**Data visualization**

**Experiment No.: 1**

**To study and analyze given dataset using visualization tool** Experiment No. 1

**1. Aim:** To understand and analyse a given dataset using a visualization tool, enabling the

identification of patterns, trends, and insights within the data.

**2. Objectives:**

* To gain hands-on experience with data visualization tools.
* To learn the significance of data visualization in interpreting large datasets.
* To develop skills in identifying key features and outliers in datasets through visual analysis.
* To relate the practical understanding of data visualization to real-world applications.

**3. Course Outcomes:**

* **CO1:** Explore open-source tools for data visualization and apply them to analyse datasets.
* **CO2:** Develop the ability to interpret and present data in a visually effective manner.
* **CO6:** Evaluate and critically assess the results of data visualization techniques.

**4. Hardware / Software Required:**

* **Software:** Python with libraries like Matplotlib, Seaborn, or an equivalent visualization tool such as Tableau, PowerBI.
* **Hardware:** A standard PC or laptop with at least 8GB RAM, 500GB HDD, and a compatible operating system (Windows/Linux/Mac OS).

**5. Theory:**

Data visualization is a crucial aspect of data analysis, allowing for the graphical representation of data to identify patterns, trends, and outliers. Effective visualization can reveal insights that might be missed in raw data, making it easier to comprehend complex datasets. Common types of visualizations include bar charts, histograms, scatter plots, and heatmaps, each serving a different purpose depending on the data being analysed. Visualization not only aids in better data understanding but also plays a significant role in presenting data-driven insights to stakeholders in a comprehensible manner.

**6. Algorithm / Design / Procedure / Flowchart / Analysis:**

**Procedure:**

1. Load the dataset into the chosen visualization tool.
2. Understand the dataset attributes and identify key features.
3. Select appropriate visualization techniques (e.g., histograms for distribution, scatter plots for correlations).
4. Generate the visualizations using the tool.
5. Analyse the visualizations to identify patterns, trends, and outliers.
6. Interpret the visualizations in the context of the dataset and derive meaningful insights.
7. Install and configure the required software (e.g., Python, Jupyter Notebook, and visualization libraries like Matplotlib or Seaborn).
8. Import the dataset and relevant libraries into the environment.
9. Perform exploratory data analysis to understand the dataset structure.
10. Choose the type of visualization based on the data and the objectives.
11. Generate visualizations and modify them for clarity and effectiveness.
12. Analyse the visualizations to draw conclusions about the data.

**Analysis:**

1. Use visualization to identify data distribution, outliers, trends, and correlations.
2. Discuss the implications of the findings and how they relate to the problem being studied.

**7. Results/Output Analysis:**

1. The results will include various visual representations such as bar charts, scatter plots, and

histograms.

1. The analysis will focus on the patterns observed, such as trends in data distribution or

relationships between variables.

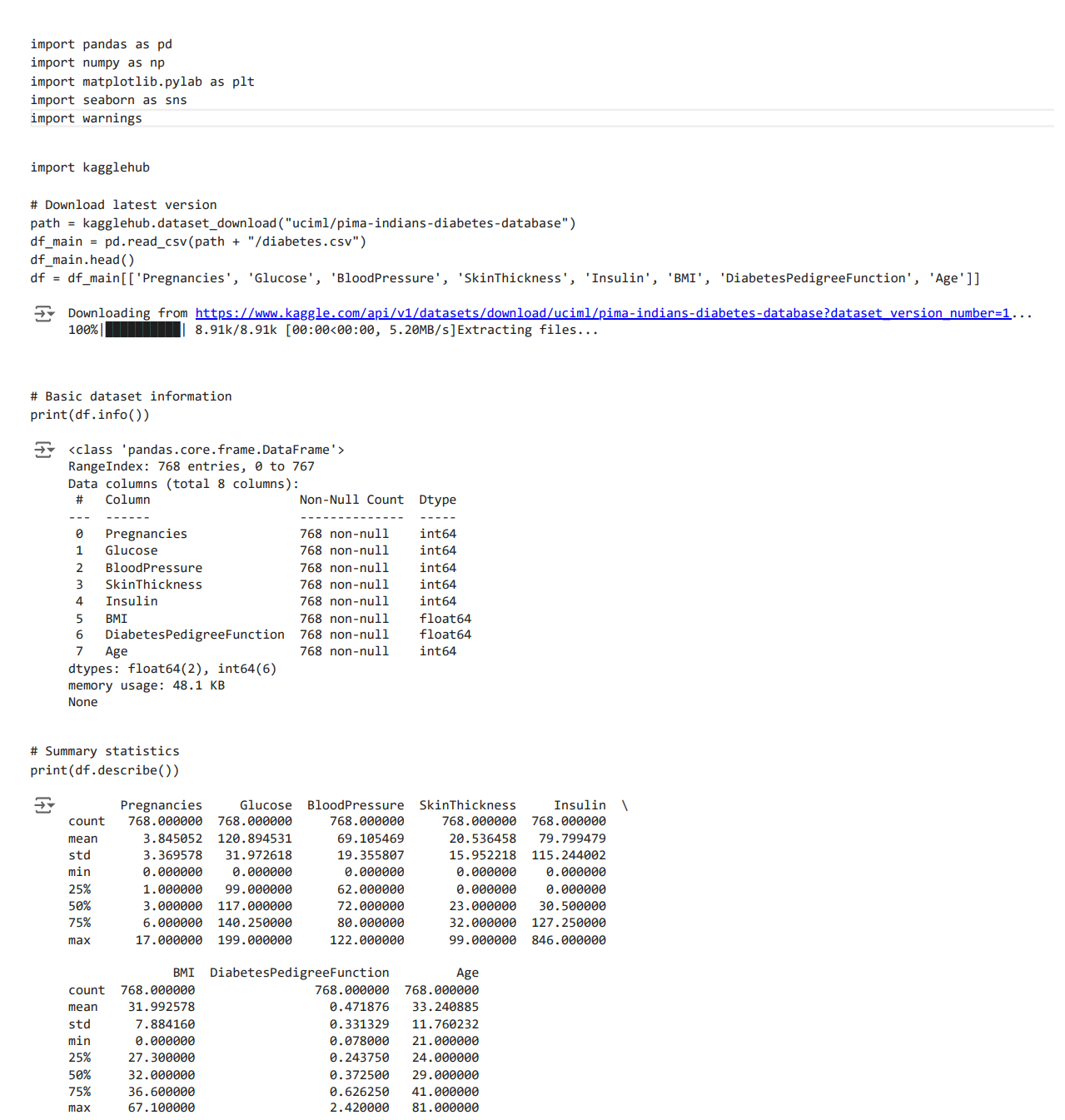
1. Students should attach screenshots of their visualizations along with a brief explanation of

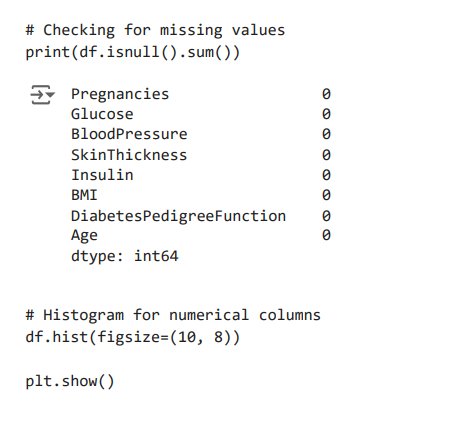
each.

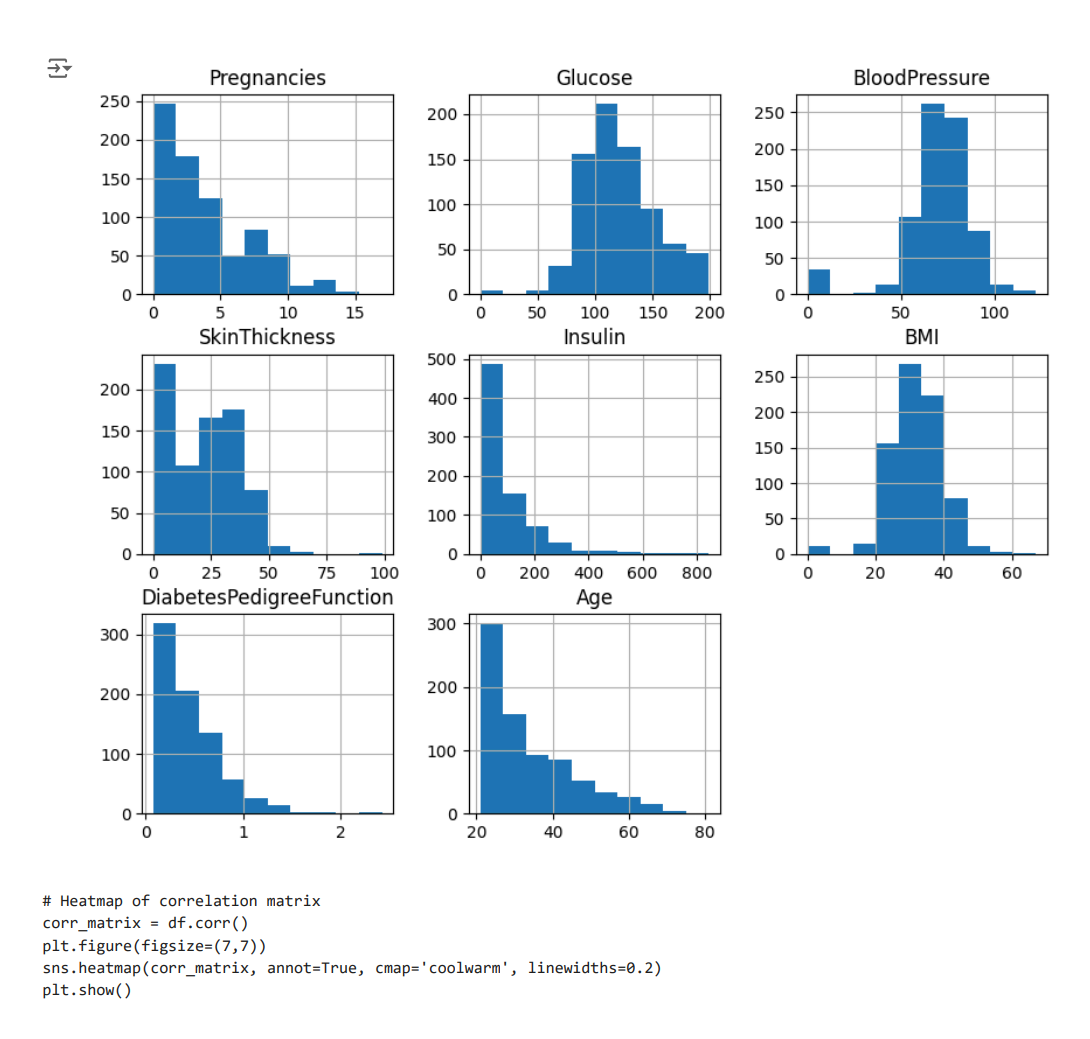
**8. Conclusions:**

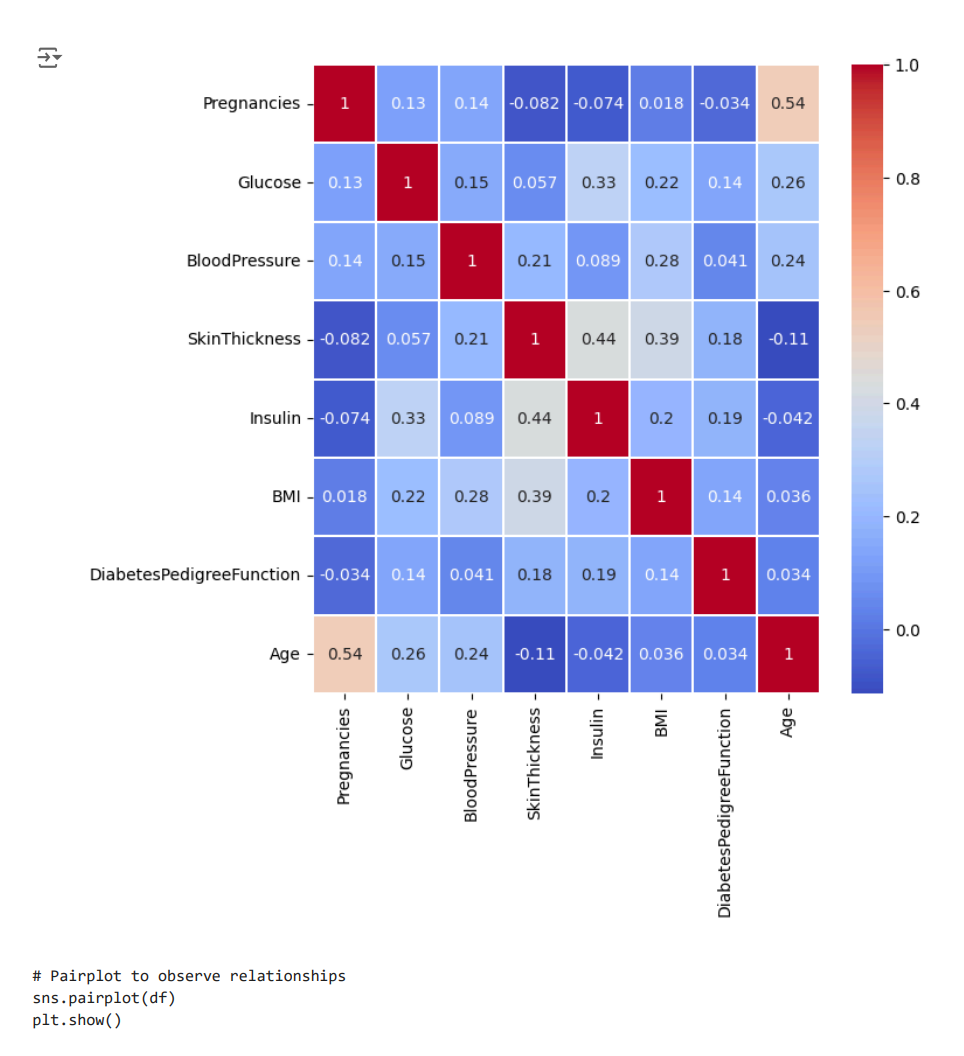
The experiment successfully demonstrates the importance of data visualization in analysing datasets. By using visual tools, students were able to identify key features and patterns in the data that would be difficult to detect otherwise. The results confirm that visual analysis is an essential step in data science, providing clarity and aiding in decision-making processes.

