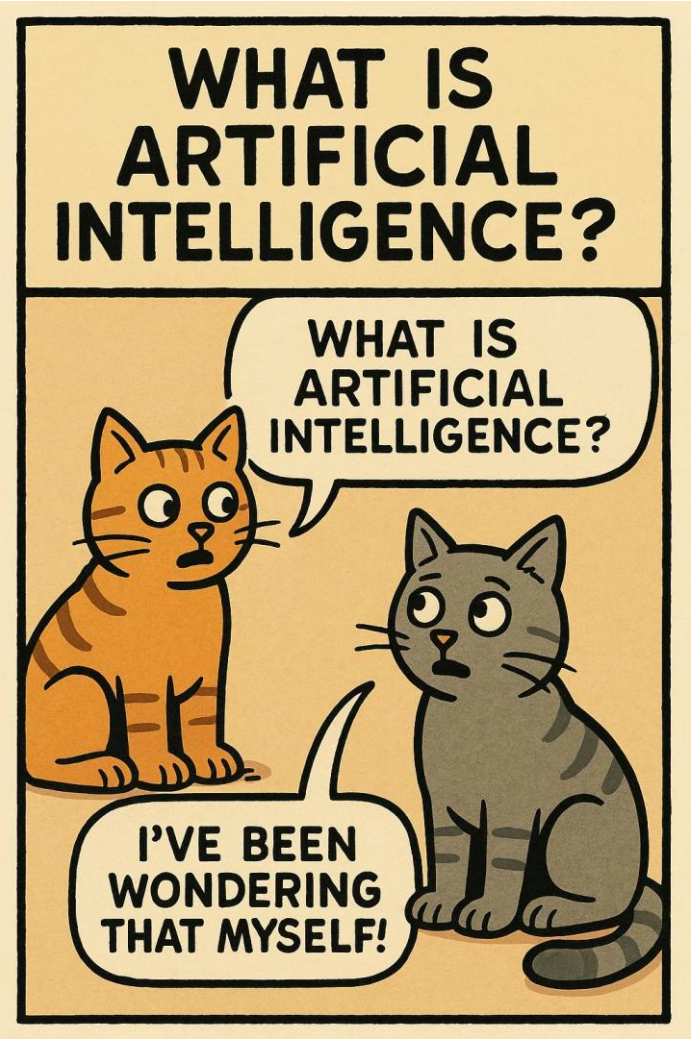
SPE AI Machine Learning for Energy Professionals Boot Camp

Dr. Luigi Saputelli

July 31st, 2025

Course Content at a Glance

- **Session 1: Introduction to AI and Machine Learning**
  - Fundamentals of AI and machine learning
  - Overview of key algorithms and their relevance to energy workflows
  - Resources, communities, and learning paths for continued development
- **Session 2: Data Processing & Python for Energy Analytics**
  - Data collection, cleaning, and transformation
  - Introduction to Python and essential libraries
- **Session 3: Use Cases & Hands-On Implementation**
  - Upstream, Midstream & Downstream
  - Enterprise & Cross-Functional
- **Way Forward**



About your Instructor

- Dr. Luigi Saputelli
  - 35+ years in industry (Production and Reservoir Engineering, Integrated Production Modeling)
  - 4+ years in Research and Academia (Advanced Process Control, Real-time Optimization)
  - 12+ years in consulting (Digitalization of Upstream Processes, Data Management, Machine Learning)

About You

1. Where are you from (country)?
2. How may years of work experience?
3. Which discipline are you in?
4. Which Company are you from?
5. What do you hope to take out of this course?
6. What are your objectives after completing this course?

What do you know about AI & ML?

1. What is AI? (in 2-3 words)
2. What is ML? (in 2-3 words)
3. Did you take Google Colab Tutorial for Beginners ?
4. Did you take Learn Python Basics using Google Colab ?
5. Did you take AI Python for Beginners from Andrew Ng – Deeplearning.ai ?
6. Have you studied any machine learning before?
7. Do you know the difference between Python and Anaconda?

Objective

- To provide a comprehensive foundation in AI for Energy Professionals, enabling them to tackle common problems in field development and reservoir management effectively using AI and machine learning Python tools.

Overview

- An immersive one-day course introduces key AI and machine learning concepts, practical data processing workflows, and hands-on coding review.

- Real-life case studies drawn from field operations will help participants understand how to implement solutions and generate insights that matter.
  - Upstream
  - Midstream
  - Downstream
  - Enterprise & Cross-Functional

- The boot camp concludes with a guided project consolidating the skills gained.

Detailed Program Outline

- **Session 1: Introduction to AI and Machine Learning**
  - Fundamentals of AI and machine learning
  - Overview of key algorithms and energy workflows
  - Resources and learning paths for continued development
- **Session 2: Data Processing & Python for Energy Analytics**
  - Data collection, cleaning, and transformation
  - Introduction to Python and essential libraries
- **Session 3: Use Cases & Hands-On Implementation**
  - **Upstream**
    - Predicting shut-in wells using performance classification
    - Virtual metering from real-time production data
    - Log permeability prediction from core data
    - Saturation and pressure map generation from well logs
    - Seismic feature detection using pattern recognition.
    - Drilling ROP (Rate of Penetration) Prediction
  - **Midstream**
    - Pipeline leak detection and predictive maintenance using sensor data.
    - Flow rate forecasting in transportation networks
    - Anomaly detection in SCADA systems for asset integrity
    - Predictive corrosion modeling in pipelines
  - **Downstream**
    - Demand forecasting and inventory optimization at refineries
    - Quality prediction and control in process streams
    - Energy consumption modeling and efficiency optimization in plant operations
    - Scheduling and optimization of blending operations
  - **Enterprise & Cross-Functional**
    - Automated report generation from structured and unstructured data
    - Document classification and Natural Language Programming for regulatory compliance
    - AI-driven market analysis for crude/product pricing
- **Final Project:**
  - Participants to develop their own application of AI and ML techniques to a sample dataset relevant to their work challenges.
  - Participants will have ~2 weeks to submit their project results as a final presentation and code.
  - Top projects will be showcased in the following SPE Bahrain Local section meeting in August 2025.

Prerequisites

- Participants must bring their own personal laptops with minimum recommended capabilities:
  - Processor: Intel Core i5 (8th Gen or newer) or AMD Ryzen 5
  - Storage: Available 50 GB Hard disk space
  - Display: minimum 14" screen with Full HD (1920×1080) resolution
  - Operating System: Windows 10/11
  - Internet Access: Provided during the course
- Candidates need to show basic Python programming knowledge by taken these videos:
  - Google Colab Tutorial for Beginners (10 min) - required.
  - Learn Python Basics using Google Colab (44 min) - required.
  - AI Python for Beginners from Andrew Ng – Deeplearning.ai (4.25 hrs)

Instructions for Running Python Notebooks

- **Running Python**
  1. Create an account in **COLAB** https://colab.research.google.com/
  2. Sign in to **COLAB** https://colab.research.google.com/
  3. Open Notebook Files stored in your Google Drive or GITHUB repository
  4. Optionally you can access any other Python local or online

- **Downloading Course Materials**
  1. Create an account in **GITHUB** → Sign up at https://github.com/
  2. Sign in to **GITHUB** https://github.com/
  3. Access Repository https://github.com/saputelli/SPE-AI_ML-Course
  4. Download SPE-AI_ML-Course repository (Code→Download Zip)
  5. Fork it to your account (Fork → +Create New Fork → Create fork)

SPE AI Machine Learning for Energy Professionals Boot Camp

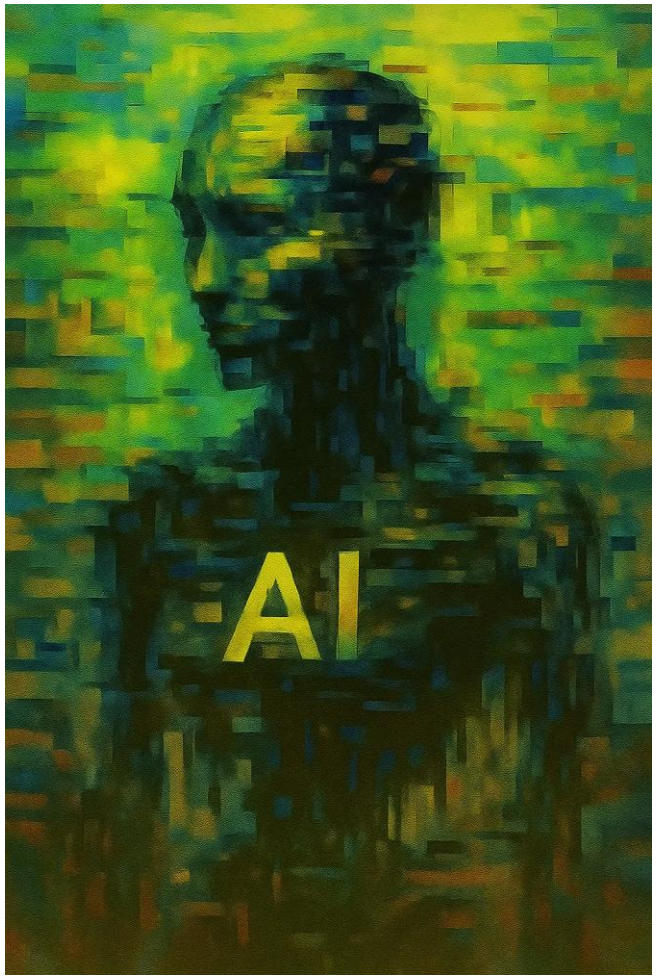
Session 1: Introduction to AI and Machine Learning

Agenda

- **Session 1: Introduction to AI and Machine Learning**
  - Fundamentals of AI and machine learning
  - Overview of key algorithms and their relevance to energy workflows
  - Resources and learning paths for continued development
- **Session 2: Data Processing & Python for Energy Analytics**
  - Data collection, cleaning, and transformation
  - Introduction to Python and essential libraries
- **Session 3: Use Cases & Hands-On Implementation**
  - Upstream, Midstream & Downstream
  - Enterprise & Cross-Functional
- **Way Forward**

Why AI and Machine Learning in Energy?

- Better informed decision making
- Availability of data sources (volume, variety, velocity)
- Support traditional models with unorthodox evidence-based engineering



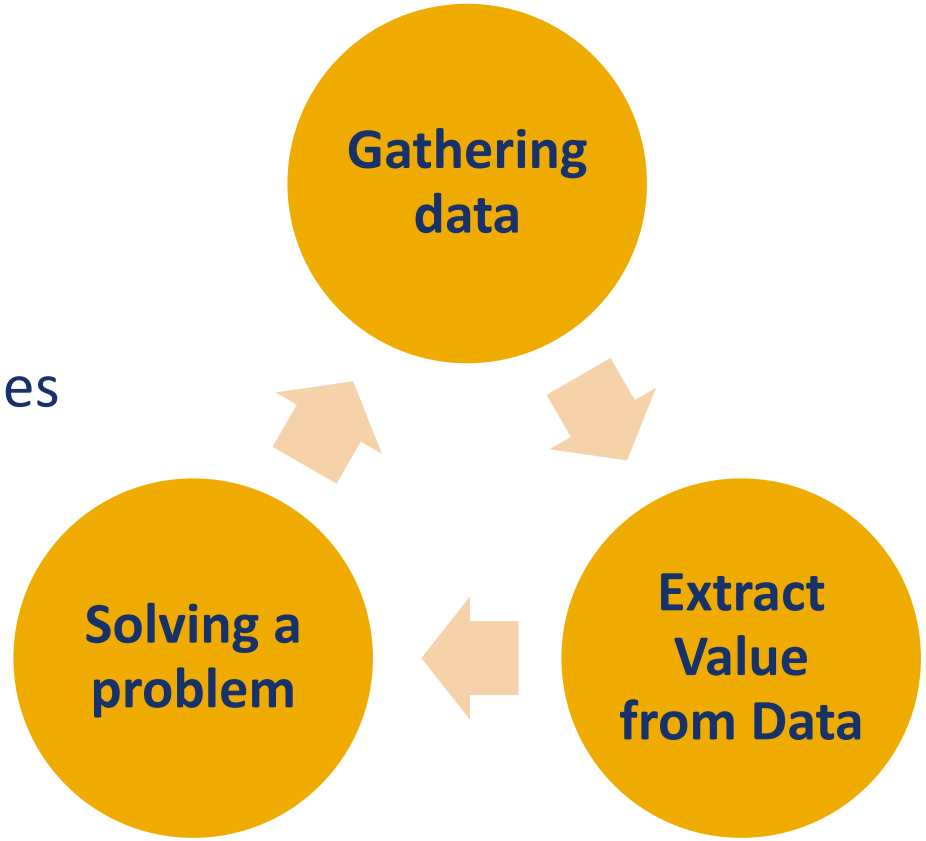
AI+ML makes us aware of (data) relationships that were not seen before

AI+ML raises fear among us because of knowledge about programming

Data Science and Analytics

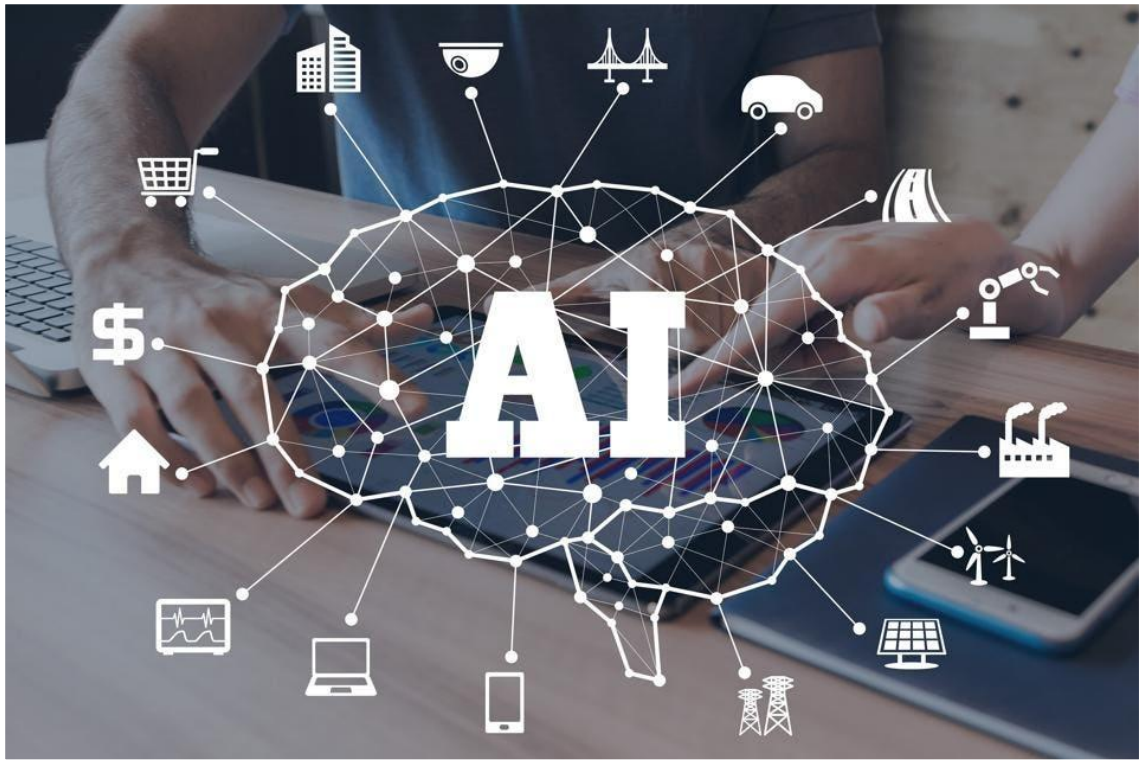
Solving a problem, discovering patterns and unexpected rules, etc

- Gathering data
  - numbers, text, images, sound
  - Private vs public.
- Extract Value from Data
  - Discovering patterns and unexpected rules
  - Reevaluating Hypothesis
- Solving a problem



AI is here

- When you book a flight, AI decides **what you pay**.
- AI determine whether you **get a loan**, are **eligible for welfare**, or get **hired**.
- AI help determine who gets **released from jail**.
- AI **translate the texts** you read.
- **Virtual assistants**, operated by speech recognition, entered many households.
- **Self-driving cars** are a reality.
- Generative AI allows millions of **document interrogation, customized customer service, multimedia and new software creation**.

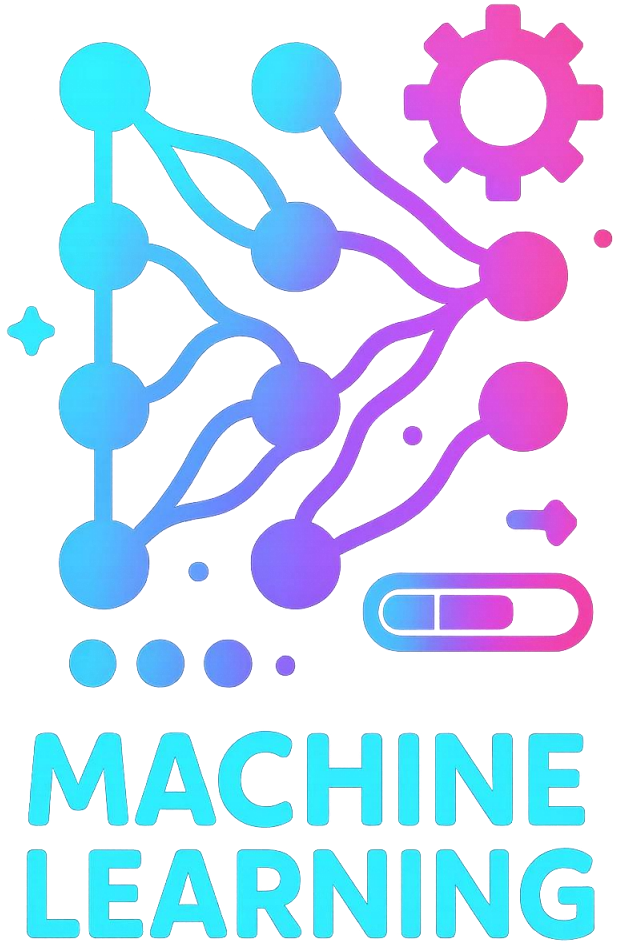


Machine Learning (ML)

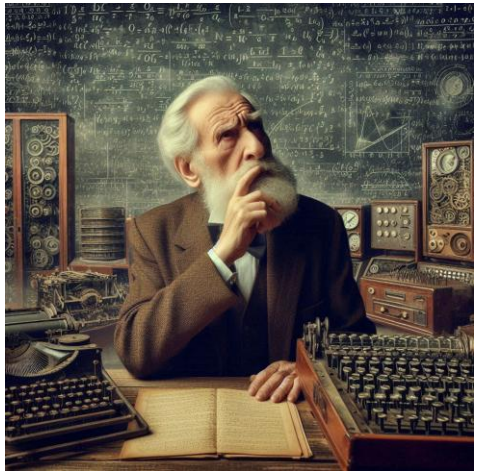
ML focuses on developing **algorithms that allow machines to learn from data and make predictions or decisions based on it**.

Key topics relevant to this study include:

- types and algorithms of machine learning,
- modeling approach and performance assessment,
- data processing and quality controls and
- key steps for deployment and integration.

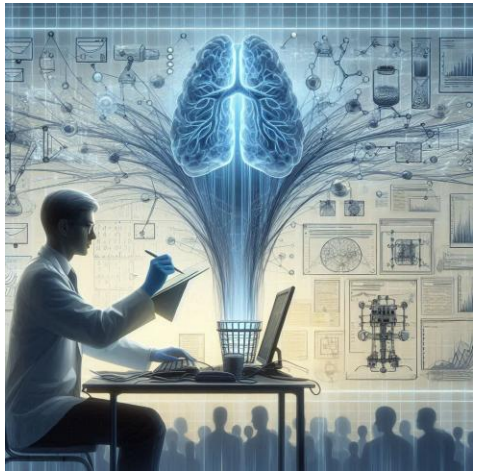


AI Evolution



1950s-1980s
Early AI research: laying the groundwork

1980s-1990s
AI Winter: A Period of Stagnation

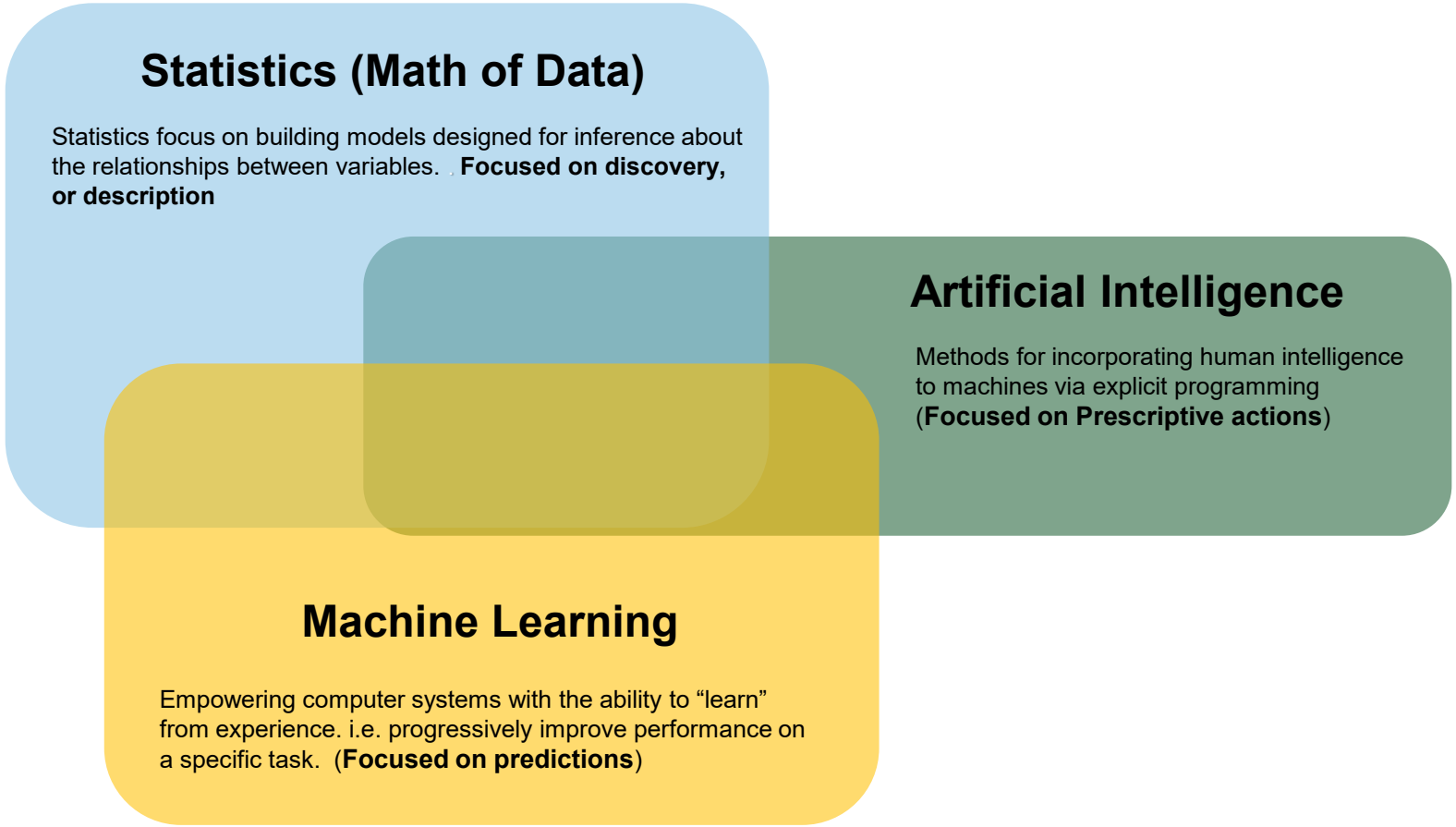


1990s-2010s
Machine Learning Renaissance: Data-Driven Insights

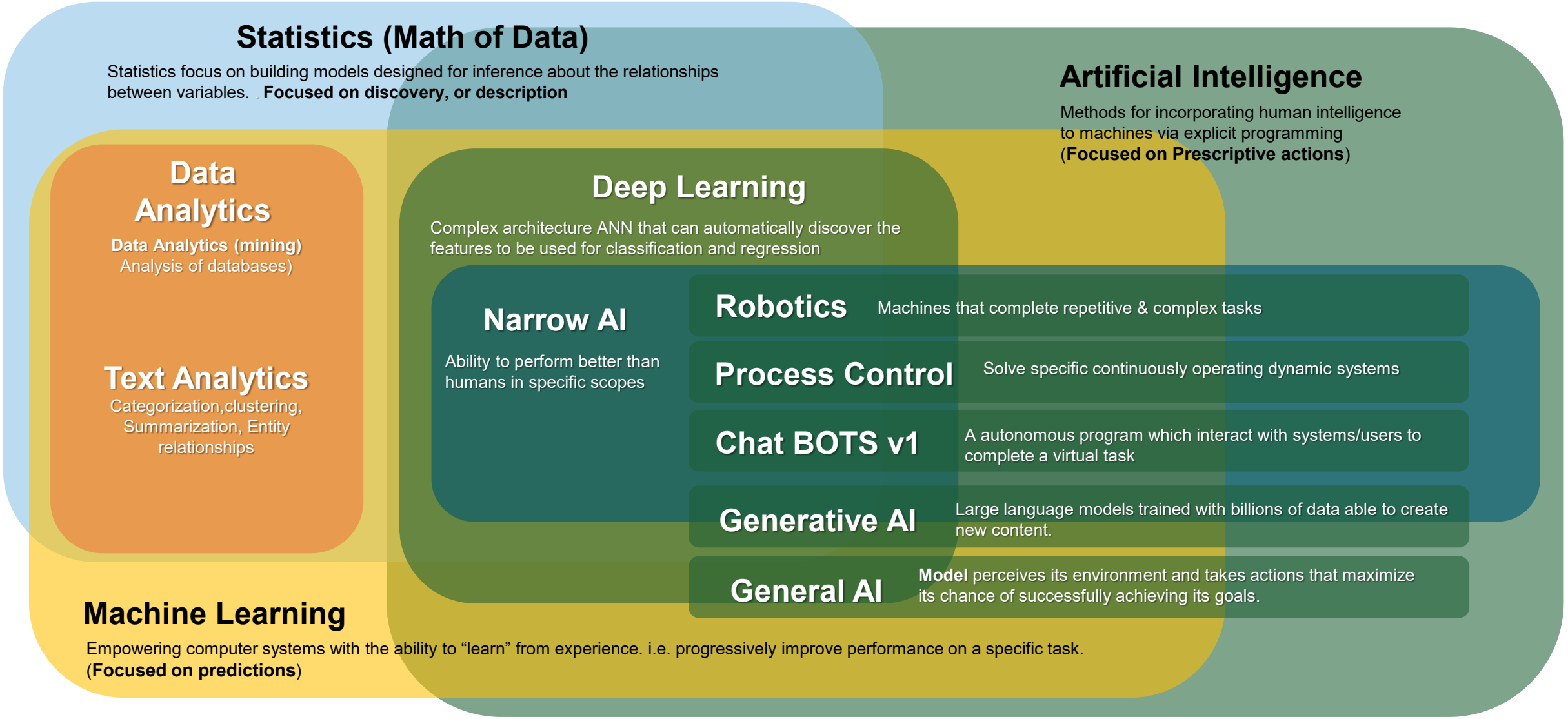
2010s-Presente:
Deep Learning Revolution: The Rise of Powerful AI

The Age of Platforms:
La Platform Era: Building Apps on the Shoulders of Giants

From Statistics to AI



From Statistics to AI



Statistics, AI and Machine Learning Capabilities

As we mature technologies and process capabilities, Data Analytics & AI help industry reshape its own future…

Statistics

Machine Learning

Artificial Intelligence

Access & Gather

- Monitoring and Surveillance
- Alerts on abnormal increase in pressure, temperature, etc.

Analyze & Diagnose

- Early detection of abnormal behavior of pumps
- Patterns to identify changes in behavior

Predict & Optimize

- Understand impact of key parameters
- Select optimum production scenario

Prescribe & Recommend

- Proactive maintenance of equipment
- Minimize downtime and improve oilfield economics

Cognitive (Ask, Discover, Decide)

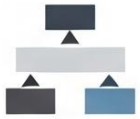
- Self correcting and evolving algorithms
- Drive Insights and improve Decision Making

Provide Awareness → **Improve Prediction** → **What-if scenarios** → **Improve decisions** → **Improve the future**

Value Proposition: From Descriptive to Prescriptive

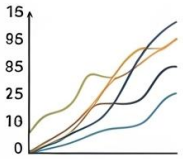
Applications	Well Performance	Rotating Equipment Analytics	Processing Plant	Drilling Analytics	Petrographic Image Analysis
Descriptive – Focus on gathering facts about past performance.	How does well performance rank against targets?	How does motor and pump perform against targets?	How does plant perform against targets?	How does drilling perform against targets?	What are dominant lithofacies driving reservoir performance?
Diagnostic - Understand the reasons behind the observed values.	What are the precursor events to well performance?	What are the precursor events to motor failure?	Why did plant performance degrade?	What are the causes of NPT?	How does lithofacies and completion affect well success?
Predictive – Forecasting to investigate decision influencing the process.	How will the well perform if intervention is performed today?	When will the motor or pump fail?	How will plant perform if targets are changed?	How will drilling performance evolve at current conditions?	How will the well perform for various location direction and completion?
Prescriptive – Recommend decisions that improve performance.	Well will improve 30% performance if WSO is performed.	Pump will improve 10% performance if frequency is changed	Plant will increase 5% performance if certain plan is implemented	Drilling will improve if WOB/ROP changes are implemented	The optimum well completion and direction are XYZ.

Key AI & ML Algorithms



Classification

Regression



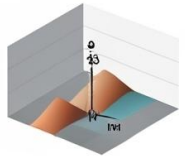
Time-Series Forecasting

Interpolation/
Extrapolation

Patternl Recognition

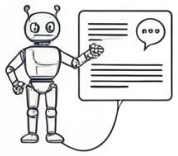
Anomaly Detection

Image Processing



Optimization Algorithms

Natural Language Processing (NLP)



Natural Language Generation (NLG)

Supervised vs Unsupervised Classification: By Learning Paradigm

Supervised Learning

- Algorithms that learn from labeled data, where each input has a corresponding desired output. **The goal is to predict an output for new, unseen inputs.**
- **Classification**: Categorizing data into discrete classes (e.g., spam detection, disease diagnosis, churn prediction) including pattern recognition (images and NLP)
- **Regression**: Predicting a continuous value (e.g., stock price prediction, house price estimation, sales forecasting) including timeseries

Unsupervised Learning

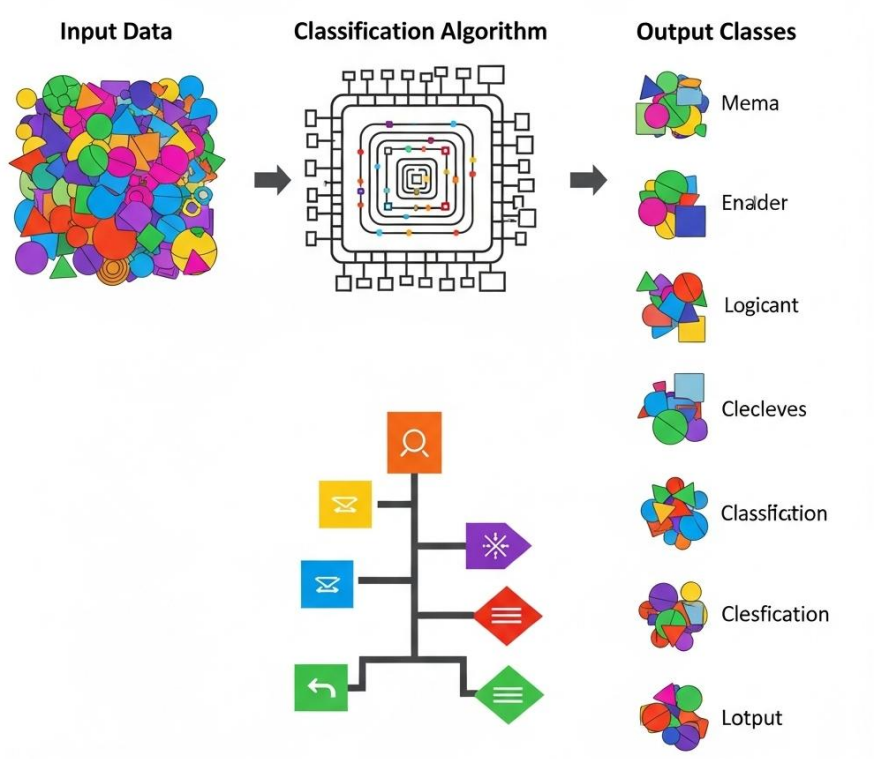
- Algorithms that **find patterns, structures, or relationships** within unlabeled data without explicit output guidance.
- **Clustering**: Grouping similar data points (e.g., image processing, customer segmentation, document categorization (NLP), anomaly detection).
- **Dimensionality Reduction**: Reducing number of features while retaining most of the information
- **Interpolation/Extrapolation** finding underlying data structure

Generative AI

- Algorithms capable of generating new data instances that resemble the training data, rather than just classifying or predicting.
- **Natural Language Generation (NLG):** Specifically focuses on producing human-like text from data.
- **Pattern Recognition** and **Natural Language Processing (NLP)** when dealing with labeled data

Classification

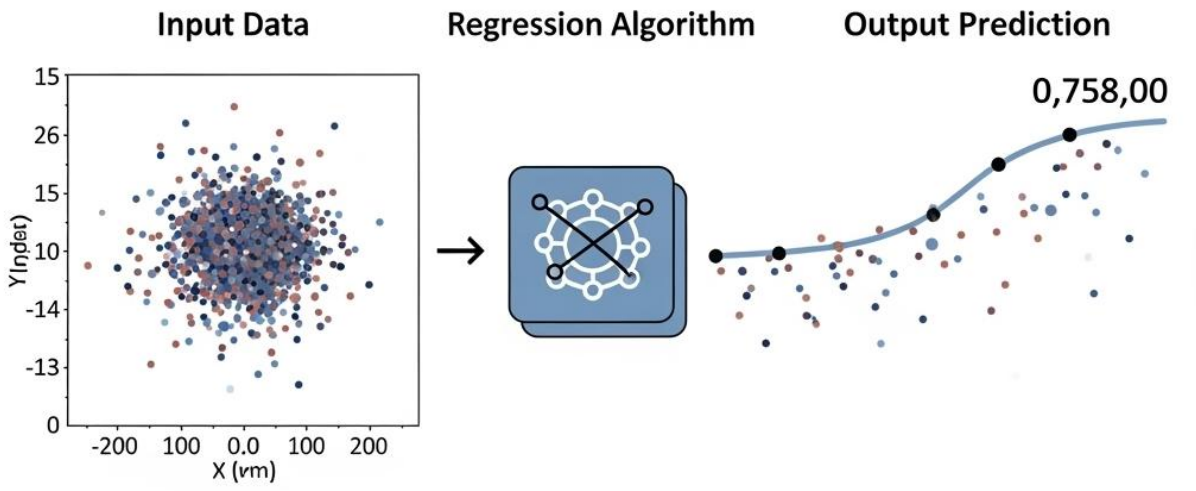
- **Definition**: A supervised learning technique used to categorize data into predefined discrete classes or categories.
- **Key Characteristics**: Requires labeled training data, produces discrete output, and uses evaluation metrics like accuracy and F1-score.
- **Common Use Cases**: Email spam detection, medical diagnosis, and customer churn prediction.



- Classification/Regression for Predictive Maintenance: SVMs, Random Forests, Neural Networks.

Regression

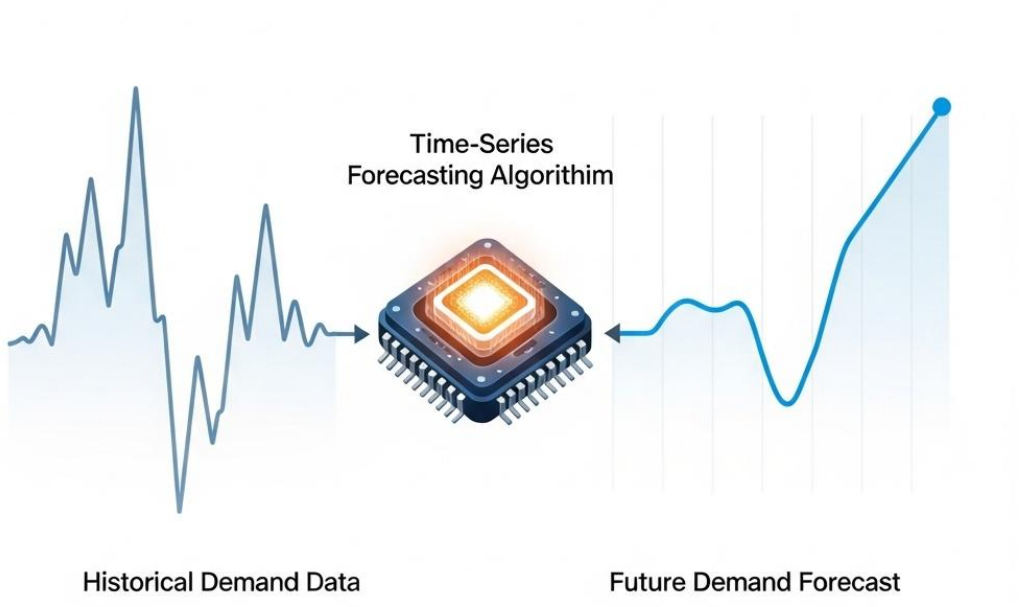
- **Definition**: A supervised learning technique used to predict a continuous numerical value based on input features.
- **Key Characteristics**: Requires labeled training data, produces continuous output, and uses evaluation metrics like Mean Squared Error (MSE) and R-squared.
- **Common Use Cases**: House price prediction, stock price forecasting, and sales forecasting



- **Regression Algorithms**: Linear & Polynomial Regression, Random Forests, Gradient Boosting, Ridge/Lasso Regression, Partial Least Squares (PLS), Principal Component Regression (PCR), Support Vector Regression (SVR), Neural Networks, Recurrent Neural Networks (RNNs) for time-series data.

