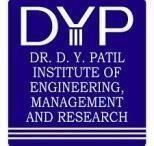


**Dr. D. Y. Patil Pratishthan’s**

**DR. D. Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT &**

**RESEARCH**

**Approved by A.I.C.T.E, New Delhi , Maharashtra State Government, Affiliated to Savitribai Phule Pune University**Sector No. 29, PCNTDA , Nigidi Pradhikaran, Akurdi, Pune 411044. Phone: 020–27654470, Fax: 020- 27656566 Website [:www.dypiemr.ac.in](http://www.dypiemr.ac.in/) Email : [principal.dypiemr@gmail.com](mailto:principal.dypiemr@gmail.com)



# Department of

**Artificial Intelligence and Data Science**

**LAB MANUAL**

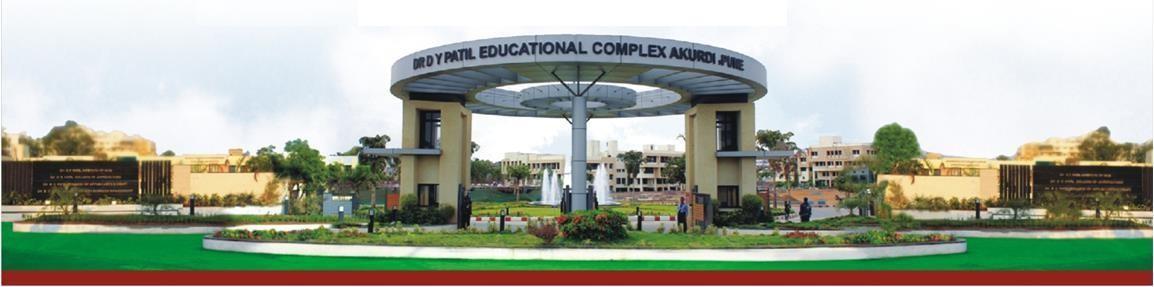
**Computer Laboratory I**

**Fourth Year Engineering (2020 Course) Semester-I**

**Prepared By: Dr. Suvarna Patil**

**Mrs. Shivganga Gavhane**

**Mrs. Rasika Kachore**



**417525: Computer Laboratory I**

| **Teaching Scheme Practical: 02 Hours/Week** | **Credit Scheme 01** | **Examination Scheme and Marks Term Work: 25 Marks** |
| --- | --- | --- |

**Companion Course: 417525: Computer Laboratory I**

**Course Objectives:**

● Apply regression, classification and clustering algorithms for creation of ML models

● Introduce and integrate models in the form of advanced ensembles

● Conceptualized representation of Data objects

● Create associations between different data objects, and the rules

● Organized data description, data semantics, and consistency constraints of data

**Course Outcomes:**

After completion of the course, learners should be able to-

CO1: Implement regression, classification and clustering models

CO2: Integrate multiple machine learning algorithms in the form of ensemble learning

CO3: Apply reinforcement learning and its algorithms for real world applications

CO4: Analyze the characteristics, requirements of data and select an appropriate data model

CO5: Apply data analysis and visualization techniques in the field of exploratory data science

CO6: Evaluate time series data

|  | Computer Laboratory I Lab Manual | | | BE-Sem 1 | (2020 | course) | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Table of Contents** | |  |  | |
|  | Sr.No |  | Title of the Experiment | | | | Page No |
| **Part I: Machine Learning** | | | | | | |
| 1 | A1 | **Feature Transformation (Any one)**  **A**. To use PCA Algorithm for dimensionality reduction. You have a dataset that includes measurements for different variables on wine (alcohol, ash, magnesium, and so on). Apply PCA algorithm & transform this data so that most variations in the measurements of the variables are captured by a small number of principal components so that it is easier to distinguish between red and white wine by inspecting these principal components. Dataset Link: <https://media.geeksforgeeks.org/wp-content/uploads/Wine.csv>  **B**. Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link:https://www.kaggle.com/datasets/uciml/iris | | | | 4 |
| 2 | A2 | **Regression Analysis:(Any one)**   1. Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset.   2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and ridge, Lasso regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: <https://www.kaggle.com/datasets/yasserh/uber-fares-dataset>   1. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets Dataset   link: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database | | | | 18 |
| 3 | A3 | **Classification Analysis (Any one)**   1. Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9). 2. Implement K-Nearest Neighbours’ algorithm on Social network ad dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.   Dataset link:https://www.kaggle.com/datasets/rakeshrau/social-network-ads | | | | 23 |
| 4 | A4 | **Clustering Analysis (Any one)**    **A.**Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method. Dataset Link: <https://www.kaggle.com/datasets/uciml/iris>  **B.** Implement K-Mediod Algorithm on a credit card dataset. Determine the number of clusters using the Silhouette Method. Dataset link: https://www.kaggle.com/datasets/arjunbhasin2013/ccdata | | | | 28 |
| 5 | A5 | **Ensemble Learning (Any one)**  **A.** Implement Random Forest Classifier model to predict the safety of the car. Dataset link: <https://www.kaggle.com/datasets/elikplim/car-evaluation-data-set>  **B.** Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient Tree Boosting (GBM), XGBoost classification on Iris dataset and compare the performance of three models using different evaluation measures. Dataset Link: https://www.kaggle.com/datasets/uciml/iris | | | | 32 |
| 6 | A6 | **Reinforcement Learning (Any one)**  **A.** Implement Reinforcement Learning using an example of a maze environment that the agent needs to explore.  **B.** Solve the Taxi problem using reinforcement learning where the agent acts as a taxi driver to pick up a passenger at one location and then drop the passenger off at their destination.  C. Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks a. Setting up the environment b. Defining the Tic-Tac-Toe game c. Building the reinforcement learning model d. Training the model e. Testing the model | | | | 41 |
| **Part II: Data Modeling and Visualization** | | | | | | |
| 7 | B1 | **Data Loading, Storage and File Formats**  **Problem Statement:** Analyzing Sales Data from Multiple File Formats  **Dataset:** Sales data in multiple file formats (e.g., CSV, Excel, JSON)  **Description:** The goal is to load and analyze sales data from different file formats, including CSV, Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset.  **Tasks to Perform:** Obtain sales data files in various formats, such as CSV, Excel, and JSON.  1. Load the sales data from each file format into the appropriate data structures or dataframes.  2. Explore the structure and content of the loaded data, identifying any inconsistencies, missing values, or data quality issue  3. Perform data cleaning operations, such as handling missing values, removing duplicates, or correcting inconsistencies.  4. Convert the data into a unified format, such as a common dataframe or data structure, to enable seamless analysis  5. Perform data transformation tasks, such as merging multiple datasets, splitting columns, or deriving new variables.  6. Analyze the sales data by performing descriptive statistics, aggregating data by specific variables, or calculating metrics such as total sales, average order value, or product category distribution.  7. Create visualizations, such as bar plots, pie charts, or box plots, to represent the sales data and gain insights into sales trends, customer behavior, or product performance. | | | | 47 |
| 8 | B2 | **Interacting with Web APIs**  **Problem Statement:** Analyzing Weather Data from OpenWeatherMap API    **Dataset:** Weather data retrieved from OpenWeatherMap API  **Description**: The goal is to interact with the OpenWeatherMap API to  retrieve weather data for a specific location and perform data modeling and visualization to analyze weather patterns over time.  **Tasks to Perform:**  1. Register and obtain API key from OpenWeatherMap.  2. Interact with the OpenWeatherMap API using the API key to retrieve weather data for a specific location.  3. Extract relevant weather attributes such as temperature, humidity, wind speed, and precipitation from the API response.  4. Clean and preprocess the retrieved data, handling missing values or inconsistent formats.  5. Perform data modeling to analyze weather patterns, such as calculating average temperature, maximum/minimum values, or trends over time.  6. Visualize the weather data using appropriate plots, such as line charts, bar plots, or scatter plots, to represent temperature changes, precipitation levels, or wind speed variations.  7. Apply data aggregation techniques to summarize weather statistics by specific time periods (e.g., daily, monthly, seasonal).  8. Incorporate geographical information, if available, to create maps or geospatial visualizations representing weather patterns across different locations.  9. Explore and visualize relationships between weather attributes, such as temperature and humidity, using correlation plots or heatmaps. | | | | 50 |
| 9 | B3 | **Data Cleaning and Preparation**  **Problem Statement**: Analyzing Customer Churn in a Telecommunications Company  **Dataset:** "Telecom\_Customer\_Churn.csv"  **Description:** The dataset contains information about customers of a telecommunications company and whether they have churned (i.e., discontinued their services). The dataset includes various attributes of the customers, such as their demographics, usage patterns, and account information. The goal is to perform data cleaning and preparation to gain insights into the factors that contribute to customer churn.  **Tasks to Perform:**  1. Import the "Telecom\_Customer\_Churn.csv" dataset.  2. Explore the dataset to understand its structure and content.  3. Handle missing values in the dataset, deciding on an appropriate strategy. 4. Remove any duplicate records from the dataset.  5. Check for inconsistent data, such as inconsistent formatting or spelling variations, and standardize it.  6. Convert columns to the correct data types as needed.  7. Identify and handle outliers in the data.  8. Perform feature engineering, creating new features that may be relevant to predicting customer churn.  9. Normalize or scale the data if necessary.  10. Split the dataset into training and testing sets for further analysis. 11. Export the cleaned dataset for future analysis or modeling. | | | | 56 |
|  | 10 | B4 | **Data Wrangling**  **Problem Statement:** Data Wrangling on Real Estate Market  **Dataset:** "RealEstate\_Prices.csv"  **Description:** The dataset contains information about housing prices in a specific real estate market. It includes various attributes such as property characteristics, location, sale prices, and other relevant features. The goal is to perform data wrangling to gain insights into the factors influencing housing prices and prepare the dataset for further analysis or modeling.  **Tasks to Perform:**  1. Import the "RealEstate\_Prices.csv" dataset. Clean column names by removing spaces, special characters, or renaming them for clarity.  2. Handle missing values in the dataset, deciding on an appropriate strategy (e.g., imputation or removal).  3. Perform data merging if additional datasets with relevant information are available (e.g., neighborhood demographics or nearby amenities).  4. Filter and subset the data based on specific criteria, such as a particular time period, property type, or location.  5. Handle categorical variables by encoding them appropriately (e.g., one-hot encoding or label encoding) for further analysis.  6. Aggregate the data to calculate summary statistics or derived metrics such as average sale prices by neighborhood or property type.  7. Identify and handle outliers or extreme values in the data that may affect the analysis or modeling process | | | |  |
|  | 11 | B5 | **Data Visualization using matplotlib**  **Problem Statement:** Analyzing Air Quality Index (AQI) Trends in a City  **Dataset:** "City\_Air\_Quality.csv"  **Description:** The dataset contains information about air quality measurements in a specific city over a period of time. It includes attributes such as date, time, pollutant levels (e.g., PM2.5, PM10, CO), and the Air Quality Index (AQI) values. The goal is to use the matplotlib library to create visualizations that effectively represent the AQI trends and patterns for different pollutants in the city.  **Tasks to Perform:**  1. Import the "City\_Air\_Quality.csv" dataset.  2. Explore the dataset to understand its structure and content.  3. Identify the relevant variables for visualizing AQI trends, such as date, pollutant levels, and AQI values.  4. Create line plots or time series plots to visualize the overall AQI trend over time.  5. Plot individual pollutant levels (e.g., PM2.5, PM10, CO) on separate line plots to visualize their trends over time.  6. Use bar plots or stacked bar plots to compare the AQI values across different dates or time periods.  7. Create box plots or violin plots to analyze the distribution of AQI values for different pollutant categories.  8. Use scatter plots or bubble charts to explore the relationship between AQI values and pollutant levels.  9. Customize the visualizations by adding labels, titles, legends, and appropriate color schemes | | | |  |
|  | 12 | B6 | **Data Aggregation**  **Problem Statement:** Analyzing Sales Performance by Region in a Retail Company  **Dataset:** "Retail\_Sales\_Data.csv"  **Description:** The dataset contains information about sales transactions in a retail company. It includes attributes such as transaction date, product category, quantity sold, and sales amount. The goal is to perform data aggregation to analyze the sales performance by region and identify the top-performing regions.  **Tasks to Perform:**  1. Import the "Retail\_Sales\_Data.csv" dataset.  2. Explore the dataset to understand its structure and content.  3. Identify the relevant variables for aggregating sales data, such as region, sales amount, and product category.  4. Group the sales data by region and calculate the total sales amount for each region.  5. Create bar plots or pie charts to visualize the sales distribution by region.  6. Identify the top-performing regions based on the highest sales amount.  7. Group the sales data by region and product category to calculate the total sales amount for each combination.  8. Create stacked bar plots or grouped bar plots to compare the sales amounts across different regions and product categories. | | | |  |
|  | 13 | B7 | **Time Series Data Analysis**  **Problem statement**: Analysis and Visualization of Stock Market Data  **Dataset:** "Stock\_Prices.csv"  **Description:** The dataset contains historical stock price data for a particular company over a period of time. It includes attributes such as date, closing price, volume, and other relevant features. The goal is to perform time series data analysis on the stock price data to identify trends, patterns, and potential predictors, as well as build models to forecast future stock prices.  **Tasks to Perform:**  1. Import the "Stock\_Prices.csv" dataset.  2. Explore the dataset to understand its structure and content.  3. Ensure that the date column is in the appropriate format (e.g., datetime) for time series analysis. 4. Plot line charts or time series plots to visualize the historical stock price trends over time.  5. Calculate and plot moving averages or rolling averages to identify the underlying trends and smooth out noise.  6. Perform seasonality analysis to identify periodic patterns in the stock prices, such as weekly, monthly, or yearly fluctuations.  7. Analyze and plot the correlation between the stock prices and other variables, such as trading volume or market indices.  8. Use autoregressive integrated moving average (ARIMA) models or exponential smoothing models to forecast future stock prices | | | |  |
|  |  |  | **Part III: Mini Project(Mandatory Assignments)** | | | |  |
|  |  |  | Mini Project (Mandatory- Group Activity)  It is recommended that group of 3 to 5 students should undergo a mini project (considering  the Machine Learning and Data modeling and Visualizing concepts) as content beyond  syllabus. Some of the problem statements are mentioned below:  1. Development of a happiness index for schools (including mental health and well-being  parameters, among others) with self-assessment facilities.  2. Automated Animal Identification and Detection of Species  3. Sentimental analysis on Govt. Released Policies  4. Identification of Flood Prone Roads  5. Identification of Missing Bridges which would increase the connectivity between regions  Note: Instructor can also assign similar problem statements  References:  For Dataset https://data.gov.in/  For Problem statements: https://sih.gov.in/sih2022PS | | | |  |

**EXPERIMENT NO. 1 A (Group A)**

* + **Aim:** To use PCA Algorithm for dimensionality reduction. You have a dataset that includes measurements for different variables on wine (alcohol, ash, magnesium, and so on). Apply PCA algorithm & transform this data so that most variations in the measurements of the variables are captured by a small number of principal components so that it is easier to distinguish between red and white wine by inspecting these principal components
  + **Outcome: At end of this experiment, student will be able understand the scheduler, and how its behaviour influences the performance of the system**
  + **Hardware Requirement:**
* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures
  + **Software Requirement:**

Jupyter Nootbook/Ubuntu

* + **Theory:**

## Principal Component Analysis (PCA)

PCA is an unsupervised [machine learning algorithm](https://www.analyticssteps.com/blogs/top-10-machine-learning-algorithms). PCA is mainly used for dimensionality reduction in a dataset consisting of many variables that are highly correlated or lightly correlated with each other while retaining the variation present in the dataset up to a maximum extent.

It is also a great tool for exploratory data analysis for making predictive models.

PCA performs a linear transformation on the data so that most of the variance or information in your high-dimensional dataset is captured by the first few principal components. The first principal component will capture the most variance, followed by the second principal component, and so on.

Each principal component is a linear combination of the original variables. Because all the principal components are orthogonal to each other, there is no redundant information. So, the total variance in the data is defined as the sum of the variances of the individual component. So decide the total number of principal components according to cumulative variance ‘‘explained’’ by them.

**Implementation:**

**import** pandas **as** pd  
**from** sklearn.decomposition **import** PCA  
**from** sklearn.preprocessing **import** StandardScaler  
**import** matplotlib.pyplot **as** plt

df = pd.read\_csv("C:/Users/HP/Dropbox/PC/Downloads/Wine.csv")

df.keys()

print(df['DESCR'])

df.head(5)

Alcohol Malic\_Acid Ash Ash\_Alcanity Magnesium Total\_Phenols \  
0 14.23 1.71 2.43 15.6 127 2.80   
1 13.20 1.78 2.14 11.2 100 2.65   
2 13.16 2.36 2.67 18.6 101 2.80   
3 14.37 1.95 2.50 16.8 113 3.85   
4 13.24 2.59 2.87 21.0 118 2.80   
  
 Flavanoids Nonflavanoid\_Phenols Proanthocyanins Color\_Intensity Hue \  
0 3.06 0.28 2.29 5.64 1.04   
1 2.76 0.26 1.28 4.38 1.05   
2 3.24 0.30 2.81 5.68 1.03   
3 3.49 0.24 2.18 7.80 0.86   
4 2.69 0.39 1.82 4.32 1.04   
  
 OD280 Proline Customer\_Segment   
0 3.92 1065 1   
1 3.40 1050 1   
2 3.17 1185 1   
3 3.45 1480 1   
4 2.93 735 1

df.Customer\_Segment.unique()

array([1, 2, 3], dtype=int64)

print(df.isnull().sum()) *#checking is null*

Alcohol 0  
Malic\_Acid 0  
Ash 0  
Ash\_Alcanity 0  
Magnesium 0  
Total\_Phenols 0  
Flavanoids 0  
Nonflavanoid\_Phenols 0  
Proanthocyanins 0  
Color\_Intensity 0  
Hue 0  
OD280 0  
Proline 0  
Customer\_Segment 0  
dtype: int64

X = df.drop('Customer\_Segment', axis=1) *# Features*  
y = df['Customer\_Segment'] *# Target variable*

**for** col **in** X.columns:  
 sc = StandardScaler() *#Standardize features by removing the mean and scaling to unit variance.z = (x - u) / s mean=0, Stddeviation=1*  
 X[col] = sc.fit\_transform(X[[col]]) *#Fit to data, then transform it.Compute the mean and std to be used for later scaling.*

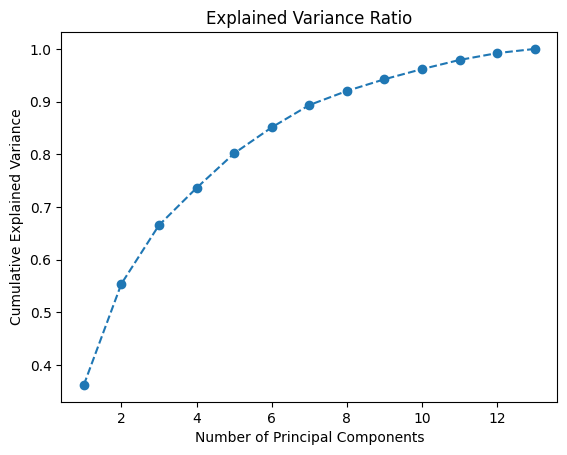
X.head(5)

Alcohol Malic\_Acid Ash Ash\_Alcanity Magnesium Total\_Phenols \  
0 1.518613 -0.562250 0.232053 -1.169593 1.913905 0.808997   
1 0.246290 -0.499413 -0.827996 -2.490847 0.018145 0.568648   
2 0.196879 0.021231 1.109334 -0.268738 0.088358 0.808997   
3 1.691550 -0.346811 0.487926 -0.809251 0.930918 2.491446   
4 0.295700 0.227694 1.840403 0.451946 1.281985 0.808997   
  
 Flavanoids Nonflavanoid\_Phenols Proanthocyanins Color\_Intensity \  
0 1.034819 -0.659563 1.224884 0.251717   
1 0.733629 -0.820719 -0.544721 -0.293321   
2 1.215533 -0.498407 2.135968 0.269020   
3 1.466525 -0.981875 1.032155 1.186068   
4 0.663351 0.226796 0.401404 -0.319276   
  
 Hue OD280 Proline   
0 0.362177 1.847920 1.013009   
1 0.406051 1.113449 0.965242   
2 0.318304 0.788587 1.395148   
3 -0.427544 1.184071 2.334574   
4 0.362177 0.449601 -0.037874

pca = PCA()  
X\_pca = pca.fit\_transform(X)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

plt.plot(range(1, len(explained\_variance\_ratio) + 1), explained\_variance\_ratio.cumsum(), marker='o', linestyle='--')  
plt.xlabel('Number of Principal Components')  
plt.ylabel('Cumulative Explained Variance')  
plt.title('Explained Variance Ratio')  
plt.show()

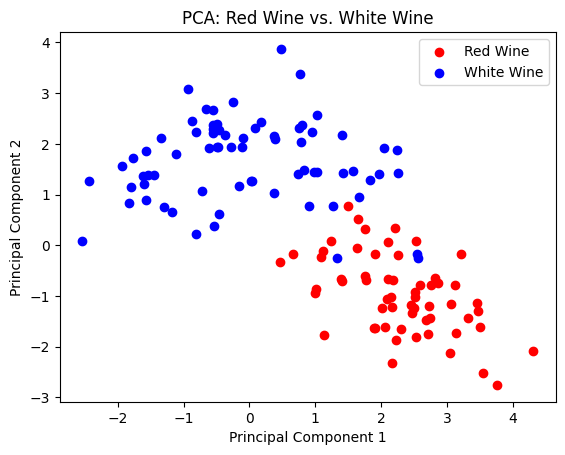


n\_components = 12 *# Choose the desired number of principal components you want to reduce a dimention to*  
pca = PCA(n\_components=n\_components)  
X\_pca = pca.fit\_transform(X)

X\_pca.shape

X.shape

red\_indices = y[y == 1].index  
white\_indices = y[y == 2].index  
  
plt.scatter(X\_pca[red\_indices, 0], X\_pca[red\_indices, 1], c='red', label='Red Wine')  
plt.scatter(X\_pca[white\_indices, 0], X\_pca[white\_indices, 1], c='blue', label='White Wine')  
plt.xlabel('Principal Component 1')  
plt.ylabel('Principal Component 2')  
plt.legend()  
plt.title('PCA: Red Wine vs. White Wine')  
plt.show()



*#Conclusion: Here we have reduce the dimention now we can able to apply any algorithm like classification, Regression etc.*

**EXPERIMENT NO. 1 B**

* **Aim**: Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link:https://www.kaggle.com/datasets/uciml/iris
* **Hardware Requirement:**
* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures
* **Software Requirement:**

Jupyter Nootbook/Ubuntu

* **Theory:**

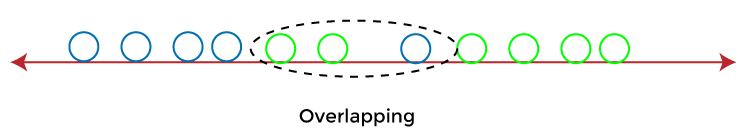
*Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).*

This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs. In this topic, "Linear Discriminant Analysis (LDA) in machine learning”, we will discuss the LDA algorithm for classification predictive modeling problems, limitation of logistic regression, representation of linear Discriminant analysis model, how to make a prediction using LDA, how to prepare data for LDA, extensions to LDA and much more. So, let's start with a quick introduction to Linear Discriminant Analysis (LDA) in machine learning.

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

*Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning*. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

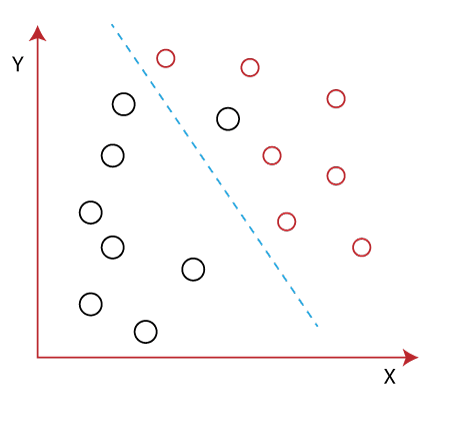
Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.



To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

### Example:

Let's assume we have to classify two different classes having two sets of data points in a 2-dimensional plane as shown below image:



However, it is impossible to draw a straight line in a 2-d plane that can separate these data points efficiently but using linear Discriminant analysis; we can dimensionally reduce the 2-D plane into the 1-D plane. Using this technique, we can also maximize the separability between multiple classes.

