clusters and the clustering algorithm's effectiveness.

Key Concepts:

Silhouette Score: For each data point, the silhouette score is calculated using the following formula:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where: $a(i)$ is the average distance between the data point i and all other points in the same cluster. $b(i)$ is the average distance between the data point i and all points in the nearest cluster (the cluster with the smallest average distance to i).

Interpretation of Silhouette Scores: A silhouette score ranges from -1 to 1. A score close to 1 indicates that the data point is [wellclustered](#) and far from neighboring clusters. A score close to 0 indicates that the data point is on or very close to the decision boundary between two neighboring clusters. A negative score indicates that the data point might have been assigned to the wrong cluster.

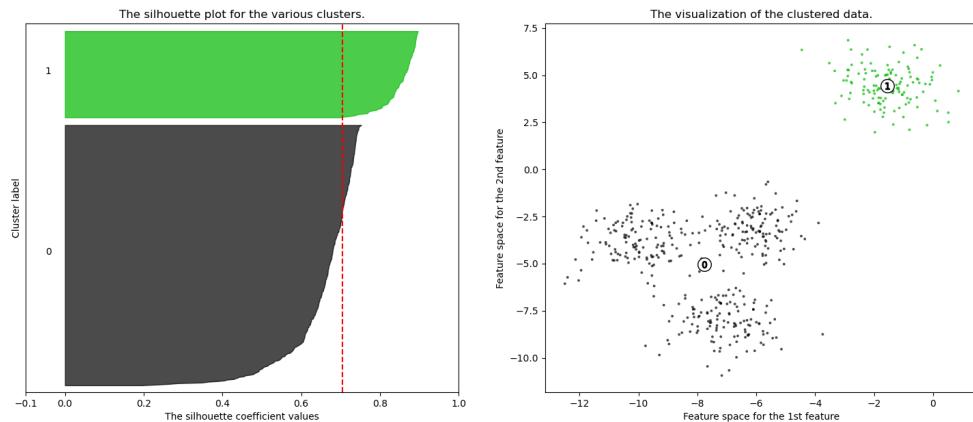
Silhouette Plot:

A silhouette plot displays the silhouette scores of all data points in a dataset. It provides a visual representation of how well each data point lies within its cluster.

The plot is divided into regions, each corresponding to a cluster, and the width of each region represents the average silhouette score of the points in that cluster.

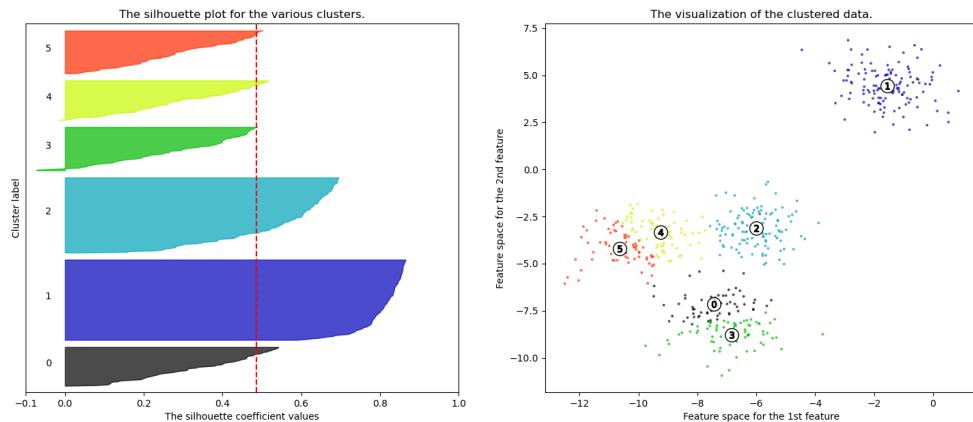
Good

Silhouette analysis for KMeans clustering on sample data with n_clusters = 2



Bad

Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



Applications:

Determining the Optimal Number of Clusters: By calculating the average silhouette score for different numbers of clusters, one can identify the number of clusters that results in the highest average silhouette score, indicating the best clustering structure.

Cluster Validation: Silhouette analysis helps in validating the consistency within clusters and identifying potential misclassifications.

Similarity Search

Single Source Of Truth

Sending data from across an enterprise into a centralized system such as a:

- Database
- Data Warehouse
- Data Lakehouse
- Data Lakehouse
- master data management

results in a single unified location for accessing and analyzing all the information that is flowing through an organization.

[Single Source of Truth](#) **Tags:** #data_management, #data_storage

Sklearn Pipeline

```
# Naivebayesfor email prediction
from sklearn.pipeline import Pipeline
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])
clf.fit(X_train, y_train)
clf.score(X_test,y_test)
clf.predict(user_input)
```

Sklearn

Terms of interest:

Also called Scikit-learn.

train_test_split

X and y are separate things (y is the target variable/column) and X is multiple is columns used to get y.

Given any pandas df use .to_numpy to convert first.

classifier score?

-0.018 bad 0.72 good

import #data_cleaning (puts all values between -1 and 1)

skileanr.pipeline allows you to combine steps.

to save model use pickle with

```
with open(out_file,"wb") as out:
    pickle.dump(pipe,out)
```

[p-values in linear regression in sklearn](#)

[Sklearn](#)

Slowly Changing Dimension

A Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a Data Warehouse.

It is considered and implemented as one of the most critical ETL tasks in tracking the history of dimension records.

How do you track slowly changing dimensions in a database

Take a customer dimension in a retail database.

Consider a retail company that tracks customer information, including attributes such as name, address, and membership status. Over time, customers may change their addresses or upgrade their membership levels.

Implementation of SCD

- The original record for John Doe is retained with an end date to indicate that it is no longer current.
- A new record is created for the updated information, allowing the company to maintain a history of changes over time.
- This approach allows analysts to query the data and understand customer behavior and trends over time, which is essential for reporting and decision-making.
- Current Data:** The current state of the customer dimension might look like this:

Customer ID	Name	Address	Membership Status
1	John Doe	123 Elm St, City A	Gold
2	Jane Smith	456 Oak St, City B	Silver

1. **Change Occurs:** If John Doe moves to a new address and upgrades his membership to Platinum, the company needs to track this change.

2. **Historical Data:** Using the SCD approach, the dimension table might be updated as follows:

Customer ID	Name	Address	Membership Status	Effective Date	End Date
1	John Doe	123 Elm St, City A	Gold	2022-01-01	2023-10-01
1	John Doe	789 Pine St, City A	Platinum	2023-10-01	NULL
2	Jane Smith	456 Oak St, City B	Silver	2022-01-01	NULL

Small Language Models

[LLM|LLMs](#) dominate many general-purpose NLP tasks, small [Language Models](#) have their own place in specialized tasks, where they excel due to computational efficiency, [interpretability](#), and task-specific fine-tuning.

SLMs remain highly relevant for [Edge Machine Learning Models](#) and edge computing, [domain-specific tasks](#), and applications requiring [interpretability](#), making them a crucial tool in the NLP landscape.

Use Cases for Small Language Models (SLMs)

- [Contrastive Decoding](#): Improve the quality of generated content by filtering out low-quality outputs, by having a SLM guide and critique a LLM or other way ([inference](#))
 - Mitigate hallucinations
 - Augmented Reasoning
- [Distillation](#): Transfer the knowledge from a larger model to a smaller one, retaining performance but reducing computational requirements (see [BERT Teacher model](#)).
- [Data Synthesis](#): Generate or augment datasets in scenarios with limited data.
- [Model Cascading](#): Use a combination of smaller models and larger models in a cascading architecture, where simpler tasks are handled by SLMs and more complex ones by LLMs. Model cascading and routing allow SMs to handle simpler tasks, reducing computational overhead. Or the other way first do a general search with a LLM then refine to domain specific small model which is more [interpretability|interpretable](#) and specific.
- Domain specific & Limited Data Availability: SMs, however, can be effectively fine-tuned on smaller, [domain-specific datasets](#) and outperform general LLMs in tasks with limited data availability.
- [RAG](#) (Retrieval Augmented Generation): Lightweight [retrievers](#) (SMs) can support LLMs in finding relevant external information.

Advantages of SLMs

- Require less computational power and are faster in [inference](#) compared to LLMs.
- [Interpretability](#)
- Accessible for those without resources in power and data

Smart Grids

Smart Grids

Want adaptive grid that can handle the volatility of energy coming on or off. This occurs more often due to the variety of sources i.e wind.

Help with carbon commitment

- [Overview](#): Smart grids utilize advanced technology and data analytics to improve the efficiency and reliability of electricity distribution. [RL](#) can optimize the operation and management of these grids.
- [Applications](#):
 - [Demand Forecasting](#): RL algorithms predict electricity demand based on historical data and real-time inputs. They adjust energy production and distribution to match forecasted demand.
 - [Load Balancing](#): RL can manage the distribution of electricity by dynamically balancing load across different sources, minimizing energy loss and enhancing stability.
 - [Renewable Energy Integration](#): RL helps in integrating renewable energy sources (e.g., solar, wind) into the grid by optimizing the usage of these variable resources and managing their unpredictability.

Snowflake Schema

Snowflake Schema

- Description: A more [Normalised Schema](#) normalized form of a star schema where dimension tables are further broken down into additional tables.
- Advantages: Reduces data redundancy and can save storage space, but may be more complex to query.
- A variation of the [Star Schema](#), the snowflake schema normalizes dimension tables into multiple related tables. This can reduce data redundancy and improve data integrity but may complicate queries due to the additional joins required.

Snowflake

1. Architecture:

- **Cloud-Native:** Snowflake is a fully managed, cloud-native data warehousing service. It operates entirely on cloud platforms like AWS, Azure, and Google [Cloud](#).
- **Separation of Storage and Compute:** Snowflake separates storage from compute, allowing for independent scaling of each. This means you can scale up compute resources without affecting storage capacity and vice versa.
- **Multi-Cluster Shared Data Architecture:** Snowflake uses a multi-cluster architecture to handle concurrent workloads, ensuring high performance and minimal contention.

2. Data Storage:

- **Structured Data:** Primarily designed for structured data and optimized for SQL queries and analytics.
- **Semi-Structured Data:** Also supports semi-structured data like JSON, Avro, and Parquet through its native capabilities.

3. Management:

- **Fully Managed Service:** Snowflake handles infrastructure management, optimization, and maintenance tasks automatically, requiring minimal administrative effort from users.

4. Performance:

- **High Performance:** Optimized for fast query performance, particularly for complex analytical queries. It uses advanced optimizations like automatic clustering and caching.

5. Use Cases:

- **Data Warehouse:** Ideal for enterprise data warehousing, business intelligence, and analytics.
- **Data Lake:** Can function as a data lake with support for semi-structured data.

Soft Deletion

Soft deletion is a technique used in databases to **mark records as deleted without physically removing them from the database**.

This approach is particularly useful in scenarios where [data integrity](#) and synchronization are important, such as during [Incremental Synchronization](#).

When using [incremental synchronization](#) modes, fully deleted records from a source system are not replicated. To handle this, a field can be added to each record to indicate whether it should be treated as deleted. This allows the system to maintain a complete history of records while still functioning as if certain records are removed.

Implementation

A common way to implement soft deletion is by adding a boolean flag, such as `is_deleted`, to the record [Database Schema](#). Here's how it works:

1. Flagging Records:

- When a record is "deleted," the `is_deleted` flag is set to `true`.

2. Querying Data:

- All [Querying|queries](#) must be designed to exclude records where `is_deleted` is `true`. For example:

```
SELECT * FROM table_name WHERE is_deleted = false;
```

3. Background Jobs:

- Periodically, background jobs can be executed to permanently remove records marked as deleted, if necessary, or to archive them for future reference.

Benefits

- Data Integrity:** Maintains a complete history of records, which can be useful for auditing and recovery.
- Ease of Recovery:** Records can be easily restored by simply resetting the `is_deleted` flag.
- Synchronization:** Facilitates incremental synchronization by ensuring that deleted records are still present in the database.

Considerations

- Query Complexity:** Requires careful query design to ensure that deleted records are consistently excluded.
- Storage:** Over time, soft-deleted records can accumulate, potentially leading to increased storage requirements.

Example of Soft Deletion

Let's say we have a table named `users` that stores user information. We will add a boolean column called `is_deleted` to indicate whether a user is "deleted."

In this example, we demonstrated how to implement soft deletion using a boolean flag in the `users` table. This approach allows for easy recovery of deleted records and maintains [data integrity](#) while facilitating incremental synchronization.

Step 1: Modify the Table Structure

First, we need to alter the `users` table to add the `is_deleted` column:

```
ALTER TABLE users ADD COLUMN is_deleted BOOLEAN DEFAULT false;
```

Step 2: Soft Delete a User

When a user wants to delete their account, instead of removing the record from the database, we update the `is_deleted` flag:

```
UPDATE users SET is_deleted = true WHERE user_id = 123;
```

Step 3: Querying Active Users

To retrieve a list of active users (those who are not deleted), we write our queries to exclude soft-deleted records:

```
SELECT * FROM users WHERE is_deleted = false;
```

Step 4: Restoring a Soft Deleted User

If a user decides to restore their account, we can simply set the `is_deleted` flag back to `false`:

```
UPDATE users SET is_deleted = false WHERE user_id = 123;
```

Step 5: Permanently Deleting Soft Deleted Users

If we want to permanently remove users who have been soft deleted for a certain period, we can run a background job to delete those records:

```
DELETE FROM users WHERE is_deleted = true AND deletion_date < NOW() - INTERVAL '30 days';
```

Software Design Patterns

[10 Design Patterns Explained in 10 Minutes](#)

Software Design Patterns

What Are Software Design Patterns?

Software design patterns provide reusable solutions to common software design problems. They help standardize approaches, making code easier to understand, maintain, and extend. The influential book *Design Patterns* by the "Gang of Four" categorizes design patterns into three types:

- **Creational Patterns:** Handle object creation mechanisms.
- **Structural Patterns:** Define how objects and components relate.
- **Behavioral Patterns:** Govern object communication and workflows.

Using design patterns effectively can improve code quality, but excessive or incorrect use may introduce unnecessary complexity.

Key Software Design Patterns

Singleton Pattern

The **Singleton pattern** ensures that only one instance of a class exists and provides a global point of access.

Use Cases: [Database](#) connections, Logging services, Global configurations

Example (JavaScript):

```
class Singleton {
  constructor() {
    if (!Singleton.instance) {
      Singleton.instance = this;
    }
    return Singleton.instance;
  }
}

const instance1 = new Singleton();
const instance2 = new Singleton();
console.log(instance1 === instance2); // true
```

Prototype Pattern

The **Prototype pattern** allows new objects to be created by cloning an existing object rather than instantiating a new class.

Use Cases: Performance optimization, Object cloning

Example (JavaScript):

```
const carPrototype = {
  start: function () {
    console.log("Engine started!");
  }
};

const myCar = Object.create(carPrototype);
myCar.start(); // Engine started!
```

Builder Pattern

The **Builder pattern** simplifies object creation when multiple configuration options exist.

Use Cases: Constructing complex objects, UI component creation

Example (JavaScript):

```
class CarBuilder {
  constructor() {
    this.car = {};
  }
  setColor(color) {
    this.car.color = color;
    return this;
  }
  setWheels(wheels) {
    this.car.wheels = wheels;
    return this;
  }
  build() {
    return this.car;
  }
}
const myCar = new CarBuilder().setColor("red").setWheels(4).build();
console.log(myCar);
```

Factory Pattern

The **Factory pattern** encapsulates object creation logic, making code more modular and easier to extend.

Use Cases: Dependency injection, Platform-specific object creation

Example (JavaScript):

```
class CarFactory {
  static createCar(type) {
    const carTypes = {
      sedan: { type: "sedan", doors: 4 },
      coupe: { type: "coupe", doors: 2 },
    };
    return carTypes[type] || null;
  }
}
const myCar = CarFactory.createCar("sedan");
console.log(myCar);
```

Facade Pattern

The **Facade pattern** provides a simplified interface to a complex system.

Use Cases: Simplifying APIs, Reducing dependencies

Example (JavaScript):

```
class Computer {
  start() { console.log("Computer starting..."); }
  shutdown() { console.log("Computer shutting down..."); }
}

class ComputerFacade {
  constructor() {
    this.computer = new Computer();
  }
  turnOn() {
    this.computer.start();
  }
  turnOff() {
    this.computer.shutdown();
  }
}

const myComputer = new ComputerFacade();
myComputer.turnOn();
```

Proxy Pattern

The **Proxy pattern** acts as an intermediary to control access to an object.

Use Cases: Lazy loading, [Security](#) proxies

Example (JavaScript):

```

class RealImage {
  constructor(filename) {
    this.filename = filename;
  }
  display() {
    console.log("Displaying " + this.filename);
  }
}
class ProxyImage {
  constructor(filename) {
    this.realImage = null;
    this.filename = filename;
  }
  display() {
    if (!this.realImage) {
      this.realImage = new RealImage(this.filename);
    }
    this.realImage.display();
  }
}
const image = new ProxyImage("test.jpg");
image.display();

```

Iterator Pattern

The **Iterator pattern** provides a way to access elements of a collection sequentially without exposing its internal structure.

Use Cases: Collection traversal, Data processing

Example (JavaScript):

```

class Iterator {
  constructor(items) {
    this.items = items;
    this.index = 0;
  }
  next() {
    return this.index < this.items.length ? { value: this.items[this.index++], done: false } : { done: true };
  }
}
const iterator = new Iterator(["a", "b", "c"]);
console.log(iterator.next());
console.log(iterator.next());
console.log(iterator.next());

```

Observer Pattern

The **Observer pattern** enables a one-to-many dependency between objects, ensuring changes to one object are reflected in its dependents.

Use Cases: Event handling, Reactive programming

Example (JavaScript):

```
class Subject {
  constructor() {
    this.observers = [];
  }
  subscribe(observer) {
    this.observers.push(observer);
  }
  notify(data) {
    this.observers.forEach(observer => observer.update(data));
  }
}
class Observer {
  update(data) {
    console.log("Received update: " + data);
  }
}
const subject = new Subject();
const observer1 = new Observer();
subject.subscribe(observer1);
subject.notify("Hello World");
```

Mediator Pattern

The **Mediator pattern** centralizes communication between objects to reduce dependencies.

Use Cases: Chat applications, Workflow coordination

Example (JavaScript):

```

class Mediator {
  constructor() {
    this.participants = [];
  }
  register(participant) {
    this.participants.push(participant);
  }
  send(message, sender) {
    this.participants.forEach(participant => {
      if (participant !== sender) {
        participant.receive(message);
      }
    });
  }
}

class Participant {
  constructor(name, mediator) {
    this.name = name;
    this.mediator = mediator;
    mediator.register(this);
  }
  send(message) {
    this.mediator.send(message, this);
  }
  receive(message) {
    console.log(this.name + " received: " + message);
  }
}

const mediator = new Mediator();
const p1 = new Participant("Alice", mediator);
p1.send("Hello");

```

Software Development Life Cycle

A structured approach like the Software Development Life Cycle (SDLC) ensures systematic progression through various stages of development. SDLC remains relevant today by outlining the essential stages a product must undergo to achieve success.

SDLC Stages

The SDLC comprises several phases, each critical to the overall development process:

1. Planning and Analysis

- **Purpose:** Collect business and user requirements, perform cost and time estimation, and conduct scoping activities.

Introduction

- **Activities:** Define what the final product must do and how it should work. This phase may include a separate requirements analysis stage.
-

2. Designing

- **Purpose:** Prepare the product's architecture and design.
- **Activities:** A software architect sets the high-level structure of the future system, selecting technology and drafting user experience and visual design.

3. Development

- **Purpose:** Transform the design into actual code.
- **Activities:** Programmers write code according to the design specifications, revealing some aspects of the final product to stakeholders.

4. Testing

- **Purpose:** Ensure the quality and functionality of the product.
- **Activities:** Testers and QA professionals review the code and usability, identifying bugs and errors for correction before deployment.

5. Deployment

- **Purpose:** Release the product to users.
- **Activities:** Launch the product, making it available for use. Often combined with the maintenance phase.

6. Maintenance

- **Purpose:** Provide ongoing support and updates.
- **Activities:** Gather user feedback, fix issues, and add new features as needed.

Additionally, some projects include a **Prototyping** stage between planning and designing to validate ideas with minimal effort and cost.

SDLC Methodologies

Waterfall

The Waterfall model, strictly follows the SDLC stages sequentially. **Each phase must be completed before the next begins, making it straightforward and easy to manage.** However, its lack of flexibility and late testing phase often lead to delays and budget overruns when changes are needed. Out of favour. Opposite of flexible, needs roll backs.

- detail documentation

Agile

- Produce only essential documentation. Collaborative effort.
- Focuses on features users will like.

In response to the rigid Waterfall model, Agile emerged in the early 2000s with a **focus on flexibility and continuous delivery.** Agile methodologies like Scrum, Lean, and Extreme Programming (XP) integrate testing throughout the development process and welcome changes even in late stages.

See agile manifesto : Working software over documentation

- **Scrum:** Breaks development into short cycles called **sprints**, each lasting 1-4 weeks. At the end of each sprint, **a functional product increment is presented to stakeholders**, allowing for quick adaptation based on feedback.
 - **Lean:** Aims to **eliminate waste** through a build-measure-learn feedback loop, continuously refining the product. Steps: build, measure, learn.
-

- **Extreme Programming (XP):** Emphasizes technical excellence with practices such as test-driven development, code refactoring, and pair programming.
- Kanban: management method for efficiency.

DevOps

SparseCategoricalCrossentropy Or CategoricalCrossentropy

To understand the differences and use cases for `SparseCategoricalCrossentropy` and `CategoricalCrossentropy` in [TensorFlow](#), let's break down each one:

CategoricalCrossentropy

- **Use Case:** This [loss function](#) is used when you have one-hot encoded labels. [One-hot encoding](#) means that each label is represented as a vector with a length equal to the number of classes, where the correct class is marked with a 1 and all other classes are marked with 0s.
- **Example:** If you have three classes, a label might look like `[0, 1, 0]` for class 2.
- **Functionality:** It calculates the [cross entropy](#) loss between the true labels and the predicted probabilities.

SparseCategoricalCrossentropy

- **Use Case:** This loss function is used when your labels are integers instead of one-hot encoded vectors. Each label is represented by a single integer corresponding to the correct class.
- **Example:** If you have three classes, a label might simply be `1` for class 2.
- **Functionality:** It also calculates the cross-entropy loss but expects the labels to be in integer form, which can be more memory efficient.

Key Differences

- **Input Format:** The main difference is the format of the labels. `CategoricalCrossentropy` requires one-hot encoded labels, while `SparseCategoricalCrossentropy` works with integer labels.
- **Efficiency:** `SparseCategoricalCrossentropy` can be more efficient in terms of memory and computation, especially when dealing with a large number of classes.

When to Use Which

- Use `CategoricalCrossentropy` if your labels are already one-hot encoded or if you prefer to work with one-hot encoded labels for any specific reason.
- Use `SparseCategoricalCrossentropy` if your labels are integers, which is often the case when labels are directly loaded from datasets.

Specificity

Specificity, also known as the true negative rate, measures the proportion of actual negatives that are correctly identified by the model. It indicates how well the model is at identifying negative instances. Formula:

$$\text{Specificity} = \frac{TN}{TN+FP}$$

- Specificity is crucial in scenarios where it is important to correctly identify negative instances, such as in medical testing where a false positive could lead to unnecessary treatment.
-

Spreadsheets Vs Databases

Compared to spreadsheets, databases offer:

1. **Scalability**: Databases are designed to handle large volumes of data, making them suitable for applications with millions or even billions of records. In contrast, spreadsheets can become unwieldy and slow when dealing with large datasets.
2. **Update Frequency**: Databases support real-time updates and continuous operations, allowing for dynamic [data management](#). Spreadsheets, on the other hand, are more static and may require manual updates, which can lead to outdated information.
3. **Speed**: Databases are optimized for quick access, retrieval, and manipulation of data. They can efficiently handle multiple concurrent users, making them ideal for environments where data is frequently accessed and modified. Spreadsheets can lag in performance under similar conditions.