Time-Series Forecasting

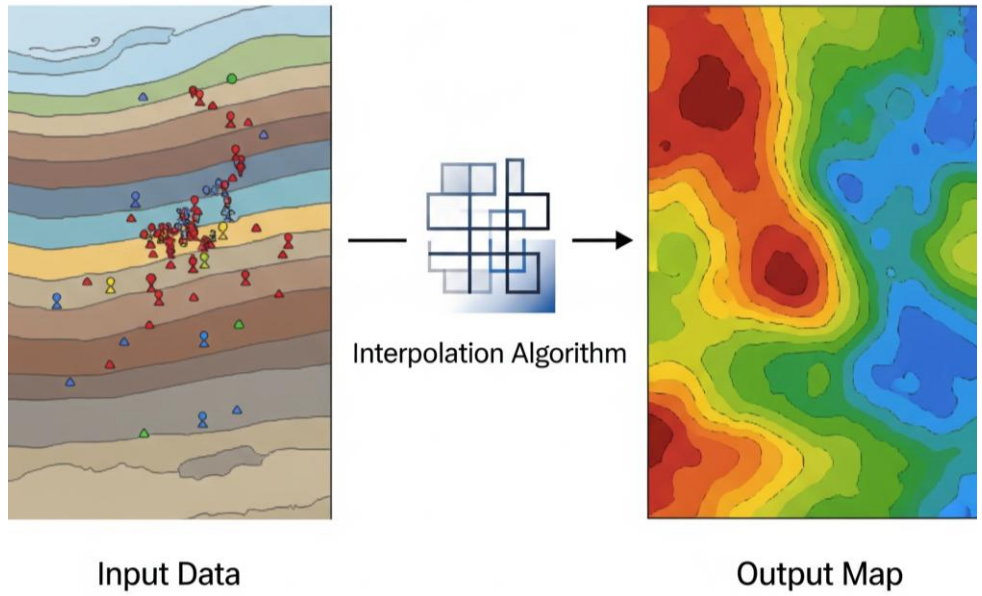
- **Definition**: A specialized type of regression that uses historical time-ordered data to predict future values.
- **Key Characteristics**: Involves temporal dependence, patterns like trends and seasonality, and uses evaluation metrics similar to regression.
- **Common Use Cases**: Weather forecasting, demand planning, and financial market prediction



- **Algorithms:** ARIMA, SARIMA, Exponential Smoothing, Prophet, Neural Networks (LSTMs), Recurrent Neural Networks (RNNs) for time-series data.

Spatial Interpolation/Extrapolation

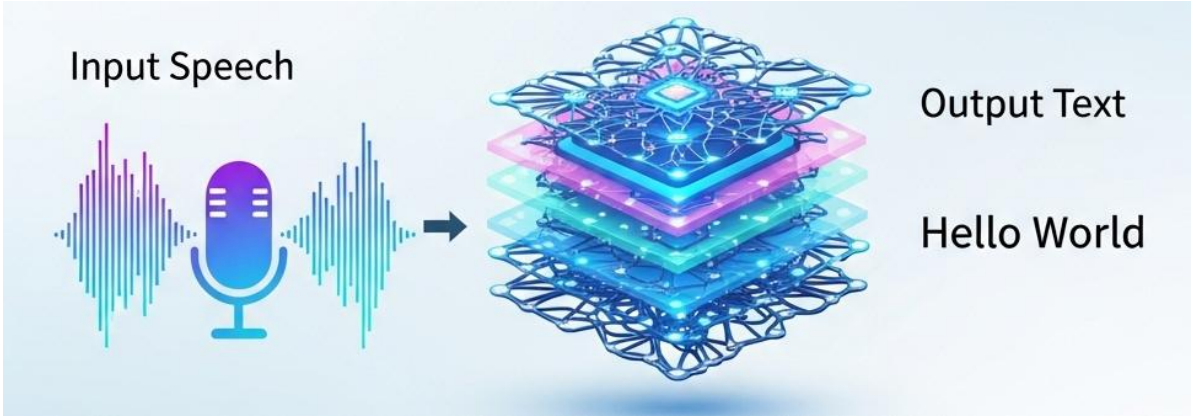
- **Definition**: Interpolation estimates values within a range of known data points, while extrapolation estimates values outside that range.
- **Key Characteristics**: Interpolation is more reliable with dense data points, while extrapolation is riskier.
- **Common Use Cases**: Missing data imputation, geostatistics, and scientific modeling.



- **Interpolation/Extrapolation Algorithms**: Kriging, Inverse Distance Weighting (IDW), Nearest Neighbor, Radial Basis Functions (RBFs).

Pattern Recognition

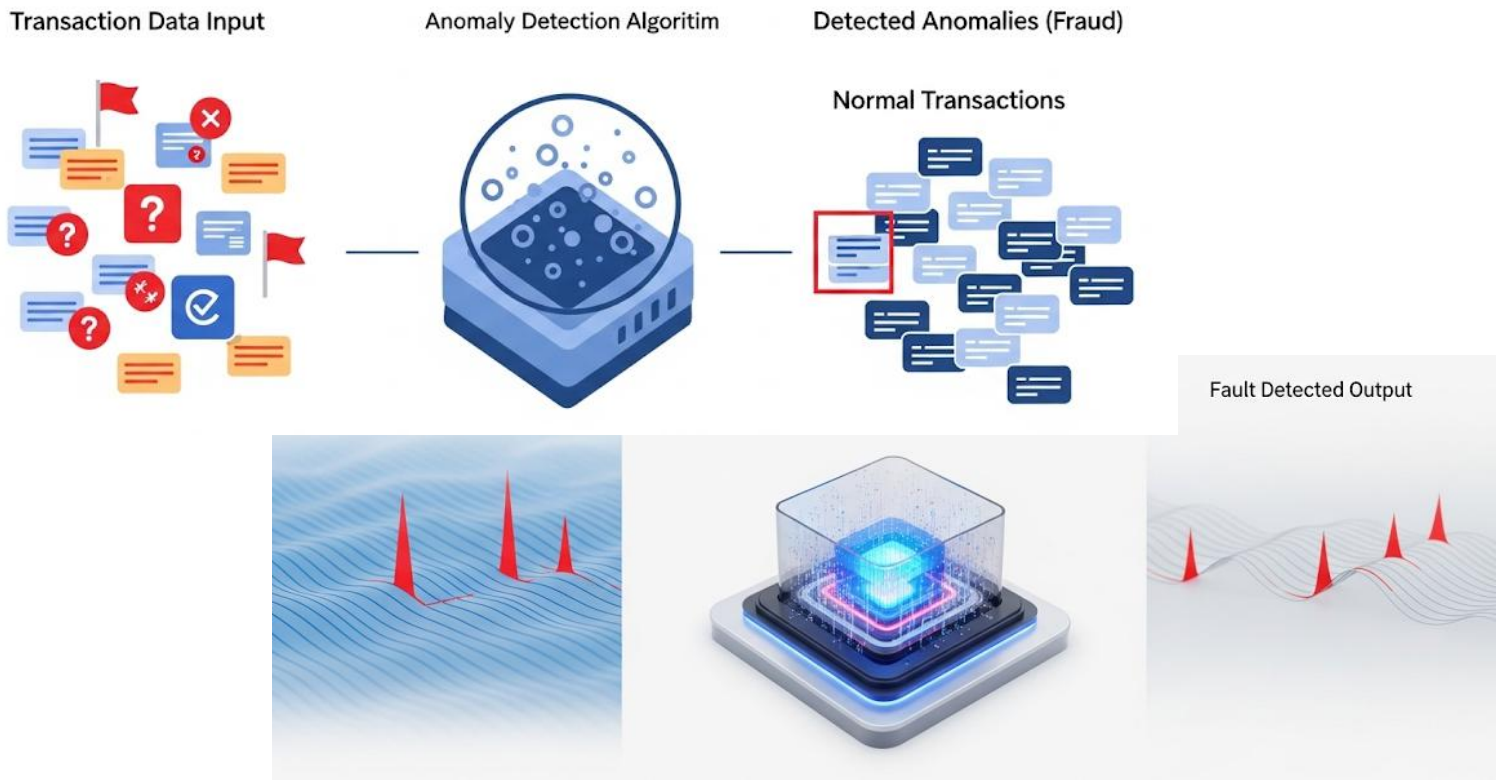
- **Definition**: Identifying meaningful regularities or patterns in data.
- **Key Characteristics**: Involves feature extraction, learning from examples, and generalization.
- **Common Use Cases**: Biometrics, speech recognition, and medical image analysis



- **Pattern Recognition Algorithms**: Convolutional Neural Networks (CNNs), Clustering (K-Means, DBSCAN), Principal Component Analysis (PCA)

Anomaly Detection

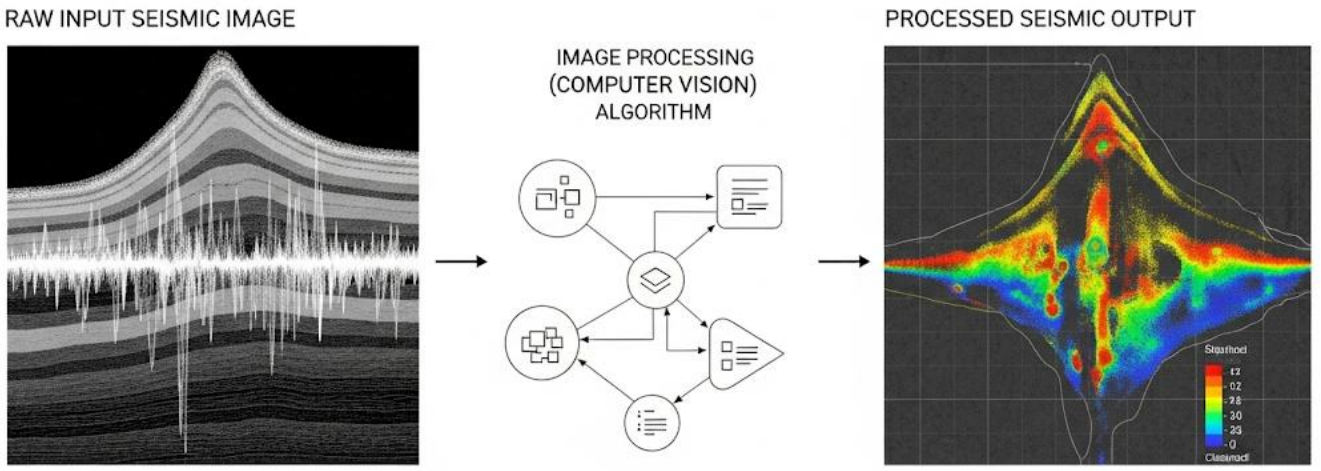
- **Definition**: Identifying data points that deviate significantly from the majority, signaling unusual behavior.
- **Key Characteristics**: Anomalies are rare, require a definition of normal behavior, and can be domain-specific.
- **Common Use Cases**: Fraud detection, network intrusion detection, and industrial fault detection



- **Anomaly Detection Algorithms**: Isolation Forest, One-Class SVM, Autoencoders, Statistical Process Control (SPC) methods, Time-Series Anomaly Detection (e.g., ARIMA models with residuals).

Image Processing (Computer Vision)

- **Definition**: Manipulation and analysis of digital images to enhance them, extract information, or prepare them for other AI tasks.
- **Key Characteristics**: Operates on pixel data, involves transformations, feature extraction.
- **Common Use Cases**: Photo editing, acoustic seismic images, medical imaging, and satellite imagery analysis



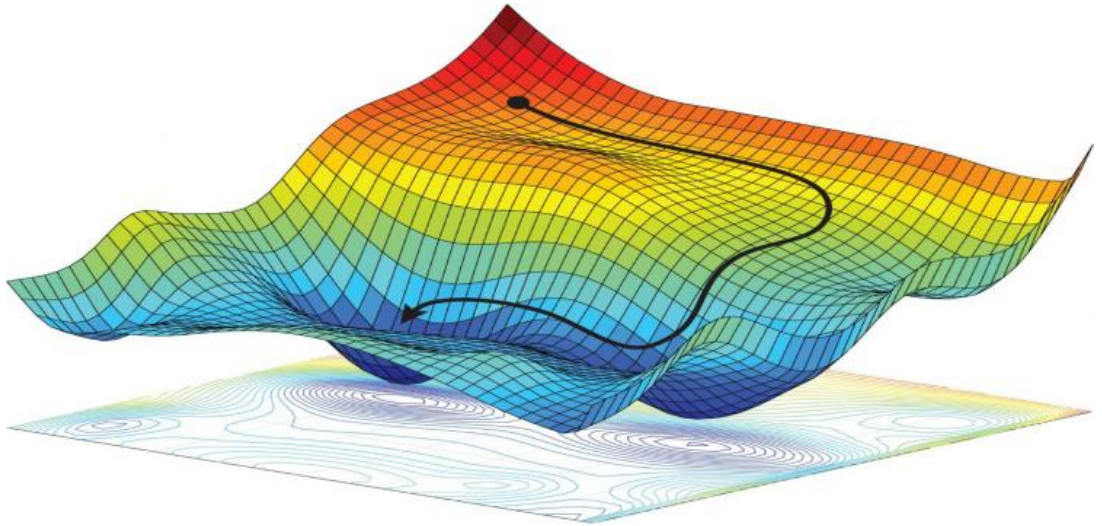
- **Image Processing Algorithms**: Convolutional Neural Networks (CNNs), Clustering (K-Means, DBSCAN), Principal Component Analysis (PCA)

Automatic (Autonomous) Control

- **Definition**: to regulate (stability, quality) the behavior of dynamic processes without continuous human intervention at the right time (responsiveness) and possibly improve with time (optimize).
- **Key Characteristics**: measuring system outputs, comparing them to desired setpoints (with accuracy), and adjusting inputs to minimize error (feedback loop) and rejecting external disturbances (robustness).
- **Common Use Cases**: from HVAC, car cruise control, ABS, house machines, gas-lift, chemical processing, and airplane autopilot and flight stability
- **Control Algorithms**: Model Predictive Control (MPC), PID Controllers, LQR (Linear Quadratic Regulator), Fuzzy Logic, Adaptive Control, Neural Network Control, Sliding Mode Control, Reinforcement Learning (RL).

Optimization

- **Definition**: Finding the best solution from a set of feasible solutions by maximizing or minimizing an objective function.
- **Key Characteristics**: Involves an objective function, decision variables, constraints, and iterative processes.
- **Common Use Cases**: Resource allocation, supply chain management, and model training

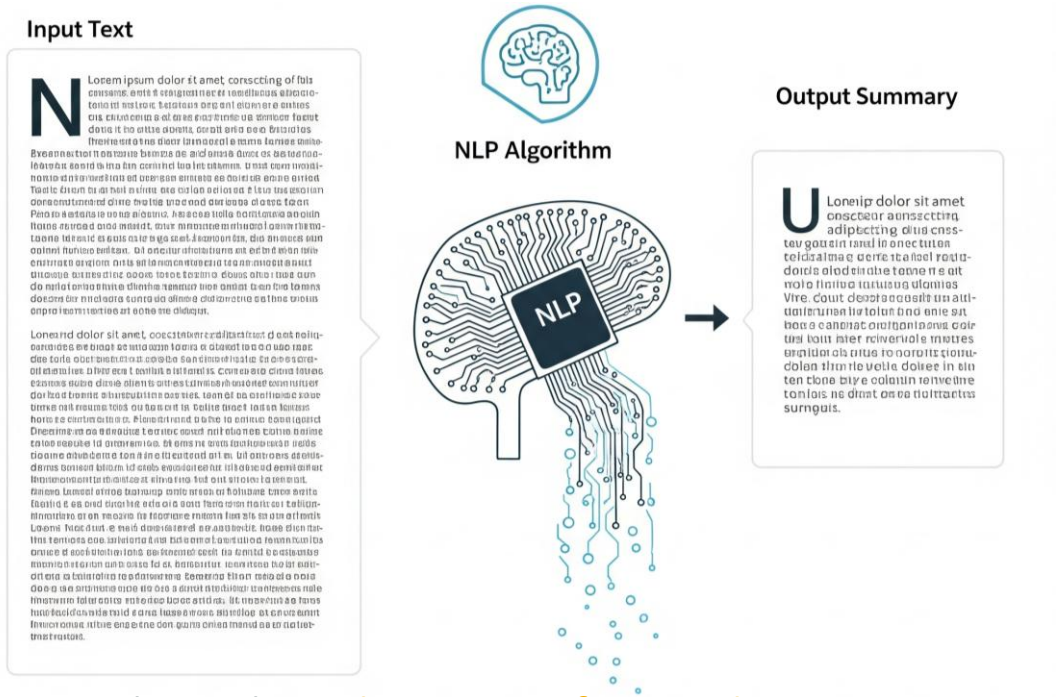


- **Optimization Algorithms**: Linear Programming, Integer Programming, Genetic Algorithms, Simulated Annealing, Reinforcement Learning.

Natural Language Processing (NLP)

Key step: Numerical representation of words and concepts

- **Definition**: Enabling computers to understand, interpret, and generate human language.
- **Key Characteristics**: Deals with text and speech data, addresses linguistic challenges, and involves preprocessing.
- **Common Use Cases**: Sentiment analysis, text summarization, and named entity recognition

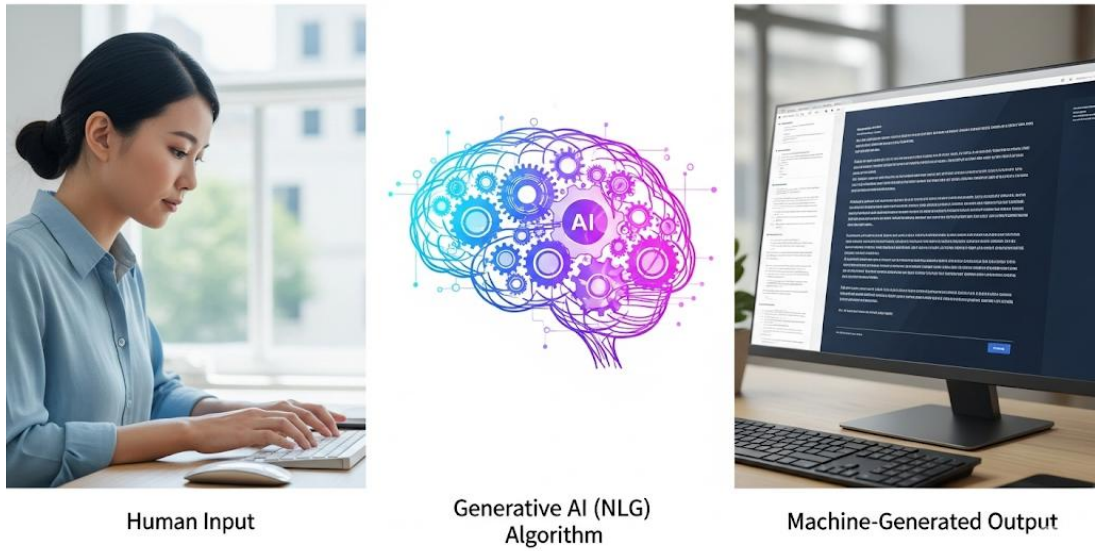


See Also Video: The Basics of Natural Language Processing

- **Natural Language Processing (NLP) Algorithms**: Text Classification (Naïve Bayes, SVM, Logistic Regression, Deep Learning with CNNs/RNNs/Transformers), Named Entity Recognition (NER), Topic Modeling (LDA), Text Summarization. **Sentiment Analysis Algorithms**: Lexicon-based, ML(SVM, Naïve Bayes), Deep Learning-based.

Natural Language Generation (NLG)

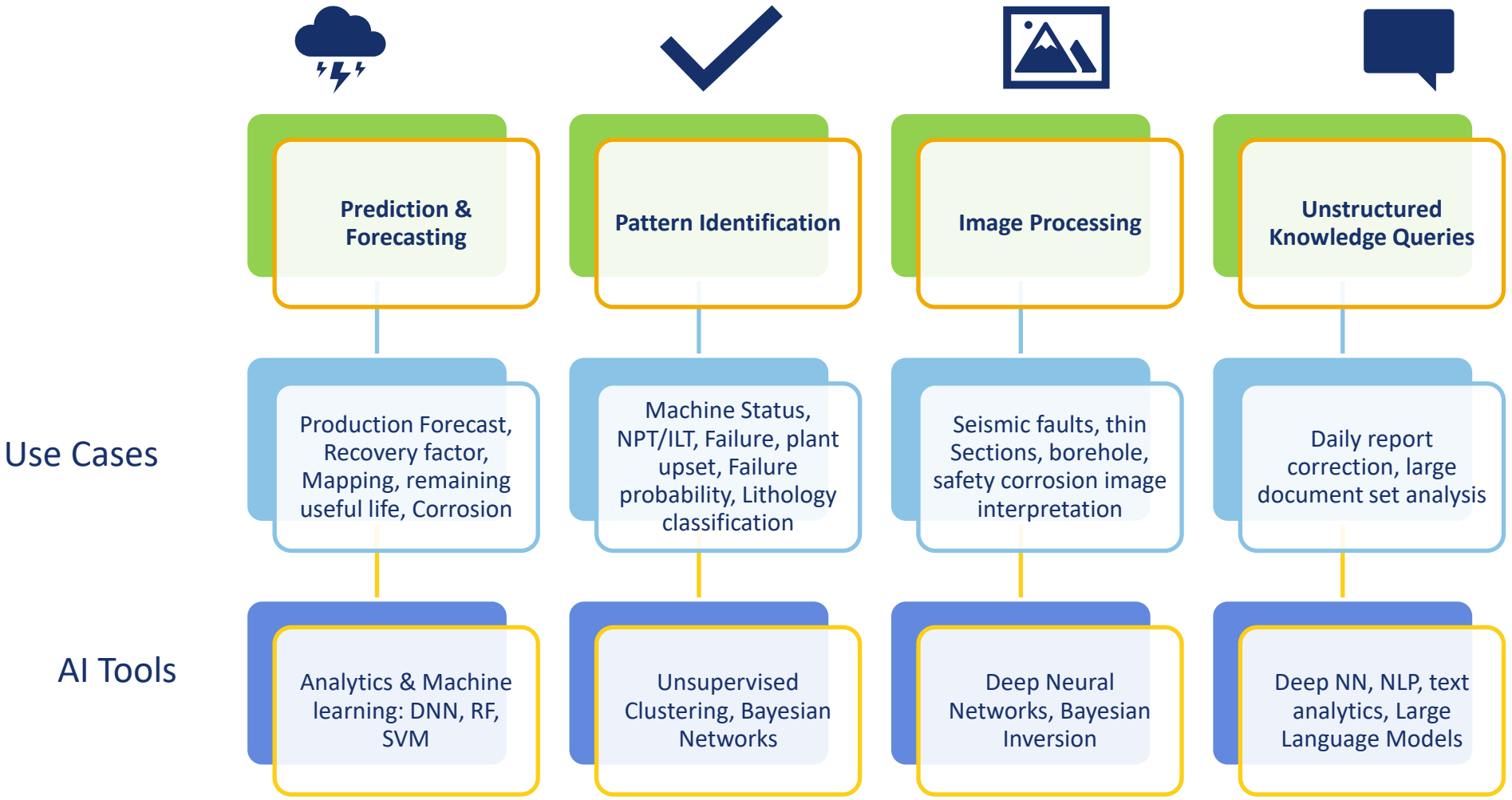
- **Definition**: Automatically generating human-like text from structured data or non-linguistic inputs.
- **Key Characteristics**: Involves content determination, text structuring, lexicalization, and realization.
- **Common Use Cases**: Automated report generation, chatbots, and content creation



Large Language Models explained briefly

- **Natural Language Generation (NLG) Algorithms**: Rule-based systems, Template-based systems, Neural Network-based NLG (e.g., Transformers like GPT for more complex generation).

Selected Areas of AI impacting Energy Industry



Energy Industry (Selected) Workflows

Upstream

- Predicting shut-in wells
- Virtual metering
- Permeability prediction from core
- Saturation and pressure mapping
- Seismic feature detection.
- Drilling Events Prediction & ROP Optimization

Midstream

- Pipeline leak detection
- Predictive maintenance
- Forecasting in transportation networks
- Anomaly detection in asset integrity
- Predictive corrosion modeling in pipelines

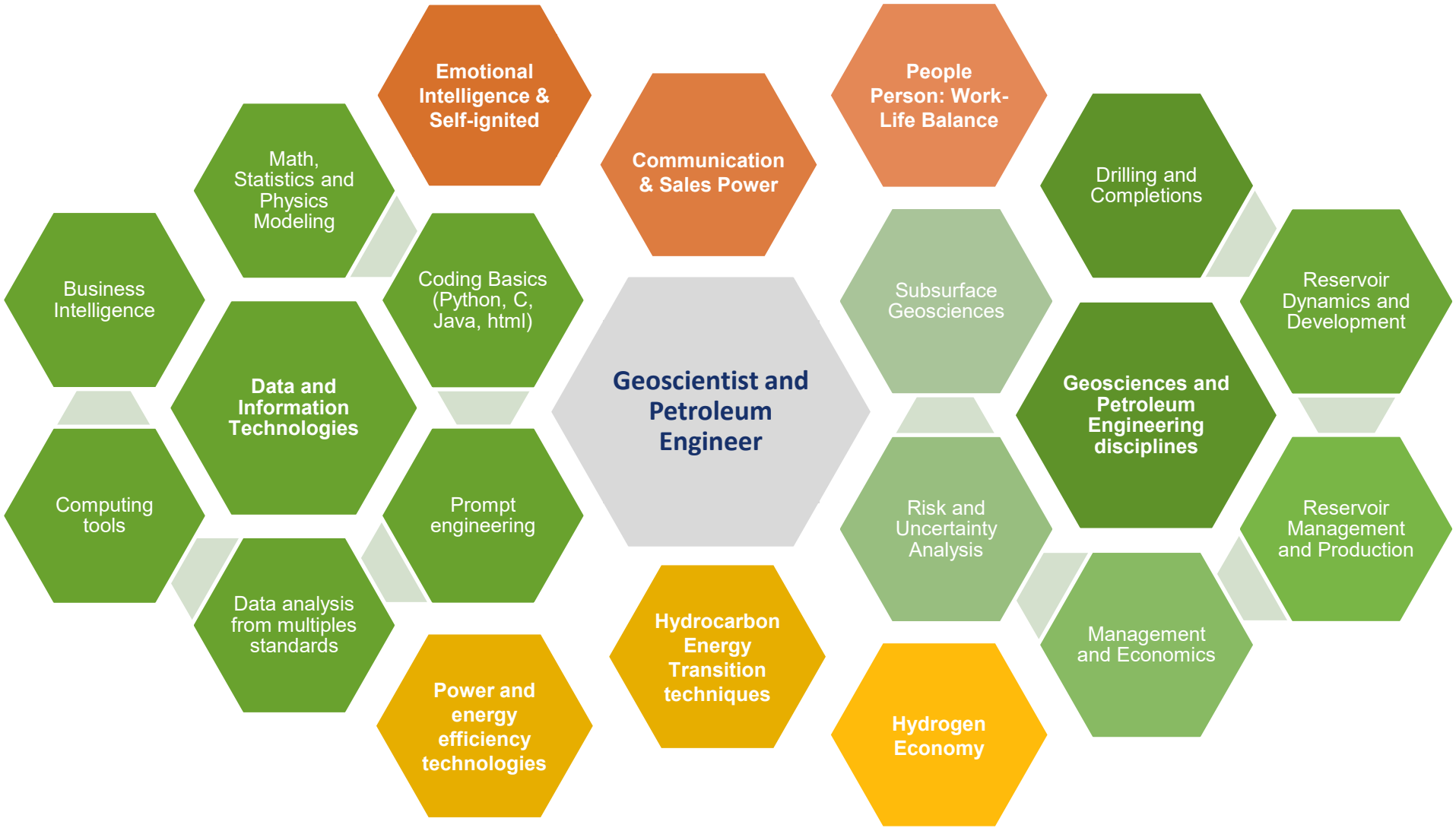
Downstream

- Demand forecasting and inventory optimization
- Quality prediction and control
- Energy consumption modeling and efficiency
- Scheduling and optimization

Enterprise & Cross-Functional

- Automated report generation
- Document classification compliance
- Market analysis for crude/product pricing

The Evolving Geoscientist & PE of the Future



Resources

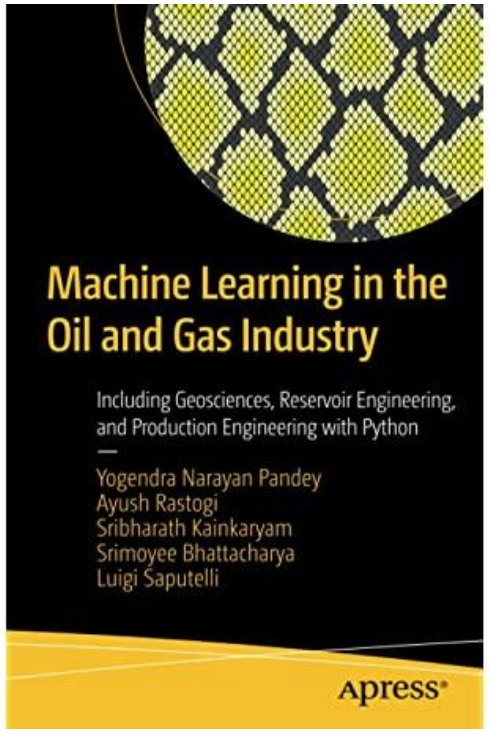
- SPE webinars (DSEA, ML, AI)
- Education programs (EDX, Coursera, LinkedIn, Datacamp)
- Help resources (Github, Youtube, SPE, SEG, SPWLA)
- Industry Challenges (Kaggle, DoE, SPE)
- Public Oil, Gas & Energy Data Sets

- ***A detailed list with links available in the Appendix***

Book Resources

Data Analytics in Reservoir Engineering eBook - June 2020

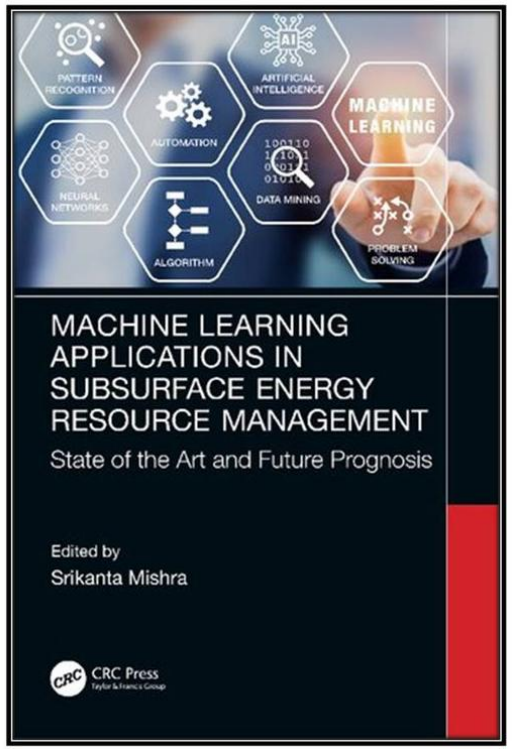
https://onepetro.org/books/book/52/Data-Analytics-in-Reservoir-Engineering



Machine Learning in the Oil and Gas Industry: … November 2020

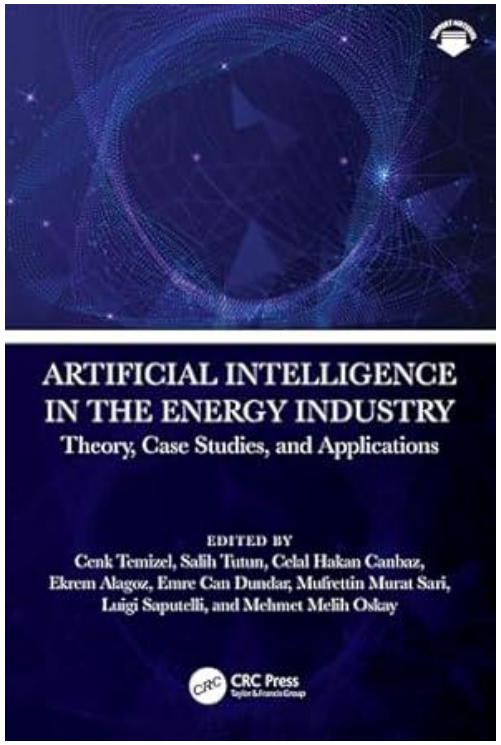
https://www.apress.com/gp/book/9781484260937

https://github.com/Apress/machine-learning-oil-gas-industry



Machine Learning Apps in Subsurface Energy Resource Management:

https://www.taylorfrancis.com/books/edit/10.1201/9781003207009



Artificial Intelligence in the Energy Industry November 2025

SPE AI Machine Learning for Energy Professionals Boot Camp

Session 2: Data Processing & Python for Energy Analytics

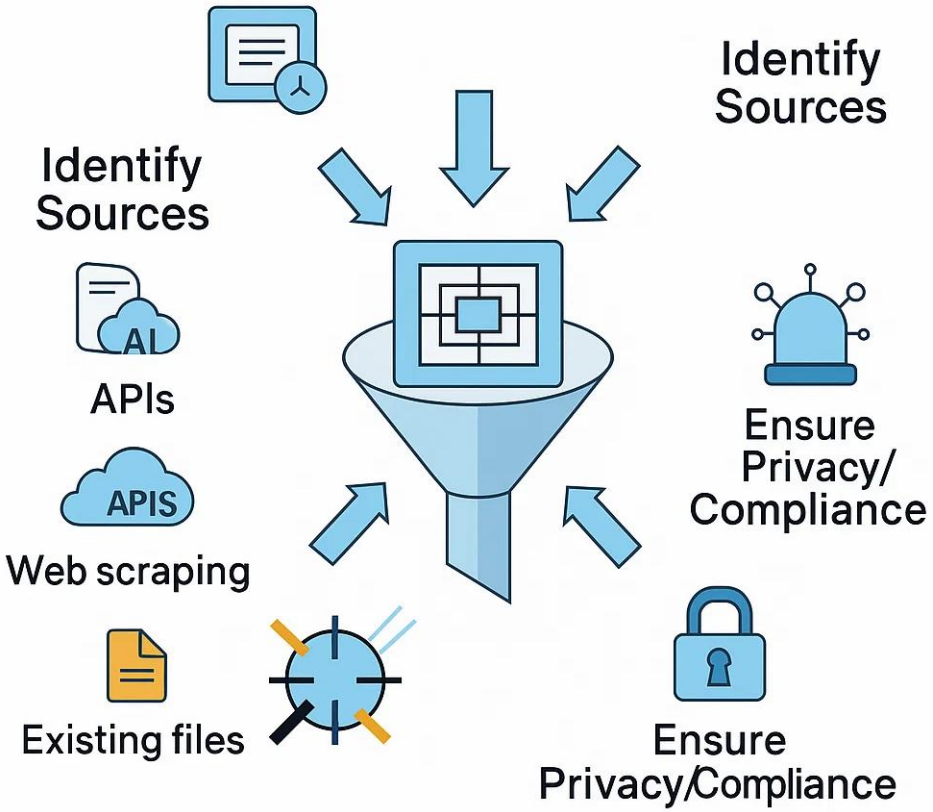
Agenda

- **Session 1: Introduction to AI and Machine Learning**
  - Fundamentals of AI and machine learning
  - Overview of key algorithms and their relevance to energy workflows
  - Resources, communities, and learning paths for continued development
- **Session 2: Data Processing & Python for Energy Analytics**
  - Data collection, cleaning, and transformation
  - Introduction to Python and essential libraries
- **Session 3: Use Cases & Hands-On Implementation**
  - Upstream, Midstream & Downstream
  - Enterprise & Cross-Functional
- **Way Forward**

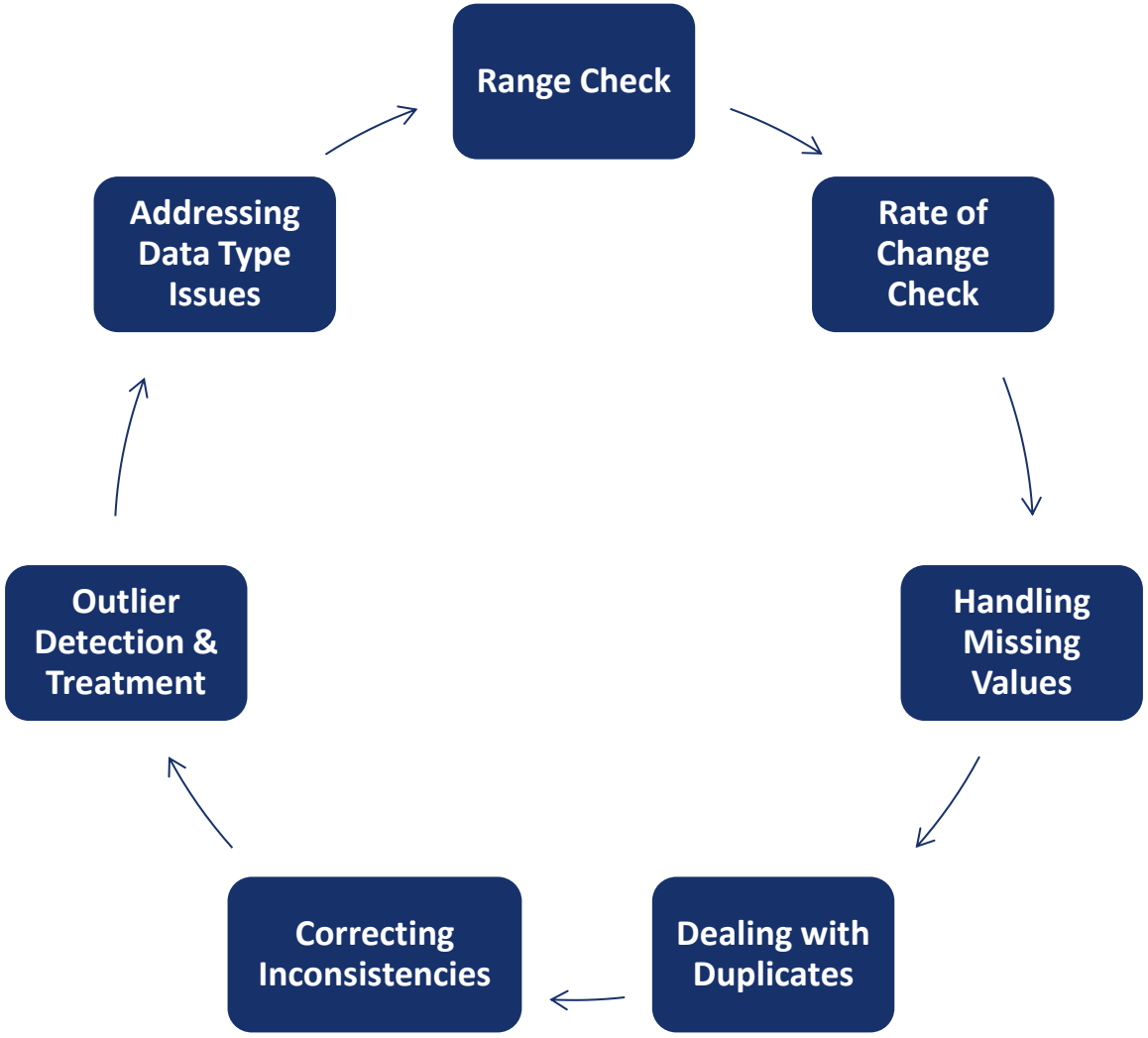
Data Collection

- **Objective:** Gather raw data from various sources relevant to the problem.