**Implementation:**

**import** pandas **as** pd

Reference Link: <https://medium.com/@betulmesci/dimensionality-reduction-with-principal-component-analysis-and-linear-discriminant-analysis-on-iris-dc1731c07fad>

df = pd.read\_csv("Iris.csv")

print(df)

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \  
0 1 5.1 3.5 1.4 0.2   
1 2 4.9 3.0 1.4 0.2   
2 3 4.7 3.2 1.3 0.2   
3 4 4.6 3.1 1.5 0.2   
4 5 5.0 3.6 1.4 0.2   
.. ... ... ... ... ...   
145 146 6.7 3.0 5.2 2.3   
146 147 6.3 2.5 5.0 1.9   
147 148 6.5 3.0 5.2 2.0   
148 149 6.2 3.4 5.4 2.3   
149 150 5.9 3.0 5.1 1.8   
  
 Species   
0 Iris-setosa   
1 Iris-setosa   
2 Iris-setosa   
3 Iris-setosa   
4 Iris-setosa   
.. ...   
145 Iris-virginica   
146 Iris-virginica   
147 Iris-virginica   
148 Iris-virginica   
149 Iris-virginica   
  
[150 rows x 6 columns]

df.Species.unique()

array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

X = df.drop(['Id','Species'],axis=1)  
y = df['Species']

**from** sklearn.preprocessing **import** StandardScaler  
  
*# Scale the features*  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)

**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
  
*# Create an instance of LDA*  
lda = LinearDiscriminantAnalysis(n\_components=2)  
  
*# Apply LDA on the scaled features*  
X\_lda = lda.fit\_transform(X\_scaled, y)

**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.model\_selection **import** train\_test\_split  
  
*# Split the data into training and testing sets*  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_lda, y, test\_size=0.2, random\_state=42)  
  
*# Train a logistic regression classifier*  
classifier = LogisticRegression()  
classifier.fit(X\_train, y\_train)

LogisticRegression()

*# Suppose you have a new flower with the following measurements:*  
new\_flower = [[6.7,3.0,5.2,2.3 ]] *# Sepal length, sepal width, petal length, petal width*  
  
*# Scale the new flower using the same scaler used for training*  
new\_flower\_scaled = scaler.transform(new\_flower)  
  
*# Apply LDA on the scaled new flower*  
new\_flower\_lda = lda.transform(new\_flower\_scaled)  
  
*# Predict the species of the new flower*  
predicted\_species = classifier.predict(new\_flower\_lda)  
  
*# Map the predicted label to the actual species*  
species\_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}  
predicted\_species\_name = species\_mapping[predicted\_species[0]]  
  
*# Print the predicted species*  
print("Predicted species:", predicted\_species\_name)

Predicted species: 2

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names  
 warnings.warn(

**EXPERIMENT NO. 2 A**

* **Aim**: Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following

tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and ridge, Lasso regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

* **Hardware Requirement:**
* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures
* **Software Requirement:**

Jupyter Nootbook/Ubuntu

* **Theory:**

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as temperature, age, salary, price, etc.

## Why do we use Regression Analysis?

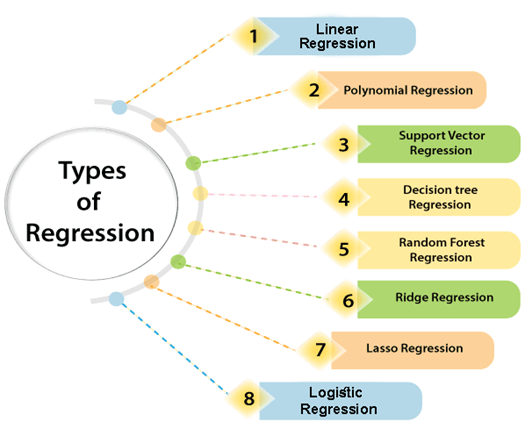
As mentioned above, Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science. Below are some other reasons for using Regression analysis:

* Regression estimates the relationship between the target and the independent variable.
* It is used to find the trends in data.
* It helps to predict real/continuous values.
* By performing the regression, we can confidently determine the most important factor, the least important factor, and how each factor is affecting the other factors.

## Types of Regression

There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables. Here we are discussing some important types of regression which are given below:

* Linear Regression
* Logistic Regression
* Polynomial Regression
* Support Vector Regression
* Decision Tree Regression
* Random Forest Regression
* Ridge Regression
* Lasso Regression:



Implementation:

**EXPERIMENT NO. 2 B**

* **Aim**: Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets Dataset link: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>
* **Hardware Requirement:**
* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures
* **Software Requirement:**

Jupyter Nootbook/Ubuntu

* **Theory:**

Descriptive statistics are brief informational coefficients that summarize a given data set, which can be either a representation of the entire population or a sample of a population. Descriptive statistics are broken down into measures of central tendency and measures of variability (spread). Measures of central tendency include the mean, median, and mode, while measures of variability include standard deviation, variance, minimum and maximum variables, [kurtosis](https://www.investopedia.com/terms/k/kurtosis.asp), and [skewness](https://www.investopedia.com/terms/s/skewness.asp).

## Types of Descriptive Statistics

All descriptive statistics are either measures of central tendency or measures of [variability](https://www.investopedia.com/terms/v/variability.asp), also known as measures of dispersion.

### Central Tendency

Measures of central tendency focus on the average or middle values of data sets, whereas measures of variability focus on the dispersion of data. These two measures use graphs, tables and general discussions to help people understand the meaning of the analyzed data.

Measures of central tendency describe the center position of a distribution for a data set. A person analyzes the frequency of each data point in the distribution and describes it using the mean, median, or mode, which measures the most common patterns of the analyzed data set.

### Measures of Variability

Measures of variability (or the measures of spread) aid in analyzing how dispersed the distribution is for a set of data. For example, while the measures of central tendency may give a person the average of a data set, it does not describe how the data is distributed within the set.

So while the average of the data maybe 65 out of 100, there can still be data points at both 1 and 100. Measures of variability help communicate this by describing the shape and spread of the data set. Range, [quartiles](https://www.investopedia.com/terms/q/quartile.asp), absolute deviation, and variance are all examples of measures of variability.

Consider the following data set: 5, 19, 24, 62, 91, 100. The range of that data set is 95, which is calculated by subtracting the lowest number (5) in the data set from the highest (100).

### Distribution

Distribution (or frequency distribution) refers to the quantity of times a data point occurs. Alternatively, it is the measurement of a data point failing to occur. Consider a data set: male, male, female, female, female, other. The distribution of this data can be classified as:

* The number of males in the data set is 2.
* The number of females in the data set is 3.
* The number of individuals identifying as other is 1.
* The number of non-males is 4.

## Univariate vs. Bivariate

In descriptive statistics, univariate data analyzes only one variable. It is used to identify characteristics of a single trait and is not used to analyze any relationships or causations.

For example, imagine a room full of high school students. Say you wanted to gather the average age of the individuals in the room. This univariate data is only dependent on one factor: each person's age. By gathering this one piece of information from each person and dividing by the total number of people, you can determine the average age.

Bivariate data, on the other hand, attempts to link two variables by searching for correlation. Two types of data are collected, and the relationship between the two pieces of information is analyzed together. Because multiple variables are analyzed, this approach may also be referred to as [multivariate](https://www.investopedia.com/terms/m/multivariate-model.asp).

## Descriptive Statistics vs. Inferential Statistics

Descriptive statistics have a different function than inferential statistics, data sets that are used to make decisions or apply characteristics from one data set to another.

Imagine another example where a company sells hot sauce. The company gathers data such as the count of [sales](https://www.investopedia.com/terms/s/sale.asp), average quantity purchased per [transaction](https://www.investopedia.com/terms/t/transaction.asp), and average sale per day of the week. All of this information is descriptive, as it tells a story of what actually happened in the past. In this case, it is not being used beyond being informational.

Let's say the same company wants to roll out a new hot sauce. It gathers the same sales data above, but it crafts the information to make predictions about what the sales of the new hot sauce will be. The act of using descriptive statistics and applying characteristics to a different data set makes the data set inferential statistics. We are no longer simply summarizing data; we are using it predict what will happen regarding an entirely different body of data (the new hot sauce product).

**Implementation:**

**import** numpy **as** np  
**import** pandas **as** pd

df = pd.read\_csv("C:/Users/HP/Dropbox/PC/Downloads/diabetes.csv")

df.shape

(768, 9)

df.head()

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \  
0 6 148 72 35 0 33.6   
1 1 85 66 29 0 26.6   
2 8 183 64 0 0 23.3   
3 1 89 66 23 94 28.1   
4 0 137 40 35 168 43.1   
  
 DiabetesPedigreeFunction Age Outcome   
0 0.627 50 1   
1 0.351 31 0   
2 0.672 32 1   
3 0.167 21 0   
4 2.288 33 1

df.describe()

Pregnancies Glucose BloodPressure SkinThickness Insulin \  
count 768.000000 768.000000 768.000000 768.000000 768.000000   
mean 3.845052 120.894531 69.105469 20.536458 79.799479   
std 3.369578 31.972618 19.355807 15.952218 115.244002   
min 0.000000 0.000000 0.000000 0.000000 0.000000   
25% 1.000000 99.000000 62.000000 0.000000 0.000000   
50% 3.000000 117.000000 72.000000 23.000000 30.500000   
75% 6.000000 140.250000 80.000000 32.000000 127.250000   
max 17.000000 199.000000 122.000000 99.000000 846.000000   
  
 BMI DiabetesPedigreeFunction Age Outcome   
count 768.000000 768.000000 768.000000 768.000000   
mean 31.992578 0.471876 33.240885 0.348958   
std 7.884160 0.331329 11.760232 0.476951   
min 0.000000 0.078000 21.000000 0.000000   
25% 27.300000 0.243750 24.000000 0.000000   
50% 32.000000 0.372500 29.000000 0.000000   
75% 36.600000 0.626250 41.000000 1.000000   
max 67.100000 2.420000 81.000000 1.000000

### Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis

**for** column **in** df.columns:  
 print(f"Column: {column}")  
 print(f"Frequency:\n{df[column].value\_counts()}\n")  
 print(f"Mean: {df[column].mean()}")  
 print(f"Median: {df[column].median()}")  
 print(f"Mode:\n{df[column].mode()}")  
 print(f"Variance: {df[column].var()}")  
 print(f"Standard Deviation: {df[column].std()}")  
 print(f"Skewness: {df[column].skew()}")  
 print(f"Kurtosis: {df[column].kurt()}")  
 print("----------\n")

Column: Pregnancies  
Frequency:  
1 135  
0 111  
2 103  
3 75  
4 68  
5 57  
6 50  
7 45  
8 38  
9 28  
10 24  
11 11  
13 10  
12 9  
14 2  
15 1  
17 1  
Name: Pregnancies, dtype: int64  
  
Mean: 3.8450520833333335  
Median: 3.0  
Mode:  
0 1  
Name: Pregnancies, dtype: int64  
Variance: 11.35405632062142  
Standard Deviation: 3.3695780626988623  
Skewness: 0.9016739791518588  
Kurtosis: 0.15921977754746486  
----------  
  
Column: Glucose  
Frequency:  
99 17  
100 17  
111 14  
129 14  
125 14  
 ..  
191 1  
177 1  
44 1  
62 1  
190 1  
Name: Glucose, Length: 136, dtype: int64  
  
Mean: 120.89453125  
Median: 117.0  
Mode:  
0 99  
1 100  
Name: Glucose, dtype: int64  
Variance: 1022.2483142519557  
Standard Deviation: 31.97261819513622  
Skewness: 0.17375350179188992  
Kurtosis: 0.6407798203735053  
----------  
  
Column: BloodPressure  
Frequency:  
70 57  
74 52  
78 45  
68 45  
72 44  
64 43  
80 40  
76 39  
60 37  
0 35  
62 34  
66 30  
82 30  
88 25  
84 23  
90 22  
86 21  
58 21  
50 13  
56 12  
52 11  
54 11  
75 8  
92 8  
65 7  
85 6  
94 6  
48 5  
96 4  
44 4  
100 3  
106 3  
98 3  
110 3  
55 2  
108 2  
104 2  
46 2  
30 2  
122 1  
95 1  
102 1  
61 1  
24 1  
38 1  
40 1  
114 1  
Name: BloodPressure, dtype: int64  
  
Mean: 69.10546875  
Median: 72.0  
Mode:  
0 70  
Name: BloodPressure, dtype: int64  
Variance: 374.6472712271838  
Standard Deviation: 19.355807170644777  
Skewness: -1.8436079833551302  
Kurtosis: 5.180156560082496  
----------  
  
Column: SkinThickness  
Frequency:  
0 227  
32 31  
30 27  
27 23  
23 22  
33 20  
28 20  
18 20  
31 19  
19 18  
39 18  
29 17  
40 16  
25 16  
26 16  
22 16  
37 16  
41 15  
35 15  
36 14  
15 14  
17 14  
20 13  
24 12  
42 11  
13 11  
21 10  
46 8  
34 8  
12 7  
38 7  
11 6  
43 6  
16 6  
45 6  
14 6  
44 5  
10 5  
48 4  
47 4  
49 3  
50 3  
8 2  
7 2  
52 2  
54 2  
63 1  
60 1  
56 1  
51 1  
99 1  
Name: SkinThickness, dtype: int64  
  
Mean: 20.536458333333332  
Median: 23.0  
Mode:  
0 0  
Name: SkinThickness, dtype: int64  
Variance: 254.47324532811953  
Standard Deviation: 15.952217567727677  
Skewness: 0.10937249648187608  
Kurtosis: -0.520071866153013  
----------  
  
Column: Insulin  
Frequency:  
0 374  
105 11  
130 9  
140 9  
120 8  
 ...   
73 1  
171 1  
255 1  
52 1  
112 1  
Name: Insulin, Length: 186, dtype: int64  
  
Mean: 79.79947916666667  
Median: 30.5  
Mode:  
0 0  
Name: Insulin, dtype: int64  
Variance: 13281.180077955281  
Standard Deviation: 115.24400235133837  
Skewness: 2.272250858431574  
Kurtosis: 7.2142595543487715  
----------  
  
Column: BMI  
Frequency:  
32.0 13  
31.6 12  
31.2 12  
0.0 11  
32.4 10  
 ..  
36.7 1  
41.8 1  
42.6 1  
42.8 1  
46.3 1  
Name: BMI, Length: 248, dtype: int64  
  
Mean: 31.992578124999998  
Median: 32.0  
Mode:  
0 32.0  
Name: BMI, dtype: float64  
Variance: 62.15998395738257  
Standard Deviation: 7.8841603203754405  
Skewness: -0.42898158845356543  
Kurtosis: 3.290442900816981  
----------  
  
Column: DiabetesPedigreeFunction  
Frequency:  
0.258 6  
0.254 6  
0.268 5  
0.207 5  
0.261 5  
 ..  
1.353 1  
0.655 1  
0.092 1  
0.926 1  
0.171 1  
Name: DiabetesPedigreeFunction, Length: 517, dtype: int64  
  
Mean: 0.47187630208333325  
Median: 0.3725  
Mode:  
0 0.254  
1 0.258  
Name: DiabetesPedigreeFunction, dtype: float64  
Variance: 0.10977863787313938  
Standard Deviation: 0.33132859501277484  
Skewness: 1.919911066307204  
Kurtosis: 5.5949535279830584  
----------  
  
Column: Age  
Frequency:  
22 72  
21 63  
25 48  
24 46  
23 38  
28 35  
26 33  
27 32  
29 29  
31 24  
41 22  
30 21  
37 19  
42 18  
33 17  
38 16  
36 16  
32 16  
45 15  
34 14  
46 13  
43 13  
40 13  
39 12  
35 10  
50 8  
51 8  
52 8  
44 8  
58 7  
47 6  
54 6  
49 5  
48 5  
57 5  
53 5  
60 5  
66 4  
63 4  
62 4  
55 4  
67 3  
56 3  
59 3  
65 3  
69 2  
61 2  
72 1  
81 1  
64 1  
70 1  
68 1  
Name: Age, dtype: int64  
  
Mean: 33.240885416666664  
Median: 29.0  
Mode:  
0 22  
Name: Age, dtype: int64  
Variance: 138.30304589037365  
Standard Deviation: 11.76023154067868  
Skewness: 1.1295967011444805  
Kurtosis: 0.6431588885398942  
----------  
  
Column: Outcome  
Frequency:  
0 500  
1 268  
Name: Outcome, dtype: int64  
  
Mean: 0.3489583333333333  
Median: 0.0  
Mode:  
0 0  
Name: Outcome, dtype: int64  
Variance: 0.22748261625380098  
Standard Deviation: 0.4769513772427971  
Skewness: 0.635016643444986  
Kurtosis: -1.600929755156027  
----------

### Bivariate analysis: Linear and logistic regression modeling

**from** sklearn.linear\_model **import** LinearRegression, LogisticRegression  
  
*# Prepare the data*  
X\_linear = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]  
y\_linear = df['Outcome']  
  
*# Fit the linear regression model*  
model\_linear = LinearRegression()  
model\_linear.fit(X\_linear, y\_linear)  
  
*# Print the coefficients*  
print('Linear Regression Coefficients:')  
**for** feature, coef **in** zip(X\_linear.columns, model\_linear.coef\_):  
 print(f'{feature}: {coef}')  
  
*# Make predictions*  
predictions\_linear = model\_linear.predict(X\_linear)

Linear Regression Coefficients:  
Glucose: 0.005932504680360896  
BloodPressure: -0.00227883712542089  
SkinThickness: 0.00016697889986787442  
Insulin: -0.0002096169514137912  
BMI: 0.013310837289280066  
DiabetesPedigreeFunction: 0.1376781570786881  
Age: 0.005800684345071733

*# Prepare the data*  
X\_logistic = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]  
y\_logistic = df['Outcome']  
  
*# Fit the logistic regression model*  
model\_logistic = LogisticRegression()  
model\_logistic.fit(X\_logistic, y\_logistic)  
  
*# Print the coefficients*  
print('Logistic Regression Coefficients:')  
**for** feature, coef **in** zip(X\_logistic.columns, model\_logistic.coef\_[0]):  
 print(f'{feature}: {coef}')  
  
*# Make predictions*  
predictions\_logistic = model\_logistic.predict(X\_logistic)

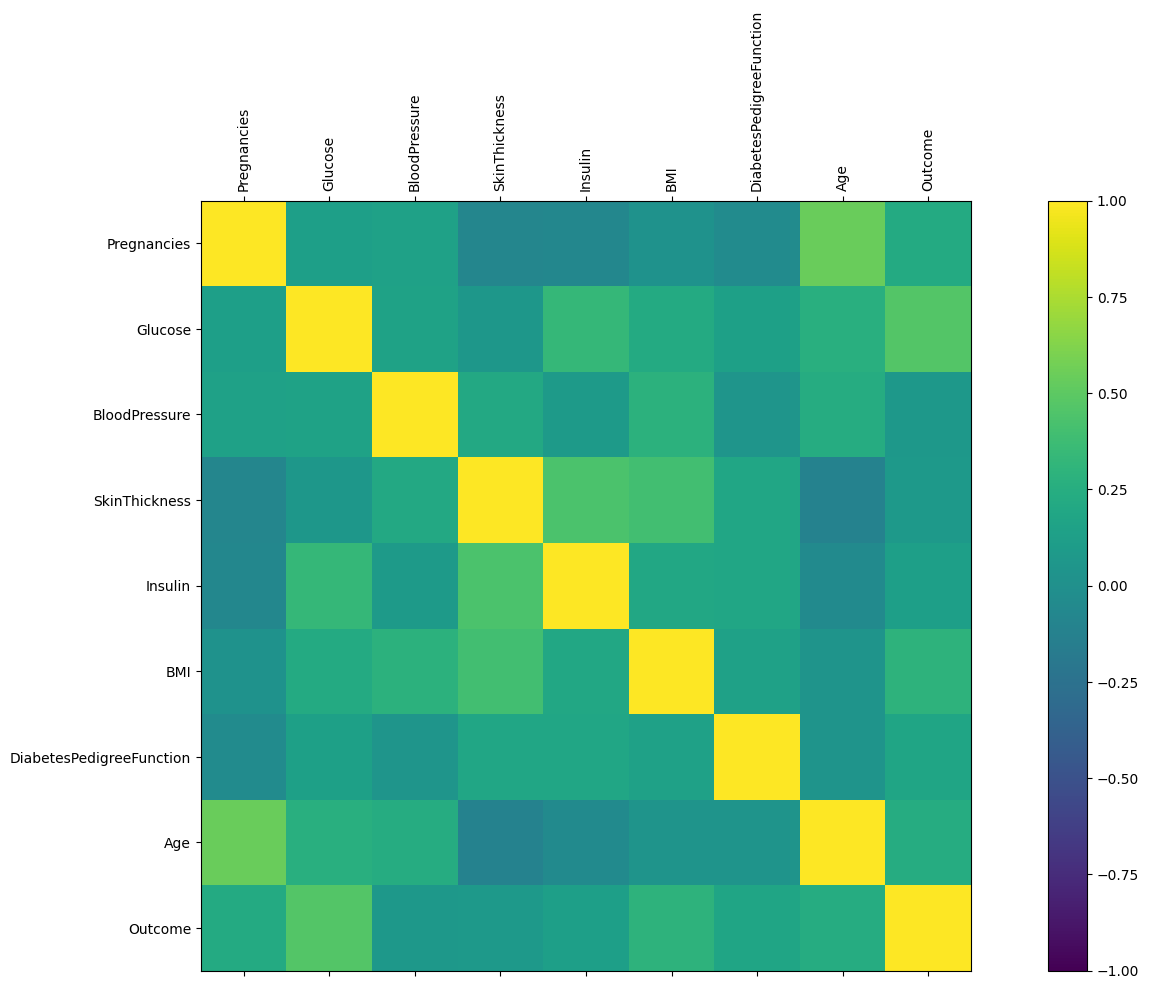
Logistic Regression Coefficients:  
Glucose: 0.03454477124790582  
BloodPressure: -0.01220824032665116  
SkinThickness: 0.0010051963882454211  
Insulin: -0.0013499454083243116  
BMI: 0.08780751006435426  
DiabetesPedigreeFunction: 0.8191678019528903  
Age: 0.032699759788267134

### Multiple Regression analysis

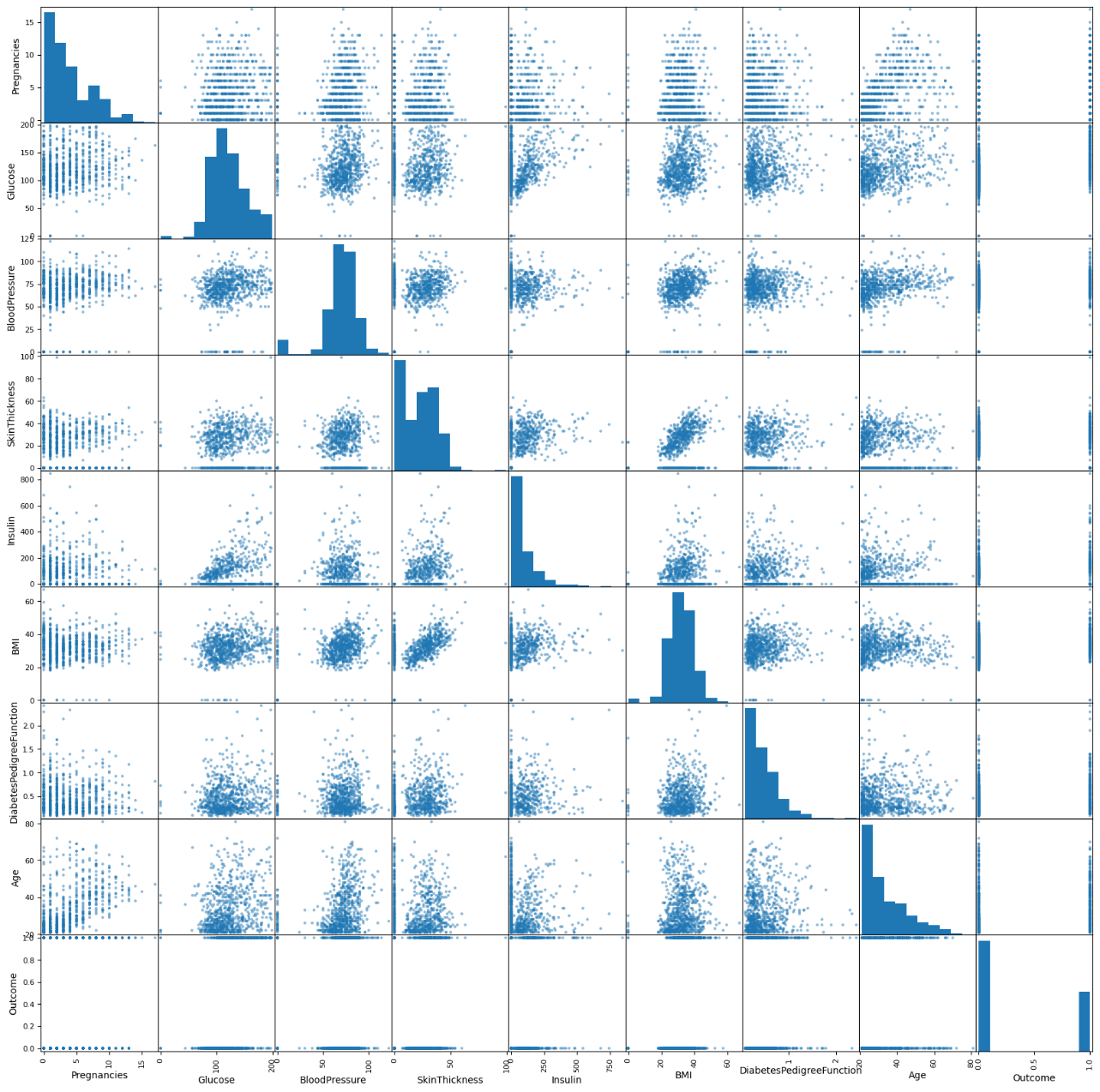
**import** statsmodels.api **as** sm  
  
*# Split the dataset into the independent variables (X) and the dependent variable (y)*  
X = df.drop('Outcome', axis=1) *# Independent variables*  
y = df['Outcome'] *# Dependent variable*  
  
