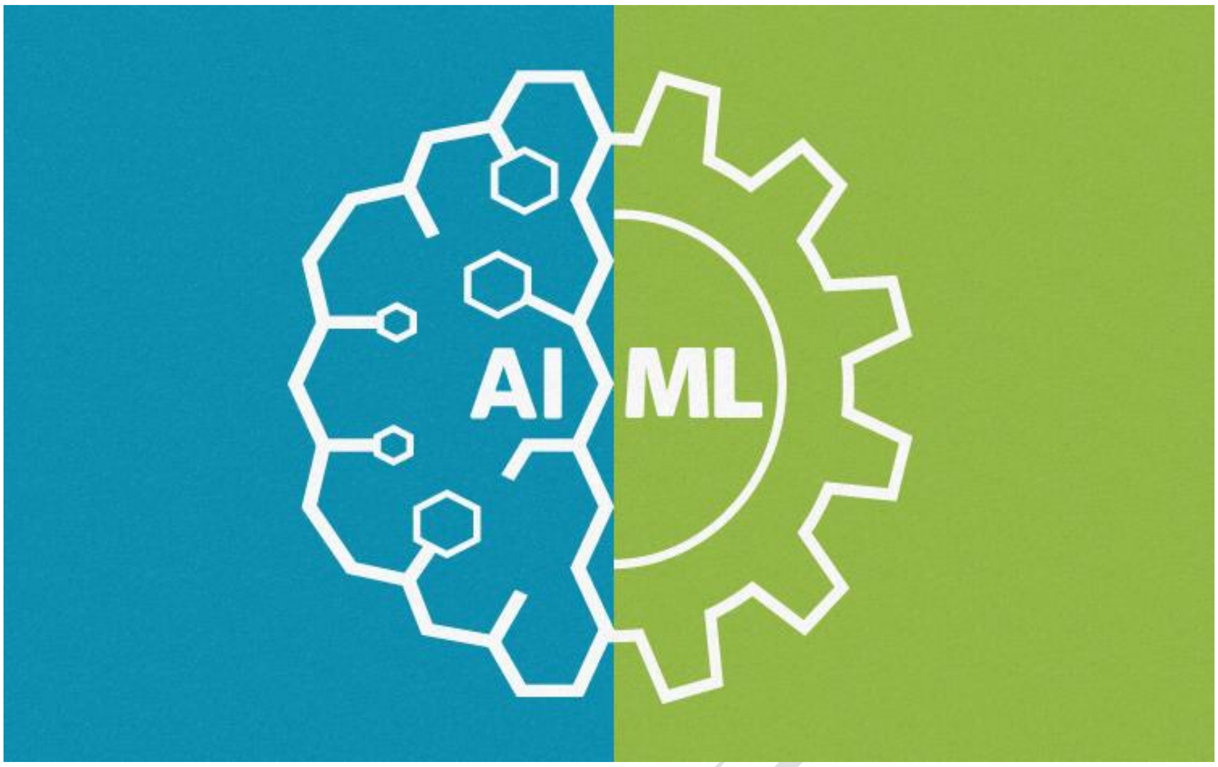


MACHINE LEARNING LAB MANUAL

A Hands-on Guide with 10 Practical Projects

Prepared by

ABHISHEK KUMAR

October 2024

Table of Contents

Practical 1

Data Preprocessing in Python

Objective:

- The objective of data preprocessing is to prepare and clean raw data to enhance its quality and usability for analysis, ensuring accurate and reliable results in subsequent data analysis or machine learning tasks.

Code:

- Import libraries & load the data

```
- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns
- import plotly.express as px
- df = pd.read_csv('/content/simulated_dataset.csv')
- print(df)
```

- Understand data structure & perform basic EDA

```
- df.describe()
- df.info()
- df.duplicated().sum()
- df.isnull().sum()
- df.fillna(df.mean(), inplace=True)
- df.isnull().sum()
```

- Remove outliers

```
- # Remove outliers using IQR method
- for column in df.select_dtypes(include=np.number).columns:
-     Q1 = df[column].quantile(0.25)
-     Q3 = df[column].quantile(0.75)
-     IQR = Q3 - Q1
-     lower_bound = Q1 - 1.5 * IQR
-     upper_bound = Q3 + 1.5 * IQR
-     df = df[(df[column] >= lower_bound) & (df[column] <=
  upper_bound)]
-
- print(df)
```

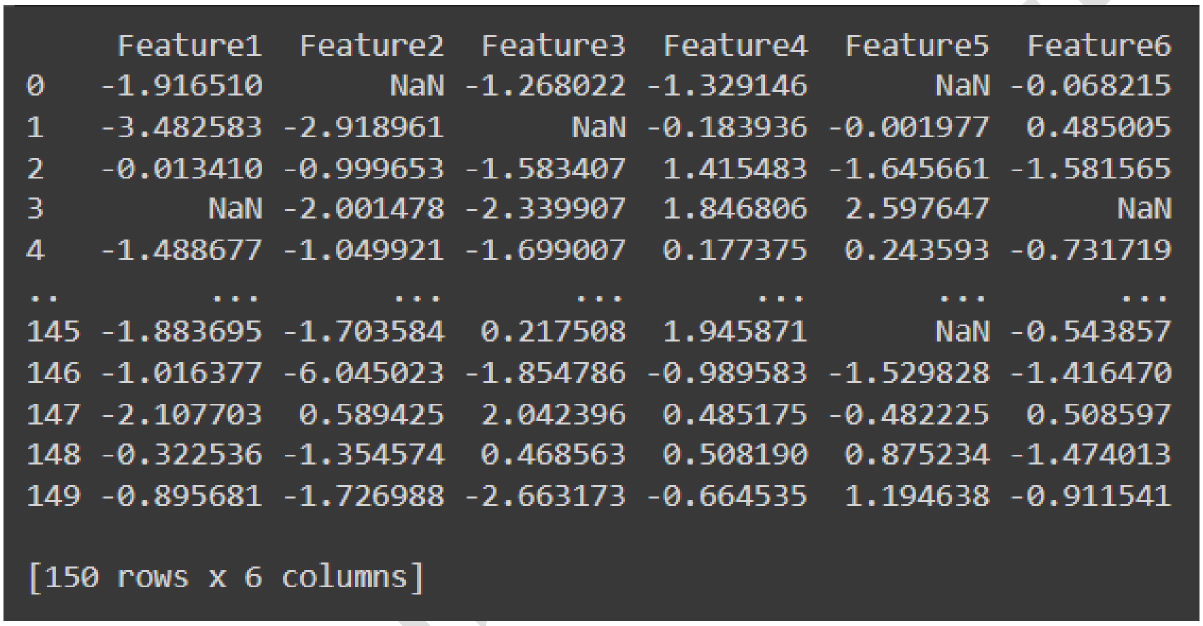
- Normalization operation

```
- from sklearn.preprocessing import MinMaxScaler
-
- # Normalize the dataset using MinMaxScaler
- scaler = MinMaxScaler()
- numerical_features = df.select_dtypes(include=np.number).columns
- df[numerical_features] =
  scaler.fit_transform(df[numerical_features])
-
- print(df)
```