- **Activities:**
  - Identifying data sources (databases, APIs, web scraping, sensors, logs, surveys, existing files).
  - Extracting data in its raw format (e.g., CSV, JSON, XML, images, audio, text).
  - Ensuring data privacy and compliance (e.g., GDPR, HIPAA) during collection.



Data Cleaning (or “Data Wrangling"):



Data Transformation

- **Objective:** Convert cleaned data into a suitable format and scale for the specific machine learning algorithm.
- **Activities:**
  - **Feature Scaling/Normalization:**
    - **Standardization (Z-score normalization):** Rescaling data to have a mean of 0 and a standard deviation of 1. Useful for algorithms sensitive to feature magnitudes.
    - **Min-Max Scaling (Normalization):** Rescaling data to a fixed range, usually 0 to 1.
  - **Encoding Categorical Variables:**
    - **One-Hot Encoding:** Converting categorical variables into a binary (0/1) matrix.
    - **Label Encoding:** Assigning a unique numerical label to each category.
    - **Target Encoding:** Encoding categories based on the mean of the target variable.
  - **Feature Engineering:** Creating new features from existing ones to improve model performance (e.g., combining two columns, extracting month from a date, creating interaction terms).
  - **Dimensionality Reduction:** Reducing the number of features while retaining most of the important information (e.g., using PCA).
  - **Date/Time Feature Extraction:** Decomposing date/time stamps into components like year, month, day of week, hour.
  - **Text Preprocessing (for NLP):** Tokenization, lowercasing, stop-word removal, stemming/lemmatization, vectorization (e.g., TF-IDF, Word Embeddings).
  - **Image Preprocessing (for Computer Vision):** Resizing, cropping, normalization, data augmentation.

What is Python?

What is Python?	• Created by Guido van Rossum and first released in 1991 • Interpreted, flexible, general purpose programming Language • Human readable by design
Why Python?	• Easy to start • Ideal as an advance language • Extensive open community
What can I build with Python?	• Machine Learning Models • Artificial Intelligence Projects • Web Applications • Automation utilities

https://en.wikipedia.org/wiki/Python_(programming_language)

Access and Run Python

Platform/Tool	Type	Key Features
Jupyter Notebook	Interactive IDE	Web-based, cell-by-cell execution, great for data science & visualization
Google Colab	Cloud IDE	Free GPU/TPU, integrates with Google Drive, ideal for ML & collaboration
Anaconda Navigator	Local Suite	GUI launcher for Jupyter, Spyder, VS Code; includes package management
Spyder	Scientific IDE	MATLAB-like interface, variable explorer, great for engineering workflows
VS Code	Code Editor	Lightweight, extensible, Python plugin support, Git integration
PyCharm	Full IDE	Powerful debugging, refactoring, and project management
Cursor	AI-powered Editor	Copilot-style suggestions, fast navigation, built for Python productivity
Terminal/Command Line	Native Execution	Run .py files directly using python script.py, ideal for automation
Online Interpreters	Web-based	Platforms like Replit, Trinket, or PythonAnywhere for quick code testing

Python Kick Start Program

Python Basics

1. Print('hello'), Print(), /n
2. A=input('enter A=')
3. String to Numeric
4. String Count
5. Sentence Upper, Lower, Capitalize
6. Date and Time
7. Conditionals (if then elif)
8. Collections: Lists/arrays, dictionaries
9. Loops
10. Functions, Lambda
11. Managing the file system

Essential Libraries

1. OS
2. Numpy
3. Pandas
4. Matplotlib
5. Seaborn
6. Scipy
7. Scikit Learn (sklearn)
8. LangChain

OS (Library)

Standard Python module for interacting with the operating system, including file and directory operations.

- **Sample Code:**

```
import os
print(os.getcwd())  # Current working directory
os.mkdir("new_folder")
```

- **Official Site:** docs.python.org - os module

NumPy (Library)

- Fundamental package for numerical computing in Python.
- It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions.

Sample Code:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
print(np.mean(a))  # Output: 2.5
```

- **Official Site:** numpy.org

Pandas (Library)

What is pandas?

- Open source/BSD-licensed library
- High performance data analysis tools
- https://pandas/pydata.org

Create a Data Frame

```
import pandas as pd
OilCo = pd.DataFrame([
    ['XOM','Irvin','USA'],
    ['Oxy','Houston','USA'],
    ['BP','London','United Kingdom'],
    ['Shell','The Haghe','Netherlands']],
     columns= ['Company','City','Country'])
```

OilCo.**head(n)**- first *n* rows
OilCo.**tail(n)**- last *n* rows
OilCo.**shape** - dimensions
OilCo.**info** – detailed information
OilCo.**describe()** – detailed information

Familiar data structures and tools

Similar to a table in a database or spreadsheet

- Columns identified by name
- Rows of data
- Index column

Data Table

index	Name	City	Country
0	XOM	Irving	USA
1	Oxy	Houston	USA
2	BP	London	United Kingdom
3	Shell	The Hague	Netherlands

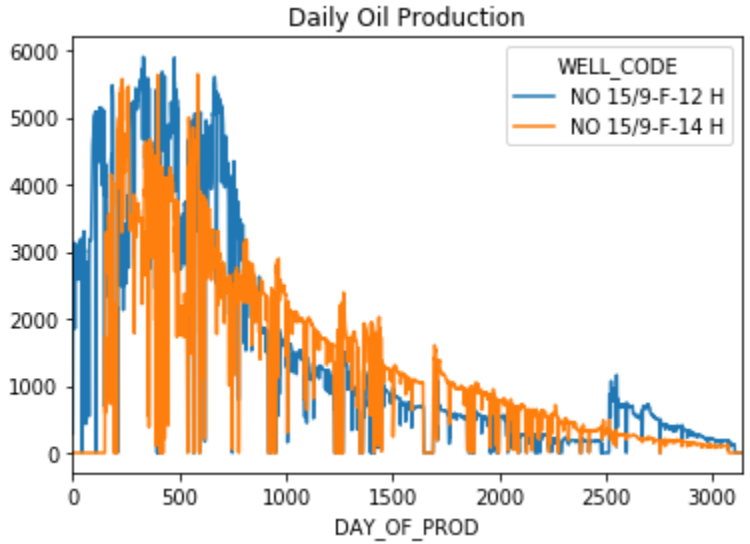
Read data from external csv files

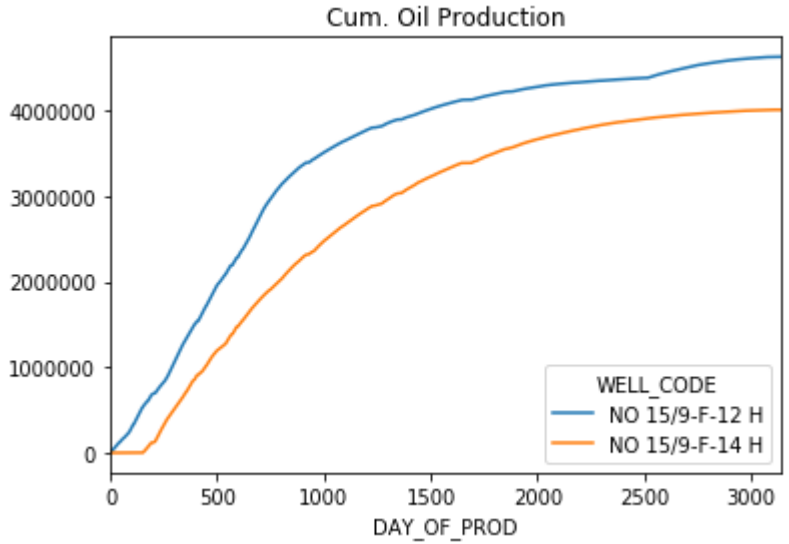
```
OilCo = pd.read_csv('OilCompanies.csv')
```

Pandas Plot

Create a Data Frame

```
import pandas as pd
df = pd.read_csv('../Volve_Oil_Production.csv')
well_codes = df.WELL_CODE.unique()
df = df.pivot(index='DAY_OF_PROD', columns='WELL_CODE', values=['OIL_PROD_VOL', 'CUM_OIL_PROD'])
df.plot(y='OIL_PROD_VOL', title='Daily Oil Production')
df.plot(y='CUM_OIL_PROD', title='Cum. Oil Production')
```





https://www.equinor.com/en/what-we-do/digitalisation-in-our-dna/volve-field-data-village-download.html

SciPy (Library)

Builds on NumPy to provide advanced scientific computing tools like **optimization, integration, interpolation, and statistics**.

- **Sample Code:**

```
from scipy import stats
data = [1, 2, 3, 4, 5]
print(stats.zscore(data))
```

- **Official Site:** scipy.org

$$f(x, y, z) = (x - 1)^2 + (y - 2)^2 + (z - 3)^2$$

```
from scipy.optimize import minimize
import numpy as np
# 1. Define the Objective Function
def objective_function(vars_xyz):
    x, y, z = vars_xyz
    return (x - 1)**2 + (y - 2)**2 + (z - 3)**2
# 2. Define the Constraints
bounds = [(0, 5), (0, 5), (0, 5)] # 0 <= x,y,z <= 5
constraints = [{'type': 'ineq', 'fun': lambda vars_xyz: 10 -
          (vars_xyz[0] + vars_xyz[1] + vars_xyz[2])}]
# 3. Set an Initial Guess
initial_guess = np.array([0, 0, 0])
# 4. Perform the Optimization
result = minimize(objective_function, initial_guess, method ='SLSQP',
                  bounds=bounds, constraints=constraints)
# 5. Print the Results
if result.success:
    print(f"Optimal values for x, y, z: {result.x}")
else:
    print("Optimization failed.")
```

Scikit-learn (sklearn) (Library)

- Machine learning library offering tools for classification, regression, clustering, and model evaluation.

- **Sample Code:**

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit([[1], [2], [3]], [2, 4, 6])
print(model.predict([[4]]))  # Output: [8.]
```

- **Official Site:** scikit-learn.org

Matplotlib (Library)

Comprehensive library for creating static, animated, and interactive visualizations.

- **Sample Code:**

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Simple Plot")
plt.show()
```

- **Official Site:** matplotlib.org

Seaborn (Library)

High-level interface for drawing attractive statistical graphics, built on top of Matplotlib and tightly integrated with Pandas.

- **Sample Code:**

```
import seaborn as sns
sns.histplot([1, 2, 2, 3, 3, 3, 4])
```

- **Official Site:** seaborn.pydata.org

LangChain (Library)

Framework for building applications powered by large language models (LLMs), including agents, chains, and retrieval-augmented generation.

- **Sample Code:**

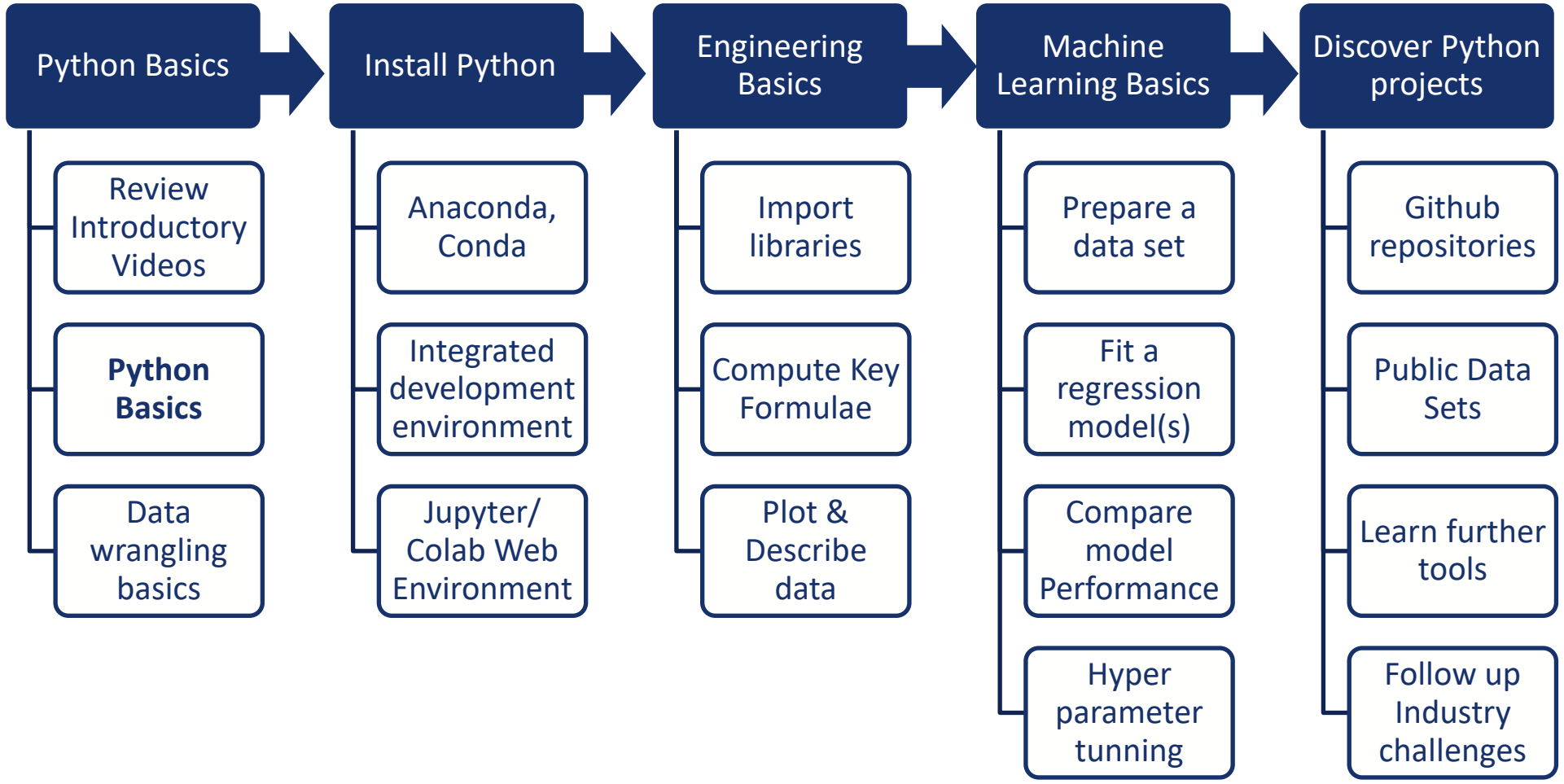
```
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI()
response = llm.predict("What is LangChain?")
print(response)
```

- **Official Site:** python.langchain.com

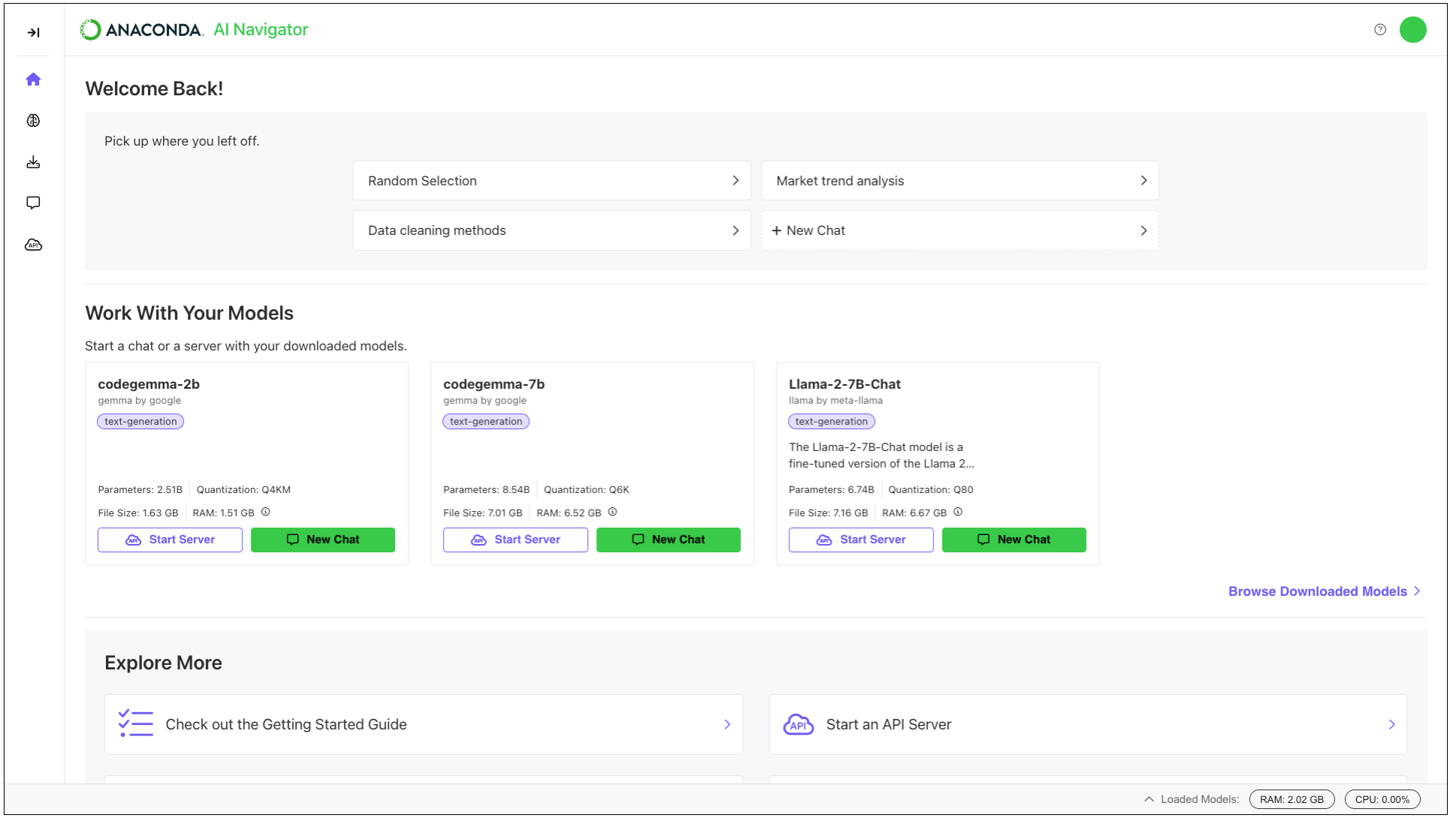
Python Advanced: Use LLM to Retrieve data from PDF

```
# 1. Load your model (e.g., using Ollama with Mistral or LLaMA)
from langchain.chat_models import ChatOllama
llm = ChatOllama(model="mistral")  # Requires Ollama installed
# 2. Extract text from PDF
from langchain.document_loaders import PyMuPDFLoader
loader = PyMuPDFLoader("your_file.pdf")
documents = loader.load()
# 3. Split documents into chunks
from langchain.text_splitter import RecursiveCharacterTextSplitter
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
docs = splitter.split_documents(documents)
# 4. Vectorize and store using FAISS
from langchain.vectorstores import FAISS
from langchain.embeddings import OllamaEmbeddings
embeddings = OllamaEmbeddings(model="mistral")  # lightweight embeddings
vectorstore = FAISS.from_documents(docs, embeddings)
# 5. Set up the retriever
retriever = vectorstore.as_retriever()
# 6. Create a retrieval chain
from langchain.chains import RetrievalQA
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)
# 7. Ask a question!
query = "What are the key topics covered in this PDF?"
answer = qa_chain.run(query)
print(answer)
```

A Roadmap to Kick Start in Python



AI Models from Anaconda Navigator



SPE AI Machine Learning for Energy Professionals Boot Camp

Session 3: Use Cases & Hands-On Implementation

Agenda

- **Session 1: Introduction to AI and Machine Learning**
  - Fundamentals of AI and machine learning
  - Overview of key algorithms and their relevance to energy workflows
  - Resources, communities, and learning paths for continued development
- **Session 2: Data Processing & Python for Energy Analytics**
  - Data collection, cleaning, and transformation
  - Introduction to Python and essential libraries
- **Session 3: Use Cases & Hands-On Implementation**
  - Upstream, Midstream & Downstream
  - Enterprise & Cross-Functional
- **Way Forward**

Upstream Examples

- Predicting shut-in wells using performance classification
- Virtual metering from real-time production data
- Log permeability prediction from core data
- Saturation and pressure map generation from well logs
- Seismic feature detection using pattern recognition.
- Drilling ROP (Rate of Penetration) Prediction

Predicting Shut-in Wells

Likelihood that an oil well's output will decline below economic thresholds due to reservoir depletion, pressure loss, or operational constraints

- **Main Challenge**
  - **Uncertainty in decline behavior**: Caused by complex reservoir heterogeneity, limited production history, and external interventions (e.g., stimulation, shut-ins).
  - Traditional DCA (e.g., Arps models) often **overestimate reserves** or fail in unconventional plays.

- **ML & AI-Based Solutions**
  - **Bayesian Neural Networks**: Quantify epistemic and aleatory uncertainty in production forecasts.
  - **Ensemble ML models**: Combine multiple learners (e.g., RF, XGBoost, GRU) for robust decline prediction.
  - **Clustering + Transfer Learning**: Group wells by production profiles and apply learned decline patterns to new wells.

- **Top 3 Use Cases**
  - **Probabilistic Forecasting of EUR & Decline Rates**
    - ML models predict future production with confidence intervals (P10–P90).
    - MIT Thesis on Probabilistic Forecasting
  - **Sweet Spot Identification via Unsupervised Learning**
    - Gaussian Mixture Models + BNNs to map high-yield zones.
    - AI-Driven Sweet Spot Mapping
  - **Production Profile Clustering for Analog-Based Forecasting**
    - K-means clustering of wells with similar decline behavior.
    - ADIPEC Case Study on AI Clustering

Probability of future diminished production

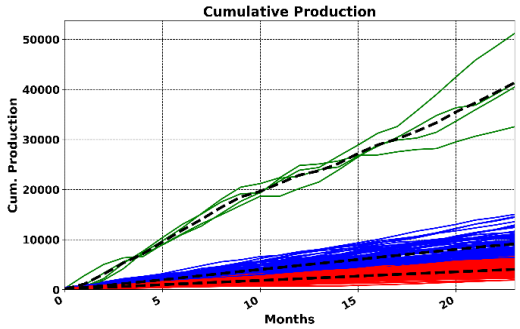
Daily Oil Production Data for many wells in a field or reservoir

Select Wells and time ranges for the analysis (e.g. 0 to 24 month and end of history status)

Well Status at the end of history, i.e. inactive wells, or Rate decline after 60 Months

Associate wells in each cluster and the well status at the end of history

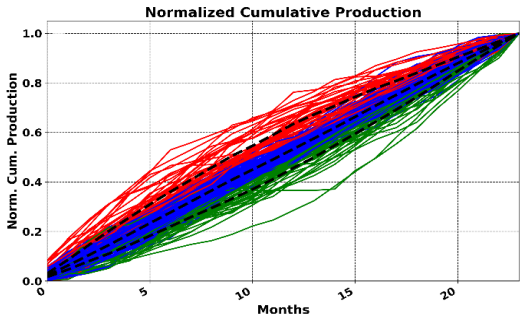
Probability of No Oil Production by Well after N Years later



Cumulative Oil Production Data

Cumulative Oil Production Clusters (e.g. Low, Medium, High Cumulative)

Combined Clusters generate 9 distinct cumulative-production categories → new data features



Normalized Cumulative Oil Production Data

Normalized Cumulative Oil Production Clusters (e.g. Decline, Steady, Increase)

https://github.com/Apress/machine-learning-oil-gas-industry

Production Rate → / Cumulative Production ↓	Declining Rate	Steady Rate	Increasing Rate
Low	0.357	0.258	0.308
Medium	0.19	0.12	0.138
High	0.000	0.000	0.000
Average WCT & Dead Well Probability Correlation **Coefficient: 0.719561**			

Virtual Metering

Estimating oil, gas, and water flow rates from wells using indirect measurements (e.g., pressure, temperature) and data-driven models, rather than physical multiphase flow meters or test separators

- **Main Challenge**
  - **Sparse and noisy data** between well tests.
  - **Complex flow regimes** and changing well conditions (GOR, WCT, reservoir depletion).
  - **High cost and limited availability** of physical multiphase meters.
- **ML & AI-Based Solutions**
  - **Multidimensional ML models** (ANN, GRU, XGBoost) trained on historical sensor & well test data to predict flow rates in real time.
  - **Closed-loop systems** integrating physics-based models with AI for dynamic calibration.
  - **Ensemble learning** to improve robustness across varying well conditions.
- **Use Cases**
  - **Real-time flow rate estimation**
    - Predict oil/gas/water rates using wellhead sensors and ML models.
    - Aramco's study on ANN, GRU, XGBoost
  - **Back-allocation of production**
    - Distribute total field production to individual wells using AI models.
    - TNO's Deep Learning Optimization for Gas Wells
  - **Intermittent production optimization**
    - Forecast optimal shut-in/start-up cycles using RNNs and optimization algorithms.
    - ALRDC Workshop Presentation

Case Study: Data-driven Virtual Metering

$$\hat{Y} = \varphi(X) = Y_m + \varepsilon$$

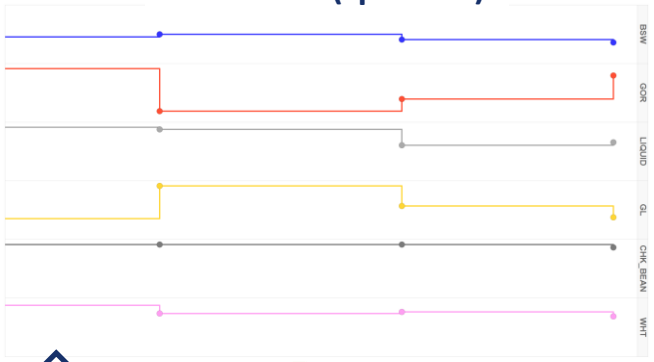
$$Liquid = f(BHP, WHP, WHT, \mathrm{GL})$$

$\hat{Y}$ is the virtual metering approximation function,
φ is the transfer function between measurements
X is the measurements input vector
Y_m is the labelled data (Well tests or measured pressure)
ε is the error

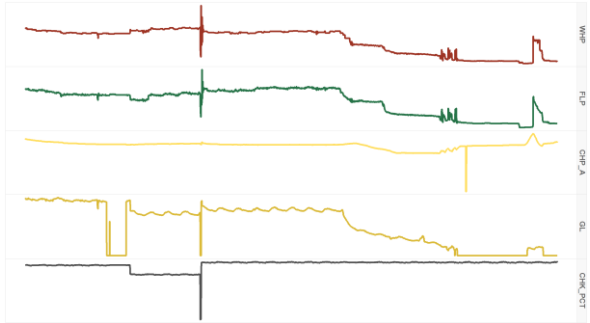
The objective of the data scientist is to find a suitable function φ including its parameters, that minimizes ε with reasonable amount of resources.

Gas-lift well Virtual Meter: How it works?

Flow Tests (sporadic)



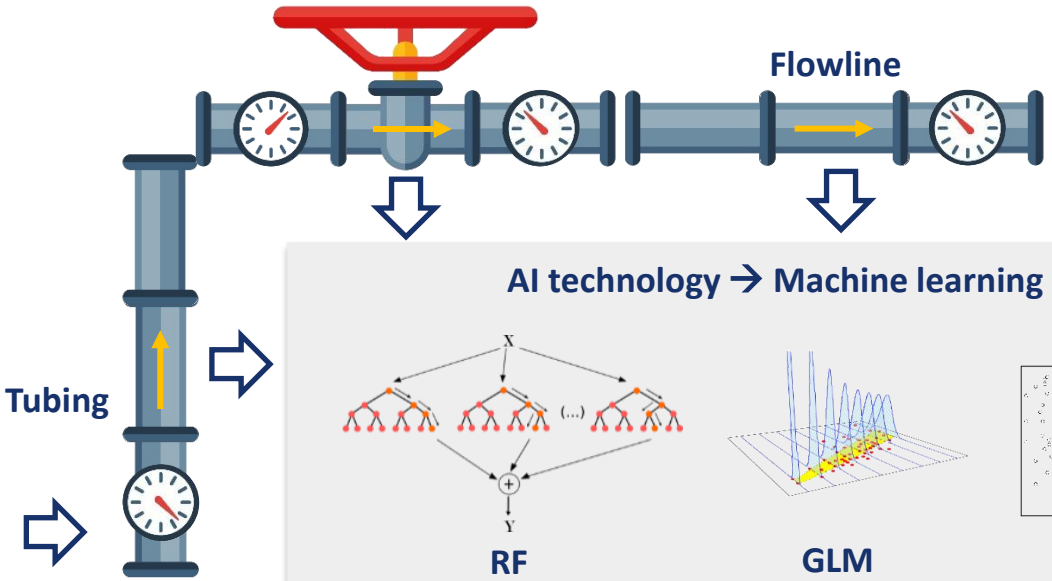
Pressures and Temperatures (real time)



Choke

Flowline

Tubing

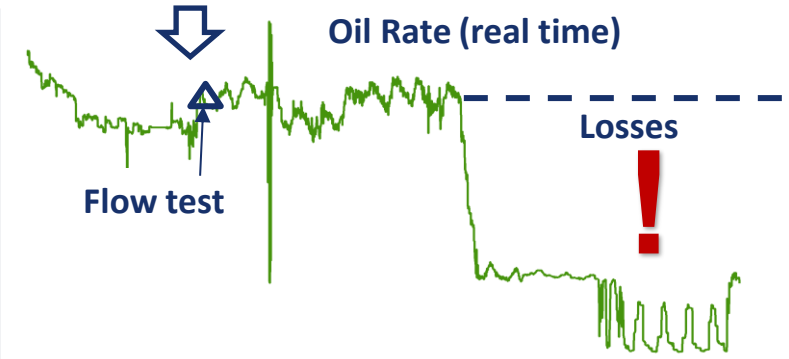


AI technology → Machine learning models

RF GLM SVR

SPE-201696-MS

Oil Rate (real time)



Machine Learning Basics: Virtual Meter (1/3)

Given a set of Rate Measurements

Qoil	Qwater	Qgas	BHP	WHP	WHT	Tsep	Psep	Choke_in
954.6	0	2.39	5410.33	3185.75	83.3	60.32	100	0.25
801.9	200.5	2.01	5388	3015.38	86.88	60.65	100	0.25
634.7	423.2	1.59	5391.13	2808.8	90.83	61.19	100	0.25
448.1	672.2	1.12	5405.72	2515.99	95.11	61.97	100	0.25
238.9	955.7	0.6	5433.8	2059.65	99.88	63.1	100	0.25

Create a Data Frame

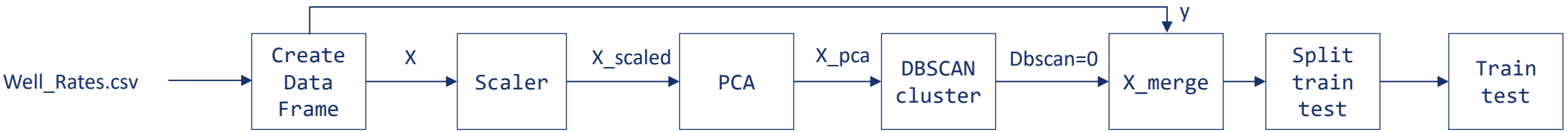
Define input and target

Split data for creating models

```
Read data
import os
import pandas as pd
cwd = os.path.dirname(os.path.abspath(__file__))
df = pd.read_csv( cwd + "..\Well_Rates.csv" )

# Create input feature and target
X = df[["BHP", "WHP", "WHT", "Tsep", "Psep", "Choke_in"]]
y = np.array(df[["Qoil"]].values).reshape(-1, )

# Use train_test_split from scikit-learn
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.9, random_state=123)
```

https://github.com/Apress/machine-learning-oil-gas-industry

Machine Learning Basics: Virtual Meter (2/3)




```
scaler = MinMaxScaler() ; X_scaled = scaler.fit_transform(X) # Scale Data
pca = PCA()
X_pca = pca.fit_transform(X_scaled) # Perform PCA
pca_df = pd.DataFrame(X_pca)

clustering = DBSCAN(eps=0.5, min_samples=12).fit(X_pca) # Cluster using DBSCAN
pca_df["DBSCAN"] = clustering.labels_ + 1
clust_df = pca_df[(pca_df["DBSCAN"] == 1)]  # Keep only the rows where DBSCAN=0
X_merge = X.merge(pca_df,left_index=True,right_index=True) # Split data
X_merge["Labelled Y"] = y

train_X, test_X, train_y, test_y = train_test_split(X.iloc[clust_df.index.tolist(),:],
y[clust_df.index.tolist()], test_size=0.5, random_state=123)
```

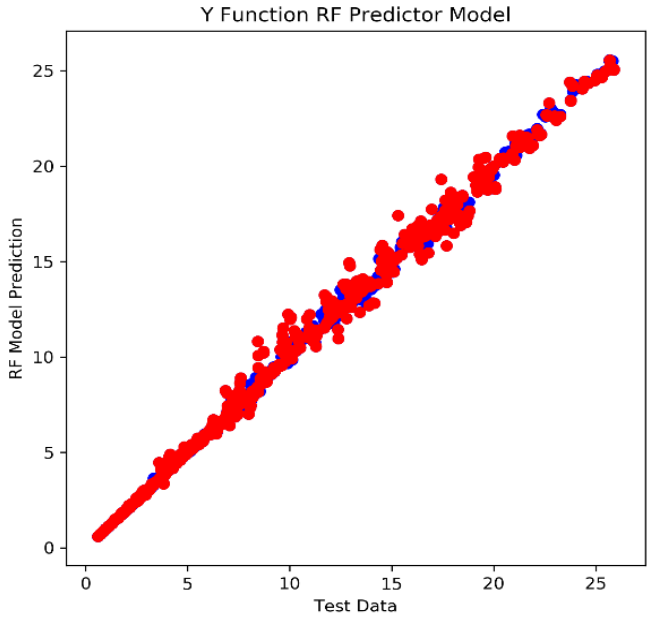
DBSCAN: Density-Based Spatial Clustering of Applications with Noise

https://github.com/Apress/machine-learning-oil-gas-industry

Machine Learning Basics: Virtual Meter (3/3)

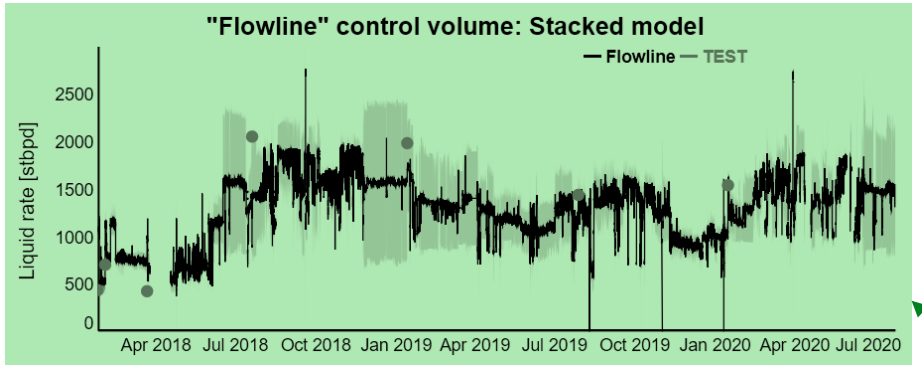
```
# Build Random Forestestimator:
model_RF = RF(n_estimators=200, random_state=123) # RF Regression
model_RF.fit(train_X, train_y)

