1. Email

The Email Spam Classification project aims to automatically detect whether an email is spam or legitimate (ham). The dataset typically contains various text-based or numeric features extracted from email messages, such as word frequencies, sender details, or special characters. The process begins with importing the dataset and preprocessing it by removing null values or unwanted data. Cleaning ensures the machine learning model receives only relevant and meaningful inputs for accurate predictions.

After preprocessing, the dataset is divided into features (x) and labels (y), where x represents the independent variables (email features) and y represents the dependent variable (spam or not spam). Using train_test_split, the dataset is divided into training and testing subsets—commonly 75% for training and 25% for testing. This ensures the model is trained on a major portion of the data but tested on unseen examples to evaluate generalization performance.

The classification model (for example, Logistic Regression, Naive Bayes, or Random Forest) learns patterns in the training data. It analyzes how certain word combinations or numeric values correlate with spam messages. Once trained, the model predicts the class (spam or ham) for test emails. Evaluation metrics such as Accuracy, Precision, and Recall are calculated to measure model performance. Precision ensures that flagged spam messages are truly spam, while Recall checks that most spam messages are correctly identified.

Visualization tools like Confusion Matrix, Precision-Recall Curve, and ROC Curve help interpret the model's performance more clearly. These plots reveal trade-offs between false positives and false negatives, allowing for better tuning of model thresholds. The system can then be deployed in real-world email filters to automatically identify spam with high accuracy and reliability.

2. Churn Modelling

The Bank Churn Prediction project aims to identify customers likely to leave a bank (churn). The dataset used, *Churn_Modelling.csv*, contains information such as customer demographics, balance, credit score, and activity. The process starts with data preprocessing, including handling categorical variables using one-hot encoding for *Geography* and *Gender* columns, and dropping irrelevant columns such as *CustomerId* and *Surname*.

After cleaning, the data is split into features (X) and labels (y) where `y = Exited` (1 if customer left, 0 if not). The dataset is then divided into training and testing sets using `train_test_split`, followed by standardization with `StandardScaler` to normalize feature scales for model efficiency.

A Neural Network model is created using TensorFlow's Keras API with one hidden layer of 100 neurons and ReLU activation, followed by a sigmoid output for binary classification. The model is trained for 100 epochs using Adam optimizer and Binary Crossentropy loss.After training, predictions are compared with actual values to compute accuracy and visualize results via a confusion matrix.

Additionally, a Scikit-learn MLPClassifier is used for comparison to validate neural network performance.This project demonstrates the end-to-end pipeline for predicting customer churn using deep learning and traditional machine learning methods.

3. Gradient Descent Algorithm

The Gradient Descent project demonstrates how the optimization algorithm works by finding the local minima of the quadratic function y=(x+3)2y = (x + 3)^2y=(x+3)2.
Starting from an initial guess x=2x = 2x=2, the algorithm iteratively updates `x` using the formula:

xnew=xold−learning rate×dydxx_{new} = x_{old} - \text{learning rate} \times \frac{dy}{dx}xnew=xold−learning rate×dxdy

where dydx=2(x+3)\frac{dy}{dx} = 2(x + 3)dxdy=2(x+3).

The loop continues until the change in x is smaller than a specified precision (0.000001) or until the maximum number of iterations is reached. The process prints each iteration's value of `x` and eventually identifies the local minimum at x=−3x = -3x=−3.

A matplotlib scatter plot visualizes the descent curve of x-values versus y-values, showing how the algorithm converges towards the minimum point.This notebook helps understand the mathematical intuition and implementation of gradient descent, a foundational concept in optimization and machine learning training.

4. K-Means

The K-Means Clustering project groups sales data from *sales_data_sample.csv* to discover natural patterns among products and customers. The dataset includes fields like product lines, countries, deal sizes, and sales amounts.

Initial preprocessing includes dropping irrelevant columns (e.g., addresses, contact names, order numbers) and encoding categorical variables (*ProductLine*, *Country*, *DealSize*) using `LabelEncoder`.
After that, features are standardized with `StandardScaler` to ensure all variables contribute equally during clustering.

The Elbow Method is used to determine the optimal number of clusters by computing the Within-Cluster Sum of Squares (WCSS) for different k values (1–14). The resulting elbow plot helps identify the value of *k* where adding more clusters yields diminishing returns — typically around k=4.

The project illustrates how unsupervised learning can be used to segment data based on feature similarity, which is valuable for market segmentation, product categorization, or sales analysis.

5. <u>Uber</u>

The Uber Fare Prediction project analyzes and models trip fare data from *uber.csv* to predict fare amounts based on ride distance and other factors.