Tags

- **Tags**: #data_management, #data_storage

Stacking

What is [Stacking](#)?; is an [Model Ensemble](#) combines predictions of multiple base models [by training a meta-model](#) on the outputs of the base models.

Standard Deviation

Standard deviation is a statistical measure that quantifies the amount of variation or dispersion in a set of data values. It indicates how much individual data points deviate from the mean (average) of the dataset.

Formula

For a dataset with n observations X_1, X_2, \dots, X_n , the standard deviation σ is calculated using the formula:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \mu)^2}$$

Where:

- σ = standard deviation
 - n = number of observations
 - X_i = each individual observation
 - μ = mean of the dataset, calculated as:
-

$$\mu = \frac{1}{n} \sum_{i=1}^n X_i$$

Why Standard Deviation is Preferred Over Variance

1. Same Units as Data

Standard deviation is expressed in the same units as the original data, making it more [interpretability|interpretable](#).

- o **Example:** If you measure height in centimeters, the standard deviation will also be in centimeters.
- o **Contrast:** Variance is expressed in squared units (e.g., square centimeters), which can be less intuitive to understand.

2. Direct Interpretation

Standard deviation provides a direct measure of the average distance of data points from the mean.

- o A **small standard deviation** indicates that the data points are close to the mean.
- o A **large standard deviation** suggests that the data points are more spread out.

3. Normal Distributions|Distribution Context

In the context of a normal distribution, standard deviation helps in understanding the spread of data:

- o Approximately **68%** of the data falls within **one standard deviation** of the mean.
- o About **95%** falls within **two standard deviations**.
- o About **99.7%** falls within **three standard deviations** (known as the empirical rule).

This property is particularly useful for identifying [standardised/Outliers](#).

4. Ease of Communication

Standard deviation is more intuitive and easier to communicate to a broader audience, including those without a strong statistical background. Its direct relation to the data makes it a preferred choice for explaining variability.

Standardisation

Standardisation is a [Preprocessing|data preprocessing](#) technique used to [Data Transformation](#) features so that they have a mean of 0 and a standard deviation of 1. Centers data with zero mean and unit variance, suitable for algorithms sensitive to variance.

Definition: Standardisation involves rescaling the features of your data so that they have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean of the feature from each data point and then dividing by the standard deviation.

Purpose:

- Useful for algorithms that assume the data is normally distributed.
- Uniformity: It helps in bringing all features to the same scale.

Use Case

- **Centred Data Assumption:** Standardisation is beneficial when the model assumes that the data is centred around zero. This is common in algorithms such as linear regression, logistic regression, and [principal component analysis \(PCA\)](#), and distance-based algorithms like [K-nearest neighbours|KNN](#) and [Gradient Descent](#) descent optimization.
- **Improved Performance:** It can improve the performance and convergence speed of machine learning algorithms by ensuring that each feature contributes equally to the result.

Formula

The formula for standardisation is:

$$z = \frac{x - \mu}{\sigma}$$

Where:

- x is the original data point.
- μ is the mean of the feature.
- σ is the standard deviation of the feature.

By applying this transformation, the data becomes more suitable for training models that rely on the assumption of [Gaussian Distribution](#).

Rescales the data to have a mean of 0 and a standard deviation of 1 (unit variance). This method is particularly useful when the data follows a Gaussian distribution.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_standardized = scaler.fit_transform(df) # Rescales each feature to have mean 0 and std deviation 1
```

Star Schema

Star Schema

- This schema consists of a central fact table surrounded by dimension tables. The fact table contains quantitative data, while the dimension tables provide descriptive attributes. The star schema is easy to

understand and query, making it popular for OLAP applications.

Star Schema

- Description: A simple and widely used form of dimensional modeling where a central fact table is connected to multiple dimension tables.
- Advantages: Easy to understand and query, with straightforward joins between fact and dimension tables.

Statistical Assumptions

Statistical assumptions are essential conditions that must be met for various statistical methods and models to produce valid results. Necessary for robustness and reliability of statistical analyses. If any assumptions are violated, it may be necessary to employ alternative statistical methods or to transform the data accordingly.

Purpose of Statistical Assumptions:

- [Data Analysis](#): Ensures that the chosen statistical methods are appropriate for the data.
- [Interpretability](#): Facilitates accurate interpretation of results and conclusions drawn from analyses.

Key Assumptions:

- [Assumption of Normality](#): This assumption posits that the data follows a normal distribution. Many [Statistical Tests](#), such as t-tests and ANOVA, rely on this assumption to validate their results. If the data is not normally distributed, alternative methods or transformations may be necessary. Heavy tailed distributions can violate this.
- Homoscedasticity: This refers to the assumption that the variance of the residuals (errors) remains constant across all levels of the independent variable(s). Violations of this assumption can lead to inefficient estimates and impact [hypothesis testing](#).
- [Independence](#): This assumption states that the observations in the dataset should be independent of one another. Dependence among observations can result in biased estimates and incorrect conclusions.
- Normality of Residuals: In regression analysis, it is assumed that the residuals (the differences between observed and predicted values) are normally distributed. This is critical for conducting hypothesis tests on regression coefficients.

Broader Categories of Assumptions:

Model Assumptions: These are overarching assumptions that apply to specific statistical models. For example:

- Linear Regression: Assumes a [linear relationship](#) between the independent and dependent variables.
- [Logistic Regression](#): Assumes a binary outcome for the dependent variable.

Distribution Assumptions: Different statistical tests make specific assumptions about the distribution of the data:

- Parametric Tests: Assume that the data follows a certain distribution (e.g., normal).
- Non-parametric Tests: Do not require such distributional assumptions and can be applied to data that does not meet these criteria.

Additional Considerations:

Testing Assumptions: It is important to test these assumptions before conducting statistical analyses. Common methods include:

Introduction

- Visual Inspection: Using plots (e.g., Q-Q plots, residual plots) to visually assess normality and homoscedasticity.
- Statistical Tests: Employing tests like the Shapiro-Wilk test for normality or Levene's test for homoscedasticity.

Consequences of Violating Assumptions: Understanding the implications of assumption violations is crucial. For example, violating the assumption of normality can lead to:

- Increased Type I or Type II error rates.
- Misleading confidence intervals and p-values.

Transformations and Alternatives: When assumptions are violated, consider:

- Data Transformations: Such as log, square root, or Box-Cox transformations to meet assumptions.
- Alternative Methods: Using robust statistical techniques that are less sensitive to assumption violations, such as bootstrapping or non-parametric tests.
- Contextual Relevance: The relevance of specific assumptions may vary depending on the context of the analysis and the nature of the data. Always consider the specific characteristics of the dataset when evaluating assumptions.

Statistical Tests

Statistical tests are methods used to determine if there is a significant difference between groups or if a relationship exists between variables. Each test has its specific [Statistical Assumptions](#) and applications.

Types of Statistical Tests

[Z-Test](#)

[T-test](#)

[Chi-Squared Test](#)

[Proportion Test](#)

Test Statistics

For each statistical test, a test statistic is calculated. This statistic measures the degree of deviation from the null hypothesis ([Hypothesis testing](#)). The [estimator](#) is centered by the population mean, and then it is divided by the population standard deviation, a process known as [Standardisation](#).

Statistics

Statistics want to understand the world. The world is made of probabilities, we model probabilities with functions, and we model functions with parameters.

"Observe data and construct models, infer and refine hypotheses "

Portal for all statistics notes:

Statistical theorems:

- Asymptotic Theorem: Law of large numbers: Sample mean approaches the population mean.

Introduction

- finite mean assumption

Statistical Assumptions

Type 1 error and Power

Distributions

Statistical Tests

Monte Carlo Simulation

Logistic Regression: model how change and covariance influence the odds of an event

Proportional Hazard Model: time to an event

Hypothesis testing p values Confidence Interval

Central Limit Theorem

Correlation vs Causation

Markov chain

parametric vs non-parametric tests

Multicollinearity

univariate vs multivariate

R tidyverse: visualisation in R

Over parameterised models

Causal Inference

Bootstrap

Adaptive decision analysis: interrupting the experiment in the middle

Estimation Problems: using data to estimate model parameters

- Maximum Likelihood Estimation
- Expectation Maximisation Algorithm

Likelihood ratio: Type 1 error and Power UMP test. used to maximise power. T-test is a consequence of this.

Dashboards/Animations: Shiny, ganimate

Estimator

Stemming

Shorting words to the key term.

Stochastic Gradient Descent

Gradient Descent

Stored Procedures

Stored procedures are a way to automate SQL statements, allowing them to be executed repeatedly without rewriting the code.

Demonstration with the Boston MFA Database

We will use the Boston MFA database to illustrate stored procedures. Previously, we implemented a soft-delete feature for the `collections` table using views. Now, we will create a stored procedure to achieve similar functionality.

1. Select the Database:

```
USE `mfa`;
```

2. Add a Deleted Column:

The `deleted` column needs to be added to the `collections` table to track soft deletions.

```
ALTER TABLE `collections`
ADD COLUMN `deleted` TINYINT DEFAULT 0;
```

The `TINYINT` type is appropriate since the column will only hold values of 0 or 1, with a default of 0 to retain all existing collections.

3. Change the Delimiter:

Before creating a stored procedure, change the delimiter to allow multiple statements.

```
delimiter //
```

4. Create the Stored Procedure:

Define the stored procedure to select current collections that are not marked as deleted.

```
CREATE PROCEDURE `current_collection`()
BEGIN
    SELECT `title`, `accession_number`, `acquired`
    FROM `collections`
    WHERE `deleted` = 0;
END//
```

5. Reset the Delimiter:

After creating the procedure, reset the delimiter back to the default.

```
delimiter ;
```

6. Call the Stored Procedure:

Execute the procedure to see the current collections.

```
CALL current_collection();
```

7. Soft Delete an Item:

If we soft-delete an item, such as “Farmers working at dawn,” and call the procedure again, the deleted row will not appear in the output.

```
UPDATE `collections`  
SET `deleted` = 1  
WHERE `title` = 'Farmers working at dawn';
```

Parameters in Stored Procedures

Stored procedures can accept parameters. For example, we can create a procedure to handle the sale of artwork.

1. Create the Transactions Table:

```
CREATE TABLE `transactions` (  
    `id` INT AUTO_INCREMENT,  
    `title` VARCHAR(64) NOT NULL,  
    `action` ENUM('bought', 'sold') NOT NULL,  
    PRIMARY KEY(`id`)  
);
```

2. Create the Sell Procedure:

This procedure updates the `collections` table and logs the transaction.

```

delimiter //
CREATE PROCEDURE `sell`(`sold_id` INT)
BEGIN
    UPDATE `collections` SET `deleted` = 1
    WHERE `id` = `sold_id`;
    INSERT INTO `transactions` (`title`, `action`)
    VALUES ((SELECT `title` FROM `collections` WHERE `id` = `sold_id`), 'sold');
END//
delimiter ;

```

3. Call the Sell Procedure:

To sell a specific item, call the procedure with the item's ID.

```
CALL `sell`(2);
```

Considerations

- **Multiple Calls:**

If the `sell` procedure is called with the same ID multiple times, it may lead to multiple entries in the `transactions` table. Logic can be added to prevent this.

- **Programming Constructs:**

Stored procedures can be enhanced with programming constructs available in MySQL, allowing for more complex logic.

Strongly Vs Weakly Typed Language

A **strongly typed** programming language is one where **types** are strictly enforced. This means that once a variable is assigned a type, it cannot be implicitly converted to another type without an explicit conversion. The goal is to **minimize errors related to incorrect type handling**, as the compiler or interpreter will detect type mismatches early in the development process.

Characteristics of a Strongly Typed Language:

1. **Type Enforcement:** The language does not allow operations between incompatible types (e.g., trying to add a string to an integer).
2. **Explicit Conversions:** If you need to change the type of a variable, you must explicitly convert it (casting). The compiler or interpreter won't do it automatically.
3. **Compile-time/Runtime Type Checking:** The language performs thorough checks either at compile time (for compiled languages) or at runtime (for interpreted languages) to ensure type safety.

Example:

In [Java](#), which is a strongly typed language:

```
int number = 5;
String text = "Hello";

// The following line will result in an error since you cannot add an integer to a string directly:
text = text + number; // Type mismatch error

// You must explicitly convert the integer to a string to perform this operation:
text = text + Integer.toString(number); // Correct
```

Benefits:

- **Error Prevention:** Type mismatches are caught early, reducing runtime errors.
- **Code Clarity:** Since types are explicitly defined, it's easier to understand what kind of data is being handled.
- **Efficiency:** Some strongly typed languages can optimize code better due to the predictability of data types.

Contrast with Weakly Typed Languages:

Weakly typed languages, like [JavaScript](#), allow implicit type conversions, leading to more flexibility but also potential runtime errors due to unexpected conversions:

```
let number = 5;
let text = "Hello";

// JavaScript allows this, and it implicitly converts the number to a string:
text = text + number; // No error, result is "Hello5"
```

Structuring And Organizing Data

Structuring and organizing data.

In [DE_Tools](#) see:

- https://github.com/rhyslwellis/DE_Tools/blob/main/Explorations/Transformation/multi_index.ipynb
- https://github.com/rhyslwellis/DE_Tools/blob/main/Explorations/Transformation/reshaping.ipynb

Related terms:

- [Multi-level index](#)

Summarisation

Summarization in NLP

Summarization in natural language processing (NLP) is the process of condensing a text document into a shorter version while retaining its main ideas and key information. There are two primary forms of summarization:

The unsupervised summarization process involves splitting text, tokenizing sentences, assigning scores based on importance, and selecting top sentences. Effective scoring methods include calculating sentence similarity and analyzing word frequencies to ensure that the summary captures the essence of the original text.

Extraction:

- This method involves selecting specific words or sentences directly from the original text to create a summary. It focuses on identifying and pulling out the most important parts of the text without altering the original wording.

Abstraction:

- The abstraction method generates a summary that may include new words and phrases not present in the original text. This approach is more complex as it requires understanding the content and rephrasing it, often using techniques like paraphrasing.

Unsupervised Summarization Process

The basic idea behind unsupervised summarization involves the following steps:

1. **Split Text into Sentences:** The text is divided into individual sentences for analysis.
2. **Tokenize Sentences:** Each sentence is tokenized into separate words, allowing for detailed examination of word usage.
3. **Assign Scores to Sentences:** Sentences are evaluated based on their importance, which is a crucial step in the summarization process.
4. **Select Top Sentences:** The highest-scoring sentences are selected and displayed in their original order to form the summary.

Methods for Assigning Scores

The main point of summarization is effectively assigning scores to sentences. Here are some common methods for doing this:

- **Similarity Calculation:** Calculate the similarity between each pair of sentences and select those that are most similar to the majority of sentences. This helps identify sentences that capture the central themes of the text.
- **Word Frequencies:** Analyze word frequencies to identify the most common words in the text. Sentences that contain a higher number of these frequent words are then selected for the summary.

Supervised Learning

Supervised learning is a type of machine learning where an algorithm learns from labeled data to make predictions or decisions.

In supervised learning, the training data consists of input-output pairs, where each input (features) is associated with a known output (label or target).

The algorithm's goal is to learn a mapping from the input to the output so that it can predict the output for new, unseen data.

Examples of Supervised Machine Learning Algorithms:

- K-nearest neighbours (KNN)
- Naive Bayes
- Decision Tree
- Linear Regression
- Support Vector Machines (SVM)

Key Characteristics of Supervised Learning:

- **Labeled Data:** The training dataset contains both the input data and the corresponding correct outputs (labels).
- **Training Phase:** The model is trained using this labeled data to minimize errors in predicting the output.
- **Prediction:** After training, the model can predict the output (label) for new data based on what it learned.

Types of Supervised Learning Algorithms:

Supervised learning algorithms learn from **labeled data**, where each example is associated with a target label.

Labelled data can look like the following:

Email Content	Label
"Congratulations! You won a free iPhone."	Spam
"Meeting scheduled for 2 PM tomorrow."	Not Spam
"Special offer: Buy 1 get 1 free!"	Spam
"Please find the attached report."	Not Spam

Labelling can be expensive and time consuming.

- **Classification:** Predicts discrete labels (e.g., categories).
- **Regression:** Predicts continuous values.

Example:

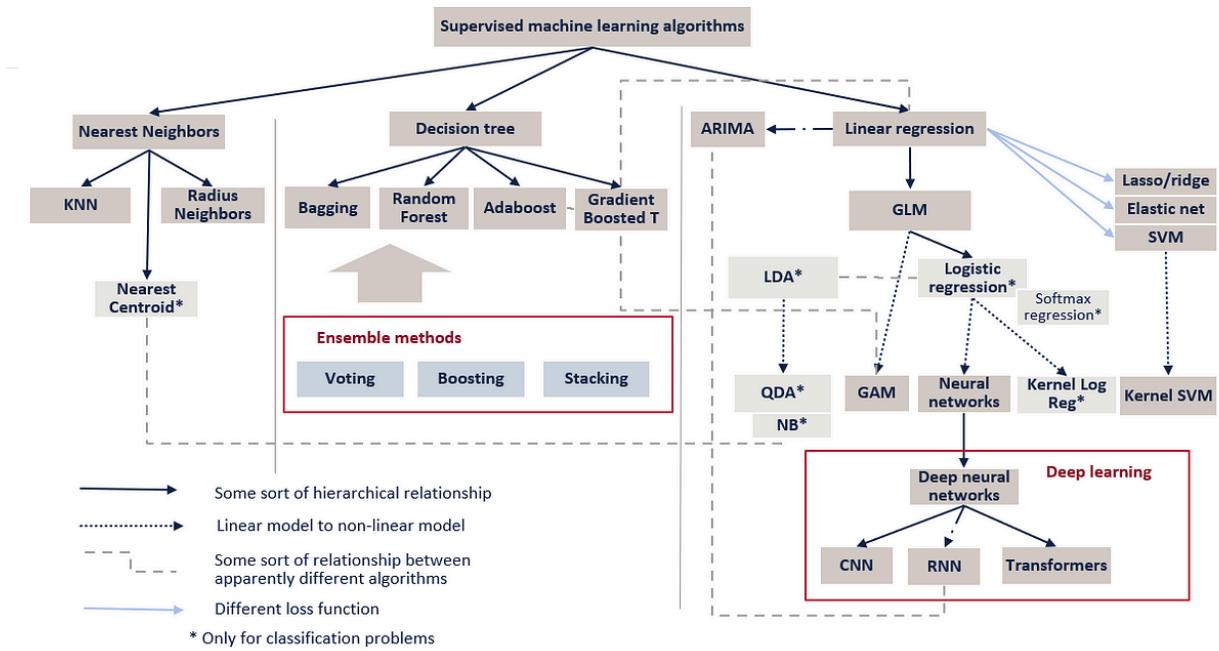
In a **house price prediction** task:

- The input features could be the size of the house, number of rooms, and location.
- The output (label) would be the price of the house.

The model is trained on a dataset where the house prices are known (labeled data), and it learns to predict the price for new houses.

Code implementation

That is there is a `y_train`, then uses to get `y_pred` (from `X_test`) and compare against `y_test`.



Support Vector Classifier (SVC)

Overview

Support Vector Classifiers (SVCs) are a fundamental component of [Support Vector Machines|SVMs](#), designed to find the optimal hyperplane that separates data into distinct classes. The primary objective of an SVC is to maximize the margin between different classes, ensuring that the separation is as clear as possible.

Key Concepts

- **Hyperplane:** A decision boundary that separates different classes in the feature space. In a two-dimensional space, this is a line; in higher dimensions, it becomes a plane or hyperplane.
- **Support Vectors:** The data points that are closest to the hyperplane. These points are critical as they define the position and orientation of the hyperplane.
- **Margin:** The distance between the hyperplane and the nearest data point from either class. SVC aims to maximize this margin to improve classification robustness.

SVC vs. SVM

- **SVC:** Primarily focuses on placing a hyperplane between datasets for separation. It is effective when the data is linearly separable.
- **SVM:** Extends the concept of SVC by using kernel functions to handle cases where data is **not linearly separable** in its original space. Kernels transform the data into a higher-dimensional space where a linear separation is possible.

Support Vector Machines

Support Vector Machines (SVM) are a type of [supervised learning](#) algorithm primarily used for [classification](#) tasks, though they can also be adapted for regression. The main idea is to find an optimal hyperplane that divides data into different classes by maximizing the margin between them. The support vectors are the data points closest to the hyperplane, influencing its position and orientation.

Key Features

- Hyperplane: Finds a hyperplane that maximizes the margin between classes.
- High-Dimensional Spaces: Robust in high-dimensional spaces, such as image and text classification.

Advantages

- Highly effective for high-dimensional data (datasets with many features).
- Useful for classification tasks where a clear margin of separation exists between classes.

Disadvantages

- Can be computationally expensive for large datasets and sensitive to the choice of hyperparameters.
- Performance is highly dependent on the [Kernelling](#) choice, requiring careful tuning.

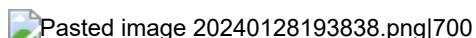
How SVM Works

In [ML_Tools](#) see: [SVM_Example.py](#)

1. Initial Space: Start in the low-dimensional space, where the data may not be linearly separable.
2. Kernel Function: Use a [Kernelling](#) function to move the data into a higher dimension where separation is clearer.
3. Hyperplane Placement: Place hyperplanes (decision boundaries) between the data clusters to classify the data.

Margins

- Outliers and Soft Margins: SVM allows for some miscalculations or errors in classification to handle outliers. This is part of the [Bias and variance](#) tradeoff, where the model is allowed to make a few mistakes to improve generalization.
- Soft Margins: Allow some data points to be within the margin or even on the wrong side of the hyperplane, enabling SVM to handle imperfect separations.



Support Vector Regression

Support Vector Regression use similar principles to [Support Vector Machines|SVMs](#) but for predicting continuous variables.

Symbolic Computation

[Mathematical Reasoning in Transformers](#)

Summary of Wolfram Alpha's Approach:

1. **Uses symbolic computation** with precise algorithms.
2. **Leverages predefined mathematical rules** for various domains.
3. **Provides step-by-step solutions** to explain problem-solving processes.
4. **Handles natural language inputs** and translates them into mathematical expressions.
5. **Produces both exact and numerical solutions**, depending on the problem type.
6. **Visualizes results** with graphs and interactive displays.
7. **Accesses a curated knowledge base** for real-world data integration.

Example: Wolfram alpha

Wolfram Alpha is designed specifically for symbolic computation and uses rule-based algorithms to perform exact and precise mathematical operations. Here's an overview of how Wolfram Alpha processes math problems:

1. Symbolic Computation Engine

- Wolfram Alpha is powered by **Mathematica**, a robust computational engine that specializes in **symbolic computation**. Unlike neural network models, which rely on statistical pattern recognition, symbolic computation manipulates symbols and formulas directly according to established mathematical rules. This allows Wolfram Alpha to handle a wide variety of mathematical problems with precision, from basic arithmetic to complex calculus and algebraic manipulations.

2. Predefined Mathematical Rules and Algorithms

- Wolfram Alpha uses **predefined algorithms and mathematical rules** that are coded into the system.
- Each mathematical operation is treated according to formal rules, so Wolfram Alpha can handle exact symbolic results (like expressing results in terms of radicals or π) or provide numerical approximations when needed.

3. Step-by-Step Solutions

5. Natural Language Processing (NLP)

- Wolfram Alpha uses **NLP** to interpret queries that are entered in natural language. For example, a user might type "solve $x^2 + 2x + 1 = 0$ " or simply "solve quadratic equation," and Wolfram Alpha translates this into formal mathematical expressions to process through its symbolic engine.
- This NLP capability allows users to input problems in a variety of ways, making it more accessible to non-expert users.

6. Exact vs. Numerical Solutions

- Wolfram Alpha can handle both **exact symbolic solutions** (such as expressing a result in terms of fractions, square roots, or constants like π) and **numerical approximations**. For example, for equations that have no simple symbolic solutions, it can provide highly accurate numerical answers using **numerical methods** such as Newton's method or Monte Carlo simulations.

7. Knowledge-Based System

- Wolfram Alpha is connected to a vast database of curated knowledge, not only in mathematics but across many disciplines. This allows it to draw on data, formulas, and algorithms to solve not only pure math problems but also applied math problems, such as in physics, engineering, and economics.

8. Graphing and Visualization

9. Error Handling and Interpretation

- Wolfram Alpha handles user input carefully, identifying potential errors or ambiguities. For example, if an equation is underdetermined (too few equations for the number of variables), it may provide a parametric solution. If input is ambiguous, it often offers multiple possible interpretations or asks the user to clarify.

Sympy

Semantic Layer

A [Semantic Layer](#) is much more flexible and makes the most sense on top of [transformed data](#) in a [Data Warehouse](#).

A semantic layer in the context of a data warehouse is an abstraction layer that sits between the raw data stored in the warehouse and the end users who need to access and analyze that data.

Its primary purpose is to simplify complex data structures and present them in a more user-friendly and business-oriented way. This allows users to interact with the data without needing to understand the underlying complexities of the database schema or query languages.

Bridging the gap between complex data systems and business users, enabling more effective and efficient data-driven decision-making.

Avoid extensive reshuffles or reprocesses of large amounts of data.

Think of [OLAP](#).md) cubes where you can dice-and-slice ad-hoc on significant amounts of data without storing them ahead of time

Key Features of a Semantic Layer

1. Business-Friendly Terminology:

- Translates technical database terms into business-friendly language that is easier for non-technical users to understand.
- For example, instead of using column names like `cust_id` or `prod_sku`, the semantic layer might present them as "Customer ID" or "Product SKU."

2. Data Abstraction:

- Hides the complexity of the underlying data model, such as joins, table structures, and data transformations.
- Users can focus on business concepts rather than technical details.

3. Consistent [Metric](#) and Calculations:

- Provides a centralized definition of key metrics and calculations, ensuring consistency across reports and analyses.

Introduction

- For example, a metric like "Total Revenue" would be consistently calculated and presented, regardless of who is querying the data.

4. Security and Access Control:

- Implements security rules and access controls to ensure that users only see data they are authorized to access.
- This can include row-level security, column-level security, and user-specific data views.

5. Enhanced Query Performance:

- Optimizes queries by pre-aggregating data or using materialized views, reducing the load on the data warehouse and improving response times for users.

Benefits of a Semantic Layer

- Ease of Use: Makes it easier for business users to access and analyze data without needing deep technical knowledge.
- Faster Insights: Users can quickly generate reports and dashboards using familiar business terms and concepts.
- Consistency: Ensures that all users are working with the same definitions and calculations, reducing discrepancies in reporting.
- Scalability: Supports a wide range of analytical tools and applications, allowing organizations to scale their data analytics capabilities.

Implementation

A semantic layer can be implemented using various tools and technologies, such as:

- Business Intelligence (BI) Tools: Many BI platforms, like [Tableau](#), [PowerBI](#), and Looker, offer built-in semantic layer capabilities.
- [Data Virtualization](#) Tools: Tools like Denodo or Dremio provide semantic layer functionality by creating virtual views of data.
- Custom Solutions: Organizations can build custom semantic layers using middleware or data modeling tools.

Semi Structured Data

Semi-structured data is data that lacks a rigid structure and that does not conform directly to a data model, but that has tags, metadata, or elements that describe the data.

Examples of semi-structured data are JSON or [XML](#) files.

Semi-structured data often contains enough information that it can be relatively easily converted into [structured data](#).

[JSON](#) data embedded inside of a string, is an example of semi-structured data. The string contains all the information required to understand the structure of the data, but is still for the moment just a string -- it hasn't been structured yet.

	data
Record 1	\\"{"id": 1, "name": "Mary X"}\\"
Record 2	\\"{"id": 2, "name": "John D"}\\"

It is often relatively straightforward to convert semi-structured data into structured data. Converting semi-structured data into structured data is often done during the [Data Transformation](#) stage in an [ETL](#) or [ELT](#) process.

Shapefile

A shapefile is a popular geospatial vector data format for geographic information system (GIS) software. It is widely used for storing the location, shape, and attributes of geographic features. Developed by Esri, shapefiles are commonly used in the GIS community for exchanging and managing geospatial data.

A shapefile is a widely used [GIS](#) vector data format consisting of multiple files that store both spatial geometry and attribute data. Its ease of use and broad compatibility have made it a standard format for geospatial data exchange and analysis in the GIS community.

Components of a Shapefile

A shapefile is not a single file, but rather a set of several files that work together. The primary components include:

1. **.shp file:** This file contains the geometry data (points, lines, polygons) that represents the spatial features.
2. **.shx file:** This is an index file that allows for quick access to the geometry data in the .shp file.
3. **.dbf file:** This file stores attribute data in tabular format, linked to the spatial data in the .shp file. It uses the dBASE format to hold the attributes of each shape, such as names, categories, or other descriptive information.

In addition to these mandatory files, a shapefile can also include several optional files that provide additional information:

1. **.prj file:** Contains the coordinate system and projection information for the spatial data. This file is crucial for ensuring that the data is displayed correctly in GIS software.
2. **.cpg file:** Defines the character encoding to be used for the .dbf file, ensuring that text attributes are interpreted correctly.
3. **.qix file:** An optional spatial index file that can improve the performance of spatial queries on the shapefile.

Characteristics of Shapefiles

- **Geometry Types:** Shapefiles can store different types of geometric data, including points, lines, and polygons. However, each shapefile can contain only one type of geometry.
- **Attribute Data:** The .dbf file allows shapefiles to store descriptive data about each spatial feature, which can be used for analysis and mapping.
- **Limitations:** Shapefiles have some limitations, such as a maximum file size of 2 GB for each component file, lack of support for advanced geometric types (like curves), and potential data redundancy and inefficiencies.

Usage of Shapefiles

Shapefiles are extensively used in GIS for various purposes, including:

- **Mapping:** Displaying geographic features on maps for visualization.
- **Spatial Analysis:** Performing spatial queries, analysis, and geoprocessing tasks.
- **Data Exchange:** Sharing geospatial data between different GIS software and systems.

Example Scenario

Consider a city planning department that wants to map all the parks within the city. They might use a shapefile to store the polygon geometries representing park boundaries along with attributes such as park names, areas, and facilities available. This shapefile can then be loaded into GIS software to create maps, analyze park distributions, and manage urban planning tasks.

Sklearn Datasets

make a dataframe by

Introduction