# Predict for training and validation input sets:
pred_y_RF_test = model_RF.predict(test_X) # Blind test
pred_y_RF_train = model_RF.predict(train_X)
pred_y_RF_all = model_RF.predict(X.iloc[clust_df.index.tolist(),:])
r2_RF_train = r2_score(train_y, pred_y_RF_train) # Compute r2 scores
r2_RF_test = r2_score(test_y, pred_y_RF_test)
r2_RF_all = r2_score(y[clust_df.index.tolist()], pred_y_RF_all)
```

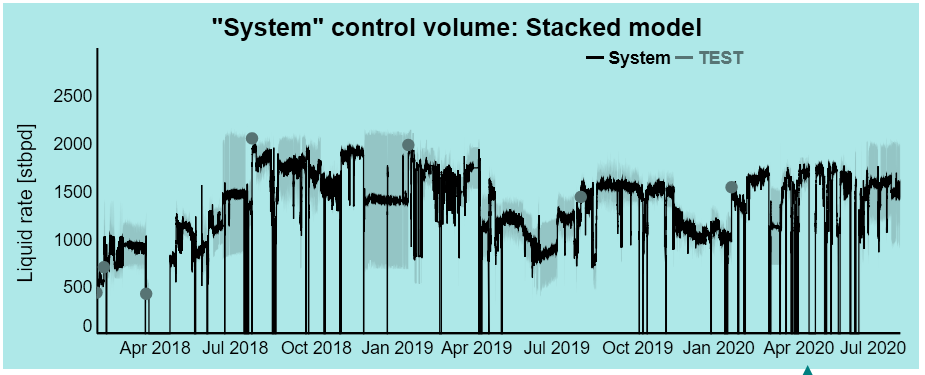
```
R^2 for train data Random Forest is 0.999169.
R^2 for blind test data Random Forest is 0.995007.
R^2 for all data Random Forest is 0.997096.
```

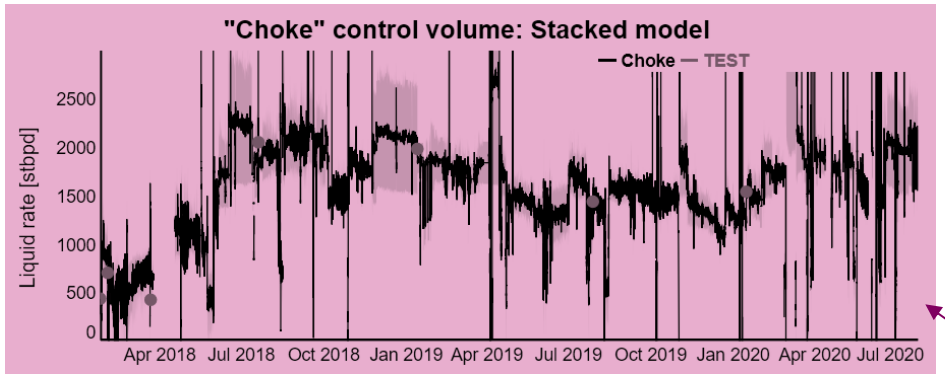


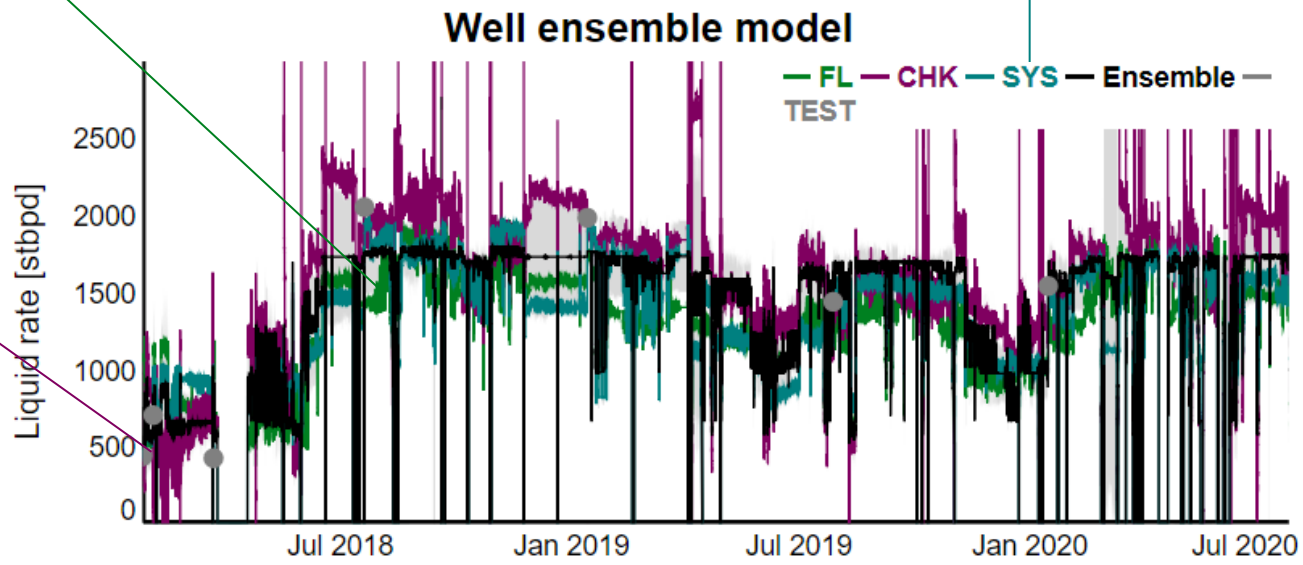
https://github.com/Apress/machine-learning-oil-gas-industry

Gas-Lift Well Virtual Meter - Ensemble model results









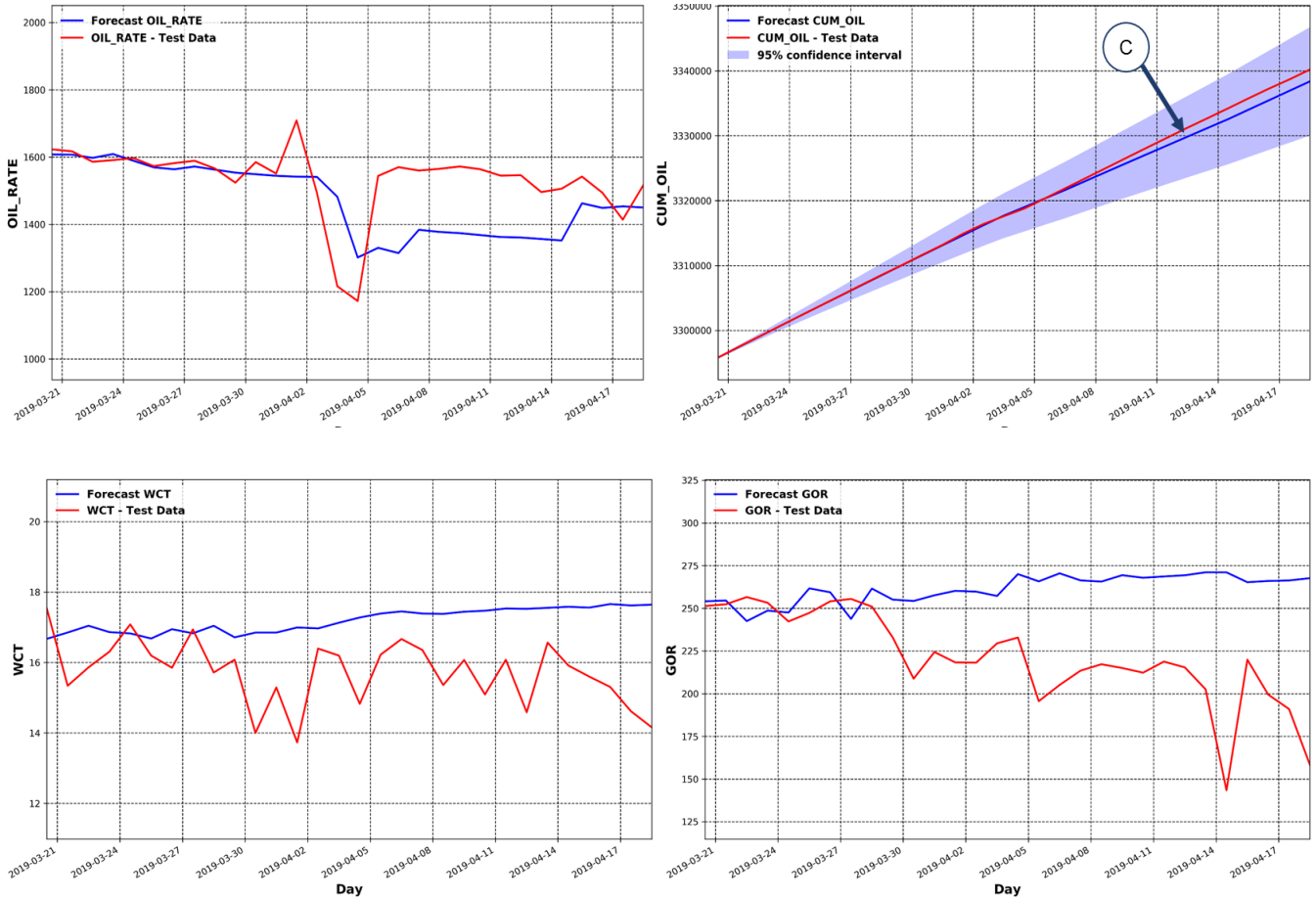
SPE-201696-MS

Gas-lift well Data Driven Model for Performance Forecasting

i. ARIMAX models order (14, 1, 0):
   i. 14 days auto-regression,
   ii. 1st order difference,
   iii. no moving average

ii. Choke, and gas-lift rates used as exogenous variables.

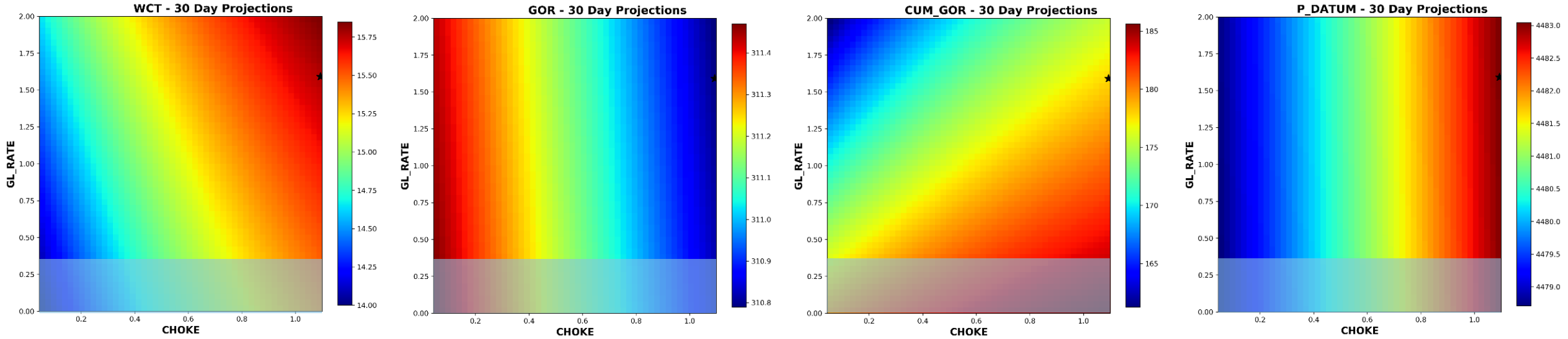
iii. Cumulative oil, cumulative water, and cumulative gas were modeled using ARIMAX.



SPE-201696-MS

Gas-lift well Data Driven Choke vs Gas-lift rate sensitivity analysis

Sensitivity analysis results showing 2D response surfaces for different variables including 30-day forecast of WCT, GOR, Cumulative GOR and Reservoir pressure.



"What-If" scenarios useful to answer questions such as:

- What would be the production if the well is operated at 100% choke open?
- What would be the production if gas injection was started for the well in question at a certain point of time?

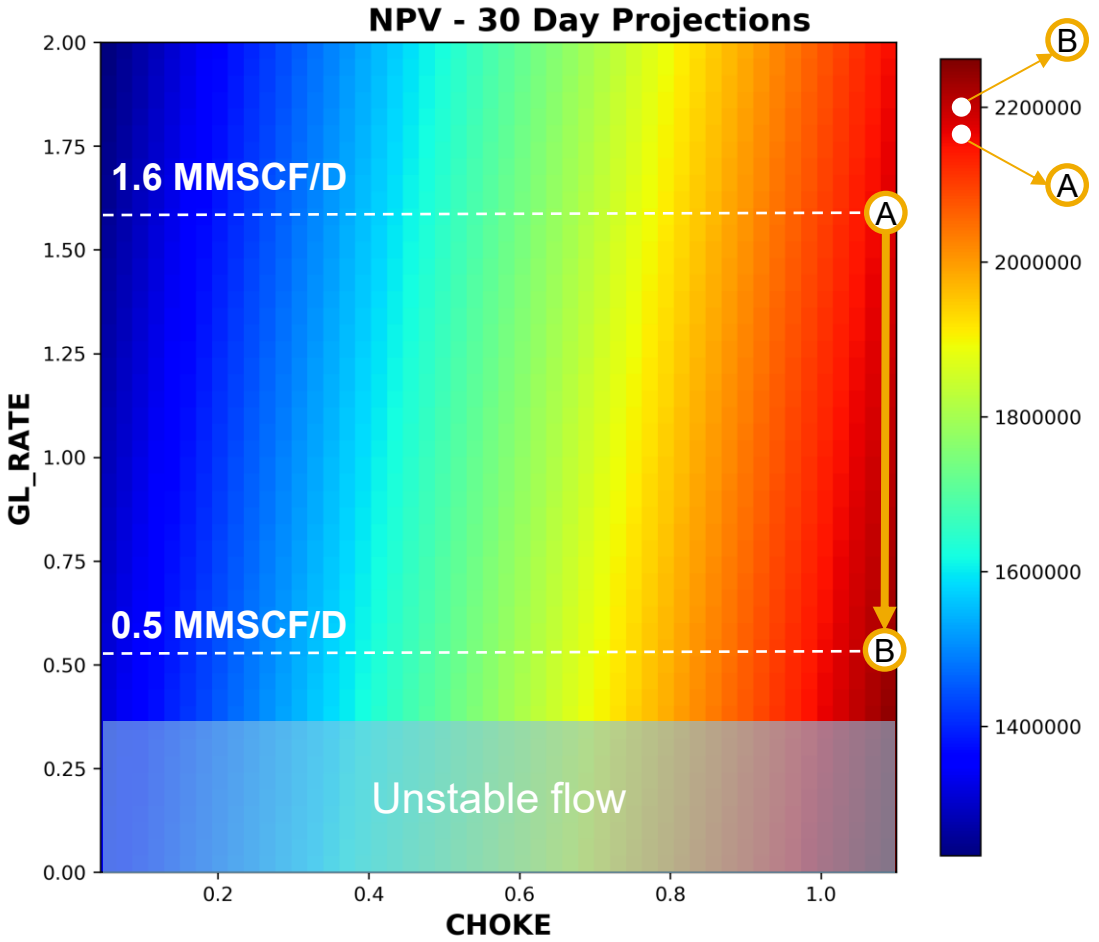
SPE-201696-MS

Gas Lift Well 30-day NPV Forecast and Optimization

- A simple NPV formula used

$$NPV = 50 * Np + 2500 * Gp - (500 * Gi_{gl} + 0.5 * Wp)$$

- Sensitivity analysis results showing 2D response surfaces for different variables.
- Shaded region is potential zone of unstable operations. Star shows current operating settings.
- Analysis suggests that the well is operating in near optimal zone.
- Slight decrease in the gas-injection rate increases NPV by ~0.05 MM$ in 30 days, or +2.5% in net profit gain.



SPE-201696-MS

Log Permeability Prediction From Core Data

Generating a continuous permeability log (k) by learning the relationship between conventional well logs and core-measured permeability data.

- **Key Challenges**
  - Limited core data coverage across wells and depths
  - Scale mismatch between core measurements and log resolution
  - Heterogeneity in lithology affecting correlation robustness

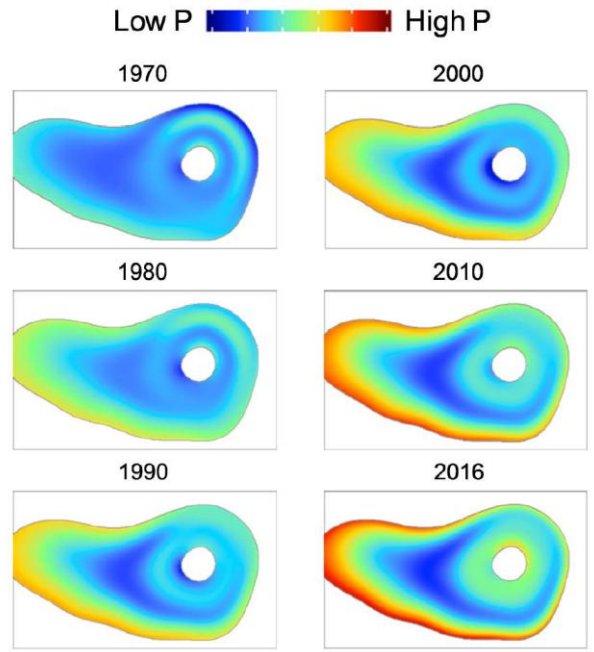
- **Solutions**
  - **Empirical Models**: Use porosity–permeability transforms (e.g., Buckles' method)
  - **Machine Learning**: Train supervised models (e.g., Random Forest, ANN) on core-log pairs
  - **Data Preprocessing**: Upscale core data or filter incompatible samples for better log alignment

- **Example Use Cases**
  - Predict permeability in uncored intervals to support facies modeling. Assam-Arakan Basin
  - **Digital Core Upscaling**
    Generate continuous permeability curves from sparse plug data SubsurfaceAI Preprocessing workflow
  - **AI-Driven Field Development**
    Apply ML models to optimize well placement and completions. GitHub ML workflow – Volve dataset
  - **Tight Reservoir Characterization**
    Apply DNNs to predict permeability in complex lithologies (e.g., Ordos Basin) DNN in Tight Reservoirs
  - **Ensemble Learning for Field Development**
    Use AdaBoost and stacking models to upscale sparse core data Springer – AdaBoost Ensemble Study

Saturation and Pressure Map Generation from Well Logs

- **Key Challenge**
  - Sparse, irregularly sampled data across space and time
  - Non-stationary behavior of saturation and pressure due to dynamic reservoir conditions
  - High-dimensional correlations between logs, pressure, and geological features
- **Solution Approach**
  - Multidimensional Kriging:
    - Handles spatial and temporal interpolation using variograms and drift terms
    - Supports universal kriging with external drift (e.g., porosity, facies)
    - Tools: PyKrige documentation
  - Machine Learning & AI:
    - Ensemble models (e.g., Random Forest, XGBoost) capture nonlinear trends
    - Deep learning (e.g., LSTM, CNN) for temporal forecasting and uncertainty quantification
    - Hybrid models combine kriging with ML for improved accuracy

Use Case	Description	Source
4D Seismic Pressure-Saturation Inversion	Predict fluid fronts and pressure changes using DNNs	GitHub: 4D Seismic Neural Inversion
Reservoir Monitoring via Multi-Vintage Kriging	Time-lapse saturation/pressure mapping using 5D ensembles	Sharp Reflections Whitepaper
Hybrid Kriging-ML for Geological Attribute Estimation	Combines kriging variance with ML ensemble weighting	Springer Study



Drilling ROP (Rate of Penetration) Prediction

- **Main Challenge**
  - **Nonlinear, multivariate dependencies**: ROP is influenced by lithology, bit type, WOB, RPM, mud properties, hydraulics, and downhole dynamics.
  - **Traditional models (e.g., Bourgoyne & Young)** lack generalizability across formations and drilling environments.
- **ML & AI-Based Solutions**
  - **Multidimensional regression** (ANN, SVM, RF) trained on real-time surface and downhole data.
  - **Hybrid frameworks** combining physics-based models with ML for better interpretability and generalization.
  - **Metaheuristic optimization** (PSO, GA, DE) to fine-tune drilling parameters (WOB, RPM) for maximum ROP while minimizing vibration and tool wear.

- **Use Cases**
  - **Real-Time ROP Prediction & Advisory Systems**
    - ML models integrated with rig sensors to forecast ROP and suggest optimal setpoints.
    - AI-Driven Optimization of Drilling Performance
  - **Torque-Vibration Mitigation via ML-Enhanced Control**
    - Dual-objective optimization: maximize ROP while minimizing stick-slip and torsional vibrations.
    - Real-Time Drilling Optimization with Vibration Control
  - **Global ROP Models for Multi-Well Deployment**
    - Transferable ML models trained across diverse lithologies and BHA configurations.
    - Ensemble ML for ROP Modeling in Iraqi Fields

Seismic Feature Detection using Pattern Recognition.

ML/AI-based classification and clustering of seismic attributes to reveal geologic patterns not easily visible to human interpreters

- **Main Challenge**
  - **High-dimensional, noisy data**: Seismic volumes contain terabytes of multi-attribute data with overlapping features and low signal-to-noise ratios.
  - **Subjectivity in interpretation**: Manual picking is time-consuming and prone to bias, especially in complex geological settings.

- **ML & AI-Based Solutions**
  - **Unsupervised learning** (e.g., Self-Organizing Maps, PCA) to reduce dimensionality and reveal natural clusters.
  - **Supervised CNNs & RNNs** for fault, salt dome, and facies classification from labeled seismic volumes.
  - **Transfer learning & fine-tuning** to adapt pretrained models across basins and acquisition geometries.

- **Use Cases**
  - **Fault & Fracture Detection via CNNs**
    - Deep learning models trained on labeled seismic slices to detect discontinuities.
    - Seismic-phase detection using multiple deep learning models
  - **Seismic Facies Classification with SOM + PCA**
    - Multi-attribute clustering to identify depositional environments and sweet spots.
    - Geologic Pattern Recognition from Seismic Attributes
  - **Salt Dome & Geobody Recognition using GANs**
    - Synthetic image generation and segmentation for complex structures.
    - AI-Powered Seismic Data Interpretation Services

Midstream Examples

- Pipeline leak detection and predictive maintenance using sensor data.
- Flow rate forecasting in transportation networks
- Anomaly detection in SCADA systems for asset integrity
- Predictive corrosion modeling in pipelines
- Combined Cycle Power Plant: Electricity Prediction

Pipeline Leak Detection And Predictive Maintenance

Identifying unintended fluid escapes using internal (flow/pressure) or external (acoustic/fiber optic) methods.

- **Main Challenge**
  - **Sparse, noisy, and high-dimensional data** from distributed sensors.
  - Difficulty in detecting **small or gradual leaks** and predicting failures in aging infrastructure.
  - **False positives** and delayed detection in remote or buried segments.

- **ML & AI-Based Solutions**
  - **Ensemble ML models** (Random Forest, XGBoost, LSTM) for anomaly detection and leak localization.
  - **Hybrid physics-AI frameworks** combining fluid dynamics with ML for interpretable diagnostics.
  - **IoT + AI integration** for real-time monitoring and predictive analytics across vast pipeline networks.

- **Top 3 Use Cases**
  - **Real-Time Leak Detection via Acoustic & Pressure Sensors**
    - ML models analyze sensor streams to detect leaks with >95% accuracy.
    - Springer study on simulation-driven diagnostics
  - **Predictive Maintenance Forecasting with Time-Series Models**
    - Prophet + Random Forest used to predict corrosion and failure hotspots.
    - Science Publishing Group case study
  - **Anomaly Detection in Underground Pipelines Using Smart Sensors**
    - LSTM and CNN models process acoustic, temperature, and vibration data.
    - EJAET smart sensing framework

Flow Rate Forecasting In Transportation Networks

Predicting the volume of vehicles or passengers passing through a network segment per unit time (e.g., vehicles/hour).

- **Main Challenge**
  - **Spatio-temporal complexity**: Traffic flow is influenced by dynamic factors like weather, events, road conditions, and human behavior.
  - **Data sparsity & heterogeneity**: Incomplete sensor coverage, noisy GPS data, and inconsistent formats across regions.
  - Critical for **traffic management**, **infrastructure planning**, and **congestion mitigation**
- **ML & AI-Based Solutions**
  - **Graph Neural Networks (GNNs)**: Model road networks as graphs to capture spatial dependencies.
  - **Transformer-based architectures**: Handle long-range temporal dependencies and multi-periodicity.
  - **Hybrid models**: Combine physics-informed PDEs with deep learning for interpretable and generalizable forecasting.
- **Use Cases**
  - **Urban Congestion Prediction & Signal Optimization**
    - Real-time traffic flow prediction for adaptive signal control.
    - GNN-based Traffic Forecasting Survey
  - **Public Transit Demand Forecasting**
    - Predict passenger flow across metro/bus networks using transformer-based models.
    - EMBSFormer for Traffic Flow Prediction
  - **Incident-Aware Traffic Management**
    - AI models detect anomalies and forecast flow disruptions before incidents are reported.
    - FHWA Predictive Traffic Management

Anomaly Detection in SCADA Systems For Asset Integrity

Identifying deviations from monitored assets ensures **early fault detection**, **cybersecurity**, and **process reliability** across industrial control systems.

- **Main Challenge**
  - **High-dimensional, heterogeneous data**: SCADA time-series, categorical, event-driven data across protocols.
  - **Sparse labeled anomalies**: Rare failure events make supervised learning difficult.
  - **Real-time constraints**: Detection must occur without disrupting operations.
- **ML & AI-Based Solutions**
  - **Unsupervised Learning**: Autoencoders, Isolation Forests, SVMs detect outliers without labeled data.
  - **Deep Temporal Models**: LSTM, GRU, and GPT capture long-term dependencies in sensor data.
  - **Explainable AI (XAI)**: SHAP and LIME provide interpretable insights for anomaly root cause analysis.
  - **Multispectral Data Acquisition**: Combine passive, active, and historian data for full asset visibility.
  - **Digital Twin Integration**: Simulate asset behavior to validate anomaly predictions.
- **Use Cases**
  - **Predictive Maintenance of Critical Assets**
    - Detect early signs of wear or failure in turbines, compressors, and pumps.
    - ABB Genix System Anomaly Detection App
  - **Cyber Intrusion Detection in SCADA Networks**
    - Identify unauthorized access or command injection via protocol-aware ML.
    - Springer SCADA Intrusion Detection Study
  - **Process Integrity Monitoring in Smart Grids**
    - Detect abnormal voltage, frequency, or load patterns using hybrid ML.
    - AI-Powered SCADA Sensor Networks

Predictive Corrosion Modeling In Pipelines

Forecasting corrosion rates and patterns in pipelines supports **asset integrity**, **risk mitigation**, and **maintenance optimization** in oil & gas infrastructure.

- **Main Challenge**
  - **Nonlinear, multivariable interactions**: chemistry, flow, temperature, pressure, and material properties.
  - **Sparse, noisy, and heterogeneous data**: Limited labeled data and inconsistent coverage hinder model generalization.

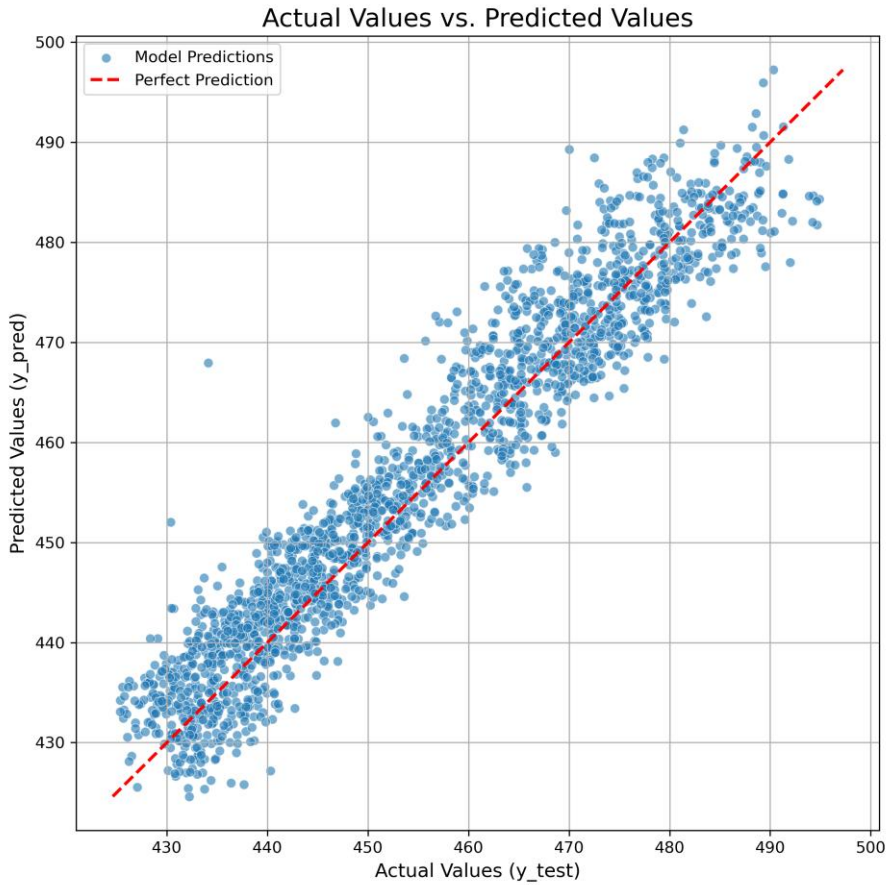
- **ML & AI-Based Solutions**
  - **Hybrid ML architectures**: Combine PCA, PSO, and BPNNs for noise reduction and nonlinear pattern learning.
  - **Transfer learning & domain adaptation**: Leverage labeled data from similar pipelines to improve predictions on new assets.
  - **Knowledge graph + neural networks**: Integrate domain knowledge with deep learning for contextual corrosion rate modeling.

- **Top 3 Use Cases**
  - **Corrosion Rate Forecasting in Natural Gas Pipelines**
    - Hybrid ML models (NLFE + NGO + ELM) outperform traditional methods.
    - Applied Sciences ML Corrosion Model
  - **External Corrosion Prediction Using Ensemble Learning**
    - XGBoost-based models trained on pipeline attributes and operating conditions.
    - Dynamic Risk ML Pipeline Study
  - **Knowledge Graph-Driven Corrosion Modeling**
    - BERT-BiLSTM-CRF + Neo4j graph database for contextual prediction.
    - MDPI Knowledge Graph Corrosion Model

Combined Cycle Power Plant: Electricity Prediction

Predicting power output enables smoother integration with national or regional grids

- **Main Challenge**
  - Nonlinear relationships between environmental variables and output
  - High sensitivity to ambient conditions and turbine dynamics
  - Difficulty in modeling transient behavior and load-following scenarios
- **ML & AI Solutions**
  - Regression Models, Deep Learning Transformer + DNN) & Optimization Algorithms for hyperparameter tuning
- **Top 3 Use Cases**
  - **Hourly Output Forecasting for Grid Optimization**
    Transformer-DNN models trained on 6-year CCPP datasets for full-load prediction MDPI Transformer Forecasting Model
  - **Operational Efficiency Enhancement via RNN Optimization**
    Recurrent Neural Networks tuned with metaheuristics to improve predictive accuracy Frontiers Energy Research – RNN + WWPA



Downstream Examples

- Demand forecasting and inventory optimization at refineries
- Quality prediction and control in process streams
- Energy consumption modeling and efficiency optimization in plant operations
- Scheduling and optimization of blending operations

Demand forecasting

Predicting future customer demand using historical data, market signals, and external variables, is critical for **inventory optimization**, **production planning**, and **supply chain resilience**.

- **Main Challenge**
  - **Volatile demand & fragmented data**: Rapid market shifts, seasonality, and siloed systems lead to inaccurate forecasts and costly misalignments.

- **ML & AI-Based Solutions**
  - **Time-series models + deep learning**: Capture trends, seasonality, and anomalies across SKUs and geographies.
  - **Generative AI + LLMs**: Integrate unstructured data (e.g., social sentiment, weather) for contextual forecasting.
  - **Reinforcement learning**: Optimize replenishment and safety stock dynamically based on forecast feedback loops.

- **Top 3 Use Cases**
  - **SKU-Level Forecasting for Retail & Distribution**
    - ML models reduce forecast error by 30–50% and lost sales by 65%.
    - Algonomy Inventory Forecasting Guide
  - **Multi-Echelon Inventory Optimization**
    - AI balances inventory across warehouses and stores using demand signals.
    - EY ML Forecasting Platform
  - **Real-Time Demand Sensing**
    - AI integrates POS, weather, and social data for dynamic adjustments.
    - RapidInnovation AI Forecasting Guide

Inventory Optimization

Strategic process to **balance inventory levels** across the supply chain while minimizing cost and maximizing service. Ensures **right product, right place, right time** — critical for refining, distribution, and retail operations.

- **Main Challenge**
  - **Demand variability + supply chain complexity**: Leads to overstock, stockouts, and working capital inefficiencies.
- **ML & AI-Based Solutions**
  - **Predictive analytics**: Forecast demand using historical, seasonal, and external data.
  - **Reinforcement learning**: Dynamically adjust reorder points and safety stock based on feedback loops.
  - **Deep learning models**: Capture nonlinear patterns across SKUs, geographies, and channels.
- **Use Cases**
  - **AI-Enhanced Inventory & Demand Forecasting**
    - Combines ML, NLP, and RPA for dynamic inventory control.
    - World Journal of Advanced Research and Reviews (2024)
  - **Gradient Boosting for Retail Inventory Optimization**
    - Predictive model improves stock levels and automates reordering.
    - European Journal of Advances in Engineering and Technology (2022)
  - **AI-Powered MRP in Industrial Manufacturing**
    - Automatic reorder point planning using ML algorithms.
    - International Journal of Engineering Trends and Technology (2023)

Quality Prediction and Control In Process Streams

Prediction of **product quality attributes** (e.g., sulfur content, distillation curves, UV transmittance) in **continuous process units** like **distillation towers**, **reactors**, and **fractionators e**nables **proactive control** to minimize off-spec production and optimize yield.

- **Main Challenge**
  - **Delayed or infrequent lab-based measurements** of critical quality parameters (e.g., ASTM distillation points, sulfur content, UV transmittance).
  - Leads to **off-spec production**, **energy inefficiencies**, and **safety risks**.

- **ML & AI-Based Solutions**
  - **Soft sensors**: ML models trained on real-time process data to estimate hard-to-measure quality variables.
  - **Deep learning**: Captures nonlinear dynamics in complex units (e.g., hydrotreaters, MEG distillation).
  - **Reinforcement learning**: Adjusts control parameters to maintain quality targets dynamically.

- **Use Cases**
  - **Crude Distillation Unit (CDU) Side-Stream Quality Prediction**
    - Predicts ASTM 95% distillation curve using ML models.
    - SimulateLive Soft Sensor Applications
  - **UV Transmittance Prediction in Monoethylene Glycol (MEG) Plants**
    - Soft sensors estimate UV quality online to reduce off-spec batches.
    - SimulateLive MEG Soft Sensor Case
  - **Sulfur Content Estimation in Diesel Hydrotreatment Units**
    - ML models predict sulfur levels in trickle-bed reactors for real-time control.
    - Springer Predictive Quality Review

Refinery Planning and Optimization

Optimize crude slate, unit throughput, blending, and product mix over 4–6 week horizon.

- **Key Variables**:
  - **Demand**: Forecasted product volumes (gasoline, diesel, jet, etc.)
  - **Supply**: Crude availability, external feedstocks, additives
  - **Plant Capacity**: CDU, FCC, HCU, reformer limits and constraints
  - **Yield**: Unit-specific product outputs, blending ratios, quality specs
- **Main Challenge**
  - **Nonlinear blending & inter-unit dependencies**: LP models oversimplify nonlinearities in blending, cut point shifts, and dynamic crude behavior.
  - **Slow response to market volatility**: Monthly LP cycles lack agility to adapt to intraday price swings or crude quality shifts.
- **ML & AI-Based Solutions**
  - **Hybrid modeling**: Combine LP with ML-based surrogate models for CDU, FCC, and blending units
  - **Reinforcement learning (RL)**: Decomposes plant-wide LP into submodels with dynamic pricing of intermediates
  - **Predictive analytics**: Forecast crude quality, equipment degradation, and demand variability to adjust LP constraints
- **Use Cases**
  - **RL-Driven Monthly Planning Optimization**
    - RL + model decomposition improves LP coordination and profitability
    - arXiv:2504.08642 – RL Refinery Planning
  - **Hybrid Modeling for CDU & FCC Yield Prediction**
    - AI + first-principles modeling enhances LP accuracy
    - DigitalRefining – AI in Refinery Modeling
  - **Predictive Analytics for Crude Quality & Equipment Health**
    - ML forecasts crude assay variability and unit degradation
    - Infosys White Paper – Dynamic Optimization

Energy Consumption Modeling And Efficiency Optimization

Align energy efficiency with **production throughput**, **quality**, and **sustainability goals**

- **Core Focus**
  - Quantify and optimize **energy use across equipment, processes, and utilities**
  - Model consumption patterns using **real-time sensor data**, **historical trends**, and **activity metrics**
- **Main Challenge**
  - **Fragmented data & nonlinear dependencies**: Traditional models fail to capture dynamic interactions between equipment, process states, and environmental factors
  - **Manual audits & static KPIs** miss transient inefficiencies and hidden energy drains
- **ML & AI-Based Solutions**
  - **Hybrid modeling**: Combine physics-based models with ML for accurate energy prediction across machines and systems
  - **Predictive analytics**: Forecast energy demand based on plant activity, weather, and production schedules
  - **Anomaly detection**: Identify abnormal consumption patterns using unsupervised learning and signal processing
- **Use Cases**
  - **AI-Driven Predictive Maintenance for Energy Efficiency**
    - Detect equipment degradation before it causes energy spikes
    - AIOT Insight – Predictive Maintenance in Plants
  - **Energy Forecasting Using Plant Activity Metrics**
    - ML models correlate SCADA data with energy usage for real-time forecasting
    - IEEE CSECS 2025 – Industrial Energy Forecasting
  - **Hybrid Intelligent Modeling for Process Optimization**
    - Combine ML (e.g., XGBoost, ANFIS) with optimization algorithms for multi-objective energy reduction
    - MDPI Energies – Hybrid Modeling Editorial

Scheduling and Optimization of Blending Operations

Coordinate **tank inventory**, **blending recipes**, and **dispatch timing** across CDU, FCC, reformers, and blend-shops

- **Core Focus**
  - **Daily scheduling** of blending operations to meet product specs (e.g. RON, sulfur, vapor pressure)
  - Optimize **plant capacity utilization**, **intermediate stream routing**, and **final product yields**
- **Main Challenge**
  - **Nonlinear blending behavior**: Octane, vapor pressure, and sulfur exhibit non-additive properties
  - **Real-time variability**: Crude assay shifts, unit upsets, and demand changes disrupt static schedules
  - **Manual recipe adjustments**: Lead to off-spec products, reblends, and margin loss
- **ML & AI-Based Solutions**
  - **Hybrid AI blending models**: Combine ANN + GA to predict nonlinear blend properties and optimize ratios
  - **Reinforcement learning (RL)**: Learns optimal scheduling policies under dynamic constraints
  - **Real-time anomaly detection**: Flags deviations in blend quality or tank levels using unsupervised ML
- **Use Cases**
  - **AI-Integrated Fuel Blending Optimization**
    - ANN + GA predicts RON/MON/AKI with high accuracy, reducing quality giveaway
    - MDPI ChemEngineering – AI Fuel Blending Case Study
  - **RL-Based Scheduling for Blend-Shops**
    - RL decomposes blend scheduling into subproblems, improving coordination and yield
    - MDPI Processes – Blend Scheduling Optimization
  - **AI-Driven Real-Time Optimization (RTO)**
    - Closed-loop AI adjusts blend ratios and dispatch timing based on market signals
    - Fidelis Associates – AI in Refinery Operations

Enterprise & Cross-Functional Examples

- Automated report generation from structured and unstructured data
- Document classification and Natural Language Programming for regulatory compliance
- AI-driven market analysis for crude/product pricing

Automated Report Generation from Structured and Unstructured data

Enables **real-time insights**, **decision support**, and **compliance tracking** across upstream, midstream, and downstream operations.

- **Definition**
  - Automated generation of **context-rich reports** (e.g., equipment status, rotating machinery performance, drilling logs) using **structured data** (SCADA, historians, CMMS) and **unstructured data** (PDFs, emails, logs, images, transcripts).
- **Main Challenge**
  - **Data fragmentation & overload**: Reports rely on siloed systems and inconsistent formats.
  - Difficulty in **prioritizing urgent vs. important** events due to lack of semantic context and real-time correlation.
- **ML & AI-Based Solutions**
  - **LLM-powered summarization**: Extract key insights from logs, emails, and free-text reports.
  - **RAG (Retrieval-Augmented Generation)**: Combines structured KPIs with unstructured narratives for contextual reporting.
  - **Anomaly detection + urgency scoring**: ML models rank events by severity, recurrence, and operational impact.
- **Use Cases**
  - **Rotating Equipment Performance Reports**
    - AI extracts vibration, temperature, and runtime anomalies from sensor logs + technician notes.
    - Springer Review on Unstructured Document Analysis
  - **Drilling Morning Reports (DMRs)**
    - NLP models summarize daily drilling activities, flag NPT events, and correlate with BHA and mud logs.
    - ArXiv RAG Experience Report from PDFs
  - **Operations & Asset Status Dashboards**
    - ML ranks equipment alerts by urgency using historical failure patterns and semantic context.
    - IJFMR LLM-Powered Report Generation Study
- 🧠 **Additional Key Aspects**
  - **Urgency Detection Frameworks**: Combine anomaly scores, recurrence, and asset criticality.
  - **Template-aware LLMs**: Generate reports in standard formats (e.g., DMR, shift logs, maintenance summaries).
  - **Human-in-the-loop validation**: SME feedback loops improve model accuracy and trust.

Document Classification and Natural Language Programming

classify, extract, and validate structured/unstructured documents for **regulatory adherence**.

- **Definition**
  - Use of **ML/NLP algorithms** to classify, extract, and validate structured/unstructured documents for **regulatory adherence**.
  - Applies to **project gates**, **equipment inspections**, **safety checks**, and **SOP compliance** across upstream, midstream, and downstream operations.

- **Main Challenge**
  - **Disparate formats & semantic ambiguity** across disciplines (engineering, HSE, drilling, operations).
  - Manual review is **slow**, **error-prone**, and lacks **traceability** for audits and cross-functional compliance.

- **ML & AI-Based Solutions**
  - **Transformer-based models (e.g., BERT, RoBERTa)** for semantic classification of technical and legal documents.
  - **RAG (Retrieval-Augmented Generation)** for contextual report synthesis from structured KPIs + unstructured logs.
  - **Urgency scoring models** rank anomalies by recurrence, severity, and asset criticality.

- **Use Cases**
  - **Drilling Morning Reports (DMRs) Compliance Extraction**
    - NLP models extract NPT events, correlate with BHA/mud logs, and flag regulatory gaps.
    - ACL 2025 – NLP in Regulatory Compliance
  - **Rotating Equipment Inspection Reports**
    - ML detects vibration/temperature anomalies and validates against inspection SOPs.
    - IJTPE 2025 – NLP for Document Verification
  - **Project Approval Gate Document Classification**
    - Transformer-based DocBERT models route documents to relevant departments for review.
    - Springer – Utility Document Classification

- **Additional Key Aspects**
  - **Ontology-based rule extraction**: Maps SOPs, inspection protocols, and regulatory clauses into machine-readable formats.
  - **Cross-discipline compliance detection**: NLP links engineering specs, HSE logs, and legal clauses for unified validation.
  - **Human-in-the-loop feedback**: SME validation improves model accuracy and trust.

AI-driven Market Analysis for Crude/Product Pricing

Analyze historical, real-time, and predictive data for **price forecasting**, **market opportunity detection**, and **risk mitigation**.

- **Main Challenge**
  - **Volatile, nonlinear market dynamics**: Driven by geopolitical events, macroeconomic indicators, supply-demand imbalances, and sentiment shifts.
  - Applies to **crude oil**, **refined products**, **petrochemicals**, and **energy derivatives**.
  - Traditional models (e.g., regression, LP) struggle with **multivariate complexity**, **data sparsity**, and **real-time responsiveness**.

- **ML & AI-Based Solutions**
  - **Deep learning models** (e.g., LSTM, GRU): Capture temporal dependencies in price movements.
  - **Reinforcement learning**: Optimize hedging and trading strategies under uncertainty.
  - **Transformer-based NLP models**: Extract sentiment and signals from news, social media, and analyst reports.
  - **Generative AI + RAG**: Synthesize structured and unstructured data for contextual pricing insights.

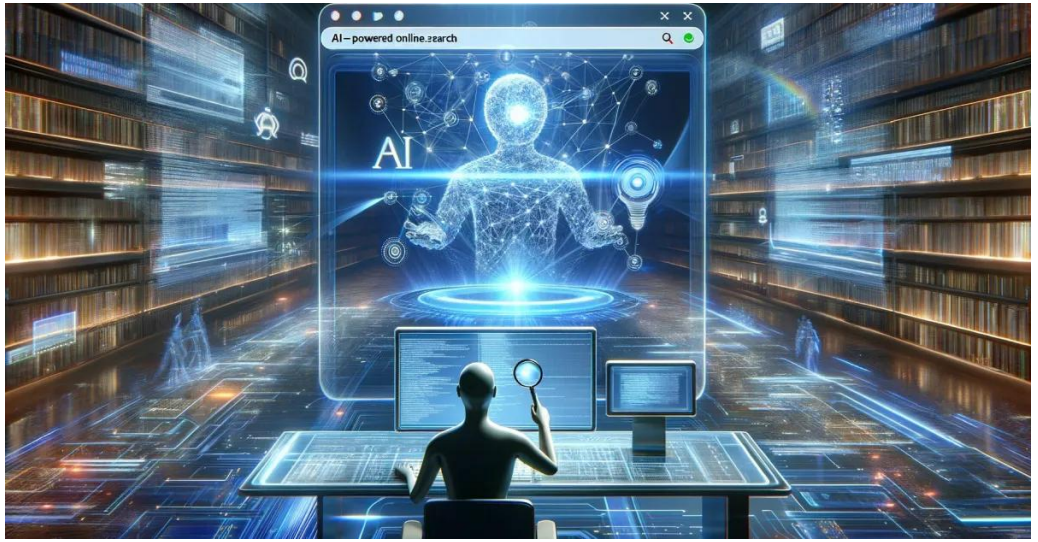
- **Use Cases**
  - **Crude Oil Price Forecasting Using Hybrid ML Models**
    - Combines CNN + LSTM + sentiment analysis for Brent/WTI prediction.
    - Journal of Applied Economics – AI & Dynamic Pricing Review
  - **Commodity Price Forecasting with Self-Learning AI**
    - SC Analytics & Thoucentric Labs use ML to predict daily/weekly crude prices.
    - Oil & Gas Middle East – AI Forecasting Case
  - **AI-Enhanced Revenue Optimization in Product Pricing**
    - ML models adapt to demand elasticity, competitor pricing, and customer behavior.
    - IJMRGE – AI in Pricing & Revenue Optimization
  - EY Whitepaper – The Art of Pricing in the Age of AI

- **Additional Key Aspects**
  - **Price elasticity modeling**: AI quantifies consumer response to price changes.
  - **Scenario simulation**: ML models test pricing strategies under macroeconomic shifts.
  - **Explainable AI (XAI)**: Enhances trust and interpretability for regulatory compliance.
  - **Integration with ERP & trading platforms**: Enables real-time pricing decisions.

AI-powered online search

AI is transforming online search, giving us personalized, contextual, and predictive experience

- AI algorithms tailor results to user preferences to get more relevant and timely information.
- Contextual understanding ensures accurate results even for complex queries.
- Conversational search, powered by NLP processing, enables natural interactions with search engines.
- Visual search allows users to search using images or videos.
- **Main challenges is keeping the answers in the right context to gaining customer' trust.**
- Consumers concerns about accuracy and biases..



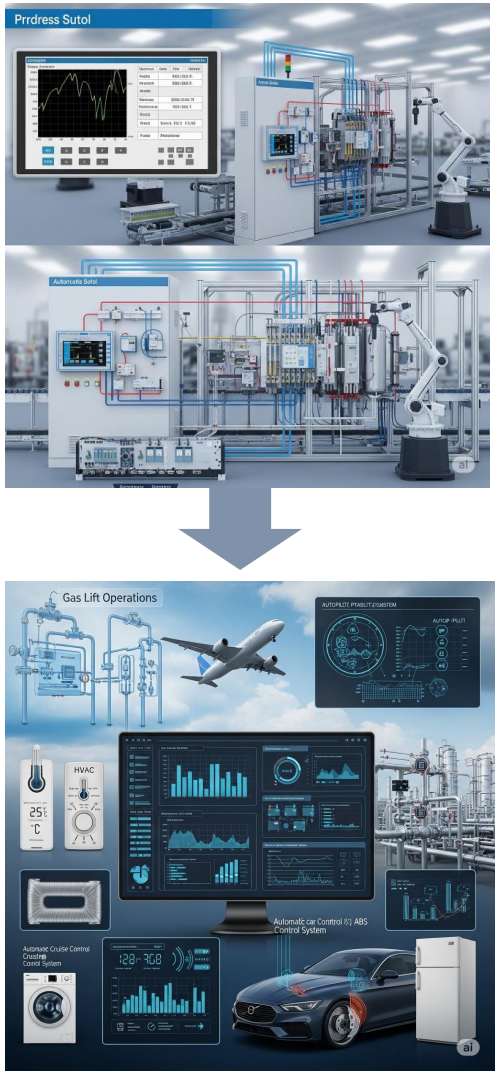
Statista (February 2023) showed that consumers are curious about AI-powered search but have concerns about its accuracy and biases.

AI predictions: Top 13 AI trends for 2024, Pragmatic Coders, Dec 10, 2023. medium.com

Create an Image Prompt

Got to → https://gemini.google.com/

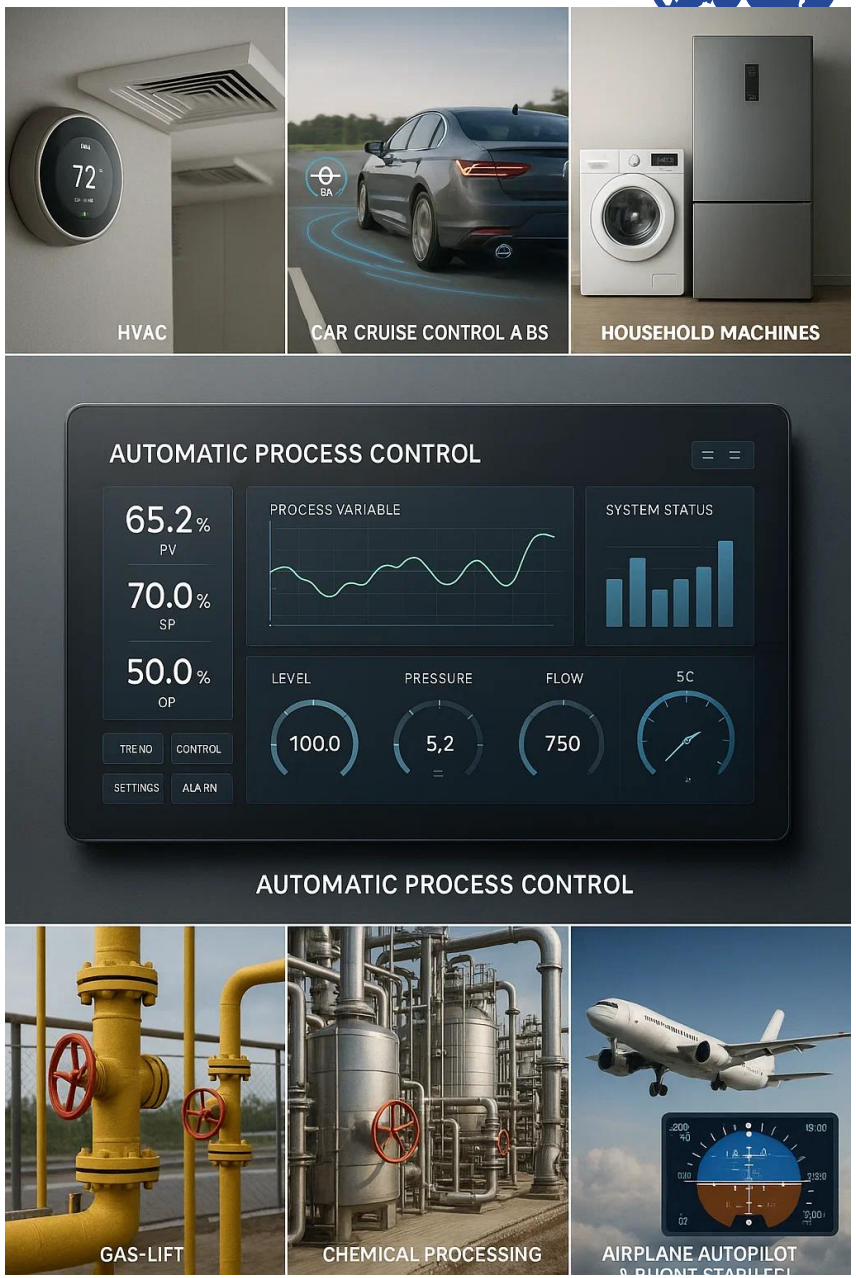
- Create a realistic, high-fidelity image depicting the overarching concept of automatic process control. The central focus should be a modern, sleek control panel or interface displaying dynamic data, graphs, and system readouts. Around this central element, visually integrate distinct, realistic scenes or elements representing the following common use cases:
  - HVAC: A smart thermostat displaying temperature and a subtle background showing a ventilation system.
  - Car Cruise Control & ABS: A modern car on a road, with visual cues (like subtle glowing lines or icons) indicating active cruise control and the anti-lock braking system engaging.
  - Household Machines: Everyday appliances like a washing machine or refrigerator, with digital interfaces highlighting their automated functions.
  - Gas-Lift: A simplified yet realistic depiction of gas-lift equipment, perhaps a network of pipes and valves, with subtle indications of automated flow control.
  - Chemical Processing: A section of a chemical plant with vats, pipes, and control valves, emphasizing automated process regulation.
  - Airplane Autopilot & Flight Stability: An airplane in flight, with an overlay or split view showing a realistic cockpit display indicating autopilot activation and flight stability parameters.
- Ensure the overall image conveys a sense of interconnectedness and advanced technology, with clean lines and a professional aesthetic.




Prrdress Sutol
Automatla Sutol


Gas Lift Operations
HVAC
Automatic car Control ⅀ ABS
Control System
Automatic Cruise Control
Cruisfee
Control System


HVAC
720
CAR CRUISE
CONTROL & ABS
AUTOMATIC PROCESS CONTROL
SETPOINT
WEASURED
120.0
118.5
CONTROLLER OUTPUT
59%
PRESSURE
OUTUIDUT
72
62
852
MODE
MANUAL
1:20
M
PRESSURE
LEVEL
GAS-LIFT
AIRPLANE
AUTOPILOT &
FLIGHT STABILITY
HOUSEHOLD
MACHINES
MODE
MANUAL
HOUSEHOLD
MACHINES
CHEMICAL
PROCESSING


72
HVAC
CAR CRUISE CONTROL A BS
HOUSEHOLD MACHINES
AUTOMATIC PROCESS CONTROL
65.2%
PV
70.0%
SP
50.0%
OP
PROCESS VARIABLE
SYSTEM STATUS
TRENO
CONTROL
SETTINGS
ALARN
LEVEL
PRESSURE
FLOW
5C
100.0
5,2
750
AUTOMATIC PROCESS CONTROL
GAS-LIFT
CHEMICAL PROCESSING
AIRPLANE AUTOPILOT

Final Project

- Participants to develop their own application of AI and ML techniques to a sample dataset relevant to their work challenges.
- Project Steps:
  1. Define the problem: what problem can be addressed with simple AI/ML methods?
  2. Define project, success metrics (accuracy, time savings, decision making)
  3. Define and access required data sources and types
  4. Define required algorithm steps: data loading, cleaning, transforming, model train, prediction, visualization
  5. Build code
  6. Assess model performance (error, predictability)
  7. Final Report (3 slides)
     - 1 - Problem definition, Background, Data Sources,
     - 2 - Code structure, performance analysis
     - 3 - Results, Conclusion, Way Forward
- Participants will have ~2 weeks to submit their project results as a final presentation and code.
- Top projects will be showcased in the following SPE Bahrain Local section meeting in August 2025

SPE AI Machine Learning for Energy Professionals Boot Camp

APPENDIX

Python Kick Start Program

Python Basics

1. Print('hello'), Print(), /n
2. A=input('enter A=')
3. String to Numeric
4. String Count
5. Sentence Upper, Lower, Capitalize
6. Date and Time
7. Conditionals (if then elif)
8. Collections: Lists/arrays, dictionaries
9. Loops
10. Functions, Lambda
11. Managing the file system

Essential Libraries

1. OS
2. Numpy
3. Pandas
4. Matplotlib
5. Seaborn
6. Scipy
7. Scikit Learn (sklearn)
8. LangChain

Python Basics: Print

- Syntax is straightforward, and easy to read (forcing indented blocks)
- Coding can start with only 16 statements (if, while/for, import…)

Print displays output to console