The workflow begins with data cleaning, removing irrelevant columns and null entries. Outliers are filtered using logical constraints on latitude, longitude, fare amount, and passenger count.
A custom distance calculation function is applied using the Haversine formula to compute trip distance from pickup and dropoff coordinates.

Date-time features are extracted — *year, month, hour, and weekday* — and categorized into weekday/weekend and time-of-day intervals. These engineered features provide better input for modeling.

After preprocessing, a correlation matrix and scatter plot reveal a strong positive relationship between distance and fare amount. Data is split into training and testing sets, and both features and targets are scaled with `StandardScaler`.

Two regression models are trained and compared:

- Linear Regression — to understand the linear relationship between distance and fare.
- Random Forest Regressor — to capture nonlinear patterns and improve accuracy.

Performance metrics such as R^2 score, RMSE, and MAE quantify model effectiveness.
This project demonstrates the process of transforming raw location data into meaningful insights and building predictive models for fare estimation — a practical application in transport analytics.

🧠 1. Bank Churn Prediction (Neural Network Classification)

Operation	Description	Time Complexity
Reading CSV	`pd.read_csv()` to load ~N rows, M columns	O(N·M)
One-Hot Encoding	For categorical columns like *Geography* and *Gender* (K categories)	O(N·K)
Dropping Columns	Removing unwanted columns	O(N)
Train-Test Split	Randomly partitioning dataset into subsets	O(N)
StandardScaler (fit_transform)	Compute mean, std for each feature and scale	O(N·M)
Neural Network Training (TensorFlow)	For L layers, H hidden units, epochs = E	O(E·N·H·M) (dominant)
Prediction (forward pass)	For N_test samples	O(N_test·H·M)
Confusion Matrix + Accuracy	Simple array comparisons	O(N_test)

Overall complexity:
✅ Training phase → O(E·N·H·M) (dominated by NN training)
✅ Inference phase → O(N·H·M)

🧮 2. Gradient Descent Example ($y = (x + 3)^2$)

Operation	Description	Time Complexity
Gradient Calculation	Single slope evaluation (`2*(x+3)`)	O(1)
Iterative Updates	Loop until	$x_{□} - x_{□-1}$
Plotting Points	Plot all points (≈T)	O(T)

Overall complexity:
✅ O(T) — linear in the number of iterations until convergence.
(For a convex quadratic function, T is small; convergence is guaranteed.)

🧩 3. K-Means Clustering (Sales Data)

Operation	Description	Time Complexity
Label Encoding / Cleaning	Categorical → numeric	O(N)
Standard Scaling	Normalize all M features	O(N·M)
K-Means (per iteration)	Compute distance between every point and k centroids	O(N·k·M)
K-Means (over I iterations)	Total cost for convergence	O(N·k·M·I)
Elbow Method Loop (1→15)	Repeats clustering 14 times	O(15·N·k·M·I) ≈ O(N·k·M·I) (constant factor 15)
Plotting Elbow Curve	Simple plotting	O(k)

Overall complexity:
✅ O(N·k·M·I) — dominates due to repeated distance computations per iteration.
K-Means scales linearly with dataset size but quadratically with k if very large.

🚕 4. Uber Fare Prediction (Regression Models)

Operation	Description	Time Complexity
Reading CSV & Cleaning	Dropping nulls and invalid coords	O(N)
Distance Calculation (Haversine)	For each row, constant arithmetic ops	O(N)
Feature Engineering (datetime extraction)	Extract Year, Month, Hour, etc.	O(N)
Label Encoding / Mapping (week_day, hour)	Apply functions to each element	O(N)
StandardScaler	Compute mean/std and scale	$O(N \cdot M)$
Train-Test Split	Random shuffle and split	O(N)
Linear Regression (training)	Solving Normal Equation or gradient descent	$O(M^3 + N \cdot M^2)$ (if closed form)
Random Forest (training)	For T trees, depth D	$O(T \cdot N \cdot \log N)$
Prediction (RF or LR)	Linear regression: $O(N \cdot M)$; Random forest: $O(T \cdot \log N)$	
Error Metrics (MSE, R^2)	Vector operations	O(N)

Overall complexity:
✅ Linear Regression: $O(M^3 + N \cdot M^2)$ (fast for small M).
✅ Random Forest: $O(T \cdot N \cdot \log N)$ (dominates overall runtime).

- Summary Table

Project	Dominant Operation	Approx. Big-O
Bank Churn Prediction	Neural Net training	O(E·N·H·M)
Gradient Descent	Iterative optimization	O(T)
K-Means Clustering	Distance computation loop	O(N·k·M·I)
Uber Fare Prediction	Random Forest training	O(T·N·log N)