**9. Code and Output**

****



****

****

****

**Simple linear regression**

**Experiment No.: 2**

**To study and analyze simple linear regression and different cost functions**

Experiment No. 2

1. **Aim**:  
   To understand and implement simple linear regression, including the derivation of cost functions, and analyze their impact on model performance. The aim is to develop practical skills in applying regression techniques to real-world data and evaluating model accuracy.
2. **Objectives**:
   * To grasp the fundamental principles of simple linear regression.
   * To learn how to implement simple linear regression models.
   * To understand and compute different cost functions used in regression analysis.
   * To evaluate and compare the performance of regression models using various cost functions.
3. **Course Outcomes**:
   * **CO2**: Implement regression methods for real-world problems.
   * Understand the application of cost functions in evaluating regression models.
   * Analyze and interpret the performance of linear regression models.
4. **Hardware / Software Required**:
   * **Hardware**: Standard computer with internet access.
   * **Software**: Python/R with libraries such as NumPy, pandas, scikit-learn (Python) or tidyverse, caret (R).
5. **Theory**:  
   **Simple Linear Regression**:   
   Simple linear regression is a statistical technique used to model the relationship between a dependent variable and an independent variable . The model is represented as:

* is the observed value of the dependent variable for observation ,
* is the y-intercept of the regression line,
* is the slope of the regression line,
* is the value of the independent variable for observation ,
* is the error term, assumed to be normally distributed with mean 0 and variance .

1. **Cost Functions**:
   * **Mean Squared Error (MSE)**:  
     The MSE is a common cost function used to measure the average squared difference between the observed actual outcomes and the outcomes predicted by the model. It is calculated as:

where is the predicted value for observation based on the model.

* + **Mean Absolute Error (MAE)**:  
    The MAE measures the average magnitude of errors in a set of predictions, without considering their direction. It is calculated as:

1. **Algorithm / Design / Procedure / Flowchart / Analysis**:
   * **Algorithm**:
     1. **Import Libraries**: Import required libraries such as NumPy, pandas, and scikit-learn.
     2. **Load Data**: Load the dataset and perform necessary preprocessing steps like handling missing values.
     3. **Split Data**: Divide the dataset into training and testing sets.
     4. **Model Training**:
        + **Compute Coefficients**: Use the Ordinary Least Squares (OLS) method to estimate the parameters and . The estimates are given by:
        + **Fit Model**: Apply the linear regression model to the training data using these coefficients.
     5. **Prediction**: Use the trained model to predict outcomes on the testing set.
     6. **Evaluate Model**:
        + **Compute MSE**:
        + **Compute MAE**:
     7. **Analysis**: Compare the performance of the model using different cost functions and discuss the results.
   * **Design**:  
     Create a systematic approach to load, preprocess, and analyze the dataset. Visualize the regression line and residuals to understand the model’s fit.
   * **Procedure**:
     1. **Data Preparation**: Load the dataset, handle missing values, and scale features if necessary.
     2. **Model Training**: Fit the simple linear regression model to the training data using the derived coefficients.
     3. **Evaluation**: Test the model on the testing data, compute MSE and MAE, and analyze the results.
     4. **Analysis**: Use visualizations and statistical summaries to interpret the performance of the model.
   * **Flowchart**:
     1. Start
     2. Load Data
     3. Preprocess Data
     4. Split Data
     5. Train Model (Compute Coefficients, Fit Model)
     6. Predict
     7. Compute MSE and MAE
     8. Analyze Results
     9. End
   * **Analysis**:  
     Examine the MSE and MAE values to determine the accuracy of the regression model. Discuss any discrepancies or patterns observed and how they align with the theoretical expectations.
2. **Results/Output Analysis**:  
   Present the results of the cost functions (MSE and MAE) for the simple linear regression model. Include graphs such as the regression line plot and residual plots. Discuss how different cost functions affect model performance.
3. **Conclusions**:  
   Summarize the findings of the experiment, including the effectiveness of the simple linear regression model and the impact of different cost functions. Confirm whether the experiment’s aim was achieved and discuss any insights gained.
4. **Code and Output**

# Importing necessary libraries import pandas as pd

import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error from sklearn.preprocessing import LabelEncoder

import kagglehub import warnings

warnings.filterwarnings('ignore')

# Download latest version

path = kagglehub.dataset\_download("shibumohapatra/house-price")

df = pd.read\_csv(f"{path}/1553768847-housing.csv")

df.head()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **longitude** | **latitude** | **housing\_median\_age** | **total\_rooms** | **total\_bedrooms** | **population** | **households** | **median\_income** | **ocean\_proximity** | **med** |
| **0** -122.23 | 37.88 | 41 | 880 | 129.0 | 322 | 126 | 8.3252 | NEAR BAY |  |
| **1** -122.22 | 37.86 | 21 | 7099 | 1106.0 | 2401 | 1138 | 8.3014 | NEAR BAY |  |
| **2** -122.24 | 37.85 | 52 | 1467 | 190.0 | 496 | 177 | 7.2574 | NEAR BAY |  |
| **3** -122.25 | 37.85 | 52 | 1274 | 235.0 | 558 | 219 | 5.6431 | NEAR BAY |  |

Next steps:

# Encoding the 'ocean\_proximity' column label\_encoder = LabelEncoder()

df['ocean\_proximity'] = label\_encoder.fit\_transform(df['ocean\_proximity']) df.head()

Generate code with df

 View recommended plots

New interactive sheet



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **longitude** | **latitude** | **housing\_median\_age** | **total\_rooms** | **total\_bedrooms** | **population** | **households** | **median\_income** | **ocean\_proximity** | **med** |
| **0** -122.23 | 37.88 | 41 | 880 | 129.0 | 322 | 126 | 8.3252 | 3 |  |
| **1** -122.22 | 37.86 | 21 | 7099 | 1106.0 | 2401 | 1138 | 8.3014 | 3 |  |
| **2** -122.24 | 37.85 | 52 | 1467 | 190.0 | 496 | 177 | 7.2574 | 3 |  |
| **3** -122.25 | 37.85 | 52 | 1274 | 235.0 | 558 | 219 | 5.6431 | 3 |  |

Next steps:

# Basic dataset information df.info()

 <class 'pandas.core.frame.DataFrame'> RangeIndex: 20640 entries, 0 to 20639 Data columns (total 10 columns):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # |  | Column | Non-Null Count |  | Dtype |
| 0 |  | longitude | 20640 non-null |  | float64 |
| 1 |  | latitude | 20640 non-null |  | float64 |
| 2 |  | housing\_median\_age | 20640 non-null |  | int64 |
| 3 |  | total\_rooms | 20640 non-null |  | int64 |
| 4 |  | total\_bedrooms | 20433 non-null |  | float64 |
| 5 |  | population | 20640 non-null |  | int64 |
| 6 |  | households | 20640 non-null |  | int64 |
| 7 |  | median\_income | 20640 non-null |  | float64 |
| 8 |  | ocean\_proximity | 20640 non-null |  | int64 |
| 9 |  | median\_house\_value | 20640 non-null |  | int64 |

dtypes: float64(4), int64(6) memory usage: 1.6 MB

# Summary statistics df.describe()

# 

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **longitude** | **latitude** | **housing\_median\_age** | **total\_rooms** | **total\_bedrooms** | **population** | **households** | **median\_income** | **ocean\_** |
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20 |
| **mean** | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.870553 | 1425.476744 | 499.539680 | 3.870671 |  |
| **std** | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.385070 | 1132.462122 | 382.329753 | 1.899822 |  |
| **min** | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 | 0.499900 |  |
| **25%** | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.000000 | 787.000000 | 280.000000 | 2.563400 |  |
| **50%** | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.000000 | 1166.000000 | 409.000000 | 3.534800 |  |
| **75%** | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.000000 | 1725.000000 | 605.000000 | 4.743250 |  |

# Checking for missing values df.isna().sum()

**0**

**longitude** 0

**latitude** 0

**housing\_median\_age** 0

**total\_rooms** 0

**total\_bedrooms** 207

**population** 0

**households** 0

**median\_income** 0

**ocean\_proximity** 0

**median\_house\_value** 0

df = df.dropna() df.isna().sum()

**0**

**longitude** 0

**latitude** 0

**housing\_median\_age** 0

**total\_rooms** 0

**total\_bedrooms** 0

**population** 0

**households** 0

**median\_income** 0

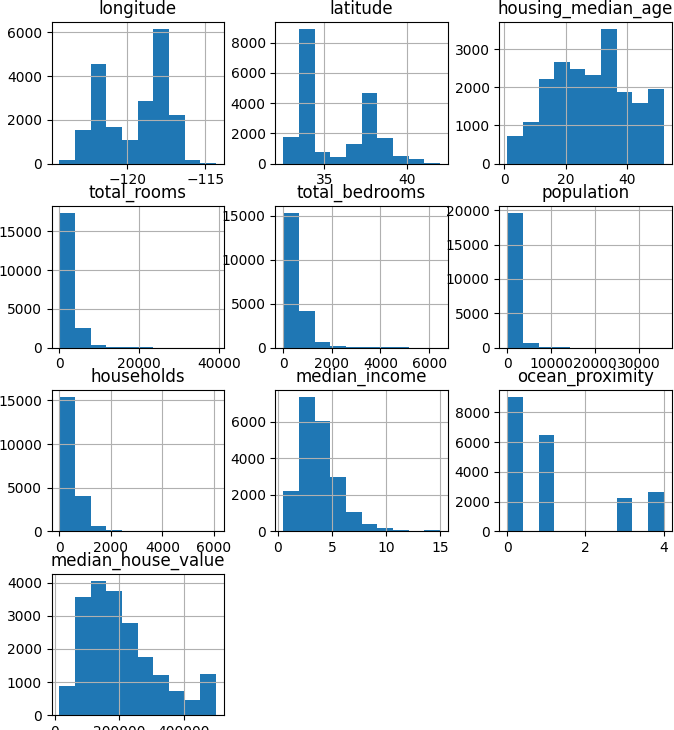
**ocean\_proximity** 0

**median\_house\_value** 0

# Histograms for numerical columns df.hist(figsize=(8, 9))

plt.show()

# 



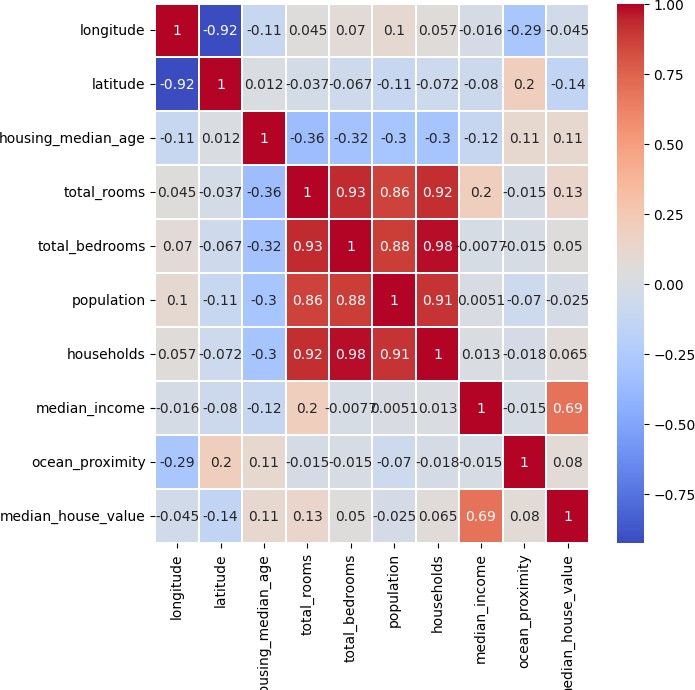
# Correlation matrix

corr\_matrix = df.corr()

# Heatmap of correlation matrix plt.figure(figsize=(7,7))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.2) plt.show()

# 



# Pairplot to observe relationships sns.pairplot(df)

plt.show()

# 

# Boxplots to observe distributions plt.figure(figsize=(6, 2))

sns.boxplot(data=df, orient='h')

plt.show()

# Plotting observed vs. predicted values for training data plt.figure(figsize=(6,2))

plt.scatter(y\_train, y\_pred\_train)

plt.xlabel('Observed Median House Value') plt.ylabel('Predicted Median House Value')

plt.title('Observed vs. Predicted Values on Training Data') plt.show()



**Bayesian classifier**

**Experiment No.: 3**

**To study and analyze Machine learning classifier**

Experiment No. 3

1. **Aim:**

To understand and implement the Machine learning classifier for classification tasks and evaluate its performance using a given dataset.