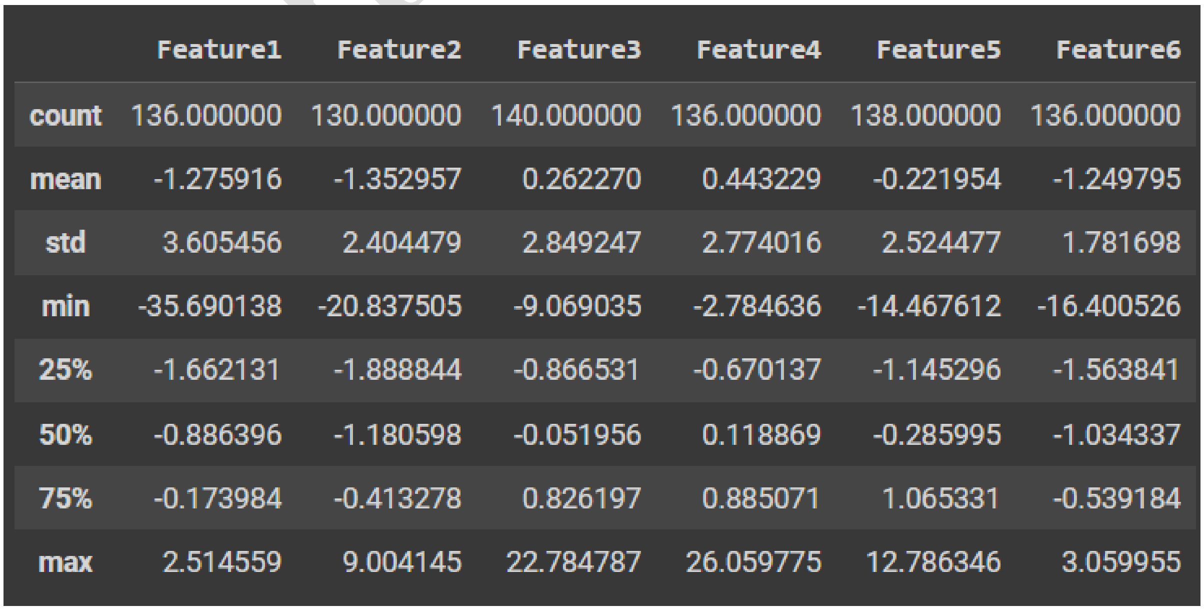
- Standardization operation

```
-   from sklearn.preprocessing import MinMaxScaler, StandardScaler
-
-   # Standardize the dataset using StandardScaler
-   scaler = StandardScaler()
-   numerical_features = df.select_dtypes(include=np.number).columns
-   df[numerical_features] =
    scaler.fit_transform(df[numerical_features])
-
-   print(df)
```

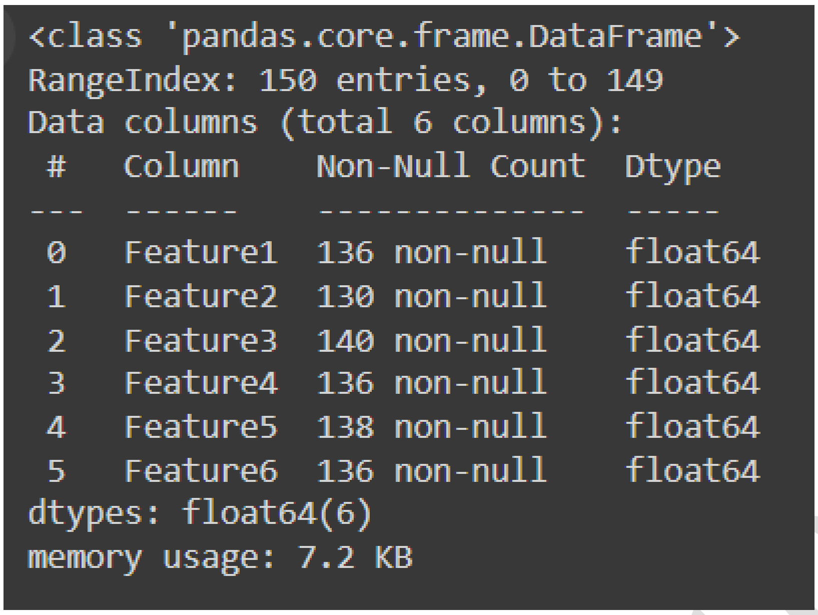
Result:



```
     Feature1  Feature2  Feature3  Feature4  Feature5  Feature6
0   -1.916510       NaN -1.268022 -1.329146       NaN -0.068215
1   -3.482583 -2.918961       NaN -0.183936 -0.001977  0.485005
2   -0.013410 -0.999653 -1.583407  1.415483 -1.645661 -1.581565
3         NaN -2.001478 -2.339907  1.846806  2.597647       NaN
4   -1.488677 -1.049921 -1.699007  0.177375  0.243593 -0.731719
..        ...       ...       ...       ...       ...       ...
145 -1.883695 -1.703584  0.217508  1.945871       NaN -0.543857
146 -1.016377 -6.045023 -1.854786 -0.989583 -1.529828 -1.416470
147 -2.107703  0.589425  2.042396  0.485175 -0.482225  0.508597
148 -0.322536 -1.354574  0.468563  0.508190  0.875234 -1.474013
149 -0.895681 -1.726988 -2.663173 -0.664535  1.194638 -0.911541

[150 rows x 6 columns]
```

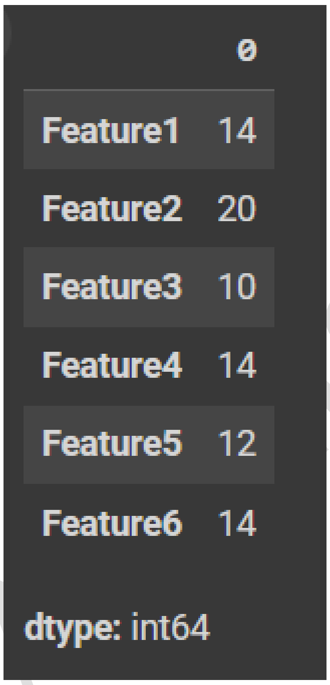


	Feature1	Feature2	Feature3	Feature4	Feature5	Feature6
count	136.000000	130.000000	140.000000	136.000000	138.000000	136.000000
mean	-1.275916	-1.352957	0.262270	0.443229	-0.221954	-1.249795
std	3.605456	2.404479	2.849247	2.774016	2.524477	1.781698
min	-35.690138	-20.837505	-9.069035	-2.784636	-14.467612	-16.400526
25%	-1.662131	-1.888844	-0.866531	-0.670137	-1.145296	-1.563841
50%	-0.886396	-1.180598	-0.051956	0.118869	-0.285995	-1.034337
75%	-0.173984	-0.413278	0.826197	0.885071	1.065331	-0.539184
max	2.514559	9.004145	22.784787	26.059775	12.786346	3.059955