*# Add a constant column to the independent variables*  
X = sm.add\_constant(X)  
  
*# Fit the multiple regression model*  
model = sm.OLS(y, X)  
results = model.fit()  
  
*# Print the regression results*  
print(results.summary())

OLS Regression Results   
==============================================================================  
Dep. Variable: Outcome R-squared: 0.303  
Model: OLS Adj. R-squared: 0.296  
Method: Least Squares F-statistic: 41.29  
Date: Sat, 08 Jul 2023 Prob (F-statistic): 7.36e-55  
Time: 15:59:17 Log-Likelihood: -381.91  
No. Observations: 768 AIC: 781.8  
Df Residuals: 759 BIC: 823.6  
Df Model: 8   
Covariance Type: nonrobust   
============================================================================================  
 coef std err t P>|t| [0.025 0.975]  
--------------------------------------------------------------------------------------------  
const -0.8539 0.085 -9.989 0.000 -1.022 -0.686  
Pregnancies 0.0206 0.005 4.014 0.000 0.011 0.031  
Glucose 0.0059 0.001 11.493 0.000 0.005 0.007  
BloodPressure -0.0023 0.001 -2.873 0.004 -0.004 -0.001  
SkinThickness 0.0002 0.001 0.139 0.890 -0.002 0.002  
Insulin -0.0002 0.000 -1.205 0.229 -0.000 0.000  
BMI 0.0132 0.002 6.344 0.000 0.009 0.017  
DiabetesPedigreeFunction 0.1472 0.045 3.268 0.001 0.059 0.236  
Age 0.0026 0.002 1.693 0.091 -0.000 0.006  
==============================================================================  
Omnibus: 41.539 Durbin-Watson: 1.982  
Prob(Omnibus): 0.000 Jarque-Bera (JB): 31.183  
Skew: 0.395 Prob(JB): 1.69e-07  
Kurtosis: 2.408 Cond. No. 1.10e+03  
==============================================================================  
  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
[2] The condition number is large, 1.1e+03. This might indicate that there are  
strong multicollinearity or other numerical problems.

fig = pyplot.figure()  
ax = fig.add\_subplot(111)  
cax = ax.matshow(corr, vmin=-1, vmax=1)  
fig.colorbar(cax)  
ticks = np.arange(0,9,1)  
ax.set\_xticks(ticks)  
ax.set\_yticks(ticks)  
names = df.columns  
*# Rotate x-tick labels by 90 degrees*  
ax.set\_xticklabels(names,rotation=90)  
ax.set\_yticklabels(names)  
pyplot.show()



*# Import required package*  
**from** pandas.plotting **import** scatter\_matrix  
pyplot.rcParams['figure.figsize'] = [20, 20]  
*# Plotting Scatterplot Matrix*  
scatter\_matrix(df)  
pyplot.show()



**EXPERIMENT NO. 3 A**

* **Aim**: Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).
* **Hardware Requirement:**
* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures
* **Software Requirement:**

Jypiter Nootbook/Ubuntu

* **Theory:**

## Classification Analysis: Definition

This analysis is a data mining technique used to determine the structure and categories within a given dataset. Classification analysis is commonly used in machine learning, text analytics, and statistical modelling. Above all, it can help identify patterns or groupings between individual observations, enabling researchers to understand their datasets better and make more accurate predictions.

Classification analysis is used to group or classify objects according to shared characteristics. Moreover, this analysis can be used in many applications, from segmenting customers for marketing campaigns to forecasting stock market trends.

## Classification Analysis Example

### Classifying images

One example of a classification analysis is the use of [supervised learning](https://emeritus.org/in/learn/data-science-supervised-learning-techniques/) algorithms to classify images. In this case, the algorithm is provided with an image dataset (the training set) that contains labelled images.

The algorithm uses labels to learn how to distinguish between different types of objects in the picture. Once trained, it can then be used to classify new images as belonging to one category or another.

### Customer Segmentation

Another example of classification analysis would be customer segmentation for marketing campaigns. Classification algorithms group customers into segments based on their characteristics and behaviours.

This helps marketers target specific groups with tailored content, offers, and promotions that are more likely to appeal to them.

### Stock Market Prediction

Finally, classification analysis can also be used for stock market prediction. Classification algorithms can identify patterns between past stock prices and other economic indicators, such as interest rates or unemployment figures. By understanding these correlations, analysts can better predict future market trends and make more informed investment decisions.

These are just some examples of how classification analysis can be applied to various scenarios. Unquestionably, classification algorithms can be used to analyse datasets in any domain, from healthcare and finance to agriculture and logistics.

## Classification Analysis Techniques

This analysis is a powerful technique used in data science to analyse and categorise data. Classification techniques are used in many areas, from predicting customer behaviours to finding patterns and trends in large datasets.

This analysis can help businesses make informed decisions about marketing strategies, product development, and more. So, let’s delve into the various techniques

### 1. Supervised Learning

Supervised learning algorithms require labelled data. This means the algorithm is provided with a dataset that has already been categorised or labelled with class labels. The algorithm then uses this label to learn how to distinguish between different class objects in the data. Once trained, it can use its predictive power to classify new datasets.

### 2. Unsupervised Learning

[Unsupervised learning](https://emeritus.org/in/learn/data-science-unsupervised-learning/) algorithms do not require labelled data. Instead, they use clustering and dimensionality reduction techniques to identify patterns in the dataset without any external guidance. These algorithms help segment customers or identify outlier items in a dataset.

### 3. Deep Learning

Deep learning is a subset/division of machine learning technologies that use artificial neural networks. These algorithms are capable of learning from large datasets and making complex decisions. Deep learning can be used for tasks such as image classification, natural language processing, and predictive analytics.

Classification algorithms can help uncover patterns in the data that could not be detected using traditional methods. By using classification analysis, businesses can gain valuable insights into their customers’ behaviours and preferences, helping them make more informed decisions.

**Implementation:**

*# Import Libraries*  
  
**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib **as** mpl  
**import** matplotlib.pyplot **as** plt

### Handwritten Digit Recognition

Use the sklearn.dataset load\_digits() method. It loads the handwritten digits dataset. The returned data is in the form of a Dictionary. The 'data' attribute contains a flattenned array of 64 (each digit image is of 8\*8 pixels) elements representing the digits.

The 'target' attribute is the 'class' of Digit (0-9) Each individual digit is represented through a flattendded 64 digit array numbers of Greyscale values. There are 1797 samples in total and each class or digit has roughly 180 samples.

**from** sklearn.datasets **import** load\_digits  
digits = load\_digits(n\_class=10)

digits

{'data': array([[ 0., 0., 5., ..., 0., 0., 0.],  
 [ 0., 0., 0., ..., 10., 0., 0.],  
 [ 0., 0., 0., ..., 16., 9., 0.],  
 ...,  
 [ 0., 0., 1., ..., 6., 0., 0.],  
 [ 0., 0., 2., ..., 12., 0., 0.],  
 [ 0., 0., 10., ..., 12., 1., 0.]]),  
 'target': array([0, 1, 2, ..., 8, 9, 8]),  
 'frame': None,  
 'feature\_names': ['pixel\_0\_0',  
 'pixel\_0\_1',  
 'pixel\_0\_2',  
 'pixel\_0\_3',  
 'pixel\_0\_4',  
 'pixel\_0\_5',  
 'pixel\_0\_6',  
 'pixel\_0\_7',  
 'pixel\_1\_0',  
 'pixel\_1\_1',  
 'pixel\_1\_2',  
 'pixel\_1\_3',  
 'pixel\_1\_4',  
 'pixel\_1\_5',  
 'pixel\_1\_6',  
 'pixel\_1\_7',  
 'pixel\_2\_0',  
 'pixel\_2\_1',  
 'pixel\_2\_2',  
 'pixel\_2\_3',  
 'pixel\_2\_4',  
 'pixel\_2\_5',  
 'pixel\_2\_6',  
 'pixel\_2\_7',  
 'pixel\_3\_0',  
 'pixel\_3\_1',  
 'pixel\_3\_2',  
 'pixel\_3\_3',  
 'pixel\_3\_4',  
 'pixel\_3\_5',  
 'pixel\_3\_6',  
 'pixel\_3\_7',  
 'pixel\_4\_0',  
 'pixel\_4\_1',  
 'pixel\_4\_2',  
 'pixel\_4\_3',  
 'pixel\_4\_4',  
 'pixel\_4\_5',  
 'pixel\_4\_6',  
 'pixel\_4\_7',  
 'pixel\_5\_0',  
 'pixel\_5\_1',  
 'pixel\_5\_2',  
 'pixel\_5\_3',  
 'pixel\_5\_4',  
 'pixel\_5\_5',  
 'pixel\_5\_6',  
 'pixel\_5\_7',  
 'pixel\_6\_0',  
 'pixel\_6\_1',  
 'pixel\_6\_2',  
 'pixel\_6\_3',  
 'pixel\_6\_4',  
 'pixel\_6\_5',  
 'pixel\_6\_6',  
 'pixel\_6\_7',  
 'pixel\_7\_0',  
 'pixel\_7\_1',  
 'pixel\_7\_2',  
 'pixel\_7\_3',  
 'pixel\_7\_4',  
 'pixel\_7\_5',  
 'pixel\_7\_6',  
 'pixel\_7\_7'],  
 'target\_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),  
 'images': array([[[ 0., 0., 5., ..., 1., 0., 0.],  
 [ 0., 0., 13., ..., 15., 5., 0.],  
 [ 0., 3., 15., ..., 11., 8., 0.],  
 ...,  
 [ 0., 4., 11., ..., 12., 7., 0.],  
 [ 0., 2., 14., ..., 12., 0., 0.],  
 [ 0., 0., 6., ..., 0., 0., 0.]],  
   
 [[ 0., 0., 0., ..., 5., 0., 0.],  
 [ 0., 0., 0., ..., 9., 0., 0.],  
 [ 0., 0., 3., ..., 6., 0., 0.],  
 ...,  
 [ 0., 0., 1., ..., 6., 0., 0.],  
 [ 0., 0., 1., ..., 6., 0., 0.],  
 [ 0., 0., 0., ..., 10., 0., 0.]],  
   
 [[ 0., 0., 0., ..., 12., 0., 0.],  
 [ 0., 0., 3., ..., 14., 0., 0.],  
 [ 0., 0., 8., ..., 16., 0., 0.],  
 ...,  
 [ 0., 9., 16., ..., 0., 0., 0.],  
 [ 0., 3., 13., ..., 11., 5., 0.],  
 [ 0., 0., 0., ..., 16., 9., 0.]],  
   
 ...,  
   
 [[ 0., 0., 1., ..., 1., 0., 0.],  
 [ 0., 0., 13., ..., 2., 1., 0.],  
 [ 0., 0., 16., ..., 16., 5., 0.],  
 ...,  
 [ 0., 0., 16., ..., 15., 0., 0.],  
 [ 0., 0., 15., ..., 16., 0., 0.],  
 [ 0., 0., 2., ..., 6., 0., 0.]],  
   
 [[ 0., 0., 2., ..., 0., 0., 0.],  
 [ 0., 0., 14., ..., 15., 1., 0.],  
 [ 0., 4., 16., ..., 16., 7., 0.],  
 ...,  
 [ 0., 0., 0., ..., 16., 2., 0.],  
 [ 0., 0., 4., ..., 16., 2., 0.],  
 [ 0., 0., 5., ..., 12., 0., 0.]],  
   
 [[ 0., 0., 10., ..., 1., 0., 0.],  
 [ 0., 2., 16., ..., 1., 0., 0.],  
 [ 0., 0., 15., ..., 15., 0., 0.],  
 ...,  
 [ 0., 4., 16., ..., 16., 6., 0.],  
 [ 0., 8., 16., ..., 16., 8., 0.],  
 [ 0., 1., 8., ..., 12., 1., 0.]]]),  
 'DESCR': ".. \_digits\_dataset:\n\nOptical recognition of handwritten digits dataset\n--------------------------------------------------\n\n\*\*Data Set Characteristics:\*\*\n\n :Number of Instances: 1797\n :Number of Attributes: 64\n :Attribute Information: 8x8 image of integer pixels in the range 0..16.\n :Missing Attribute Values: None\n :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmaps of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of on pixels are counted in each block. This generates\nan input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensionality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n Applications to Handwritten Digit Recognition, MSc Thesis, Institute of\n Graduate Studies in Science and Engineering, Bogazici University.\n - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n Linear dimensionalityreduction using relevance weighted LDA. School of\n Electrical and Electronic Engineering Nanyang Technological University.\n 2005.\n - Claudio Gentile. A New Approximate Maximal Margin Classification\n Algorithm. NIPS. 2000.\n"}

digits['data'][0].reshape(8,8)

array([[ 0., 0., 5., 13., 9., 1., 0., 0.],  
 [ 0., 0., 13., 15., 10., 15., 5., 0.],  
 [ 0., 3., 15., 2., 0., 11., 8., 0.],  
 [ 0., 4., 12., 0., 0., 8., 8., 0.],  
 [ 0., 5., 8., 0., 0., 9., 8., 0.],  
 [ 0., 4., 11., 0., 1., 12., 7., 0.],  
 [ 0., 2., 14., 5., 10., 12., 0., 0.],  
 [ 0., 0., 6., 13., 10., 0., 0., 0.]])

digits['data'][0]

array([ 0., 0., 5., 13., 9., 1., 0., 0., 0., 0., 13., 15., 10.,  
 15., 5., 0., 0., 3., 15., 2., 0., 11., 8., 0., 0., 4.,  
 12., 0., 0., 8., 8., 0., 0., 5., 8., 0., 0., 9., 8.,  
 0., 0., 4., 11., 0., 1., 12., 7., 0., 0., 2., 14., 5.,  
 10., 12., 0., 0., 0., 0., 6., 13., 10., 0., 0., 0.])

digits['images'][1]

array([[ 0., 0., 0., 12., 13., 5., 0., 0.],  
 [ 0., 0., 0., 11., 16., 9., 0., 0.],  
 [ 0., 0., 3., 15., 16., 6., 0., 0.],  
 [ 0., 7., 15., 16., 16., 2., 0., 0.],  
 [ 0., 0., 1., 16., 16., 3., 0., 0.],  
 [ 0., 0., 1., 16., 16., 6., 0., 0.],  
 [ 0., 0., 1., 16., 16., 6., 0., 0.],  
 [ 0., 0., 0., 11., 16., 10., 0., 0.]])

digits['target'][0:9]

array([0, 1, 2, 3, 4, 5, 6, 7, 8])

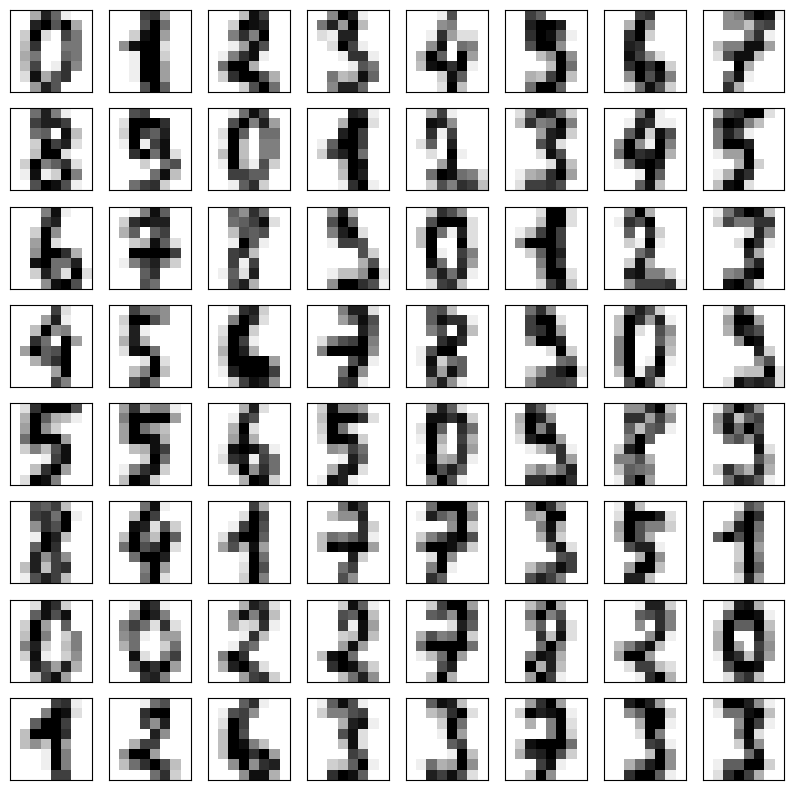
digits['target'][0]

0

digits.images[0]

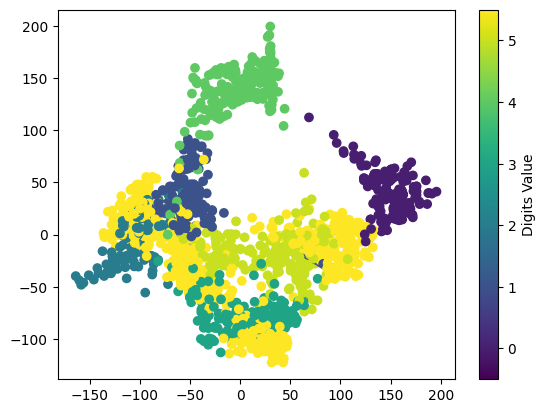
array([[ 0., 0., 5., 13., 9., 1., 0., 0.],  
 [ 0., 0., 13., 15., 10., 15., 5., 0.],  
 [ 0., 3., 15., 2., 0., 11., 8., 0.],  
 [ 0., 4., 12., 0., 0., 8., 8., 0.],  
 [ 0., 5., 8., 0., 0., 9., 8., 0.],  
 [ 0., 4., 11., 0., 1., 12., 7., 0.],  
 [ 0., 2., 14., 5., 10., 12., 0., 0.],  
 [ 0., 0., 6., 13., 10., 0., 0., 0.]])

*# Each Digit is represented in digits.images as a matrix of 8x8 = 64 pixels. Each of the 64 values represent*  
*# a greyscale. The Greyscale are then plotted in the right scale by the imshow method.*  
  
fig, ax = plt.subplots(8,8, figsize=(10,10))  
**for** i, axi **in** enumerate(ax.flat):  
 axi.imshow(digits.images[i], cmap='binary')  
 axi.set(xticks=[], yticks=[])



*# Plotting - Clustering the data points after using Manifold Learning*  
  
**from** sklearn.manifold **import** Isomap  
  
iso = Isomap(n\_components=2)  
  
projection = iso.fit\_transform(digits.data) *# digits.data - 64 dimensions to 2 dimensions*  
  
plt.scatter(projection[:, 0], projection[:, 1], c=digits.target, cmap="viridis")  
  
plt.colorbar(ticks=range(10), label='Digits Value')  
plt.clim(-0.5, 5.5)

/usr/local/lib/python3.10/dist-packages/sklearn/manifold/\_isomap.py:373: UserWarning: The number of connected components of the neighbors graph is 2 > 1. Completing the graph to fit Isomap might be slow. Increase the number of neighbors to avoid this issue.  
 self.\_fit\_transform(X)  
/usr/local/lib/python3.10/dist-packages/scipy/sparse/\_index.py:103: SparseEfficiencyWarning: Changing the sparsity structure of a csr\_matrix is expensive. lil\_matrix is more efficient.  
 self.\_set\_intXint(row, col, x.flat[0])

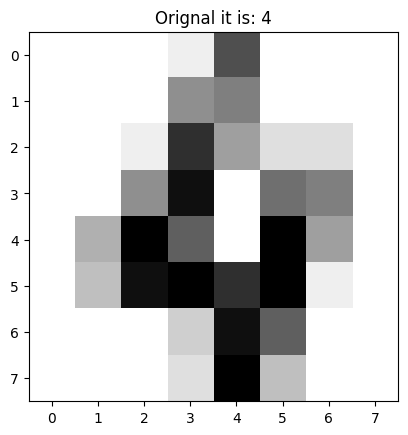


print(projection[:, 0][70], projection[:, 1][70])

-56.60683580684862 61.95022367117501

**def** view\_digit(index):  
 plt.imshow(digits.images[index] , cmap = plt.cm.gray\_r)  
 plt.title('Orignal it is: '+ str(digits.target[index]))  
 plt.show()

view\_digit(4)



### Use the Support Vector Machine Classifier to train the Data

Use part of the data for train and part of the data for test (predicion)

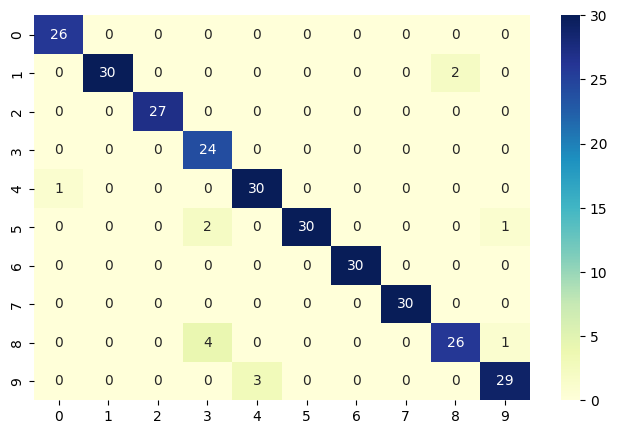
main\_data = digits['data']  
targets = digits['target']

**from** sklearn **import** svm  
  
svc = svm.SVC(gamma=0.001 , C = 100)  
  
*# GAMMA is a parameter for non linear hyperplanes.*  
*# The higher the gamma value it tries to exactly fit the training data set*  
*# C is the penalty parameter of the error term.*  
*# It controls the trade off between smooth decision boundary and classifying the training points correctly.*  
  
svc.fit(main\_data[:1500] , targets[:1500])  
  
predictions = svc.predict(main\_data[1501:])  
  
list(zip(predictions , targets[1501:]))