1. **Objectives:**
2. To learn the theoretical foundations of the Bayesian classifier.
3. To implement the ML classifier using a sample dataset.
4. To calculate and interpret various probabilities and predicted values involved in ML classification.
5. To compare the performance of the ML classifier with other classification methods.
6. **Course Outcomes:**
7. **CO3:** Build a Bayesian classifier for real-world problems.
8. **CO6:** Evaluate all implemented methods for real-world problems.
9. **Hardware / Software Required:**
10. **Software:** Python with libraries such as NumPy, pandas, and Scikit-learn (or any other suitable data analysis tool).
11. **Hardware:** Computer with a basic configuration to run the software.
12. **Theory:**

The Bayesian classifier is a probabilistic classifier based on Bayes' Theorem, which uses conditional probability principles to classify data. It assumes that features are conditionally independent given the class label. The classifier calculates the posterior probability of each class given the feature values and chooses the class with the highest probability.

**Bayes' Theorem:**

where:

is the posterior probability of class given feature vector .

is the likelihood of observing features given class .

is the prior probability of class .

is the marginal likelihood of the features .

**Support Vector Machine (SVM):**

SVM is a supervised learning algorithm used for classification and regression tasks. It aims to find a hyperplane that best separates the classes in the feature space. The main concepts in SVM include:

1. **Hyperplane:** A decision boundary that separates different classes in the feature space.
2. **Margin:** The distance between the hyperplane and the nearest data points from either class. SVM aims to maximize this margin.
3. **Support Vectors:** Data points that are closest to the hyperplane and influence its position and orientation.

**Mathematical Formulation:**

1. **Linear SVM:**

For a linearly separable dataset, the goal is to find a hyperplane that maximizes the margin. The optimization problem can be formulated as:

subject to:

where is the weight vector, is the bias, and is the class label of .

1. **Non-linear SVM:**

For non-linearly separable data, SVM uses kernel functions to transform the data into a higher-dimensional space where a linear separator can be found. Common kernels include:

* + **Polynomial Kernel:**
  + **Radial Basis Function (RBF) Kernel:**

**Decision Tree:**

A decision tree is a supervised learning algorithm used for classification and regression tasks. It splits the data into subsets based on the value of input features, creating a tree-like model of decisions.

* **Tree Structure:**
  + **Root Node:** Represents the entire dataset, which is split into two or more homogeneous sets.
  + **Decision Nodes:** Nodes where the data is split based on feature values.
  + **Leaf Nodes:** Terminal nodes that represent the outcome (class label in classification).
* **Entropy:** A measure of impurity or disorder in the dataset.

where is the proportion of class in set .

* **Information Gain:** The reduction in entropy after a split.

where is the attribute, and is the subset of for value .

* **Gini Index:** A metric for measuring node impurity.

1. **Analysis:**
   1. Evaluate the accuracy of the classifier using metrics such as precision, recall, and F1-score.
   2. Compare the performance of the Bayesian classifier with other classifiers if required.
2. **Results/Output Analysis:**
3. Present the calculated prior, likelihood, and posterior probabilities for each class.
4. The predicted class for new instances based on the highest posterior probability.
5. Include confusion matrices, classification reports, or accuracy scores as part of the output.
6. **Conclusions:**
7. Discuss the effectiveness of the Bayesian classifier for the given dataset.
8. Explain how well the calculated probabilities align with the actual class labels.
9. Reflect on the strengths and limitations of using a Bayesian classifier.
10. **Code and Output**

# Importing necessary libraries import pandas as pd

import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import precision\_score, recall\_score, f1\_score, confusion\_matrix, classification\_report import warnings

warnings.filterwarnings('ignore')

import kagglehub

# Download latest version

path = kagglehub.dataset\_download("uciml/iris")

df = pd.read\_csv(f"{path}/Iris.csv")

df.head()

 Downloading from <https://www.kaggle.com/api/v1/datasets/download/uciml/iris?dataset_version_number=2>...

100%|██████████| 3.60k/3.60k [00:00<00:00, 3.29MB/s]Extracting files...

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **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 |

Next steps:

Generate code with df

 View recommended plots

New interactive sheet

df.shape

 (150, 6)

df.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

df.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 |

df.isna().sum()

1. **Id** 0
2. **SepalLengthCm** 0
3. **SepalWidthCm** 0
4. **PetalLengthCm** 0
5. **PetalWidthCm** 0
6. **Species** 0

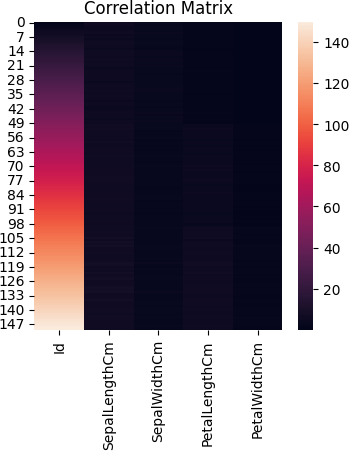
sns.pairplot(df) plt.show()



plt.figure(figsize=(4,4))

sns.heatmap(df.drop(columns=["Species"])) plt.title('Correlation Matrix')

plt.show()



# Calculating precision, recall, and F1-score

precision = precision\_score(y\_test, y\_pred, average='weighted') recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted') print(f'Precision: {precision}')

**Principal Component Analysis**

**Experiment No.: 4**

**To study and implement PCA algorithm for real world data set**

Experiment No. 4

1. **Aim:**

To understand and implement the Principal Component Analysis (PCA) algorithm on a real-world dataset to reduce dimensionality and analyze the results.

1. **Objectives:**
2. To grasp the mathematical foundation of PCA, including eigenvectors and

eigenvalues.

1. To apply PCA for dimensionality reduction on a high-dimensional dataset.
2. To visualize the data in reduced dimensions and interpret the significance of

principal components.

4. To evaluate the variance retained after applying PCA.

1. **Course Outcomes:**  
    By the end of this experiment, students should be able to:

1. Explain the PCA algorithm and its importance in reducing dimensionality.

2. Implement PCA on a real-world dataset.

3. Interpret the results of PCA, including the explained variance by principal

components.

4. Use PCA to enhance the performance of machine learning models by reducing

noise and redundancy in data.

1. **Hardware / Software Required:**
2. **Hardware:** A computer with at least 8 GB of RAM for handling large datasets.
3. **Software:** Python (with libraries like NumPy, pandas, matplotlib, scikit-learn), R
4. **Theory:**  
    Principal Component Analysis (PCA) is a dimensionality reduction technique used to

reduce the number of variables in a dataset while retaining the maximum possible

variance. PCA identifies the directions (principal components) in which the data

varies the most and projects the data onto these directions.

**Mathematical Concepts:**

* **Covariance Matrix:** Measures the variance and the relationship between different variables in the data.
* **Eigenvectors and Eigenvalues:** Eigenvectors of the covariance matrix point to the directions of maximum variance (principal components), and the corresponding eigenvalues represent the magnitude of variance in these directions.

**Steps in PCA:**

1. **Standardization:** Standardize data to have a mean of 0 and a variance of 1.
2. **Covariance Matrix Computation:** Compute the covariance matrix to understand how the variables of the dataset relate to each other.
3. **Eigen Decomposition:** Compute the eigenvectors and eigenvalues of the covariance matrix.
4. **Feature Vector Formation:** Select the top k eigenvectors corresponding to the largest eigenvalues to form a feature vector.
5. **Projection of Data:** Transform the original dataset using the feature vector to obtain the principal components.
6. **Algorithm / Design / Procedure / Flowchart / Analysis:**

**Algorithm:**

1. **Standardize the Dataset:**

is the standardized data, is the original data, is the mean, and is the standard deviation.

1. **Compute the Covariance Matrix:**

is the covariance matrix.

1. **Calculate Eigenvectors and Eigenvalues:** Solve the equation:

are the eigenvalues and are the eigenvectors.

1. **Sort and Select Principal Components:** Choose the top k eigenvectors corresponding to the largest eigenvalues.
2. **Transform the Data:** Project the original data onto the new k-dimensional

space:

Where is the matrix with the selected eigenvectors.

1. **Procedure:**
2. Load the dataset and standardize it.
3. Calculate the covariance matrix.
4. Perform eigen decomposition on the covariance matrix.
5. Select the top k principal components.
6. Project the data onto these components.
7. Visualize the transformed data in the reduced dimensions.

Consider a dataset with two features. After applying PCA, you might find that 95% of the variance is explained by the first principal component. The original two-dimensional data can be effectively reduced to one dimension, simplifying further analysis.

1. **Results/Output Analysis:**  
    Present the principal components obtained, the explained variance, and the resulting

reduced-dimensional data. Analyze how much of the original variance is retained and

how PCA has simplified the dataset.

1. **Conclusions:**  
   Summarize the effectiveness of PCA in reducing dimensionality while preserving important information. Discuss how PCA can improve the efficiency of machine learning algorithms by removing noise and redundant features.
2. **Code and Output**

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

import warnings

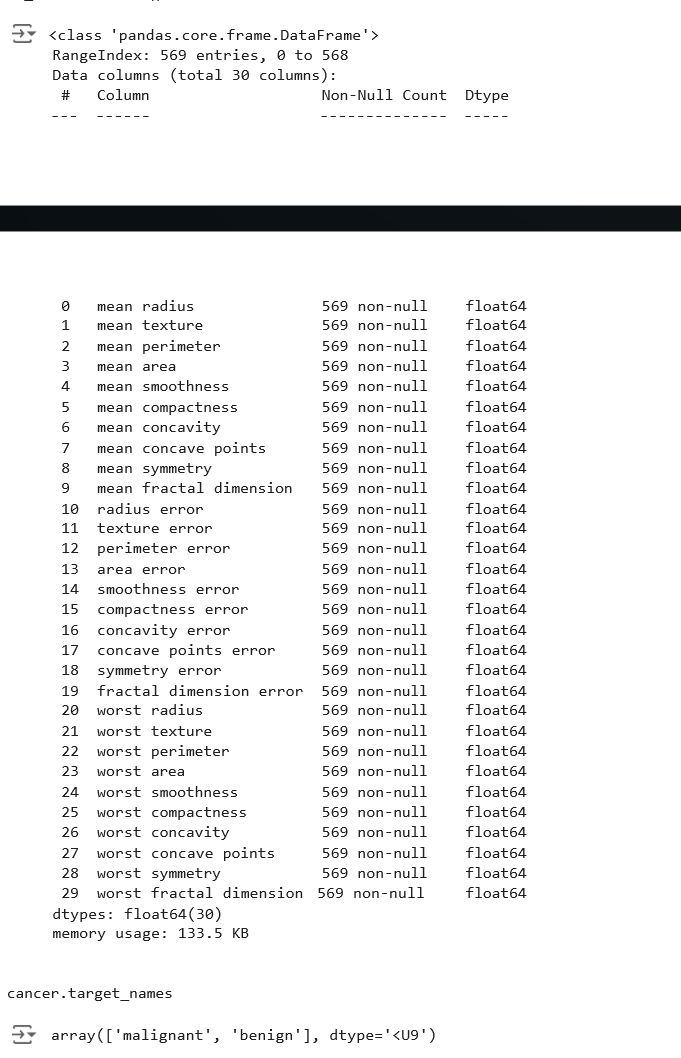
warnings.simplefilter(action='ignore', category=FutureWarning)

%matplotlib inline

from sklearn.datasets import load\_breast\_cancer cancer = load\_breast\_cancer()

cancer.keys()

df\_features = pd.DataFrame(cancer.data, columns = cancer.feature\_names) df\_features.info()



 array(['malignant', 'benign'], dtype='<U9')

df\_target = pd.DataFrame(cancer.target, columns=['target']) df\_target['target'].value\_counts()

df = pd.concat([df\_features, df\_target], axis=1)

df['target'] = df['target'].apply(lambda x: "Benign"

if x == 1 else "Malignant")

df.head(5)



**mean radius**

**mean texture**

**mean perimeter**

**mean area**

**mean smoothness**

**mean compactness**

**mean concavity**

**mean concave points**

**mean symmetry**

**mean fractal**

**dimension**

**...**

**worst texture**

**worst perimeter**

**wor ar**

**0** 17.99 10.38 122.80 1001.0 0.11840 0.27760 0.3001 0.14710 0.2419 0.07871 ... 17.33 184.60 201

**1** 20.57 17.77 132.90 1326.0 0.08474 0.07864 0.0869 0.07017 0.1812 0.05667 ... 23.41 158.80 195

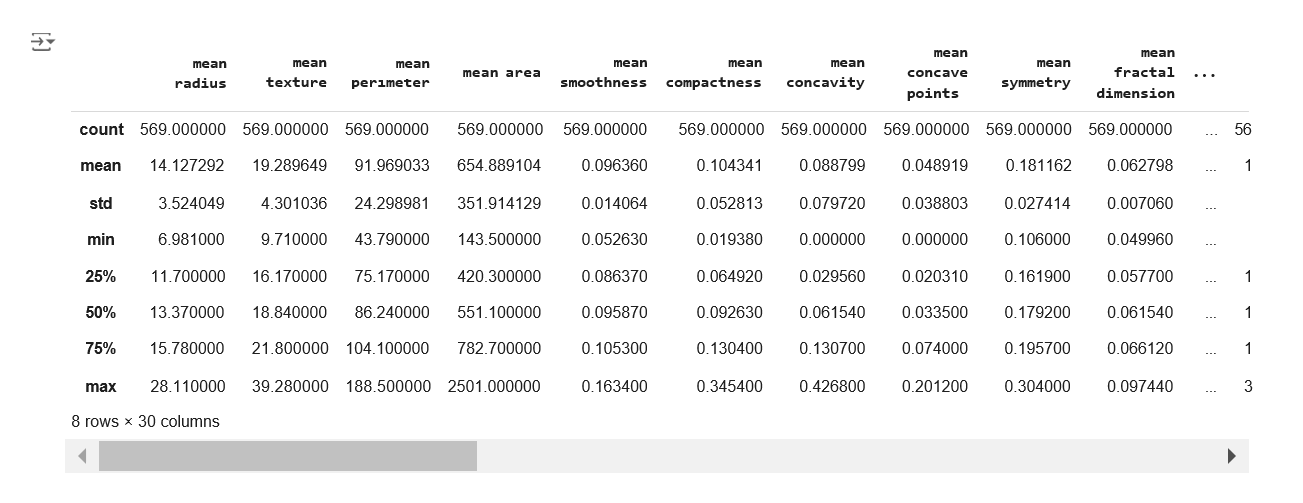
**2** 19.69 21.25 130.00 1203.0 0.10960 0.15990 0.1974 0.12790 0.2069 0.05999 ... 25.53 152.50 170