```
ds = datasets.load_dataset()
df = pd.DataFrame(ds.data,columns=ds.feature_names)
df.head()
#add target column
df['target'] = ds.target
```

Spacy {#spacy}

Storage Layer Object Store {#storage-layer-object-store}

A storage layer or object storage are services from the three big [Cloud Providers]({#cloud-providers}),

AWS S3,[S3 bucket]({#s3-bucket})

Azure Blob Storage,

and Google Cloud Storage.

The web user interface is easy to use. **Its features are very basic, where, in fact, these object stores store distributed

Structured Data {#structured-data}

Structured data refers to data that has been formatted into a well-defined schema ([Database Schema]({#database-schema})). A

Example of structure data

Below is an example of structured data as it would appear in a database:

	age	**name**	**phone**
Record 1	29	Bob	123-456
Record 2	30	Sue	789-123

It may seem that all data is structured, but this is not always the case -- data can be unstructured, or semi-structured.

Structured data vs. unstructured data

Structured data can be contrasted with [unstructured data]({unstructured%20data.md}), which does not conform to a data model

An simple example of unstructured data is a string that contains interesting information inside of it, but that has not been

	UnstructuredString
Record 1	"Bob is 29"
Record 2	"Mary just turned 30"

Introduction

```
## Structuring of unstructured data

Converting unstructured data into structured data can be done during the [Data Transformation](Data%20Transformation.md) s

For example, in order to efficiently make use of the unstructured data given in the previous example, it may desirable to

|           | **name** | **age** |
|-----| -----|----|
|Record 1| "Bob" | 29 |
|Record 2| "Mary" | 30 |

Storing the data in a structured manner makes it much more efficient to query the data. For example, after structuring the

``` SQL
SELECT * FROM X where Age=29

```

A query such as this would be expensive and/or more difficult to execute on unstructured data.

## Structured data vs. semi-structured data

Structured data can also be contrasted with [semi-structured data](#), which lacks a rigid structure and does not conform directly to a data model. However, semi-structured data has tags and elements that describe the data.

Examples of semi-structured data are [JSON](#) or [XML](#) files. Semi-structured data often contains enough information that it can be relatively easily converted into structured data.

[structured data](#) refers to data that has been formatted into a well-defined schema. An example would be data that is stored with precisely defined columns in a relational database or excel spreadsheet. Examples of structured fields could be age, name, phone number, credit card numbers or address.

## Syntactic Relationships

Syntactic relationships refer to the structural connections between words or phrases in a sentence, focusing on grammar and the arrangement of words. They determine how words combine to form phrases, clauses, and sentences, following the rules of syntax.

[Semantic relationships](#), on the other hand, deal with the meaning and interpretation of words and phrases. They focus on how words relate to each other in terms of meaning, such as synonyms, antonyms, and hierarchical relationships like hypernyms and hyponyms.

The key difference is that syntactic relationships are concerned with the form and structure of language, while semantic relationships are concerned with meaning and interpretation.

# T

---

## Table of Contents

- T-test
- TF-IDF
- TOML
- TS\_Anomaly\_Detection
- TS\_Anomaly\_Detection.py
- Tableau
- Tags
- Technical Debt
- Technical Design Doc Template
- Telecommunications
- Tensorflow
- Terminal commands
- Test Loss When Evaluating Models
- Testing
- Testing\_Pytest.py
- Testing\_unittest.py
- Text2Cypher
- Thinking Systems
- Time Series Forecasting
- Time Series Identify Trends and Patterns
- Time Series
- Tokenisation
- Train-Dev-Test Sets
- Transaction
- Transfer Learning
- Transformed Target Regressor
- Transformer
- Transformers vs RNNs
- Turning a flat file into a database
- TypeScript
- Types of Computational Bugs
- Types of Database Schema
- Types of Neural Networks
- Typical Output Formats in Neural Networks
- t-SNE
- tool.bandit
- tool.ruff
- tool.uv
- topic modeling
- transfer\_learning.py

# T Test

---

The T-test is a statistical method used to determine if there is a significant difference between the means of two groups, especially when the population standard deviation is unknown. It is particularly useful when dealing with small sample sizes.

## Types of T-tests

1. **One-Sample T-test:** This test compares the mean of a single sample to a known value (often the population mean). It helps determine if the sample mean significantly differs from the population mean.
2. **Two-Sample T-test:** This test compares the means of two independent samples. It can be further categorized into:
  - o **Two-Sample T-test with Known Variance:** Used when the variances of the two groups are known and assumed to be equal.
  - o **Two-Sample T-test with Unknown Variance:** Used when the variances are unknown and may differ between the two groups. This version is more common in practice.

## Characteristics of the T-distribution

The T-distribution resembles the normal Distributions|distribution but has fatter tails. This characteristic accounts for the increased variability expected with smaller sample sizes. As the sample size increases, the T-distribution approaches the normal distribution.

## Assumptions

For the T-test to be valid, certain assumptions must be met:

- The data should be approximately normally distributed, especially for small sample sizes.
- The samples should be independent of each other.
- For the two-sample T-test, the variances of the two groups should be equal (for the equal variance version).

## Estimation of Standard Deviation

Since the population standard deviation is unknown, the sample standard deviation is used to estimate it. This estimation is crucial for calculating the test statistic.

## Test Statistic

The test statistic for the T-test is calculated using the formula:

$$T = \frac{\bar{X} - \mu}{s/\sqrt{n}}$$

where:

- $\bar{X}$  = sample mean
- $\mu$  = population mean (or mean of the second sample in the two-sample test)
- $s$  = sample standard deviation

- $n$  = sample size

This formulation condenses all the data into a single variable, allowing for [hypothesis testing](#).

## Importance of the T-test

The T-test is a Uniformly Most Powerful Unbiased (UMPU) test, meaning it is optimal for detecting differences in means under the specified conditions.

## Tf Idf

TF-IDF is a statistical technique used in text analysis to determine the importance of a word in a document relative to a collection of documents (corpus). It balances two ideas:

- Term Frequency (TF): Captures how often a term occurs in a document.
- Inverse Document Frequency (IDF): Discounts terms that appear in many documents.

High TF-IDF scores indicate terms that are frequent in a document but rare in the corpus, making them useful for distinguishing between documents in tasks such as information retrieval, document classification, and recommendation.

TF-IDF combines local and global term [Statistics](#):

- TF gives high scores to frequent terms in a document
- IDF reduces the weight of common terms across documents
- TF-IDF identifies terms that are both frequent and distinctive

## Equations

### Term Frequency

$TF(t, d)$  measures how often a term  $t$  appears in a document  $d$ , normalized by the total number of terms in  $d$ :

$$TF(t, d) = \frac{f_{t,d}}{\sum_k f_{k,d}}$$

Where:

- $f_{t,d}$  is the raw count of term  $t$  in document  $d$
- $\sum_k f_{k,d}$  is the total number of terms in  $d$  (i.e. the document length)

### Inverse Document Frequency

IDF assigns lower weights to frequent terms:

$$IDF(t, D) = \log \left( \frac{N}{1 + |\{d \in D : t \in d\}|} \right)$$

Where:

- $N$  is the number of documents in the corpus  $D$
- $|\{d \in D : t \in d\}|$  is the number of documents containing term  $t$
- Adding 1 to the denominator avoids division by zero

## TF-IDF Score

The final score is:

$$TF-IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

## Related Notes

- Bag of words
- Tokenisation
- Clustering
- Search
- Recommender systems
- nltk

## Exploratory Ideas

- Can track TF-IDF over time (e.g., note evolution)
- Can cluster or classify the documents using TF-IDF?

## Implementations

### Python Script (scikit-learn version)

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

Step 1: Tokenize and vectorize using Bag of Words
bow = CountVectorizer(tokenizer=normalize_document)
X_counts = bow.fit_transform(corpus)

Step 2: Apply TF-IDF transformation
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X_counts)

Optional: View TF-IDF scores per document
for doc_id in range(len(corpus)):
 print(f"Document {doc_id}: {corpus[doc_id]}")
 print("TF-IDF values:")
 tfidf_vector = X_tfidf[doc_id].T.toarray()
 for term, score in zip(bow.get_feature_names_out(), tfidf_vector):
 if score > 0:
 print(f"{term.rjust(10)} : {score[0]:.4f}")
```

## Python Script (custom TF-IDF implementation)

```

import math
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.util import bigrams, trigrams

stop_words = stopwords.words('english')
tokenizer = RegexpTokenizer(r'\w+')

def tokenize(text):
 tokens = tokenizer.tokenize(text.lower())
 tokens = [t for t in tokens if len(t) > 2 and t not in stop_words]
 return tokens + [' '.join(b) for b in bigrams(tokens)] + [' '.join(t) for t in trigrams(tokens)]

def tf(term, doc_tokens):
 return doc_tokens.count(term) / len(doc_tokens)

def idf(term, docs_tokens):
 doc_count = sum(1 for doc in docs_tokens if term in doc)
 return math.log(len(docs_tokens) / (1 + doc_count))

def compute_tfidf(docs):
 docs_tokens = [tokenize(doc) for doc in docs]
 all_terms = set(term for doc in docs_tokens for term in doc)
 tfidf_scores = []
 for tokens in docs_tokens:
 tfidf = {}
 for term in all_terms:
 if term in tokens:
 tfidf[term] = tf(term, tokens) * idf(term, docs_tokens)
 tfidf_scores.append(tfidf)
 return tfidf_scores

```

## Toml

A `.toml` file is a configuration file format that stands for "Tom's Obvious, Minimal Language."

It is designed to be easy to read due to its simple syntax. TOML files are often used for configuration because they are straightforward to parse and write for both humans and machines.

The format supports basic data types like strings, integers, floats, booleans, arrays, and tables (which are similar to dictionaries or objects in other programming languages).

```

title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00Z

[database]
server = "192.168.1.1"
ports = [8001, 8001, 8002]
connection_max = 5000
enabled = true

```

In this example, you can see how different data types and structures are represented in a TOML file.

## What can you do with these files?

TOML files are primarily used for configuration purposes.

1. **Application Configuration:** Many software applications use TOML files to store configuration settings. This allows users to easily modify settings without altering the code.
2. **Project Metadata:** In some programming environments, TOML files are used to define project metadata, such as dependencies, version numbers, and other project-specific information.
3. **Data Serialization:** TOML can be used to serialize data in a format that is both human-readable and easy to parse programmatically.
4. **Environment Settings:** TOML files can be used to manage environment-specific settings, such as database connections or API keys, which can vary between development, testing, and production environments.
5. **Configuration for Build Tools:** Some build tools and package managers use TOML files to define build configurations and dependencies.

## Contents of TOML file

tool.ruff

tool.bandit

tool.uv

Pytest

## Ts\_Anomaly\_Detection

### Ts\_Anomaly\_Detection.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/TimeSeries/TS\\_Anomaly\\_Detection.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/TimeSeries/TS_Anomaly_Detection.py)

# Tableau

---

## Next Steps

- Load a [PostgreSQL](#) database and perform analytics as an example.

## Resources

- [Tableau How-To Videos](#)
- [Tutorial Link](#)
- [Example Usage Video](#)

## Features

- Can publish to blogs and embed dashboards online ([Data Visualisation](#))
- Dashboards can be shared online.
- Easier than doing visualizations in Python.

## Tags

## Technical Debt

Technical debt refers to the concept in software development where developers take shortcuts or make suboptimal decisions to speed up the delivery of a project.

These shortcuts can lead to increased complexity and potential issues in the codebase, which may require additional effort to address in the future.

Just like financial debt, technical debt incurs "interest," meaning that the longer it remains unaddressed, the more costly it becomes to fix.

## How Can Businesses Reduce Technical Debt?

### 1. Automate Testing and Code Quality Checks:

- Implement automated tests to ensure code quality and catch issues early. Tools like RUFF, mypy, and fixit can help enforce coding standards and identify potential problems.
- Use type checkers and automated checks for coding conventions to maintain consistency and reduce errors.

### 2. Track Technical Debt:

- Use dashboards to monitor and visualize technical debt. This helps in identifying areas that need attention and prioritizing them accordingly.

### 3. Code Refactoring and "Spa Days":

- Schedule regular "spa days" for the codebase, where the focus is on cleaning and refactoring specific areas. This helps in gradually reducing technical debt without impacting ongoing development.

### 4. Empower Developers:

---

## Introduction

- Allow developers to identify and address technical debt as they work on the codebase. They are often best positioned to recognize areas that need improvement.

### 5. Prioritize and Plan:

- Make technical debt reduction a part of the project planning process. Prioritize tasks that address high-impact debt and allocate time for refactoring in each development cycle.

# [Project name] Design Doc

## About this doc

*Metadata about this document. Describe the scope and current status.*

*This doc describes the technical approach, milestones, and work planned for the [Project name linked to Product Requirements Doc]*

Sign off deadline	<i>Date</i>
Status	<i>Draft</i>
Author(s)	<i>Name 1, Name 2</i>

### Sign offs

- *Name 1*
- *Name 2*
- Add your name here to sign off

## Context

*A sentence or two on the “what.” What is being built.*

*Then include the “why” (metrics we intend for). Link off to Product Requirements Doc for details.*

## Non-goals

*What is out of scope for this project that we don’t want to focus on?*

- *Out of scope detail 1*
- *Out of scope detail 2*
- ...

## Terminology

*Define any new terms that are used in the document.*

## High level approach

---

*Explain the technical approach in a few sentences so the reader understands the system flow*

## Alternatives considered

*Bullet points of alternative approaches considered and why you're not going with them.*

- Alternative 1
- Alternative 2
- ...

## Detailed design

*APIs, DB tables modified, Data models changed, and any diagrams that would help the reader understand at a high level*

## Risks

*What can go wrong with the proposed approach? How are you mitigating that?*

- Risk 1
- Risk 2
- ...

## Test plan

*How will your approach be tested? Browser testing? Manual testing? Will anything be tricky to test? Adding information here also makes it easier to make a case for a longer timeline.*

## Milestones

*How will the work be divided into chunks of progress?*

*Focus on the user milestones rather than technical ones. For example, having a minimal working feature behind a feature flag initially.*

*I usually add an additional 1 week per 2 weeks of expected feature work.*

- Milestone 1: Date 1
- Milestone 2: Date 2
- ...
- Rough ETA of project finish date: ...

- *Project retro:* ...
- 

## Rollout plan

*[optional] How will you gradually ramp up usage of the feature for safety?*

*A feature flag starting at 1% of users? Only testing in a specific region? What feature flags?*

## Open Questions

*Anything still being figured out that you could use some additional eyes or thoughts on.*

## Parties involved

*Who is working on this? Are there any external teams that need to sign off on this as well?*

- Eng 1: [Name]
- Eng 2: [Name]
- PM: [Name]
- Designer: [Name]
- [External team name]

## Appendix

*Add any detailed figures you didn't want to inline for space.*

## Telecommunications

### Network Optimization

- **Overview:** In telecommunications, RL is used to enhance network performance, optimize resource allocation, and manage traffic efficiently.
- **Applications:**
  - **Traffic Management:** RL algorithms can analyze real-time network traffic to optimize routing and minimize congestion, ensuring that data packets are transmitted through the least congested paths.
  - **Quality of Service (QoS):** RL can be used to allocate bandwidth dynamically based on current demand and service-level agreements (SLAs), improving user experience by maintaining high QoS standards.
  - **Fault Detection and Recovery:** RL systems can learn to identify and respond to network anomalies or failures, automatically rerouting traffic or reallocating resources to maintain service continuity.

### 8.2 Dynamic Resource Allocation

- **Overview:** Dynamic resource allocation in telecommunications involves adjusting network resources (like bandwidth and processing power) in real-time based on user demand and network conditions.
  - **Applications:**
-

## Introduction

- **Load Balancing:** RL can help in distributing network loads across multiple servers or paths, ensuring optimal use of available resources while preventing any single point from becoming overloaded.
  - **Adaptive Scheduling:** RL algorithms can manage the scheduling of data transmission and resource allocation in cellular networks, allowing for efficient handling of varying traffic patterns and user behaviors.
- 

# Tensorflow

Open sourced by Google Based on a dataflow graph

[Text summarization with TensorFlow](#)

Open-source library for numerical computation and large-scale [Machine Learning](#), focusing on static dataflow graphs.

Get same code use of tensorflow example:

Basic example is

[Handwritten Digit Classification](#)

[Pytorch vs Tensorflow](#)

# Terminal Commands