```
print('Hello world’)
```

```
Hello World
```

Getting information from the user

```
name = input('Please enter your name: ')
print(name)
```

```
Please enter your name: Tom
Tom
```

Printing blank lines and next line

```
# print() = prints a blank line
print()
# \n requests to prints a blank line in the middle of string
print(‘Hello World \nHello Everyone')
```

Debugging with print

```
print('Adding numbers')
x = 42 + 206
print('Performing division')
y = x / 0
print('Math complete’)
# This is a comment - it does nothing
```

Python Basics: String to Numeric

Numbers stored as strings must be converted to numeric values before doing math

```
first_num = input('Enter first number ')
second_num = input('Enter second number ')
print(int(first_num) + int(second_num))
print(float(first_num) + float(second_num))
```

```
Enter first number 50
Enter second number 60
110
110.0
```

Python Basics: Print Strings

Combine strings with +

```
your_name = input('What is your name? :')
your_age = input('Enter your age? : ')
print ('Hello ' + your_name.capitalize() +
        ', your age is ' + your_age)
```

```
What is your name? TOM
Enter your age? 25
Hello Tom, your age is 25
```

format strings that are saved to files and databases, or display to users

Format Text

```
sentence = 'Sample Sentence'
print(sentence.upper())
print(sentence.lower())
print(sentence.capitalize())
print(sentence.count('a'))
```

```
days_in_feb = 28
print(str(days_in_feb) + ' days in Feb')
```

Python Basics: Date and Time

Datetime objects to manipulate dates

```
# datetime and timedelta to define a period
from datetime import datetime, timedelta
today = datetime.now()
print('Today is ' + str(today))

# timedelta is used to define a period of time
one_day = timedelta(days=1)
yesterday = today - one_day
print('Yesterday was ' + str(yesterday))
```

```
Today is: 2021-05-15 12:53:51.831235
Yesterday was: 2021-05-14 12:53:51.831235
```

Converting it to a datetime allows you to use the date functions

```
from datetime import datetime, timedelta
birthday = input('Enter birthday (dd/mm/yyyy)? ')
b_date = datetime.strptime(birthday, '%d/%m/%Y')
print ('Birthday: ' + str(b_date))
birthday_eve = b_date - one_day
print('Day before birthday: ' + str(birthday_eve))
```

```
When is your birthday (dd/mm/yyyy)? 23/07/2000
Birthday: 2000-06-24 00:00:00
Day before birthday: 2000-06-23 00:00:00
```

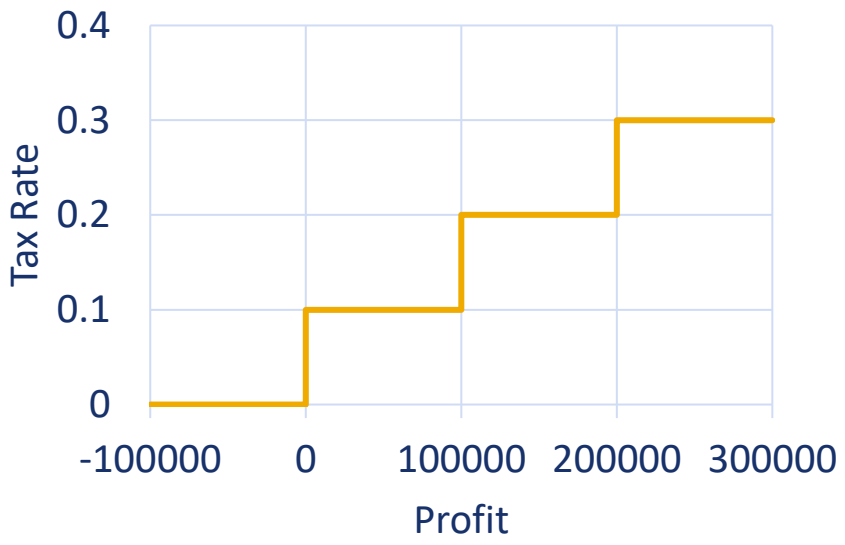
Python Basics: Conditionals

Different actions depending on conditions:

```
if Profit > 0.00:
    tax_rate = .15
else:
    tax_rate = 0
print(tax_rate)
```

Operation	Symbol
Less than	<
Less than or equal to	<=
Greater than	>
Greater or equal to	>=
Equal to	==
Not equal to	!=

```
if Profit >= 200000.00:
    tax_rate = .30
elif Profit >= 100000.00:
    tax_rate = .20
elif Profit > 0.00:
    tax_rate = .1
else:
    tax_rate = 0
print(tax_rate)
```




```
if Profit >= 100000.00 and Profit < 200000.00:
    tax_rate = .20
if Profit > 0.00 and Profit < 100000.00:
    tax_rate = .10
```

Python Basics: Collections: Lists

Lists: collections of items

```
countries = ['USA', 'UAE']
wells = []
wells.append(1202) # Add items to the end of list
wells.append(670)
print(countries)
print(wells)
print(wells[1]) # Collections are zero-indexed
```

```
['USA', 'UAE']
[1202, 670]
670
```

Lists are used to store anything, and any type:

```
names = ['Pepe', 'Juan']
names.append(2)
names.append(3.14)
print(names)
print(names[1])
print(names[3])
```

```
['Pepe', 'Juan', 2, 3.14]
Juan
3.14
```

Python Basics: Collections: Arrays

Arrays: collections of items

```
from array import array
wells = array('d')
wells.append(1202)
wells.append(670)
print(wells)
print(wells[1])
```

```
array('d', [1202.0, 670.0])
670.0
```

Arrays represent basic values and behave lists, except the type of objects stored in them is constrained. The type is specified by:

Type code	C Type	Minimum size in bytes
'b'	signed integer	1
'B'	unsigned integer	1
'u'	Unicode character	2
'h'	signed integer	2
'H'	unsigned integer	2
'i'	signed integer	2
'I'	unsigned integer	2
'l'	signed integer	4
'L'	unsigned integer	4
'q'	signed integer	8
'Q'	unsigned integer	8
'f'	floating point	4
'd'	floating point	8

Python Basics: Collections: List Ops

List Operations: count items – insert items

```
names = ['Tom', 'Carmen']
print(len(names)) # Get the number of items
names.insert(0, 'Mary') # Insert before index
print(names)
names.sort()
print(names)
```

```
2
['Mary', 'Tom', 'Carmen']
['Carmen', 'Mary', 'Tom']
```

List Operations: Retrieving ranges

```
names = ['Tom', 'Carmen', 'Mary’]
short_list = names[0:2] # Get the first two items
# Starting index and number of items to retrieve

print(names)
print(short_list)
```

```
['Tom', 'Carmen', 'Mary']
['Tom', 'Carmen']
```

Python Basics: Collections: Dictionary

Dictionary

```
person = {'first': 'Tom'}
person['last'] = 'Harrison'
print(person)
print(person['first'])
print(person['last'])
```

```
{'first': 'Tom', 'last': 'Harrison'}
Tom
Harrison
```

Dictionaries store key/value pairs; storage order is not guaranteed

```
well = {'Name': 'TX-001'}
well['API_No'] = '42-183-89876'
well['County'] = 'Fort Bend'
print(well)
print('Well name is ' + well['Name'])
print('Well API No. is ' + well['API_No'])
print('Well County is ' + well['County'])
```

```
{'Name': 'TX-001', 'API_No': '42-183-89876',
'County': 'Fort Bend'}
Well name is TX-001
Well API No. is 42-183-89876
Well County is Fort Bend
```

Python Basics: Loops

Loop through a collection

```
for name in ['Tom', 'Mary']:
    print(name)
```

```
Tom
Mary
```

Looping a number of times

```
for x in range(0, 5):
    print(x)
```

```
0
1
2
3
4
```

Looping with a condition

```
names = ['Mary', 'Tom', 'Carmen']
index = 0
while index < len(names):
    print(names[index])
    index = index + 1
```

```
Mary
Tom
Carmen
```

Python Basics: Functions

Simple function to get the initials

```
def get_initial(name):
    initial = name[0:1].upper()
    return initial
first = input('Enter first name: ')
first_initial = get_initial(first)
last = input('Enter last name: ')
last_initial = get_initial(last)
print('Initials are: ' + first_initial +
last_initial)
```

Parameter name is passed, and function is executed many times

```
Enter first name: Luigi
Enter last name: Saputelli
Initials are: LS
```

Python Basics: Functions

Simple function to print the duration of a process

```
from datetime import datetime
def print_time(task_name, time_start, time_end):
    difference = time_end - time_start
    total_seconds = difference.total_seconds()
    print(task_name + ' started on ' + str(time_start))
    print(task_name + ' ended on ' + str(time_end))
    print(task_name + ' completed in ' + str(total_seconds) + ' secs')
time_start = datetime.now()
for x in range(0,20000):
    print(x)
time_end = datetime.now()
print_time('Task 1', time_start, time_end)
```

3 parameters are passed, and function can be executed many times

```
Task 1 started on 2021-05-15 20:59:41.446098
Task 1 ended on 2021-05-15 20:59:43.515552
Task 1 completed in 2.069454 secs
```

Python Basics: Functions

Alternate way to estimated the duration of a process

```
# install requests
import requests
from timeit import default_timer

start_time=default_timer()
file_from_httpbin = requests.get('https://httpbin.org/delay/5').text
elapsed_time = default_timer() - start_time
print(f'Request access time is {elapsed_time:.2} secs')
```

```
Request access time is 5.83 secs
```

Python Basics: Lambda

Sort “items” in a Dictionary by “name”

```
members = [{'name': 'Tom', 'age': 41}, {'name': 'Juan', 'age': 62}]
members.sort(key=lambda item: item['name'])
```

```
[{'name': 'Juan', 'age': 62}, {'name': 'Tom', 'age': 41}]
```

Sort “items” in a Dictionary by length of “name”

```
members.sort(key=lambda item: len(item['name']))
```

```
[{'name': 'Tom', 'age': 41}, {'name': 'Juan', 'age': 62}]
```

Sort “items” in a Dictionary by “age”

```
presenters.sort(key=lambda item: item['age'])
```

```
[{'name': 'Tom', 'age': 41}, {'name': 'Juan', 'age': 62}]
```

Simple Lambda Function

```
# Simple y=x+3 function
(lambda x: x + 3)(3) # = 6
```

Python Basics: File System

Working with paths

```
# Import library
from pathlib import Path

# Get the current directory
cwd = Path.cwd()
print(cwd)

# Combine folder name and file name to create full path
new_file = Path.joinpath(cwd, 'file.txt')
print(new_file)
```

Some common libraries

- os
- pathlib
- math
- numpy
- pandas
- matplotlib
- scikit.learn
- keras
- tensorflow
- etc

Python Basics: Managing Files

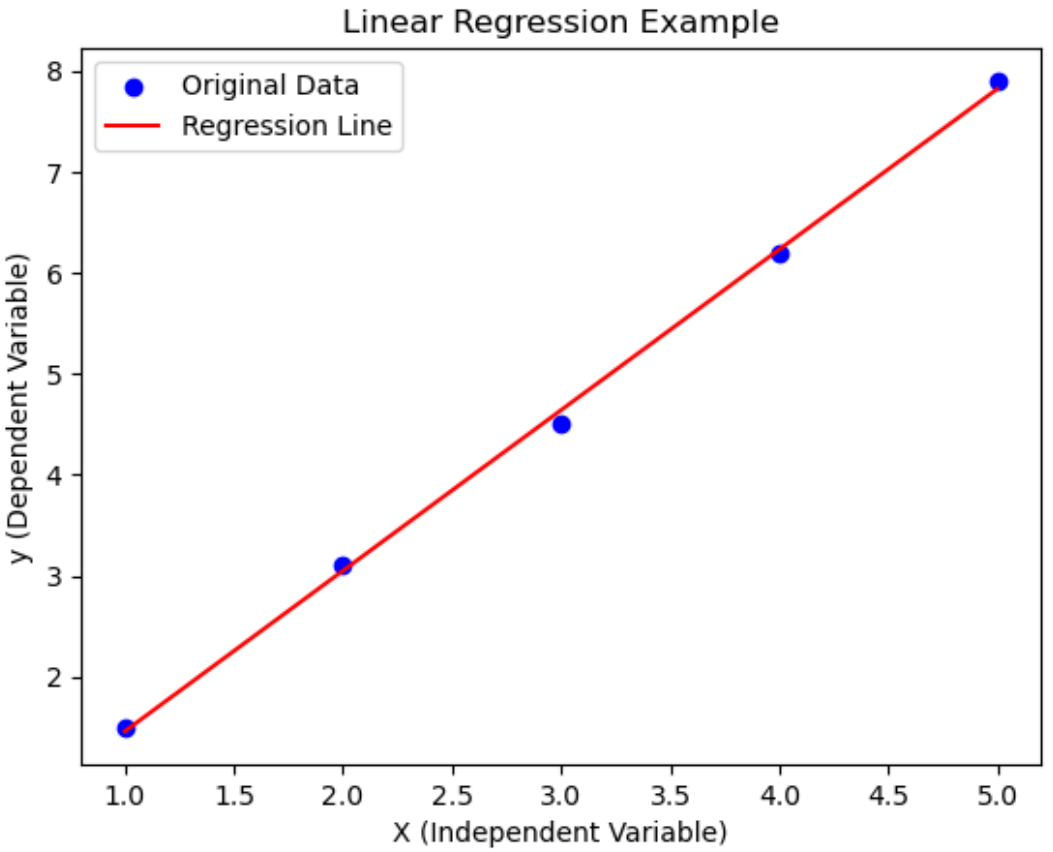
Reading and Writing files

```
stream = open('demo.txt')
print(stream.readable()) # Can we read?
print(stream.read(1)) # Read the first character
print(stream.readline(5)) # Read any line
stream.close() # close the file

stream = open('output.txt', 'wt') # write text
stream.write('H') # write a single string
stream.writelines(['ello',' ','world']) # write multiple strings
stream.write('\n') # write a new line
names = ['Susan','Christopher'] # create a list of strings
stream.writelines(names) # write list of strings
stream.close() # close the stream (and flush data)
```

Contents

Linear Regression (Univariate example)

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1.5, 3.1, 4.5, 6.2, 7.9])
# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)
# Make predictions
predicted = model.predict(X)
# Plot the data and regression line
plt.scatter(X, y, color='blue', label='Original Data')
plt.plot(X, predicted, color='red', label='Reg. Line')
plt.xlabel('X (Independent Variable)')
plt.ylabel('y (Dependent Variable)')
plt.title('Linear Regression Example')
plt.legend()
plt.show()
# Display the results
print("Coefficient (slope):", model.coef_[0])
print("Intercept:", model.intercept_)
print("Predicted values:", predicted)
```

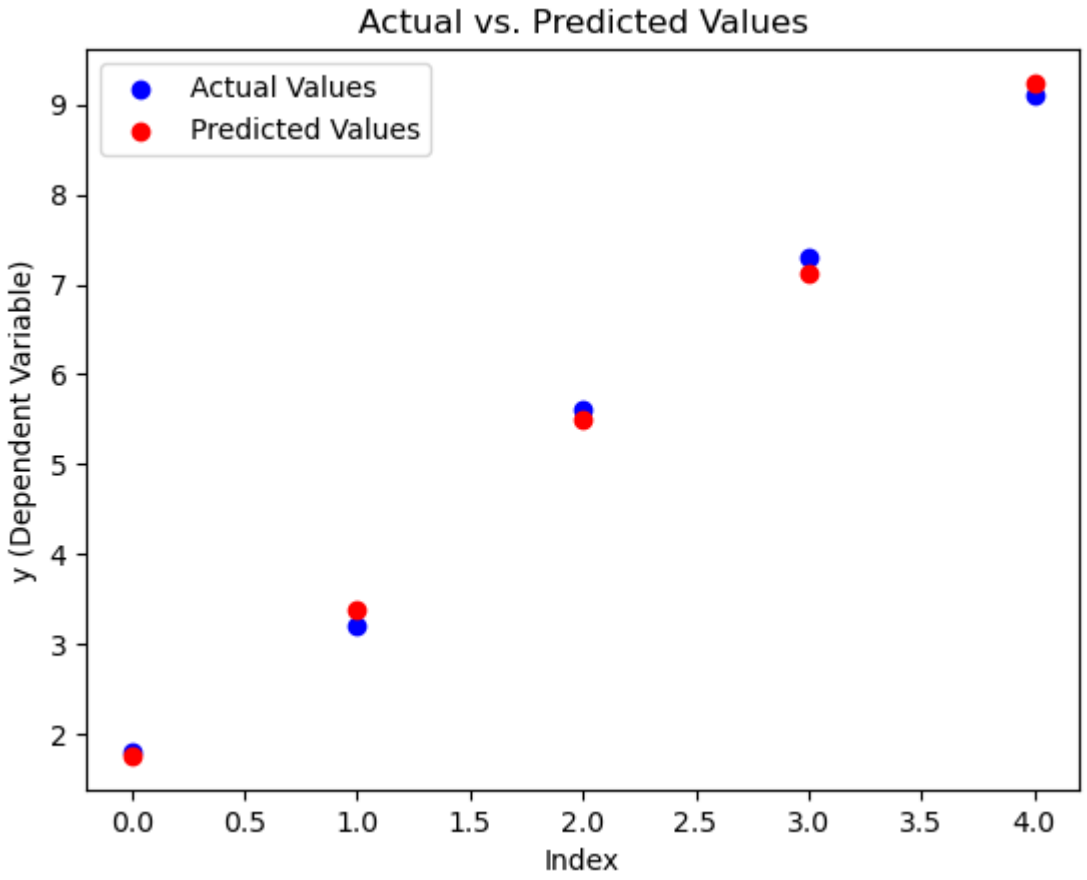



```
Coefficient (slope): 1.59
Intercept: -0.1299999999999999
Predicted values: [1.46 3.05 4.64 6.23 7.82]
```

Accuracy: 0.9407407407407408

Linear Regression (Multivariate example)

```
from sklearn.linear_model import LinearRegression
# Two independent x variables (features)
X = np.array([[1, 2], [2, 1], [3, 4], [4, 3], [5, 6]])
y = np.array([1.8, 3.2, 5.6, 7.3, 9.1])
# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)
# Make predictions
predicted = model.predict(X)
# Plotting actual vs. predicted values.
plt.scatter(range(len(y)), y, color='blue', label='Actual \
plt.scatter(range(len(predicted)), predicted, color='red',
# Add labels, legend and Display the plot
plt.xlabel('Index')
plt.ylabel('y (Dependent Variable)')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
# Display the results
print("Coefficients (slopes):", model.coef_)
print("Intercept:", model.intercept_)
print("Predicted values:", predicted)
```




```
Coefficients (slopes): [1.745 0.125]
Intercept: -0.2349999999999943
Predicted values: [1.76 3.38 5.5  7.12 9.24]
```

Decision Trees

Decision Trees are used for classification and regression tasks based on decision rules derived from the data.

Each node in the tree represents a decision point, and the leaves represent the final predictions.

They are simple to understand and interpret.