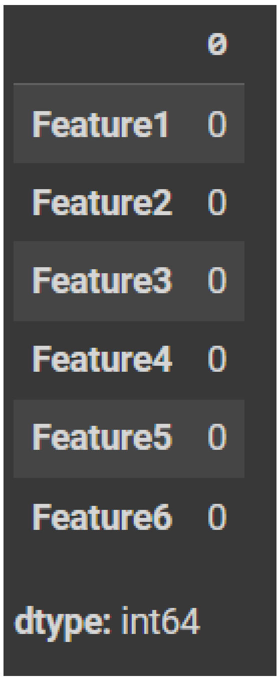


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Feature1  136 non-null    float64
 1   Feature2  130 non-null    float64
 2   Feature3  140 non-null    float64
 3   Feature4  136 non-null    float64
 4   Feature5  138 non-null    float64
 5   Feature6  136 non-null    float64
dtypes: float64(6)
memory usage: 7.2 KB
```

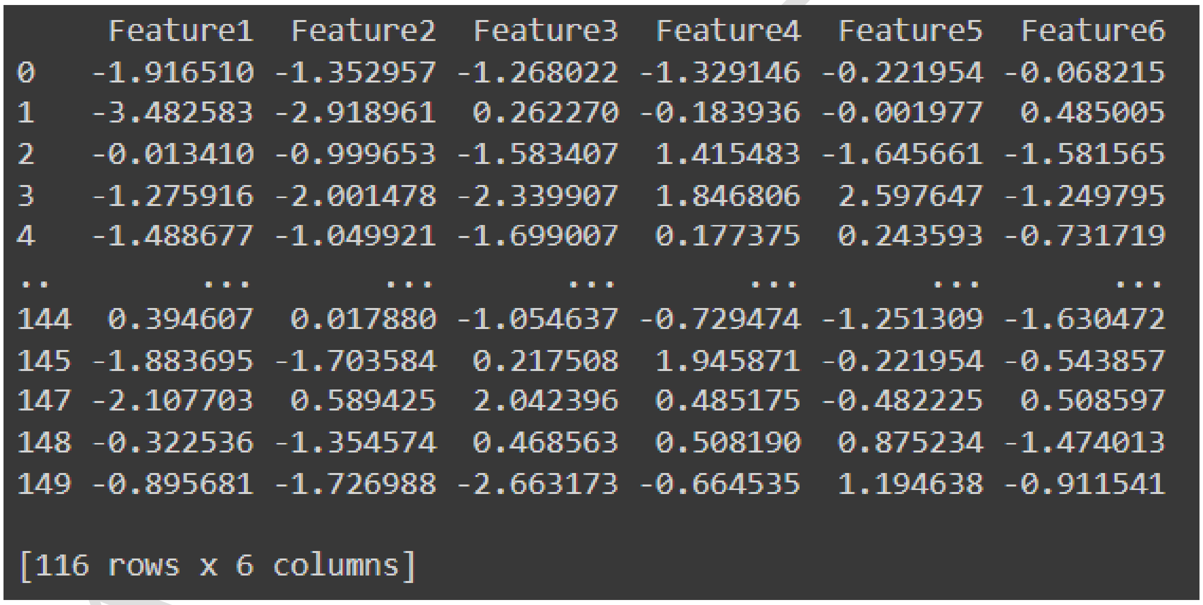
```
0
```



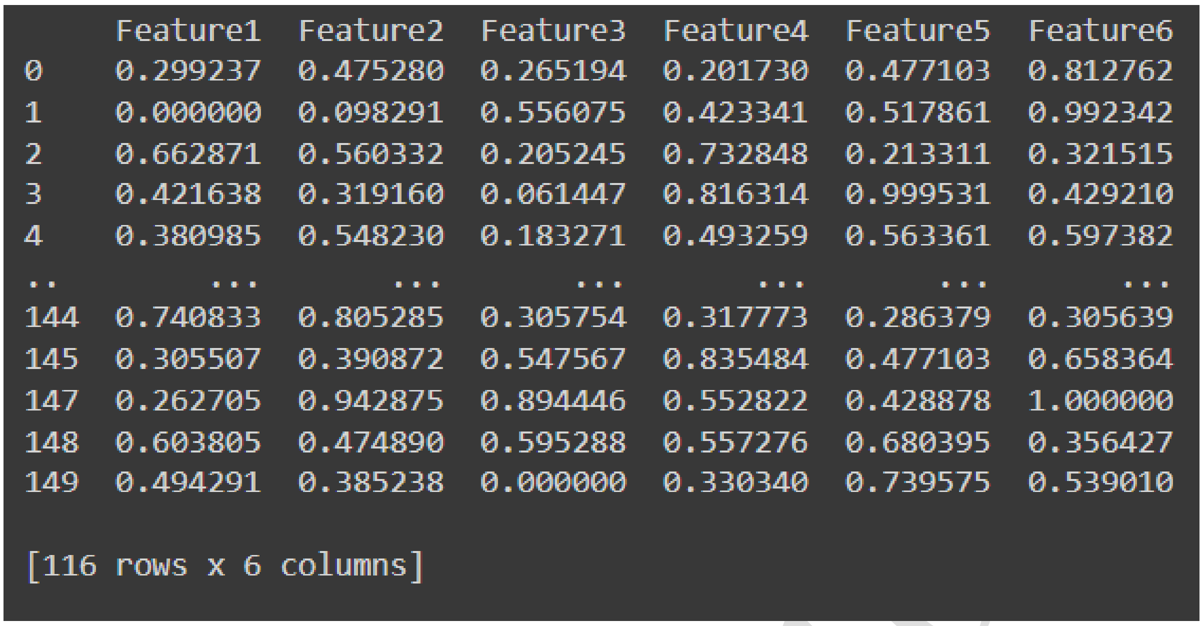
```
          0
Feature1  14
Feature2  20
Feature3  10
Feature4  14
Feature5  12
Feature6  14

dtype: int64
```



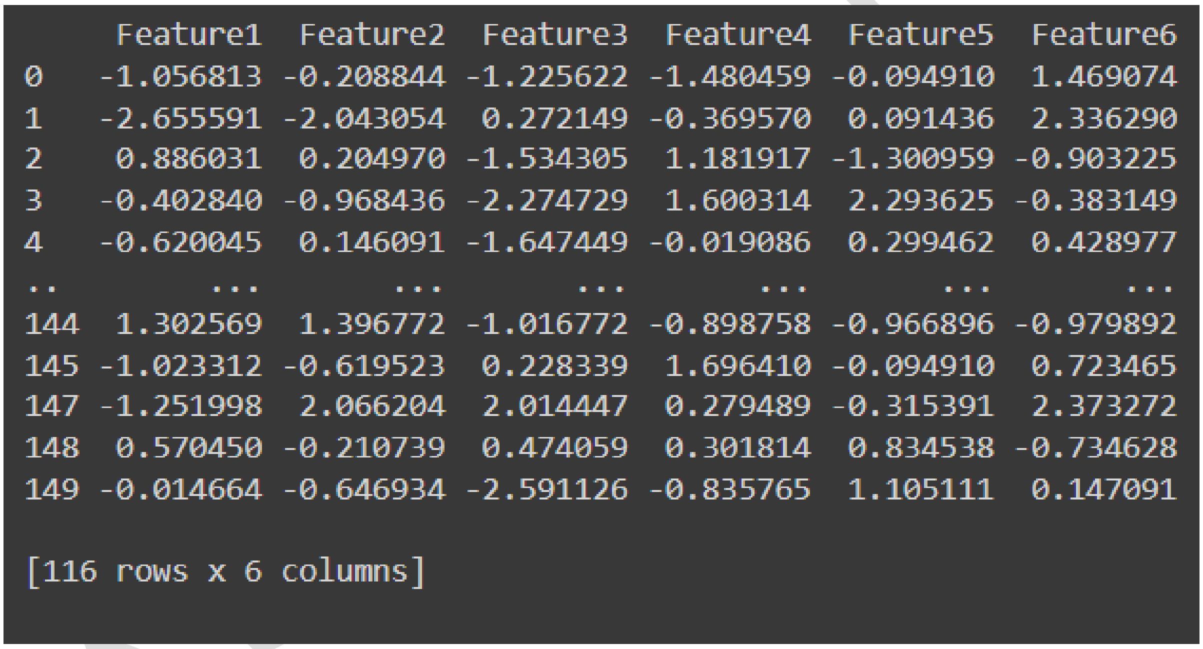
```
          0
Feature1  0
Feature2  0
Feature3  0
Feature4  0
Feature5  0
Feature6  0

dtype: int64
```



```
     Feature1  Feature2  Feature3  Feature4  Feature5  Feature6
0   -1.916510 -1.352957 -1.268022 -1.329146 -0.221954 -0.068215
1   -3.482583 -2.918961  0.262270 -0.183936 -0.001977  0.485005
2   -0.013410 -0.999653 -1.583407  1.415483 -1.645661 -1.581565
3   -1.275916 -2.001478 -2.339907  1.846806  2.597647 -1.249795
4   -1.488677 -1.049921 -1.699007  0.177375  0.243593 -0.731719
..        ...       ...       ...       ...       ...       ...
144  0.394607  0.017880 -1.054637 -0.729474 -1.251309 -1.630472
145 -1.883695 -1.703584  0.217508  1.945871 -0.221954 -0.543857
147 -2.107703  0.589425  2.042396  0.485175 -0.482225  0.508597
148 -0.322536 -1.354574  0.468563  0.508190  0.875234 -1.474013
149 -0.895681 -1.726988 -2.663173 -0.664535  1.194638 -0.911541