[(7, 7),  
 (4, 4),  
 (6, 6),  
 (3, 3),  
 (1, 1),  
 (3, 3),  
 (9, 9),  
 (1, 1),  
 (7, 7),  
 (6, 6),  
 (8, 8),  
 (4, 4),  
 (3, 3),  
 (1, 1),  
 (4, 4),  
 (0, 0),  
 (5, 5),  
 (3, 3),  
 (6, 6),  
 (9, 9),  
 (6, 6),  
 (1, 1),  
 (7, 7),  
 (5, 5),  
 (4, 4),  
 (4, 4),  
 (7, 7),  
 (2, 2),  
 (8, 8),  
 (2, 2),  
 (2, 2),  
 (5, 5),  
 (7, 7),  
 (9, 9),  
 (5, 5),  
 (4, 4),  
 (8, 8),  
 (8, 8),  
 (4, 4),  
 (9, 9),  
 (0, 0),  
 (8, 8),  
 (9, 9),  
 (8, 8),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (3, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (7, 7),  
 (1, 8),  
 (9, 9),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (3, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (9, 9),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (3, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (7, 7),  
 (1, 8),  
 (9, 9),  
 (4, 0),  
 (9, 9),  
 (5, 5),  
 (5, 5),  
 (6, 6),  
 (5, 5),  
 (0, 0),  
 (9, 9),  
 (8, 8),  
 (9, 9),  
 (8, 8),  
 (4, 4),  
 (1, 1),  
 (7, 7),  
 (7, 7),  
 (3, 3),  
 (5, 5),  
 (1, 1),  
 (0, 0),  
 (0, 0),  
 (2, 2),  
 (2, 2),  
 (7, 7),  
 (8, 8),  
 (2, 2),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (6, 6),  
 (8, 3),  
 (3, 3),  
 (7, 7),  
 (3, 3),  
 (3, 3),  
 (4, 4),  
 (6, 6),  
 (6, 6),  
 (6, 6),  
 (9, 4),  
 (9, 9),  
 (1, 1),  
 (5, 5),  
 (0, 0),  
 (9, 9),  
 (5, 5),  
 (2, 2),  
 (8, 8),  
 (0, 0),  
 (1, 1),  
 (7, 7),  
 (6, 6),  
 (3, 3),  
 (2, 2),  
 (1, 1),  
 (7, 7),  
 (9, 4),  
 (6, 6),  
 (3, 3),  
 (1, 1),  
 (3, 3),  
 (9, 9),  
 (1, 1),  
 (7, 7),  
 (6, 6),  
 (8, 8),  
 (4, 4),  
 (3, 3),  
 (1, 1),  
 (4, 4),  
 (0, 0),  
 (5, 5),  
 (3, 3),  
 (6, 6),  
 (9, 9),  
 (6, 6),  
 (1, 1),  
 (7, 7),  
 (5, 5),  
 (4, 4),  
 (4, 4),  
 (7, 7),  
 (2, 2),  
 (2, 2),  
 (5, 5),  
 (7, 7),  
 (8, 9),  
 (5, 5),  
 (9, 4),  
 (4, 4),  
 (5, 9),  
 (0, 0),  
 (8, 8),  
 (9, 9),  
 (8, 8),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (3, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (7, 7),  
 (8, 8),  
 (9, 9),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (3, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (7, 7),  
 (8, 8),  
 (9, 9),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (8, 3),  
 (4, 4),  
 (5, 5),  
 (6, 6),  
 (7, 7),  
 (8, 8),  
 (9, 9),  
 (0, 0),  
 (9, 9),  
 (5, 5),  
 (5, 5),  
 (6, 6),  
 (5, 5),  
 (0, 0),  
 (9, 9),  
 (8, 8),  
 (9, 9),  
 (8, 8),  
 (4, 4),  
 (1, 1),  
 (7, 7),  
 (7, 7),  
 (3, 3),  
 (5, 5),  
 (1, 1),  
 (0, 0),  
 (0, 0),  
 (2, 2),  
 (2, 2),  
 (7, 7),  
 (8, 8),  
 (2, 2),  
 (0, 0),  
 (1, 1),  
 (2, 2),  
 (6, 6),  
 (8, 3),  
 (8, 3),  
 (7, 7),  
 (5, 3),  
 (3, 3),  
 (4, 4),  
 (6, 6),  
 (6, 6),  
 (6, 6),  
 (4, 4),  
 (9, 9),  
 (1, 1),  
 (5, 5),  
 (0, 0),  
 (9, 9),  
 (5, 5),  
 (2, 2),  
 (8, 8),  
 (2, 2),  
 (0, 0),  
 (0, 0),  
 (1, 1),  
 (7, 7),  
 (6, 6),  
 (3, 3),  
 (2, 2),  
 (1, 1),  
 (7, 7),  
 (4, 4),  
 (6, 6),  
 (3, 3),  
 (1, 1),  
 (3, 3),  
 (9, 9),  
 (1, 1),  
 (7, 7),  
 (6, 6),  
 (8, 8),  
 (4, 4),  
 (5, 3),  
 (1, 1),  
 (4, 4),  
 (0, 0),  
 (5, 5),  
 (3, 3),  
 (6, 6),  
 (9, 9),  
 (6, 6),  
 (1, 1),  
 (7, 7),  
 (5, 5),  
 (4, 4),  
 (4, 4),  
 (7, 7),  
 (2, 2),  
 (8, 8),  
 (2, 2),  
 (2, 2),  
 (5, 5),  
 (7, 7),  
 (9, 9),  
 (5, 5),  
 (4, 4),  
 (8, 8),  
 (8, 8),  
 (4, 4),  
 (9, 9),  
 (0, 0),  
 (8, 8),  
 (9, 9),  
 (8, 8)]

### Create the Confusion Matric for Performance Evaluation

**from** sklearn.metrics **import** confusion\_matrix  
**import** seaborn **as** sns  
  
cm = confusion\_matrix(predictions, targets[1501:])  
  
conf\_matrix = pd.DataFrame(data = cm)  
  
plt.figure(figsize = (8,5))  
  
sns.heatmap(conf\_matrix, annot=True,fmt='d',cmap="YlGnBu");



cm

array([[26, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [ 0, 30, 0, 0, 0, 0, 0, 0, 2, 0],  
 [ 0, 0, 27, 0, 0, 0, 0, 0, 0, 0],  
 [ 0, 0, 0, 24, 0, 0, 0, 0, 0, 0],  
 [ 1, 0, 0, 0, 30, 0, 0, 0, 0, 0],  
 [ 0, 0, 0, 2, 0, 30, 0, 0, 0, 1],  
 [ 0, 0, 0, 0, 0, 0, 30, 0, 0, 0],  
 [ 0, 0, 0, 0, 0, 0, 0, 30, 0, 0],  
 [ 0, 0, 0, 4, 0, 0, 0, 0, 26, 1],  
 [ 0, 0, 0, 0, 3, 0, 0, 0, 0, 29]])

### Print the Classification Report

**from** sklearn.metrics **import** classification\_report  
  
print(classification\_report(predictions, targets[1501:]))

precision recall f1-score support  
  
 0 0.96 1.00 0.98 26  
 1 1.00 0.94 0.97 32  
 2 1.00 1.00 1.00 27  
 3 0.80 1.00 0.89 24  
 4 0.91 0.97 0.94 31  
 5 1.00 0.91 0.95 33  
 6 1.00 1.00 1.00 30  
 7 1.00 1.00 1.00 30  
 8 0.93 0.84 0.88 31  
 9 0.94 0.91 0.92 32  
  
 accuracy 0.95 296  
 macro avg 0.95 0.96 0.95 296  
weighted avg 0.96 0.95 0.95 296

**EXPERIMENT NO. 4 A**

* **Aim**: Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters

using the elbow method.Dataset Link: https://www.kaggle.com/datasets/uciml/iris

* **Hardware Requirement:**
* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures
* **Software Requirement:**

Jupyter Nootbook/Ubuntu

* **Theory:**

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by ‘K’ in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

## Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps −

* Step 1 − First, we need to specify the number of clusters, K, need to be generated by this algorithm.
* Step 2 − Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.
* Step 3 − Now it will compute the cluster centroids.
* Step 4 − Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more −

4.1 − First, the sum of squared distance between data points and centroids would be computed.

4.2 − Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

4.3 − At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows Expectation-Maximization approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things −

* While working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.
* Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids

**Implementation:**

### Importing the libraries and the data

import pandas as pd # Pandas (version : 1.1.5)  
 import numpy as np # Numpy (version : 1.19.2)  
 import matplotlib.pyplot as plt # Matplotlib (version : 3.3.2)  
 from sklearn.cluster import KMeans # Scikit Learn (version : 0.23.2)  
 import seaborn as sns # Seaborn (version : 0.11.1)  
 plt.style.use('seaborn')

### Importing the data from .csv file

First we read the data from the dataset using read\_csv from the pandas library.

data = pd.read\_csv('data\iris.csv')

Viewing the data that we imported to pandas dataframe object

data

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \  
 0 1 5.1 3.5 1.4 0.2   
 1 2 4.9 3.0 1.4 0.2   
 2 3 4.7 3.2 1.3 0.2   
 3 4 4.6 3.1 1.5 0.2   
 4 5 5.0 3.6 1.4 0.2   
 .. ... ... ... ... ...   
 145 146 6.7 3.0 5.2 2.3   
 146 147 6.3 2.5 5.0 1.9   
 147 148 6.5 3.0 5.2 2.0   
 148 149 6.2 3.4 5.4 2.3   
 149 150 5.9 3.0 5.1 1.8   
  
 Species   
 0 Iris-setosa   
 1 Iris-setosa   
 2 Iris-setosa   
 3 Iris-setosa   
 4 Iris-setosa   
 .. ...   
 145 Iris-virginica   
 146 Iris-virginica   
 147 Iris-virginica   
 148 Iris-virginica   
 149 Iris-virginica   
  
 [150 rows x 6 columns]

### Viewing and Describing the data

Now we view the Head and Tail of the data using head() and tail() respectively.

data.head()

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species  
 0 1 5.1 3.5 1.4 0.2 Iris-setosa  
 1 2 4.9 3.0 1.4 0.2 Iris-setosa  
 2 3 4.7 3.2 1.3 0.2 Iris-setosa  
 3 4 4.6 3.1 1.5 0.2 Iris-setosa  
 4 5 5.0 3.6 1.4 0.2 Iris-setosa

data.tail()

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \  
 145 146 6.7 3.0 5.2 2.3   
 146 147 6.3 2.5 5.0 1.9   
 147 148 6.5 3.0 5.2 2.0   
 148 149 6.2 3.4 5.4 2.3   
 149 150 5.9 3.0 5.1 1.8   
  
 Species   
 145 Iris-virginica   
 146 Iris-virginica   
 147 Iris-virginica   
 148 Iris-virginica   
 149 Iris-virginica

Checking the sample size of data - how many samples are there in the dataset using len().

len(data)

150

Checking the dimensions/shape of the dataset using shape.

data.shape

(150, 6)

Viewing Column names of the dataset using columns

data.columns

Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',  
 'Species'],  
 dtype='object')

for i,col in enumerate(data.columns):  
 print(f'Column number {1+i} is {col}')

Column number 1 is Id  
 Column number 2 is SepalLengthCm  
 Column number 3 is SepalWidthCm  
 Column number 4 is PetalLengthCm  
 Column number 5 is PetalWidthCm  
 Column number 6 is Species

So, our dataset has 5 columns named:

• Id

• SepalLengthCm

• SepalWidthCm

• PetalLengthCm

• PetalWidthCm

• Species.

View datatypes of each column in the dataset using dtype.

data.dtypes

Id int64  
 SepalLengthCm float64  
 SepalWidthCm float64  
 PetalLengthCm float64  
 PetalWidthCm float64  
 Species object  
 dtype: object

Gathering Further information about the dataset using info()

data.info()

<class 'pandas.core.frame.DataFrame'>  
 RangeIndex: 150 entries, 0 to 149  
 Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
 --- ------ -------------- -----   
 0 Id 150 non-null int64   
 1 SepalLengthCm 150 non-null float64  
 2 SepalWidthCm 150 non-null float64  
 3 PetalLengthCm 150 non-null float64  
 4 PetalWidthCm 150 non-null float64  
 5 Species 150 non-null object  
 dtypes: float64(4), int64(1), object(1)  
 memory usage: 7.2+ KB

Describing the data as basic statistics using describe()

data.describe()

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm  
 count 150.000000 150.000000 150.000000 150.000000 150.000000  
 mean 75.500000 5.843333 3.054000 3.758667 1.198667  
 std 43.445368 0.828066 0.433594 1.764420 0.763161  
 min 1.000000 4.300000 2.000000 1.000000 0.100000  
 25% 38.250000 5.100000 2.800000 1.600000 0.300000  
 50% 75.500000 5.800000 3.000000 4.350000 1.300000  
 75% 112.750000 6.400000 3.300000 5.100000 1.800000  
 max 150.000000 7.900000 4.400000 6.900000 2.500000

### Checking the data for inconsistencies and further cleaning the data if needed.

Checking data for missing values using isnull().

data.isnull()

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species  
 0 False False False False False False  
 1 False False False False False False  
 2 False False False False False False  
 3 False False False False False False  
 4 False False False False False False  
 .. ... ... ... ... ... ...  
 145 False False False False False False  
 146 False False False False False False  
 147 False False False False False False  
 148 False False False False False False  
 149 False False False False False False  
  
 [150 rows x 6 columns]

Checking summary of missing values

data.isnull().sum()

Id 0  
 SepalLengthCm 0  
 SepalWidthCm 0  
 PetalLengthCm 0  
 PetalWidthCm 0  
 Species 0  
 dtype: int64

The 'Id' column has no relevence therefore deleting it would be better.

Deleting 'customer\_id' colummn using drop().

data.drop('Id', axis=1, inplace=True)  
 data.head()

SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species  
 0 5.1 3.5 1.4 0.2 Iris-setosa  
 1 4.9 3.0 1.4 0.2 Iris-setosa  
 2 4.7 3.2 1.3 0.2 Iris-setosa  
 3 4.6 3.1 1.5 0.2 Iris-setosa  
 4 5.0 3.6 1.4 0.2 Iris-setosa

### Modelling

#### K - Means Clustering

K-means clustering is a clustering algorithm that aims to partition n observations into k clusters. Initialisation – K initial “means” (centroids) are generated at random Assignment – K clusters are created by associating each observation with the nearest centroid Update – The centroid of the clusters becomes the new mean, Assignment and Update are repeated iteratively until convergence The end result is that the sum of squared errors is minimised between points and their respective centroids. We will use KMeans Clustering. At first we will find the optimal clusters based on inertia and using elbow method. The distance between the centroids and the data points should be less.

First we need to check the data for any missing values as it can ruin our model.

data.isna().sum()

SepalLengthCm 0  
 SepalWidthCm 0  
 PetalLengthCm 0  
 PetalWidthCm 0  
 Species 0  
 dtype: int64

We conclude that we don't have any missing values therefore we can go forward and start the clustering procedure.

We will now view and select the data that we need for clustering.

data.head()

SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species  
 0 5.1 3.5 1.4 0.2 Iris-setosa  
 1 4.9 3.0 1.4 0.2 Iris-setosa  
 2 4.7 3.2 1.3 0.2 Iris-setosa  
 3 4.6 3.1 1.5 0.2 Iris-setosa  
 4 5.0 3.6 1.4 0.2 Iris-setosa

Checking the value count of the target column i.e. 'Species' using value\_counts()

data['Species'].value\_counts()

Iris-setosa 50  
 Iris-versicolor 50  
 Iris-virginica 50  
 Name: Species, dtype: int64

Splitting into Training and Target data

Target Data

target\_data = data.iloc[:,4]  
 target\_data.head()

0 Iris-setosa  
 1 Iris-setosa  
 2 Iris-setosa  
 3 Iris-setosa  
 4 Iris-setosa  
 Name: Species, dtype: object

Training data

clustering\_data = data.iloc[:,[0,1,2,3]]  
 clustering\_data.head()

SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm  
 0 5.1 3.5 1.4 0.2  
 1 4.9 3.0 1.4 0.2  
 2 4.7 3.2 1.3 0.2  
 3 4.6 3.1 1.5 0.2  
 4 5.0 3.6 1.4 0.2

Now, we need to visualize the data which we are going to use for the clustering. This will give us a fair idea about the data we're working on.

fig, ax = plt.subplots(figsize=(15,7))  
 sns.set(font\_scale=1.5)  
 ax = sns.scatterplot(x=data['SepalLengthCm'],y=data['SepalWidthCm'], s=70, color='#f73434', edgecolor='#f73434', linewidth=0.3)  
 ax.set\_ylabel('Sepal Width (in cm)')  
 ax.set\_xlabel('Sepal Length (in cm)')  
 plt.title('Sepal Length vs Width', fontsize = 20)  
 plt.show()

This gives us a fair Idea and patterns about some of the data.

### Determining No. of Clusters Required

The Elbow Method

The Elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes an average score for all clusters. By default, the distortion score is computed, the sum of square distances from each point to its assigned center.

When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. If the line chart looks like an arm, then the “elbow” (the point of inflection on the curve) is the best value of k. The “arm” can be either up or down, but if there is a strong inflection point, it is a good indication that the underlying model fits best at that point.

We use the Elbow Method which uses Within Cluster Sum Of Squares (WCSS) against the the number of clusters (K Value) to figure out the optimal number of clusters value. WCSS measures sum of distances of observations from their cluster centroids which is given by the below formula.

formula

where Yi is centroid for observation Xi. The main goal is to maximize number of clusters and in limiting case each data point becomes its own cluster centroid.

With this simple line of code we get all the inertia value or the within the cluster sum of square.

from sklearn.cluster import KMeans  
 wcss=[]  
 for i in range(1,11):  
 km = KMeans(i)  
 km.fit(clustering\_data)  
 wcss.append(km.inertia\_)  
 np.array(wcss)

array([680.8244 , 152.36870648, 78.94084143, 57.31787321,  
 46.53558205, 38.93096305, 34.29998554, 30.21678683,  
 28.23999745, 25.95204113])

Inertia can be recognized as a measure of how internally coherent clusters are.

Now, we visualize the Elbow Method so that we can determine the number of optimal clusters for our dataset.

fig, ax = plt.subplots(figsize=(15,7))  
 ax = plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")  
 plt.axvline(x=3, ls='--')  
 plt.ylabel('WCSS')  
 plt.xlabel('No. of Clusters (k)')  
 plt.title('The Elbow Method', fontsize = 20)  
 plt.show()

It is clear, that the optimal number of clusters for our data are 3, as the slope of the curve is not steep enough after it. When we observe this curve, we see that last elbow comes at k = 3, it would be difficult to visualize the elbow if we choose the higher range.

### Clustering

Now we will build the model for creating clusters from the dataset. We will use n\_clusters = 3 i.e. 3 clusters as we have determined by the elbow method, which would be optimal for our dataset.

Our data set is for unsupervised learning therefore we will use fit\_predict() Suppose we were working with supervised learning data set we would use fit\_tranform()

from sklearn.cluster import KMeans  
  
 kms = KMeans(n\_clusters=3, init='k-means++')  
 kms.fit(clustering\_data)

KMeans(n\_clusters=3)

Now that we have the clusters created, we will enter them into a different column

clusters = clustering\_data.copy()  
 clusters['Cluster\_Prediction'] = kms.fit\_predict(clustering\_data)  
 clusters.head()

SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \  
 0 5.1 3.5 1.4 0.2   
 1 4.9 3.0 1.4 0.2   
 2 4.7 3.2 1.3 0.2   
 3 4.6 3.1 1.5 0.2   
 4 5.0 3.6 1.4 0.2   
  
 Cluster\_Prediction   
 0 1   
 1 1   
 2 1   
 3 1   
 4 1

We can also get the centroids of the clusters by the cluster\_centers\_ attribute of KMeans algorithm.

kms.cluster\_centers\_

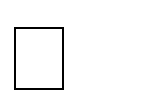
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],  
 [5.006 , 3.418 , 1.464 , 0.244 ],  
 [6.85 , 3.07368421, 5.74210526, 2.07105263]])

Now we have all the data we need, we just need to plot the data. We will plot the data using scatterplot which will allow us to observe different clusters in different colours.

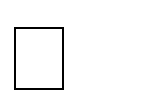
fig, ax = plt.subplots(figsize=(15,7))  
 plt.scatter(x=clusters[clusters['Cluster\_Prediction'] == 0]['SepalLengthCm'],  
 y=clusters[clusters['Cluster\_Prediction'] == 0]['SepalWidthCm'],  
 s=70,edgecolor='teal', linewidth=0.3, c='teal', label='Iris-versicolor')  
  
  
 plt.scatter(x=clusters[clusters['Cluster\_Prediction'] == 1]['SepalLengthCm'],  
 y=clusters[clusters['Cluster\_Prediction'] == 1]['SepalWidthCm'],  
 s=70,edgecolor='lime', linewidth=0.3, c='lime', label='Iris-setosa')  
  
  
 plt.scatter(x=clusters[clusters['Cluster\_Prediction'] == 2]['SepalLengthCm'],  
 y=clusters[clusters['Cluster\_Prediction'] == 2]['SepalWidthCm'],  
 s=70,edgecolor='magenta', linewidth=0.3, c='magenta', label='Iris-virginica')  
  
 plt.scatter(x=kms.cluster\_centers\_[:, 0], y=kms.cluster\_centers\_[:, 1], s = 170, c = 'yellow', label = 'Centroids',edgecolor='black', linewidth=0.3)  
 plt.legend(loc='upper right')  
 plt.xlim(4,8)  
 plt.ylim(1.8,4.5)  
 ax.set\_ylabel('Sepal Width (in cm)')  
 ax.set\_xlabel('Sepal Length (in cm)')  
 plt.title('Clusters', fontsize = 20)  
 plt.show()

**EXPERIMENT NO. 5 B**

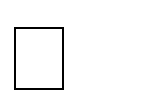
* **Aim**: Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient Tree Boosting (GBM), XGBoost classification on Iris dataset and compare the
* performance of three models using different evaluation measures.
* Dataset Link: https://www.kaggle.com/datasets/uciml/iris

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

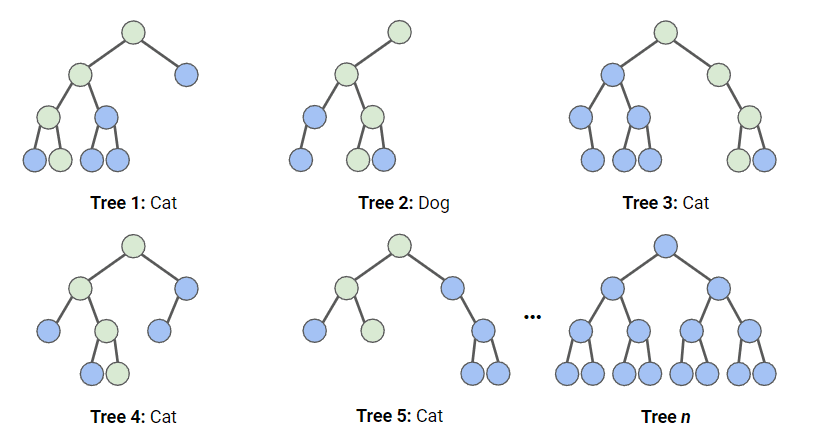
Jypiter Nootbook/Ubuntu

**Theory:**

Imagine you have a complex problem to solve, and you gather a group of experts from different fields to provide their input. Each expert provides their opinion based on their expertise and experience. Then, the experts would vote to arrive at a final decision.

In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result. (For regression, predictions use an averaging technique instead.)

In the diagram below, we have a random forest with n decision trees, and we’ve shown the first 5, along with their predictions (either “Dog” or “Cat”). Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction. Looking at the first 5 trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest would count the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.

****

**Implementation:**

import pandas as pd  
 from sklearn.datasets import load\_digits  
 digits = load\_digits()

dir(digits)

['DESCR', 'data', 'feature\_names', 'frame', 'images', 'target', 'target\_names']

%matplotlib inline  
 import matplotlib.pyplot as plt

plt.gray()  
 for i in range(4):  
 plt.matshow(digits.images[i])

<Figure size 640x480 with 0 Axes>

df = pd.DataFrame(digits.data)  
 df.head()

0 1 2 3 4 5 6 7 8 9 ... 54 55 56 \  
 0 0.0 0.0 5.0 13.0 9.0 1.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 1 0.0 0.0 0.0 12.0 13.0 5.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 2 0.0 0.0 0.0 4.0 15.0 12.0 0.0 0.0 0.0 0.0 ... 5.0 0.0 0.0   
 3 0.0 0.0 7.0 15.0 13.0 1.0 0.0 0.0 0.0 8.0 ... 9.0 0.0 0.0   
 4 0.0 0.0 0.0 1.0 11.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
  
 57 58 59 60 61 62 63   
 0 0.0 6.0 13.0 10.0 0.0 0.0 0.0   
 1 0.0 0.0 11.0 16.0 10.0 0.0 0.0   
 2 0.0 0.0 3.0 11.0 16.0 9.0 0.0   
 3 0.0 7.0 13.0 13.0 9.0 0.0 0.0   
 4 0.0 0.0 2.0 16.0 4.0 0.0 0.0   
  
 [5 rows x 64 columns]

df['target'] = digits.target

df[0:12]

0 1 2 3 4 5 6 7 8 9 ... 55 56 57 \  
 0 0.0 0.0 5.0 13.0 9.0 1.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 1 0.0 0.0 0.0 12.0 13.0 5.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 2 0.0 0.0 0.0 4.0 15.0 12.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 3 0.0 0.0 7.0 15.0 13.0 1.0 0.0 0.0 0.0 8.0 ... 0.0 0.0 0.0   
 4 0.0 0.0 0.0 1.0 11.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 5 0.0 0.0 12.0 10.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 6 0.0 0.0 0.0 12.0 13.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 7 0.0 0.0 7.0 8.0 13.0 16.0 15.0 1.0 0.0 0.0 ... 0.0 0.0 0.0   
 8 0.0 0.0 9.0 14.0 8.0 1.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 9 0.0 0.0 11.0 12.0 0.0 0.0 0.0 0.0 0.0 2.0 ... 0.0 0.0 0.0   
 10 0.0 0.0 1.0 9.0 15.0 11.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
 11 0.0 0.0 0.0 0.0 14.0 13.0 1.0 0.0 0.0 0.0 ... 0.0 0.0 0.0   
  
 58 59 60 61 62 63 target   
 0 6.0 13.0 10.0 0.0 0.0 0.0 0   
 1 0.0 11.0 16.0 10.0 0.0 0.0 1   
 2 0.0 3.0 11.0 16.0 9.0 0.0 2   
 3 7.0 13.0 13.0 9.0 0.0 0.0 3   
 4 0.0 2.0 16.0 4.0 0.0 0.0 4   
 5 9.0 16.0 16.0 10.0 0.0 0.0 5   
 6 1.0 9.0 15.0 11.0 3.0 0.0 6   
 7 13.0 5.0 0.0 0.0 0.0 0.0 7   
 8 11.0 16.0 15.0 11.0 1.0 0.0 8   
 9 9.0 12.0 13.0 3.0 0.0 0.0 9   
 10 1.0 10.0 13.0 3.0 0.0 0.0 0   
 11 0.0 1.0 13.0 16.0 1.0 0.0 1   
  
 [12 rows x 65 columns]

Train and the model and prediction

X = df.drop('target',axis='columns')  
 y = df.target

from sklearn.model\_selection import train\_test\_split  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2)

from sklearn.ensemble import RandomForestClassifier  
 model = RandomForestClassifier(n\_estimators=20)  
 model.fit(X\_train, y\_train)

RandomForestClassifier(n\_estimators=20)

model.score(X\_test, y\_test)

0.9805555555555555

y\_predicted = model.predict(X\_test)

Confusion Matrix

from sklearn.metrics import confusion\_matrix  
 cm = confusion\_matrix(y\_test, y\_predicted)  
 cm

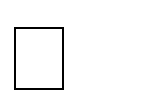
array([[32, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [ 0, 30, 0, 0, 0, 0, 0, 0, 0, 0],  
 [ 0, 0, 32, 0, 0, 0, 0, 0, 0, 0],  
 [ 0, 0, 0, 37, 0, 0, 0, 0, 0, 0],  
 [ 0, 0, 0, 0, 35, 0, 0, 0, 0, 0],  
 [ 0, 0, 0, 0, 0, 41, 1, 0, 0, 1],  
 [ 0, 0, 0, 0, 1, 0, 35, 0, 0, 0],  
 [ 0, 0, 0, 0, 0, 0, 0, 52, 0, 2],  
 [ 1, 0, 0, 0, 0, 0, 0, 0, 32, 0],  
 [ 0, 0, 0, 0, 1, 0, 0, 0, 0, 27]])

%matplotlib inline  
 import matplotlib.pyplot as plt  
 import seaborn as sn  
 plt.figure(figsize=(10,7))  
 sn.heatmap(cm, annot=True)  
 plt.xlabel('Predicted')  
 plt.ylabel('Truth')

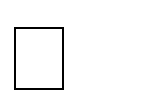
Text(95.72222222222221, 0.5, 'Truth')

**EXPERIMENT NO. 6 C**

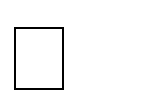
* **Aim**: Build a Tic-Tac-Toe game using reinforcement learning in Python by using following
* tasks
* a. Setting up the environment
* b. Defining the Tic-Tac-Toe game
* c. Building the reinforcement learning model
* d. Training the model
* e. Testing the model

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu

**Theory:**

In reinforcement learning, developers devise a method of rewarding desired behaviors and punishing negative behaviors. This method assigns positive values to the desired actions to encourage the agent and negative values to undesired behaviors. This programs the agent to seek long-term and maximum overall reward to achieve an optimal solution.