**3** 11.42 20.38 77.58 386.1 0.14250 0.28390 0.2414 0.10520 0.2597 0.09744 ... 26.50 98.87 56

**4** 20.29 14.34 135.10 1297.0 0.10030 0.13280 0.1980 0.10430 0.1809 0.05883 ... 16.67 152.20 157

5 rows × 31 columns

df.describe()



# Set style

sns.set\_style('darkgrid')

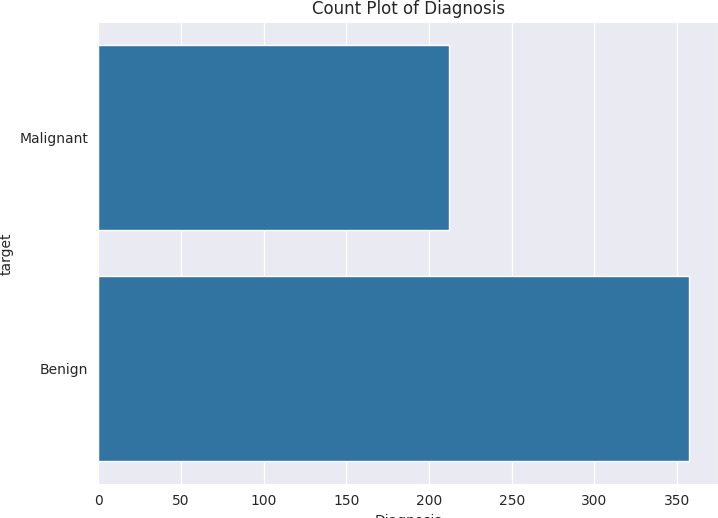
df['target'].value\_counts()



plt.figure(figsize=(8, 6)) sns.countplot(df['target']) plt.xlabel("Diagnosis")

plt.title("Count Plot of Diagnosis")

 Text(0.5, 1.0, 'Count Plot of Diagnosis')



from sklearn.preprocessing import StandardScaler scaler = StandardScaler()

scaler.fit(df\_features)

features\_scaled = scaler.transform(df\_features)

features\_scaled = pd.DataFrame(data=features\_scaled,

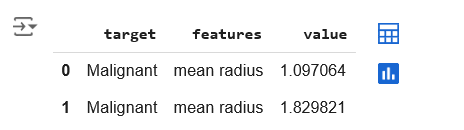
columns=df\_features.columns)

df\_scaled = pd.concat([features\_scaled, df['target']], axis=1)

df\_scaled\_melt = pd.melt(df\_scaled, id\_vars='target',

var\_name='features', value\_name='value')

df\_scaled\_melt.head(3)



def violin\_plot(features, name): """

This function creates violin plots of features given in the argument. """

# Create query query = ''

for x in features:

query += "features == '" + str(x) + "' or " query = query[0:-4]

# Create data for visualization

data = df\_scaled\_melt.query(query)

# Plot figure

plt.figure(figsize=(12, 6)) sns.violinplot(x='features',

y='value',

hue='target', data=data,

split=True,

inner="quart") plt.xticks(rotation=45)

plt.title(name)

plt.xlabel("Features")

plt.ylabel("Standardize Value")

def swarm\_plot(features, name): """

This function creates swarm plots of features given in the argument. """

# Create query query = ''

for x in features:

query += "features == '" + str(x) + "' or " query = query[0:-4]

# Create data for visualization

data = df\_scaled\_melt.query(query)

# Plot figure

plt.figure(figsize=(12, 6))

sns.swarmplot(x='features', y='value', hue='target', data=data) plt.xticks(rotation=45)

plt.title(name)

plt.xlabel("Features")

plt.ylabel("Standardize Value")

def box\_plot(features, name): """

This function creates box plots of features given in the argument. """

# Create query query = ''

for x in features:

query += "features == '" + str(x) + "' or " query = query[0:-4]

# Create data for visualization

data = df\_scaled\_melt.query(query)

# Plot figure

plt.figure(figsize=(12, 6))

sns.boxplot(x='features', y='value', hue='target', data=data) plt.xticks(rotation=45)

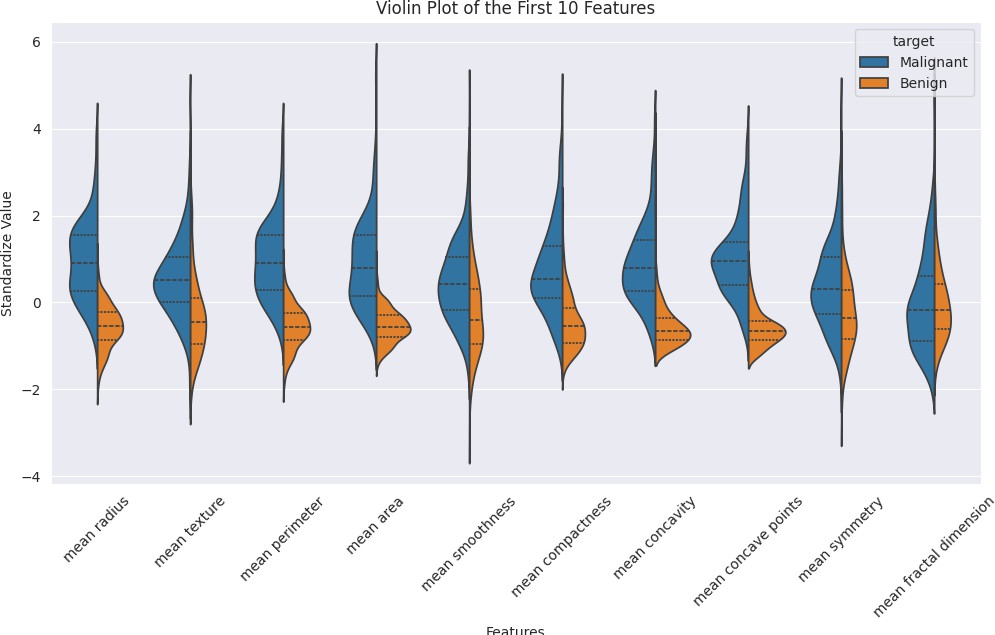
plt.title(name)

plt.xlabel("Features")

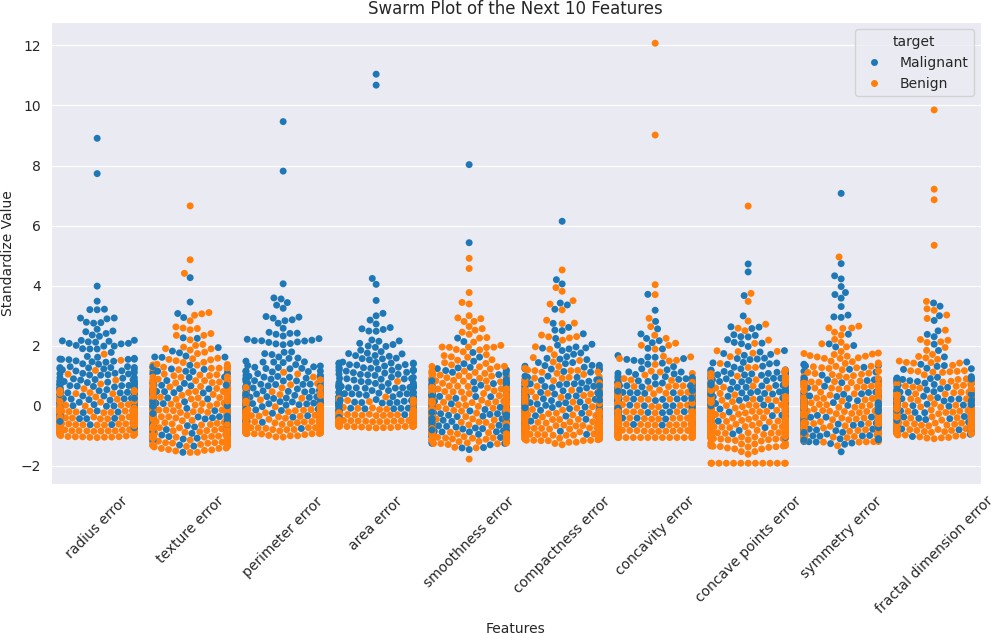
plt.ylabel("Standardize Value")

violin\_plot(df.columns[0:10], "Violin Plot of the First 10 Features")



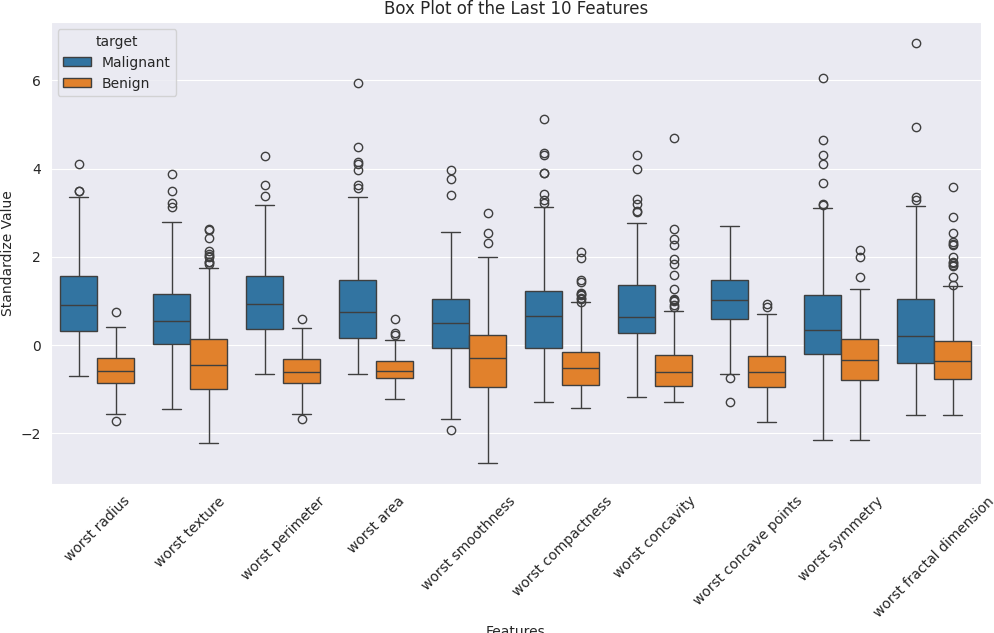


swarm\_plot(df.columns[10:20], "Swarm Plot of the Next 10 Features")



box\_plot(df.columns[20:30], "Box Plot of the Last 10 Features")





def correlation(var): """

1. Print correlation
2. Create jointplot """

# Print correlation

print("Correlation: ", df[[var[0], var[1]]].corr().iloc[1, 0])

# Create jointplot

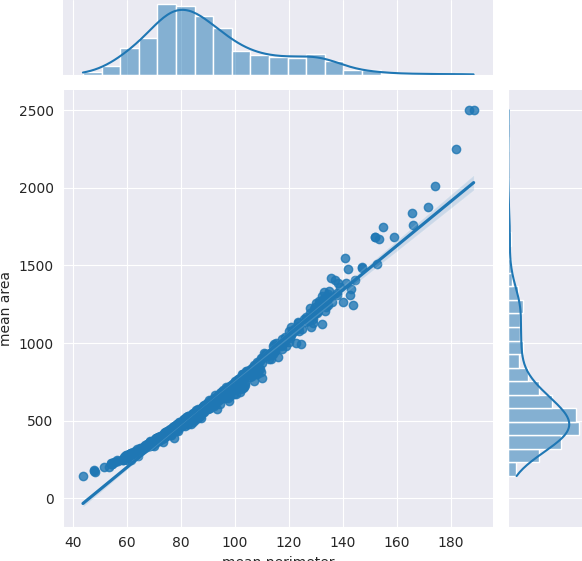
plt.figure(figsize=(6, 6))

sns.jointplot(x=var[0], y=var[1], data=df, kind='reg')

correlation(['mean perimeter', 'mean area'])

 Correlation: 0.9865068039913907

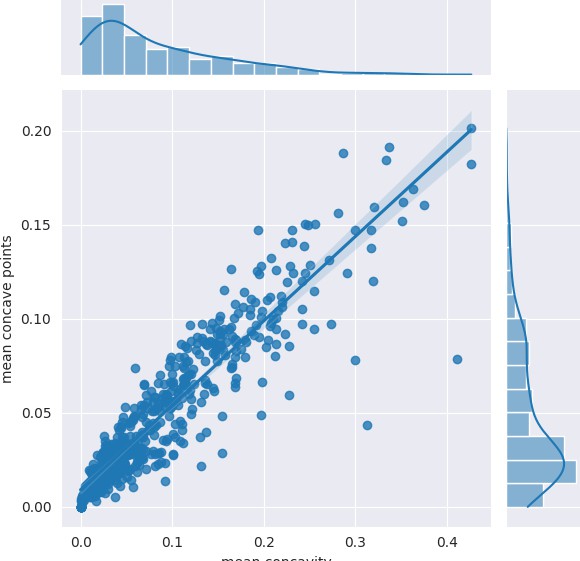
<Figure size 600x600 with 0 Axes>



correlation(['mean concavity', 'mean concave points'])