[116 rows x 6 columns]
```



```
     Feature1  Feature2  Feature3  Feature4  Feature5  Feature6
0    0.299237  0.475280  0.265194  0.201730  0.477103  0.812762
1    0.000000  0.098291  0.556075  0.423341  0.517861  0.992342
2    0.662871  0.560332  0.205245  0.732848  0.213311  0.321515
3    0.421638  0.319160  0.061447  0.816314  0.999531  0.429210
4    0.380985  0.548230  0.183271  0.493259  0.563361  0.597382
..        ...       ...       ...       ...       ...       ...
144  0.740833  0.805285  0.305754  0.317773  0.286379  0.305639
145  0.305507  0.390872  0.547567  0.835484  0.477103  0.658364
147  0.262705  0.942875  0.894446  0.552822  0.428878  1.000000
148  0.603805  0.474890  0.595288  0.557276  0.680395  0.356427
149  0.494291  0.385238  0.000000  0.330340  0.739575  0.539010

[116 rows x 6 columns]
```



```
     Feature1  Feature2  Feature3  Feature4  Feature5  Feature6
0   -1.056813 -0.208844 -1.225622 -1.480459 -0.094910  1.469074
1   -2.655591 -2.043054  0.272149 -0.369570  0.091436  2.336290
2    0.886031  0.204970 -1.534305  1.181917 -1.300959 -0.903225
3   -0.402840 -0.968436 -2.274729  1.600314  2.293625 -0.383149
4   -0.620045  0.146091 -1.647449 -0.019086  0.299462  0.428977
..        ...       ...       ...       ...       ...       ...
144  1.302569  1.396772 -1.016772 -0.898758 -0.966896 -0.979892
145 -1.023312 -0.619523  0.228339  1.696410 -0.094910  0.723465
147 -1.251998  2.066204  2.014447  0.279489 -0.315391  2.373272
148  0.570450 -0.210739  0.474059  0.301814  0.834538 -0.734628
149 -0.014664 -0.646934 -2.591126 -0.835765  1.105111  0.147091

[116 rows x 6 columns]
```

Conclusion:

- In this task, null value imputation, outlier removal, and data standardization and normalization were applied to ensure the dataset's integrity and suitability for machine learning models. Imputing missing values addressed gaps in the dataset, allowing for a complete analysis without losing valuable data points. Outlier removal ensured that extreme values, which could distort model performance, were appropriately handled. Standardization and normalization transformed the features into comparable scales, enhancing model convergence and performance. Together, these preprocessing steps are crucial for improving the accuracy and reliability of machine learning models, ensuring robust predictions and insightful analysis from the data.

Practical 2

Statistical Distribution

Objective:

- To analyze statistical distribution using Kernel Density Estimation (KDE), Gaussian distribution, and Q-Q plots.

Code:

- Import libraries & create a synthetic dataset

```
-  import pandas as pd
-  import numpy as np
-  import matplotlib.pyplot as plt
-  import seaborn as sns
-  import statsmodels.api as sm
-  np.random.seed(42)
-  data = np.random.normal(loc=0, scale=1, size=1000)
-  df = pd.DataFrame(data, columns=['Value'])
-  print(df)
```

- Mean & standard deviation of dataset

```
-  mean = df['Value'].mean()
-  std_dev = df['Value'].std()
-
-  print(f"Mean: {mean}")
-  print(f"Standard Deviation: {std_dev}")
```

- Plot Gaussian distribution

```
-  sns.histplot(df['Value'])
-  plt.title('Gaussian Distribution')
-  plt.xlabel('Value')
-  plt.ylabel('Frequency')
-  plt.show()
```

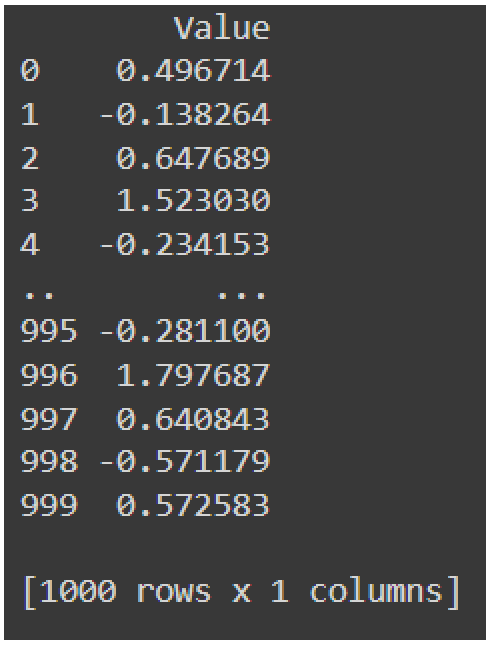
- Plot KDE

```
-  sns.kdeplot(df['Value'])
-  plt.title('Kernel Density Estimation')
-  plt.xlabel('Value')
-  plt.ylabel('Density')
-  plt.show()
```

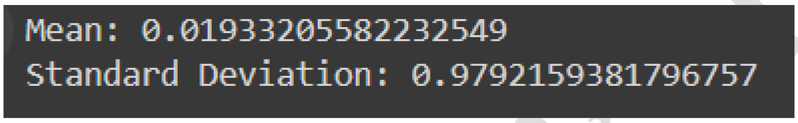
- Plot QQ chart