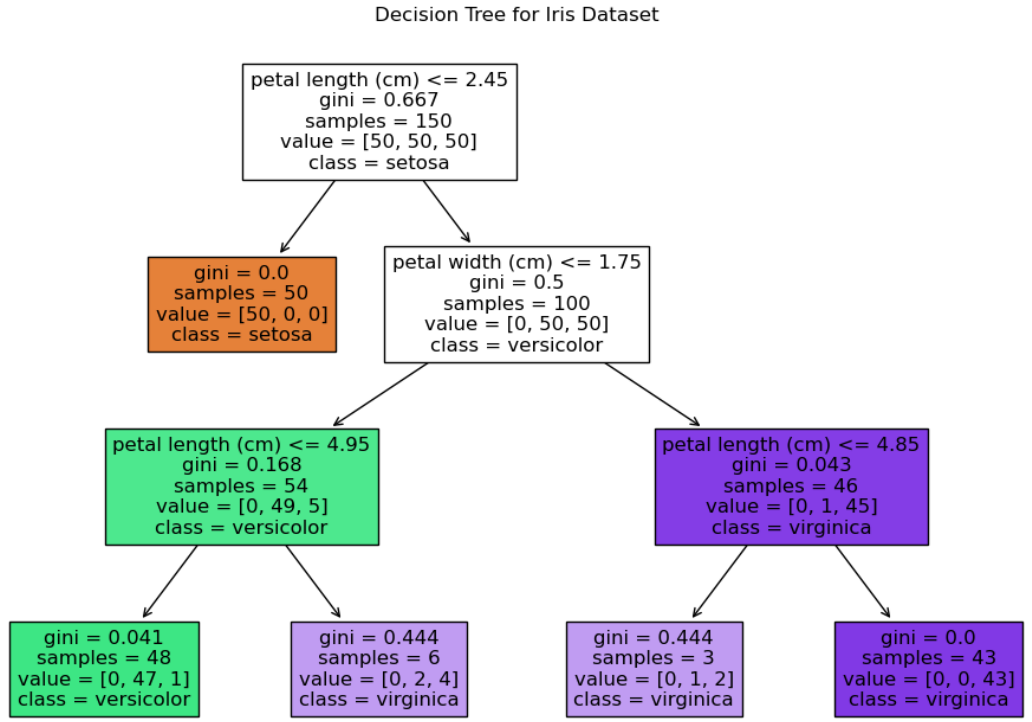
Decision Tree Classification (Iris Example)

Sample	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5	3.6	1.4	0.2	setosa
..	...	...	...	...	...
145	6.7	3	5.2	2.3	virginica
146	6.3	2.5	5	1.9	virginica
147	6.5	3	5.2	2	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3	5.1	1.8	virginica

```
# Load the Iris dataset
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()

# Create a DataFrame for the dataset
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Add the target column with species names
iris_df['species'] = [iris.target_names[target] for target in iris.target]

# Display the first few rows of the dataset
print(iris_df)
```

Decision Tree Classification (Iris Example)

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Load the Iris dataset (a dataset for classification)
iris = load_iris()
X = iris.data  # Features (petal and sepal measurements)
y = iris.target  # Target (species of iris plant)

# Convert class names to a list
class_names = list(iris.target_names)

# Create and fit the Decision Tree Classifier
model = DecisionTreeClassifier(max_depth=3, random_state=42)
model.fit(X, y)

# Plot the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=iris.feature_names, class_names=class_names, filled=True)
plt.title("Decision Tree for Iris Dataset")
plt.show()
```



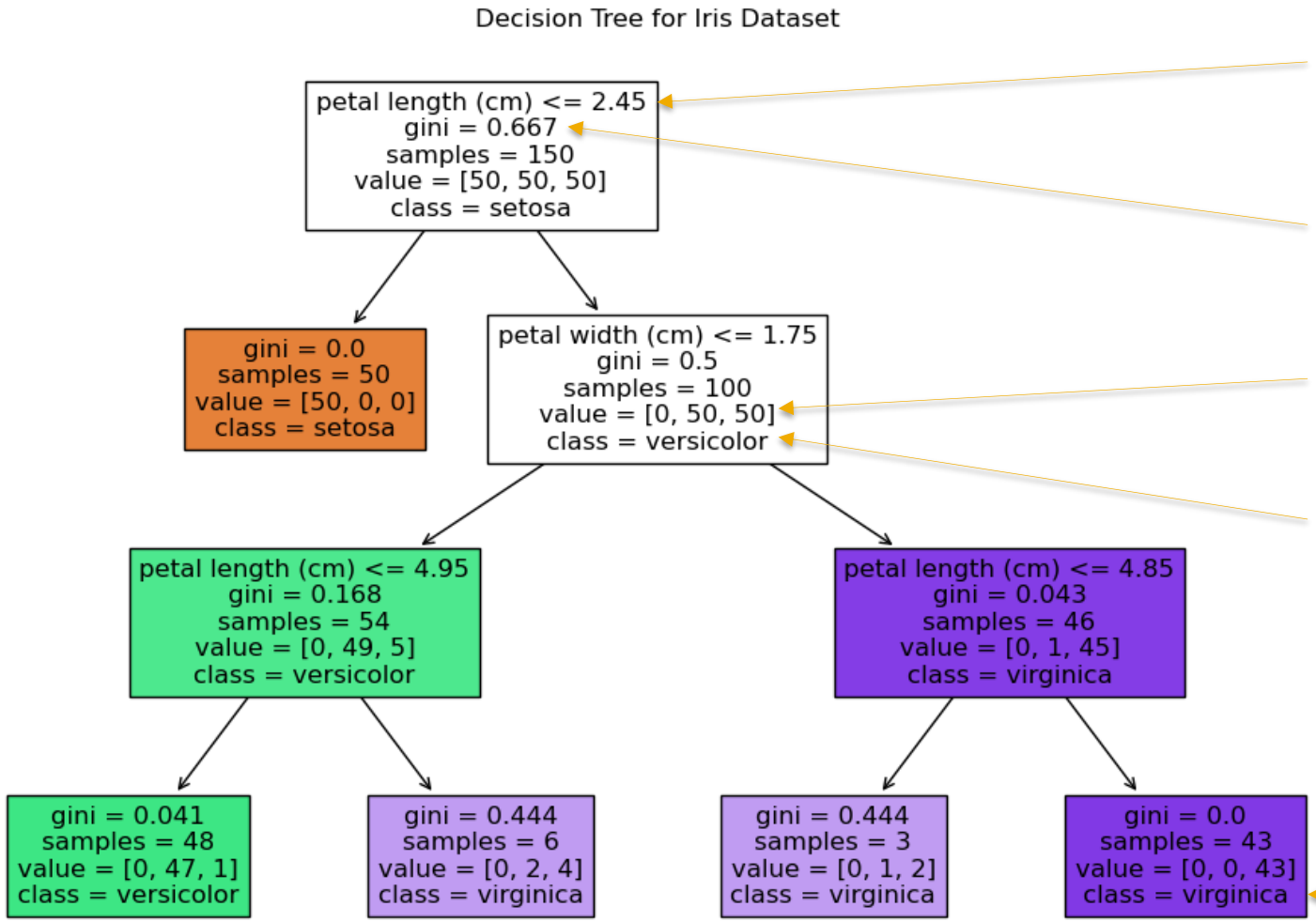
Decision Tree Interpretation

Root Node

Internal Node

Internal Node

Leaf Node



Data is divided into two groups based on whether the feature value is <= 2.45

Gini measures the **probability of incorrect classification** or how pure the split is. Lower values indicate purer splits.

Samples indicate how many samples (data points) are in this node

Class Distribution: proportion of data points belonging to each class at that node

Predicted Class (Leaf Nodes):
The class with the highest count at a leaf node is the prediction for that branch

Random Forests

Random Forest is a machine learning algorithm based on ensemble learning.

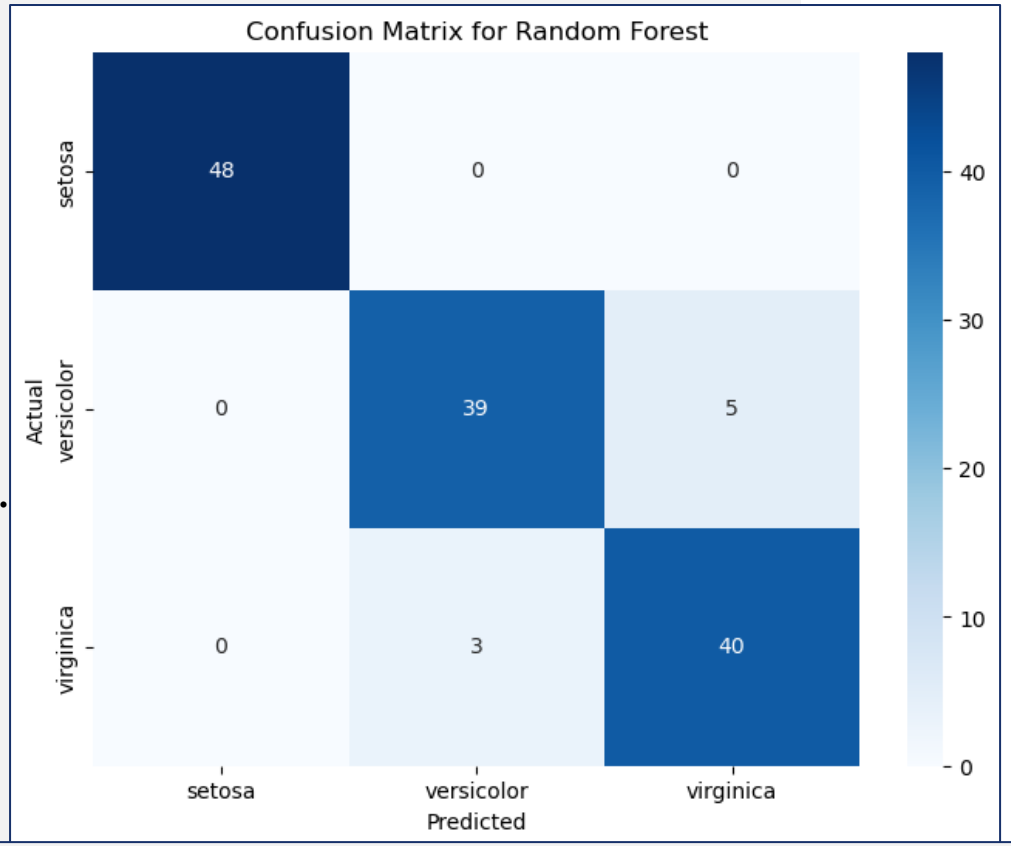
It constructs multiple decision trees during training and merges their outputs (through averaging in regression or majority voting in classification) to improve accuracy, reduce overfitting, and handle missing data effectively.

Key Features:

- **Robust to Overfitting**: Combining multiple trees reduces the risk of overfitting compared to a single decision tree.
- **Handles High-Dimensional Data**: Works well with datasets that have many features and instances.
- **Feature Importance**: Provides insights into which features are most influential in the model.

Random Forest Classification (Iris Example)

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target (species of iris plant)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
# Create and fit the Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predict on the test data
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Plot a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.show()
```




```
Accuracy: 0.9407407407407408
```

Logistic Regression

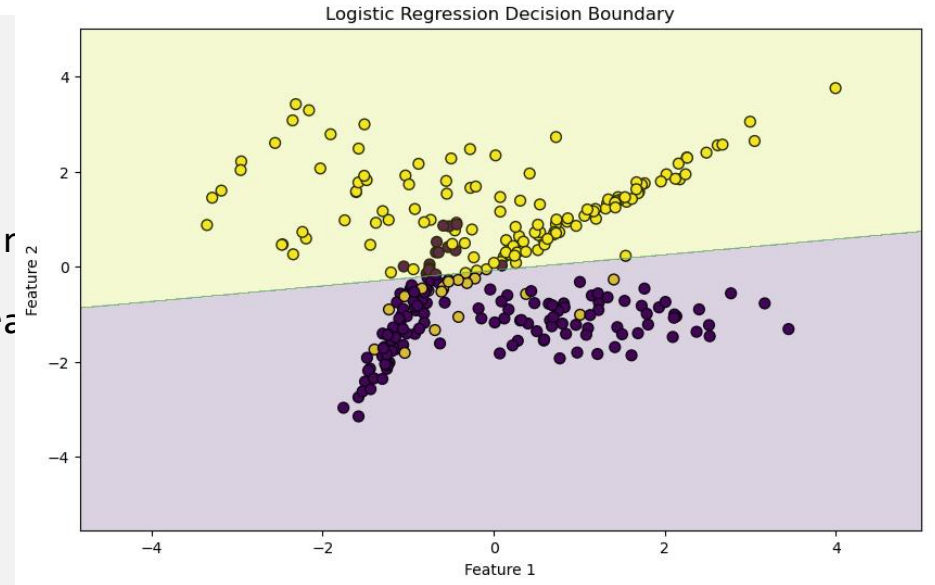
Logistic Regression is employed for binary classification problems. It predicts the probability that an instance belongs to a particular class using the logistic function, which maps any real-valued number into a value between 0 and 1. The logistic regression model is defined as:

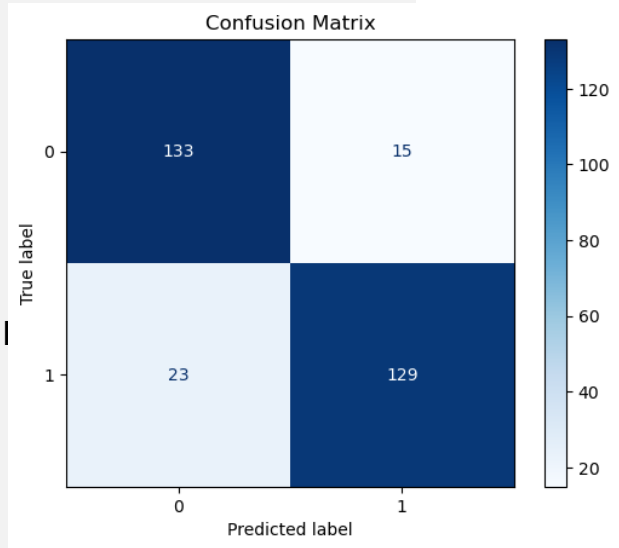
$$\text{logit}(P) = ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1$$

- Where P is the probability of the positive class.

Logistic Regression (Iris Example)

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Generate a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, r
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
# Create and fit a Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
# Predict the test set results
y_pred = model.predict(X_test)
# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', edgecolor='k', s=50)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis')
plt.title('Logistic Regression Decision Boundary'); plt.xlabel('Feature 1') plt.ylabel('F
# Compute confusion matrix and display it
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
```



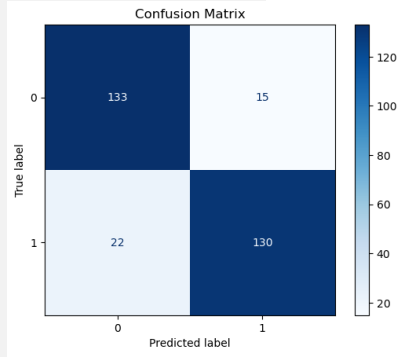


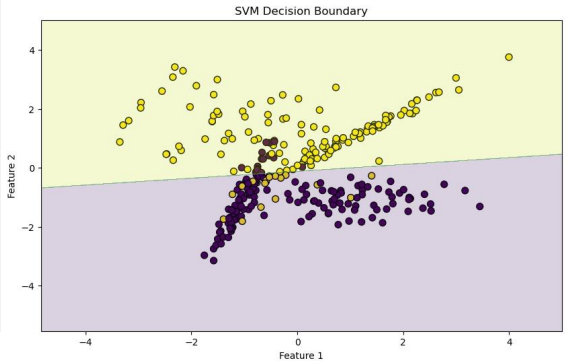
Support Vector Machines (SVMs)

- Support Vector Machines (SVMs) find a hyperplane that best separates the different classes in the feature space. The objective is to maximize the margin between the classes, improving the model's generalization capabilities.

Logistic Regression (Iris Example)

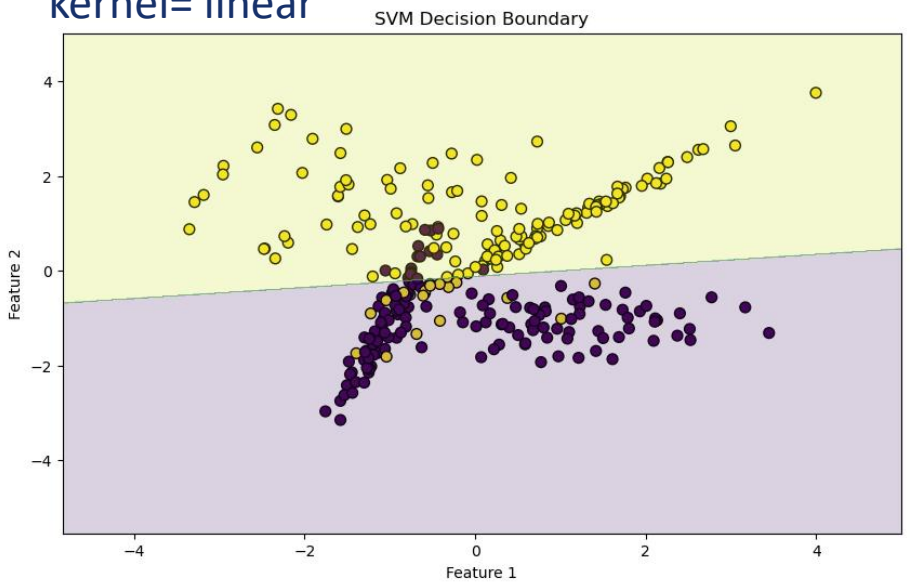
```
from sklearn.datasets import make_classification
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Generate a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, random_state=42)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and fit the SVM model
model = SVC(kernel='linear', random_state=42)
model.fit(X_train, y_train)
# Predict the test set results
y_pred = model.predict(X_test)

# Plot the decision boundary
Similar to previous

# Compute confusion matrix and display it
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

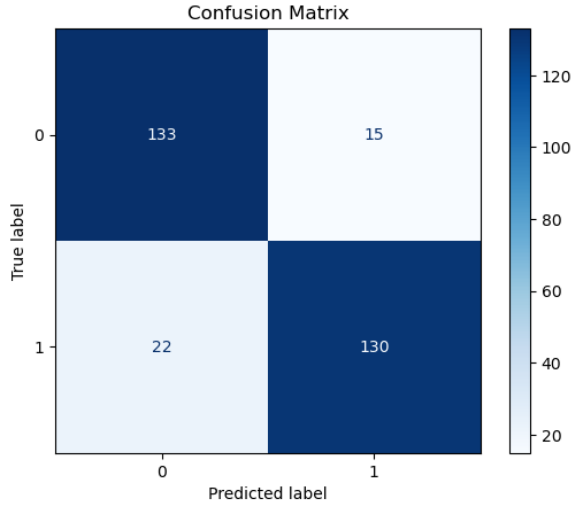




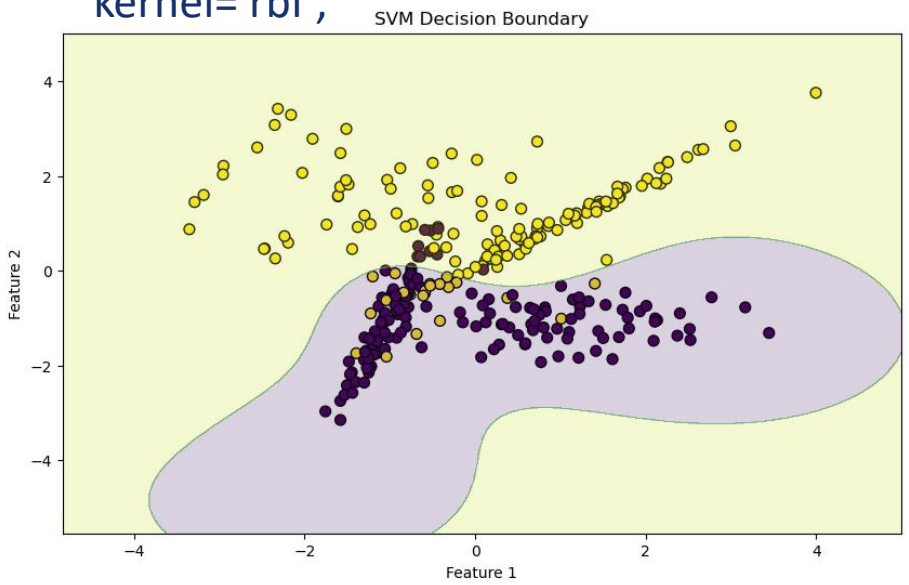
Logistic Regression (kernel for classification)

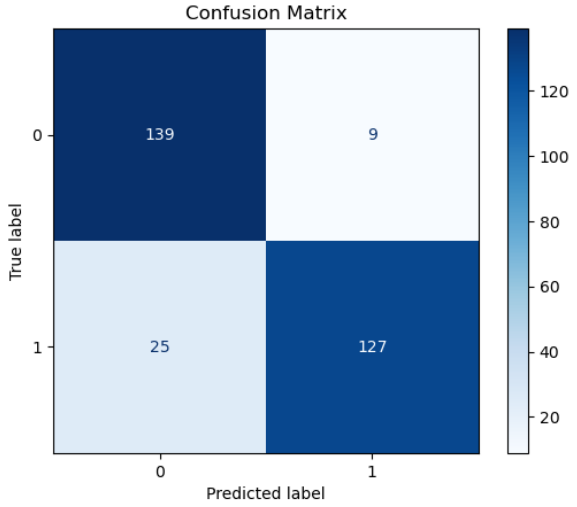
kernel='linear'





kernel='rbf',





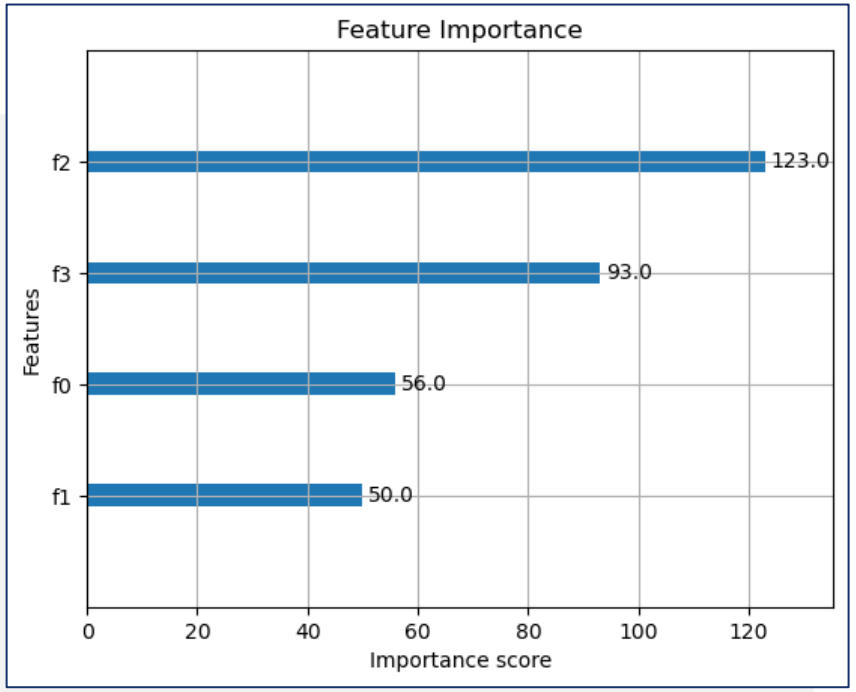
Gradient Boosting

Gradient Boosting is an ensemble method that builds models sequentially, with each new model correcting the errors of its predecessors, based on an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. XGBoost and LightGBM are popular implementations of this algorithm, known for its efficiency and performance in competitive machine learning tasks.

1. **XGBoost (eXtreme Gradient Boosting)**: XGBoost uses a decision-tree-based ensemble learning method to improve prediction accuracy, particularly for structured (tabular) data. XGBoost is known for its speed and performance in training complex models, thanks to features like regularization, tree pruning, and handling missing values. It gained popularity for its strong performance in machine learning competitions like Kaggle.
2. **LightGBM (Light Gradient Boosting Machine)**: LightGBM is uses a technique called **leaf-wise tree growth**, which splits the tree based on the leaf with the largest loss reduction, leading to deeper trees and potentially better accuracy. It's also memory-efficient and can handle categorical features without explicit preprocessing.

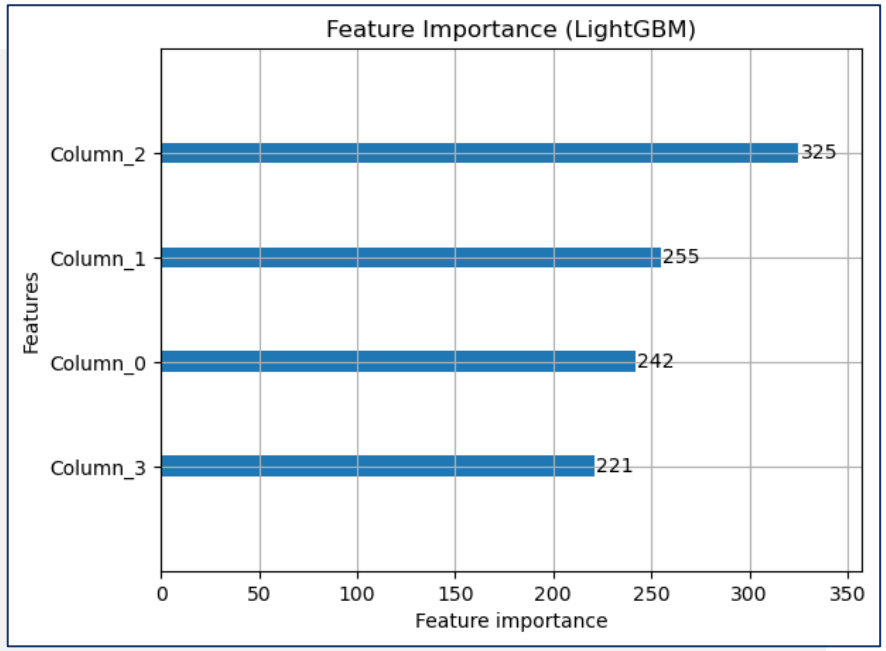
XGBoost (Example)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import xgboost as xgb
# Load the Iris dataset
iris = load_iris()
X = iris.data    # Features
y = iris.target  # Target (species of iris plant)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Create DMatrix for XGBoost (optimized data structure for XGBoost)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Set parameters for XGBoost
params = {"objective": "multi:softmax", "num_class": 3 "max_depth": 4,"eta": 0.3, “seed": 42 }
# Train the XGBoost model
bst = xgb.train(params, dtrain, num_round=50)
# Make predictions
y_pred = bst.predict(dtest)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

# Feature importance plot
xgb.plot_importance(bst)
plt.title("Feature Importance")
plt.show()
```



Light GBM (Example)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from lightgbm import LGBMClassifier
# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target (species of iris plant)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Initialize the LightGBM model
model = LGBMClassifier(objective="multiclass", num_class=3,
    learning_rate=0.1,max_depth=6, n_estimators=100)

# Train the model with early stopping
model.fit(X_train, y_train,eval_set=[(X_test, y_test)], eval_metric="multi_logloss")

# Make predictions
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

# Plot feature importance
lgb.plot_importance(model, max_num_features=10)
plt.title("Feature Importance (LightGBM)")
plt.show()
```

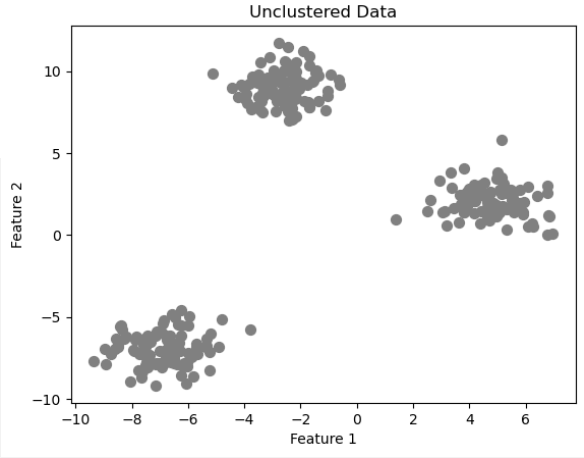


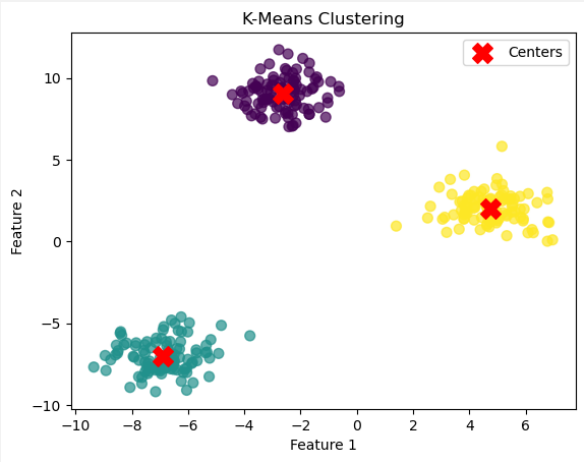
Kmeans

- K-means is an unsupervised clustering algorithm that groups data into K clusters based on feature similarity.
- The algorithm iteratively assigns data points to the nearest cluster center and updates the cluster centers until convergence.

Kmeans (Example)

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Generate a synthetic dataset
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0, random_state=42)

# Plot the data points
plt.scatter(X[:, 0], X[:, 1], s=50, c='gray')
plt.title("Unclustered Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)  # Specify 3 clusters
kmeans.fit(X)  # Fit the model to the data

# Get the cluster labels and cluster centers
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Plot the clustered data
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7)  # Data points with cluster labels
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X', label='Centers')  # Cluster centers
plt.title("K-Means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```



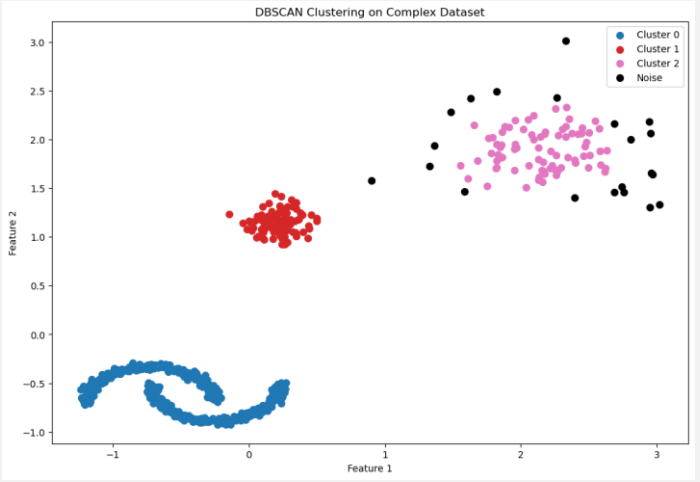


DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- DBSCAN is a clustering algorithm used to group data points based on their density. Unlike other clustering algorithms like K-Means, DBSCAN does not require to specify the number of clusters in advance and can handle clusters of arbitrary shapes. It’s especially good at identifying noise (outliers) in the data.

DBSCAN (Example)

```
from sklearn.datasets import make_moons, make_blobs
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
# Generate a complex dataset with non-linear shapes and varying densities
X1, _ = make_moons(n_samples=500, noise=0.05, random_state=42)  # Moon-shaped clusters
X2, _ = make_blobs(n_samples=200, centers=[[2, 5], [6, 7]], cluster_std=[0.3, 0.8], random_state=42)  # Circular
clusters
X = np.vstack((X1, X2))  # Combine the datasets
# Standardize the data (DBSCAN works better on standardized data)
X = StandardScaler().fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.2, min_samples=10)  # Adjust parameters for this dataset
labels = dbscan.fit_predict(X)  # Perform clustering

# Plot the results
plt.figure(figsize=(12, 8))
unique_labels = set(labels)  # Get unique cluster labels
colors = [plt.cm.tab10(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Noise points are labeled as -1
        col = [0, 0, 0, 1]  # Black for noise
    class_member_mask = (labels == k)  # Boolean mask for the current cluster
    plt.scatter(X[class_member_mask, 0], X[class_member_mask, 1], color=tuple(col), s=50, label=f"Cluster {k}" if k !=
-1 else "Noise")
```



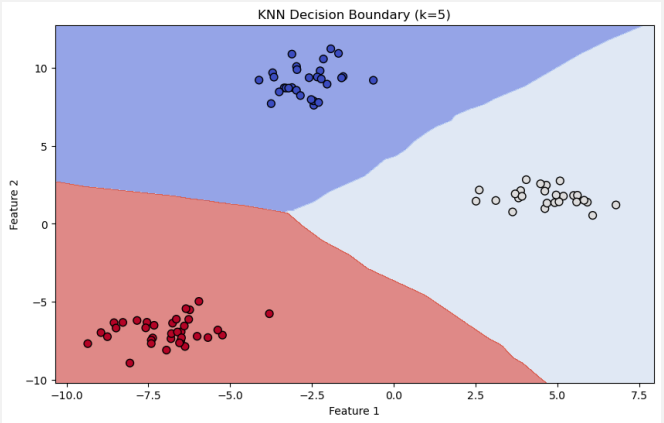
K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN)

- K-Nearest Neighbors (KNN) classifies an instance based on the classes of its k nearest neighbors in the feature space. It is a simple and intuitive algorithm that can be used for both classification and regression tasks.

KNN (Example)

```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Generate a synthetic dataset
X, y = make_blobs(n_samples=300, centers=3, cluster_std=1.0, random_state=42)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create and train the KNN model
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
# Make predictions on the test set
y_pred = knn.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"KNN Test Accuracy (k={k}): {accuracy}")
# Visualize the decision boundary
plt.figure(figsize=(10, 6))
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.6, cmap='coolwarm')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, edgecolor='k', cmap='coolwarm', s=50)
plt.title(f"KNN Decision Boundary (k={k})")
plt.xlabel("Feature 1") - plt.ylabel("Feature 2") - plt.show()
```



Naive Bayes

- Naive Bayes is a classification algorithm based on Bayes' theorem. It assumes that the features are conditionally independent given the class label. Despite this simplifying assumption, Naive Bayes performs well on a variety of classification tasks.

Hierarchical Clustering

- Hierarchical Clustering groups data into a cluster hierarchy by iteratively combining or splitting existing clusters. The resulting dendrogram provides a visual representation of the cluster structure.

Meanshift Clustering

- Meanshift Clustering assigns data points to clusters by iteratively shifting the points toward maximum data densities. This algorithm does not require the number of clusters to be specified in advance.

Association Algorithm (Apriori)

- The Apriori algorithm is used to find association rules in transactional datasets. It is particularly useful in market basket analysis to identify frequently co-occurring items.

Principal Component Analysis (PCA)

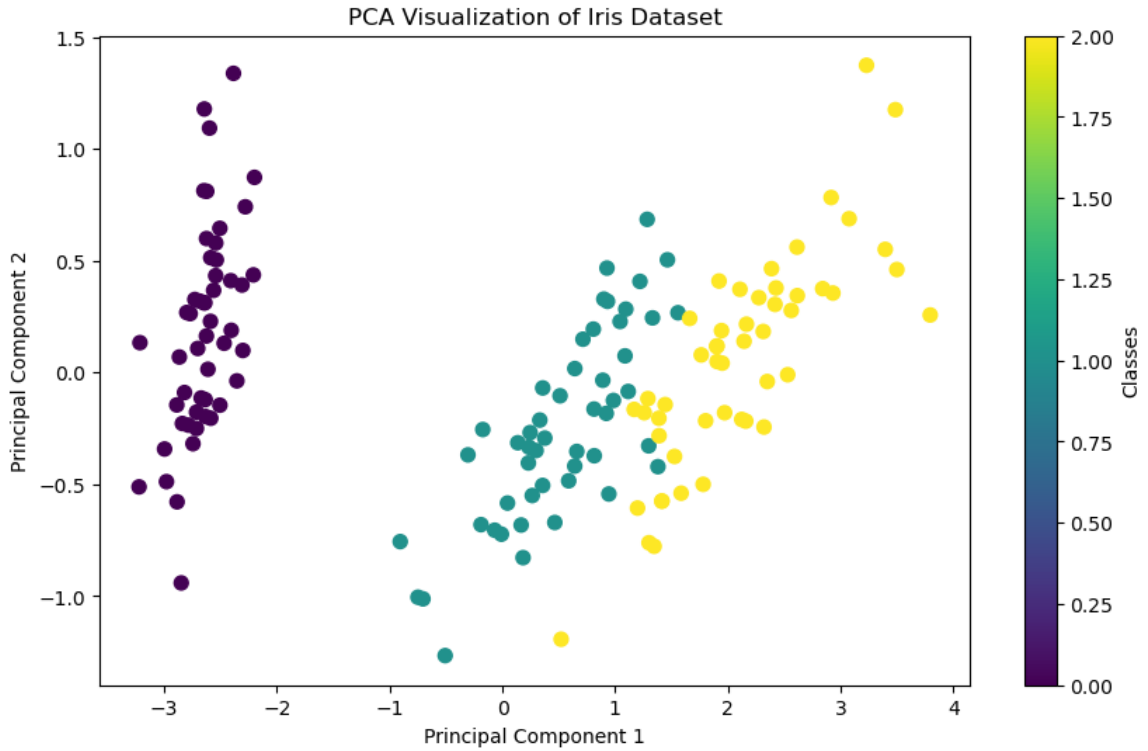
Principal Component Analysis (PCA) is a technique for dimensionality reduction. It transforms the original features into a new set of uncorrelated features called principal components, ordered by the amount of variance they capture.

Benefits of PCA:

- Reduces dimensionality while retaining most of the important information.
- Helps in visualizing high-dimensional datasets.
- Can improve computational efficiency for machine learning algorithms.

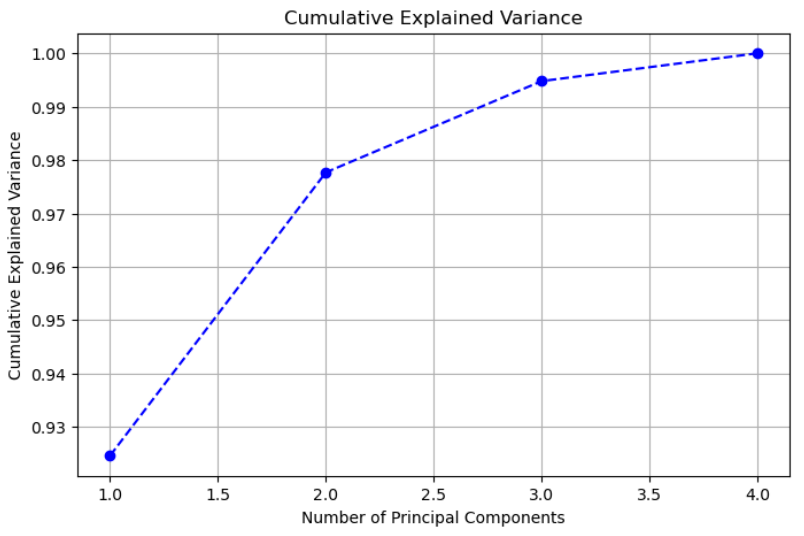
PCA (Example)

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

# Apply PCA to reduce the dimensionality to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot the PCA-transformed data
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y,
cmap='viridis', s=50)
plt.title('PCA Visualization of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(scatter, label='Classes')
plt.show()
```



PCA (Example)

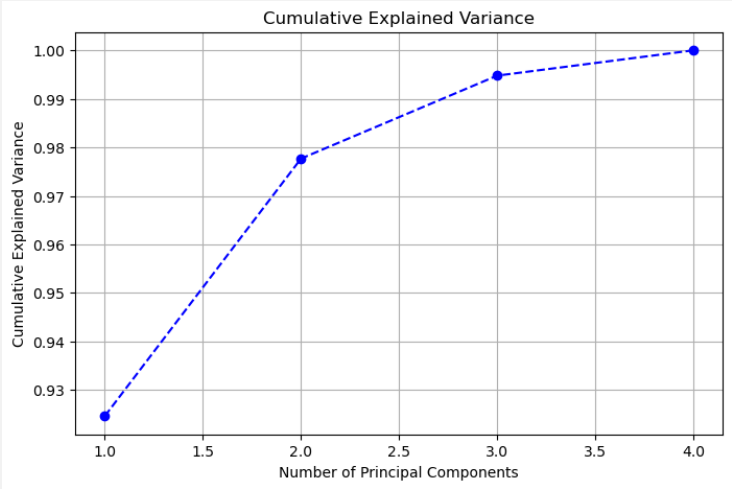
```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features

# Apply PCA
pca = PCA(n_components=None)  # Keep all components
X_pca = pca.fit_transform(X)

# Display the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained Variance Ratio:", explained_variance_ratio)

# Plot the cumulative explained variance
import matplotlib.pyplot as plt
cumulative_variance = np.cumsum(explained_variance_ratio)
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='--', color='b')
plt.title('Cumulative Explained Variance')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid()
plt.show()
```




```
Explained Variance Ratio: [0.92461872 0.05306648 0.01710261 0.00521218]
```

PCA (Example)

```
from sklearn.decomposition import PCA
from skimage import color, io

# Load a grayscale image (for simplicity)
image = io.imread('image.png', as_gray=True)
# Display the original image
plt.figure(figsize=(8, 4))
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis('off')
plt.show()
# Apply PCA to compress the image
def compress_image(image, num_components):
    # Flatten the image into a 2D array (rows = pixels, columns = features)
    pca = PCA(n_components=num_components)
    transformed = pca.fit_transform(image)
    reconstructed = pca.inverse_transform(transformed)  # Reconstruct the image
    return reconstructed
# Compress the image with reduced components
num_components = 50  # Number of principal components
compressed_image = compress_image(image, num_components)
# Display the compressed image
plt.figure(figsize=(8, 4))
plt.imshow(compressed_image, cmap='gray')
plt.title(f"Compressed Image with {num_components} Components")
plt.axis('off')
plt.show()
```



PCA Applications in many fields

1. Facial Recognition (Eigenfaces):

PCA is widely used in image processing for dimensionality reduction in facial recognition systems. Instead of storing full facial images, the algorithm reduces high-dimensional pixel data into a lower-dimensional subspace called eigenfaces.

- Why it's useful: PCA reduces computation time and storage requirements while retaining crucial facial features.

2. Stock Market Analysis:

PCA can identify patterns in financial data, such as correlations between stock prices.

- Example: Applying PCA to a dataset of stock prices can reduce hundreds of stocks into a few principal components that explain market trends.

- Why it's useful: Helps in portfolio optimization and understanding dominant factors influencing the market.

3. Gene Expression Data (Bioinformatics):

In genomics, PCA is often applied to high-dimensional gene expression datasets.

- Example: Reduce thousands of gene expression profiles into a few principal components to visualize differences between healthy and diseased samples.

- Why it's useful: Helps researchers identify patterns and clusters (e.g., cancerous vs. non-cancerous cells).

4. Anomaly Detection:

PCA can highlight outliers in high-dimensional datasets, like fraud detection in finance.

- Example: Apply PCA to credit card transaction data to reduce dimensions and identify anomalies (fraudulent transactions).

- Why it's useful: Simplifies the data while still allowing detection of suspicious patterns.

5. Topic Modeling in Natural Language Processing:

PCA can be used on textual data (e.g., term-document matrices) to uncover topics.

- Example: Reduce the dimensionality of a term frequency-inverse document frequency (TF-IDF) matrix to identify topics in a corpus.

- Why it's useful: Captures latent semantics without requiring advanced NLP methods.

6. Climate Science and Weather Analysis:

PCA is applied to analyze patterns in climate data (e.g., temperature, precipitation).

- Example: Reduce a dataset containing decades of weather readings to a few components to study dominant patterns like seasonal trends.

- Why it's useful: Helps simplify and interpret large-scale environmental datasets.

One Hot Encoder

Given a data set of categorical values, OneHotEncoder transforms the categories into binary columns.

sparse=False ensures the result is a dense array rather than a sparse matrix.

Output: Each row corresponds to a single input value, encoded as a binary vector.

Applications: Used in preprocessing categorical data for machine learning models. Common in linear models or distance-based models (like KNN) where numerical representation matters. Helps avoid issues like treating categories as ordinal values.