These long-term goals help prevent the agent from stalling on lesser goals. With time, the agent learns to avoid the negative and seek the positive. This learning method has been adopted in artificial intelligence ([AI](https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence)) as a way of directing [unsupervised machine learning](https://www.techtarget.com/whatis/definition/unsupervised-learning) through rewards and penalties.

### Common reinforcement learning algorithms

[Rather than referring to a specific algorithm](https://www.techtarget.com/searchenterpriseai/feature/5-types-of-machine-learning-algorithms-you-should-know), the field of reinforcement learning is made up of several algorithms that take somewhat different approaches. The differences are mainly due to their strategies for exploring their environments.

* State-action-reward-state-action (SARSA). This reinforcement learning algorithm starts by giving the agent what's known as a *policy*. The policy is essentially a probability that tells it the odds of certain actions resulting in rewards, or beneficial states.
* Q-learning. This approach to reinforcement learning takes the opposite approach. The agent receives no policy, meaning its exploration of its environment is more self-directed.
* Deep Q-Networks. These algorithms utilize neural networks in addition to reinforcement learning techniques. They utilize the self-directed environment exploration of reinforcement learning. Future actions are based on a random sample of past beneficial actions learned by the neural network.

**Implementation:**

import numpy as np  
  
 class TicTacToeEnvironment:  
 def \_\_init\_\_(self):  
 self.state = [0] \* 9  
 self.is\_terminal = False  
  
 def reset(self):  
 self.state = [0] \* 9  
 self.is\_terminal = False  
  
 def get\_available\_moves(self):  
 return [i for i, mark in enumerate(self.state) if mark == 0]  
  
 def make\_move(self, move, player\_mark):  
 self.state[move] = player\_mark  
  
 def check\_win(self, player\_mark):  
 winning\_states = [  
 [0, 1, 2], [3, 4, 5], [6, 7, 8], # rows  
 [0, 3, 6], [1, 4, 7], [2, 5, 8], # columns  
 [0, 4, 8], [2, 4, 6] # diagonals  
 ]  
 for state\_indices in winning\_states:  
 if all(self.state[i] == player\_mark for i in state\_indices):  
 self.is\_terminal = True  
 return True  
 return False  
  
 def is\_draw(self):  
 return 0 not in self.state

class QLearningAgent:  
 def \_\_init\_\_(self, learning\_rate=0.9, discount\_factor=0.9, exploration\_rate=0.3):  
 self.learning\_rate = learning\_rate  
 self.discount\_factor = discount\_factor  
 self.exploration\_rate = exploration\_rate  
 self.q\_table = np.zeros((3\*\*9, 9))  
  
 def get\_state\_index(self, state):  
 state\_index = 0  
 for i, mark in enumerate(state):  
 state\_index += (3 \*\* i) \* (mark + 1)  
 return state\_index  
  
 def choose\_action(self, state, available\_moves):  
 state\_index = self.get\_state\_index(state)  
 if np.random.random() < self.exploration\_rate:  
 return np.random.choice(available\_moves)  
 else:  
 return np.argmax(self.q\_table[state\_index, available\_moves])  
  
 def update\_q\_table(self, state, action, next\_state, reward):  
 state\_index = self.get\_state\_index(state)  
 next\_state\_index = self.get\_state\_index(next\_state) if next\_state is not None else None  
 max\_q\_value = np.max(self.q\_table[next\_state\_index]) if next\_state is not None else 0  
 self.q\_table[state\_index, action] = (1 - self.learning\_rate) \* self.q\_table[state\_index, action] + \  
 self.learning\_rate \* (reward + self.discount\_factor \* max\_q\_value)

def evaluate\_agents(agent1, agent2, num\_episodes=1000):  
 environment = TicTacToeEnvironment()  
 agent1\_wins = 0  
 agent2\_wins = 0  
 draws = 0  
  
 for \_ in range(num\_episodes):  
 environment.reset()  
 current\_agent = agent1  
 while not environment.is\_terminal:  
 available\_moves = environment.get\_available\_moves()  
 current\_state = environment.state.copy()  
 action = current\_agent.choose\_action(current\_state, available\_moves)  
 environment.make\_move(action, 1 if current\_agent == agent1 else -1)  
  
 if environment.check\_win(1 if current\_agent == agent1 else -1):  
 current\_agent.update\_q\_table(current\_state, action, None, 10)  
 if current\_agent == agent1:  
 agent1\_wins += 1  
 else:  
 agent2\_wins += 1  
 break  
 elif environment.is\_draw():  
 current\_agent.update\_q\_table(current\_state, action, None, 0)  
 draws += 1  
 break  
  
 next\_state = environment.state.copy()  
 reward = 0  
 if environment.check\_win(1 if current\_agent == agent1 else -1):  
 reward = -10  
 current\_agent.update\_q\_table(current\_state, action, next\_state, reward)  
  
 current\_agent = agent2 if current\_agent == agent1 else agent1  
  
 return agent1\_wins, agent2\_wins, draws

# Create agents  
 agent1 = QLearningAgent()  
 agent2 = QLearningAgent()  
  
 # Evaluate agents  
 agent1\_wins, agent2\_wins, draws = evaluate\_agents(agent1, agent2)  
  
 # Print results  
 print(f"Agent 1 wins: {agent1\_wins}")  
 print(f"Agent 2 wins: {agent2\_wins}")  
 print(f"Draws: {draws}")

Agent 1 wins: 458  
 Agent 2 wins: 470  
 Draws: 72

TicTacToeEnvironment:  
 This class represents the Tic-Tac-Toe game environment. It maintains the current state of the game, checks for a win or draw, and provides methods to reset the game and make moves.  
 The \_\_init\_\_ method initializes the game state and sets the terminal flag to False.  
 The reset method resets the game state and the terminal flag.  
 The get\_available\_moves method returns a list of indices representing the available moves in the current game state.  
 The make\_move method updates the game state by placing a player's mark at the specified move index.  
 The check\_win method checks if a player has won the game by examining the current state.  
 The is\_draw method checks if the game has ended in a draw.  
  
 QLearningAgent:  
 This class represents the Q-learning agent. It learns to play Tic-Tac-Toe by updating a Q-table based on the rewards received during gameplay.  
 The \_\_init\_\_ method initializes the learning rate, discount factor, exploration rate, and the Q-table.  
 The get\_state\_index method converts the current game state into a unique index for indexing the Q-table.  
 The choose\_action method selects the action (move) to be taken based on the current game state and the exploration-exploitation tradeoff using the epsilon-greedy policy.  
 The update\_q\_table method updates the Q-table based on the current state, action, next state, and the reward received.  
  
 evaluate\_agents:  
 This function performs the evaluation of two Q-learning agents by playing multiple episodes of Tic-Tac-Toe games.  
 It takes the two agents and the number of episodes to play as input.  
 In each episode, the environment is reset, and the agents take turns making moves until the game is over (either a win or a draw).  
 The agents update their Q-tables based on the rewards received during the episode.  
 The function keeps track of the wins and draws for each agent and returns the counts.  
  
 Main code:  
 The main code creates two Q-learning agents, agent1 and agent2, using the QLearningAgent class.  
 The evaluate\_agents function is called to evaluate the agents by playing a specified number of episodes.  
 The results (number of wins and draws) for each agent are printed.

The Q-learning algorithm involves the following steps:

The agents choose their moves based on the current game state and the exploration-exploitation policy.  
 The environment updates the game state based on the chosen moves.  
 The environment checks if the game has ended (win or draw).  
 The agents update their Q-tables based on the rewards received.  
 The agents continue playing until the specified number of episodes is completed.

**EXPERIMENT NO. 8 (Group B)**

* **Aim**: Interacting with Web APIs
* **Problem Statement:** Analyzing Weather Data from OpenWeatherMap API
* **Dataset:** Weather data retrieved from OpenWeatherMap API
* **Description:** The goal is to interact with the OpenWeatherMap API to retrieve weather data
* for a specific location and perform data modeling and visualization to analyze weather
* patterns over time.

Tasks to Perform:

1. Register and obtain API key from OpenWeatherMap.

2. Interact with the OpenWeatherMap API using the API key to retrieve weather data for

a specific location.

3. Extract relevant weather attributes such as temperature, humidity, wind speed, and

precipitation from the API response.

4. Clean and preprocess the retrieved data, handling missing values or inconsistent

formats.

5. Perform data modeling to analyze weather patterns, such as calculating average

temperature, maximum/minimum values, or trends over time.

6. Visualize the weather data using appropriate plots, such as line charts, bar plots, or

scatter plots, to represent temperature changes, precipitation levels, or wind speed

variations.

7. Apply data aggregation techniques to summarize weather statistics by specific time

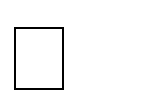
periods (e.g., daily, monthly, seasonal).

8. Incorporate geographical information, if available, to create maps or geospatial

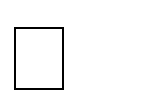
visualizations representing weather patterns across different locations.

9. Explore and visualize relationships between weather attributes, such as temperature

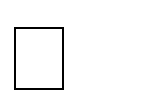
and humidity, using correlation plots or heatmaps.

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu



**Implementation:**

import requests  
 import pandas as pd  
 import datetime  
  
 # Set your OpenWeatherMap API key  
 api\_key = 'fb365aa6104829b44455572365ff3b4e'

Get the lat(itude) and lon(gitude) from [here](https://www.latlong.net/).

# Set the location for which you want to retrieve weather data  
 lat = 18.184135  
 lon = 74.610764  
  
 #https://openweathermap.org/api/one-call-3#how How to use api call  
 # Construct the API URL  
 api\_url = f"http://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={api\_key}"

# Send a GET request to the API  
 response = requests.get(api\_url)  
 weather\_data = response.json() #pass response to weather\_data object(dictionary)

weather\_data.keys()

dict\_keys(['cod', 'message', 'cnt', 'list', 'city'])

weather\_data['list'][0]

{'dt': 1690189200,  
 'main': {'temp': 298.21,  
 'feels\_like': 298.81,  
 'temp\_min': 298.1,  
 'temp\_max': 298.21,  
 'pressure': 1006,  
 'sea\_level': 1006,  
 'grnd\_level': 942,  
 'humidity': 78,  
 'temp\_kf': 0.11},  
 'weather': [{'id': 804,  
 'main': 'Clouds',  
 'description': 'overcast clouds',  
 'icon': '04d'}],  
 'clouds': {'all': 100},  
 'wind': {'speed': 6.85, 'deg': 258, 'gust': 12.9},  
 'visibility': 10000,  
 'pop': 0.59,  
 'sys': {'pod': 'd'},  
 'dt\_txt': '2023-07-24 09:00:00'}

len(weather\_data['list'])

40

weather\_data['list'][0]['weather'][0]['description']

{"type":"string"}

#getting the data from dictionary and taking into one variable  
 # Extract relevant weather attributes using list comprehension  
 temperatures = [item['main']['temp'] for item in weather\_data['list']] #it will extract all values (40) and putting into one variable  
 timestamps = [pd.to\_datetime(item['dt'], unit='s') for item in weather\_data['list']]  
 temperature = [item['main']['temp'] for item in weather\_data['list']]  
 humidity = [item['main']['humidity'] for item in weather\_data['list']]  
 wind\_speed = [item['wind']['speed'] for item in weather\_data['list']]  
 weather\_description = [item['weather'][0]['description'] for item in weather\_data['list']]  
  
 # Create a pandas DataFrame with the extracted weather data  
 weather\_df = pd.DataFrame({  
 'Timestamp': timestamps,  
 'Temperature': temperatures,  
 'humidity': humidity,  
 'wind\_speed': wind\_speed,  
 'weather\_description': weather\_description,  
 })  
  
 # Set the Timestamp column as the DataFrame's index  
 weather\_df.set\_index('Timestamp', inplace=True)

max\_temp = weather\_df['Temperature'].max()

max\_temp

298.9

min\_temp = weather\_df['Temperature'].min()

min\_temp

294.92

# Clean and preprocess the data  
  
 # Handling missing values  
 weather\_df.fillna(0, inplace=True) # Replace missing values with 0 or appropriate value  
  
 # Handling inconsistent format (if applicable)  
 weather\_df['Temperature'] = weather\_df['Temperature'].apply(lambda x: x - 273.15 if isinstance(x, float) else x) # Convert temperature from Kelvin to Celsius  
  
 # Print the cleaned and preprocessed data  
 print(weather\_df)

Temperature humidity wind\_speed weather\_description  
 Timestamp   
 2023-07-24 09:00:00 25.06 78 6.85 overcast clouds  
 2023-07-24 12:00:00 24.52 81 6.92 light rain  
 2023-07-24 15:00:00 23.73 84 7.18 light rain  
 2023-07-24 18:00:00 23.69 83 6.44 light rain  
 2023-07-24 21:00:00 23.06 85 5.54 light rain  
 2023-07-25 00:00:00 22.28 92 4.57 moderate rain  
 2023-07-25 03:00:00 22.46 92 3.95 moderate rain  
 2023-07-25 06:00:00 22.98 90 6.10 moderate rain  
 2023-07-25 09:00:00 24.55 79 6.46 light rain  
 2023-07-25 12:00:00 23.53 84 5.00 light rain  
 2023-07-25 15:00:00 22.87 88 5.00 overcast clouds  
 2023-07-25 18:00:00 22.77 89 3.93 overcast clouds  
 2023-07-25 21:00:00 22.56 84 5.47 overcast clouds  
 2023-07-26 00:00:00 22.35 87 3.97 overcast clouds  
 2023-07-26 03:00:00 23.05 85 3.47 light rain  
 2023-07-26 06:00:00 23.34 85 3.84 light rain  
 2023-07-26 09:00:00 23.08 89 4.16 light rain  
 2023-07-26 12:00:00 24.09 83 5.52 light rain  
 2023-07-26 15:00:00 23.10 87 5.59 light rain  
 2023-07-26 18:00:00 22.43 91 5.42 light rain  
 2023-07-26 21:00:00 22.29 92 5.17 light rain  
 2023-07-27 00:00:00 22.53 90 5.31 light rain  
 2023-07-27 03:00:00 22.78 88 4.30 light rain  
 2023-07-27 06:00:00 22.83 90 5.19 moderate rain  
 2023-07-27 09:00:00 22.57 91 6.65 moderate rain  
 2023-07-27 12:00:00 22.28 91 5.27 moderate rain  
 2023-07-27 15:00:00 22.03 93 5.12 light rain  
 2023-07-27 18:00:00 21.82 92 4.65 light rain  
 2023-07-27 21:00:00 21.77 90 5.27 light rain  
 2023-07-28 00:00:00 22.01 88 5.41 light rain  
 2023-07-28 03:00:00 23.30 81 6.19 overcast clouds  
 2023-07-28 06:00:00 25.19 72 7.19 light rain  
 2023-07-28 09:00:00 24.95 76 7.22 light rain  
 2023-07-28 12:00:00 24.72 75 6.93 overcast clouds  
 2023-07-28 15:00:00 23.41 83 5.12 overcast clouds  
 2023-07-28 18:00:00 22.76 86 4.56 overcast clouds  
 2023-07-28 21:00:00 22.63 87 4.15 overcast clouds  
 2023-07-29 00:00:00 22.74 84 4.35 overcast clouds  
 2023-07-29 03:00:00 23.87 77 6.16 overcast clouds  
 2023-07-29 06:00:00 25.75 66 7.23 overcast clouds

import matplotlib.pyplot as plt  
  
 daily\_mean\_temp = weather\_df['Temperature'].resample('D').mean()  
 daily\_mean\_humidity = weather\_df['humidity'].resample('D').mean()  
 daily\_mean\_wind\_speed = weather\_df['wind\_speed'].resample('D').mean()  
  
 # Plot the mean daily temperature over time (Line plot)  
 plt.figure(figsize=(10, 6))  
 daily\_mean\_temp.plot(color='red', linestyle='-', marker='o')  
 plt.title('Mean Daily Temperature')  
 plt.xlabel('Date')  
 plt.ylabel('Temperature (°C)')  
 plt.grid(True)  
 plt.show()  
  
 # Plot the mean daily humidity over time (Bar plot)  
 plt.figure(figsize=(10, 6))  
 daily\_mean\_humidity.plot(kind='bar', color='blue')  
 plt.title('Mean Daily Humidity')  
 plt.xlabel('Date')  
 plt.ylabel('Humidity (%)')  
 plt.grid(True)  
 plt.show()  
  
 # Plot the relationship between temperature and wind speed (Scatter plot)  
 plt.figure(figsize=(10, 6))  
 plt.scatter(weather\_df['Temperature'], weather\_df['wind\_speed'], color='green')  
 plt.title('Temperature vs. Wind Speed')  
 plt.xlabel('Temperature (°C)')  
 plt.ylabel('Wind Speed (m/s)')  
 plt.grid(True)  
 plt.show()

###Heatmap

import seaborn as sns  
  
 heatmap\_data = weather\_df[['Temperature', 'humidity']]  
 sns.heatmap(heatmap\_data, annot=True, cmap='coolwarm')  
 plt.title('Temperature vs Humidity Heatmap')  
 plt.show()  
  
 # Create a scatter plot to visualize the relationship between temperature and humidity  
 plt.scatter(weather\_df['Temperature'], weather\_df['humidity'])  
 plt.xlabel('Temperature (°C)')  
 plt.ylabel('Humidity (%)')  
 plt.title('Temperature vs Humidity Scatter Plot')  
 plt.show()

###Geospatial Map

import requests  
 import pandas as pd  
 import geopandas as gpd  
 import folium  
  
 # Set your OpenWeatherMap API key  
 api\_key = 'fb365aa6104829b44455572365ff3b4e'  
  
 # Specify the locations for which you want to retrieve weather data  
 locations = ['London', 'Paris', 'New York']  
  
 weather\_df = pd.DataFrame()  
  
 # Retrieve weather data for each location  
 for location in locations:  
 # Construct the API URL  
 api\_url = f'http://api.openweathermap.org/data/2.5/weather?q={location}&appid={api\_key}'  
  
 # Send a GET request to the API  
 response = requests.get(api\_url)  
 weather\_data = response.json()  
  
 # Extract relevant weather attributes  
 temperature = weather\_data['main']['temp']  
 humidity = weather\_data['main']['humidity']  
 wind\_speed = weather\_data['wind']['speed']  
 latitude = weather\_data['coord']['lat']  
 longitude = weather\_data['coord']['lon']  
  
 # Create a DataFrame for the location's weather data  
 location\_df = pd.DataFrame({  
 'Location': [location],  
 'Temperature': [temperature],  
 'Humidity': [humidity],  
 'Wind Speed': [wind\_speed],  
 'Latitude': [latitude],  
 'Longitude': [longitude]  
 })  
  
 # Append the location's weather data to the main DataFrame  
 weather\_df = weather\_df.append(location\_df, ignore\_index=True)

<ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 weather\_df = weather\_df.append(location\_df, ignore\_index=True)  
 <ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 weather\_df = weather\_df.append(location\_df, ignore\_index=True)  
 <ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 weather\_df = weather\_df.append(location\_df, ignore\_index=True)

weather\_df

Location Temperature Humidity Wind Speed Latitude Longitude  
 0 London 289.02 88 3.60 51.5085 -0.1257  
 1 Paris 290.96 83 6.17 48.8534 2.3488  
 2 New York 296.82 61 4.47 40.7143 -74.0060

# Load a world map shapefile using geopandas  
 world\_map = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_cities'))  
  
 # Rename the column used for merging in the world map DataFrame  
 world\_map.rename(columns={'name': 'Location'}, inplace=True)  
  
 # Merge the weather data with the world map based on location  
 weather\_map = world\_map.merge(weather\_df, on='Location')  
  
 # Create a folium map centered around the mean latitude and longitude of all locations  
 map\_center = [weather\_df['Latitude'].mean(), weather\_df['Longitude'].mean()]  
 weather\_map\_folium = folium.Map(location=map\_center, zoom\_start=2)  
  
 # Add weather markers to the folium map  
 for index, row in weather\_map.iterrows():  
 location = [row['Latitude'], row['Longitude']]  
 temperature = row['Temperature']  
 marker\_text = f'Temperature: {temperature} K'  
 folium.Marker(location, popup=marker\_text, icon=folium.Icon(icon='cloud', color='red')).add\_to(weather\_map\_folium)  
  
 # display the folium map  
 weather\_map\_folium

<ipython-input-19-c9bd718791be>:2: FutureWarning: The geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0. You can get the original 'naturalearth\_cities' data from https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.  
 world\_map = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_cities'))

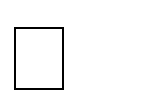
<folium.folium.Map at 0x7f242a56f430>

type(weather\_map\_folium)

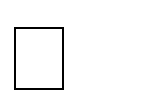
folium.folium.Map

**EXPERIMENT NO. 9 (Group B)**

* **Aim**: Data Cleaning and Preparation
* **Problem Statement:** Analyzing Customer Churn in a Telecommunications Company
* **Dataset:** "Telecom\_Customer\_Churn.csv"
* **Description:** The dataset contains information about customers of a telecommunications company and whether they have churned (i.e., discontinued their services). The dataset includes various attributes of the customers, such as their demographics, usage patterns, and account information. The goal is to perform data cleaning and preparation to gain insights into the factors that contribute to customer churn.
* **Tasks to Perform:**
* 1. Import the "Telecom\_Customer\_Churn.csv" dataset.
* 2. Explore the dataset to understand its structure and content.
* 3. Handle missing values in the dataset, deciding on an appropriate strategy.
* 4. Remove any duplicate records from the dataset.
* 5. Check for inconsistent data, such as inconsistent formatting or spelling variations,
* and standardize it.
* 6. Convert columns to the correct data types as needed.
* 7. Identify and handle outliers in the data.