```
-  sm.qqplot(df['Value'], line='s')
-  plt.title('QQ Plot')
-  plt.show()
```

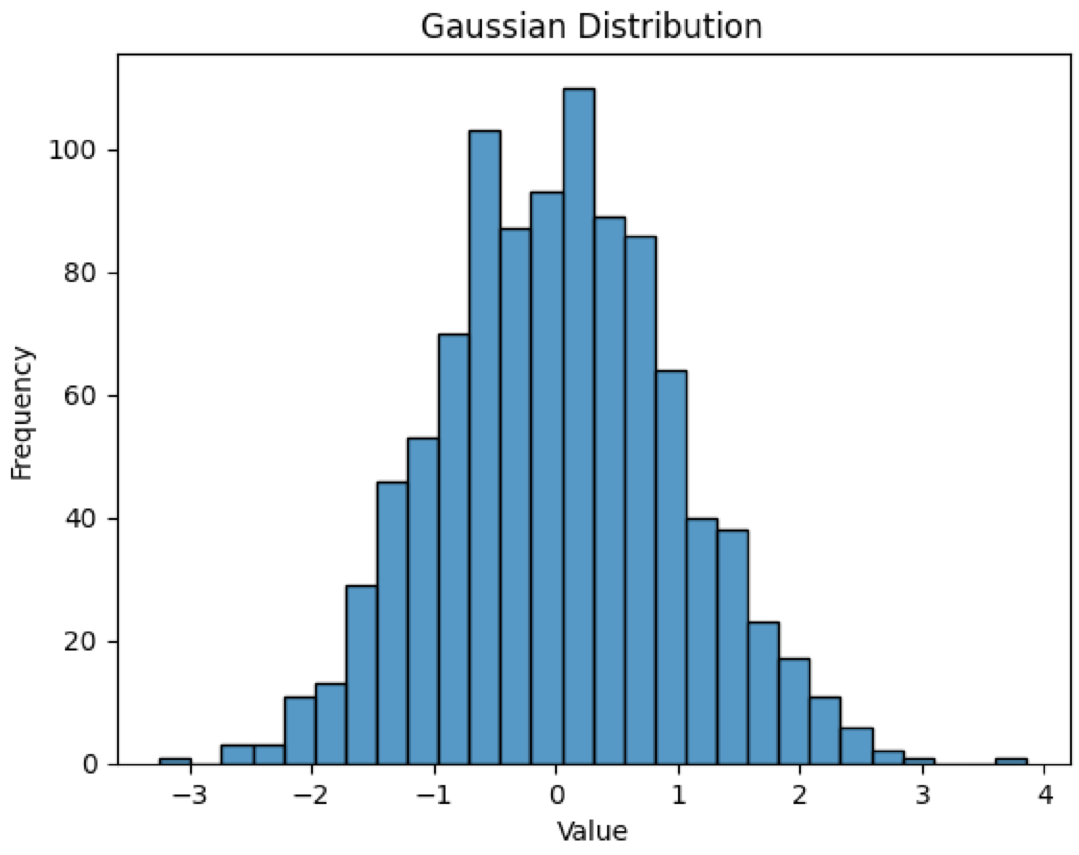
Result:

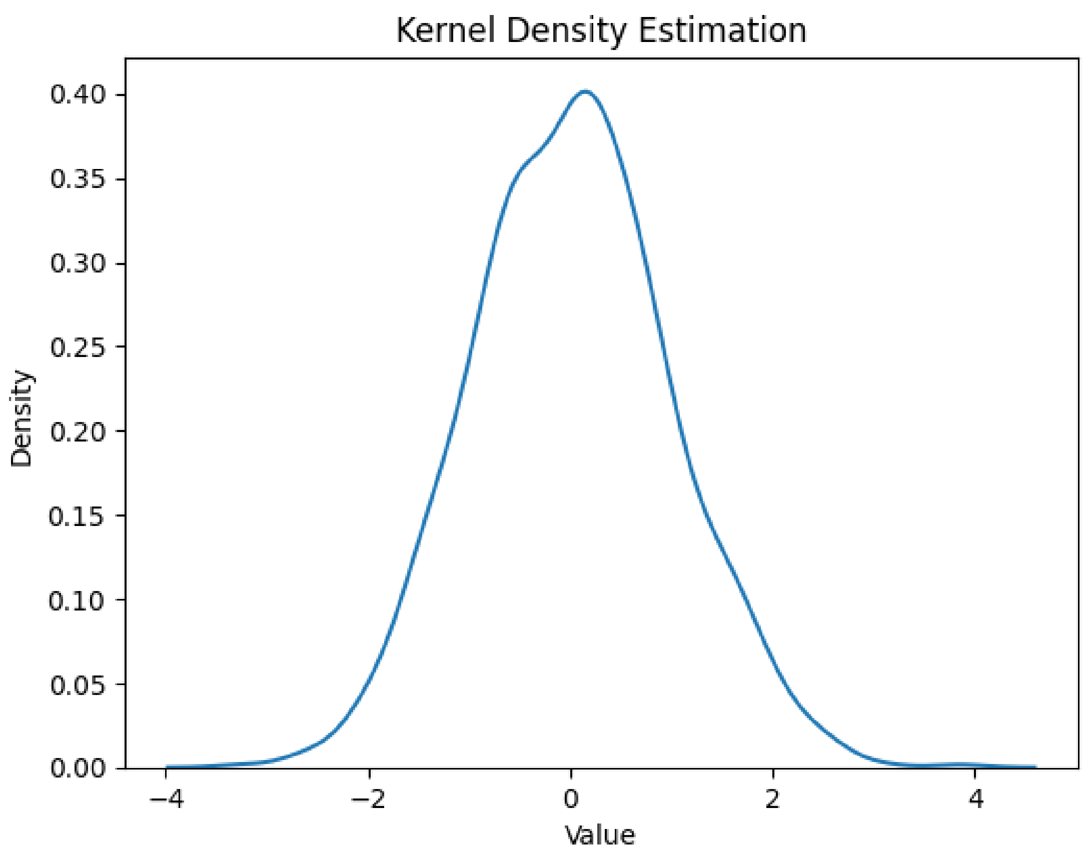


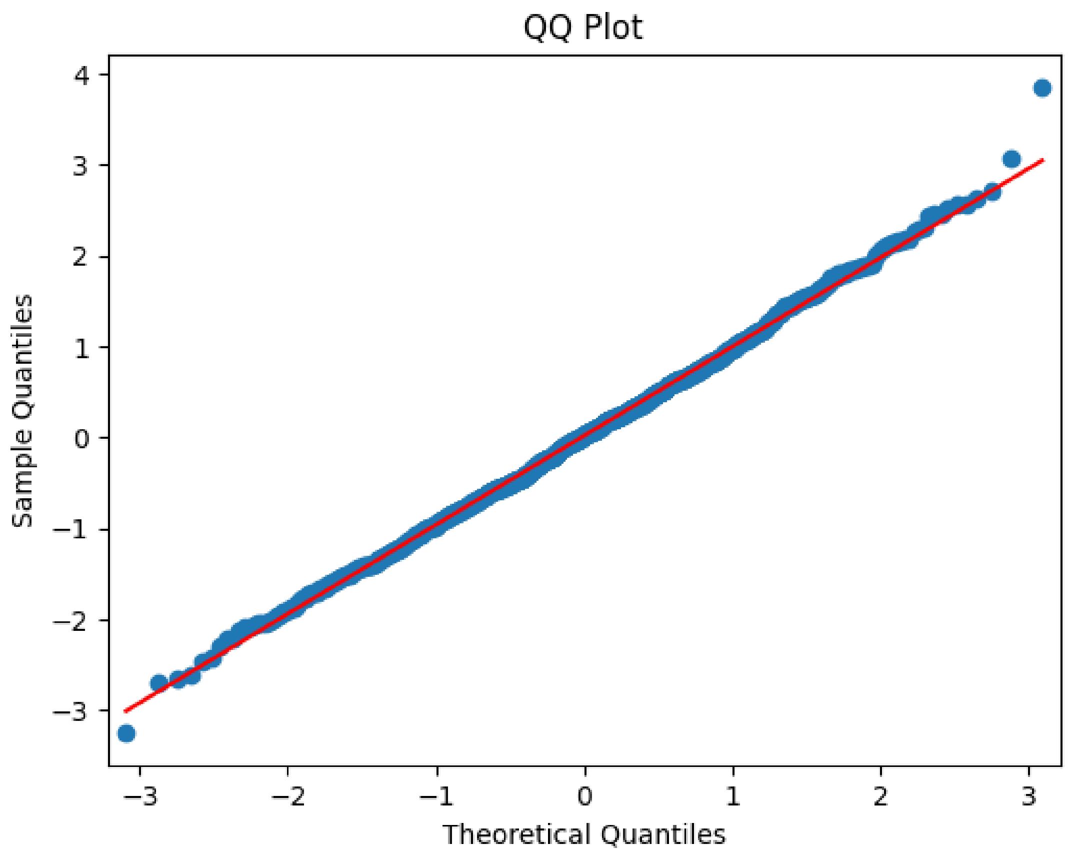
```
         Value
0     0.496714
1    -0.138264
2     0.647689
3     1.523030
4    -0.234153
..         ...
995 -0.281100
996  1.797687
997  0.640843
998 -0.571179
999  0.572583

[1000 rows x 1 columns]
```



```
Mean: 0.01933205582232549
Standard Deviation: 0.9792159381796757
```




Kernel Density Estimation
Density
Value
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4


QQ Plot
Sample Quantiles
Theoretical Quantiles
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

Conclusion:

- In this task, Gaussian distribution analysis was performed alongside the creation of Q-Q plots and Kernel Density Estimation (KDE) on a synthetically generated dataset. The analysis aimed to assess the normality of the data distribution, providing insights into its underlying characteristics. The Q-Q plot visually compares the quantiles of the dataset to the quantiles of a standard normal distribution, allowing for a straightforward assessment of normality. Meanwhile, the KDE offered a smooth estimate of the probability density function, facilitating a deeper understanding of the data distribution's shape. Together, these techniques highlight the importance of statistical analysis in understanding data behaviour and validating assumptions for subsequent modelling tasks, ensuring that the chosen algorithms are appropriately aligned with the data's properties.

Practical 3

Logistics Regression

Objective:

- The objective of this practical is to implement Logistic Regression to model the relationship between a binary dependent variable and one or more independent variables.

Code:

- Import libraries & load dataset

```
-   import warnings
-   warnings.simplefilter(action='ignore', category=FutureWarning)
-   from sklearn.datasets import load_breast_cancer
-   data = load_breast_cancer()
```

- Data description