```
# Import necessary libraries
from sklearn.preprocessing import OneHotEncoder
categories = np.array([['Red'], ['Green'], ['Blue'], ['Red'], ['Blue']])

# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse=False)
# Apply One-Hot Encoding
one_hot_encoded = encoder.fit_transform(categories)
# Print the result
print("Original Categories:")
print(categories)
print("\nOne-Hot Encoded Data:")
print(one_hot_encoded)
```

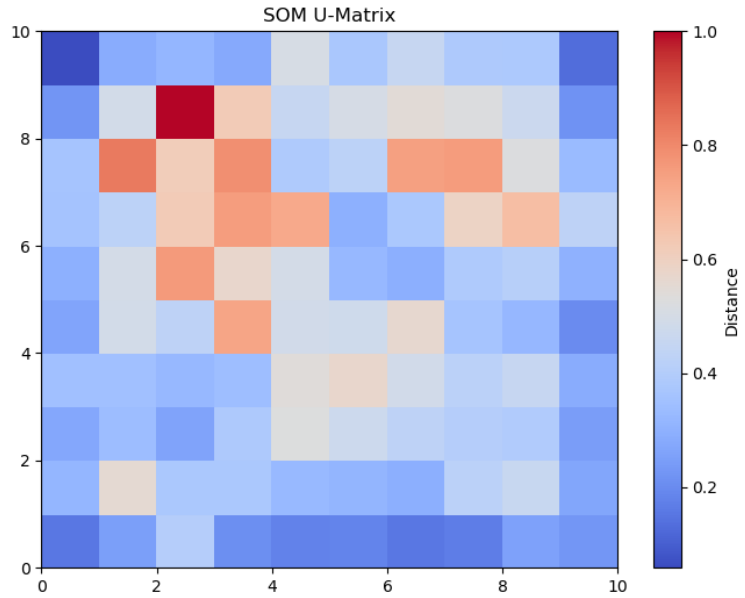
```
Original Categories:
[['Red']
 ['Green']
 ['Blue']
 ['Red']
 ['Blue']]

One-Hot Encoded Data:
[[1. 0. 0.]   # Red
 [0. 1. 0.]   # Green
 [0. 0. 1.]   # Blue
 [1. 0. 0.]   # Red
 [0. 0. 1.]]  # Blue
```

Self-Organizing Map (SOM)

A **Self-Organizing Map (SOM)** is a type of artificial neural network used for unsupervised learning. It projects high-dimensional data into a lower-dimensional (usually 2D) grid while preserving the topological structure of the data.

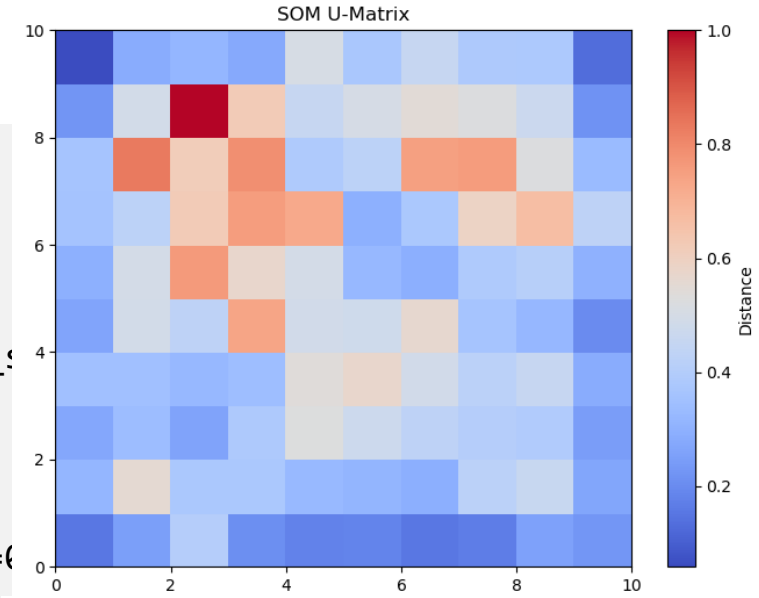
Key Characteristics of SOM:

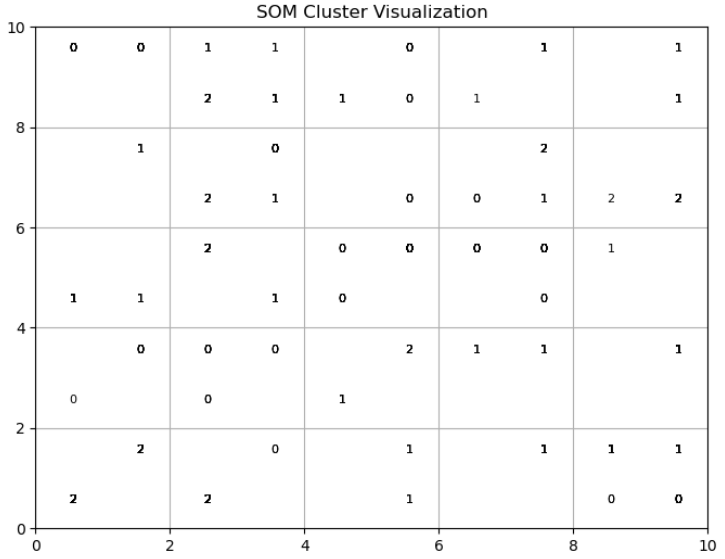
1. **Unsupervised Learning**: SOM learns without labeled data.
2. **Dimensionality Reduction**: Projects data into a lower-dimensional space (e.g., 2D grid).
3. **Topology Preservation**: Maintains the relationship between data points in the reduced space.



SOM (Example)

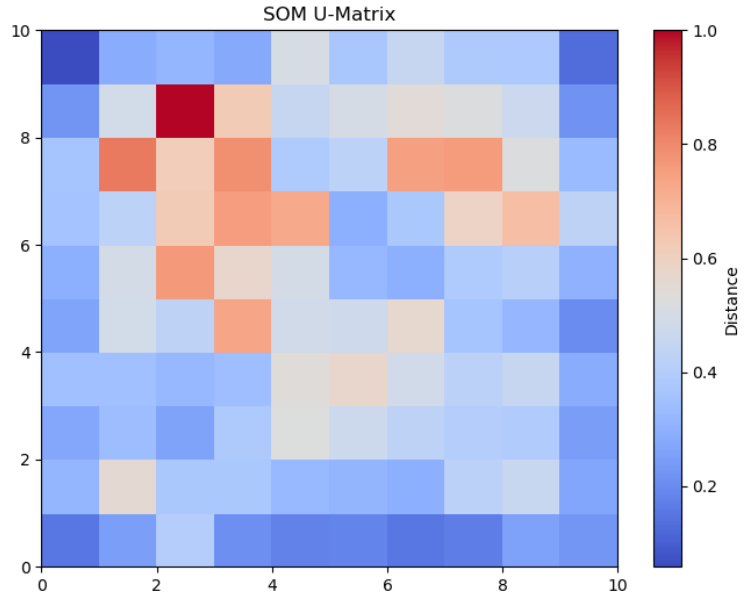
```
# Import
from minisom import MiniSom
from sklearn.datasets import make_blobs

# Dataset
data, labels = make_blobs(n_samples=500, centers=3, cluster_std=1.0, random_s
data_min, data_max = data.min(axis=0), data.max(axis=0)
data = (data - data_min) / (data_max - data_min)

# Initialize the SOM
som = MiniSom(x=10, y=10, input_len=data.shape[1], sigma=1.0, learning_rate=0
# Randomly initialize the weights
som.random_weights_init(data)
# Train the SOM
som.train_random(data, num_iteration=100)

# Visualize the SOM using a distance map (U-matrix)
plt.figure(figsize=(8, 6))
plt.pcolor(som.distance_map().T, cmap='coolwarm')  # Distance map as a heat
plt.colorbar(label='Distance')
plt.title("SOM U-Matrix")
plt.show()
```

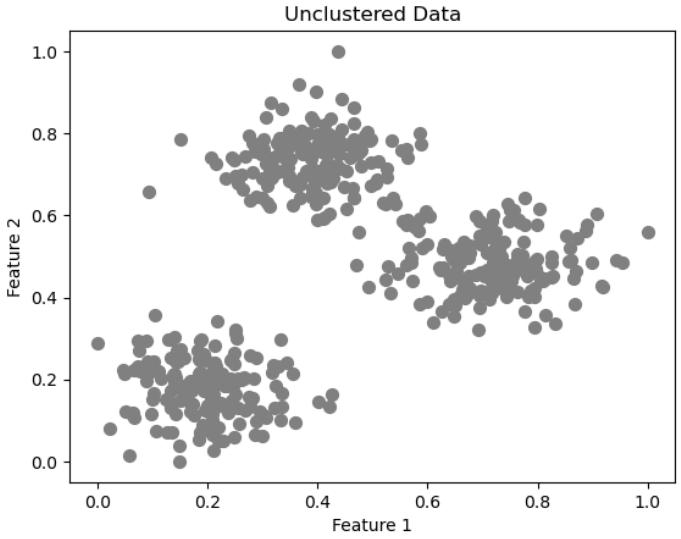




SOM Distance Map (Example)

The "U-Matrix" (Unified Distance Matrix), a visualization showing distances between the weight vectors of neighboring neurons on the SOM grid. It provides insight into the clustering structure of the data and the relationships between different regions of the map.



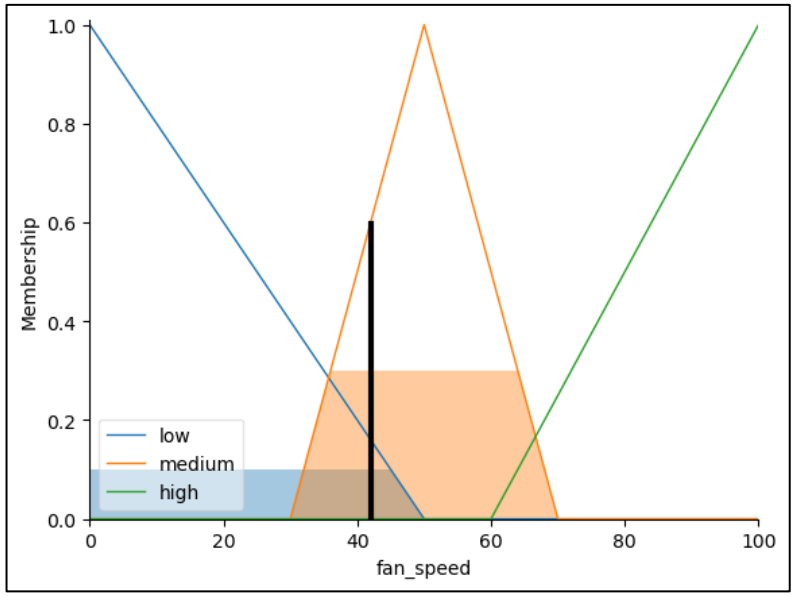


Key Points About the Distance Map:

1. **High Distance Areas**:
   1. Regions with large distances between neurons are typically represented by darker colors on the map.
   2. These areas indicate that the data points mapped to these neurons are quite distinct, forming boundaries between clusters.
2. **Low Distance Areas**:
   1. Regions with smaller distances between neurons are typically represented by lighter colors.
   2. These areas suggest that the neurons (and the corresponding data points) are very similar, potentially belonging to the same cluster.
3. **Clustering Insight**:
   1. The distance map is incredibly helpful in visually identifying the clusters in the data.
   2. For example, a cluster would appear as a "light island" surrounded by "dark borders," where the borders represent the separation between clusters.
4. **Interpretation**:
   1. By analyzing the U-Matrix, you can determine the number of clusters and how well-separated they are.
   2. The map can also reveal noisy or ambiguous regions where clusters are less distinct.

Fuzzy Logic

- Fuzzy logic is a mathematical approach to reasoning that resembles human decision-making by allowing for approximate values and reasoning rather than fixed, binary "true/false" logic.
- Widely used in control systems, pattern recognition, and machine learning.
- Unlike traditional binary logic, fuzzy logic deals with degrees of truth, where values range between 0 and 1.

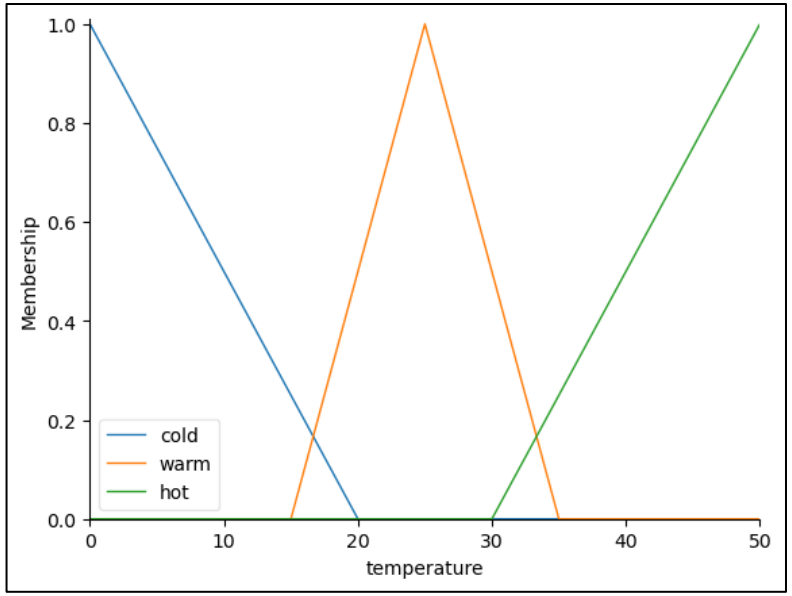


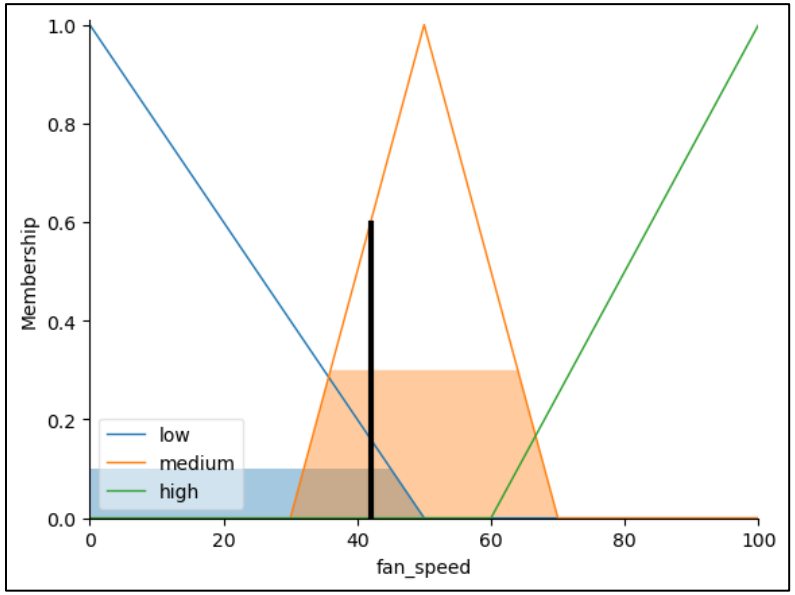
SOM (Example)

```
# Import
import skfuzzy as fuzz
from skfuzzy import control as ctrl
# Define fuzzy variables
temperature = ctrl.Antecedent(np.arange(0, 51, 1), 'temperature')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')
# Define membership functions
temperature['cold'] = fuzz.trimf(temperature.universe, [0, 0, 20])
temperature['warm'] = fuzz.trimf(temperature.universe, [15, 25, 35])
temperature['hot'] = fuzz.trimf(temperature.universe, [30, 50, 50])
fan_speed['low'] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [30, 50, 70])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [60, 100, 100])
# Define fuzzy rules
rule1 = ctrl.Rule(temperature['cold'], fan_speed['low'])
rule2 = ctrl.Rule(temperature['warm'], fan_speed['medium'])
rule3 = ctrl.Rule(temperature['hot'], fan_speed['high'])
# Create control system
fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fan = ctrl.ControlSystemSimulation(fan_ctrl)
# Simulate
fan.input['temperature'] = 18  # Input temperature
fan.compute()
print(f"Fan Speed: {fan.output['fan_speed']}")
```

```
Fan Speed:
41.9598997493734
```

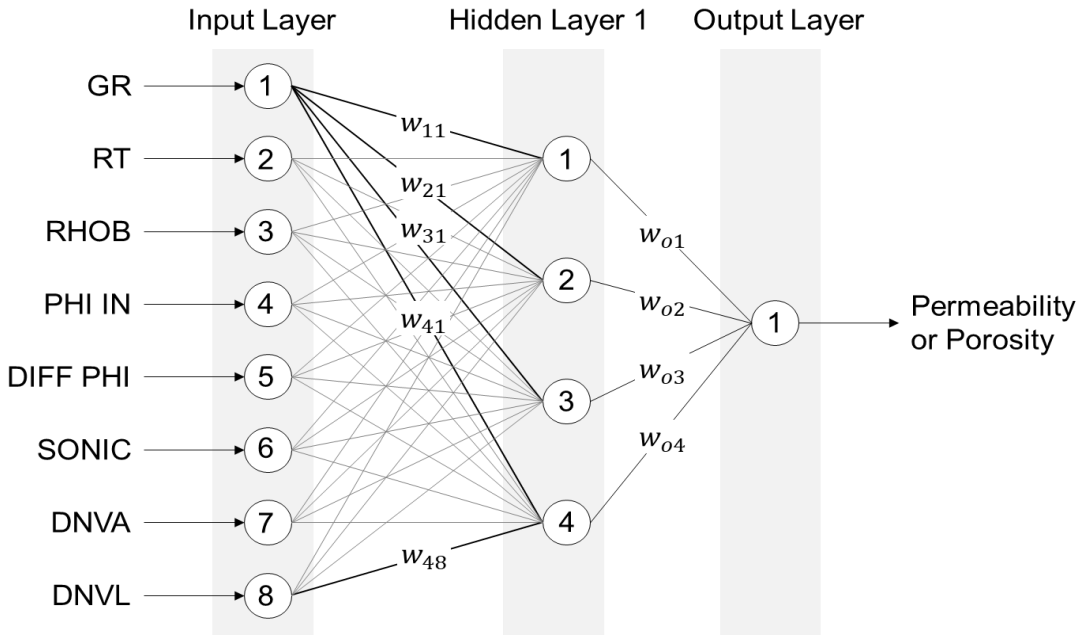
```
# Visualize results
temperature.view()
fan_speed.view()
fan_speed.view(sim=fan)
```





Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are inspired by the structure and function of the human brain. They consist of layers of neurons that can learn complex representations of data and capture non-linear patterns, making them suitable for various tasks



Input Layer: Takes input data and passes it to the network.

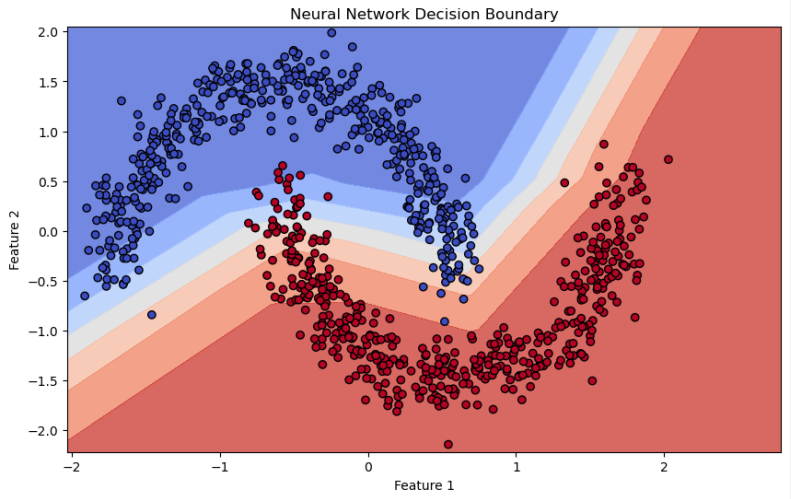
Hidden Layers: Perform intermediate computations and extract features.

Output Layer: Provides the model's predictions (e.g., probabilities or labels).

Activation Functions: Non-linear functions like ReLU, Sigmoid, or Softmax that help the network learn complex patterns.

Neural Network (Example)

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Generate a synthetic dataset
X, y = make_moons(n_samples=3000, noise=0.1, random_state=1234)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Standardize the data for better model performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build a simple neural network
model = Sequential([Dense(10, input_dim=2, activation='relu'),  # 10 neuron layer with ReLU activation
    Dense(1, activation='sigmoid') classification]) # Output layer with 1 neuron and Sigmoid activation
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metri
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, v

# Evaluate the model
accuracy = model.evaluate(X_test, y_test, verbose=0)[1]
print("Test Accuracy:", accuracy)
```




```
Test Accuracy: 0.9477777481079102
6461/6461 ━━━━━━━━━━━━━━━━━━━━ 8s 1ms/step
```

Convolutional Neural Networks (CNNs)

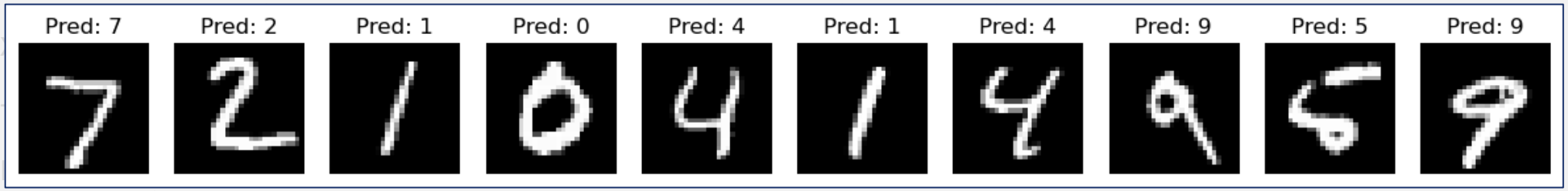
- Convolutional Neural Networks (CNNs) are specialized neural networks effective for processing imaging data. By using convolutions, CNNs can detect local patterns and spatial features, excelling in tasks such as image classification and object detection.

Convolutional Neural Network (Example)

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Preprocess the data
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255  # Reshape and normalize
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255    # Reshape and normalize
y_train = to_categorical(y_train, 10)  # One-hot encode labels
y_test = to_categorical(y_test, 10)    # One-hot encode labels

# Build the CNN model
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),  # Convolutional layer
    MaxPooling2D(pool_size=(2, 2)),  # Max pooling layer
    Flatten(),  # Flatten the 2D arrays into 1D
    Dense(128, activation='relu'),  # Fully connected dense layer
    Dense(10, activation='softmax') ])  # Output layer for 10 classes
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_accuracy)
```

Convolutional Neural Network (Example)

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Preprocess the data
X_train =
X_test =
y_train =
y_test =

# Build t
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),  # Convolutional layer
    p
    D
    F
# Co
mode

# Tr
hist

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_accuracy)
```



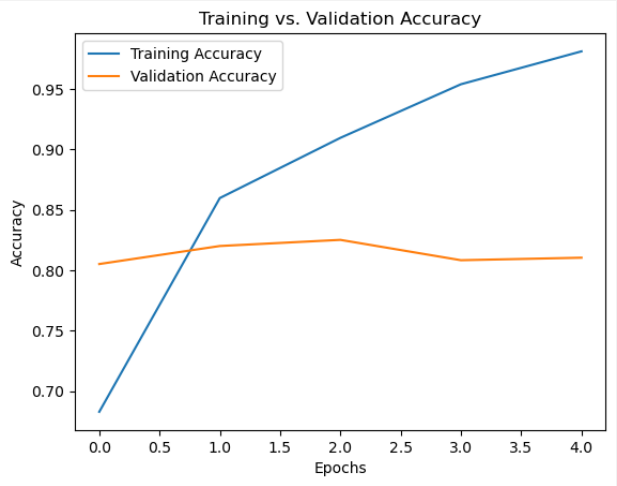

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - accuracy: 0.9103 - loss: 0.2960 - val_accuracy: 0.9795 - val_loss: 0.0588
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9849 - loss: 0.0503 - val_accuracy: 0.9833 - val_loss: 0.0508
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9905 - loss: 0.0297 - val_accuracy: 0.9873 - val_loss: 0.0401
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9935 - loss: 0.0203 - val_accuracy: 0.9846 - val_loss: 0.0477
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9955 - loss: 0.0145 - val_accuracy: 0.9856 - val_loss: 0.0475
Test Accuracy: 0.9855999946594238
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
```

Recurrent Neural Networks (RNNs)

- Recurrent Neural Networks (RNNs) are designed for sequential data such as time series and text. They have recurrent connections that allow them to maintain short-term memory, making them well-suited for tasks that involve temporal dependencies. They are commonly used in applications like time series forecasting, natural language processing, and speech recognition. However, RNNs suffer from issues like the **vanishing gradient problem**, which can make it difficult for them to learn long-term dependencies.

Recurrent Neural Network (Example)

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the IMDB dataset
vocab_size = 10000  # Limit the vocabulary size to the top 10,000 words
max_length = 100    # Limit each review to 100 words
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)
# Preprocess the data (pad sequences to have the same length)
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)

# Build the RNN model
model = Sequential([ Embedding(vocab_size, 32, input_length=max_length),  # Embedding layer
    SimpleRNN(32, activation='tanh'), # Simple RNN layer
    Dense(1, activation='sigmoid') ]) # Output layer for binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test), verbose=1)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_accuracy)
```



LSTM (Long Short-Term Memory)

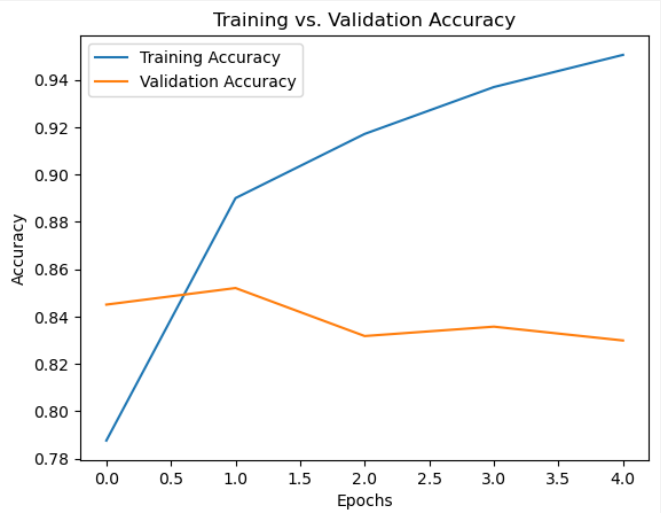
- LSTM is a special type of RNN designed to address the limitations of standard RNNs. It introduces memory cells and gates (input, output, and forget gates) that control the flow of information, allowing it to capture long-term dependencies more effectively. LSTMs are widely used in tasks like text generation, speech recognition, and language translation.

Why LSTMs?

- LSTMs address the limitations of traditional RNNs by using memory cells and gates (input, forget, and output gates).
- This allows them to retain important information over long sequences and discard irrelevant details

LSTM Neural Network (Example)

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Load the IMDB dataset
vocab_size = 10000  # Limit vocabulary size to top 10,000 words
max_length = 100    # Limit reviews to 100 words
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)

# Preprocess the data (pad sequences to have the same length)
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)

# Build the LSTM model
model = Sequential([
    Embedding(vocab_size, 32, input_length=max_length),   # Embedding layer
    LSTM(64, return_sequences=False, activation='tanh'),  # LSTM layer with 64 units
    Dense(1, activation='sigmoid')                        # Output layer for binary classification])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test), verbose=1)
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", test_accuracy)
```




```
391/391 ━━━━━━━━━━━━━━━━━━━━ 26s 67ms/step - accuracy: 0.9552 - loss: 0.1228 - val_accuracy: 0.8298 - val_loss: 0.4893
Test Accuracy: 0.8298400044441223
```

GPT

```
pip install langchain sentence-transformers faiss-cpu (if needed)
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.document_loaders import TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
import os

# Step 1: Load and Split Your Documents
folder_path = "data"
docs = []
for file_name in os.listdir(folder_path):
        if file_name.endswith(".txt"):
                loader = TextLoader(os.path.join(folder_path, file_name))
                docs.extend(loader.load())

# Step 2: Create Embeddings and Vector Store
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
chunks = splitter.split_documents(docs)
embed_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2") # Lightweight Model
vector_store = FAISS.from_documents(chunks, embed_model)

# Step 3: Ask a Question
query = "What was the pressure drop recorded in June?"results =
vector_store.similarity_search(query, k=3)

# Step 4: Print Top Answers
for i, doc in enumerate(results):
        print(f"\nResult {i+1}:\n{doc.page_content}")
```

Python Kick Start Program for PE

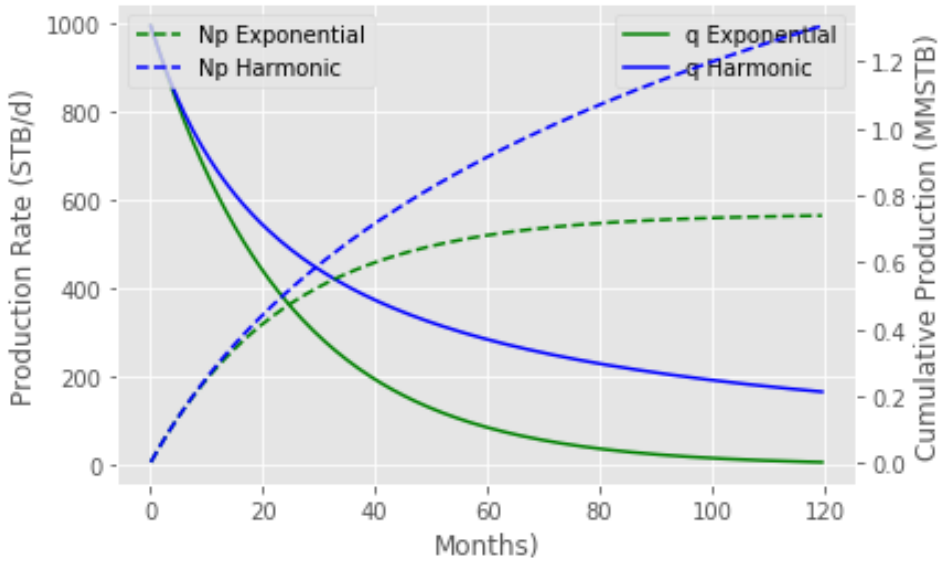
Petroleum Engineering Basics

1. Decline Curve analysis
2. Material Balance Equation
3. Well Nodal Analysis (IPR/VLP)
4. Wellbore Pressure Response
5. Probabilistic Oil Reserves

PE Basics: Decline Curve

```
import numpy as np
import matplotlib.pyplot as plt

# Input Data
t_1m = 1     # month
q_1m = 960   # stb/day
t_0m = 0     # month
q_0m = 1000 # stb/day
t = np.arange(0.1,120,0.5)

Exponential Decline
b = np.log(q_0m/q_1m)/(t_1m - t_0m)
q = q_0m  np.exp( -b  t ) # Forecast
Np = (q_0m - q)  365/(b 12)

Harmonic Decline
b2 = (q_0m/q_1m - 1)/(t_1m - t_0m)
q2 = q_0m / ( 1 + b2 t)
Np2 = (np.log(q_0m) - np.log(q2))
((q_0m 365) /(12b2))
```

```
# Plot Production and Cumulative vs Time (months)
ax = plt.subplot()
ax.plot(t, q, c = "g" ,  label= "q Exponential")
ax.plot(t, q2, c = "b" ,  label= "q Harmonic")
ax.set_xlabel("Months)")
ax.set_ylabel("Production Rate (STB/d)")
plt.legend()
ax2 = ax.twinx()
ax2.plot(t, Np/1000000, c = "g" , label = "Np Exponential", linestyle = "--")
ax2.plot(t, Np2/1000000, c = "b" , label = "Np Harmonic", linestyle = "--")
ax2.set_ylabel("Cumulative Production (MMSTB)")
plt.legend()
plt.grid(False)
```



https://github.com/Navdeep-Dhaka/Data-science-in-oil-and-gas

PE Basics: MBE Gas Cap (1/5)

$$F = N * (E_o + mE_g)$$

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Input Data
Pr = [3330,3150,3000,2850,2700,2550,2400] # psi
Np = [0,3.295,5.903,8.852,11.503,14.513,17.73] # MMSTB
Gp = [0,3460,6257,10268,14206,18359,23049] #MMSCF
Bo = [1.2511,1.2353,1.2222,1.2122,1.2022,1.1922,1.1822] # rb/stb
Bg = [0.00087,0.00092,0.00096,0.00101,0.00107,0.00113,0.0012] # rb/scf
Rs = [510,477,450,425,401,375,352] # scf/stb
Rsi = 510
Boi = 1.2511
Bgi = 0.00087
# Build Dataframe
df = pd.DataFrame({'Pr':Pr, 'Np':Np, 'Gp':Gp, 'Bo':Bo, 'Bg':Bg, 'Rs':Rs})

# Build Rp and add to df
df['Rp']= df.Gp/df.Np
# Build F, Eo and Eg and add to df
df['F'] = df['Np'](df['Bo'] + (df['Rp'] - df['Rs'])  df['Bg']) # MMbbl
df['Eo'] = df['Bo'] - Boi + (Rsi - df['Rs'] )  df['Bg'] # rb/stb
df['Eg'] = Boi (df['Bg']/Bgi - 1) # rb/stb
```

https://github.com/Navdeep-Dhaka/Data-science-in-oil-and-gas

Build a Data Frame with Input Data

Index	Pr	Np	Gp	Bo	Bg	Rs
0	3330	0	0	1.2511	0.00087	510
1	3150	3.295	3460	1.2353	0.00092	477
2	3000	5.903	6257	1.2222	0.00096	450
3	2850	8.852	10268	1.2122	0.00101	425
4	2700	11.503	14206	1.2022	0.00107	401
5	2550	14.513	18359	1.1922	0.00113	375
6	2400	17.73	23049	1.1822	0.0012	352

Compute additional columns in the same Data Frame (Rp, F, Eo and Eg)

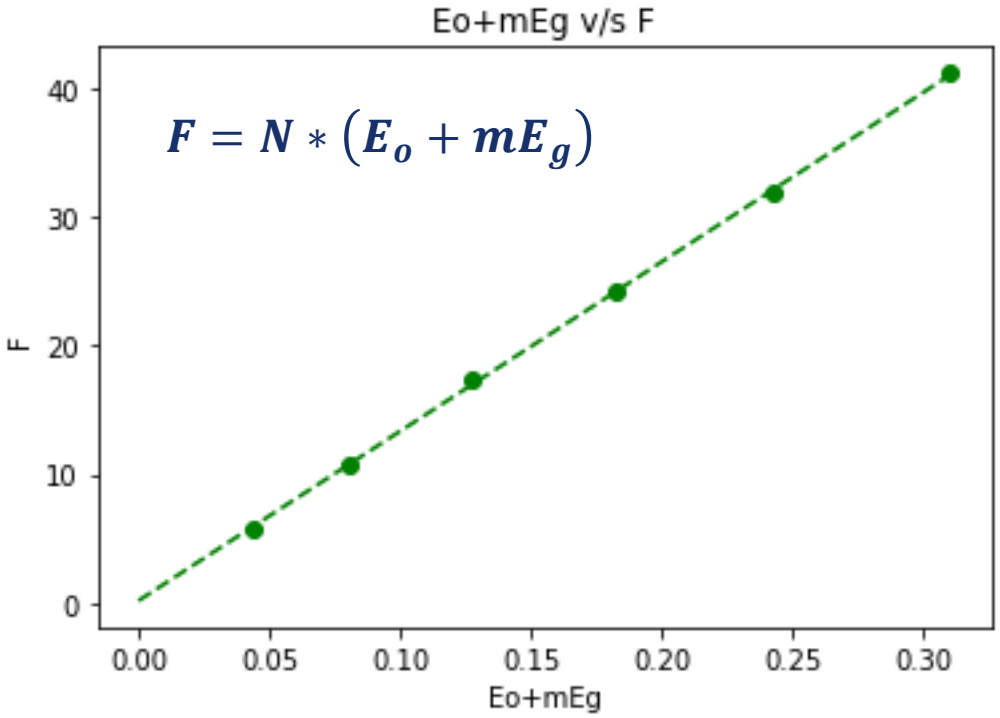
Rp	F	Eo	Eg
nan	nan	0	0
1050.08	5.80754	0.01456	0.0719023
1059.97	10.6713	0.0287	0.129424
1159.96	17.3014	0.04695	0.201326
1234.98	24.0937	0.06773	0.287609
1265	31.8982	0.09365	0.373892
1300	41.1301	0.1207	0.474555