```
jupyter nbconvert K-Means_VideoGames_Raw.ipynb --to python --no-prompt
```

# Test Loss When Evaluating Models

Test loss is used for [Model Evaluation](#) to assess how well a model generalizes to unseen data, which is essential for evaluating its performance in real-world applications.

## Importance of Test Loss

- Test Accuracy: Indicates the percentage of correct predictions.
- **Test Loss:** Measures the magnitude of errors in predictions, providing complementary information to accuracy.
- Balancing Metrics: Depending on the application, you might prioritize [Accuracy](#) (e.g., in classification tasks) or loss (e.g., when evaluating prediction confidence or calibrating probabilistic models). Balancing both is crucial for most real-world problems.

Test loss is an [Evaluation Metrics](#) that uses the [loss function](#) to measure the model's performance on new, unseen data.

# Key Considerations

Balance Between Accuracy and Error Magnitude:

- Accuracy reflects the percentage of correct predictions but not the degree of correctness.
  - Loss can reveal situations where the model is confident but wrong, or struggling despite being correct in many cases, helping to understand prediction quality beyond simple accuracy.
-

## Introduction

### Overfitting or Underfitting Detection:

- High accuracy but high loss may indicate overfitting, where the model memorizes patterns rather than learning the underlying structure.
- Low accuracy and high loss suggest underfitting, meaning the model hasn't learned the data well enough.

### Model Calibration:

- In probabilistic models, test loss is crucial for understanding calibration.
- A model that's accurate but poorly calibrated (where predicted probabilities don't match true outcomes) will have low test accuracy but high loss.
- For example, in classification tasks, cross-entropy loss indicates how confidently and correctly the model assigns probabilities to each class.

### Hyperparameter Tuning

- During hyperparameter tuning (e.g., learning rate, batch size), configurations might yield high accuracy but poor loss (or vice versa).
- Considering both metrics provides a balanced view of performance, aiding in fine-tuning the model for both high accuracy and low error.

### Model Comparison: [Model Selection](#)

- Models with similar accuracy can have significantly different losses.
- The model with lower test loss is generally preferred as it suggests reliability in predicting probabilities, especially in critical applications like medical diagnoses or risk assessment.

### Outlier Sensitivity: [standardised/Outliers/ standardised/Outliers|Handling Outliers](#)

- Test loss can help identify model sensitivity to outliers.
- A model might achieve high accuracy but perform poorly in terms of test loss if it incorrectly classifies a few outliers.
- Conversely, a model with low test loss might be more stable in making predictions, even for edge cases.

## Testing

Testing in coding projects refers to the systematic process of evaluating software to ensure it meets specified requirements and functions correctly. It enhances software robustness, reduces maintenance costs, and improves user satisfaction.

### Testing is crucial for:

- Identifying bugs
- Ensuring code quality
- Validating that the software behaves as expected under various conditions

### Key Insights

- Testing reduces the probability of software failure,  $P(\text{failure})$ , by identifying defects before [Model Deployment|deployment](#).
- Effective testing strategies can lead to a decrease in the expected cost of errors,  $E(\text{cost})$ , associated with software bugs.

# Comprehensive Python Testing Strategy

Testing a [Python](#) program effectively involves multiple levels to ensure correctness, performance, and security. Key testing types include:

## 1. Unit Testing

- Tests individual functions and methods in isolation.
- **Tools:** pytest, unittest, doctest.
- **Best Practices:** Use meaningful test names, mock dependencies, and write focused tests.

## 2. Integration Testing

- Verifies interactions between modules and external dependencies (databases, APIs).
- **Best Practices:** Use in-memory databases, mock services, and reset the environment before tests.

## 3. Functional Testing

- Ensures the application behaves as expected from a user's perspective.
- **Best Practices:** Simulate real-world scenarios, automate UI/API tests, and validate expected outputs.

## 4. Performance Testing

- Measures execution speed, scalability, and resource usage.
- **Best Practices:** Profile bottlenecks, stress-test under load, and monitor system performance.

## 5. Security Testing [Common Security Vulnerabilities in Software Development](#)

- Identifies vulnerabilities like SQL injection, XSS, and authentication flaws.
- **Best Practices:** Validate inputs, enforce authentication, and use static analysis tools (e.g., Bandit).

## Related Topics

6. [Continuous integration](#) and deployment (CI/CD)
7. Test-driven development (TDD)
8. Software quality assurance methodologies
9. Performance testing and optimization techniques
10. [Pytest](#)
11. Unit testing
12. Integration testing
13. System testing
14. [Hypothesis testing](#)
15. Test coverage
16. [Types of Computational Bugs](#)

## Testing\_Pytest.py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Deployment/Testing\\_Pytest.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Deployment/Testing_Pytest.py)

In [ML\\_Tools](#) see: [Testing\\_Pytest.py](#)

The `pytest` example script demonstrates several key features of the [Pytest](#) testing framework:

1. **Fixtures:** The script uses a fixture named `sample_data` to provide common test data that can be reused across multiple test functions. This helps reduce code duplication and enhances test maintainability.

2. **Parametrization:** The script employs the `@pytest.mark.parametrize` decorator to run a test function with multiple sets of arguments. This allows for testing a function with various inputs without writing separate test cases for each scenario.
3. **Custom Markers:** A custom marker `@pytest.mark.slow` is used to categorize tests. This enables selective test execution based on markers, allowing you to run specific groups of tests using the `-m` option.
4. **Mocking:** The script demonstrates how to use `unittest.mock.patch` with `pytest` to replace a function with a mock. This allows for controlling the behavior of the function during the test, facilitating isolated testing of units.

## Testing\_ Unittest.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Deployment/Testing\\_unittest.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Deployment/Testing_unittest.py)

To explore testing in Python, let's focus on some key concepts and provide a simple example using the `unittest` framework, which is a built-in module for writing and running tests.

By writing and running tests, you can ensure that your code behaves as expected and catch bugs early in the development process.

[Pytest](#)

### Key Concepts in Testing

1. **Unit Testing:** Testing individual components or functions in isolation to ensure they work as expected. Unit tests are typically small and fast.
2. **Test-Driven Development (TDD):** A development approach where tests are written before the actual code. This helps define the expected behavior and ensures the code meets these expectations.
3. **Assertions:** Statements in tests that check if a condition is true. If an assertion fails, the test fails.
4. **Test Suites:** Collections of test cases that can be run together.
5. **Mocking:** Simulating the behavior of complex objects or systems to isolate the unit being tested.

### How to Run the Tests

1. Save the script to a file, e.g., `test_math_operations.py`.
2. Run the script using Python: `python test_math_operations.py`.
3. The `unittest` framework will automatically discover and run the test cases, reporting any failures or errors.

### Exploring Further

- **Test Coverage:** Measure how much of your code is covered by tests. Tools like `coverage.py` can help identify untested parts of your code.
- **Continuous Integration (CI):** Automate the running of tests using CI tools like Jenkins, Travis CI, or GitHub Actions to ensure code quality in every commit.
- **Behavior-Driven Development (BDD):** An extension of TDD that uses natural language to describe the behavior of software, often using tools like `pytest-bdd` or `behave`.
- **Edge Cases and Error Handling:**
  - The `divide` function raises a `ValueError` if division by zero is attempted.
  - The `test_divide` method includes a test case to check for this exception using `assertRaises`.

- **Mocking:**
  - The `test_mock_add` method demonstrates how to use `unittest.mock.patch` to replace the `add` function with a mock that returns a fixed value.
  - This is useful for isolating the unit under test and controlling its behavior.
- **Comprehensive Testing:**
  - Each function is tested with multiple inputs, including positive, negative, and zero values, to ensure robustness.

## Text2Cypher

Text2Cypher is a concept that allows users to convert natural language queries into Cypher queries, which are used to interact with GraphRAG|graph database like Neo4j. This functionality enables users to ask questions in a more intuitive/interpretability|interpretable, conversational manner, rather than needing to know the specific syntax of Cypher.

Allows the user to ask vague questions. Allows for multihop queries on the graph

Overall, Text2Cypher aims to simplify the interaction with graph databases, making it accessible to users who may not be familiar with query languages.

### Key Features of Text2Cypher:

1. **Natural Language Processing:** It utilizes natural language processing (NLP) techniques to understand user queries and translate them into structured Cypher queries.
2. **Flexibility:** Users can ask vague or complex questions that may not directly relate to the underlying data structure, making it easier to retrieve information from a graph database.
3. **Traversal Queries:** Text2Cypher can generate traversal queries that navigate through the graph, allowing for multi-hop queries that explore relationships between entities.
4. **Explainability:** By converting natural language into Cypher, it helps provide a clearer understanding of how the data is structured and how the queries are executed, enhancing interpretability.

## Thinking Systems

A thinking system is a point of view that helps solve a problem. Part of Knowledge Work. We view problems through the view of our own specialism (mathematics). Thinking systems help with perspective.

Types of thinking systems:

- Design
- Engineering

### Design thinking

Historically we trained people to use a product.

- users have choices,
- be user focused,
- empathy and contextual enquiry for the product/system

This puts human at the center, and focus on thinking about pain points of their experience.

Remember:

- User segments are not the same and should be handled separately.
- It is important to understand how the user interacts with the environment/other beings
- **Ludic properties** reduce design load, for example chatgpt and chat feature, or use of the iphone.

## Scientific thinking:

Strong ideas but loosely held.

Examples:

- AlphaFold and protein folding. They had lots of data and were able to derive insights from that using AI.

As problems are too complex for human minds [Scientific Method](#) now:

- We have data,
- Pick an algo ,
- Compute then gives hypothesis.

## System Thinking

We need to design for the whole system not just the individual.

I am stuck in traffic, versus you are the traffic.

Emergent behaviour with lime moulds, tokyo metro network.

Collective intelligence.

Designing system so that individuals impact the whole.

## Time Series Forecasting

With [Time Series](#) dataset we often want to predict future terms. These are methods to do so.

Resources: [TimeSeries Forecasting](#)

## Statistical Methods

- [Forecasting\\_Baseline.py](#)
- [Forecasting\\_Exponential\\_Smoothing.py](#)
- [Forecasting\\_AutoArima.py](#)

## Machine Learning Methods

[XGBoost](#)

[LightGBM](#)

## Time Series Identify Trends And Patterns

Analyze long-term trends, seasonal patterns, and cyclical behaviors.

ARIMA or SARIMA

# Time Series

Time series data is a sequence of data points collected or recorded at successive points in time, typically at uniform intervals. It captures the temporal ordering of data, which is crucial for analyzing trends, patterns, and changes over time.

Time series data is widely used across various domains, including:

- Finance: Stock prices, interest rates, and economic indicators.
- Weather Forecasting: Temperature, precipitation, and wind speed data.

In [ML\\_Tools](#) see: TimeSeries folder

- [https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/TimeSeries](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/TimeSeries)"

## What Can You Do with Time Series Data?

With time series data, you can:

- [Time Series Forecasting](#)
- [Time Series Identify Trends and Patterns](#)
- [Anomaly Detection](#)

```
We set the 'day' column as the index to facilitate time-series operations.
df.set_index('day', inplace=True)
print(df)
```

# Tokenisation

**Tokenisation** is a fundamental process in natural language processing ([NLP](#)) that involves breaking down text into smaller units called tokens. These tokens can be words, sentences, or [subwords](#), depending on the level of tokenization.

Word tokenisation

```
from nltk.tokenize import word_tokenize #keeps punctuation
text_word_tokens_nltk = word_tokenize(text_original)
print(text_word_tokens_nltk)
```

Sentence tokenisation

```
from nltk.tokenize import sent_tokenize
text_sentence_tokens_nltk = sent_tokenize(text_original)
print(text_sentence_tokens_nltk)
```

Basic tokenisation

```

temp = text_original.lower()
temp = re.sub(r"[^a-zA-Z0-9]", " ", temp) # just letters and numbers
temp = re.sub(r"\[[0-9]+\]", "", temp) #remove weird stuff
temp = word_tokenize(temp) #break up text to word list
tokens_no_stopwords = [token for token in temp if token not in stopwords.words("english")] #remove common words
print(tokens_no_stopwords)

```

## Train Dev Test Sets

In [Model Building](#) train the model using the prepared data to learn patterns and make predictions. The model is trained on your dataset, which is typically divided into three main subsets: training, development (dev), and test sets.

### Purpose of Each Set

- Training Set ([training data](#)) : Used to fit the model. This is where the model learns the patterns and relationships within the data. The majority of the data is allocated here to ensure the model has enough information to learn effectively.
- Development (Dev) Set: Also known as the [validation data](#), it is used to tune the [model parameters](#) and make decisions about model architecture. It helps in preventing overfitting by providing a separate dataset to evaluate the model's performance during training.
- Test Set: Used to evaluate the final model's performance/ [Model Evaluation](#). This set is not used during the training process and provides an unbiased evaluation of the model's ability to generalize to new, unseen data.

### Why do it this way

**Preventing Overfitting:** By monitoring performance on the validation set, practitioners can detect overfitting early and take corrective actions, such as adjusting model complexity or applying regularization techniques.

**Hyperparameter Tuning:** The validation set is crucial for tuning hyperparameters (e.g., learning rate, regularization strength) to optimize model performance.

#### Historical Suggestions

- Train-Test Sets: 70% training, 30% testing.
- Train-Dev-Test: 60% training, 20% development, 20% testing.

#### Modern Approach

- With larger [Datasets](#), a split of 98% training, 1% development, and 1% testing is often used. This is because modern models require more data to learn effectively, and larger datasets allow for smaller proportions to be allocated to dev and test sets while still maintaining sufficient data for evaluation.

#### Code Example

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.3)

```

#### Considerations

## Introduction

- Data Setup: Be careful when setting up the training and test data to ensure they are representative of the problem domain.
- Distributions: Dev and test sets should be from the same distribution to ensure consistent evaluation metrics.  
Avoid having subsets that are biased, such as data from the same geographical area.
- Handling Different Distributions: Randomly shuffle the data before splitting to ensure that each subset is representative of the whole dataset.
- Cross Validation: Consider using cross-validation techniques to make the most of your data, especially when the dataset is small.

# Transaction

Transactions are used for maintaining [Data Integrity](#) and should adhere to the [ACID Transaction](#).

## Transaction Operations

- Commit: Saves all changes made during the transaction.
- Rollback: Reverts the database to its previous state if an error occurs during the transaction.

## Concurrency and Transactions

- Concurrency: Allows multiple [queries](#) to run simultaneously, essential for high-traffic applications.
- Race Conditions: Occur when concurrent transactions access and modify shared data, potentially causing inconsistencies.

## Transaction Locks

To prevent [race conditions](#), transactions and locking mechanisms are employed to ensure that operations occur sequentially. Locks manage access to the database during [Transaction|Transactions](#):

- UNLOCK: Allows anyone to read or add data.
- SHARED: Permits reading while allowing others to access the data.
- LOCKED: Grants exclusive write access to ensure that no other transactions can interfere.

## Types of Locks

- Shared Locks: Used for read operations.
- Exclusive Locks: Used for write operations.

## Granularity of Locks

[SQLite](#) locks the entire database during exclusive transactions. While finer granularity (e.g., row-level locks) is possible in other database management systems ([Database Management System \(DBMS\)|DBMS](#)), SQLite's approach is simpler but can lead to contention in high-concurrency scenarios.

## Timestamping

Using timestamping can help manage access to exclusive locks, allowing for more efficient handling of concurrent transactions ([Concurrency](#)).

# Transfer Learning

---

Transfer learning is a technique in machine learning that leverages knowledge gained from one setting (source domain) to improve performance on a different but related setting (target domain).

The core idea is to train a model on a large dataset in the source domain, learning rich feature representations that capture general patterns and relationships in the data. These learned representations can then be transferred to the target domain, where they can be fine-tuned with a smaller dataset to achieve good performance on the target task.

Transfer learning makes sense when:

- It makes sense to do when there is lots of examples for basic layers and training, and there are few of the specialised data set.
- Task A and B have the same input x.
- You have a lot more data for Task A than Task B.
- Low level features from A could be helpful for learning B.
- When labelled data is scarce.

We can use pretrained models (i.e. from [Hugging Face](#), [Keras](#) applications, [PyTorch](#) pretrained models, model zoo).

## Examples of Transfer Learning

- **Image Recognition:** A model trained on a large dataset of labelled images (e.g., ImageNet) can learn features like edges, shapes, and textures that are useful for recognising a wide variety of objects. These features can then be transferred to a different image recognition task with a smaller dataset, such as classifying medical images or identifying different species of plants.
  - Pretraining - training on image recognition. Fine-training - retraining on radiology.
- **Natural Language Processing:** A language model trained on a massive text corpus can learn word embeddings that capture semantic relationships between words. These embeddings can then be transferred to tasks like sentiment analysis or machine translation, where they can improve performance, even with limited labelled data in the target language.

## Types of Transfer Learning

- **Unsupervised Pretraining for Supervised Learning:** The sources describe how unsupervised pretraining with models like denoising autoencoders can be used to learn useful representations that can be transferred to supervised learning tasks.
- **Cross-Domain Transfer Learning:** This involves transferring knowledge between domains with different input distributions but the same task.
- **Performance Drift:** This is a form of transfer learning where the data distribution changes gradually over time. The model needs to adapt to these changes to maintain good performance.

## Benefits of Transfer Learning

- **Improved Generalisation:** By leveraging knowledge from a larger dataset, transfer learning can help models generalise better to new data, especially when the target dataset is small.
- **Reduced Data Requirements:** Transfer learning can significantly reduce the amount of labelled data needed to train a model in the target domain.== This is particularly beneficial for tasks where labelled data is expensive or time-consuming to obtain.
- **Faster Training:** Fine-tuning a pretrained model on a smaller dataset is typically faster than training a model from scratch.

## Follow up questions

- Why might fine-tuning a pre-trained model like GPT yield better results than training from scratch
- 

## Practical Implementation

In [ML\\_Tools](#) see: [transfer\\_learning.py](#)

## Links

- [Video Overview](#)
- [Deep Learning Video](#)

# Transformed Target Regressor

## Sklearn

The `TransformedTargetRegressor` is a utility class in `scikit-learn` that applies a transformation to the target values in a regression problem. This can be useful in several scenarios:

1. **Non-normal target distribution:** Many regression algorithms assume that the target variable is normally distributed. If your target variable has a skewed distribution, applying a transformation (like a log transformation) can help improve the performance of the model.
2. **Heteroscedasticity:** This is a situation where the variance of the error terms in a regression model is not constant. In such cases, applying a transformation to the target variable can help stabilize the variance and improve the model's performance.
3. **Non-linear relationships:** If the relationship between the predictors and the target variable is not linear, a transformation can help capture the non-linearity.

The `TransformedTargetRegressor` applies the transformation before training the model and automatically applies the inverse transformation when making predictions. This makes it easier to work with transformed target variables, as you don't have to manually apply the inverse transformation every time you want to make a prediction.

# Transformer

A transformer in machine learning (ML) refers to a deep learning model architecture designed to process sequential data, such as natural language processing ([NLP](#)). It was introduced in the paper "[standardised/Attention Is All You Need](#)" and has since become a cornerstone in NLP tasks.

Transformers excel at handling sequence-based data and are particularly known for their self-attention mechanisms [Attention mechanism](#), which allow them to process long-range dependencies in data.

## Key Concepts of a Transformer

### 1. Architecture Overview:

- A transformer model consists of an encoder and a decoder, although some models use only the **encoder** (like [BERT](#) only consists of encoders) or only the **decoder** (like GPT3). Each of these components is made up of layers that include mechanisms for attention and [Feed Forward Neural Network](#).
- Encoder learns the context, decoder does the task.

### 2. Self-Attention Mechanism:

---

- The core innovation of transformers is the self-attention mechanism, which allows the model to weigh the importance of different words in a sentence relative to each other. This is crucial for understanding context and relationships in language. See [Attention mechanism].
- **Scaled Dot-Product Attention:** For each word in a sentence, the model computes attention scores with every other word. These scores are used to create a weighted representation of the input, emphasizing relevant words and de-emphasizing less relevant ones.

### 3. Multi-head attention

- Instead of having a single attention mechanism, transformers use multiple attention heads. Each head learns different aspects of the relationships between words, allowing the model to capture various linguistic features.

### 4. Positional Encoding:

- Since transformers do not inherently understand the order of words (unlike [Recurrent Neural Networks|RNNs](#)), they use positional encoding to inject information about the position of each word in the sequence.

### 5. Feed-Forward Neural Network:

- After the attention mechanism, the output is passed through a feed-forward neural network, which is applied independently to each position.

### 6. Layer Normalization and Residual Connections:

- Transformers use layer normalization and residual connections to stabilize training and help with gradient flow, making it easier to train deep networks.

### 7. Training and Applications:

- Transformers are trained on large corpora of text data using [Unsupervised Learning|unsupervised](#) or semi-supervised learning techniques. They are used for a variety of NLP tasks, including translation, summarization, and question answering.

## Additional Concepts

### • Encoder-Decoder Structure:

- The encoder processes the input sequence to build a representation, while the decoder takes this representation and generates the output sequence. This setup is particularly useful for tasks like translation.

### • Parallelization:

- Unlike Recurrent Neural Networks ([Recurrent Neural Networks](#)), transformers do **not require sequential processing**, making them more efficient, especially when training large datasets.

Follow-up questions:

- [Transformers vs RNNs](#)

## Transformers Vs Rnns

[Transformer|Transformers](#) and Recurrent Neural Networks ([Recurrent Neural Networks](#)) are both deep learning architectures **used for processing sequential data**, but they differ significantly in structure, operation, and performance.

While RNNs have been essential for sequence modeling, transformers have become the dominant architecture in ML due to their ability to handle large-scale data and long-range dependencies more efficiently.

RNNs still have use cases, especially for tasks where memory constraints are critical or for smaller datasets, but transformers are the go-to solution for most modern ML applications.

## Summary Table:

Aspect	RNNs	Transformers
<b>Architecture</b>	Sequential (step-by-step)	Parallel (process all at once)
<b>Attention</b>	Implicit through hidden states	Explicit via self-attention
<b>Parallelization</b>	Not parallelizable	Fully parallelizable
<b>Handling Long Sequences</b>	Struggles with long dependencies	Excellent with long dependencies
<b>Efficiency</b>	Slower training	Faster due to parallelization
<b>Scalability</b>	Poor scalability to long sequences	Scalable but memory-intensive
<b>Use Cases</b>	Time-series, small datasets	NLP, large datasets, vision

## 1. Architecture

- **RNNs:**
  - RNNs process data sequentially, one time step at a time. They maintain a hidden state that is updated as the model processes each token in the sequence, making them suitable for time-dependent tasks.
  - Common variants include **LSTM (Long Short-Term Memory)** and **GRU (Gated Recurrent Units)**, which are designed to capture long-term dependencies more effectively.
- **Transformer:**
  - Transformers do not process data sequentially. Instead, they process the entire sequence in parallel, allowing them to model relationships between tokens regardless of their position. This is achieved through the **self-attention mechanism**.
  - Transformers include **positional encodings** to account for the order of tokens, since their architecture doesn't have an inherent understanding of sequence order.

## 2. Processing Mechanism

- **RNNs:**
  - RNNs depend on the previous hidden state to process the next token, which means they inherently process information sequentially.
  - The hidden state is updated at each time step, which can lead to issues like **vanishing and exploding gradients problem**, especially in long sequences, making it difficult for RNNs to capture long-range dependencies.
- **Transformers:**
  - Transformers use **Attention mechanism** to allow each token to interact directly with every other token in the sequence. This allows transformers to capture long-range dependencies more effectively and efficiently.
  - The self-attention mechanism enables **parallelization** of the computation for all tokens in the sequence, which speeds up training and inference.

### 3. Parallelization and Efficiency

- **RNNs:**
  - Since RNNs must process sequences one step at a time, they **cannot be easily parallelized**. This makes them less efficient, especially for long sequences.
  - RNNs are also slower to train because of this sequential dependency.
- **Transformers:**
  - Transformers can process entire sequences at once, making it easier to parallelize computation, especially on GPUs. This leads to much faster training times compared to RNNs.
  - This parallelization is a major reason transformers have become the preferred model in large-scale tasks.

### 4. Handling Long-Term Dependencies

- **RNNs:**
  - RNNs often struggle with capturing long-term dependencies because the information must be passed through multiple time steps, which can lead to forgetting or corruption of information over long sequences.
  - **LSTMs and GRUs were developed to mitigate this problem, but they are still not as effective as transformers for capturing long-range relationships.**
- **Transformers:**
  - The self-attention mechanism in transformers allows the model to directly connect tokens from distant parts of the sequence. This makes transformers much better at modeling long-range dependencies.
  - Transformers can also model relationships across sequences regardless of their length, leading to better performance on tasks requiring a global understanding of the data.