```
-   print(data.DESCR)
```

- Tain & test split

```
-   from sklearn.model_selection import train_test_split
-
-   X_train, X_test, y_train, y_test = train_test_split(
-       data.data, data.target, test_size=0.2, random_state=42
-   )
```

- Model training and evaluation

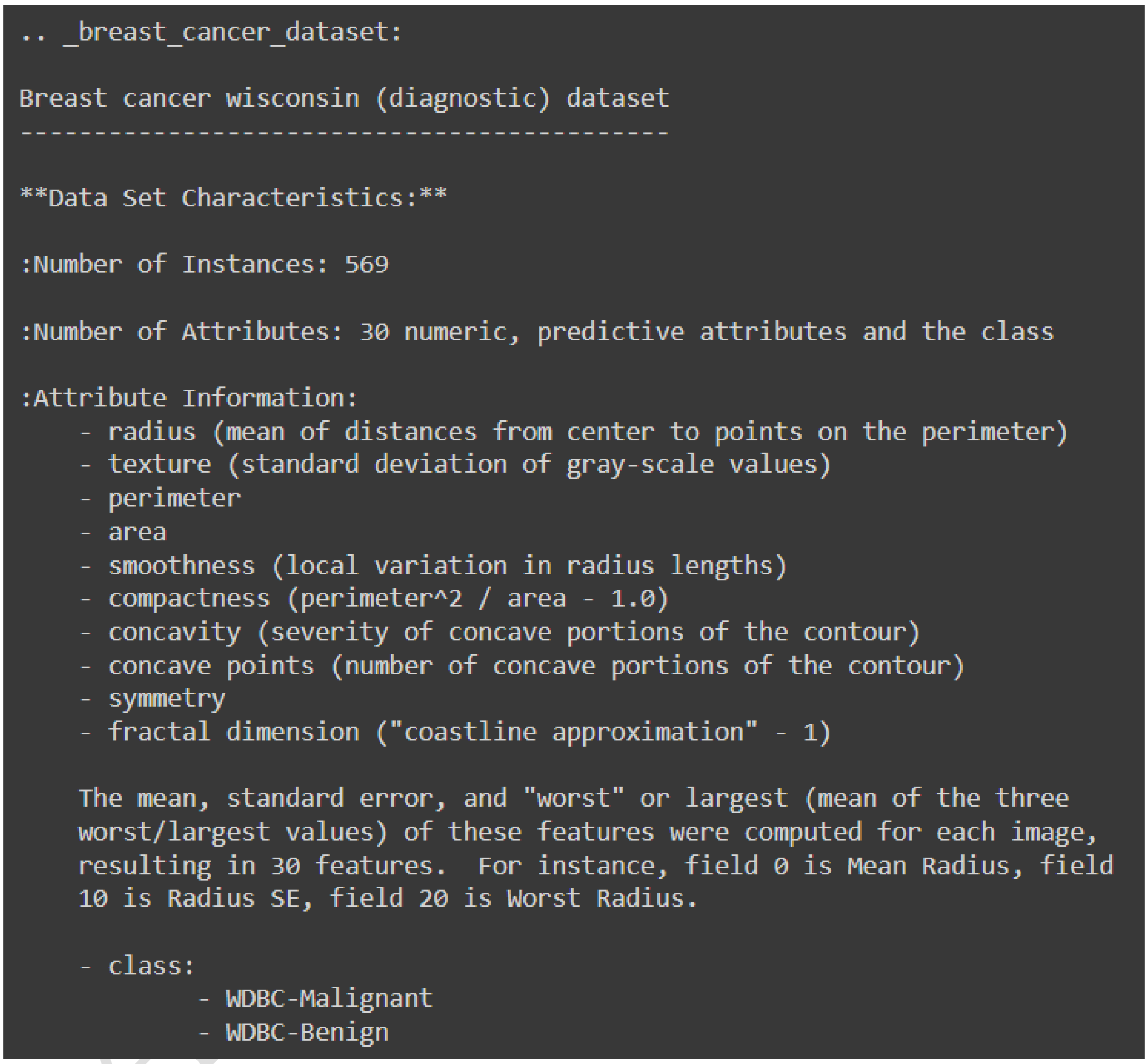
```
-   from sklearn.linear_model import LogisticRegression
-   from sklearn.metrics import classification_report, accuracy_score
-
-   # Initialize and train the Logistic Regression model
-   model = LogisticRegression(tol=0.001,max_iter=10000)   # Increase
    max_iter if needed
-   model.fit(X_train, y_train)
-
-   # Make predictions on the test set
-   y_pred = model.predict(X_test)
-
-   # Evaluate the model
-   print(classification_report(y_test, y_pred))
-   print("Accuracy:", accuracy_score(y_test, y_pred))
```

- Train model with L2 regularization

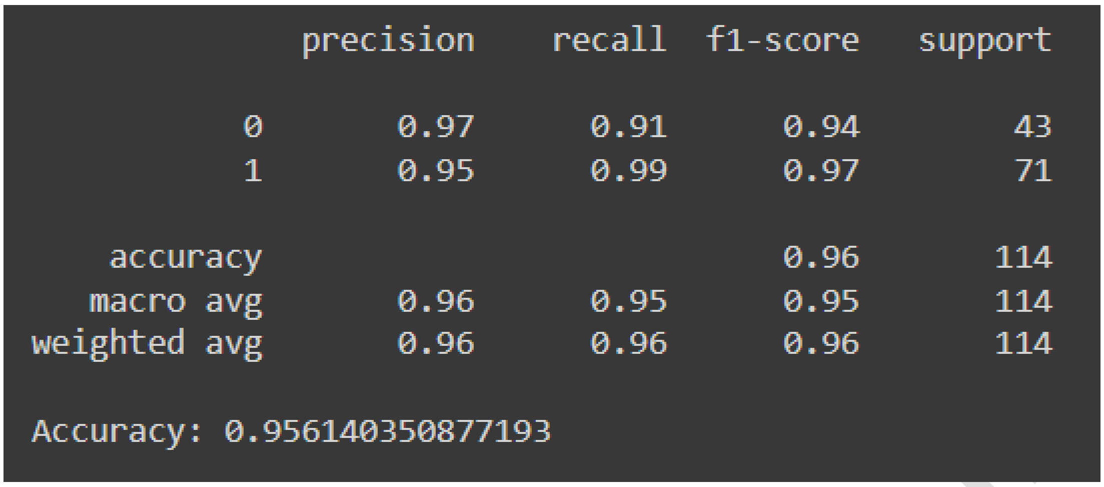
```
-   # Initialize and train the Logistic Regression model with L2
    regularization
-   model = LogisticRegression(penalty='l2', C=1.0, tol=0.001,
    max_iter=10000)  # C is the inverse of regularization strength
-   model.fit(X_train, y_train)
-
-   # Make predictions on the test set
-   y_pred = model.predict(X_test)
-
```

```
-   # Evaluate the model
-   print(classification_report(y_test, y_pred))
-   print("Accuracy:", accuracy_score(y_test, y_pred))
```

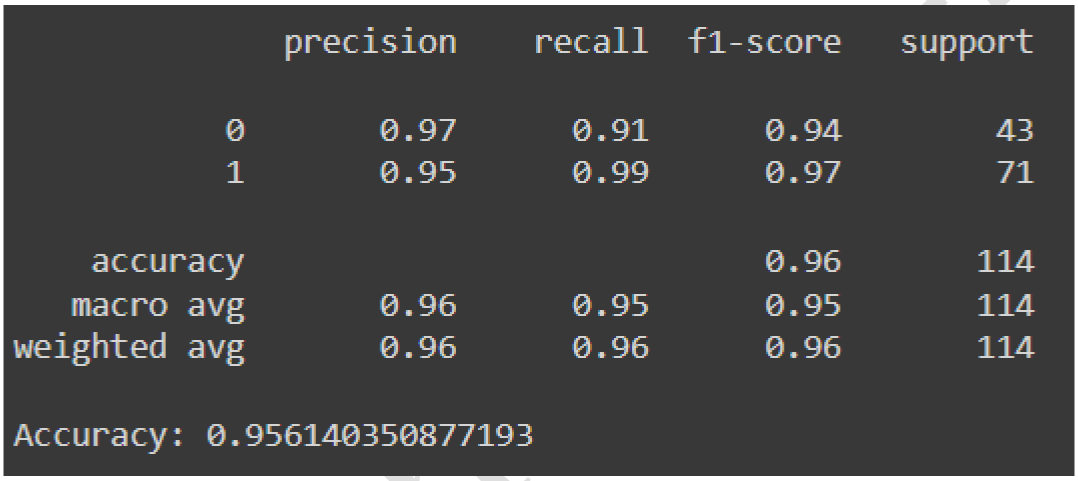
Result:



```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:
    - radius (mean of distances from center to points on the perimeter)
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features.  For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign
```



```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Accuracy: 0.956140350877193
```



```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Accuracy: 0.956140350877193
```

Conclusion:

- In this task, logistic regression was applied to the breast cancer dataset to predict the likelihood of cancerous outcomes. The model's performance was evaluated using key classification metrics such as accuracy, precision, recall, and the F1-score, which provided a comprehensive understanding of the model's effectiveness. To further optimize the model and prevent overfitting, L2 regularization (Ridge) was used, penalizing large coefficient values and improving generalization. This process underscores the importance of evaluating and fine-tuning machine learning models to enhance prediction accuracy, particularly in sensitive applications like medical diagnosis, where model reliability is critical.

Practical 4

K-Nearest Neighbours

Objective:

- The objective of this practical is to implement the K-Nearest Neighbours (KNN) algorithm and evaluate its performance using different distance metrics, such as Euclidean, Manhattan, and Minkowski.

Code:

- Import libraries & load the dataset

```
- import pandas as pd
- import numpy as np
- import plotly.express as px
- from sklearn.neighbors import KNeighborsClassifier
- from sklearn.datasets import load_wine
- from sklearn.model_selection import train_test_split
- from sklearn.metrics import accuracy_score
- wine = load_wine()
- X = wine.data
- y = wine.target
- X_train, X_test, y_train, y_test = train_test_split(X, y,
  test_size=0.3,random_state=42)
```

- Train & test split

```
- wine = load_wine()
- X = wine.data
- y = wine.target
- X_train, X_test, y_train, y_test = train_test_split(X, y,
  test_size=0.3,random_state=42)
```

- List of distance parameters

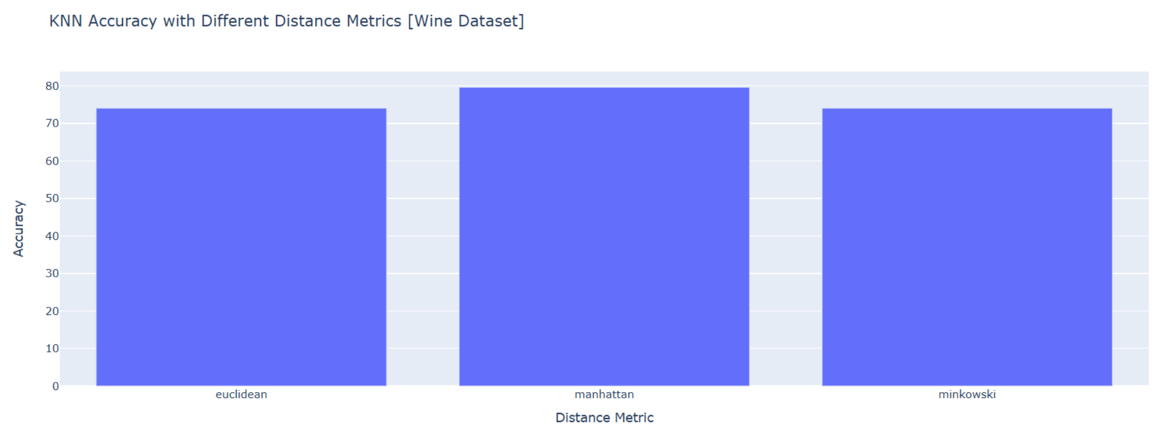
```
- distance_metrics = {
-   'euclidean': 'euclidean',
-   'manhattan': 'manhattan',
-   'minkowski': 'minkowski'
- }
```

- Model training & evaluation

```
- accuracy_scores = []
- for metric in distance_metrics.values():
-   knn = KNeighborsClassifier(n_neighbors=3, metric=metric)
-   knn.fit(X_train, y_train)
-   y_pred = knn.predict(X_test)
-   accuracy = accuracy_score(y_test, y_pred)
-   accuracy_scores.append(accuracy*100)
-
- df = pd.DataFrame({'Distance Metric':
  list(distance_metrics.keys()), 'Accuracy': accuracy_scores})
- fig = px.bar(df, x='Distance Metric', y='Accuracy', title='KNN
  Accuracy with Different Distance Metrics [Wine Dataset]')
```

```
-   fig.show()
```

Result:



Conclusion:

- In this task, the k-Nearest Neighbours (KNN) algorithm was applied to the wine dataset, using different distance metrics such as Euclidean, Minkowski, and Manhattan distances to evaluate the model's performance. After comparison, the Manhattan distance produced the highest accuracy, indicating that the city-block distance metric was better suited to the dataset's structure. This task highlights the importance of selecting the appropriate distance metric for KNN, as it can significantly impact model accuracy and overall performance. Such experimentation is essential in refining machine learning models to yield the best results based on the dataset characteristics.

Practical 5

Support Vector Machine – Classifier

Objective:

- The objective of this practical is to implement a Support Vector Machine (SVM) classifier using both Radial Basis Function (RBF) and linear kernels.

Code:

- Import libraries & load the dataset