PE Basics: MBE Gas Cap (2/5)

Case 1. N is unknown, m is known

```
# The slope of Eo+mEg v/s F curve give vale of OIIP
# Given m = 0.4
m = float(input('enter value of m = '))
mode = np.polyfit((Eo[1:7] + mEg[1:7]),F[1:7],1)

plt.plot(Eo + m  Eg, F, 'og')
plt.plot(Eo + m  Eg, mode[0](Eo + m  Eg) + mode[1] , 'g--')
plt.title('Eo+mEg v/s F' )
plt.xlabel('Eo+mEg')
plt.ylabel('F')
print(f'OIIP is {mode[0]:0.5} MMSTB for m = {m} ' )
```

```
enter value of m = 0.4
OIIP is 131.44 MMSTB for m = 0.4
```



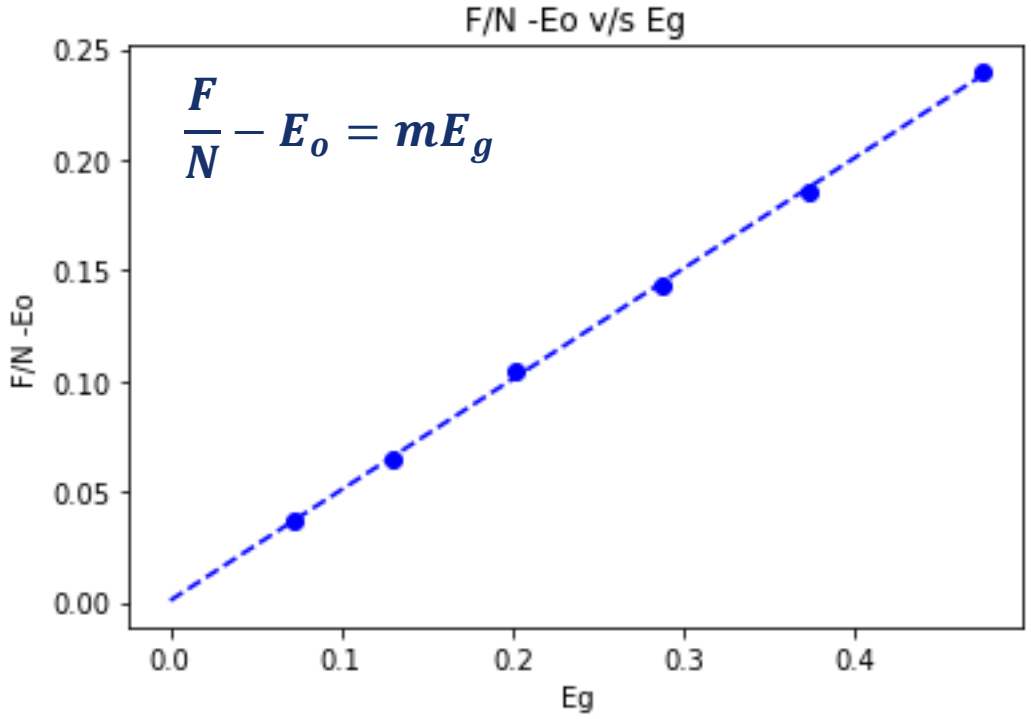
After https://github.com/Navdeep-Dhaka/Data-science-in-oil-and-gas

PE Basics: MBE Gas Cap (3/5)

Case 2. m is unknown and N is known

```
# The slope of (F/N - Eo) v/s Eg give value of m
# Given N is 114.2 MMSTB
N = float(input('enter value of OIIP in mmstb = '))
fit = np.polyfit(Eg[1:7],(F[1:7]/N - Eo[1:7]),1)

plt.plot(Eg, (F/N -Eo), 'ob')
plt.plot(Eg, fit[0]Eg + fit[1], 'b--') # fit[1]~0
plt.title('F/N -Eo v/s Eg' )
plt.xlabel('Eg')
plt.ylabel('F/N -Eo')
print(f' m is {fit[0]:0.3} for {N} MMSTB OIIP' )
```

```
enter value of OIIP in mmstb = 114.2
 m is 0.5 for 114.2 MMSTB OIIP
```



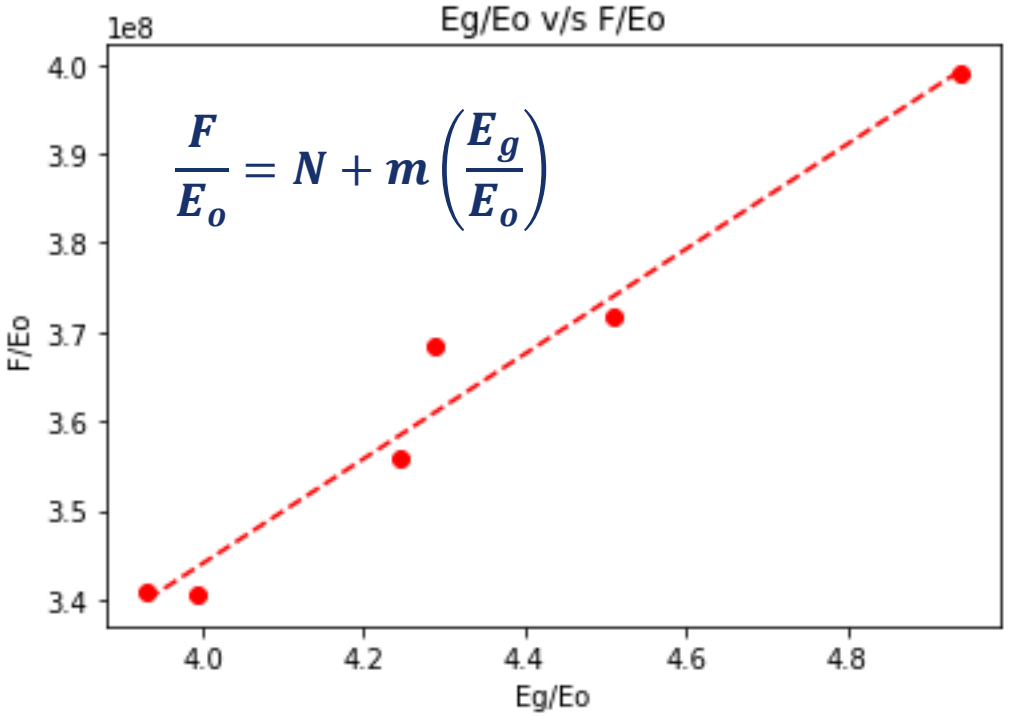
After https://github.com/Navdeep-Dhaka/Data-science-in-oil-and-gas

PE Basics: MBE Gas Cap (4/5)

Case 3. N and m are unknown

```
# The slope of (F/Eo) v/s Eg/Eo give value of m
df['F/Eo'] = df['F'] 106/df['Eo']
df['Eg/Eo'] = df['Eg']/df['Eo’]
model = np.polyfit(df['Eg/Eo'][1:7],df['F/Eo'][1:7],1)

plt.plot( df['Eg/Eo'], df['F/Eo'] , 'or')
plt.plot( df['Eg/Eo'], model[0]df['Eg/Eo']+model[1] , 'r--')
plt.title('Eg/Eo v/s F/Eo')
plt.xlabel('Eg/Eo')
plt.ylabel('F/Eo’)

print(f'OIIP is {model[1]/1000000:0.5} MMSTB for
m={model[0]/model[1]:0.3}')
```

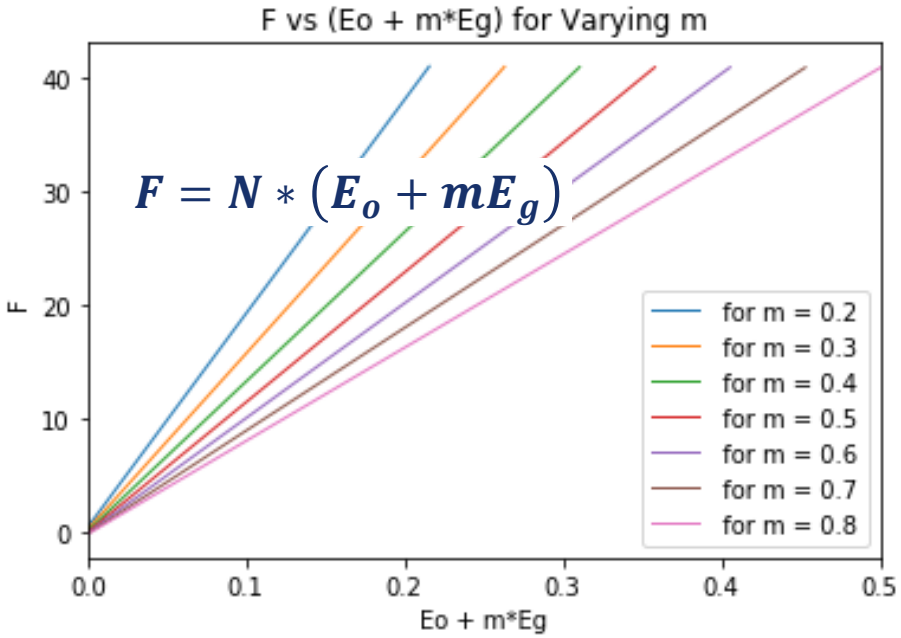
```
OIIP is 108.65 MMSTB for m = 0.542
```



After https://github.com/Navdeep-Dhaka/Data-science-in-oil-and-gas

PE Basics: MBE Gas Cap (5/5)

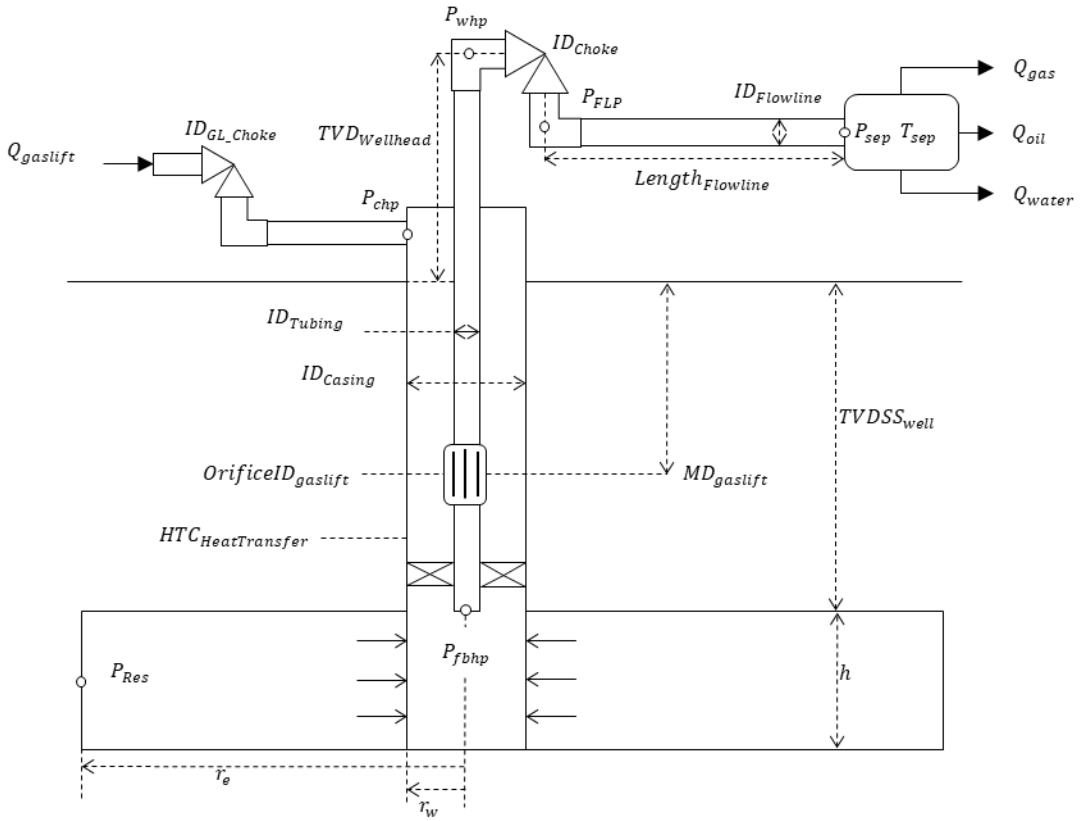
```
m = [0.2,0.3,0.4,0.5,0.6,0.7,0.8]
for i in m: df[f'Eo + {i}Eg'] = df['Eo'] + i  df['Eg']
plt.title('F vs (Eo + mEg) for Varying m')
plt.xlabel('Eo + mEg')
plt.ylabel('F')
plt.xlim(0,0.5)

for i in m:
    model = np.polyfit(df[f'Eo + {i}Eg'][1:7],df['F'][1:7],1)
    n_fitted = []
    for j in df[f'Eo + {i}Eg']:
        vv = model[0]j + model[1]
        n_fitted.append(vv)
    plt.plot(df[f'Eo + {i}Eg'], n_fitted, label = f' for m = {i}')
    plt.legend()
    print(f'OIIP is {model[0]:0.4} MMSTB for m = {i} ' )
```



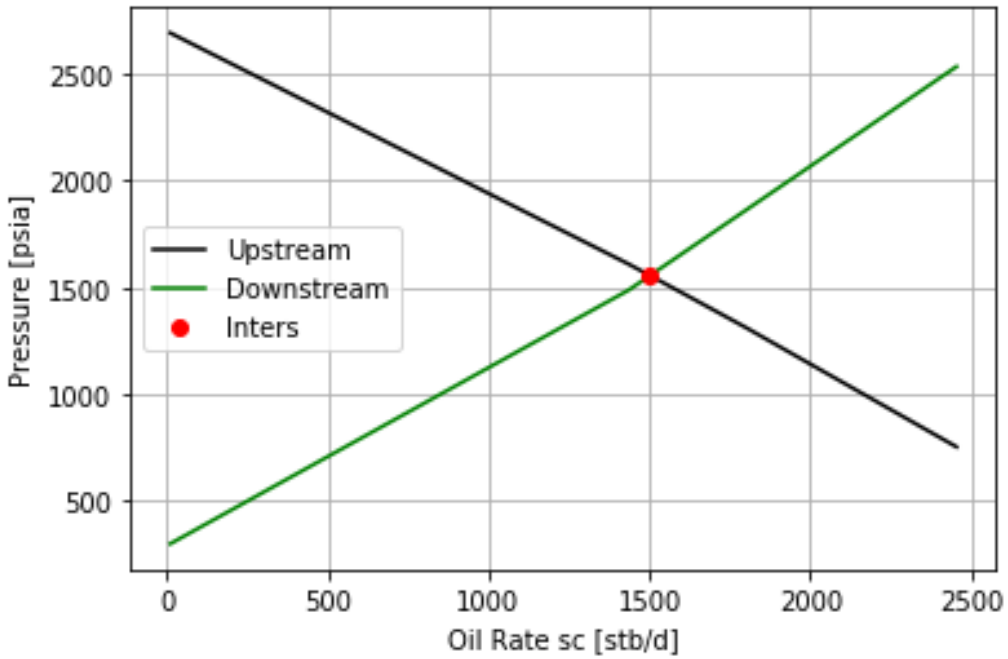
Index	Pr	Np	Gp	Bo	Bg	Rs	Rp	F	Eo	Eg	F/Eo	Eg/Eo	Eo + 0.2Eg	Eo + 0.3Eg	Eo + 0.4Eg	Eo + 0.5Eg	Eo + 0.6Eg	Eo + 0.7Eg	Eo + 0.8Eg
0	3330	0	0	1.2511	0.00087	510	nan	nan	0	0	nan	nan	0	0	0	0	0	0	0
1	3150	3.295	3460	1.2353	0.00092	477	1050.08	5.80754	0.01456	0.0719023	3.98869e+08	4.93834	0.0289405	0.0361307	0.0433209	0.0505111	0.0577014	0.0648916	0.0720818
2	3000	5.903	6257	1.2222	0.00096	450	1059.97	10.6713	0.0287	0.129424	3.71821e+08	4.50955	0.0545848	0.0675272	0.0804697	0.0934121	0.106354	0.119297	0.132239
3	2850	8.852	10268	1.2122	0.00101	425	1159.96	17.3014	0.04695	0.201326	3.68506e+08	4.2881	0.0872153	0.107348	0.127481	0.147613	0.167746	0.187879	0.208011
4	2700	11.503	14206	1.2022	0.00107	401	1234.98	24.0937	0.06773	0.287609	3.55732e+08	4.24641	0.125252	0.154013	0.182774	0.211535	0.240296	0.269056	0.297817
5	2550	14.513	18359	1.1922	0.00113	375	1265	31.8982	0.09365	0.373892	3.40611e+08	3.99244	0.168428	0.205818	0.243207	0.280596	0.317985	0.355374	0.392764
6	2400	17.73	23049	1.1822	0.0012	352	1300	41.1301	0.1207	0.474555	3.40763e+08	3.93169	0.215611	0.263067	0.310522	0.357978	0.405433	0.452889	0.500344

https://github.com/Navdeep-Dhaka/Data-science-in-oil-and-gas

PE Basics: Nodal Analysis



$$Q_{Liquid} = \left(\frac{k_{ro}}{\mu_o \beta_o} + \frac{k_{rw}}{\mu_w \beta_w}\right)\left(\frac{kh}{ln\frac{r_e}{r_w} + S}\right)\left(P_{Res} - P_{fbhp}\right)$$



https://github.com/Apress/machine-learning-oil-gas-industry

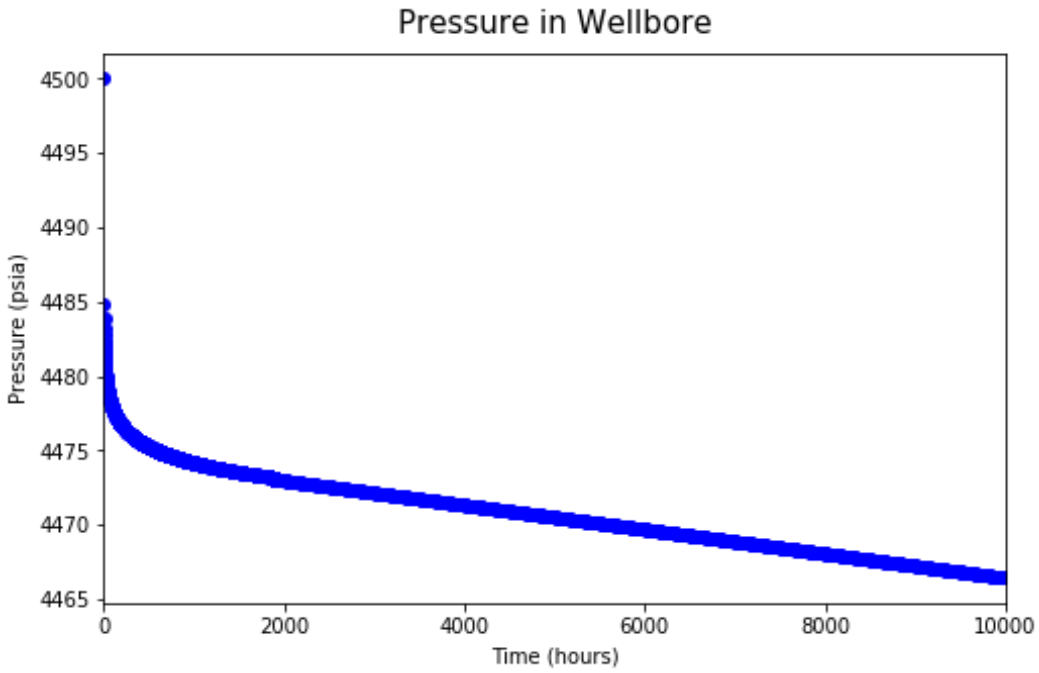
PE Basics: Wellbore Pressure Response

Constant-Rate Solutions: Bounded Cylindrical Reservoir

```
r_eD = re / rw          # condition for finite acting (fa)
t_Dw = 0.25  r_eD2
t_fa = (poro  mu_oil  ct  (rw2)  t_Dw)/(0.0002637  k)
tsat = (t_Dw  poro  ct  (rw_conv2))

if time[i] < t_fa:
   T_Dw = (0.000263ktime[i])/(poromu_oilct(rw2))
        if T_Dw > 100:
           P_D = 0.5 (((np.log(t_Dw)) + 0.80907))   # Eq 6.20
           Pwf = pi-((P_DqBomu_oil)/(0.007082kh))
        if T_Dw < 100: P_D = "NaN“ ; Pwf = "NaN"
elif time[i] >= t_fa:
    T_Dw = (0.000263ktime[i])/(poromu_oilct(rw2))
         if T_Dw > 25:
            P_D=(2T_Dw/(r_eD2))+np.log(r_eD)-0.75 # Eq 6.19
            Pwf = pi - ((P_DqBomu_oil)/(0.007082kh))
         if T_Dw < 25: P_D = "NaN“ ; Pwf = "NaN"
```

```
The well becomes FINITE ACTING after 1896.09 hours
```

SPE Textbook Vol. 8 Fundamental Principles of Reservoir Engineering by Brian F. Towler
After Nuwara, 2020. https://github.com/yohanesnuwara



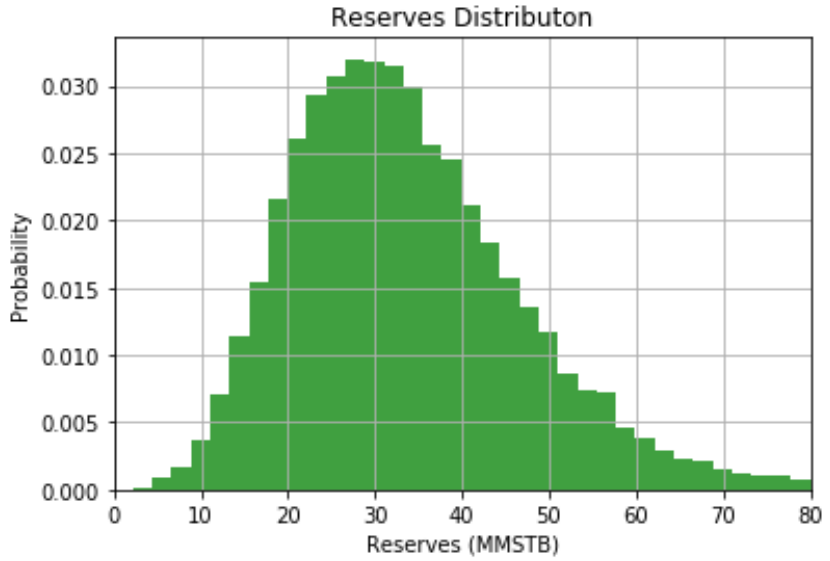

```
poro = 0.2     # Reservoir Porosity, fraction
ct = 20E-06    # Reservoir Compressibility, in psi^-1
k = 100        # Reservoir permeability, mD
rw = 8         # Wellbore radius, in
h = 1000       # Reservoir thickness, ft
mu_oil = 2     # Oil viscosity, cP
pi = 4500      # Reservoir initial pressure, psia
re = 5000      # Reservoir boundary radius, ft
q = 1000       # Oil flow rate, in STB/d
Bo = 1.1       # Oil FVF, in RB/STB
```

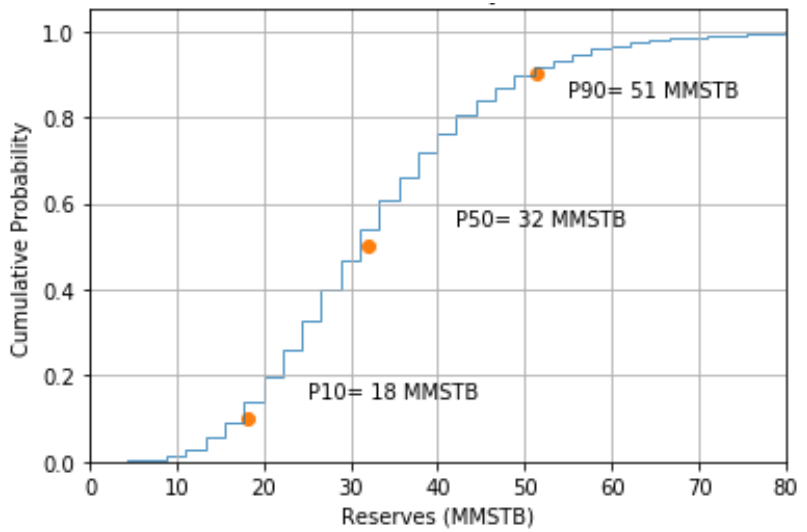
PE Basics: Probabilistic Reserves

Reserves calculations using the Monte Carlo simulation

```
from scipy.stats import norm, describe
import numpy as np

iterations = 10000
A = norm(640, 32).rvs(iterations)
h = norm(100, 20).rvs(iterations)
phi = norm(.2,.05).rvs(iterations)
Swi = norm(.25,.05).rvs(iterations)
NTG = norm(0.7,.2).rvs(iterations)
RF = norm(0.5,.1).rvs(iterations)
Bo = norm(1.1,.01).rvs(iterations)

Reserves = ( 7758  A  h  phi  (1-Swi)  RF / Bo ) / 1e6
n, mm, m, v, sk, kurt = describe(Reserves, axis=0)

# Calculate the P90, P50 and P10 values
P10 = np.percentile(Reserves,10)
P50 = np.percentile(Reserves,50)
P90 = np.percentile(Reserves,90)
```

After Rémy Ghalayini. https://github.com/rghalayini





Python Kick Start Program for PE

Python Basics

1. Print('hello'), Print(), /n
2. A=input('enter A=')
3. String to Numeric
4. String Count
5. Sentence Upper, Lower, Capitalize
6. Date and Time
7. Conditionals (if then elif)
8. Collections: Lists/arrays, dictionaries
9. Loops
10. Functions, Lambda
11. Managing the file system
12. Numpy
13. Create Pandas Data frame
14. Scikit Learn
15. Pandas Plot: Plot Volve Data Set

ML Algorithms

1. Regression
2. Decision Trees
3. Random Forest (RF)
4. Logistic Regression
5. Support Vector Machine (SVM)
6. Gradient Boosting (XGM, LGBM)
7. K-means
8. DBSCAN
9. K-Nearest Neighbor
10. Naive Bayes
11. Hierarchical Clustering
12. Mean shift Clustering
13. Association Algorithm (Apriori)
14. Principal Component Analysis (PCA)
15. One Hot Encoder
16. Self Organizing Maps (SOM)
17. Fuzzy Logic
18. Neural Networks
19. Convolutional Neural Networks (CNN)
20. Recurrent Neural Networks (RNN)
21. Long Short-Term Memory (LTSM)

Petroleum Engineering Basics

1. Decline Curve analysis
2. Material Balance Equation
3. Well Nodal Analysis (IPR/VLP)
4. Wellbore Pressure Response
5. Probabilistic Oil Reserves

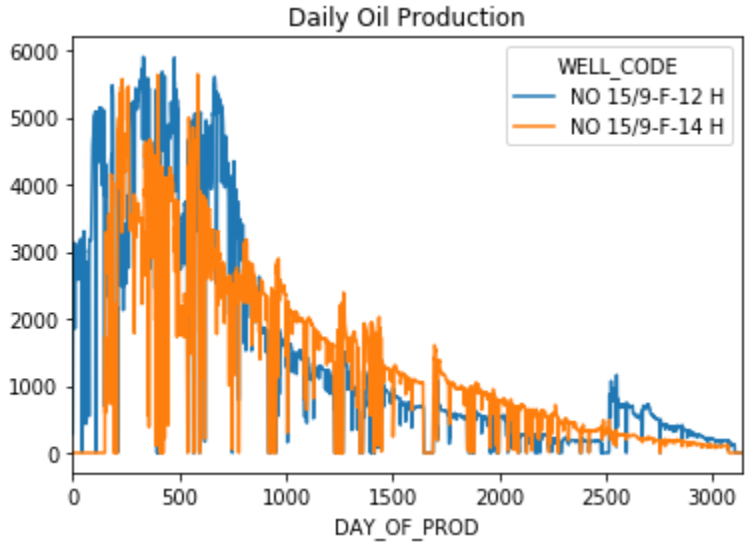
Machine Learning for VM

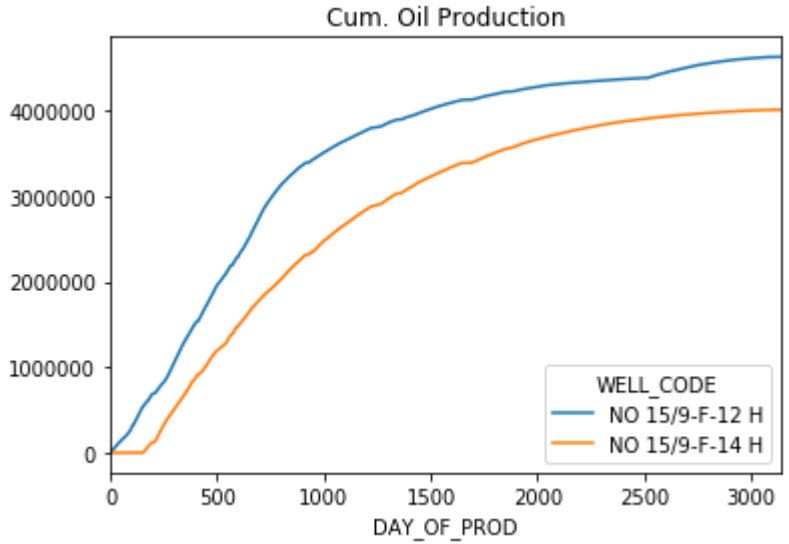
1. Define input and target (VM)
2. Split data for creating models
3. Clustering data
4. Model training and testing
5. Making predictions
6. Access public data sets

Python Basics: Pandas Plot

Create a Data Frame

```
import pandas as pd
df = pd.read_csv('../Volve_Oil_Production.csv')
well_codes = df.WELL_CODE.unique()
df = df.pivot(index='DAY_OF_PROD', columns='WELL_CODE', values=['OIL_PROD_VOL', 'CUM_OIL_PROD'])
df.plot(y='OIL_PROD_VOL', title='Daily Oil Production')
df.plot(y='CUM_OIL_PROD', title='Cum. Oil Production')
```





https://www.equinor.com/en/what-we-do/digitalisation-in-our-dna/volve-field-data-village-download.html

Public Oil, Gas & Energy Data Sets

Volve Field Data	https://www.equinor.com/en/what-we-do/digitalisation-in-our-dna/volve-field-data-village-download.html
FracFocus (chemicals, additives, vols)	http://fracfocus.org/
West Texas Lands (land and mineral)	http://www.utlands.utsystem.edu/
Alaska O&G Conservation Commission	https://www.commerce.alaska.gov/web/aogcc/Data.aspx
California Div. of O&G and Geothermal	http://www.conservation.ca.gov/dog/Pages/Index.aspx
Colorado O&G Conservation Commission	http://cogcc.state.co.us/
Oklahoma Corporation Commission	http://www.occeweb.com/og/oghome.htm
Texas Railroad Commission	http://www.rrc.state.tx.us/
Wyoming O&G Conservation Commission	http://wogcc.state.wy.us/
USGS National Map Viewer	http://viewer.nationalmap.gov/viewer/
Energy Information Agency	https://www.eia.gov/totalenergy/data/browser/
Department of Energy	https://www.energy.gov/data/open-energy-data
Data World (misc)	https://data.world/datasets/energy

Public Oil, Gas & Energy Data Sets

Poseidon NW Australia SD seismic & well logs	https://drive.google.com/drive/folders/0B7brcf-eGK8Cbk9ueHA0QUU4Zjg
World Stress Map – Crustal stress field	http://www.world-stress-map.org/
NOPIMS – Open petroleum geoscience data	https://nopims.dmp.wa.gov.au/nopims/
UK National Data Repository	https://ndr.ogauthority.co.uk/dp/controller/
Athabasca Oil Sands Well Dataset	https://ags.aer.ca/publication/spe-006
ICGEM – Hosts gravity field spherical	http://icgem.gfz-potsdam.de/home
TerraNubis –*Open Seismic Repository*	https://terranubis.com/datalist/free/
Quantarctica: QGIS basemap for Antarctica	https://www.npolar.no/quantarctica/
Digital Rocks Portal porous micro-structures	https://www.digitalrocksportal.org/
GSQ Open Data Portal –Queensland resource	https://geoscience.data.qld.gov.au/
GSQ GitHub Repository	https://github.com/geological-survey-of-queensland
Geoscience Australia Portal	https://portal.ga.gov.au/
SARIG – South Australian Resources	https://map.sarig.sa.gov.au/
SEG Open Data Catalog	https://wiki.seg.org/wiki/Open_data

Selected Industry Challenges

Department of Energy	• http://energychallenge.energy.gov/ • https://www.energy.gov/data/articles/think-outside-box-during-our-open-data-design-contest
Kaggle	• https://www.kaggle.com/tags/energy
Production Forecasting	• https://www.kaggle.com/c/datascienceatraisa
Predicting Oil Share Price	• https://www.kaggle.com/javierbravo/a-tour-of-the-oil-industry?scriptVersionId=1445335&cellId=1
Analytics Vidhya	• https://datahack.analyticsvidhya.com/contest/all/
SPE Colombia Hackathon Oil & Gas	• https://www.spe.org.co/hackathon-oilgas/

Some great help resources

ML Implementations in O&G - Success Stories & Challenges Ahead	https://seg.org/Events/SEG-Live/session/machine-learning-implementations-in-o-and-g
Cegal	https://www.youtube.com/user/bluebackreservoir https://github.com/cegaldev
Orkahub Energy	https://www.youtube.com/channel/UCPdJnvwDFA_7pDeE0H_jr4A https://github.com/orkahub
Yohanes Nuwara	https://github.com/yohanesnuwara
Awesome Reservoir Engineering (Marcelo Albuquerque)	https://github.com/mralbu/awesome-reservoir-engineering
APMonitor	http://apmonitor.com/ https://github.com/APMonitor
Petroleum From Scratch	https://www.youtube.com/channel/UC_lT10npISN5V32HDLAklsw

Get aware: SPE webinars

Data Driven Model for Anomaly Detection – Case Studies from Oil and Gas Applications	• https://webevents.spe.org/products/data-driven-model-for-anomaly-detection-case-studies-from-oil-and-gas-applications
AI/ML Drilling Systems Need Timely Trusted Data to Deliver Trusted Results	• https://webevents.spe.org/products/aiml-drilling-systems-need-timely-trusted-data-to-deliver-trusted-results
Data Science Project from End to End: A Sucker-Rod Pump Example	• https://webevents.spe.org/products/data-science-project-from-end-to-end-a-sucker-rod-pump-example
Smart Fields a data driven approach to making oil fields smart	• https://webevents.spe.org/products/smart-fields-a-data-driven-approach-to-making-oil-fields-smart
Using Machine Learning to optimize Completion design	• https://webevents.spe.org/products/**using-machine-learning-to-optimize-completion-design**
Developing Data Science Specialties Targeting Oil & Gas Industry	• https://webevents.spe.org/products/developing-data-science-specialties-targeting-oil-gas-industry
Application of Petroleum Data Analytics to Upstream Oil and Gas Use Cases	• https://webevents.spe.org/products/application-of-petroleum-data-analytics-to-upstream-oil-and-gas-use-cases

Get aware: SPE webinars

A Foundation for Petroleum Data Analytics	• https://webevents.spe.org/products/a-foundation-for-petroleum-data-analytics
Review of Data Analytics Techniques and Data Management Infrastructure	• https://webevents.spe.org/products/a-review-of-data-analytics-techniques-and-data-management-infrastructure
Machine Learning And Data Analytics In Flow Assurance	• https://webevents.spe.org/products/machine-learning-and-data-analytics-in-flow-assurance
Introduction to Data Analysis	• https://webevents.spe.org/products/introduction-to-data-analysis
Tools of Data Analysis (paid)	• https://webevents.spe.org/products/tools-of-data-analysis
Statistics as a Managerial Tool (paid)	• https://webevents.spe.org/products/statistics-as-a-managerial-tool
Data Analysis in the Real World (paid)	• https://webevents.spe.org/products/data-analysis-in-the-real-world

Get aware: SPE webinars

Whose data is it anyway?	• https://webevents.spe.org/products/whose-data-is-it-anyway
Rock Type Classification Using Machine Learning	• https://webevents.spe.org/products/rock-type-classification-using-machine-learning
From Data to Decisions - The Analytics Lifecycle	• https://webevents.spe.org/products/from-data-to-decisions-the-analytics-lifecycle
Data-Driven Analytics (6 sessions program)	• https://webevents.spe.org/products/data-driven-analytics#tab-product_tab_overview
Transforming E&P Applications through Big Data Analytics	• https://webevents.spe.org/products/transforming-ep-applications-through-big-data-analytics-2
PetroTalk: The Case for Incorporating Data Sciences	• https://webevents.spe.org/products/petrotalk-the-case-for-incorporating-data-sciences

Some Recommended online programs

Stanford University - Machine Learning	• https://www.coursera.org/learn/machine-learning
Python for Data Science and AI	• https://www.coursera.org/learn/python-for-applied-data-science-ai
Master of Science of Machine Learning (Imperial)	• https://www.coursera.org/degrees/msc-machine-learning-imperial
Applied Data Science with Python	• https://www.coursera.org/specializations/data-science-python
The R Programming Environment	• https://www.coursera.org/learn/r-programming-environment
Statistics and Machine Learning	• https://www.coursera.org/specializations/data-science-statistics-machine-learning
Data science courses on edX	• https://www.edx.org/course/subject/data-science
Data science courses on Udemy	• https://www.udemy.com/topic/data-science/
Data Science On DataCamp	• https://www.datacamp.com/
The School Of Data Science in Udacity	• https://www.udacity.com/school-of-data-science
Data Science in KDnuggets	• https://www.kdnuggets.com/
The Open Source Data Science Masters	• http://datasciencemasters.org/