### 5. Memory and Scalability

- **RNNs:**
  - RNNs are relatively **more memory-efficient for shorter sequences** but become inefficient for longer ones due to their sequential nature and the need to store hidden states at each time step.
  - They also scale poorly to long sequences or large datasets because of the need to compute one step at a time.
- **Transformers:**
  - Transformers, while faster, require more memory due to the computation of attention matrices, which scale quadratically with the sequence length. This can be a bottleneck for very long sequences or resource-constrained environments.
  - New transformer variants (e.g., **Longformer** or **Reformer**) have been introduced to improve memory efficiency for longer sequences.

### 6. Use Cases

- **RNNs:**
  - Traditionally used for **time-series data**, **speech recognition**, and **sequence generation** tasks.
  - They are useful when the order of data is crucial and when handling smaller datasets or shorter sequences.

### 7. Performance

- **RNNs:**

- While effective in small-scale, low-latency tasks, RNNs often perform worse than transformers on complex tasks that involve large-scale data or long-range dependencies.

- **Transformers:**

- Transformers have significantly outperformed RNNs in most tasks requiring sequential data processing, particularly in NLP. Pre-trained models like **BERT**, **GPT**, and **T5** are based on the transformer architecture and have set state-of-the-art results in many benchmarks.

Transformer-based models like **BERT** and GPT outperform traditional RNNs in NLP tasks for several key reasons:

1. **Parallelization:** Unlike [Recurrent Neural Networks|RNNs](#), which process sequences sequentially (one time step at a time), transformers can process entire sequences in parallel. This significantly speeds up training and allows for more efficient use of computational resources.
2. **Self-Attention Mechanism:** Transformers utilize a self-attention mechanism that enables them to weigh the importance of different words in a sentence relative to each other. This allows the model to capture long-range dependencies and relationships between words more effectively than RNNs, which often struggle with long-term dependencies due to their sequential nature.
3. **Handling Long Sequences:** RNNs, especially vanilla ones, can suffer from issues like vanishing and exploding gradients, making it difficult to learn from long sequences. Transformers, on the other hand, can directly connect tokens from distant parts of the sequence, making them much better at modeling long-range dependencies.
4. **Multi-Head Attention:** Transformers employ multi-head attention, which allows the model to focus on different parts of the input sequence simultaneously. Each attention head can learn different aspects of the relationships between words, enhancing the model's ability to understand context and meaning.
5. **Positional Encoding:** Since transformers do not inherently understand the order of words, they use positional encoding to inject information about the position of each word in the sequence. This allows them to maintain the sequential nature of language while still benefiting from parallel processing.
6. **Scalability and Performance:** Transformers have shown to be more scalable and perform better on large datasets, which is crucial for many NLP tasks. Pre-trained models like BERT and GPT have set state-of-the-art results in various benchmarks due to their architecture and training methodologies.

The combination of parallel processing, self-attention mechanisms, and the ability to handle long-range dependencies makes transformer-based models like BERT and GPT significantly more effective than traditional RNNs in NLP tasks.

For further reading, you can refer to the note on [Transformers vs RNNs](#) for a detailed comparison of their architectures and performance.

## Sources:

- Transformer
- [Transformers vs RNNs](#)
- [BERT](#)
- [LSTM](#)
- [Attention mechanism](#)
- [Mathematical Reasoning in Transformers](#)
- [Recurrent Neural Networks](#)
- [Transfer Learning](#)
- [Multi-head attention](#)
- [BERT Pretraining of Deep Bidirectional Transformers for Language Understanding](#)

- [LLM](#)
- [Evaluating Language Models](#)
- [Bert Pretraining](#)
- [Reasoning tokens](#)
- [NLP](#)
- [Hugging Face](#)
- [Questions](#)
- [Neural network](#)
- [Boosting](#)
- [Named Entity Recognition](#)
- [Deep Learning](#)
- [Language Model Output Optimisation](#)
- [Small Language Models](#)

## Turning A Flat File Into A Database

### Summary:

1. Read and Clean the Data: Load the data from the Excel sheet and clean it.
2. Split the Data: Separate the data into two DataFrames, one for customers and one for orders.
3. Create Tables: Create the SQLite tables with appropriate foreign key relationships.
4. Insert Data: Insert the cleaned and separated data into the respective tables.
5. Verify [Foreign Key](#): Ensure that the foreign key relationships are valid.

## Example Data Structure

### Combined Excel Data ( Sheet1 ):

order_id	order_date	customer_id	customer_name	contact_name	country	amount
1	2024-01-15	101	John Doe	Jane Doe	USA	100.50
2	2024-02-20	102	Alice Smith	Bob Smith	Canada	200.00
3	2024-03-10	101	John Doe	Jane Doe	USA	150.75
4	2024-04-05	103	Michael Brown	Sarah Brown	UK	250.00

# Steps to Process and Split Data

## Step 1: Import Libraries and Read Data

```

import pandas as pd
import sqlite3

Example data for demonstration purposes
data = {
 'order_id': [1, 2, 3, 4],
 'order_date': ['2024-01-15', '2024-02-20', '2024-03-10', '2024-04-05'],
 'customer_id': [101, 102, 101, 103],
 'customer_name': ['John Doe', 'Alice Smith', 'John Doe', 'Michael Brown'],
 'contact_name': ['Jane Doe', 'Bob Smith', 'Jane Doe', 'Sarah Brown'],
 'country': ['USA', 'Canada', 'USA', 'UK'],
 'amount': [100.50, 200.00, 150.75, 250.00]
}

Create a DataFrame from the example data
df = pd.DataFrame(data)

```

## Step 2: Clean Data

```

Example cleaning function
def clean_data(df):
 df.dropna(inplace=True)
 df.columns = [col.strip().replace(" ", "_").lower() for col in df.columns]
 return df

Clean the data
df = clean_data(df)

```

## Step 3: Split Data into Customers and Orders

```

Extract unique customers
customers_df = df[['customer_id', 'customer_name', 'contact_name', 'country']] #customer_id-customer_name-contact_name-count

Extract orders
orders_df = df[['order_id', 'order_date', 'customer_id', 'amount']] #order_id-order_date-customer_id-amount

```

## Step 4: Create Tables with Foreign Keys in SQLite

```
Connect to SQLite database (or create it)
conn = sqlite3.connect('data.db')
cursor = conn.cursor()

Enable foreign key support
cursor.execute("PRAGMA foreign_keys = ON")

Create 'customers' table
cursor.execute('''
CREATE TABLE IF NOT EXISTS customers (
 customer_id INTEGER PRIMARY KEY,
 customer_name TEXT NOT NULL,
 contact_name TEXT,
 country TEXT
)
''')

Create 'orders' table with a foreign key referencing 'customers'
cursor.execute('''
CREATE TABLE IF NOT EXISTS orders (
 order_id INTEGER PRIMARY KEY,
 order_date TEXT,
 customer_id INTEGER,
 amount REAL,
 FOREIGN KEY (customer_id) REFERENCES customers (customer_id)
)
''')

Commit changes
conn.commit()
```

## Step 5: Insert Data into Tables

```
Insert data into 'customers' table
customers_df.to_sql('customers', conn, if_exists='append', index=False)

Insert data into 'orders' table
orders_df.to_sql('orders', conn, if_exists='append', index=False)

Commit changes and close the connection
conn.commit()
conn.close()
```

## Verification: Ensure Foreign Key Relationships

```

conn = sqlite3.connect('data.db')
cursor = conn.cursor()

Query to check if all customer_id in orders table exist in customers table
cursor.execute('''
SELECT order_id
FROM orders
WHERE customer_id NOT IN (SELECT customer_id FROM customers)
''')

invalid_orders = cursor.fetchall()
conn.close()

if invalid_orders:
 print("Invalid foreign key references found:", invalid_orders)
else:
 print("All foreign key references are valid.")

```

## TypeScript

Superset of JavaScript adding static typing and object-oriented features for building large-scale applications.

## Types Of Computational Bugs

Each of these types of bugs can have significant impacts on software functionality and performance, and understanding them is crucial for effective [Debugging](#) and software development.

### Types of Computational Bugs

- Cumulative Rounding Error:** This occurs when small rounding errors accumulate over time, potentially leading to significant inaccuracies. An example is the Vancouver Stock Exchange issue.
- Cascades:** These are bugs that trigger a series of failures or errors in a system.
- Integer Overflow:** This happens when an arithmetic operation attempts to create a numeric value that is outside the range that can be represented with a given number of bits.
- Backend Issues:** These are problems that occur on the server-side of an application, affecting its functionality or performance.

### Additional Types of Computational Bugs

- Off-by-One Error:** This is a common programming error where an iterative loop iterates one time too many or one time too few. For example, iterating over an array with incorrect bounds can lead to accessing an out-of-bounds index.
- Null Pointer Dereference:** Occurs when a program attempts to access or modify data through a null pointer, leading to crashes or undefined behavior. For instance, trying to access an object method without checking if the object is null.

7. **Race Condition:** This happens when the behavior of software depends on the sequence or timing of uncontrollable events, such as threads accessing shared resources. An example is two threads modifying a shared variable simultaneously without proper synchronization.
8. **Memory Leak:** Occurs when a program fails to release memory that is no longer needed, leading to reduced performance or system crashes. This is common in languages like C++ where manual memory management is required.
9. **Buffer Overflow:** This happens when a program writes more data to a buffer than it can hold, potentially leading to data corruption or security vulnerabilities. An example is a classic stack buffer overflow attack.
10. **Logic Error:** This is a bug where the program compiles and runs but produces incorrect results due to a flaw in the algorithm or logic. For example, using the wrong formula to calculate a result.
11. **Deadlock:** Occurs when two or more processes are unable to proceed because each is waiting for the other to release resources. This is common in multithreaded applications.
12. **Syntax Error:** These are errors in the code that violate the rules of the programming language, preventing the code from compiling or running. For example, missing a semicolon in languages like Java or C++.
13. **Concurrency Issues:** These arise when multiple processes or threads execute simultaneously, leading to unpredictable results if not managed correctly. Examples include data races and inconsistent data states.
14. **Configuration Error:** Occurs when software is incorrectly configured, leading to unexpected behavior or failures. An example is a misconfigured database connection string.

## Types Of Database Schema

There are several types of database schemas commonly used in data warehousing and database design.

[Star Schema](#)

[Snowflake Schema](#)

Galaxy Schema (or Fact Constellation Schema):

- This schema consists of multiple fact tables that share dimension tables. It is useful for complex data models that require analysis across different business processes. The galaxy schema allows for more flexibility in querying and reporting.

[Normalised Schema](#)

Denormalized Schema:

- A denormalized schema combines data from multiple tables into fewer tables to improve query performance. This approach is often used in data marts and data warehouses where read performance is prioritized over write performance.

Entity-Relationship Model ([ER Diagrams](#))

- This is a conceptual schema that represents the data and its relationships in a graphical format. It is often used during the design phase of a database to visualize how entities (tables) relate to one another.

Columnar Schema:

- In a [Columnar Storage](#) database, data is stored in columns rather than rows. This schema is optimized for read-heavy operations and analytical queries, making it suitable for data warehousing applications. Examples include Apache Cassandra and Google BigQuery.

# Types Of Neural Networks

---

Types of [Neural network](#):

[Feed Forward Neural Network](#)

[Convolutional Neural Networks](#)

[Recurrent Neural Networks](#)

[Generative Adversarial Networks](#)

[Transformer](#)

## Typical Output Formats In Neural Networks

The output format of a [Neural network](#) is largely determined by the specific task it is designed to perform.

## Classification

### Binary Classification

Single Output Node: This involves a single output node with a value between 0 and 1, representing the probability of the input belonging to the positive class.

Example: A spam classifier might output a value close to 1 for a spam email and a value close to 0 for a legitimate email.

### Multiclass Classification

Multiple Output Nodes: Each class has its own output node, with values typically between 0 and 1, representing the probability of the input belonging to that class. These probabilities often sum to 1.

Example: An image classifier for different types of animals (cat, dog, bird) might output a vector like [0.2, 0.7, 0.1], indicating a 70% probability of the image being a dog.

## Regression

Single Output Node: This involves a single output node representing a continuous value.

Example: A neural network predicting house prices would output a single value representing the predicted price.

## Sequence-to-Sequence Tasks

Sequence of Outputs: The output is often represented as a list or a tensor.

Example: A neural machine translation model would output a sequence of words or subword units in the target language.

Example Applications

- **Machine Translation:** Converts a sentence from one language to another.

- **Text Summarization:** Generates a concise summary from a longer text.
  - **Speech Recognition:** Transcribes spoken language into written text.
- 

## Generative Tasks (e.g., Image Generation, Music Composition)

Data in the Same Format as the Input: The output is typically in the same format as the input data.

Example: An image generation model might output a tensor representing a generated image.

See [Generative AI](#)

## Key Considerations

[Activation Function](#): The choice of activation function in the output layer can significantly influence the output format.

Loss Functions: The [loss function](#) used during training also guides the output format. For example, binary crossentropy is commonly used for binary classification, while mean squared error is often used for regression.

## T-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) is a [Dimensionality Reduction](#) technique used primarily for visualizing high-dimensional data. Unlike methods such as [Principal Component Analysis|PCA](#) (Principal Component Analysis), which are linear, t-SNE is a **non-linear** method that excels at preserving the local structure of the data.

### Key Characteristics of t-SNE:

- **Non-linear Mapping:** It attempts to capture non-linear relationships in the data by embedding it in a lower-dimensional space (usually 2D or 3D).
- **Local Similarities:** t-SNE preserves the local structure of the data. This means that points that are close in the high-dimensional space remain close in the lower-dimensional space.
- **Global Structure:** t-SNE may distort global structures to focus more on local relationships, which is both a strength and limitation.

### How t-SNE Works:

1. **Pairwise Similarities:** t-SNE first calculates pairwise similarities between data points in the high-dimensional space.
2. **Probability Distribution:** These similarities are transformed into probabilities representing how likely it is that two points are neighbors.
3. **Lower-Dimensional Mapping:** t-SNE tries to replicate this distribution of neighbors in the lower-dimensional space by iteratively adjusting the positions of the points.

### Applications:

- **Data Visualization:** t-SNE is widely used in data visualization, especially when exploring clusters or patterns in high-dimensional datasets.
-

- **Exploratory Data Analysis (EDA):** It helps in finding clusters or subgroups in complex datasets, such as in genomics, image processing, or natural language processing.
- 

## Limitations:

- **Computationally Intensive:** t-SNE can be slow and resource-heavy, particularly on large datasets.
- **Random Initialization:** Results can vary due to its sensitivity to initialization and the perplexity parameter (which controls how t-SNE balances attention between local and global data structure).
- **Difficult to Interpret:** While t-SNE is great for visualization, interpreting the precise distances and positions of points can be tricky.

## example

```
Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

Applying t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

Plotting the results
plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')
plt.colorbar()
plt.title('t-SNE visualization of Iris dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



t-SNE visualizations of word embeddings. Left: Number Region;

## Bandit: A Security Linter for Python

### Resources

- [Bandit Documentation](#)

### How to Use Bandit

Installation To install Bandit, use pip by running the following command in your terminal:

```
pip install bandit
```

Running Bandit After installation, you can run Bandit on your Python files or directories. For example, to scan a file named `example.py`, use:

```
bandit example.py
```

This command will analyze the file and report any security issues it finds.

### 3. Customizing Bandit

You can customize Bandit's behavior by specifying options. For example, to scan a directory and exclude certain subdirectories, use:

```
bandit -r example_directory -x example_directory/venv
```

- `-r` specifies the directory to scan.
- `-x` specifies directories to exclude.

### 4. Example Script

Here's a simple Python script that Bandit can analyze:

```
import subprocess

user_input = input("Enter your name: ")
subprocess.call(["echo", user_input])
```

This script takes user input and passes it to the `echo` command using `subprocess.call()`. This can be dangerous as it might allow command injection.

To analyze the script, run:

```
bandit example.py
```

Bandit will generate a report highlighting potential security issues. For the script above, it might flag the use of `subprocess.call()` as a potential injection vector.

Fixing Issues Based on Bandit's report, you can modify your code to fix vulnerabilities. For example, to mitigate the risk of command injection, you can set `shell=False`:

```
import subprocess

user_input = input("Enter your name: ")
subprocess.call(f"echo {user_input}", shell=True)
```

Then rerun [Bandit example output](#)

## Example Script for Bandit Analysis

In [ML\\_Tools](#) see: [Bandit\\_Example\\_Nonfixed.py](#)

Features Demonstrated: [Common Security Vulnerabilities in Software Development](#)

- 1. Command Injection:** The `dangerous_subprocess` function uses `subprocess.call` with `shell=True`, which can lead to command injection if user input is not properly sanitized.
- 2. Hardcoded Password:** The `hardcoded_password` function contains a hardcoded password, which is a common security issue.
- 3. Use of eval:** The `unsafe_eval` function uses `eval`, which can execute arbitrary code if the input is not controlled.
- 4. Insecure Deserialization:** The `insecure_deserialization` function uses `pickle.loads`, which can be exploited if untrusted data is serialized.

## Running Bandit on the Example Script

To analyze this script with Bandit, save it as `example.py` and run:

```
bandit example.py
```

Bandit will generate a report highlighting the security issues in the script, providing insights into how each feature can be potentially exploited and suggesting ways to mitigate these risks.

By following these steps, you can use Bandit to identify and address security vulnerabilities in your Python code. Remember, while Bandit is a powerful tool, it's important to complement it with good coding practices and thorough security testing.

## Tool.Ruff

Ruff is a fast [Python](#) linter and code formatter.

It is designed to enforce coding style and catch potential errors in Python code.

Ruff aims to be efficient and comprehensive, supporting a wide range of linting rules and style checks. It can be used to automatically format code to adhere to a specified style guide, making it a useful tool for maintaining consistent code quality across a project.

## in TOML

have:

```
[tool.ruff.format]
Like Black, use double quotes for strings.
quote-style = "double"
Like Black, indent with spaces, rather than tabs.
indent-style = "space"
Like Black, respect magic trailing commas.
skip-magic-trailing-comma = false
Like Black, automatically detect the appropriate line ending.
line-ending = "auto"
```

## Tool.Uv

Appears in [TOML](#) file

Link: <https://github.com/astral-sh/uv>

---

`uv` is a tool for managing Python development [Virtual environments](#), projects, and dependencies. It offers a range of features that streamline various aspects of Python development, from installing Python itself to managing projects and dependencies.

- 1. Python Version Management:** `uv` allows you to install, list, find, pin, and uninstall Python versions. This is useful for managing multiple Python versions across different projects, ensuring compatibility and ease of switching between environments.
- 2. Script Execution:** You can run standalone Python scripts and manage their dependencies directly with `uv`. This simplifies the process of executing scripts with specific dependencies without needing a full project setup.
- 3. Project Management:** `uv` provides commands to create new projects, manage dependencies, sync environments, and build and publish projects. This is particularly useful for maintaining consistent environments and dependencies across development, testing, and production stages.

4. **Tool Management:** It supports running and installing tools from Python package indexes, making it easier to integrate tools like linters and formatters into your workflow.
5. **Pip Interface:** `uv` offers a pip-like interface for managing packages and environments, which can be used in legacy workflows or when more granular control is needed.
6. **Utility Commands:** It includes utility commands for managing cache, directories, and performing self-updates, which help maintain the tool's efficiency and keep it up-to-date.

## Topic Modeling

### Transfer\_Learning.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/Neural\\_Network/transfer\\_learning.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/Neural_Network/transfer_learning.py)

For deep learning, to do [Transfer Learning](#) we take out and replace a few end layers of the network. We can then train just the last layer of weights of a neural network.

The number of layers to remove and then added from pretrained depends on the similarity between tasks. Higher layers in networks are able to recognise higher detail components.

# U

---

## Table of Contents

- UML
- Ubuntu
- Unsupervised Learning
- Untitled 1
- Untitled 2
- Untitled
- Use Cases for a Simple Neural Network Like
- Use of RNNs in energy sector
- Utilities
- unittest
- univariate vs multivariate
- unstructured data

## Uml

<https://www.drawio.com/>

[https://www.reddit.com/r/SoftwareEngineering/comments/133iw7n/is\\_there\\_any\\_free\\_handy\\_tool\\_to\\_create\\_uml/](https://www.reddit.com/r/SoftwareEngineering/comments/133iw7n/is_there_any_free_handy_tool_to_create_uml/)

<https://plantuml.com/>

## Ubuntu

Ubuntu is a popular open-source operating system based on the [Linux](#) kernel. It is designed to be user-friendly:

1. **Desktop Environment:** Ubuntu provides a graphical user interface (GUI) that makes it accessible to users who may not be familiar with command-line interfaces. It is often used as a desktop operating system for personal computers.
2. **Server Use:** Ubuntu Server is a version of Ubuntu designed for server environments. It is commonly used for hosting websites, applications, and databases due to its stability and security.
3. **Development:** Many developers use Ubuntu for software development because it supports a wide range of programming languages and development tools. It is also compatible with various software libraries and frameworks.
4. **Education:** Ubuntu is often used in educational institutions for teaching computer science and programming due to its open-source nature and the availability of free software.
5. **Customization:** Being open-source, Ubuntu allows users to customize their operating system according to their needs. Users can modify the source code, install different desktop environments, and choose from a variety of applications.
6. **Community Support:** Ubuntu has a large community of users and developers who contribute to its development and provide support through forums, documentation, and tutorials.

# Unsupervised Learning

---

Unsupervised learning is a type of machine learning where the algorithm is trained on data without explicit labels or predefined outputs.

Unsupervised learning involves discovering hidden patterns in data without predefined labels. It is valuable for exploratory data analysis, [Clustering](#), and [Isolated Forest](#).

The goal is to find hidden patterns, relationships, or structures in the data. Unlike supervised learning, which uses labeled input-output pairs, unsupervised learning relies solely on input data, allowing the algorithm to uncover insights independently.

## Key Concepts

1. No Labeled Data: There is no ground truth or correct output associated with the input data.
2. Data Patterns: The algorithm identifies inherent structures, clusters, or associations within the dataset.
3. Objective: The primary objective is to explore the data and organize it to reveal underlying patterns.

## Common Types of Unsupervised Learning

### Clustering

Description: The algorithm groups similar data points together based on their features.

Example: Customer segmentation in marketing, where a clustering algorithm divides customers into groups based on purchasing behavior, demographics, or browsing history.

Popular Algorithms:

- [K-means](#): Divides the data into ( k ) clusters, where each data point belongs to the nearest cluster.
- Hierarchical Clustering
- [DBScan](#)
- [Support Vector Machines](#)
- K-nearest neighbours

### Dimensionality Reduction

Description: Reduces the number of input variables (features) while preserving as much information as possible.

This is helpful for high-dimensional data, where visualization and analysis become challenging.

Popular Algorithms:

- [Principal Component Analysis](#)

### Isolated Forest

Description: Identifies [standardised/Outliers](#) or unusual data points that don't conform to the expected pattern in the dataset.

Example: Detecting fraudulent credit card transactions by identifying transactions that deviate significantly from typical spending patterns.

Mechanism: Works by randomly partitioning the data and identifying [standardised/Outliers|anomalies](#) as points that can be isolated quickly.

## Untitled 1

---

## Untitled 2

## Untitled

# Use Cases For A Simple Neural Network Like

Scenarios where a simple [Neural network](#)|[Neural Network](#) work like this might be useful:

**Regression with Multiple Features** If you have multiple input features and you want to predict a continuous output, this network can learn the appropriate weights for each feature. For instance:

- Predicting **fuel efficiency** of a car based on features like engine size, horsepower, and weight.
- Predicting **sales** based on multiple factors like marketing spend, seasonality, and economic indicators.

**Binary Classification** With slight modification (e.g., adding a **Sigmoid activation** to the output layer), you could use this network for binary classification tasks. For example:

- Classifying whether an email is **spam** or not based on features like word frequency and sender information.

**Multi-Feature Time Series Forecasting** If you have time series data with multiple variables, you can feed it into this simple network to predict future values based on past trends. For instance:

- Predicting **stock prices** based on multiple features like historical prices, trading volume, and economic data.
- Training and Optimization (Next Steps)** The provided code only defines the network and performs a **forward pass**, but to use this model for real-world tasks, you would need to:
- **Define a loss function** (e.g., Mean Squared Error for regression or Cross-Entropy Loss for classification).
  - **Train the network** using an optimizer like **Stochastic Gradient Descent (SGD)**, **Adam**, or another optimization algorithm.
  - **Backpropagate** the gradients to update the model's weights using gradient descent.

# Use Of Rnns In Energy Sector

For energy data problems, many **interpretable machine learning algorithms** can be applied in place of or alongside RNNs. These models offer transparency, making it easier to understand the relationships between features and predictions, which is critical in areas like energy management, where interpretability can be as important as accuracy.

For each of the energy data questions that RNNs might solve, **interpretable alternatives** [Machine Learning Algorithms](#): such as **linear regression**, **decision trees**, **random forests**, and **ARIMA** models can be employed. These models provide **transparency** by revealing which features (e.g., weather, demand) influence predictions the most, making them suitable for stakeholders who need clear explanations of the decisions made by the model.

## Demand forecasting

- **Algorithms:**

- **Linear Regression:** Can model simple linear relationships between energy consumption and time (e.g., daily/seasonal trends).
- **Decision Trees:** Provides clear if-then rules for predicting future energy usage based on historical consumption, time of day, and other factors.
- **Random Forests:** An ensemble of decision trees that provides better accuracy than individual trees while still being interpretable using feature importance.
- **Gradient Boosting (GBM):** Can be used with feature importance or SHapley Additive exPlanations|SHAP values to understand which factors (e.g., time, weather) drive energy demand.
- **Why:** These models allow for clear interpretation of how factors like temperature, time of day, and previous energy use contribute to predictions.

## 2. Renewable Energy Generation Prediction

- **Algorithms:**
  - **Linear Regression:** For simple relationships, like the effect of sunlight hours or wind speed on energy generation.
  - **Support Vector Machines (SVM):** Can create interpretable linear boundaries when predicting renewable energy outputs, with clear separation of factors (e.g., wind speed thresholds).
  - **Random Forests:** Offers feature importance metrics that explain which weather factors are most important for predicting energy generation.
  - **GBM:** Using SHapley Additive exPlanations|SHAP values or feature importance to interpret the impact of weather variables on the energy output.
- **Why:** These algorithms can provide insights into the key weather conditions driving renewable energy generation and give transparent predictions for decision-making.

## 3. Energy Price Forecasting

- **Algorithms:**
  - **ARIMA (AutoRegressive Integrated Moving Average):** A traditional time series forecasting method that models linear relationships in energy prices over time.
  - **Linear Regression:** Can model the impact of factors like demand, supply, and historical prices in an interpretable way.
  - **Decision Trees:** Easy to interpret and can show thresholds where prices change based on inputs like demand or fuel costs.
  - **XGBoost:** Provides interpretability through SHAP values or feature importance, explaining which market factors (e.g., demand, fuel prices) drive price changes.
- **Why:** These algorithms offer interpretable insights into what drives price fluctuations, making them useful for energy market analysis and trading.

## 4. Anomaly Detection in Energy Consumption

- **Algorithms:**
  - **Isolation Forests:** Specifically designed for anomaly detection and provides interpretable results by isolating outliers.
  - **k-Nearest Neighbors (k-NN):** Can flag anomalies by comparing new consumption data to known normal consumption patterns, with simple explanations of "closeness" to typical patterns.
  - **Logistic Regression:** Can be used to classify energy consumption data into "normal" and "anomalous" categories based on clear feature contributions.
  - **One-Class SVM:** A linear model that can classify whether energy usage deviates from typical patterns.

- **Why:** These interpretable algorithms can identify unusual patterns in energy data, providing clear reasons (e.g., thresholds exceeded) for flagging certain periods as anomalous.
- 

## 5. Load Balancing and Optimization

- **Algorithms:**
  - **Linear Programming (Optimization):** Provides interpretable rules for how energy should be distributed across the grid to minimize costs and prevent overloads.
  - **Decision Trees:** Can clearly show the impact of different factors (e.g., region, time of day) on grid load, and thresholds for balancing loads.
  - **Rule-Based Systems:** Set explicit rules for load balancing based on historical data and real-time demand, offering full transparency.
- **Why:** These interpretable models can assist grid operators in understanding which regions or time periods contribute most to load imbalances and suggest corrective actions.

## 6. Customer Energy Usage Profiling

- **Algorithms:**
  - **k-Means Clustering:** Can group customers into distinct profiles based on energy usage patterns, with each cluster representing a clear profile (e.g., high-energy consumers, off-peak users).
  - **Decision Trees:** Can predict customer profiles based on historical usage data and explain which features (e.g., time of usage, appliance usage) define each profile.
  - **Logistic Regression:** Can be used to classify customers into different segments based on usage characteristics, providing clear coefficient-based interpretations.
- **Why:** These models provide transparency into what factors drive a customer's energy usage profile, which is essential for creating personalized recommendations.

## 7. Demand Response Optimization

- **Algorithms:**
  - **Linear Programming (Optimization):** Provides interpretable solutions for when and where to implement demand response programs to minimize peak energy use.
  - **Decision Trees:** Can clearly define rules for when demand response should be triggered based on time of day, weather, and current load.
  - **k-Nearest Neighbors (k-NN):** Can identify similar past scenarios where demand response was implemented successfully and explain why the current situation matches.
- **Why:** These methods give clear, interpretable guidelines for when and how to reduce energy demand during peak times, based on past patterns.

## 8. Fault Detection in Power Systems

- **Algorithms:**
    - **Decision Trees:** Can explain why certain operational conditions (e.g., voltage drops, temperature increases) are likely to lead to faults, with clear rules and thresholds.
    - **Random Forests:** Provides feature importance scores that highlight which factors (e.g., temperature, load) are most indicative of impending faults.
    - **Logistic Regression:** Offers simple, interpretable probabilities for whether a fault will occur, based on key factors like current and voltage.
-

- **Why:** Fault detection requires clear, interpretable models that help engineers understand the most important factors leading to equipment failures.
- 

## 9. Energy Usage Forecasting for Smart Buildings

- **Algorithms:**
  - **Multiple Linear Regression:** Can model the relationship between building factors (e.g., temperature, occupancy) and energy usage, offering clear coefficients.
  - **Decision Trees:** Provides an interpretable way to understand which building features (e.g., time of day, external temperature) influence energy consumption the most.
  - **k-Means Clustering:** Can group similar time periods or usage patterns to explain different operational modes of the building.
- **Why:** These algorithms provide interpretable insights into how building features and external factors impact energy consumption, allowing for more efficient energy management.

## 10. Time Series Forecasting for Energy Production in Microgrids

- **Algorithms:**
  - **ARIMA:** Traditional interpretable time series model that predicts future production based on past production data.
  - **Linear Regression:** Can predict energy production based on simple factors like weather data, fuel availability, and historical output.
  - **Decision Trees:** Helps identify which weather or resource factors are most critical for predicting energy production at a given time.
- **Why:** Time series models like ARIMA are highly interpretable and useful for understanding how different factors contribute to energy production in microgrids.

## 11. Battery Storage Optimization

- **Algorithms:**
  - **Linear Programming (Optimization):** Provides a clear, interpretable approach to optimizing charge/discharge schedules based on forecasted energy generation and consumption.
  - **Decision Trees:** Can explain when and why batteries should be charged or discharged based on energy production, consumption, and cost factors.
  - **Rule-Based Systems:** Establish clear rules for battery storage optimization, offering fully interpretable decision-making processes.
- **Why:** Optimizing battery storage requires clear, rule-based or linear models to understand how different variables (e.g., energy prices, consumption) impact storage decisions.

# Utilities

## Unittest

### `@patch (from unittest.mock ) Explanation`

`@patch` is used to replace objects/functions with mock versions during tests. It is part of Python's `unittest.mock` module.

---

## Example & Usage

python

Copy code

```
from unittest.mock import patch def fetch_data(): """Simulated function that fetches data from an API"""\n return "Real Data"\n\n@patch("__main__.fetch_data", return_value="Mocked Data") def test_fetch_data(mock_fetch):\n assert fetch_data() == "Mocked Data"
```

### ◆ How it works:

- `@patch("__main__.fetch_data", return_value="Mocked Data")` replaces `fetch_data()` with a mocked version returning `"Mocked Data"`.
- Inside the test, `fetch_data()` will always return `"Mocked Data"` instead of calling the real function.

## Why use `@patch` ?

- Prevents tests from making actual API/database calls.
- Speeds up testing by mocking expensive operations.
- Allows control over return values and side effects.

## Your Case:

- `@pytest.fixture` is used to provide reusable test data (`mock_files`).
- `@patch` is used to:
  - Mock file operations (`builtins.open`, `os.walk`).
  - Mock function calls (`process_file`, `log_action`, `write_updated_file`).
  - Prevent real file modifications while testing.

## Univariate Vs Multivariate

Single feature versus multiple features

## Unstructured Data

[!Important] Unstructured data is data that does not conform to a data model and has no easily identifiable structure.

Unstructured data cannot be easily used by programs, and is difficult to analyze. Examples of unstructured data could be the contents of an **email**, **contents of a word document**, **data from social media**, **photos**, **videos**, **survey results**, etc.

## An example of unstructured data

An simple example of unstructured data is a string that contains interesting information inside of it, but that has not been formatted into a well defined schema. An example is given below:

	<b>UnstructuredString</b>
Record 1	"Bob is 29"
Record 2	"Mary just turned 30"

## Unstructured vs structured data

In contrast with unstructured data, [structured data](#) refers to data that has been formatted into a well-defined schema. An example would be data that is stored with precisely defined columns in a relational database or excel spreadsheet. Examples of structured fields could be age, name, phone number, credit card numbers or address. Storing data in a structured format allows it to be easily understood and queried by machines and with tools such as [SQL](#).

**V**

## Table of Contents

- [Vacuum](#)
- [Variance](#)
- [Vector Database](#)
- [Vector Embedding](#)
- [Vector\\_EMBEDDING.py](#)
- [Vectorisation](#)
- [Vectorized Engine](#)
- [Vercel](#)
- [View Use Case](#)
- [Views](#)
- [Violin plot](#)
- [Virtual environments](#)
- [vanishing and exploding gradients problem](#)

## Vacuum

## Variance

Variance in a dataset is a statistical measure that represents the degree of spread or dispersion of the data points around the mean (average) of the dataset.

It quantifies how much the individual data points differ from the mean value.

A higher variance indicates that the data points are more spread out from the mean, while a lower variance indicates that they are closer to the mean.

Variance is calculated as the average of the squared differences between each data point and the mean.

See also: [Boxplot Distributions](#)

### Variance:

- Measures how much a single variable deviates from its mean.
- For variable  $X$ , variance is:

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)^2$$

- Variance determines the **spread** of data along a particular dimension.

## Overview

Vector databases are specialized systems designed to handle and manage [Vector Embedding](#).

As most real-world data is unstructured, such as text, images, and audio, vector databases play a role in organizing and querying this data effectively.

Why use them:

- 
- data is [unstructured data](#) i.e image
  -

## Key Features

- **Vector Embeddings:** At the core, vector databases store embeddings generated by machine learning models. These embeddings transform complex data into fixed-size vectors that encapsulate semantic information.
- **Similarity Search:** By leveraging the geometric properties of vector spaces, vector databases can quickly identify similar items. This is achieved by measuring distances (e.g., cosine similarity, Euclidean distance) between vectors.
- **Indexing Methods:** Various indexing techniques, such as HNSW (Hierarchical Navigable Small World) graphs, IVF (Inverted File), and PQ (Product Quantization), are employed to optimize search speed and accuracy. Allows faster searching.

## Querying Vectors

To query vectors, users typically specify a target vector and a similarity metric.

The database then retrieves vectors that are closest to the target, based on the chosen metric. This process is crucial for applications like recommendation systems, where finding similar items is essential.

## Use Cases

1. **Long-term Memory for LLM:** Vector databases can store vast amounts of contextual information, enhancing the memory and retrieval capabilities of large language models (LLMs). Implemented using [Langchain](#).
2. Rank and Recommendation system using nearest neighbours.
3. **Semantic Search:** Unlike traditional keyword-based search, semantic search understands the context and meaning, providing more relevant results. This is particularly useful in natural language processing (NLP) applications.
4. **Similarity Search:** Beyond text, vector databases support similarity searches for multimedia data, enabling applications in image recognition, audio analysis, and video retrieval.

## Options

Several vector database solutions are available, each with unique features and optimizations:

- **Pincone:** Known for its scalability and ease of integration with machine learning workflows.
- **Weaviate:** Offers a semantic graph database with built-in vector search capabilities.
- **Chroma:** Focuses on simplicity and performance for embedding-based applications.
- **Redis:** Provides vector search capabilities through its modules, suitable for real-time applications.
- **Qdrant:** Designed for high-performance vector search with a focus on scalability.
- **Milvus:** An open-source solution optimized for handling large-scale vector data.
- **Vespa:** Combines vector search with traditional search capabilities, ideal for complex applications.

## Related Concepts

- **standardised/Vector Embedding:** The process of converting data into vector form, capturing its semantic essence.
- **standardised/Vector Embedding:** A specific type of vector embedding used in NLP to represent words in a continuous vector space.
- **Semantic Relationships:** A search technique that leverages the meaning and context of queries and data to deliver more relevant results.
- **Cosine Similarity**

## Resources

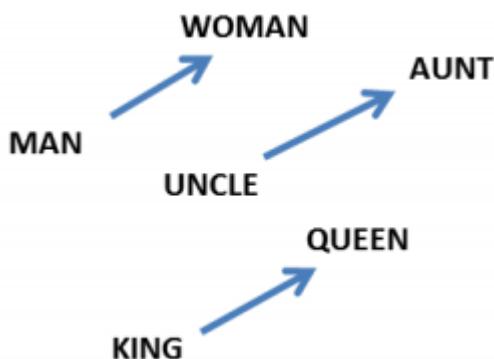
[Vector Databases simply explained! \(Embeddings & Indexes\)](#)

# Vector Embedding

Vector Embedding is a technique used in machine learning and [NLP](#) to represent data in a continuous vector space. This representation captures the [Semantic Relationships](#) of data, such as words or sentences, allowing similar items to be positioned close to each other in the vector space.

## Key Concepts

- Data Compression: Embeddings compress data into a lower-dimensional space, making it easier to process and analyze. This is particularly useful for high-dimensional data like text or images.
- Semantic Similarity: In the embedding space, similar items are positioned close to each other. This proximity reflects semantic similarity, meaning that items with similar meanings or characteristics have similar vector representations.
- [Dimensionality Reduction](#): Words are represented in a lower-dimensional space compared to traditional methods like one-hot encoding, resulting in more efficient computations.
- [Semantic Relationships](#): Words with similar meanings or contexts are located close to each other in the vector space. For example, "king" and "queen" might be closer to each other than "king" and "apple."



1. Contextual Understanding: (Vector) Word embeddings capture the context in which words appear, allowing models to understand nuances and relationships in language.

Popular methods for generating vector (word) embeddings include:

- [Word2vec](#),
- [GloVe](#),

- FastText.
- spaCy

## Types of Similarity Measures

- Euclidean Distance
- Cosine Similarity

## Applications

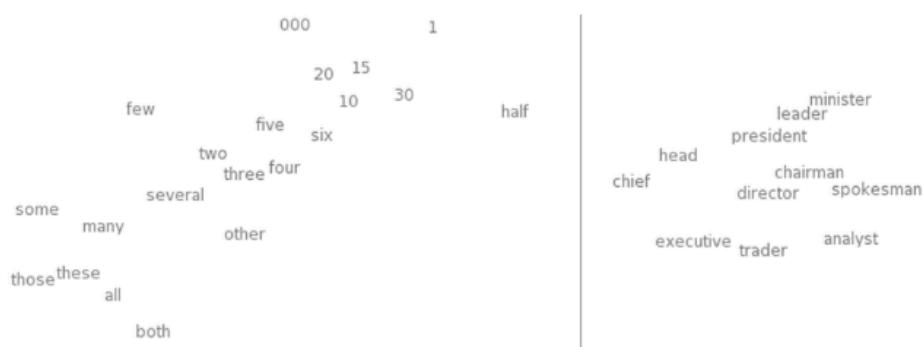
- [Language Models](#): Vector embeddings are widely used in language models to represent words, phrases, or sentences, enabling models to understand and generate human language more effectively.
- [Attention mechanism](#): Embeddings are often used with attention mechanisms to enhance model performance in tasks like translation, summarization, and question answering.

## Example Use Cases

- Word Embeddings: Techniques like Word2Vec and GloVe create word embeddings that capture semantic relationships between words, enabling tasks like word similarity and analogy solving.
- Sentence Embeddings: Models like [BERT](#) and Sentence Transformers generate embeddings for entire sentences, facilitating tasks like sentiment analysis and semantic search.

## Visualizations

- [t-SNE](#): A technique for visualizing high-dimensional data, often used to display word embeddings in a two-dimensional space.



t-SNE visualizations of word embeddings. Left: Number Region;

## Implementation

How to do vector embeddings in [PyTorch](#) that show [Semantic Relationships](#) between terms.

In [ML\\_Tools](#) see: [Vector\\_EMBEDDING.py](#)

## Related Terms

[How to search within a graph](#) [How would you decide between using TF-IDF and Word2Vec for text vectorization](#)  
[embeddings for OOV words](#)

# Vector\_EMBEDDING.PY

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/NLP/Vector\\_EMBEDDING.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/NLP/Vector_EMBEDDING.py)

## Explanation of the Script

### 1. Vocabulary and Embedding Layer:

- o Terms are mapped to indices using a dictionary.
- o The embedding layer learns continuous vector representations for these terms.

### 2. Cosine Similarity:

- o The cosine similarity function measures how similar two terms are in the embedding space. Higher values indicate closer relationships.

### 3. Visualization:

- o Embeddings are plotted in a 2D space to show semantic relationships. Terms with similar meanings (e.g., "king" and "queen") are expected to cluster together.

### 4. t-SNE for Dimensionality Reduction:

- o If the embedding dimension is higher than 2, t-SNE can reduce it to 2D for visualization while preserving semantic relationships.

## Outputs

### 1. Cosine Similarities:

- o Pairwise similarity scores between terms to quantify their semantic closeness.

### 2. Visualization:

- o A scatter plot showing the positions of terms in the embedding space.

# Vectorisation

[Link](#)

 Pasted image 20241217204829.png|500 Numpy dot is better than for loop and summing. Why does it run faster? A: Designed to run in parallel

Sequentially versus simultaneously in parallel.

Related concepts:

- [Numpy](#)
- [gpu](#)

# Vectorized Engine

A modern database query execution engine designed to optimize data processing by leveraging vectorized operations and SIMD (Single Instruction, Multiple Data) capabilities of modern CPUs. Vectorized engines, such as [DuckDB](#), process data in large blocks or batches using SIMD instructions, allowing for improved parallelism, cache locality, and reduced overhead compared to traditional row-at-a-time processing engines, using [Columnar Storage](#).

# Vercel

---

## View Use Case

### Scenario

A company wants to generate monthly performance reports for its employees. The performance data is spread across multiple tables, including `employees`, `departments`, and `performance_reviews`. Instead of writing complex queries every time a report is needed, the company can create a view that simplifies data retrieval.

### Step 1: Define the Tables

Assume we have the following tables:

- **employees**: Contains employee details.
  - `employee_id`
  - `name`
  - `department_id`
- **departments**: Contains department details.
  - `department_id`
  - `department_name`
- **performance\_reviews**: Contains performance review data.
  - `review_id`
  - `employee_id`
  - `review_score`
  - `review_date`

### Step 2: Create a View

To simplify the reporting process, we create a view that joins these tables and aggregates the performance scores:

```

CREATE VIEW employee_performance AS
SELECT
 e.employee_id,
 e.name,
 d.department_name,
 AVG(pr.review_score) AS average_score
FROM
 employees e
JOIN
 departments d ON e.department_id = d.department_id
JOIN
 performance_reviews pr ON e.employee_id = pr.employee_id
GROUP BY
 e.employee_id, e.name, d.department_name;

```

## Step 3: Query the View

Now, whenever the HR department needs to generate a performance report, they can simply query the `employee_performance` view:

```
SELECT * FROM employee_performance WHERE average_score >= 4.0;
```

This query retrieves all employees with an average performance score of 4.0 or higher, making it easy to identify top performers.

## Benefits of Using Views in This Use Case

1. **Simplification:** The view encapsulates complex joins and aggregations, allowing HR to retrieve performance data without needing to understand the underlying table structure.
2. **Reusability:** The view can be reused for different reports, such as quarterly reviews or department-specific performance assessments.
3. **Maintainability:** If the logic for calculating performance scores changes, the HR team only needs to update the view definition, not every individual query.
4. **Data Consistency:** All reports generated from the view will be consistent, as they rely on the same underlying logic for calculating average scores.
5. **Security:** If sensitive employee data needs to be protected, the view can be designed to exclude certain columns, ensuring that only necessary information is accessible.

# Views

Views are virtual tables defined by SQL [Querying|Query](#) that **simplify complex data representation**. They can remove unnecessary columns, aggregate results, partition data, and secure sensitive information.

In [DE\\_Tools](#) see: [https://github.com/rhyslwells/DE\\_Tools/blob/main/Explorations/SQLite/Viewing/Viewing.ipynb](https://github.com/rhyslwells/DE_Tools/blob/main/Explorations/SQLite/Viewing/Viewing.ipynb)

Basic Usage:

- Simplification
- Aggregation (using GROUP)
- [Common Table Expression](#)
- Securing data: can give all values in a field the same value.

Advanced Usage

- Temporary Views: Exist only for the **duration of the database connection**.
- [Common Table Expression](#): Serve as temporary views for a single query.
- [Soft Deletion](#): Use views and triggers to mark records as deleted without physically removing them from the table.

Related topics:

- [View Use Case](#)

# Why Use Views?

## 1. Simplification and Abstraction:

- Views encapsulate complex queries, allowing users to interact with data without needing to understand the underlying structure. This simplifies data retrieval by hiding complexity.

## 2. Security:

- Views restrict access to specific data by granting users access to views instead of underlying tables, which can help protect sensitive information. Note: Access controls may vary by database system (e.g., not available in [SQLite](#)).

## 3. Reusability and Maintainability:

- Define complex queries once in a view and reuse them across multiple applications, simplifying maintenance when logic changes.

## 4. Data Consistency and Integrity:

- Ensure consistent data presentation across applications and users by encapsulating business logic for uniform calculations.

## 5. Performance Optimization:

- While regular views do not inherently improve performance, materialized views can enhance performance by storing precomputed results.

## 6. Logical Data Independence:

- Provide a layer of abstraction between physical data storage and access methods, allowing [Database Schema|schema](#) changes without affecting view users.

## 7. Aggregation and Partitioning:

- Views can be used to calculate and store aggregated results (e.g., average ratings) and organize data by specific criteria (e.g., years or categories).

# Violin Plot

An extension of a [Boxplot](#) showing the data distribution. Useful when comparing distributions, skewness.