 Correlation: 0.9213910263788588

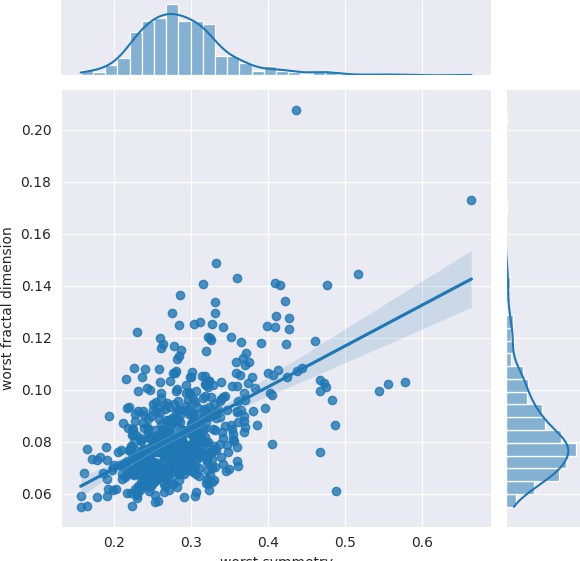
<Figure size 600x600 with 0 Axes>



correlation(['worst symmetry', 'worst fractal dimension'])

 Correlation: 0.537848206253609

<Figure size 600x600 with 0 Axes>



# Create correlation matrix

numerical\_df = df.select\_dtypes(include=['number'])

corr\_mat = numerical\_df.corr() # Use numerical\_df here

# Create mask

mask = np.zeros\_like(corr\_mat, dtype=bool) # bool is preferred over np.bool mask[np.triu\_indices\_from(mask, k=1)] = True

# Plot heatmap

plt.figure(figsize=(15, 10))

sns.heatmap(corr\_mat, annot=True, fmt='.1f', cmap='RdBu\_r', vmin=-1, vmax=1,

mask=mask, cbar\_kws={'shrink': 0.8}) # Adjust color bar size

plt.title('Correlation Matrix Heatmap') # Add a title for clarity plt.tight\_layout() # Improve layout to prevent overlapping elements plt.show() # Display the plot

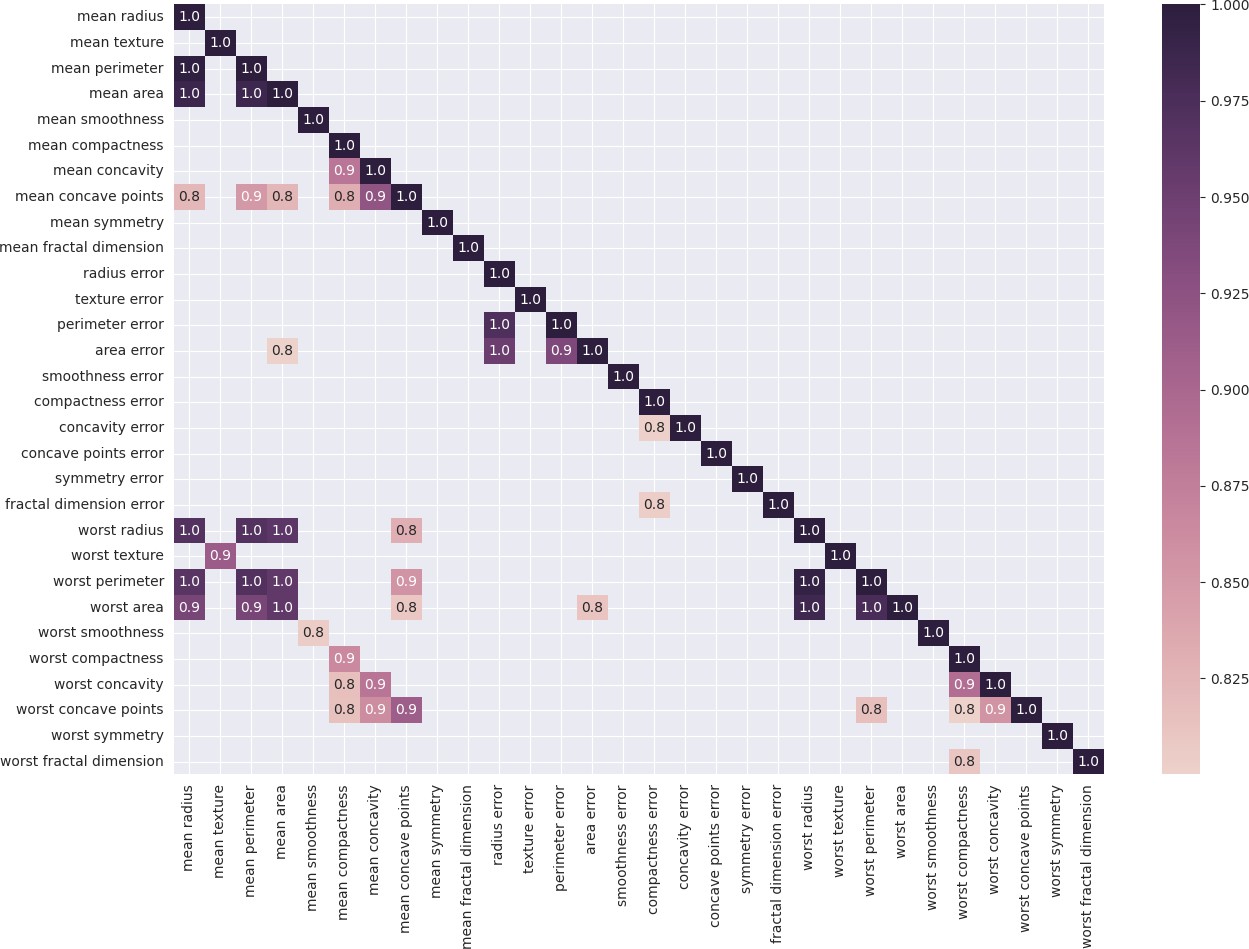
 

plt.figure(figsize=(15, 10))

sns.heatmap(corr\_mat[corr\_mat > 0.8], annot=True,

fmt='.1f', cmap=sns.cubehelix\_palette(200), mask=mask)





from sklearn.feature\_selection import SelectKBest, chi2 feature\_selection = SelectKBest(chi2, k=5)

feature\_selection.fit(df\_features, df\_target)

selected\_features = df\_features.columns[feature\_selection.get\_support()] print("The five selected features are: ", list(selected\_features))

 The five selected features are: ['mean perimeter', 'mean area', 'area error', 'worst perimeter', 'worst area']

X = pd.DataFrame(feature\_selection.transform(df\_features), columns=selected\_features)

X.head()

Generate code with X

 View recommended plots

New interactive sheet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **mean perimeter** | **mean area** | **area error** | **worst perimeter** | **worst area** |
| **0** 122.80 | 1001.0 | 153.40 | 184.60 | 2019.0 |
| **1** 132.90 | 1326.0 | 74.08 | 158.80 | 1956.0 |
| **2** 130.00 | 1203.0 | 94.03 | 152.50 | 1709.0 |
| **3** 77.58 | 386.1 | 27.23 | 98.87 | 567.7 |

Next steps:

sns.pairplot(pd.concat([X, df['target']], axis=1), hue='target')

<seaborn.axisgrid.PairGrid at 0x78932f3ad690>

from sklearn.model\_selection import train\_test\_split y = df\_target['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.33, random\_state=42)

from sklearn.ensemble import RandomForestClassifier rfc = RandomForestClassifier(n\_estimators=200)

rfc.fit(X\_train, y\_train)

y\_pred = rfc.predict(X\_test)

from sklearn.metrics import confusion\_matrix, classification\_report print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

**K-means clustering**

**Experiment No.: 5**

**To study and implement K-means clustering**

Experiment No. 5

1. **Aim:**  
   The objective of this experiment is to study the K-means clustering algorithm and implement it on a real-world dataset. The experiment aims to demonstrate how K-means clustering groups data into distinct clusters based on feature similarities.
2. **Objectives:**
   1. Understand the working principle of the K-means clustering algorithm.
   2. Learn to implement K-means clustering on a dataset.
   3. Analyze the results of clustering and evaluate the quality of clusters.
   4. Develop skills to interpret the significance of clustering in real-world applications.
3. **Course Outcomes:**
   1. CO1: Students will be able to explain the concept of clustering and its importance in unsupervised learning.
   2. CO2: Students will demonstrate the ability to implement the K-means clustering algorithm on a dataset.
   3. CO3: Students will analyze and interpret the results obtained from clustering to draw meaningful conclusions.
4. **Hardware/Software Required:**
   1. Software: Python with libraries like NumPy, Pandas, Matplotlib, and Scikit-learn.
   2. Hardware: Standard computing system with at least 4 GB RAM.
5. **Theory:**  
   K-means is an iterative clustering algorithm that aims to partition a dataset into distinct, non-overlapping clusters. The algorithm works by initializing centroids randomly, assigning each data point to the nearest centroid, and then recalculating the centroids based on the assigned points. This process is repeated until the centroids no longer change significantly, indicating convergence.

The objective of K-means clustering is to minimize the within-cluster variance, defined mathematically as:

Where:

* 1. is a data point,
  2. is the centroid of cluster ,
  3. is the squared Euclidean distance between a point and the cluster centroid.

1. **Algorithm / Design / Procedure / Flowchart / Analysis:**

**Algorithm:**

* 1. Initialize centroids randomly.
  2. Assign each data point to the nearest centroid.
  3. Recalculate the centroids as the mean of all points assigned to each cluster.
  4. Repeat steps 2-3 until the centroids do not change significantly.

1. **Numerical Example:**

Consider the following 2D data points: , , , . Assume and the initial centroids are randomly chosen as and .

**Step 1:** Compute the distance of each point to the centroids using the Euclidean distance formula.

Assign each point to the nearest centroid:

* and to the first centroid,
* and to the second centroid.

**Step 2:** Recalculate the centroids:

* New centroid 1: Mean of and =
* New centroid 2: Mean of and =

**Step 3:** Repeat the process until convergence.

1. **Results/Output Analysis:**
   1. After applying K-means clustering, the dataset should be divided into two clusters. The centroids should represent the mean of the data points in each cluster.
   2. Analyze the quality of clustering using metrics such as within-cluster sum of squares (WCSS) or silhouette score.
2. **Conclusions:**  
   The experiment successfully demonstrated the implementation of the K-means clustering algorithm. The results show how data points can be grouped into clusters based on their features, and the iterative nature of the algorithm was observed through the convergence of centroids.
3. **Code and Output**

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

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

**%matplotlib inline**

**import warnings**

**warnings.filterwarnings('ignore')**

**import kagglehub**

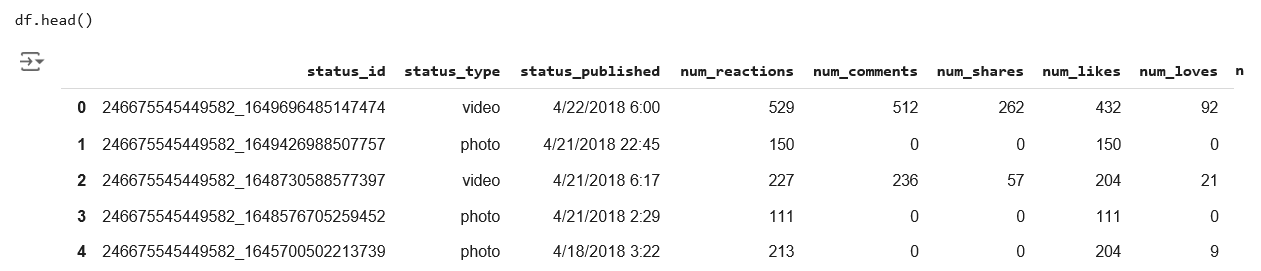
**# Download latest version**

**path = kagglehub.dataset\_download("ashishg21/facebook-live-sellers-in-thailand-uci-ml-repo") df = pd.read\_csv(f"{path}/Live.csv")**

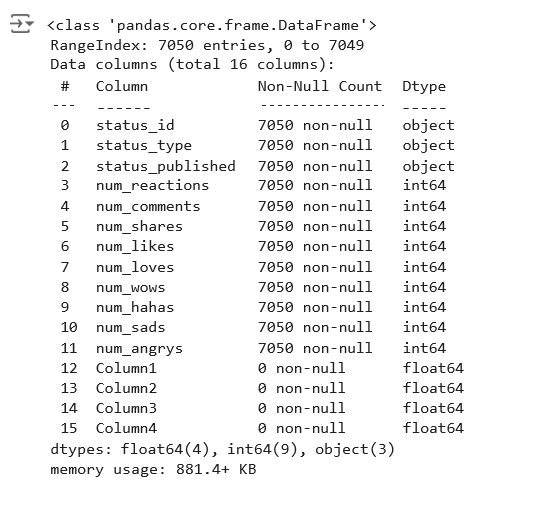
** Downloading from** [**https://www.kaggle.com/api/v1/datasets/download/ashishg21/facebook-live-sellers-in-thailand-uci-ml-repo?dataset\_ve**](https://www.kaggle.com/api/v1/datasets/download/ashishg21/facebook-live-sellers-in-thailand-uci-ml-repo?dataset_version_number=1) **100%|██████████| 138k/138k [00:00<00:00, 35.7MB/s]Extracting files...**