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu

**Implementation:**

Import necessary libraries

import pandas as pd #data manipulation  
 import numpy as np #numerical computations  
 from sklearn.model\_selection import train\_test\_split # scikit-learn for machine learning models split the dataset into training and testing sets for model evaluation  
 from sklearn import metrics #evaluating the performance of machine learning models

Load the dataset

data = pd.read\_csv("Telecom\_Customer\_Churn.csv")  
 print(data.index)

RangeIndex(start=0, stop=7043, step=1)

Explore the dataset

print(data)

customerID gender SeniorCitizen Partner Dependents tenure \  
 0 7590-VHVEG Female 0 Yes No 1   
 1 5575-GNVDE Male 0 No No 34   
 2 3668-QPYBK Male 0 No No 2   
 3 7795-CFOCW Male 0 No No 45   
 4 9237-HQITU Female 0 No No 2   
 ... ... ... ... ... ... ...   
 7038 6840-RESVB Male 0 Yes Yes 24   
 7039 2234-XADUH Female 0 Yes Yes 72   
 7040 4801-JZAZL Female 0 Yes Yes 11   
 7041 8361-LTMKD Male 1 Yes No 4   
 7042 3186-AJIEK Male 0 No No 66   
  
 PhoneService MultipleLines InternetService OnlineSecurity ... \  
 0 No No phone service DSL No ...   
 1 Yes No DSL Yes ...   
 2 Yes No DSL Yes ...   
 3 No No phone service DSL Yes ...   
 4 Yes No Fiber optic No ...   
 ... ... ... ... ... ...   
 7038 Yes Yes DSL Yes ...   
 7039 Yes Yes Fiber optic No ...   
 7040 No No phone service DSL Yes ...   
 7041 Yes Yes Fiber optic No ...   
 7042 Yes No Fiber optic Yes ...   
  
 DeviceProtection TechSupport StreamingTV StreamingMovies Contract \  
 0 No No No No Month-to-month   
 1 Yes No No No One year   
 2 No No No No Month-to-month   
 3 Yes Yes No No One year   
 4 No No No No Month-to-month   
 ... ... ... ... ... ...   
 7038 Yes Yes Yes Yes One year   
 7039 Yes No Yes Yes One year   
 7040 No No No No Month-to-month   
 7041 No No No No Month-to-month   
 7042 Yes Yes Yes Yes Two year   
  
 PaperlessBilling PaymentMethod MonthlyCharges TotalCharges \  
 0 Yes Electronic check 29.85 29.85   
 1 No Mailed check 56.95 1889.5   
 2 Yes Mailed check 53.85 108.15   
 3 No Bank transfer (automatic) 42.30 1840.75   
 4 Yes Electronic check 70.70 151.65   
 ... ... ... ... ...   
 7038 Yes Mailed check 84.80 1990.5   
 7039 Yes Credit card (automatic) 103.20 7362.9   
 7040 Yes Electronic check 29.60 346.45   
 7041 Yes Mailed check 74.40 306.6   
 7042 Yes Bank transfer (automatic) 105.65 6844.5   
  
 Churn   
 0 No   
 1 No   
 2 Yes   
 3 No   
 4 Yes   
 ... ...   
 7038 No   
 7039 No   
 7040 No   
 7041 Yes   
 7042 No   
  
 [7043 rows x 21 columns]

print(data.columns)

Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',  
 'tenure', 'PhoneService', 'MultipleLines', 'InternetService',  
 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',  
 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',  
 'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],  
 dtype='object')

data.shape

(7043, 21)

print(data.head())

customerID gender SeniorCitizen Partner Dependents tenure PhoneService \  
 0 7590-VHVEG Female 0 Yes No 1 No   
 1 5575-GNVDE Male 0 No No 34 Yes   
 2 3668-QPYBK Male 0 No No 2 Yes   
 3 7795-CFOCW Male 0 No No 45 No   
 4 9237-HQITU Female 0 No No 2 Yes   
  
 MultipleLines InternetService OnlineSecurity ... DeviceProtection \  
 0 No phone service DSL No ... No   
 1 No DSL Yes ... Yes   
 2 No DSL Yes ... No   
 3 No phone service DSL Yes ... Yes   
 4 No Fiber optic No ... No   
  
 TechSupport StreamingTV StreamingMovies Contract PaperlessBilling \  
 0 No No No Month-to-month Yes   
 1 No No No One year No   
 2 No No No Month-to-month Yes   
 3 Yes No No One year No   
 4 No No No Month-to-month Yes   
  
 PaymentMethod MonthlyCharges TotalCharges Churn   
 0 Electronic check 29.85 29.85 No   
 1 Mailed check 56.95 1889.5 No   
 2 Mailed check 53.85 108.15 Yes   
 3 Bank transfer (automatic) 42.30 1840.75 No   
 4 Electronic check 70.70 151.65 Yes   
  
 [5 rows x 21 columns]

print(data.tail())

customerID gender SeniorCitizen Partner Dependents tenure \  
 7038 6840-RESVB Male 0 Yes Yes 24   
 7039 2234-XADUH Female 0 Yes Yes 72   
 7040 4801-JZAZL Female 0 Yes Yes 11   
 7041 8361-LTMKD Male 1 Yes No 4   
 7042 3186-AJIEK Male 0 No No 66   
  
 PhoneService MultipleLines InternetService OnlineSecurity ... \  
 7038 Yes Yes DSL Yes ...   
 7039 Yes Yes Fiber optic No ...   
 7040 No No phone service DSL Yes ...   
 7041 Yes Yes Fiber optic No ...   
 7042 Yes No Fiber optic Yes ...   
  
 DeviceProtection TechSupport StreamingTV StreamingMovies Contract \  
 7038 Yes Yes Yes Yes One year   
 7039 Yes No Yes Yes One year   
 7040 No No No No Month-to-month   
 7041 No No No No Month-to-month   
 7042 Yes Yes Yes Yes Two year   
  
 PaperlessBilling PaymentMethod MonthlyCharges TotalCharges \  
 7038 Yes Mailed check 84.80 1990.5   
 7039 Yes Credit card (automatic) 103.20 7362.9   
 7040 Yes Electronic check 29.60 346.45   
 7041 Yes Mailed check 74.40 306.6   
 7042 Yes Bank transfer (automatic) 105.65 6844.5   
  
 Churn   
 7038 No   
 7039 No   
 7040 No   
 7041 Yes   
 7042 No   
  
 [5 rows x 21 columns]

# to know unique values  
 data.nunique()

customerID 7043  
 gender 2  
 SeniorCitizen 2  
 Partner 2  
 Dependents 2  
 tenure 73  
 PhoneService 2  
 MultipleLines 3  
 InternetService 3  
 OnlineSecurity 3  
 OnlineBackup 3  
 DeviceProtection 3  
 TechSupport 3  
 StreamingTV 3  
 StreamingMovies 3  
 Contract 3  
 PaperlessBilling 2  
 PaymentMethod 4  
 MonthlyCharges 1585  
 TotalCharges 6531  
 Churn 2  
 dtype: int64

Handle Missing Values

# data.isna().sum() is used to count the number of missing values (NaN values) in each column of a pandas DataFrame called data.  
 data.isna().sum()

customerID 0  
 gender 0  
 SeniorCitizen 0  
 Partner 0  
 Dependents 0  
 tenure 0  
 PhoneService 0  
 MultipleLines 0  
 InternetService 0  
 OnlineSecurity 0  
 OnlineBackup 0  
 DeviceProtection 0  
 TechSupport 0  
 StreamingTV 0  
 StreamingMovies 0  
 Contract 0  
 PaperlessBilling 0  
 PaymentMethod 0  
 MonthlyCharges 0  
 TotalCharges 0  
 Churn 0  
 dtype: int64

# isna() and isnull() are essentially the same method in Pandas, and they both return a boolean mask of the same shape as the input object, indicating where missing values (NaN or None) are present.  
 data.isnull().sum()

customerID 0  
 gender 0  
 SeniorCitizen 0  
 Partner 0  
 Dependents 0  
 tenure 0  
 PhoneService 0  
 MultipleLines 0  
 InternetService 0  
 OnlineSecurity 0  
 OnlineBackup 0  
 DeviceProtection 0  
 TechSupport 0  
 StreamingTV 0  
 StreamingMovies 0  
 Contract 0  
 PaperlessBilling 0  
 PaymentMethod 0  
 MonthlyCharges 0  
 TotalCharges 0  
 Churn 0  
 dtype: int64

Remove Duplicate Records

# Check the number of rows before removing duplicates  
 print("Number of rows before removing duplicates:", len(data))

Number of rows before removing duplicates: 7043

# Remove duplicate records  
 data\_cleaned = data.drop\_duplicates()

# Check the number of rows after removing duplicates  
 print("Number of rows after removing duplicates:", len(data\_cleaned))

Number of rows after removing duplicates: 7043

data.describe()

SeniorCitizen tenure MonthlyCharges  
 count 7043.000000 7043.000000 7043.000000  
 mean 0.162147 32.371149 64.761692  
 std 0.368612 24.559481 30.090047  
 min 0.000000 0.000000 18.250000  
 25% 0.000000 9.000000 35.500000  
 50% 0.000000 29.000000 70.350000  
 75% 0.000000 55.000000 89.850000  
 max 1.000000 72.000000 118.750000

#Measure of frequency destribution  
 unique, counts = np.unique(data['tenure'], return\_counts=True)  
 print(unique, counts)

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23  
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47  
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71  
 72] [ 11 613 238 200 176 133 110 131 123 119 116 99 117 109 76 99 80 87  
 97 73 71 63 90 85 94 79 79 72 57 72 72 65 69 64 65 88  
 50 65 59 56 64 70 65 65 51 61 74 68 64 66 68 68 80 70  
 68 64 80 65 67 60 76 76 70 72 80 76 89 98 100 95 119 170  
 362]

#Measure of frequency destribution  
 unique, counts = np.unique(data['MonthlyCharges'], return\_counts=True)  
 print(unique, counts)

[ 18.25 18.4 18.55 ... 118.6 118.65 118.75] [1 1 1 ... 2 1 1]

#Measure of frequency destribution  
 unique, counts = np.unique(data['TotalCharges'], return\_counts=True)  
 print(unique, counts)

[' ' '100.2' '100.25' ... '999.45' '999.8' '999.9'] [11 1 1 ... 1 1 1]

# sns.pairplot(data) creates a grid of pairwise plots of the variables in a dataset, which can help you quickly visualize the relationships between different pairs of variables.  
 import seaborn as sns #Seaborn library for data visualization  
 sns.pairplot(data)

<seaborn.axisgrid.PairGrid at 0x7fb9cc97a680>

Check for Outliers

#checking boxplot for Fare column  
 import matplotlib.pyplot as plt #pyplot module from the Matplotlib library  
 plt.boxplot(data['tenure'])  
 plt.show()

plt.boxplot(data['MonthlyCharges'])  
 plt.show()

Split the Data

X = data.drop("Churn", axis=1)  
 y = data["Churn"]  
 # Split the dataset into training and testing sets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train.shape

(5634, 20)

y\_train.shape

(5634,)

X\_test.shape

(1409, 20)

y\_test.shape

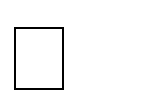
(1409,)

Export the cleaned data

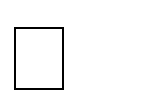
# Export the cleaned dataset to a CSV file  
 data.to\_csv("Cleaned\_Telecom\_Customer\_Churn.csv", index=False)

**EXPERIMENT NO. 10 (Group B)**

* **Aim**: Data Wrangling
* **Problem Statement:** Data Wrangling on Real Estate Market
* **Dataset:** "RealEstate\_Prices.csv"
* **Description:** The dataset contains information about housing prices in a specific real estate market. It includes various attributes such as property characteristics, location, sale prices, and other relevant features. The goal is to perform data wrangling to gain insights into the factors influencing housing prices and prepare the dataset for further analysis or modeling.
* **Tasks to Perform:**
* 1. Import the "RealEstate\_Prices.csv" dataset. Clean column names by removing spaces,
* special characters, or renaming them for clarity.
* 2. Handle missing values in the dataset, deciding on an appropriate strategy (e.g.,
* imputation or removal).
* 3. Perform data merging if additional datasets with relevant information are available
* (e.g., neighborhood demographics or nearby amenities).
* 4. Filter and subset the data based on specific criteria, such as a particular time period,
* property type, or location.
* 5. Handle categorical variables by encoding them appropriately (e.g., one-hot encoding
* or label encoding) for further analysis.
* 6. Aggregate the data to calculate summary statistics or derived metrics such as average
* sale prices by neighborhood or property type.
* 7. Identify and handle outliers or extreme values in the data that may affect the analysis
* or modeling process.

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu

**Implementation:**

import pandas as pd  
 import numpy as np  
 from matplotlib import pyplot as plt  
 %matplotlib inline  
 import matplotlib  
 matplotlib.rcParams["figure.figsize"] = (20,10)

Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging.

df1 = pd.read\_csv("/content/Bengaluru\_House\_Data.csv")  
 df1.head()

area\_type availability location size \  
 0 Super built-up Area 19-Dec Electronic City Phase II 2 BHK   
 1 Plot Area Ready To Move Chikka Tirupathi 4 Bedroom   
 2 Built-up Area Ready To Move Uttarahalli 3 BHK   
 3 Super built-up Area Ready To Move Lingadheeranahalli 3 BHK   
 4 Super built-up Area Ready To Move Kothanur 2 BHK   
  
 society total\_sqft bath balcony price   
 0 Coomee 1056 2.0 1.0 39.07   
 1 Theanmp 2600 5.0 3.0 120.00   
 2 NaN 1440 2.0 3.0 62.00   
 3 Soiewre 1521 3.0 1.0 95.00   
 4 NaN 1200 2.0 1.0 51.00

df1.shape

(13320, 9)

df1.columns

Index(['area\_type', 'availability', 'location', 'size', 'society',  
 'total\_sqft', 'bath', 'balcony', 'price'],  
 dtype='object')

df1['area\_type']

0 Super built-up Area  
 1 Plot Area  
 2 Built-up Area  
 3 Super built-up Area  
 4 Super built-up Area  
 ...   
 13315 Built-up Area  
 13316 Super built-up Area  
 13317 Built-up Area  
 13318 Super built-up Area  
 13319 Super built-up Area  
 Name: area\_type, Length: 13320, dtype: object

df1['area\_type'].unique()

array(['Super built-up Area', 'Plot Area', 'Built-up Area',  
 'Carpet Area'], dtype=object)

df1['area\_type'].value\_counts()

Super built-up Area 8790  
 Built-up Area 2418  
 Plot Area 2025  
 Carpet Area 87  
 Name: area\_type, dtype: int64

Drop features that are not required to build our model

df2 = df1.drop(['area\_type','society','balcony','availability'],axis='columns')  
 df2.shape

(13320, 5)

df2.isnull().sum()

location 1  
 size 16  
 total\_sqft 0  
 bath 73  
 price 0  
 dtype: int64

df2.shape

(13320, 5)

df3 = df2.dropna()  
 df3.isnull().sum()

location 0  
 size 0  
 total\_sqft 0  
 bath 0  
 price 0  
 dtype: int64

df3.shape

(13246, 5)

df3['size'].unique()

array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',  
 '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',  
 '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',  
 '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',  
 '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',  
 '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)

df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))

<ipython-input-15-4c4c73fbe7f4>:1: SettingWithCopyWarning:  
 A value is trying to be set on a copy of a slice from a DataFrame.  
 Try using .loc[row\_indexer,col\_indexer] = value instead  
  
 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))

df3.head()

location size total\_sqft bath price bhk  
 0 Electronic City Phase II 2 BHK 1056 2.0 39.07 2  
 1 Chikka Tirupathi 4 Bedroom 2600 5.0 120.00 4  
 2 Uttarahalli 3 BHK 1440 2.0 62.00 3  
 3 Lingadheeranahalli 3 BHK 1521 3.0 95.00 3  
 4 Kothanur 2 BHK 1200 2.0 51.00 2

df3.bhk.unique()

array([ 2, 4, 3, 6, 1, 8, 7, 5, 11, 9, 27, 10, 19, 16, 43, 14, 12,  
 13, 18])

df3[df3.bhk>20]

location size total\_sqft bath price bhk  
 1718 2Electronic City Phase II 27 BHK 8000 27.0 230.0 27  
 4684 Munnekollal 43 Bedroom 2400 40.0 660.0 43

df3.total\_sqft.unique()

array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],  
 dtype=object)

Explore total\_sqft feature

def is\_float(x):  
 try:  
 float(x)  
 except:  
 return False  
 return True

df3[~df3['total\_sqft'].apply(is\_float)].head(10)

location size total\_sqft bath price bhk  
 30 Yelahanka 4 BHK 2100 - 2850 4.0 186.000 4  
 122 Hebbal 4 BHK 3067 - 8156 4.0 477.000 4  
 137 8th Phase JP Nagar 2 BHK 1042 - 1105 2.0 54.005 2  
 165 Sarjapur 2 BHK 1145 - 1340 2.0 43.490 2  
 188 KR Puram 2 BHK 1015 - 1540 2.0 56.800 2  
 410 Kengeri 1 BHK 34.46Sq. Meter 1.0 18.500 1  
 549 Hennur Road 2 BHK 1195 - 1440 2.0 63.770 2  
 648 Arekere 9 Bedroom 4125Perch 9.0 265.000 9  
 661 Yelahanka 2 BHK 1120 - 1145 2.0 48.130 2  
 672 Bettahalsoor 4 Bedroom 3090 - 5002 4.0 445.000 4

def convert\_sqft\_to\_num(x):  
 tokens = x.split('-')  
 if len(tokens) == 2:  
 return (float(tokens[0])+float(tokens[1]))/2  
 try:  
 return float(x)  
 except:  
 return None

convert\_sqft\_to\_num('2100 - 2850')

2475.0

convert\_sqft\_to\_num('34.46Sq. Meter')

df4 = df3.copy()  
 df4.total\_sqft = df4.total\_sqft.apply(convert\_sqft\_to\_num)  
 df4

location size total\_sqft bath price bhk  
 0 Electronic City Phase II 2 BHK 1056.0 2.0 39.07 2  
 1 Chikka Tirupathi 4 Bedroom 2600.0 5.0 120.00 4  
 2 Uttarahalli 3 BHK 1440.0 2.0 62.00 3  
 3 Lingadheeranahalli 3 BHK 1521.0 3.0 95.00 3  
 4 Kothanur 2 BHK 1200.0 2.0 51.00 2  
 ... ... ... ... ... ... ...  
 13315 Whitefield 5 Bedroom 3453.0 4.0 231.00 5  
 13316 Richards Town 4 BHK 3600.0 5.0 400.00 4  
 13317 Raja Rajeshwari Nagar 2 BHK 1141.0 2.0 60.00 2  
 13318 Padmanabhanagar 4 BHK 4689.0 4.0 488.00 4  
 13319 Doddathoguru 1 BHK 550.0 1.0 17.00 1  
  
 [13246 rows x 6 columns]

df4 = df4[df4.total\_sqft.notnull()]  
 df4

location size total\_sqft bath price bhk  
 0 Electronic City Phase II 2 BHK 1056.0 2.0 39.07 2  
 1 Chikka Tirupathi 4 Bedroom 2600.0 5.0 120.00 4  
 2 Uttarahalli 3 BHK 1440.0 2.0 62.00 3  
 3 Lingadheeranahalli 3 BHK 1521.0 3.0 95.00 3  
 4 Kothanur 2 BHK 1200.0 2.0 51.00 2  
 ... ... ... ... ... ... ...  
 13315 Whitefield 5 Bedroom 3453.0 4.0 231.00 5  
 13316 Richards Town 4 BHK 3600.0 5.0 400.00 4  
 13317 Raja Rajeshwari Nagar 2 BHK 1141.0 2.0 60.00 2  
 13318 Padmanabhanagar 4 BHK 4689.0 4.0 488.00 4  
 13319 Doddathoguru 1 BHK 550.0 1.0 17.00 1  
  
 [13200 rows x 6 columns]

For below row, it shows total\_sqft as 2475 which is an average of the range 2100-2850

df4.loc[30]

location Yelahanka  
 size 4 BHK  
 total\_sqft 2475.0  
 bath 4.0  
 price 186.0  
 bhk 4  
 Name: 30, dtype: object

(2100 + 2850)/2

2475.0

Add new feature called price per square feet

df5 = df4.copy()  
 df5['price\_per\_sqft'] = df5['price']\*100000/df5['total\_sqft']  
 df5.head()

location size total\_sqft bath price bhk \  
 0 Electronic City Phase II 2 BHK 1056.0 2.0 39.07 2   
 1 Chikka Tirupathi 4 Bedroom 2600.0 5.0 120.00 4   
 2 Uttarahalli 3 BHK 1440.0 2.0 62.00 3   
 3 Lingadheeranahalli 3 BHK 1521.0 3.0 95.00 3   
 4 Kothanur 2 BHK 1200.0 2.0 51.00 2   
  
 price\_per\_sqft   
 0 3699.810606   
 1 4615.384615   
 2 4305.555556   
 3 6245.890861   
 4 4250.000000

df5\_stats = df5['price\_per\_sqft'].describe()  
 df5\_stats

count 1.320000e+04  
 mean 7.920759e+03  
 std 1.067272e+05  
 min 2.678298e+02  
 25% 4.267701e+03  
 50% 5.438331e+03  
 75% 7.317073e+03  
 max 1.200000e+07  
 Name: price\_per\_sqft, dtype: float64

df5.to\_csv("bhp.csv",index=False)

Examine locations which is a categorical variable. We need to apply dimensionality reduction technique here to reduce number of locations

len(df5.location.unique())

1298

df5.location = df5.location.apply(lambda x: x.strip())  
 location\_stats = df5['location'].value\_counts(ascending=False)  
 location\_stats

Whitefield 533  
 Sarjapur Road 392  
 Electronic City 304  
 Kanakpura Road 264  
 Thanisandra 235  
 ...  
 Rajanna Layout 1  
 Subramanyanagar 1  
 Lakshmipura Vidyaanyapura 1  
 Malur Hosur Road 1  
 Abshot Layout 1  
 Name: location, Length: 1287, dtype: int64

len(location\_stats[location\_stats>10])

240

len(location\_stats)

1287

len(location\_stats[location\_stats<=10])

1047

Any location having less than 10 data points should be tagged as "other" location. This way number of categories can be reduced by huge amount. Later on when we do one hot encoding, it will help us with having fewer dummy columns

location\_stats\_less\_than\_10 = location\_stats[location\_stats<=10]  
 location\_stats\_less\_than\_10

BTM 1st Stage 10  
 Gunjur Palya 10  
 Nagappa Reddy Layout 10  
 Sector 1 HSR Layout 10  
 Thyagaraja Nagar 10  
 ..  
 Rajanna Layout 1  
 Subramanyanagar 1  
 Lakshmipura Vidyaanyapura 1  
 Malur Hosur Road 1  
 Abshot Layout 1  
 Name: location, Length: 1047, dtype: int64

len(df5.location.unique())

1287

df5.location = df5.location.apply(lambda x: 'other' if x in location\_stats\_less\_than\_10 else x)  
 len(df5.location.unique())

241

df5.head(10)

location size total\_sqft bath price bhk \  
 0 Electronic City Phase II 2 BHK 1056.0 2.0 39.07 2   
 1 Chikka Tirupathi 4 Bedroom 2600.0 5.0 120.00 4   
 2 Uttarahalli 3 BHK 1440.0 2.0 62.00 3   
 3 Lingadheeranahalli 3 BHK 1521.0 3.0 95.00 3   
 4 Kothanur 2 BHK 1200.0 2.0 51.00 2   
 5 Whitefield 2 BHK 1170.0 2.0 38.00 2   
 6 Old Airport Road 4 BHK 2732.0 4.0 204.00 4   
 7 Rajaji Nagar 4 BHK 3300.0 4.0 600.00 4   
 8 Marathahalli 3 BHK 1310.0 3.0 63.25 3   
 9 other 6 Bedroom 1020.0 6.0 370.00 6   
  
 price\_per\_sqft   
 0 3699.810606   
 1 4615.384615   
 2 4305.555556   
 3 6245.890861   
 4 4250.000000   
 5 3247.863248   
 6 7467.057101   
 7 18181.818182   
 8 4828.244275   
 9 36274.509804

normally square ft per bedroom is 300 (i.e. 2 bhk apartment is minimum 600 sqft

df5[df5.total\_sqft/df5.bhk<300].head()

location size total\_sqft bath price bhk \  
 9 other 6 Bedroom 1020.0 6.0 370.0 6   
 45 HSR Layout 8 Bedroom 600.0 9.0 200.0 8   
 58 Murugeshpalya 6 Bedroom 1407.0 4.0 150.0 6   
 68 Devarachikkanahalli 8 Bedroom 1350.0 7.0 85.0 8   
 70 other 3 Bedroom 500.0 3.0 100.0 3   
  
 price\_per\_sqft   
 9 36274.509804   
 45 33333.333333   
 58 10660.980810   
 68 6296.296296   
 70 20000.000000

Check above data points. We have 6 bhk apartment with 1020 sqft. Another one is 8 bhk and total sqft is 600. These are clear data errors that can be removed safely

df5.shape

(13200, 7)

df6 = df5[~(df5.total\_sqft/df5.bhk<300)]  
 df6.shape

(12456, 7)

df6.columns

Index(['location', 'size', 'total\_sqft', 'bath', 'price', 'bhk',  
 'price\_per\_sqft'],  
 dtype='object')

plt.boxplot(df6['total\_sqft'])  
 plt.show()