```
data = [...] # Your data
sns.violinplot(data=data, color="purple", fill="lightblue", scale="area")
plt.show()
```

# Virtual Environments

## [Setting up virtual env](#)

For windows (need to not be in a venv before del)

```
rmdir /s /q venv
python -m venv venv
venv\Scripts\activate
```

---

pip freeze > requirements.txt

Introduction

.gitignore [https://www.youtube.com/watch?v=\\_vejzukmn4s](https://www.youtube.com/watch?v=_vejzukmn4s)

Remember to set python interpreter

Related terms:

- Poetry

## Vanishing And Exploding Gradients Problem

[Recurrent Neural Networks|RNN](#)

**vanishing and exploding gradients problem** In standard RNNs, the difficulty lies in retaining useful information over long sequences due to the exponential decrease in the gradient values, which results in poor learning of long-term dependencies.

**W**

## Table of Contents

- WCSS and elbow method
- Weak Learners
- Web Feature Server (WFS)
- Web Map Tile Service (WMTS)
- Webpages relevant
- What algorithms or models are used within the energy sector
- What algorithms or models are used within the telecommunication sector
- What are the best practices for evaluating the effectiveness of different prompts
- What can ABM solve within the energy sector
- What is the difference between odds and probability
- What is the role of gradient-based optimization in training deep learning models.
- When and why not to use regularisation
- Why JSON is Better than Pickle for Untrusted Data
- Why Type 1 and Type 2 matter
- Why and when is feature scaling necessary
- Why does increasing the number of models in an ensemble not necessarily improve the accuracy
- Why does label encoding give different predictions from one-hot encoding
- Why does the Adam Optimizer converge
- Why is named entity recognition (NER) a challenging task
- Why is the Central Limit Theorem important when working with small sample sizes
- Why use ER diagrams
- Wikipedia\_API.py
- Windows Subsystem for Linux
- Word2Vec.py
- Word2vec
- WordNet
- Wrapper Methods

## Wcss And Elbow Method

USE: WCSS (within-cluster sum of squares)

WCSS is a measure developed within the ANOVA framework. It gives a very good idea about the different distance between different clusters and within clusters, thus providing us a rule for deciding the appropriate number of clusters.

The plot will resemble an "elbow," and the goal is to find the point where the decrease in WCSS slows down, forming an elbow-like shape.

Elbow numbers are the point where the rate of decrease in WCSS starts to flatten out

The rationale behind the elbow method is that

Rationale: as you increase the number of clusters (K), the WCSS will generally decrease because each cluster becomes smaller. However, there is a point where the addition of more clusters provides diminishing returns in terms of reducing WCSS. The elbow point represents a good balance between capturing the variance in the data and avoiding excessive fragmentation.

# Code

---

```

Use WCSS and elbow method

number of clusters
wcss=[]

start=2
end=10

Create all possible cluster solutions with a loop
for i in range(start,end):
 # Cluster solution with i clusters
 kmeans = KMeans(i)
 # Fit the data
 kmeans.fit(df_scaled)
 # Find WCSS for the current iteration
 wcss_iter = kmeans.inertia_
 # Append the value to the WCSS list
 wcss.append(wcss_iter)

Create a variable containing the numbers from 1 to 6, so we can use it as X axis of the future plot

number_clusters = range(start,end)

Plot the number of clusters vs WCSS

plt.plot(number_clusters,wcss)

Name your graph

plt.title('The Elbow Method')
Name the x-axis
plt.xlabel('Number of clusters')
Name the y-axis
plt.ylabel('Within-cluster Sum of Squares')

Identify the elbow numbers (there may be more than one thats best)
elbow_nums=[4,5,6,7,8]

```

**plotting**

```
function to give scatter for each elbow number

def scatter_elbow(X, elbow_num, var1, var2):
 """
 Apply clustering with elbow method and plot a scatter plot with cluster information.

 Parameters:
 - X: DataFrame, input data for clustering
 - elbow_num: int, number of clusters determined by elbow method
 - var1, var2: str, names of the variables for the scatter plot

 Returns:
 None (plots the scatter plot)
 """

 # Apply [clustering](#clustering) with elbow number
 kmeans = KMeans(elbow_num)
 kmeans.fit(X)

 # Add cluster information
 identified_clusters = kmeans.fit_predict(X)
 X['Cluster'] = identified_clusters

 # Plot
 plt.scatter(X[var1], X[var2], c=X['Cluster'], cmap='rainbow')
 plt.xlabel(var1)
 plt.ylabel(var2)
 plt.title(f"{elbow_num}-Clustering for {var1}-{var2}")
 plt.show()

Example usage:
scatter_elbow(data, elbow_num, 'var1', 'var2')
for elbow_num in elbow_nums:
 scatter_elbow(df, elbow_num, var1, var2)
```

## Weak Learners

Weak learners are simple models that perform slightly better than random guessing. They are often used as the building blocks in [Model Ensemble](#) methods to create a strong predictive model.

## Characteristics

- **Simplicity:** Weak learners are typically simple models, such as [Decision Tree](#) stumps, which split the data based on a single feature.
- **Performance:** Individually, they may not perform well, but when combined, they can produce a powerful ensemble model.

## Role in Model Ensembling

Weak learners are a crucial component of [Model Ensemble](#) techniques, such as boosting and bagging, where multiple weak learners are combined to improve overall model performance.

## Learning Rate

- The [learning rate](#) is a [Hyperparameter](#) that controls the contribution of each weak learner to the final ensemble model.
- A smaller learning rate means that each weak learner has a smaller impact, often requiring more learners to achieve good performance.

## Web Feature Server (Wfs)

[GIS](#)

### Web Feature Server (WFS)

**Purpose:** WFS is designed to serve raw geographic features (vector data) over the web.

**Functionality:**

- **Feature-Based:** It delivers geographic features (such as points, lines, and polygons) and their associated attribute data in formats like GML (Geography Markup Language).
- **Interactivity:** Allows clients to query and retrieve specific features, perform spatial and attribute queries, and even support transactions (e.g., inserting, updating, deleting features).
- **Data Access:** Provides access to the actual data behind the map, enabling more detailed and customized analysis and processing compared to image-based services.
- **Standardization:** Also standardized by the OGC, ensuring compatibility and interoperability across various GIS applications and systems.

## Web Map Tile Service (Wmts)

[GIS](#)

### Web Map Tile Service (WMTS)

**Purpose:** WMTS is designed to serve pre-rendered, cached image tiles of maps.

**Functionality:**

- **Tile-Based:** It serves map images as small, fixed-size tiles, usually in a format such as PNG or JPEG.
- **Performance:** By using cached tiles, WMTS can quickly deliver map images, making it highly efficient for applications requiring fast map rendering, like web mapping applications.
- **Scalability:** The tile-based approach allows for easy scaling and efficient handling of high load, as the same tiles can be reused for multiple requests.
- **Standardization:** It is standardized by the Open Geospatial Consortium (OGC), ensuring interoperability between different systems and software.

## Webpages Relevant

---

Using bookmarks:

### Time Series

[https://aeturrell.com/blog/posts/time-series-explosion/?utm\\_source=substack&utm\\_medium=email](https://aeturrell.com/blog/posts/time-series-explosion/?utm_source=substack&utm_medium=email)

[#](https://otexts.com/fpp3/?utm_source=substack&utm_medium=email)

## What Algorithms Or Models Are Used Within The Energy Sector

## What Algorithms Or Models Are Used Within The Telecommunication Sector

## What Are The Best Practices For Evaluating The Effectiveness Of Different Prompts

## What Can Abm Solve Within The Energy Sector

[Agent-Based Modelling](#)

energy systems analysis

## What Is The Difference Between Odds And Probability

## What Is The Role Of Gradient Based Optimization In Training Deep Learning Models.

## When And Why Not To Us Regularisation

While regularization is tool to combat overfitting, it is not always useful. It is crucial to consider the model's

- complexity,
- the quality and
- quantity of data,
- and the appropriateness of the regularization parameters

to ensure effective performance on validation data. If your model is performing well on [training data](#) but poorly on [validation data](#), regularization might not always solve this issue for several reasons:

1. **Underfitting:** While regularization aims to reduce overfitting, it can also lead to underfitting if the penalty is too strong. This occurs when the model becomes too simplistic and fails to capture the underlying patterns in the data, resulting in poor performance on both training and validation datasets.
2. **Model Complexity:** Regularization primarily addresses the complexity of the model. If the model architecture itself is not suitable for the task (e.g., too simple or inappropriate for the data distribution), regularization won't help improve performance. The model may still struggle to learn the necessary features, leading to poor validation performance.
3. **Insufficient Data:** If the training dataset is small or not representative of the validation dataset, regularization may not compensate for the lack of data. The model might learn noise or irrelevant patterns from the training data, which regularization cannot correct.
4. **Improper Regularization Parameter ( $\lambda$ ):** The effectiveness of regularization depends on the choice of the regularization parameter  $\lambda$ . If  $\lambda$  is set too high, it can overly penalize the model's parameters, leading to underfitting. Conversely, if it's too low, it may not sufficiently reduce overfitting.
5. **Feature Interaction:** Regularization techniques like  $L_1$  and  $L_2$  may not effectively capture complex interactions between features. If the relationships in the data are intricate, regularization alone may not improve the model's ability to generalize.
6. **Validation Set Issues:** The validation set itself may not be representative of the problem space, or it may contain noise or outliers that affect the model's performance. Regularization won't address these issues if the validation data is flawed.

## Why Json Is Better Than Pickle For Untrusted Data

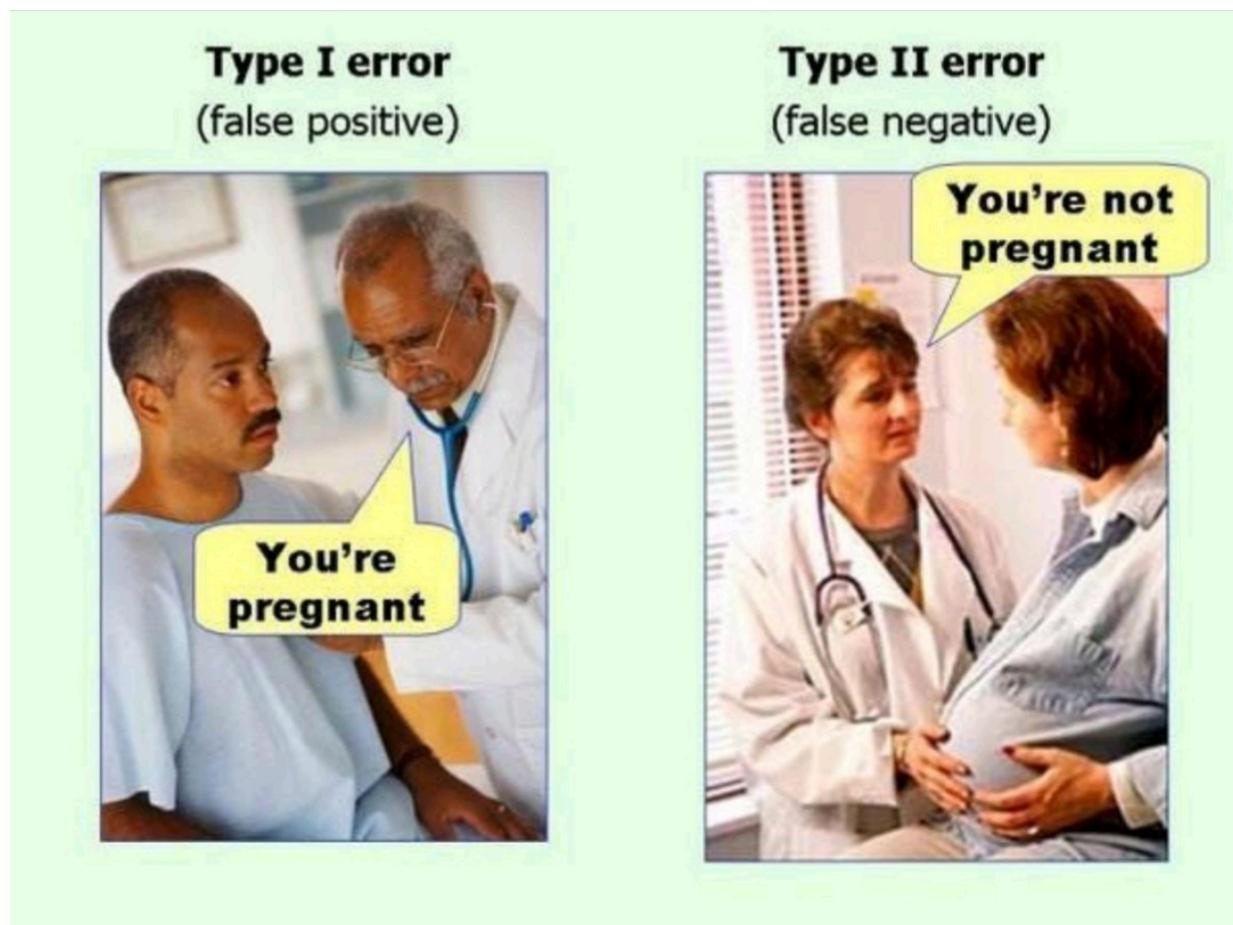
### JSON vs. Pickle:

1. **Security:**
  - o **JSON:** JSON is a text-based data format that is inherently safer for handling untrusted data. It **only** supports basic data types like strings, numbers, arrays, and objects, which reduces the risk of executing arbitrary code.
  - o **Pickle:** Pickle is a Python-specific binary serialization format that **can serialize and deserialize complex Python objects**. However, it can **execute arbitrary code** during deserialization, making it unsafe for handling untrusted data.
2. **Interoperability (the ability of computer systems or software to exchange and make use of information):**
  - o **JSON:** JSON is language-agnostic and widely used across different programming environments, making it ideal for data interchange between systems.
  - o **Pickle:** Pickle is specific to Python, which limits its use in cross-language applications.
3. **Readability:**
  - o **JSON:** Being a text format, JSON is human-readable and easy to debug.
  - o **Pickle:** Pickle produces binary data, which is not human-readable and harder to debug.

For these reasons, JSON is preferred over Pickle when dealing with untrusted data, as it minimizes security risks and offers better interoperability and readability.

# Why Type 1 And Type 2 Matter

Type I and Type II errors are used in evaluating the performance of classification models, and understanding their differences is essential for interpreting model results effectively.



## Type I Error (False Positive)

- **Definition:** A Type I error occurs when the model incorrectly predicts the positive class. In other words, it identifies a negative instance as positive.
- **Example:** If a model predicts that an email is spam (positive) when it is actually not spam (negative), this is a Type I error.
- **Consequences:** Type I errors can lead to unnecessary actions or consequences, such as misclassifying legitimate emails as spam, which may result in important messages being missed.

## Type II Error (False Negative)

- **Definition:** A Type II error occurs when the model incorrectly predicts the negative class. This means it fails to identify a positive instance.
- **Example:** If a model predicts that an email is not spam (negative) when it is actually spam (positive), this is a Type II error.
- **Consequences:** Type II errors can lead to missed opportunities or risks, such as allowing spam emails to clutter the inbox or failing to detect a disease in a medical diagnosis scenario.

## Why Both Errors Matter

1. **Impact on Decision-Making:** The consequences of Type I and Type II errors can vary significantly depending on the context. In some applications, such as medical diagnoses, a Type II error (failing to detect a disease) may be more critical than a Type I error (false alarm). Conversely, in fraud detection, a Type I error may lead to unnecessary investigations.
2. **Balancing Precision and Recall:** Understanding these errors helps in balancing precision (the proportion of true positives among all positive predictions) and recall (the proportion of true positives among all actual positives). Depending on the application, one may be prioritized over the other, influencing model tuning and evaluation.
3. **Model Evaluation:** Both types of errors are essential for a comprehensive evaluation of a model's performance. Metrics such as precision, recall, and the F1 score incorporate these errors to provide a more nuanced view of how well the model is performing.
4. **Risk Management:** By analyzing the trade-offs between Type I and Type II errors, practitioners can make informed decisions about model thresholds and operational strategies, ensuring that the model aligns with business or clinical objectives.

## Why And When Is Feature Scaling Necessary

Feature Scaling is useful for models that use distances like [Support Vector Machines|SVM](#) and [K-means](#)

### When Scaling Is Unnecessary

1. **Tree-based Algorithms:**
  - Algorithms like [Decision Tree](#), [Random Forests](#), and Gradient Boosted Trees are invariant to feature scaling because they split data based on thresholds, not distances.
  - Example: Splits are determined by feature values, not their magnitude.
2. **Data with Uniform Scales:**
  - If all features have the same range or are already normalized (e.g., percentages), scaling may not be required.

## Why Does Increasing The Number Of Models In A Ensemble Not Necessarily Improve The Accuracy

Increasing the number of models in an ensemble ([Model Ensemble](#)) does not always lead to improved accuracy due to several limiting factors:

- **Convergence of Predictions:** Additional models may lead to similar predictions, resulting in minimal changes to the overall output.
- **Limited Data Representation:** If the dataset is noisy or incomplete, more models will only aggregate existing noise without capturing new patterns.
- **Diminishing Returns:** Each new model contributes less unique information, and performance is ultimately limited by the irreducible error in the data.
- **Increased Complexity:** More models increase computational costs and training times without necessarily improving accuracy.

- **Overfitting Risk:** Adding complex models can lead to overfitting, where the ensemble learns noise instead of underlying patterns.
- 

# Why Does Label Encoding Give Different Predictions From One Hot Encoding

Label Encoding and One-Hot Encoding give different predictions because they represent categorical variables in fundamentally different ways.

- **Label Encoding** might cause issues by implying an ordinal relationship between categories, leading to biased predictions.
- **One-Hot Encoding** prevents this by treating categories independently, resulting in more accurate predictions when there's no natural order among the categories.

## Label Encoding:

- **How It Works:** Label Encoding assigns an integer value to each unique category in a feature. For example, if you have three towns: `['West Windsor', 'Robbinsville', 'Princeton']`, Label Encoding would convert them into numerical values like this:
  - West Windsor → 0
  - Robbinsville → 1
  - Princeton → 2
- **Interpretation in the Model:** When you use Label Encoding, the model interprets the numbers as continuous values, meaning it sees a numeric relationship between them (i.e., "Princeton" might be considered numerically higher than "West Windsor" and closer to "Robbinsville"). This can cause issues if the numeric values don't have any ordinal relationship.

## One-Hot Encoding:

- **How It Works:** One-Hot Encoding creates a separate binary (0 or 1) column for each unique category. For example, the three towns would be represented as:
  - West Windsor → [1, 0, 0]
  - Robbinsville → [0, 1, 0]
  - Princeton → [0, 0, 1]
- **Interpretation in the Model:** One-Hot Encoding treats each category as a separate binary feature and does not impose any ordinal relationship between them. This means the model doesn't assume that one category is greater or lesser than another. Each category is treated independently.

## Key Differences in Predictions:

### 1. Ordinal vs. Non-Ordinal Data Representation:

- With **Label Encoding**, the model might treat "Robbinsville" (encoded as 1) as closer to "West Windsor" (encoded as 0) than "Princeton" (encoded as 2), even though these categories don't have any inherent numerical relationship. This can lead the model to incorrectly infer relationships based on these numeric values.
- With **One-Hot Encoding**, no such relationship is assumed. Each category is represented as a vector of 0s and 1s, and the model treats them as distinct entities, preventing any assumptions about their order.

### 2. Model Interpretation:

- **Label Encoding** introduces an implicit ordinal relationship (e.g.,  $0 < 1 < 2$ ) that can influence the model, especially for linear models like Linear Regression, which assumes that the input features are on a similar scale. This may lead to inappropriate relationships in the regression model.
- **One-Hot Encoding** avoids this issue by using binary columns for each category, effectively preventing the model from assuming an ordinal relationship between the categories.

### 3. Feature Space:

- **Label Encoding** results in a single feature column for the categorical variable.
- **One-Hot Encoding** expands the feature space, creating as many columns as there are categories. In the case of a categorical feature with many unique values, this can significantly increase the dimensionality of the model.

## Why Predictions Differ:

- In Label Encoding, a linear regression model might learn that "Robbinsville" is numerically closer to "West Windsor" than "Princeton," and this might distort the predictions.
- In One-Hot Encoding, the model treats each category independently, leading to different relationships being learned (if any) between the categories and the target variable.

## Example:

Let's assume you are predicting house prices, and you're using a linear regression model where the `town` feature is the only predictor (along with some other features like `area` ).

- **With Label Encoding:**
  - The model will interpret the encoded numeric values (0 for West Windsor, 1 for Robbinsville, and 2 for Princeton) and might incorrectly assume a relationship such as: "Princeton" (2) is somehow numerically "higher" than "West Windsor" (0), which doesn't reflect any meaningful relationship.
  - This can lead to biased coefficients and, therefore, inaccurate predictions.
- **With One-Hot Encoding:**
  - The model will learn the effect of each category (West Windsor, Robbinsville, and Princeton) as a separate feature, with no assumption of ordinality.
  - This often results in more accurate predictions, especially when categorical features have no inherent order.

## Why the Adam Optimizer Converges

The Adam optimizer is able to efficiently handle sparse gradients and adaptively adjust learning rates. The convergence of Adam, often observed as a flattening of the cost function, can be attributed to several factors inherent to its design and the characteristics of the dataset being used.

The convergence of the Adam optimizer, resulting in a stable cost value, is a product of its adaptive learning rate, regularization effects, numerical stability mechanisms, and the dataset's characteristics.

## 1. Convergence to Local Minimum or Saddle Point

**Adaptive Learning Rate:** Adam adjusts the learning rate for each parameter individually, allowing it to quickly converge towards a local minimum or saddle point. This adaptability helps in navigating complex cost landscapes but may also cause the optimizer to plateau at a local minimum rather than reaching the global minimum. This plateauing effect can result in a stable final cost value, such as 0.146.

**Gradient Behaviour:** As Adam converges, the gradients often become small or near zero, slowing down the learning process and leading to a flattened cost curve.

---

## 2. Learning Rate

**Impact of Learning Rate ( $\alpha$ ):** The choice of learning rate significantly influences convergence speed. A larger learning rate might cause overshooting, while a smaller one might lead to slow convergence. Common values like 0.001, 0.005, 0.01, and 0.1 are used to balance these effects.

**Stability in Suboptimal Regions:** If the learning rate is not optimal, Adam might get stuck in a suboptimal region, such as a local minimum or saddle point, resulting in a stable cost value.

## 3. Regularization

**L2 Regularization:** The inclusion of L2 regularization helps prevent overfitting but also affects the optimization process by slightly increasing the final cost value. The observed cost value is a balance between error reduction and the regularization penalty.

## 4. Numerical Stability

**Moment Estimates:** Adam uses first and second moment estimates (mean of gradients and squared gradients) to update parameters. As these estimates improve, the magnitude of updates decreases, leading to smaller changes in the cost function and eventual flattening.

**Epsilon for Stability:** The epsilon parameter ensures numerical stability by preventing division by very small values, which can also lead to reduced update steps when squared gradients are small.

## 5. Dataset Characteristics

**Simplicity of the Dataset:** The characteristics of the dataset, such as its simplicity or complexity, can influence convergence. In a simple dataset with few features, the optimizer might reach a plateau quickly due to the limited complexity of the problem.

## 6. Final Cost Comparison

**Reasonable Solution:** The stable cost value, such as 0.146, indicates that Adam has found a reasonable solution given the dataset and optimizer settings. It reflects a balance between minimizing error and applying regularization.

# Why Is Named Entity Recognition (NER) A Challenging Task

Named Entity Recognition (NER) is considered a challenging task for several reasons:

1. **Ambiguity:** Entities can be ambiguous, meaning the same word or phrase can refer to different entities depending on the context. For example, "Washington" could refer to a city, a state, or a person. Disambiguating these entities requires a deep understanding of context.
2. **Variability in Language:** Natural language is highly variable and can include slang, idioms, and different syntactic structures. This variability makes it difficult for NER models to consistently identify entities across different texts.

- 
- 3. **Named Entity Diversity:** Entities can take many forms, including names, organizations, locations, dates, and more. Each type may have different characteristics, requiring the model to adapt to various patterns.
  - 4. **Lack of Annotated Data:** High-quality annotated datasets are crucial for training NER models. However, creating such datasets can be time-consuming and expensive, leading to limited training data for certain domains or languages.
  - 5. **Multilingual Challenges:** NER systems often struggle with multilingual texts, where the same entity may be represented differently in different languages. This adds complexity to the recognition process.
  - 6. **Nested Entities:** In some cases, entities can be nested within each other (e.g., "The University of California, Berkeley"). Recognizing such nested structures can be particularly challenging for NER systems.
  - 7. **Domain-Specific Language:** Different domains (e.g., medical, legal, technical) may have specific terminologies and entities that general NER models may not recognize effectively without domain-specific training.

## Why Is The Central Limit Theorem Important When Working With Small Sample Sizes

The [Central Limit Theorem](#) (CLT) is particularly important for data scientists working with small sample sizes. It enables the use of various statistical methods, and helps in making valid inferences about the population from limited data.

- 1. **Assumption of Normality:** The CLT states that the sampling [Distributions|distribution](#) of the sample means will approximate a normal distribution, regardless of the underlying population distribution, as long as the sample size is sufficiently large.
- 2.
- 3. This is crucial for data scientists because many statistical methods and tests (such as t-tests, ANOVA, and regression analysis) rely on the [assumption of normality](#). Even with small sample sizes, the CLT provides a foundation for making inferences about the population.
- 4. **Confidence Intervals and Hypothesis Testing:** The CLT enables data scientists to construct confidence intervals and perform hypothesis tests even when the sample size is small. By using the sample mean and the standard error (which is derived from the sample size), data scientists can estimate the range within which the true population mean is likely to fall, and test hypotheses about population parameters.
- 5. **Reduction of Variability:** The variance of the sampling distribution decreases as the sample size increases, which means that larger samples provide more reliable estimates of the population mean. For small sample sizes, the CLT helps data scientists understand the potential variability in their estimates and make more informed decisions based on their data.
- 6. **Practical Application:** In many real-world scenarios, obtaining large samples may not be feasible due to time, cost, or logistical constraints. The CLT allows data scientists to work with smaller samples while still applying statistical techniques that assume normality, thus broadening the scope of analysis.
- 7. **Robustness of Results:** The CLT provides a theoretical justification for the robustness of statistical methods. Even if the original data is not normally distributed, the means of sufficiently large samples will tend to be normally distributed, allowing for more reliable conclusions.

# Why Use Er Diagrams

---

## Why use ER diagrams

Cleaning a dataset before creating an [ER Diagrams](#) is crucial for ensuring accuracy and reliability in your database design

1. [Data Quality](#): Cleaning the dataset helps identify and rectify errors, inconsistencies, and missing values. This ensures that the data accurately represents the real-world entities and relationships you intend to model.
2. [Normalised Schema](#): Before creating an ER diagram, it's essential to normalize the data, which involves organizing it efficiently to reduce redundancy and dependency. Cleaning the dataset beforehand allows you to identify redundant information and eliminate it, leading to a more streamlined ER diagram.
3. Entity Identification: Through data cleaning, you can properly identify the entities within your dataset. This involves determining which attributes belong to which entity, as well as identifying any composite or derived attributes. Proper entity identification is fundamental to creating an accurate ER diagram.
4. Relationship Clarity: Cleaning the dataset helps clarify the relationships between entities. By ensuring that the data accurately reflects the relationships between different entities, you can create a more precise ER diagram that accurately represents the connections between various elements.
5. Data Consistency: [Data Cleansing](#) ensures consistency across the dataset, which is essential for maintaining integrity in the ER diagram. Consistent data allows for clearer identification of relationships and attributes, leading to a more effective database design.

# Wikipedia\_Api.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Utilities/Wikipedia\\_API.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Utilities/Wikipedia_API.py)

# Windows Subsystem For Linux

[Windows Subsystem for Linux](#) (WSL) is a compatibility layer for running Linux binary executables natively on Windows 10 and Windows 11. It allows users to run a Linux environment directly on Windows without the need for a virtual machine or dual-boot setup.

Key features of WSL include:

1. **Integration with Windows**: Users can access files from both Windows and the [Linux](#) environment seamlessly.
2. **Multiple Distributions**: WSL supports various Linux distributions, such as [Ubuntu](#), Debian, and Fedora, which can be installed from the Microsoft Store.
3. **Command-Line Tools**: Users can run Linux command-line tools and applications directly in Windows, making it easier for developers to work in a familiar environment.
4. **Performance**: WSL provides near-native performance for Linux applications, making it suitable for development and testing.

# Word2Vec.Py

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Build/NLP/Word2Vec.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Build/NLP/Word2Vec.py)

---

The script can benefit from **Word2Vec embeddings** by replacing the randomly initialized embeddings with pretrained or trained embeddings generated using Word2Vec. These embeddings provide a meaningful semantic structure that is learned from a corpus of text, enhancing the visualization and [cosine similarity](#) calculations.

## Benefits:

1. **Meaningful Relationships:** Words like "king" and "queen" will naturally be closer than "king" and "apple."
2. **Analogy Solving:** Word2Vec supports vector arithmetic to solve word analogies (e.g., "man is to king as woman is to queen").
3. **Improved Visualizations:** The embeddings reflect real-world semantic and syntactic relationships, making the 2D plots more interpretable.

## Further Enhancements

1. **Train Your Word2Vec:**
  - o Train embeddings on a custom corpus using `gensim.models.Word2Vec(#word2vec)` to reflect domain-specific semantics.
2. **Hybrid Embeddings:**
  - o Combine Word2Vec with other models (e.g., [BERT](#) or Sentence [Transformer|Transformers](#)) for tasks requiring contextual understanding.

**Using** `glove-wiki-gigaword-100` :

- A GloVe model with 100-dimensional embeddings trained on the Wikipedia Gigaword dataset.
- Approximate size: ~100MB.

## Expected Outcome

1. **Visualization:**
  - o Terms from the same category (e.g., royalty, fruits, animals) will cluster together in the t-SNE plot.
2. **Cosine Similarity:**
  - o Similar terms (e.g., "king" and "queen" or "apple" and "orange") will have higher cosine similarity scores.
3. **Semantic Diversity:**
  - o The expanded list increases the diversity of semantic relationships and highlights the strength of embeddings in grouping similar concepts.

## Word2Vec

Word2Vec is a technique for generating vector representations of words. Developed by researchers at Google, it uses a shallow [neural network](#) to produce [standardised/Vector Embedding|word embedding](#) that capture [Semantic Relationships](#) and [syntactic relationships](#). Word2Vec has two main architectures:

In [ML\\_Tools](#) see: [Word2Vec.py](#)

1. [CBOW \(Continuous Bag of Words\):](#)
  - o Predicts a target word given its context (neighboring words).
  - o Efficient for smaller datasets.
2. [Skip-Gram:](#)
  - o Predicts the context words given a target word.
  - o Performs better on larger datasets.

Word2Vec generates dense, continuous vector representations where words with similar meanings are close to each other in the embedding space. For example:

- `vector("king") - vector("man") + vector("woman") ≈ vector("queen")`

## Wordnet

## Wrapper Methods

Used in [Feature Selection](#). Wrapper methods are powerful because they directly optimize the performance of the machine learning model by selecting the most informative subset of features.

1. **Iterative Approach:** Unlike [Filter method](#), which assess the relevance of features based on statistical properties, wrapper methods **directly involve the machine learning algorithm in the feature selection process.**
2. **Subset Selection:** Wrapper methods work by creating different subsets of features from the original dataset and training a model on each subset. These subsets can be combinations of different features or a subset of all features.
3. **Model Evaluation:** After training a model on each subset of features, the performance of each model is evaluated using a performance metric, such as accuracy, precision, recall, or F1-score, depending on the problem type (classification or regression).
4. **Optimization Criterion:** The goal of wrapper methods is to find the subset of features that maximizes the performance of the machine learning model. This can be achieved by selecting the subset that yields the highest performance metric on a validation set or through cross-validation.
5. **Computational Intensity:** Wrapper methods are computationally intensive because they involve training multiple models for each possible combination of features. As a result, they can be slower and require more computational resources compared to filter methods.

## Examples of Wrapper Methods:

- **Forward Selection:** Starts with an empty set of features and iteratively adds one feature at a time, selecting the feature that improves model performance the most.
- **Backward Elimination:** Begins with all features and iteratively removes one feature at a time, selecting the feature whose removal improves model performance the least.
- **Recursive Feature Elimination (RFE):** Iteratively removes features from the full feature set based on their importance, as determined by a specified machine learning algorithm.
- **Selection Criteria:** The choice of performance metric and optimization criterion depends on the specific machine learning task and dataset characteristics. It's essential to select a metric that aligns with the goals of the project and to validate the selected subset of features on unseen data.

# X

---

## Table of Contents

- XGBoost

## Xgboost

XGBoost (eXtreme Gradient Boosting) is a highly efficient and flexible implementation of [Gradient Boosting](#) that is widely used for its accuracy and performance in machine learning tasks.

### How does XGBoost work

It works by building an [Model Ensemble](#) - ensemble of decision trees, where each tree is trained to correct the errors made by the previous ones. Here's a breakdown of how XGBoost works:

### Key Concepts

1. Gradient Boosting Framework:
    - XGBoost is based on the gradient boosting framework, which builds models sequentially. Each new model aims to reduce the errors (residuals) of the combined ensemble of previous models.
  2. Decision Trees:
    - XGBoost typically uses decision trees as the base learners. These trees are added one at a time, and existing trees in the model are not changed.
  3. Objective Function:
    - The objective function in XGBoost consists of two parts: the loss function and a regularization term.
    - [Loss function](#): Measures how well the model fits the training data. For regression, this might be mean squared error; for classification, it could be logistic loss.
    - [Regularisation](#): Helps prevent overfitting by penalizing complex models. XGBoost supports both L1 (Lasso) and L2 (Ridge) regularization.
  4. Additive Training:
    - XGBoost adds trees to the model sequentially. Each tree is trained to minimize the loss function, taking into account the errors made by the previous trees.
  5. [Gradient Descent](#)
    - The model uses gradient descent to minimize the loss function. It calculates the gradient of the loss function with respect to the model's predictions and uses this information to update the model.
  6. [learning rate](#) ( $\eta$ ):
    - A parameter that scales the contribution of each tree. A smaller learning rate requires more trees but can lead to better performance.
  7. Tree Pruning:
    - XGBoost uses a technique called "max depth" to control the complexity of the trees. It also employs a "max delta step" to ensure that the updates are not too aggressive.
  8. [Handling Missing Data](#)
    - XGBoost can handle missing data internally by learning the best direction to take when a value is missing.
-

## 9. Parallel and Distributed Computing:

- XGBoost is designed to be highly efficient and can leverage parallel and distributed computing to speed up training.

### Key Features:

- Tree Splitting: Builds [Decision Tree](#) in a level-wise manner, leading to balanced trees and efficient computation.
- Parameters: Key parameters include `eta` (learning rate) and `max_depth` (maximum depth of a tree), which control the model's complexity and learning process.

## Workflow

### 1. Initialization:

- Start with an initial prediction, often the mean of the target values for regression or a uniform probability for classification.

### 2. Iterative Training:

- For each iteration, compute the gradient of the loss function with respect to the current predictions.
- Fit a new decision tree to the negative gradient (residuals).
- Update the model by adding the new tree, scaled by the learning rate.

### 3. Model Output:

- The final model is a weighted sum of all the trees, where each tree contributes to the final prediction.

### Advantages:

- Accuracy: Known for its high accuracy and robustness across various machine learning tasks.
- [Regularisation](#): Supports L1 (Lasso) and L2 (Ridge) regularization to prevent overfitting.
- Flexibility: Offers a wide range of hyperparameters for fine-tuning models.

### Use Cases:

- Structured Data: Particularly effective for structured data and tabular datasets.
- [Interpretability](#): Suitable when model interpretability is important.
- [Hyperparameter Tuning](#): Ideal for scenarios where extensive hyperparameter tuning is feasible.

## Implementing XGBoost in Python

### Step 2: Import Necessary Libraries

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

### Step 3: Prepare Your Data

Split your dataset into training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## Step 4: Convert Data to DMatrix

Convert the data into DMatrix, the optimized data structure used by XGBoost:

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

## Step 5: Set Parameters

Define the parameters for the XGBoost model:

```
params = {
 'max_depth': 6,
 'eta': 0.1,
 'objective': 'binary:logistic', # Use 'reg:squarederror' for regression tasks
 'eval_metric': 'logloss'
}
```

## Step 6: Train the Model

Train the XGBoost model using the training data:

```
num_rounds = 100
bst = xgb.train(params, dtrain, num_rounds)
```

## Step 7: Make Predictions and Evaluate

Make predictions on the test set and evaluate the model's performance:

```
y_pred = bst.predict(dtest)
y_pred_binary = [1 if y > 0.5 else 0 for y in y_pred]
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy:.2f}")
```

## Notes

Set up an example of XGBoost. Plot the parameter space slices "Min\_Samples\_Split", "Max\_Depth" vs accuracy.

```
xgb_model = XGBClassifier(n_estimators = 500, learning_rate = 0.1, verbosity = 1, random_state = RANDOM_STATE)
xgb_model.fit(X_train_fit, y_train_fit, eval_set = [(X_train_eval, y_train_eval)], early_stopping_rounds = 10)
xgb_model.best_iteration
```

# Y

---

## Table of Contents

- [yaml](#)

## Yaml

Stands for [YAML ain't markup language](#) and is a superset of JSON

- lists begin with a hyphen
- dependent on whitespace / indentation
- better suited for configuration than [Json](#)

YAML is a data serialization language often used to write configuration files. Depending on whom you ask, YAML stands for yet another markup language, or YAML isn't markup language (a recursive acronym), which emphasizes that YAML is for data, not documents.

# Z

---

## Table of Contents

- Z-Normalisation
- Z-Score
- Z-Scores vs Prediction Intervals
- Z-Test

## Z Normalisation

[https://github.com/rhyslwells/ML\\_Tools/blob/main/Explorations/Preprocess/Outliers/outliers\\_z\\_score.py](https://github.com/rhyslwells/ML_Tools/blob/main/Explorations/Preprocess/Outliers/outliers_z_score.py)

Z-normalisation, also known as z-score normalization, is a technique used to standardize the range of independent variables or features of data.

This process is used in preparing data for [machine learning algorithms](#), especially those that rely on distance calculations, such as k-nearest neighbors and [gradient descent](#) optimization.

### Why Normalize?

- Consistency Across Features: By normalizing, the peak-to-peak range of each column is reduced from a factor of thousands to a factor of 2-3. This ensures that each feature contributes equally to the distance calculations, preventing features with larger ranges from dominating the results.
- Centered Data: The range of the normalized data (x-axis) is centered around zero and roughly +/- 2. This centering is beneficial for algorithms that assume data is normally distributed around zero.
- Improved Learning Rates: Normalization allows for a larger [learning rate](#) in [Gradient Descent](#), which can speed up convergence and improve the efficiency of the learning process.

### Z-Score Normalization

Z-score normalization transforms the data so that each feature has:

- A mean of 0
- A standard deviation of 1

To implement z-score normalization, adjust your input values using the formula:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j} \text{ Where:}$$

- $x^{(i)}_j$  is the value of the feature  $j$  for the  $i$ -th example.
- $\mu_j$  is the mean of all the values for feature  $j$ .
- $\sigma_j$  is the standard deviation of feature  $j$ .

The mean and standard deviation are calculated as follows:

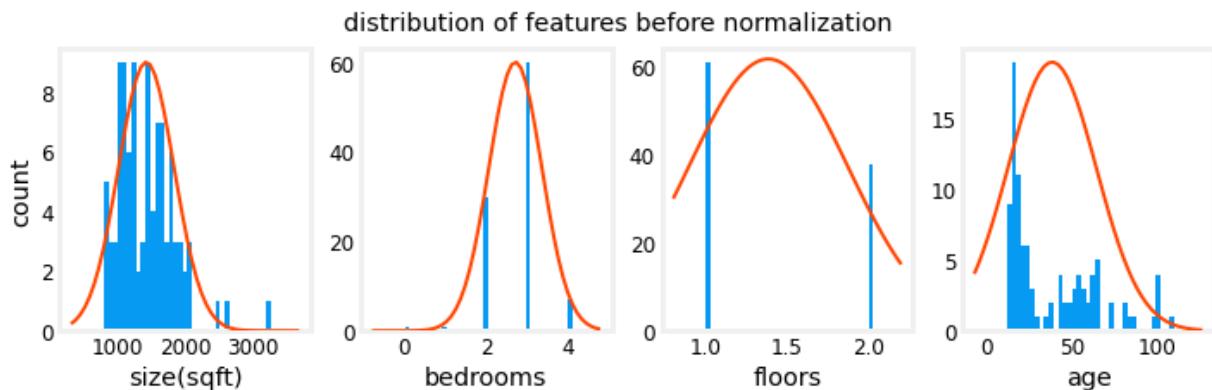
$$\mu_j = \frac{1}{m} \sum_{i=0}^{m-1} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=0}^{m-1} (x_j^{(i)} - \mu_j)^2$$

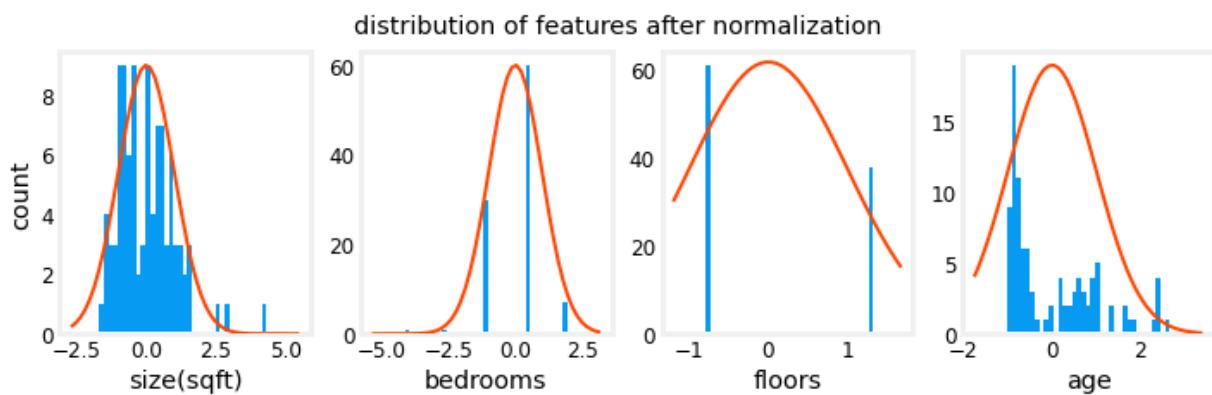
Where  $m$  is the number of examples.

---

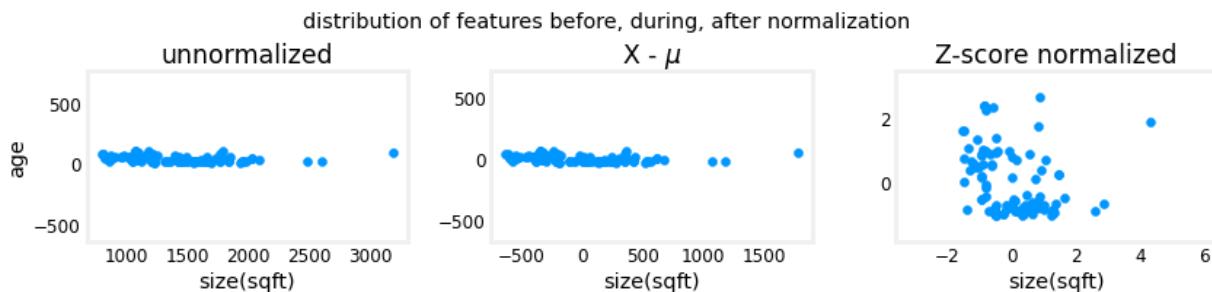
## Examples



See that they are centred around 0.



Below we see that its centered around 0 and been brought together.



Rescales the feature values to a range of [0, 1]. This is useful when you want to ensure that all features contribute equally to the distance calculations.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_normalized = scaler.fit_transform(df) # Rescales each feature to [0, 1]
```

## Z Score

Z-scores standardize a value relative to a distribution by measuring how many standard deviations it is from the mean. This is useful for [standardised/Outliers|Outliers](#) and [Normalisation](#).

## Introduction

Definition:

---

The Z-score of a value  $x$  is given by:  $Z = \frac{x - \bar{x}}{s}$

where  $\bar{x}$  is the sample mean and  $s$  is the sample standard deviation.

Interpretation:

- $Z = 0$ : The value equals the mean.
- $|Z| > 2$ : Indicates a possible outlier (if normality is assumed).
- Z-scores allow comparisons across different distributions.

Assumptions:

- Data is approximately normally distributed.
- Useful primarily when comparing existing values to a distribution.

Use Cases:

- Standardizing data for machine learning algorithms.
- Detecting anomalies.
- Ranking or scoring values.

Related terms:

- [Z-Test](#)
- [Z-Normalisation](#)
- [Z-Score](#)

## 2. Modified Z-Score

- **Formula:**

$$M = \frac{0.6745 \cdot (X - \text{median})}{\text{MAD}}$$

- $MAD$ : Median Absolute Deviation

- **Procedure:**

- Use this method for datasets with extreme outliers.
- Points with  $M > 3.5$  are typically anomalies.

## Z Scores Vs Prediction Intervals

[Z-Score](#) and [Prediction Intervals](#) serve different purposes. Z-scores assess existing values within a dataset, while prediction intervals estimate the likely range for future observations.

Use Z-scores to evaluate existing values or standardize. Use prediction intervals to express uncertainty about where a **new** observation is likely to fall.

**Comparison Table:**

---

Feature	Z-Score	Prediction Interval
<b>Purpose</b>	Assess deviation from the mean	Forecast future values
<b>Formula</b>	$Z = \frac{x - \bar{x}}{s}$	$\bar{x} \pm t_{(\alpha/2, n-1)} \cdot s \cdot \sqrt{1 + \frac{1}{n}}$
<b>Distribution</b>	Standard Normal (Z)	Student's t-distribution
<b>Use case</b>	Outlier detection, normalization	Prediction of new measurements
<b>Width of range</b>	Based on fixed $\sigma$	Wider—accounts for both sampling error and variability
<b>Needs population <math>\sigma</math>?</b>	Yes (or large $n$ to approximate)	No (uses sample $s$ and $t$ for small $n$ )

## Z Test

The Z-test is a statistical method used to determine if there is a significant difference between the means of two groups or to compare a sample mean to a known population mean when the population standard deviation is known.

It is typically applied when the sample size is large (usually  $n > 30$ ).

## Types of Z-tests

- One-Sample Z-test:** This test compares the mean of a single sample to a known population mean. It assesses whether the sample mean significantly differs from the population mean.
- Two-Sample Z-test:** This test compares the means of two independent samples. It is used when both sample sizes are large and the population variances are known or can be assumed to be equal.

## Characteristics of the Z-distribution

The Z-distribution is a normal distribution with a mean of 0 and a standard deviation of 1. It is symmetric and bell-shaped, which allows for the application of the [Central Limit Theorem](#). As sample sizes increase, the distribution of sample means approaches a normal distribution, making the Z-test applicable.

## Assumptions

For the Z-test to be valid, certain assumptions must be met:

- The data should be normally distributed, especially for smaller sample sizes. However, with large samples, the Central Limit Theorem allows for the Z-test to be used even if the data is not perfectly normal.
- The samples should be independent of each other.
- The population standard deviation should be known.

## Test Statistic

---

The test statistic for the Z-test is calculated using the formula:

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

where:

- $\bar{X}$  = sample mean
- $\mu$  = population mean (or mean of the second sample in the two-sample test)
- $\sigma$  = population standard deviation
- $n$  = sample size

This formula allows for the comparison of the sample mean to the population mean, standardizing the difference in terms of standard deviations.