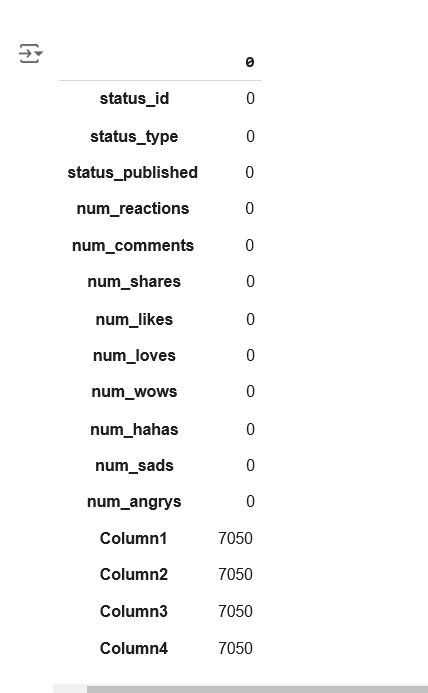
**df.shape**

** (7050, 16)**

**Next steps:**

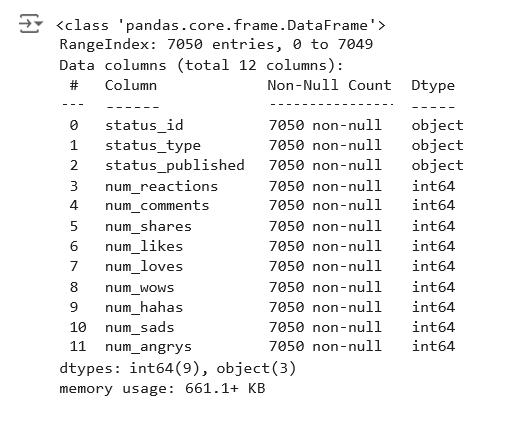
**df.info()**

****

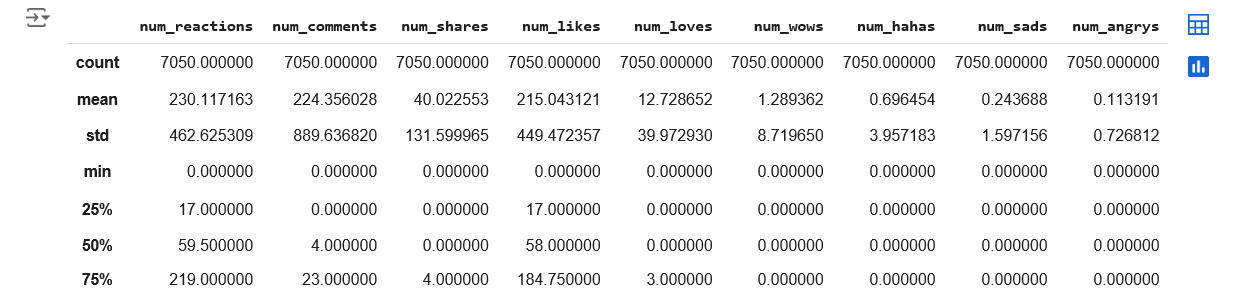
****

**df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)**

**df.info()**

****

**df.describe()**

**# view the labels in the variable df['status\_id'].unique()**

** array(['246675545449582\_1649696485147474',**

**'246675545449582\_1649426988507757',**

**'246675545449582\_1648730588577397', ..., '1050855161656896\_1060126464063099',**

**'1050855161656896\_1058663487542730',**

**'1050855161656896\_1050858841656528'], dtype=object)**

**# view how many different types of variables are there len(df['status\_id'].unique())**

** 6997**

**# view the labels in the variable df['status\_published'].unique()**

** array(['4/22/2018 6:00', '4/21/2018 22:45', '4/21/2018 6:17', ...,**

**'9/21/2016 23:03', '9/20/2016 0:43', '9/10/2016 10:30'],**

**dtype=object)**

**# view how many different types of variables are there len(df['status\_published'].unique())**

** 6913**

**# view the labels in the variable df['status\_type'].unique()**

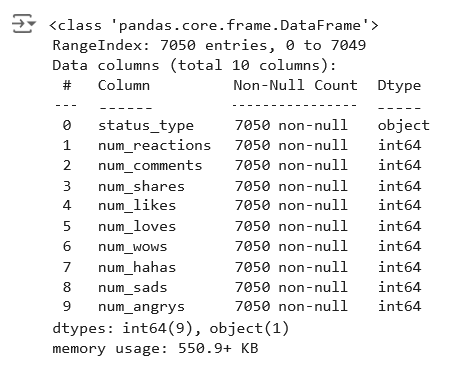
** array(['video', 'photo', 'link', 'status'], dtype=object)**

**# view how many different types of variables are there len(df['status\_type'].unique())**

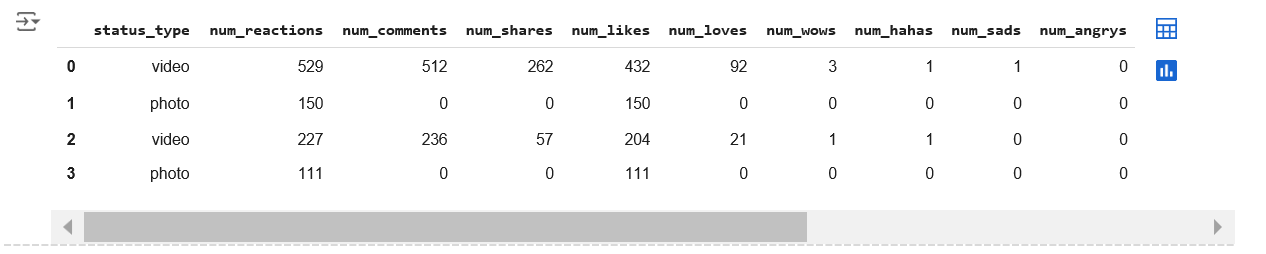
** 4**

**df.drop(['status\_id', 'status\_published'], axis=1, inplace=True)**

**df.info()**

****

**df.head()**

**Next steps:**

**X = df**

**y = df['status\_type']**

**from sklearn.preprocessing import LabelEncoder**

**le = LabelEncoder()**

**X['status\_type'] = le.fit\_transform(X['status\_type']) y = le.transform(y)**

**X.info()**

** <class 'pandas.core.frame.DataFrame'> RangeIndex: 7050 entries, 0 to 7049**

**Data columns (total 10 columns):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** |  | **Column** | **Non-Null Count** |  | **Dtype** |
| **0** |  | **status\_type** | **7050 non-null** |  | **int64** |
| **1** |  | **num\_reactions** | **7050 non-null** |  | **int64** |
| **2** |  | **num\_comments** | **7050 non-null** |  | **int64** |
| **3** |  | **num\_shares** | **7050 non-null** |  | **int64** |
| **4** |  | **num\_likes** | **7050 non-null** |  | **int64** |
| **5** |  | **num\_loves** | **7050 non-null** |  | **int64** |
| **6** |  | **num\_wows** | **7050 non-null** |  | **int64** |
| **7** |  | **num\_hahas** | **7050 non-null** |  | **int64** |
| **8** |  | **num\_sads** | **7050 non-null** |  | **int64** |
| **9** |  | **num\_angrys** | **7050 non-null** |  | **int64** |

**dtypes: int64(10)**

**memory usage: 550.9 KB**

Generate code with X

 View recommended plots

New interactive sheet

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X.head()** | **status\_type** | **num\_reactions** | **num\_comments** | **num\_shares** | **num\_likes** | **num\_loves** | **num\_wows** | **num\_hahas** | **num\_sads** | **num\_angrys** |
| **0** | **3** | **529** | **512** | **262** | **432** | **92** | **3** | **1** | **1** | **0** |
| **1** | **1** | **150** | **0** | **0** | **150** | **0** | **0** | **0** | **0** | **0** |
| **2** | **3** | **227** | **236** | **57** | **204** | **21** | **1** | **1** | **0** | **0** |
| **3** | **1** | **111** | **0** | **0** | **111** | **0** | **0** | **0** | **0** | **0** |

**Next steps:**

**cols = X.columns**

**from sklearn.preprocessing import MinMaxScaler ms = MinMaxScaler()**

**X = ms.fit\_transform(X)**

**X = pd.DataFrame(X, columns=[cols])**

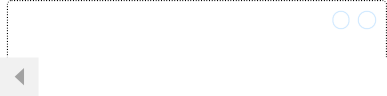
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X.head()** | **status\_type** | **num\_reactions** | **num\_comments** | **num\_shares** | **num\_likes** | **num\_loves** | **num\_wows** | **num\_hahas** | **num\_sads** | **num\_angrys** |
| **0** | **1.000000** | **0.112314** | **0.024393** | **0.076519** | **0.091720** | **0.140030** | **0.010791** | **0.006369** | **0.019608** | **0.0** |
| **1** | **0.333333** | **0.031847** | **0.000000** | **0.000000** | **0.031847** | **0.000000** | **0.000000** | **0.000000** | **0.000000** | **0.0** |
| **2** | **1.000000** | **0.048195** | **0.011243** | **0.016647** | **0.043312** | **0.031963** | **0.003597** | **0.006369** | **0.000000** | **0.0** |
| **3** | **0.333333** | **0.023567** | **0.000000** | **0.000000** | **0.023567** | **0.000000** | **0.000000** | **0.000000** | **0.000000** | **0.0** |

**Next steps:**

**from sklearn.cluster import KMeans**

**kmeans = KMeans(n\_clusters=2, random\_state=0) kmeans.fit(X)**

****



▾

KMeans

i [?](https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html)

**kmeans.cluster\_centers\_**

** array([[9.54921576e-01, 6.46330441e-02, 2.67028654e-02, 2.93171709e-02, 5.71231462e-02, 4.71007076e-02, 8.18581889e-03, 9.65207685e-03,**

**8.04219428e-03, 7.19501847e-03],**

**[3.28506857e-01, 3.90710874e-02, 7.54854864e-04, 7.53667113e-04, 3.85438884e-02, 2.17448568e-03, 2.43721364e-03, 1.20039760e-03,**

**2.75348016e-03, 1.45313276e-03]])**

**kmeans.inertia\_**

** 237.7572640441955**

**labels = kmeans.labels\_**

**# check how many of the samples were correctly labeled correct\_labels = sum(y == labels)**

**print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))**

** Result: 4288 out of 7050 samples were correctly labeled.**

**print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))**

** Accuracy score: 0.61**

**from sklearn.cluster import KMeans cs = []**

**for i in range(1, 11):**

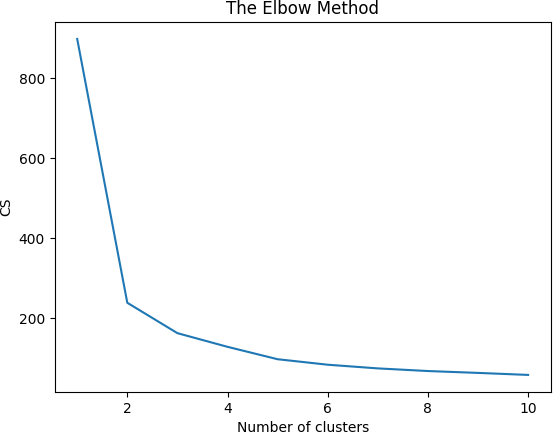
**kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0) kmeans.fit(X)**

**cs.append(kmeans.inertia\_) plt.plot(range(1, 11), cs)**

**plt.title('The Elbow Method')**

**plt.xlabel('Number of clusters') plt.ylabel('CS')**

**plt.show()**

****

**from sklearn.cluster import KMeans**

**kmeans = KMeans(n\_clusters=2,random\_state=0) kmeans.fit(X)**

**labels = kmeans.labels\_**

**# check how many of the samples were correctly labeled correct\_labels = sum(y == labels)**

**print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size)) print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))**

** Result: 4288 out of 7050 samples were correctly labeled.**

**Accuracy score: 0.61**

**kmeans = KMeans(n\_clusters=3, random\_state=0) kmeans.fit(X)**

**# check how many of the samples were correctly labeled labels = kmeans.labels\_**

**correct\_labels = sum(y == labels)**

**print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size)) print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))**

** Result: 4066 out of 7050 samples were correctly labeled.**

**Accuracy score: 0.58**

kmeans = KMeans(n\_clusters=4, random\_state=0)

**kmeans.fit(X)**

**# check how many of the samples were correctly labeled labels = kmeans.labels\_**

**correct\_labels = sum(y == labels)**

**print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size)) print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))**

** Result: 4112 out of 7050 samples were correctly labeled.**

**Accuracy score: 0.58**

**CNN**

**Experiment No.: 6**

**To study and implement CNN model**

Experiment No. 6

**Experiment 6: Convolution neural network (CNN)**

**Aim: Design and implement convolution neural network (CNN) to classify MINST images**

**Objectives:**

* To familiarize with Mathematical foundations for CNN, operator.
* To study the Applications of CNN.

**Outcomes:** The student will be able to,

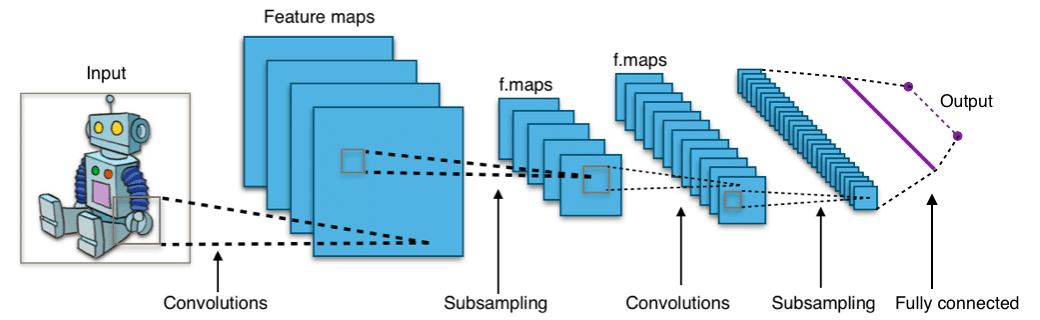
* Creating an understanding about the way the CNN is used and the domain of application.
* To appreciate the use of various GA operators in solving different types CNN.
* Match the industry requirements in the domains of Programming and Networking with the required management skills.