Q1 = np.percentile(df6['total\_sqft'], 25.) # 25th percentile of the data of the given feature  
 Q3 = np.percentile(df6['total\_sqft'], 75.) # 75th percentile of the data of the given feature  
 IQR = Q3-Q1 #Interquartile Range  
 ll = Q1 - (1.5\*IQR)  
 ul = Q3 + (1.5\*IQR)  
 upper\_outliers = df6[df6['total\_sqft'] > ul].index.tolist()  
 lower\_outliers = df6[df6['total\_sqft'] < ll].index.tolist()  
 bad\_indices = list(set(upper\_outliers + lower\_outliers))  
 drop = True  
 if drop:  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

<ipython-input-51-c46bdd7d51e2>:11: SettingWithCopyWarning:  
 A value is trying to be set on a copy of a slice from a DataFrame  
  
 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

plt.boxplot(df6['bath'])  
 plt.show()

Q1 = np.percentile(df6['bath'], 25.) # 25th percentile of the data of the given feature  
 Q3 = np.percentile(df6['bath'], 75.) # 75th percentile of the data of the given feature  
 IQR = Q3-Q1 #Interquartile Range  
 ll = Q1 - (1.5\*IQR)  
 ul = Q3 + (1.5\*IQR)  
 upper\_outliers = df6[df6['bath'] > ul].index.tolist()  
 lower\_outliers = df6[df6['bath'] < ll].index.tolist()  
 bad\_indices = list(set(upper\_outliers + lower\_outliers))  
 drop = True  
 if drop:  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

<ipython-input-54-cdb575bb4e89>:11: SettingWithCopyWarning:  
 A value is trying to be set on a copy of a slice from a DataFrame  
  
 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

plt.boxplot(df6['price'])  
 plt.show()

Q1 = np.percentile(df6['price'], 25.) # 25th percentile of the data of the given feature  
 Q3 = np.percentile(df6['price'], 75.) # 75th percentile of the data of the given feature  
 IQR = Q3-Q1 #Interquartile Range  
 ll = Q1 - (1.5\*IQR)  
 ul = Q3 + (1.5\*IQR)  
 upper\_outliers = df6[df6['price'] > ul].index.tolist()  
 lower\_outliers = df6[df6['price'] < ll].index.tolist()  
 bad\_indices = list(set(upper\_outliers + lower\_outliers))  
 drop = True  
 if drop:  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

<ipython-input-56-e0f097c1f625>:11: SettingWithCopyWarning:  
 A value is trying to be set on a copy of a slice from a DataFrame  
  
 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

plt.boxplot(df6['bhk'])  
 plt.show()

Q1 = np.percentile(df6['bhk'], 25.) # 25th percentile of the data of the given feature  
 Q3 = np.percentile(df6['bhk'], 75.) # 75th percentile of the data of the given feature  
 IQR = Q3-Q1 #Interquartile Range  
 ll = Q1 - (1.5\*IQR)  
 ul = Q3 + (1.5\*IQR)  
 upper\_outliers = df6[df6['bhk'] > ul].index.tolist()  
 lower\_outliers = df6[df6['bhk'] < ll].index.tolist()  
 bad\_indices = list(set(upper\_outliers + lower\_outliers))  
 drop = True  
 if drop:  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

<ipython-input-58-c12c1120f543>:11: SettingWithCopyWarning:  
 A value is trying to be set on a copy of a slice from a DataFrame  
  
 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

plt.boxplot(df6['price\_per\_sqft'])  
 plt.show()

Q1 = np.percentile(df6['price\_per\_sqft'], 25.) # 25th percentile of the data of the given feature  
 Q3 = np.percentile(df6['price\_per\_sqft'], 75.) # 75th percentile of the data of the given feature  
 IQR = Q3-Q1 #Interquartile Range  
 ll = Q1 - (1.5\*IQR)  
 ul = Q3 + (1.5\*IQR)  
 upper\_outliers = df6[df6['price\_per\_sqft'] > ul].index.tolist()  
 lower\_outliers = df6[df6['price\_per\_sqft'] < ll].index.tolist()  
 bad\_indices = list(set(upper\_outliers + lower\_outliers))  
 drop = True  
 if drop:  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

<ipython-input-60-d349eb2f1f03>:11: SettingWithCopyWarning:  
 A value is trying to be set on a copy of a slice from a DataFrame  
  
 See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df6.drop(bad\_indices, inplace = True, errors = 'ignore')

df6.shape

(10090, 7)

X = df6.drop(['price'],axis='columns')  
 X.head(3)

location size total\_sqft bath bhk price\_per\_sqft  
 0 Electronic City Phase II 2 BHK 1056.0 2.0 2 3699.810606  
 2 Uttarahalli 3 BHK 1440.0 2.0 3 4305.555556  
 3 Lingadheeranahalli 3 BHK 1521.0 3.0 3 6245.890861

X.shape

(10090, 6)

y = df6.price  
 y.head(3)

0 39.07  
 2 62.00  
 3 95.00  
 Name: price, dtype: float64

len(y)

10090

from sklearn.model\_selection import train\_test\_split  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=10)

X\_train.shape

(8072, 6)

y\_train.shape

(8072,)

X\_test.shape

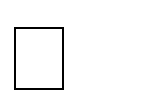
(2018, 6)

y\_test.shape

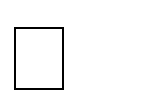
(2018,)

**EXPERIMENT NO. 11 (Group B)**

* **Aim**: Data Visualization using matplotlib
* **Problem Statement:** Analyzing Air Quality Index (AQI) Trends in a City
* **Dataset:** "City\_Air\_Quality.csv"
* **Description:** The dataset contains information about air quality measurements in a specific
* city over a period of time. It includes attributes such as date, time, pollutant levels (e.g.,
* PM2.5, PM10, CO), and the Air Quality Index (AQI) values. The goal is to use the matplotlib
* library to create visualizations that effectively represent the AQI trends and patterns for
* different pollutants in the city.
* **Tasks to Perform:**
* 1. Import the "City\_Air\_Quality.csv" dataset.
* 2. Explore the dataset to understand its structure and content.
* 3. Identify the relevant variables for visualizing AQI trends, such as date, pollutant
* levels, and AQI values.
* 4. Create line plots or time series plots to visualize the overall AQI trend over time.
* 5. Plot individual pollutant levels (e.g., PM2.5, PM10, CO) on separate line plots to
* visualize their trends over time.
* 6. Use bar plots or stacked bar plots to compare the AQI values across different dates or
* time periods.
* 7. Create box plots or violin plots to analyze the distribution of AQI values for different
* pollutant categories.
* 8. Use scatter plots or bubble charts to explore the relationship between AQI values and
* pollutant levels.
* 9. Customize the visualizations by adding labels, titles, legends, and appropriate color
* schemes.

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu

**Implementation:**

import numpy as np  
 import pandas as pd  
 import matplotlib.pyplot as plt  
 import seaborn as sns  
 from sklearn.impute import SimpleImputer  
  
 %matplotlib inline

data = pd.read\_csv("data.csv")  
 print(data.index)

RangeIndex(start=0, stop=49005, step=1)

sns.set(style="ticks", rc = {'figure.figsize':(20,15)})  
  
 # Supressing update warnings  
  
 import warnings  
 warnings.filterwarnings('ignore')

Checking the dataset

We can see that there are quite a number of NaNs in the dataset. To proceed with the EDA, we must handle these NaNs by either removing them or filling them. I will be doing both.

# checking the original dataset  
 print(data.isnull().sum())  
 print(data.shape)  
 data.info()

stn\_code 15764  
 sampling\_date 0  
 state 0  
 location 0  
 agency 16355  
 type 994  
 so2 1312  
 no2 858  
 rspm 2696  
 spm 28659  
 location\_monitoring\_station 2537  
 pm2\_5 49005  
 date 1  
 dtype: int64  
 (49005, 13)  
 <class 'pandas.core.frame.DataFrame'>  
 RangeIndex: 49005 entries, 0 to 49004  
 Data columns (total 13 columns):  
 # Column Non-Null Count Dtype   
 --- ------ -------------- -----   
 0 stn\_code 33241 non-null float64  
 1 sampling\_date 49005 non-null object  
 2 state 49005 non-null object  
 3 location 49005 non-null object  
 4 agency 32650 non-null object  
 5 type 48011 non-null object  
 6 so2 47693 non-null float64  
 7 no2 48147 non-null float64  
 8 rspm 46309 non-null float64  
 9 spm 20346 non-null float64  
 10 location\_monitoring\_station 46468 non-null object  
 11 pm2\_5 0 non-null float64  
 12 date 49004 non-null object  
 dtypes: float64(6), object(7)  
 memory usage: 4.9+ MB

# Cleaning the dataset

Removing NaNs Looking at the dataset head, we can conclude that the following columns:

1. stn\_code

1. agency

2. sampling\_date

3. location\_monitoring\_agency

do not add much to the dataset in terms of information that can't already be extracted from other columns. Therefore, we drop these columns.

Since date also has missing values, we will drop the rows containing these values as they're of little use as well.

Cleaning values Since the geographical nomenclature has changed over time, we change it here as well to correspond to more accurate insights.

The type column

Currently, the type column has several names for the same type and therefore, it is better to clean it up and make it more uniform.

# Cleaning up the data  
  
  
 # cleaning up name changes  
  
 data.state = data.state.replace({'Uttaranchal':'Uttarakhand'})  
 data.state[data.location == "Jamshedpur"] = data.state[data.location == 'Jamshedpur'].replace({"Bihar":"Jharkhand"})  
  
 #changing types to uniform format  
  
 types = {  
 "Residential": "R",  
 "Residential and others": "RO",  
 "Residential, Rural and other Areas": "RRO",  
 "Industrial Area": "I",  
 "Industrial Areas": "I",  
 "Industrial": "I",  
 "Sensitive Area": "S",  
 "Sensitive Areas": "S",  
 "Sensitive": "S",  
 np.nan: "RRO"  
 }  
 data.type = data.type.replace(types)

data.head()

state location type so2 no2 rspm spm pm2\_5 date  
 0 Andhra Pradesh Hyderabad RRO 4.8 17.4 NaN NaN NaN 1990-02-01  
 1 Andhra Pradesh Hyderabad I 3.1 7.0 NaN NaN NaN 1990-02-01  
 2 Andhra Pradesh Hyderabad RRO 6.2 28.5 NaN NaN NaN 1990-02-01  
 3 Andhra Pradesh Hyderabad RRO 6.3 14.7 NaN NaN NaN 1990-03-01  
 4 Andhra Pradesh Hyderabad I 4.7 7.5 NaN NaN NaN 1990-03-01

# defining columns of importance, which shall be used reguarly  
 VALUE\_COLS = ['so2', 'no2', 'rspm', 'spm', 'pm2\_5']

Filling NaNs Since our pollutants column contain a lot of NaNs, we must fill them to have consistent data. If we drop the rows containing NaNs, we will be left with nothing.

I use the SimpleImputer from sklearn.imputer (v0.20.2) to fill the missing values in every column with the mean.

# invoking SimpleImputer to fill missing values  
 imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  
 data[VALUE\_COLS] = imputer.fit\_transform(data[VALUE\_COLS])

---------------------------------------------------------------------------  
 ValueError Traceback (most recent call last)  
 <ipython-input-16-7a53965e699d> in <cell line: 3>()  
 1 # invoking SimpleImputer to fill missing values  
 2 imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  
 ----> 3 data[VALUE\_COLS] = imputer.fit\_transform(data[VALUE\_COLS])  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in \_\_setitem\_\_(self, key, value)  
 3966 self.\_setitem\_frame(key, value)  
 3967 elif isinstance(key, (Series, np.ndarray, list, Index)):  
 -> 3968 self.\_setitem\_array(key, value)  
 3969 elif isinstance(value, DataFrame):  
 3970 self.\_set\_item\_frame\_value(key, value)  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in \_setitem\_array(self, key, value)  
 4017  
 4018 elif isinstance(value, np.ndarray) and value.ndim == 2:  
 -> 4019 self.\_iset\_not\_inplace(key, value)  
 4020  
 4021 elif np.ndim(value) > 1:  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in \_iset\_not\_inplace(self, key, value)  
 4044 if self.columns.is\_unique:  
 4045 if np.shape(value)[-1] != len(key):  
 -> 4046 raise ValueError("Columns must be same length as key")  
 4047  
 4048 for i, col in enumerate(key):  
  
 ValueError: Columns must be same length as key

# checking to see if the dataset has any null values left over and the format  
 print(data.isnull().sum())  
 data.tail()

state 0  
 location 0  
 type 0  
 so2 1312  
 no2 858  
 rspm 2696  
 spm 28659  
 pm2\_5 49005  
 date 1  
 dtype: int64

state location type so2 no2 rspm spm pm2\_5 \  
 49000 Chandigarh Chandigarh RO 6.0 15.0 47.0 125.0 NaN   
 49001 Chandigarh Chandigarh RO NaN 12.0 54.0 161.0 NaN   
 49002 Chandigarh Chandigarh RO NaN 10.0 116.0 196.0 NaN   
 49003 Chandigarh Chandigarh RO NaN 9.0 38.0 154.0 NaN   
 49004 Chandigarh Chandigarh RO 10.0 27.0 43.0 152.0 NaN   
  
 date   
 49000 2005-03-23   
 49001 2005-03-25   
 49002 2005-03-28   
 49003 2005-03-30   
 49004 NaN

# Plotting pollutant levels as yearly averages for states

# defining a function that plots SO2, NO2, RSPM and SPM yearly average levels for a given state  
 # since data is available monthly, it was resampled to a year and averaged to obtain yearly averages  
 # years for which no data was collected has not been imputed  
 def plot\_for\_state(state):  
 fig, ax = plt.subplots(2,2, figsize=(20,12))  
 fig.suptitle(state, size=20)  
 state = aqi[aqi.state == state]  
 state = state.reset\_index().set\_index('date')[VALUE\_COLS].resample('Y').mean()  
 state.so2.plot(legend=True, ax=ax[0][0], title="so2")  
 ax[0][0].set\_ylabel("so2 (µg/m3)")  
 ax[0][0].set\_xlabel("Year")  
  
 state.no2.plot(legend=True, ax=ax[0][1], title="no2")  
 ax[0][1].set\_ylabel("no2 (µg/m3)")  
 ax[0][1].set\_xlabel("Year")  
  
 state.rspm.plot(legend=True, ax=ax[1][0], title="rspm")  
 ax[1][0].set\_ylabel("RSPM (PM10 µg/m3)")  
 ax[1][0].set\_xlabel("Year")  
  
 state.spm.plot(legend=True, ax=ax[1][1], title="spm")  
 ax[1][1].set\_ylabel("SPM (PM10 µg/m3)")  
 ax[1][1].set\_xlabel("Year")

plot\_for\_state("Uttar Pradesh")

Plotting Uttar Pradesh, we see that SO2 levels have fallen in the state while NO2 levels have risen. Information about RSPM and SPM can't be concluded since a lot of data is missing.

# Plotting highest and lowest ranking states

# defining a function to find and plot the top 10 and bottom 10 states for a given indicator (defaults to SO2)  
 def top\_and\_bottom\_10\_states(indicator="so2"):  
 fig, ax = plt.subplots(2,1, figsize=(20, 12))  
  
 ind = data[[indicator, 'state']].groupby('state', as\_index=False).median().sort\_values(by=indicator,ascending=False)  
 top10 = sns.barplot(x='state', y=indicator, data=ind[:10], ax=ax[0], color='red')  
 top10.set\_title("Top 10 states by {} (1991-2016)".format(indicator))  
 top10.set\_ylabel("so2 (µg/m3)")  
 top10.set\_xlabel("State")  
  
 bottom10 = sns.barplot(x='state', y=indicator, data=ind[-10:], ax=ax[1], color='green')  
 bottom10.set\_title("Bottom 10 states by {} (1991-2016)".format(indicator))  
 bottom10.set\_ylabel("so2 (µg/m3)")  
 bottom10.set\_xlabel("State")

top\_and\_bottom\_10\_states("so2")  
 top\_and\_bottom\_10\_states("no2")

Plotting for SO2, we can see that the top state is Uttarakhand, while the bottom state is Meghalaya.

Plotting for NO2, we can see that the top state is West Bengal, while the bottom state is Mizoram.

# Plotting the highest ever recorded levels

# defining a function to find the highest ever recorded levels for a given indicator (defaults to SO2) by state  
 # sidenote: mostly outliers  
 def highest\_levels\_recorded(indicator="so2"):  
 plt.figure(figsize=(20,10))  
 ind = data[[indicator, 'location', 'state', 'date']].groupby('state', as\_index=False).max()  
 highest = sns.barplot(x='state', y=indicator, data=ind)  
 highest.set\_title("Highest ever {} levels recorded by state".format(indicator))  
 plt.xticks(rotation=90)

highest\_levels\_recorded("no2")  
 highest\_levels\_recorded("rspm")

Plotting for NO2, we can see that Rajasthan recorded the highest ever NO2 level. Plotting for RSPM, we can see that Uttar Pradesh recorded the highest ever RSPM level.

# Plotting yearly trends

# defining a function to plot the yearly trend values for a given indicator (defaults to SO2) and state (defaults to overall)  
 def yearly\_trend(state="", indicator="so2", ):  
 plt.figure(figsize=(20,12))  
 data['year'] = data.date.dt.year  
 if state is "":  
 year\_wise = data[[indicator, 'year', 'state']].groupby('year', as\_index=False).median()  
 trend = sns.pointplot(x='year', y=indicator, data=year\_wise)  
 trend.set\_title('Yearly trend of {}'.format(indicator))  
 else:  
 year\_wise = data[[indicator, 'year', 'state']].groupby(['state','year']).median().loc[state].reset\_index()  
 trend = sns.pointplot(x='year', y=indicator, data=year\_wise)  
 trend.set\_title('Yearly trend of {} for {}'.format(indicator, state))

yearly\_trend()  
 yearly\_trend("Bihar", "no2")

---------------------------------------------------------------------------  
 AttributeError Traceback (most recent call last)  
 <ipython-input-42-e79267482a54> in <cell line: 1>()  
 ----> 1 yearly\_trend()  
 2 yearly\_trend("Bihar", "no2")  
  
 <ipython-input-30-93f123e178ba> in yearly\_trend(state, indicator)  
 2 def yearly\_trend(state="", indicator="so2", ):  
 3 plt.figure(figsize=(20,12))  
 ----> 4 data['year'] = data.date.dt.year  
 5 if state is "":  
 6 year\_wise = data[[indicator, 'year', 'state']].groupby('year', as\_index=False).median()  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in \_\_getattr\_\_(self, name)  
 5900 ):  
 5901 return self[name]  
 -> 5902 return object.\_\_getattribute\_\_(self, name)  
 5903  
 5904 def \_\_setattr\_\_(self, name: str, value) -> None:  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/accessor.py in \_\_get\_\_(self, obj, cls)  
 180 # we're accessing the attribute of the class, i.e., Dataset.geo  
 181 return self.\_accessor  
 --> 182 accessor\_obj = self.\_accessor(obj)  
 183 # Replace the property with the accessor object. Inspired by:  
 184 # https://www.pydanny.com/cached-property.html  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/indexes/accessors.py in \_\_new\_\_(cls, data)  
 510 return PeriodProperties(data, orig)  
 511  
 --> 512 raise AttributeError("Can only use .dt accessor with datetimelike values")  
  
 AttributeError: Can only use .dt accessor with datetimelike values

<Figure size 2000x1200 with 0 Axes>

Plotting for SO2, we can see the yearly trend for sulphur dioxide levels in the country. Plotting for NO2 in West Bengal, we can see the yearly trend.

# Plotting a heatmap for a particular indicator

# defining a function to plot a heatmap for yearly median average for a given indicator (defaults to SO2)  
 def indicator\_by\_state\_and\_year(indicator="so2"):  
 plt.figure(figsize=(20, 20))  
 hmap = sns.heatmap(  
 data=data.pivot\_table(values=indicator, index='state', columns='year', aggfunc='median', margins=True),  
 annot=True, linewidths=.5, cbar=True, square=True, cmap='inferno', cbar\_kws={'label': "Annual Average"})  
  
 hmap.set\_title("{} by state and year".format(indicator))

indicator\_by\_state\_and\_year('no2')

---------------------------------------------------------------------------  
 KeyError Traceback (most recent call last)  
 <ipython-input-35-39c9f3640fe4> in <cell line: 1>()  
 ----> 1 indicator\_by\_state\_and\_year('no2')  
  
 <ipython-input-34-3c4f9130ffd5> in indicator\_by\_state\_and\_year(indicator)  
 3 plt.figure(figsize=(20, 20))  
 4 hmap = sns.heatmap(  
 ----> 5 data=data.pivot\_table(values=indicator, index='state', columns='year', aggfunc='median', margins=True),  
 6 annot=True, linewidths=.5, cbar=True, square=True, cmap='inferno', cbar\_kws={'label': "Annual Average"})  
 7  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in pivot\_table(self, values, index, columns, aggfunc, fill\_value, margins, dropna, margins\_name, observed, sort)  
 8729 from pandas.core.reshape.pivot import pivot\_table  
 8730  
 -> 8731 return pivot\_table(  
 8732 self,  
 8733 values=values,  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/reshape/pivot.py in pivot\_table(data, values, index, columns, aggfunc, fill\_value, margins, dropna, margins\_name, observed, sort)  
 95 return table.\_\_finalize\_\_(data, method="pivot\_table")  
 96  
 ---> 97 table = \_\_internal\_pivot\_table(  
 98 data,  
 99 values,  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/reshape/pivot.py in \_\_internal\_pivot\_table(data, values, index, columns, aggfunc, fill\_value, margins, dropna, margins\_name, observed, sort)  
 164 values = list(values)  
 165  
 --> 166 grouped = data.groupby(keys, observed=observed, sort=sort)  
 167 msg = (  
 168 "pivot\_table dropped a column because it failed to aggregate. This behavior "  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in groupby(self, by, axis, level, as\_index, sort, group\_keys, squeeze, observed, dropna)  
 8400 axis = self.\_get\_axis\_number(axis)  
 8401  
 -> 8402 return DataFrameGroupBy(  
 8403 obj=self,  
 8404 keys=by,  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/groupby/groupby.py in \_\_init\_\_(self, obj, keys, axis, level, grouper, exclusions, selection, as\_index, sort, group\_keys, squeeze, observed, mutated, dropna)  
 963 from pandas.core.groupby.grouper import get\_grouper  
 964  
 --> 965 grouper, exclusions, obj = get\_grouper(  
 966 obj,  
 967 keys,  
  
 /usr/local/lib/python3.10/dist-packages/pandas/core/groupby/grouper.py in get\_grouper(obj, key, axis, level, sort, observed, mutated, validate, dropna)  
 886 in\_axis, level, gpr = False, gpr, None  
 887 else:  
 --> 888 raise KeyError(gpr)  
 889 elif isinstance(gpr, Grouper) and gpr.key is not None:  
 890 # Add key to exclusions  
  
 KeyError: 'year'

<Figure size 2000x2000 with 0 Axes>

# Plotting pollutant average by type

# defining a function to plot pollutant averages by type for a given indicator  
 def type\_avg(indicator=""):  
 type\_avg = data[VALUE\_COLS + ['type', 'date']].groupby("type").mean()  
 if indicator is not "":  
 t = type\_avg[indicator].plot(kind='bar')  
 plt.xticks(rotation = 0)  
 plt.title("Pollutant average by type for {}".format(indicator))  
 else:  
 t = type\_avg.plot(kind='bar')  
 plt.xticks(rotation = 0)  
 plt.title("Pollutant average by type")

type\_avg('so2')

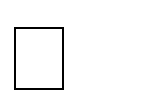
# Plotting pollutant averages by locations/state

# defining a function to plot pollutant averages for a given indicator (defaults to SO2) by locations in a given state  
 def location\_avgs(state, indicator="so2"):  
 locs = data[VALUE\_COLS + ['state', 'location', 'date']].groupby(['state', 'location']).mean()  
 state\_avgs = locs.loc[state].reset\_index()  
 sns.barplot(x='location', y=indicator, data=state\_avgs)  
 plt.title("Location-wise average for {} in {}".format(indicator, state))  
 plt.xticks(rotation = 90)

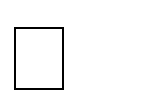
location\_avgs("Bihar", "no2")

**EXPERIMENT NO. 12 (Group B)**

* **Aim**: Data Aggregation
* **Problem Statement:** Analyzing Sales Performance by Region in a Retail Company
* **Dataset:** "Retail\_Sales\_Data.csv"
* **Description:** The dataset contains information about sales transactions in a retail company. It includes attributes such as transaction date, product category, quantity sold, and sales
* amount. The goal is to perform data aggregation to analyze the sales performance by region and identify the top-performing regions.
* **Tasks to Perform:**
* 1. Import the "Retail\_Sales\_Data.csv" dataset.
* 2. Explore the dataset to understand its structure and content.
* 3. Identify the relevant variables for aggregating sales data, such as region, sales
* amount, and product category.
* 4. Group the sales data by region and calculate the total sales amount for each region.
* 5. Create bar plots or pie charts to visualize the sales distribution by region.
* 6. Identify the top-performing regions based on the highest sales amount.
* 7. Group the sales data by region and product category to calculate the total sales
* amount for each combination.
* 8. Create stacked bar plots or grouped bar plots to compare the sales amounts across
* different regions and product categories.