**Software Required: Python/MATLAB**

**Theory:**

Image classification is the task of assigning a label or class to an input image. It is a supervised learning problem, where a model is trained on a labelled dataset of images and their corresponding class labels, and then used to predict the class label of new, unseen images.

There are many architectures for image classification, one of the most popular being convolutional neural networks (CNNs). CNNs are especially effective at image classification because they are able to automatically learn the spatial hierarchies of features, such as edges, textures, and shapes, which are important for recognizing objects in images.



Conclusion: The solar radiation prediction dataset is optimized using the genetic algorithms which include all the genetic algorithm operators. Genetic algorithm includes the selection, crossover, mutation operators along with fitness function.

In a neural network, the layers comprise interconnected nodes or neurons that process the input data and pass it through the network to produce an output: process is as shown in figure.

**CNN for Image Classification: How It Works**

CNNs consist of a series of interconnected layers that process the input data. The first hidden layer of a CNN is usually a convolutional layer, which applies a set of filters to the input data to detect specific patterns. Each filter generates a feature map by sliding over the input data and performing element-wise multiplication with the entries in the filter. These feature maps are then combined and passed through non-linear activation functions, such as the ReLU function, which introduces non-linearities into the model and allows it to learn more complex patterns in the data.

Subsequent layers in a CNN may include additional convolutional layers, pooling layers, and fully-connected layers. Pooling layers reduce the size of the feature maps. This helps reduce the overall number of parameters in the model and makes it more computationally efficient. Fully-connected layers are typically found after convolutional and pooling layers of a CNN. A fully-connected layer connects all the neurons in a layer to all the neurons in the next layer, allowing the model to learn possible non-linear combinations of the features learned by the convolutional layers.

The final layer of a CNN is typically a SoftMax layer, which produces a probability distribution across the possible class labels for the input data. The class that has the highest probability is chosen as the prediction of the model.

* **Steps for CNN:**

Step 1: Choose a Dataset

Step 2: Prepare the Dataset for Training

Step 3: Create Training Data and Assign Labels

Step 4: Define and Train the CNN Model

Step 5: Test the Model’s Accuracy

**Conclusion:**

This experiment aims to detect respiratory diseases medical images for using CNN and it is detected successfully.

**Code Output**

# Libraries import os

import pathlib

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Model

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization, AveragePooling2D, Gl from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications.resnet import ResNet101

from tensorflow.keras.optimizers import Adam, Adamax

from tensorflow.keras.metrics import categorical\_crossentropy from tensorflow.keras import regularizers

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint from sklearn.metrics import confusion\_matrix, classification\_report

import cv2

import matplotlib.pyplot as plt import seaborn as sns

# Ignore Warnings import warnings

warnings.filterwarnings("ignore")

# Define the directory containing the COVID-19 radiography images import kagglehub

# Download latest version

path = kagglehub.dataset\_download("tawsifurrahman/covid19-radiography-database")

print("Path to dataset files:", path)

data\_dir = f'{path}/COVID-19\_Radiography\_Dataset'

# Initialize two empty lists to store the image file paths and their corresponding labels image\_paths = []

labels = []

# Traverse through all four folders of the dataset directory and map the files to their class names for folder in os.listdir(data\_dir):

folder\_path = os.path.join(data\_dir, folder)

if pathlib.Path(folder\_path).suffix != '': continue

if os.path.isdir(folder\_path):

file\_list = os.listdir(folder\_path)

for file in file\_list:

file\_path = os.path.join(folder\_path, file)

if os.path.isdir(file\_path):

if pathlib.Path(folder\_path).suffix == '':

# Ignore the mask folders since they are not used

if pathlib.Path(file\_path).parts[-1] in ['masks', 'Masks', 'MASKS']: continue

else:

original\_file\_list = os.listdir(file\_path)

for f in original\_file\_list:

img\_path = os.path.join(file\_path, f) image\_paths.append(img\_path)

labels.append(folder)

else:

image\_paths.append(file\_path) labels.append(folder)

# Store the image file names and their corresponding class names in a Pandas DataFrame data\_frame = pd.DataFrame({'image\_paths': image\_paths, 'labels': labels})

# Visualize the number of images per class

label\_counts = data\_frame.groupby('labels')['image\_paths'].count() ax = label\_counts.plot(kind='bar')

plt.xlabel('Disease')

plt.ylabel('Number of Images')

plt.title('Number of Images per Disease') plt.xticks(rotation=0)

for i in ax.patches:

ax.text(i.get\_x()+.15, i.get\_height()+.5, str(i.get\_height()), fontsize=10) plt.show()

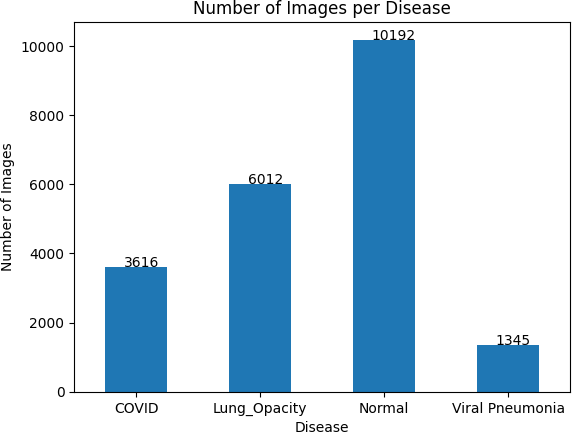
# Split the data into training, validation, and test sets at a ratio of 80-10-10 strat = data\_frame['labels']

train\_df, semi\_split\_df = train\_test\_split(data\_frame, train\_size=0.8, shuffle=True, random\_state=123, stratify=strat) strat = semi\_split\_df['labels']

valid\_df, test\_df = train\_test\_split(semi\_split\_df, train\_size=0.5, shuffle=True, random\_state=123, stratify=strat) print(f'Training set size = {len(train\_df)}, Validation set size = {len(valid\_df)}, Test set size = {len(test\_df)}')

 Downloading from [https://www.kaggle.com/api/v1/datasets/download/tawsifurrahman/covid19-radiography-database?dataset\_version\_number](https://www.kaggle.com/api/v1/datasets/download/tawsifurrahman/covid19-radiography-database?dataset_version_number=5) 100%|██████████| 778M/778M [00:04<00:00, 180MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/tawsifurrahman/covid19-radiography-database/versions/5



Training set size = 16932, Validation set size = 2116, Test set size = 2117

# Determine the length of the test dataset test\_length = len(test\_df)

# Find all factors of the test dataset length that are less than or equal to 80 factors = []

for n in range(1, test\_length + 1):

if test\_length % n == 0 and test\_length // n <= 80: factors.append(n)

# Select the largest factor as the test batch size

test\_batch\_size = max(sorted(factors, reverse=True))

# Calculate the number of steps per epoch for the test dataset test\_steps = test\_length // test\_batch\_size

# Define a function that returns the input image unchanged def identity(img):

return img

# Create image data generators for the training, validation, and test datasets

train\_augmentor = ImageDataGenerator(preprocessing\_function=identity, horizontal\_flip=True) test\_augmentor = ImageDataGenerator(preprocessing\_function=identity)

train\_gen = train\_augmentor.flow\_from\_dataframe( train\_df,

x\_col='image\_paths', y\_col='labels',

target\_size=(224, 224),

class\_mode='categorical', color\_mode='rgb',

shuffle=True, batch\_size=16

)

valid\_gen = test\_augmentor.flow\_from\_dataframe( valid\_df,

x\_col='image\_paths', y\_col='labels',

target\_size=(224, 224),

class\_mode='categorical', color\_mode='rgb',

shuffle=True, batch\_size=16

)

test\_gen = test\_augmentor.flow\_from\_dataframe( test\_df,

x\_col='image\_paths', y\_col='labels',

target\_size=(224, 224),

class\_mode='categorical', color\_mode='rgb',

shuffle=False,

batch\_size=test\_batch\_size

)

 Found 16932 validated image filenames belonging to 4 classes. Found 2116 validated image filenames belonging to 4 classes. Found 2117 validated image filenames belonging to 4 classes.

train\_data = train\_gen.class\_indices classes = list(train\_data.keys())

images, labels = next(train\_gen)

batch\_size = len(labels) # length of batch size

import matplotlib.pyplot as plt import numpy as np

# Get class names and number of classes

class\_names = list(train\_gen.class\_indices.keys()) num\_classes = len(class\_names)

# Create a dictionary to store images for each class

class\_images = {class\_name: [] for class\_name in class\_names}

# Loop over the samples until 5 images per class are found while True:

images, labels = next(train\_gen) for j in range(len(labels)):

class\_name = class\_names[np.argmax(labels[j])] if len(class\_images[class\_name]) < 5:

class\_images[class\_name].append(images[j])

# Stop once 5 images per class are found

if all(len(images) == 5 for images in class\_images.values()): break

# Create a figure with 5 rows and N columns

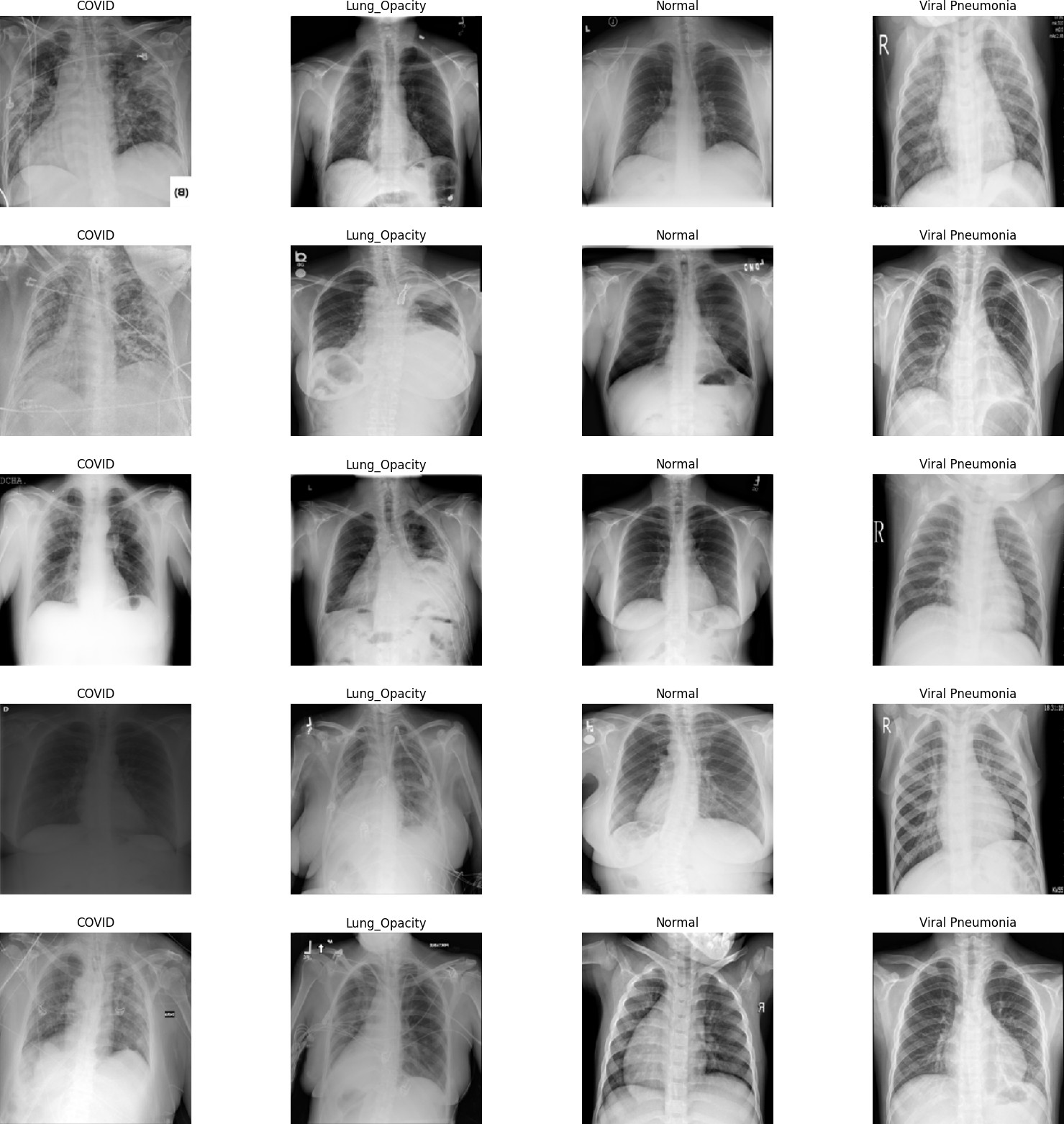
fig, axs = plt.subplots(5, num\_classes, figsize=(20, 20))

# Loop over the classes and plot the corresponding images for i in range(num\_classes):

for j in range(5):

axs[j, i].imshow(class\_images[class\_names[i]][j]/255) axs[j, i].set\_title(class\_names[i])

axs[j, i].axis('off') plt.show()

# Create Model Structure img\_size = (224, 224)

channels = 3

img\_shape = (img\_size[0], img\_size[1], channels)

class\_count = len(list(train\_gen.class\_indices.keys())) # to define number of classes in dense layer

# create pre-trained model (you can built on pretrained model such as : efficientnet, VGG , Resnet ) # we will use efficientnetb3 from EfficientNet family.

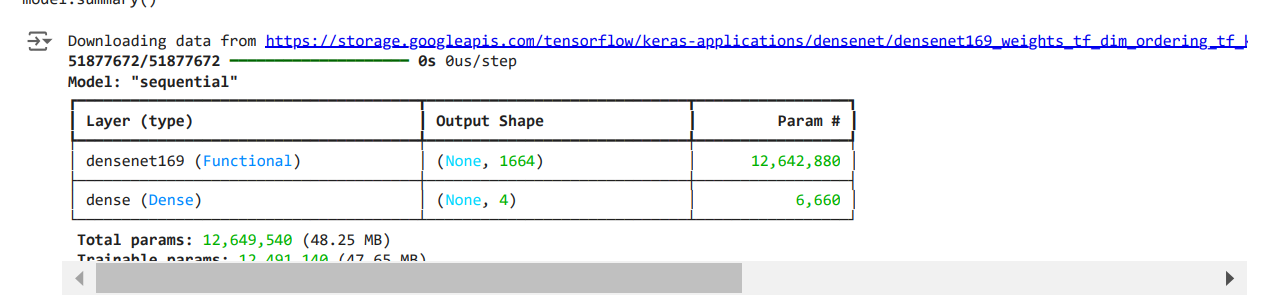
base\_model = tf.keras.applications.densenet.DenseNet169(include\_top= False, weights= "imagenet", input\_shape= img\_shape, pooling= 'max')

model = Sequential([ base\_model,

Dense(class\_count, activation= 'softmax')

])

model.summary()



# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy', metrics=['accuracy'])

# Define callbacks

early\_stop = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

# Train the model

history = model.fit( train\_gen,

epochs=20,

callbacks = early\_stop,

validation\_data=valid\_gen,)

 Epoch 1/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **674s** 412ms/step - accuracy: 0.6766 - loss: 1.2912 - val\_accuracy: 0.5997 - val\_loss: 48.7593 Epoch 2/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **235s** 222ms/step - accuracy: 0.8337 - loss: 0.4896 - val\_accuracy: 0.6352 - val\_loss: 0.9191

Epoch 3/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **236s** 222ms/step - accuracy: 0.8530 - loss: 0.4413 - val\_accuracy: 0.5520 - val\_loss: 2.4653 Epoch 4/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **258s** 219ms/step - accuracy: 0.8333 - loss: 0.4607 - val\_accuracy: 0.7500 - val\_loss: 0.7004 Epoch 5/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **232s** 219ms/step - accuracy: 0.8763 - loss: 0.3507 - val\_accuracy: 0.1706 - val\_loss: 50650.3203 Epoch 6/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **232s** 219ms/step - accuracy: 0.7693 - loss: 0.6691 - val\_accuracy: 0.8544 - val\_loss: 0.4625

Epoch 7/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **261s** 218ms/step - accuracy: 0.8500 - loss: 0.4206 - val\_accuracy: 0.7694 - val\_loss: 0.6297 Epoch 8/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **232s** 218ms/step - accuracy: 0.8366 - loss: 0.4693 - val\_accuracy: 0.6838 - val\_loss: 0.7400 Epoch 9/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **232s** 219ms/step - accuracy: 0.8595 - loss: 0.3967 - val\_accuracy: 0.8691 - val\_loss: 0.4733

Epoch 10/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **232s** 219ms/step - accuracy: 0.8924 - loss: 0.2982 - val\_accuracy: 0.7462 - val\_loss: 1.1401 Epoch 11/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **231s** 218ms/step - accuracy: 0.8950 - loss: 0.2928 - val\_accuracy: 0.9088 - val\_loss: 0.2673 Epoch 12/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **231s** 217ms/step - accuracy: 0.9087 - loss: 0.2626 - val\_accuracy: 0.6205 - val\_loss: 2.4918 Epoch 13/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **230s** 217ms/step - accuracy: 0.8982 - loss: 0.2999 - val\_accuracy: 0.8535 - val\_loss: 0.7360

Epoch 14/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **262s** 217ms/step - accuracy: 0.9042 - loss: 0.2784 - val\_accuracy: 0.8606 - val\_loss: 1.4583 Epoch 15/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **261s** 217ms/step - accuracy: 0.9072 - loss: 0.2538 - val\_accuracy: 0.7079 - val\_loss: 1.6525 Epoch 16/20

**1059/1059** ━━━━━━━━━━━━━━━━━━━━ **264s** 219ms/step - accuracy: 0.9051 - loss: 0.2816 - val\_accuracy: 0.8379 - val\_loss: 0.4433

history

 <keras.src.callbacks.history.History at 0x7d3b8cfdac80>

# Plot the training and validation accuracy over epochs acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy'] loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(len(acc)) #Change with the number of epochs

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy') plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

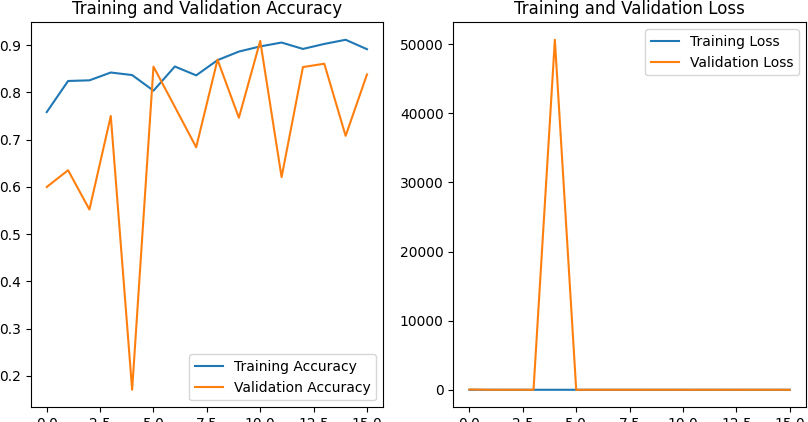
# Plot the training and validation loss over epochs plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss') plt.legend(loc='upper right')

plt.title('Training and Validation Loss') plt.show()





model.save('DenseNet\_Saved.h5')

 WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is

from tensorflow.keras.utils import plot\_model

plot\_model(model, to\_file='model.png', show\_shapes=True)

**Transfer Learning**

**Experiment No.: 7**

**To study and implement train model using Transfer learning**

Experiment No. 7

**Experiment 7: Design and implement Transfer learning model using supervise dataset**

**Aim: Implement transfer learning to classify medical images to detect respiratory diseases.**

**Objectives:**

* To familiarize with the concept of transfer learning.
* To study the applications of transfer learning to medical image classification.

**Outcomes:** The student will be,

* able to use the concept of transfer learning.
* aware of networks used in the image classification domain.
* able to be compatible with the industry requirements in the domain of deep learning.

**Software Required: Python/MATLAB**

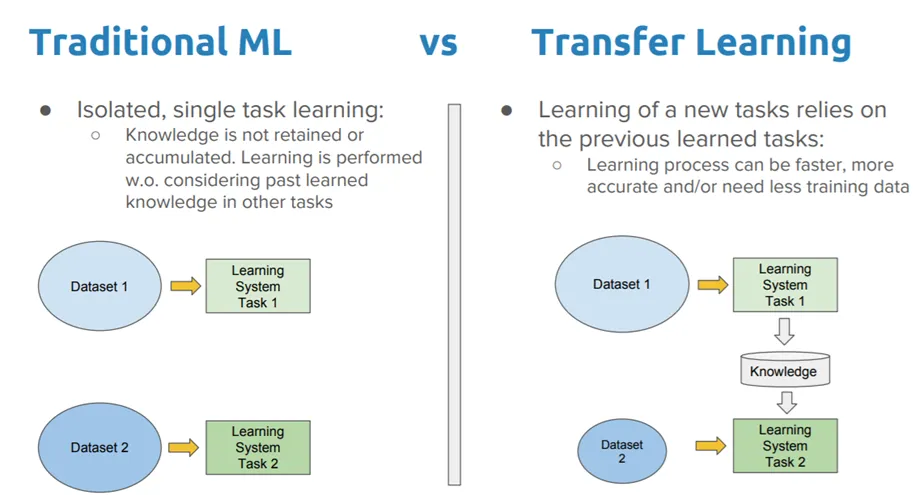
**Theory:**

Transfer learning is a technique in machine learning where a model trained on one task is used as the starting point for a model on a second task. This can be useful when the second task is similar to the first task, or when there is limited data available for the second task. By using the learned features from the first task as a starting point, the model can learn more quickly and effectively on the second task. This can also help to prevent overfitting, as the model will have already learned general features that are likely to be useful in the second task.

Why transfer learning?

Many deep neural networks trained on images have a curious phenomenon in common: in the early layers of the network, a deep learning model tries to learn a low level of features, like detecting edges, colours, variations of intensities, etc. Such features appear not to be specific to a particular dataset or a task because no matter what type of image we are processing either for detecting a lion or car. In both cases, we have to detect these low-level features. All these features occur regardless of the exact cost function or image dataset. Thus, learning these features in one task of detecting lions can be used in other tasks like detecting humans.

The following diagram shows comparison of traditional machine learning Vs transfer learning:



## 

## 

**Advantages of transfer learning:**

* Speed up the training process.
* Better performance.
* Handling small datasets.

**Disadvantages of transfer learning:**

* Domain mismatch.
* Overfitting.
* Complexity.

Image classification with pre-trained models is a popular technique in deep learning and computer vision that allows developers and researchers to leverage pre-existing, large-scale neural networks trained on vast datasets to classify new images.

**Algorithm:**

**Steps to use Transfer Learning:**

### 1. Select a pre-trained model

### 2. Choose layers to retrain

### 3. Prepare the data

### 4. Re-train the model

### 5. Fine-tune the model

### 6. Evaluate the model

**Conclusion**: In conclusion, transfer learning is a powerful technique in deep learning that allows users to leverage pre-trained models to accelerate the training process of a new model.

It can result in faster training, higher accuracy, and lower computational resources required.

However, some potential drawbacks include limited flexibility, overfitting, unintended biases, and limited generalization. Overall, transfer learning can be a valuable tool.

**Transfer Learning**

**Experiment No.: 8**

**To study and implement train model using Transfer learning**

Experiment No. 8

**Aim: Analyse solar radiation prediction dataset to optimize features using Genetic algorithm.**

**Objectives:**

* To familiarize with Mathematical foundations for Genetic algorithm, operator.
* To study the Applications of Genetic Algorithms.

**Outcomes:** The student will be able to,

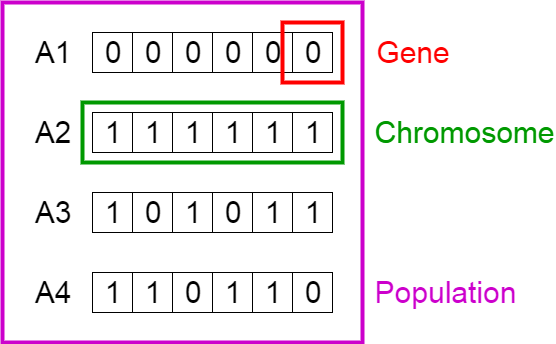
* Creating an understanding about the way the GA is used and the domain of application.
* To appreciate the use of various GA operators in solving different types of GA problems.
* Match the industry requirements in the domains of Programming and Networking with the required management skills.

**Software Required: Python/MATLAB**

**Theory:**

Genetic algorithm is a search technique used in computing to find true or approximate solutions to approximate solutions to optimization & search problems. Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved. Algorithms are started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

**Terminology for Genetic Algorithm**



**Population** contains a set of possible solutions for the stochastic search process to begin. GA will iterate over multiple generations till it finds an acceptable and optimized solution. First-generation is randomly generated.

**Chromosome** represents one candidate solution present in the generation or population. A chromosome is also referred to as a Genotype. A chromosome is composed of Genes that contain the value for the optimal variables.

**Phenotype** is the decoded parameter list for the genotype that is processed by the Genetic Algorithm. Mapping is applied to the genotype to convert to a phenotype.

The **Fitness function** or the objective function evaluates the individual solution or phenotypes for every generation to identify the fittest members.

**Outline of the Basic Genetic Algorithm**

1. [Start] Generate random population of n chromosomes (suitable solutions for the problem)

2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population

3. [New population] Create a new population by repeating following steps until the new population is complete

a. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

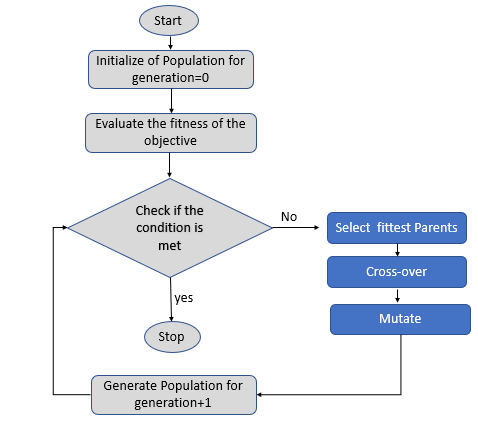
b. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

c. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).

4. [Accepting] Place new offspring in a new population 5. [Replace] Use new generated population for a further run of algorithm

6. [Test] If the end condition is satisfied, stop, and return the best solution in current population 7. [Loop] Go to step 2

**Flowchart:**



Conclusion: The solar radiation prediction dataset is optimized using the genetic algorithms which include all the genetic algorithm operators. Genetic algorithm includes the selection, crossover, mutation operators along with fitness function.

**Experiment No.: 8**

**Case Study**