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu

**Implementation:**

import pandas as pd  
 import matplotlib.pyplot as plt

Data Aggregation is important for deriving granular insights about individual customers and for better understanding their perception and expectations regarding the product.

Regardless of the size and type, every business organization needs valuable data and insights to combat the day-to-day challenges of the competitive market. If a business wants to thrive in the market, then it must understand its target audience and customer preferences, and in this, big data plays a vital role.

What is Data Aggregation?

# *About Dataset*

dataset contains shopping information from 10 different shopping malls between 2021 and 2023. We have gathered data from various age groups and genders to provide a comprehensive view of shopping habits in Istanbul. The dataset includes essential information such as invoice numbers, customer IDs, age, gender, payment methods, product categories, quantity, price, order dates, and shopping mall locations.

Attribute Information:

invoice\_no: Invoice number. Nominal. A combination of the letter 'I' and a 6-digit integer uniquely assigned to each operation.

customer\_id: Customer number. Nominal. A combination of the letter 'C' and a 6-digit integer uniquely assigned to each operation.

gender: String variable of the customer's gender.

age: Positive Integer variable of the customers age.

category: String variable of the category of the purchased product.

quantity: The quantities of each product (item) per transaction. Numeric.

price: Unit price. Numeric. Product price per unit in Turkish Liras (TL).

payment\_method: String variable of the payment method (cash, credit card or debit card) used for the transaction.

invoice\_date: Invoice date. The day when a transaction was generated.

shopping\_mall: String variable of the name of the shopping mall where the transaction was made.

# dataset source: https://www.kaggle.com/datasets/mehmettahiraslan/customer-shopping-dataset  
  
 #df = pd.read\_csv("/content/customer\_shopping\_data.csv")  
 df= pd.read\_csv("/content/customer\_shopping\_data.csv")  
 df.head()

invoice\_no customer\_id gender age category quantity price \  
 0 I138884 C241288 Female 28 Clothing 5.0 1500.40   
 1 I317333 C111565 Male 21 Shoes 3.0 1800.51   
 2 I127801 C266599 Male 20 Clothing 1.0 300.08   
 3 I173702 C988172 Female 66 Shoes 5.0 3000.85   
 4 I337046 C189076 Female 53 Books 4.0 60.60   
  
 payment\_method invoice\_date shopping\_mall   
 0 Credit Card 5/8/2022 Kanyon   
 1 Debit Card 12/12/2021 Forum Istanbul   
 2 Cash 9/11/2021 Metrocity   
 3 Credit Card 16/05/2021 Metropol AVM   
 4 Cash 24/10/2021 Kanyon

# To check the count of records grouped by region/branch of the mall  
  
 df.groupby("shopping\_mall").count()

invoice\_no customer\_id gender age category quantity \  
 shopping\_mall   
 Cevahir AVM 1349 1349 1349 1349 1349 1349   
 Emaar Square Mall 1341 1341 1341 1341 1341 1341   
 Forum Istanbul 1343 1343 1343 1343 1343 1343   
 Istinye Park 2709 2709 2709 2709 2709 2709   
 Kanyon 5481 5481 5481 5481 5481 5481   
 Mall of Istanbul 5588 5588 5588 5588 5588 5588   
 Metrocity 4193 4193 4193 4193 4193 4193   
 Metropol AVM 2856 2856 2856 2856 2856 2856   
 Viaport Outlet 1389 1389 1389 1389 1389 1389   
 Zorlu Center 1392 1392 1392 1392 1392 1392   
  
 price payment\_method invoice\_date   
 shopping\_mall   
 Cevahir AVM 1349 1349 1349   
 Emaar Square Mall 1341 1341 1341   
 Forum Istanbul 1343 1343 1343   
 Istinye Park 2709 2709 2709   
 Kanyon 5481 5481 5481   
 Mall of Istanbul 5588 5588 5588   
 Metrocity 4193 4193 4193   
 Metropol AVM 2856 2856 2856   
 Viaport Outlet 1389 1389 1389   
 Zorlu Center 1392 1392 1392

# To check the count of records grouped by the product categories  
  
 df.groupby("category").count()

invoice\_no customer\_id gender age quantity price \  
 category   
 Books 1397 1397 1397 1397 1397 1397   
 Clothin 1 1 1 1 0 0   
 Clothing 9433 9433 9433 9433 9433 9433   
 Cosmetics 4224 4224 4224 4224 4224 4224   
 Food & Beverage 4158 4158 4158 4158 4158 4158   
 Shoes 2773 2773 2773 2773 2773 2773   
 Souvenir 1402 1402 1402 1402 1402 1402   
 Technology 1435 1435 1435 1435 1435 1435   
 Toys 2819 2819 2819 2819 2819 2819   
  
 payment\_method invoice\_date shopping\_mall   
 category   
 Books 1397 1397 1397   
 Clothin 0 0 0   
 Clothing 9433 9433 9433   
 Cosmetics 4224 4224 4224   
 Food & Beverage 4158 4158 4158   
 Shoes 2773 2773 2773   
 Souvenir 1402 1402 1402   
 Technology 1435 1435 1435   
 Toys 2819 2819 2819

# total sales for each mall branch  
  
 branch\_sales = df.groupby("shopping\_mall").sum()

<ipython-input-13-64840580634c>:3: FutureWarning: The default value of numeric\_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric\_only will default to False. Either specify numeric\_only or select only columns which should be valid for the function.  
 branch\_sales = df.groupby("shopping\_mall").sum()

# total sales for each category of product  
  
 category\_sales = df.groupby("category").sum()

<ipython-input-14-732f2a6af039>:3: FutureWarning: The default value of numeric\_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric\_only will default to False. Either specify numeric\_only or select only columns which should be valid for the function.  
 category\_sales = df.groupby("category").sum()

In the above two cells, the sum method will return sums for all numeric values. For some attributes such as age, this sum is not relevant.

#to get the top performing branches  
  
 branch\_sales.sort\_values(by = "price", ascending = False)

age quantity price  
 shopping\_mall   
 Mall of Istanbul 243751 16680.0 3874873.68  
 Kanyon 237767 16464.0 3774006.38  
 Metrocity 183003 12585.0 2799049.70  
 Metropol AVM 123899 8530.0 1886384.39  
 Istinye Park 118686 8202.0 1874608.87  
 Viaport Outlet 59666 4107.0 989716.52  
 Zorlu Center 60844 4181.0 983379.89  
 Emaar Square Mall 58286 4008.0 927215.95  
 Cevahir AVM 57069 4059.0 913555.36  
 Forum Istanbul 58716 4063.0 895712.68

# to get the top selling categories  
  
 category\_sales.sort\_values(by = "price", ascending = False)

age quantity price  
 category   
 Clothing 1497054 103558 31075684.64  
 Shoes 436027 30217 18135336.89  
 Technology 216669 15021 15772050.00  
 Cosmetics 657937 45465 1848606.90  
 Toys 437032 30321 1086704.64  
 Food & Beverage 640605 44277 231568.71  
 Books 216882 14982 226977.30  
 Souvenir 216922 14871 174436.83

# to get total sales for each combination of branch and product\_category  
  
 combined\_branch\_category\_sales = df.groupby(["shopping\_mall", "category"]).sum()

<ipython-input-16-994273aad95b>:3: FutureWarning: The default value of numeric\_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric\_only will default to False. Either specify numeric\_only or select only columns which should be valid for the function.  
 combined\_branch\_category\_sales = df.groupby(["shopping\_mall", "category"]).sum()

combined\_branch\_category\_sales

# pie chart for sales by branch  
  
 plt.pie(branch\_sales["price"], labels = branch\_sales.index)  
 plt.show()

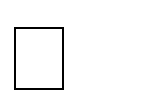
# pie chart for sales by product category  
  
 plt.pie(category\_sales["price"], labels = category\_sales.index)  
 plt.show()

combined\_pivot = df.pivot\_table(index="shopping\_mall", columns="category", values="price", aggfunc="sum")

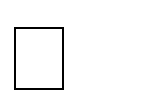
# grouped bar chart for sales of different categories at different branches  
  
 combined\_pivot.plot(kind="bar", figsize=(10, 6))  
 plt.show()

**EXPERIMENT NO. 13 (Group B)**

* **Aim**: Data Loading, Storage and File Formats
* **Problem Statement:** Analyzing Sales Data from Multiple File Formats
* **Dataset:** Sales data in multiple file formats (e.g., CSV, Excel, JSON)
* **Description:** The goal is to load and analyze sales data from different file formats, including CSV, Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset.
* **Tasks to Perform:**
* Obtain sales data files in various formats, such as CSV, Excel, and JSON.
* 1. Load the sales data from each file format into the appropriate data structures or dataframes.
* 2. Explore the structure and content of the loaded data, identifying any inconsistencies, missing values, or data quality issues.
* 3. Perform data cleaning operations, such as handling missing values, removing duplicates, or correcting inconsistencies.
* 4. Convert the data into a unified format, such as a common dataframe or data structure, to enable seamless analysis. Faculty of Science and Technology Savitribai Phule Pune University Syllabus for Fourth Year of Artificial Intelligence and Data Science (2020 Course) 46/126
* 5. Perform data transformation tasks, such as merging multiple datasets, splitting columns, or deriving new variables.
* 6. Analyze the sales data by performing descriptive statistics, aggregating data by specific variables, or calculating metrics such as total sales, average order value, or product category distribution.
* 7. Create visualizations, such as bar plots, pie charts, or box plots, to represent the sales data and gain insights into sales trends, customer behavior, or product performance.

**Hardware Requirement:**

* 6 GB free disk space.
* 2 GB RAM.
* 2 GB of RAM, plus additional RAM for virtual machines.
* 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
* Virtualization is available with the KVM hypervisor
* Intel 64 and AMD64 architectures

**Software Requirement:**

Jypiter Nootbook/Ubuntu

Implementation:

Analyzing sales data from multiple file formats can be a common task in business and data analysis. Sales data can come in various formats, including spreadsheets (Excel, CSV), databases, and even text files. Here's a step-by-step guide on how to analyze sales data from multiple file formats:

* Gather the Data: Collect all the sales data files you need from various sources and formats.
* File Format Identification: Determine the formats of the data files you have. Common formats include Excel (.xlsx), CSV (.csv), JSON (.json), SQL databases, and text files (.txt).
* Data Preparation:
  + If the data is not in a format that you can work with directly, you might need to clean and preprocess it. This includes removing duplicates, handling missing values, and standardizing date formats.
  + It's often beneficial to have a consistent data structure across all files. This means having the same columns and data types in each dataset.
* Choose Analysis Tools: Depending on your data format and the analysis you want to perform, choose appropriate analysis tools. For example:
  + For Excel files, you can use Microsoft Excel or Google Sheets.
  + For CSV files or databases, you might use Python or R with libraries like Pandas or SQL for database querying.
  + For text files or other custom formats, you may need to write custom parsing scripts.
* Load Data: Import the data into your chosen analysis tool. This often involves reading the data from the file and loading it into data structures like dataframes or database tables.
* Data Consolidation: If you have data in multiple formats or from different sources, consolidate it into a single dataset for analysis. This might involve merging or joining datasets using common identifiers like customer IDs or product codes.
* Exploratory Data Analysis (EDA):
  + Start with EDA to get a sense of your data. This includes summary statistics, data visualization (e.g., histograms, scatter plots), and identifying patterns and outliers.
  + EDA helps you understand your data and formulate hypotheses.
* Data Analysis: Perform the specific analysis you need, which might include:
  + Calculating sales totals, averages, or trends.
  + Segmenting the data to analyze different product lines, customer segments, or regions.
  + Forecasting future sales using statistical or machine learning models.
* Visualization: Present your findings using charts, graphs, and dashboards. Tools like Tableau, Power BI, or Python libraries like Matplotlib and Seaborn can help with this.
* Reporting: Create a report that summarizes your analysis, findings, and insights. Include actionable recommendations if applicable.
* Automation: If you regularly receive sales data in various formats, consider automating this process using scripts or tools to save time and ensure consistency in your analysis.
* Documentation: Document your analysis process, data sources, cleaning steps, and any assumptions made. This documentation will be valuable for future reference and for sharing your work with others.

Remember that the specific tools and steps you need to take will depend on the nature of your sales data, the file formats involved, and the objectives of your analysis.

import pandas as pd import tensorflow as tf import numpy as np

#Creating dataframes dfcsv = pd.read\_csv('sales\_data\_sample.csv', encoding = "latin") dfxlsx = pd.read\_excel('sales\_data\_sample.xlsx')

# dfjson = pd.read\_json('/content/customers.json', encoding = "latin")

#Printing first 5 rows of dataset dfcsv.head(5)

ORDERNUMBER QUANTITYORDERED PRICEEACH ORDERLINENUMBER SALES ORDERDATE STATUS QTR\_ID MONTH\_ID YEAR\_ID ... ADDRESSLINE1

2/24/2003 897 Long 0 10107 30 95.70 2 2871.00 Shipped 1 2 2003 ...

0:00 Airport Avenue

5/7/2003 59 rue de 1 10121 34 81.35 5 2765.90 Shipped 2 5 2003 ...

0:00 l'Abbaye

27 rue du 7/1/2003

2 10134 41 94.74 2 3884.34 Shipped 3 7 2003 ... Colonel Pierre

0:00

Avia

8/25/2003 78934 Hillside 3 10145 45 83.26 6 3746.70 Shipped 3 8 2003 ...

0:00 Dr.

10/10/2003 7734 Strong 4 10159 49 100.00 14 5205.27 Shipped 4 10 2003 ...

0:00 St.

5 rows × 25 columns

#Shape command to know the rows and columns in dataset dfcsv.shape

(2823, 25)

#prints information about the DataFrame dfcsv.info

<bound method DataFrame.info of ORDERNUMBER QUANTITYORDERED PRICEEACH ORDERLINENUMBER SALES \

0 10107 30 95.70 2 2871.00

1 10121 34 81.35 5 2765.90

2 10134 41 94.74 2 3884.34

3 10145 45 83.26 6 3746.70

4 10159 49 100.00 14 5205.27 ... ... ... ... ... ...

2818 10350 20 100.00 15 2244.40

2819 10373 29 100.00 1 3978.51

2820 10386 43 100.00 4 5417.57

2821 10397 34 62.24 1 2116.16

2822 10414 47 65.52 9 3079.44

ORDERDATE STATUS QTR\_ID MONTH\_ID YEAR\_ID ... \ 0 2/24/2003 0:00 Shipped 1 2 2003 ...

1 5/7/2003 0:00 Shipped 2 5 2003 ...

2 7/1/2003 0:00 Shipped 3 7 2003 ...

3 8/25/2003 0:00 Shipped 3 8 2003 ...

4 10/10/2003 0:00 Shipped 4 10 2003 ...

... ... ... ... ... ... ...

2818 12/2/2004 0:00 Shipped 4 12 2004 ...

2819 1/31/2005 0:00 Shipped 1 1 2005 ...

2820 3/1/2005 0:00 Resolved 1 3 2005 ...

2821 3/28/2005 0:00 Shipped 1 3 2005 ...

2822 5/6/2005 0:00 On Hold 2 5 2005 ...

ADDRESSLINE1 ADDRESSLINE2 CITY STATE \

0 897 Long Airport Avenue NaN NYC NY

1 59 rue de l'Abbaye NaN Reims NaN

2 27 rue du Colonel Pierre Avia NaN Paris NaN

3 78934 Hillside Dr. NaN Pasadena CA

4 7734 Strong St. NaN San Francisco CA ... ... ... ... ...

2818 C/ Moralzarzal, 86 NaN Madrid NaN

2819 Torikatu 38 NaN Oulu NaN

2820 C/ Moralzarzal, 86 NaN Madrid NaN

2821 1 rue Alsace-Lorraine NaN Toulouse NaN

2822 8616 Spinnaker Dr. NaN Boston MA

POSTALCODE COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE

0 10022 USA NaN Yu Kwai Small

1 51100 France EMEA Henriot Paul Small

2 75508 France EMEA Da Cunha Daniel Medium

3 90003 USA NaN Young Julie Medium

4 NaN USA NaN Brown Julie Medium ... ... ... ... ... ... ...

2818 28034 Spain EMEA Freyre Diego Small

2819 90110 Finland EMEA Koskitalo Pirkko Medium

2820 28034 Spain EMEA Freyre Diego Medium

2821 31000 France EMEA Roulet Annette Small

2822 51003 USA NaN Yoshido Juri Medium

[2823 rows x 25 columns]>

#Checks for NA values in columns dfcsv.isna().sum()

ORDERNUMBER 0

QUANTITYORDERED 0

PRICEEACH 0

ORDERLINENUMBER 0

SALES 0

ORDERDATE 0

STATUS 0

QTR\_ID 0

MONTH\_ID 0

YEAR\_ID 0

PRODUCTLINE 0

MSRP 0

PRODUCTCODE 0

CUSTOMERNAME 0

PHONE 0

ADDRESSLINE1 0

ADDRESSLINE2 2521

CITY 0

STATE 1486

POSTALCODE 76

COUNTRY 0

TERRITORY 1074

CONTACTLASTNAME 0

CONTACTFIRSTNAME 0 DEALSIZE 0 dtype: int64

#for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame dfcsv.describe

<bound method NDFrame.describe of ORDERNUMBER QUANTITYORDERED PRICEEACH ORDERLINENUMBER SALES \

0 10107 30 95.70 2 2871.00

1 10121 34 81.35 5 2765.90

2 10134 41 94.74 2 3884.34

3 10145 45 83.26 6 3746.70

4 10159 49 100.00 14 5205.27 ... ... ... ... ... ...

2818 10350 20 100.00 15 2244.40

2819 10373 29 100.00 1 3978.51

2820 10386 43 100.00 4 5417.57

2821 10397 34 62.24 1 2116.16

2822 10414 47 65.52 9 3079.44

ORDERDATE STATUS QTR\_ID MONTH\_ID YEAR\_ID ... \ 0 2/24/2003 0:00 Shipped 1 2 2003 ...

1 5/7/2003 0:00 Shipped 2 5 2003 ...

2 7/1/2003 0:00 Shipped 3 7 2003 ...

3 8/25/2003 0:00 Shipped 3 8 2003 ...

4 10/10/2003 0:00 Shipped 4 10 2003 ...

... ... ... ... ... ... ...

2818 12/2/2004 0:00 Shipped 4 12 2004 ...

2819 1/31/2005 0:00 Shipped 1 1 2005 ...

2820 3/1/2005 0:00 Resolved 1 3 2005 ...

2821 3/28/2005 0:00 Shipped 1 3 2005 ...

2822 5/6/2005 0:00 On Hold 2 5 2005 ...

ADDRESSLINE1 ADDRESSLINE2 CITY STATE \

0 897 Long Airport Avenue NaN NYC NY

1 59 rue de l'Abbaye NaN Reims NaN

2 27 rue du Colonel Pierre Avia NaN Paris NaN

3 78934 Hillside Dr. NaN Pasadena CA

4 7734 Strong St. NaN San Francisco CA

... ... ... ... ...

2818 C/ Moralzarzal, 86 NaN Madrid NaN

2819 Torikatu 38 NaN Oulu NaN

2820 C/ Moralzarzal, 86 NaN Madrid NaN

2821 1 rue Alsace-Lorraine NaN Toulouse NaN

2822 8616 Spinnaker Dr. NaN Boston MA

POSTALCODE COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE

0 10022 USA NaN Yu Kwai Small

1 51100 France EMEA Henriot Paul Small

2 75508 France EMEA Da Cunha Daniel Medium

3 90003 USA NaN Young Julie Medium

4 NaN USA NaN Brown Julie Medium ... ... ... ... ... ... ...

2818 28034 Spain EMEA Freyre Diego Small

2819 90110 Finland EMEA Koskitalo Pirkko Medium

2820 28034 Spain EMEA Freyre Diego Medium

2821 31000 France EMEA Roulet Annette Small

2822 51003 USA NaN Yoshido Juri Medium

[2823 rows x 25 columns]>

#Dropping unnecessary columns dfcsv = dfcsv.drop(['ADDRESSLINE1','ADDRESSLINE2','CITY','STATE','TERRITORY'],axis = 1)

dfcsv.isna().sum()

ORDERNUMBER 0

QUANTITYORDERED 0

PRICEEACH 0

ORDERLINENUMBER 0

SALES 0

ORDERDATE 0

STATUS 0

QTR\_ID 0

MONTH\_ID 0

YEAR\_ID 0

PRODUCTLINE 0

MSRP 0

PRODUCTCODE 0

CUSTOMERNAME 0

PHONE 0

POSTALCODE 76

COUNTRY 0

CONTACTLASTNAME 0

CONTACTFIRSTNAME 0 DEALSIZE 0 dtype: int64

#Filling all NA values with mode of the POSTALCODE column dfcsv = dfcsv['POSTALCODE'].fillna(dfcsv.POSTALCODE.mode(), inplace=True)

columns\_to\_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'CITY', 'STATE', 'TERRITORY']

# Drop the specified columns dfxlsx = dfxlsx.drop(columns\_to\_drop, axis=1)

postalcode\_mode = dfxlsx['POSTALCODE'].mode()[0] # [0] is used to get the first mode value

# Fill NA values in 'POSTALCODE' with the mode dfxlsx['POSTALCODE'].fillna(postalcode\_mode, inplace=True)

sns.histplot(x='STATUS', data=dfxlsx) plt.xlabel('Status') # Set the label for the x-axis plt.ylabel('Count') # Set the label for the y-axis plt.title('Distribution of STATUS') # Set the title for the plot plt.show()

plt.show()

sns.boxplot(x='STATUS', y='MONTH\_ID', data=dfxlsx) plt.xlabel('Status') # Set the label for the x-axis plt.ylabel('Month ID') # Set the label for the y-axis plt.title('Boxplot of STATUS vs MONTH\_ID') # Set the title for the plot plt.show()

plt.show()

#Checking the data only for shipped STATUS data1=dfxlsx[dfxlsx["STATUS"]=='Shipped'] data1.head()

ORDERNUMBER QUANTITYORDERED PRICEEACH ORDERLINENUMBER SALES ORDERDATE STAT

2/24/2003

0 10107 30 95.70 2 2871.00 Shipp

0:00

5/7/2003

1 10121 34 81.35 5 2765.90 Shipp

0:00

7/1/2003

2 10134 41 94.74 2 3884.34 Shipp

0:00

8/25/2003

3 10145 45 83.26 6 3746.70 Shipp

0:00

10/10/2003

4 10159 49 100.00 14 5205.27 Shipp

0:00

data1.shape

(2617, 20)

#Calculating sum for sales column sum\_sales = dfxlsx['SALES'].sum() print("Addition of all sales",sum\_sales)

Addition of all sales 10032628.85

#Calulating average for sales column sales\_avg = dfxlsx['SALES'].mean() print("Average of total sales = ",sales\_avg)

Average of total sales = 3553.889071909316

import sklearn import pandas as pd import seaborn as sns

# IQR

Q1 = np.percentile(dfxlsx['SALES'], 25, interpolation = 'midpoint')

Q3 = np.percentile(dfxlsx['SALES'], 75, interpolation = 'midpoint') IQR = Q3 - Q1 print("Old Shape: ", dfxlsx.shape)

# Upper bound upper = np.where(dfxlsx['SALES'] >= (Q3+1.5\*IQR))

# Lower bound lower = np.where(dfxlsx['SALES'] <= (Q1-1.5\*IQR))

# Removing the Outliers dfxlsx.drop(upper[0], inplace = True) dfxlsx.drop(lower[0], inplace = True) print("New Shape: ", dfxlsx.shape) sns.boxplot(x='SALES', data=dfxlsx)

Old Shape: (2823, 20)

New Shape: (2742, 20)

<ipython-input-57-b16e9d4f3aef>:6: DeprecationWarning: the `interpolation=` argument to percentile was renamed to `method=`, which Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to review the method they used. (Deprecated NumPy 1.2

Q1 = np.percentile(dfxlsx['SALES'], 25,

<ipython-input-57-b16e9d4f3aef>:9: DeprecationWarning: the `interpolation=` argument to percentile was renamed to `method=`, which Users of the modes 'nearest', 'lower', 'higher', or 'midpoint' are encouraged to review the method they used. (Deprecated NumPy 1.2

Q3 = np.percentile(dfxlsx['SALES'], 75,

<Axes: xlabel='SALES'>