```
-  import numpy as np
-  import pandas as pd
-  import plotly.express as px
-  from sklearn.datasets import load_breast_cancer
-  from sklearn.model_selection import train_test_split
-  from sklearn.svm import SVC
-  from sklearn.metrics import accuracy_score, classification_report
```

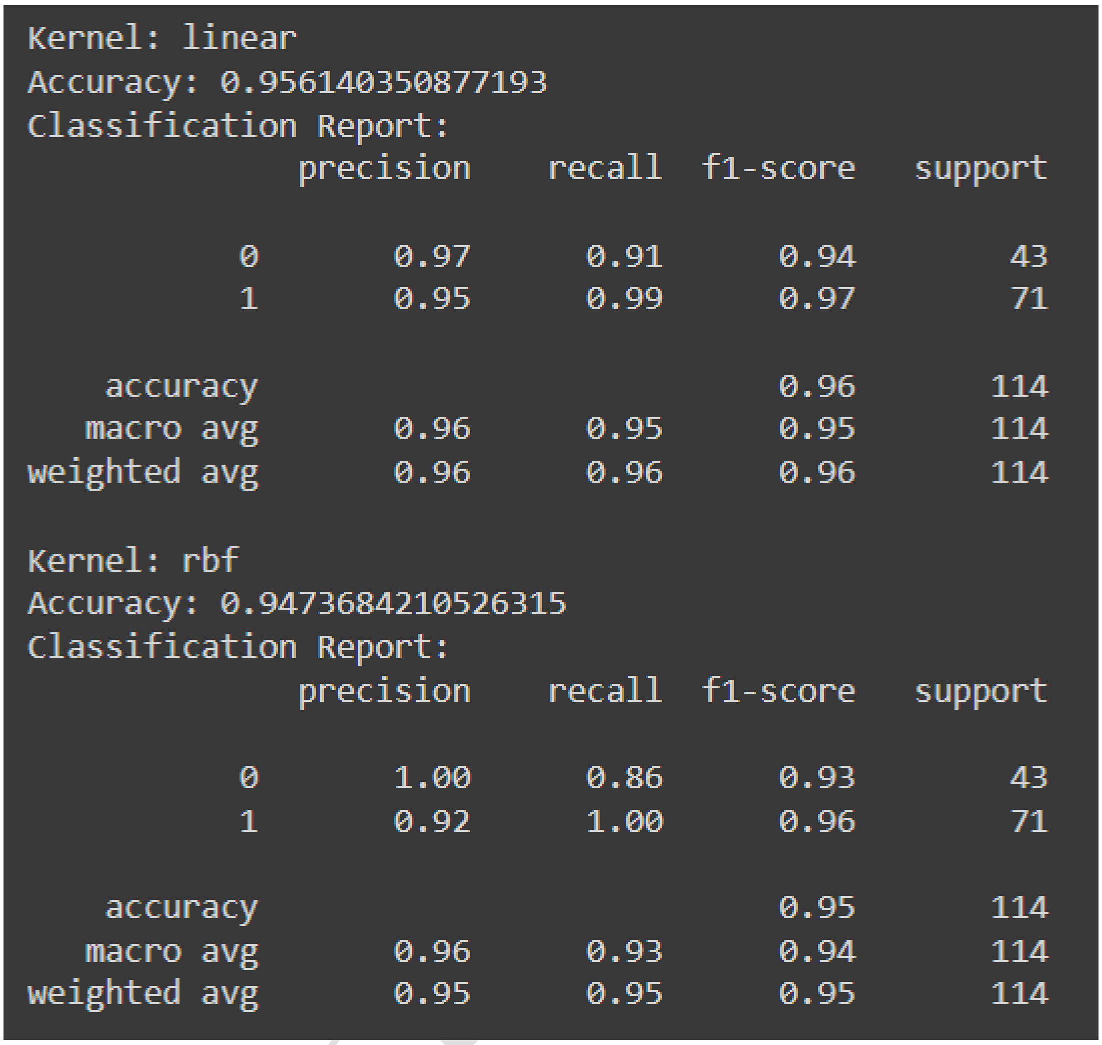
- Train & test split

```
-  breast_cancer = load_breast_cancer()
-  X_train, X_test, y_train, y_test =
   train_test_split(breast_cancer.data, breast_cancer.target,
   test_size=0.2, random_state=42)
```

- Model training & evaluation

```
-  kernel = ['linear', 'rbf']
-
-  def evaluate_kernels(X_train, X_test, y_train, y_test, kernels):
-      results = []
-      for kernel in kernels:
-          model = SVC(kernel=kernel)
-          model.fit(X_train, y_train)
-          y_pred = model.predict(X_test)
-          accuracy = accuracy_score(y_test, y_pred)
-          report = classification_report(y_test, y_pred)
-          results.append({'Kernel': kernel, 'Accuracy': accuracy,
   'Report': report})
-      return results
-
-  results = evaluate_kernels(X_train, X_test, y_train, y_test,
   kernel)
-
-  for result in results:
-      print(f"Kernel: {result['Kernel']}")
-      print(f"Accuracy: {result['Accuracy']}")
-      print(f"Classification Report:\n{result['Report']}")
```

Result:



```
Kernel: linear
Accuracy: 0.956140350877193
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Kernel: rbf
Accuracy: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.86      0.93        43
           1       0.92      1.00      0.96        71

    accuracy                           0.95       114
   macro avg       0.96      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Conclusion:

- In this task, the Support Vector Machine (SVM) classifier with the RBF kernel was compared to the linear kernel on a given dataset. While the RBF kernel showed a lower accuracy score compared to the linear kernel, it achieved a perfect recall score, correctly classifying 100% of the malignant cases. This comparison highlights the importance of selecting the right kernel based on the problem at hand. In this case, the RBF kernel proved more suitable for ensuring that all malignant cases were identified, underscoring that recall is often a crucial metric, particularly in high-stakes applications like medical diagnosis, where misclassification of critical cases must be minimized.

Practical 6

PCA for Dimensionality Reduction

Objective:

- The objective of this practical is to implement Principal Component Analysis (PCA) for dimensionality reduction.

Code:

- Import libraries & load the dataset

```
-   from sklearn.datasets import load_iris
-   iris = load_iris()
-   x = iris.data
-   y = iris.target
```

- Standardize the data

```
-   from sklearn.preprocessing import StandardScaler
-
-   scaler = StandardScaler()
-   x_scaled = scaler.fit_transform(x)
```

- Explained variance & cumulative explained variance

```
-   explained_variance = pca.explained_variance_ratio_
-   cumulative_variance = explained_variance.cumsum()
-
-   print("Explained Variance Ratio:", explained_variance)
-   print("Cumulative Explained Variance:", cumulative_variance)
```

- Plot principal components

```
-   import plotly.graph_objects as go
-
-   fig = go.Figure()
-   fig.add_trace(go.Bar(x=['PC1', 'PC2', 'PC3', 'PC4'],
    y=explained_variance, name='Explained Variance'))
-   fig.add_trace(go.Scatter(x=['PC1', 'PC2', 'PC3', 'PC4'],
    y=cumulative_variance, mode='lines+markers', name='Cumulative
    Explained Variance'))
-
-   fig.update_layout(
-       title='Explained Variance by Principal Component',
-       xaxis_title='Principal Component',
-       yaxis_title='Explained Variance Ratio',
-       yaxis_range=[0,1]
-   )
-
-   fig.show()
```

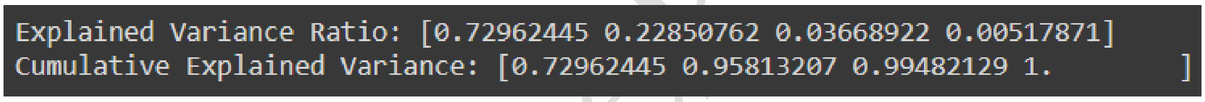
- Scatter plot of first 2 principal components

```
-   import plotly.express as px
-
-   fig = px.scatter(x=x_pca[:, 0], y=x_pca[:, 1], color=y,
-                    labels={'x': 'PC1', 'y': 'PC2'},
-                     title='Scatter Plot of First Two Principal
   Components')
-  fig.show()
```

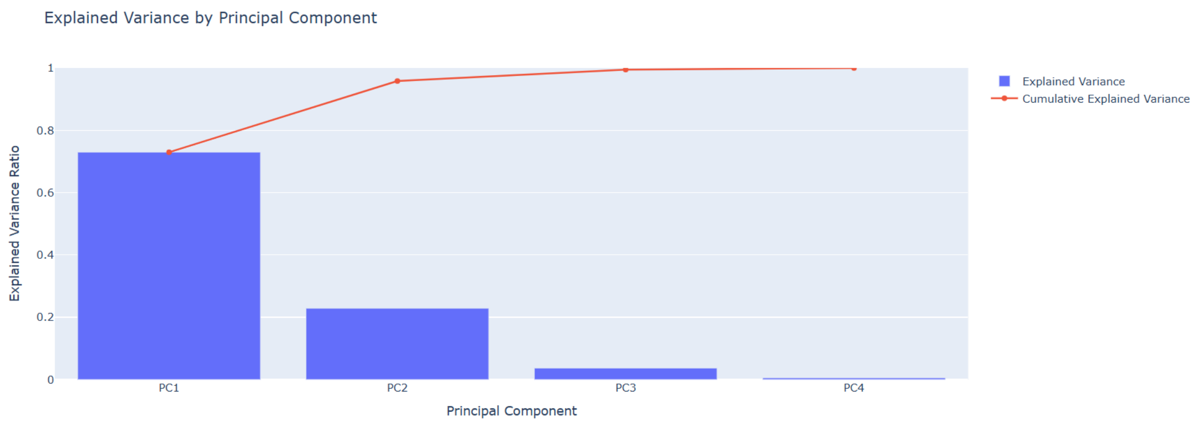
- Composition of principal components

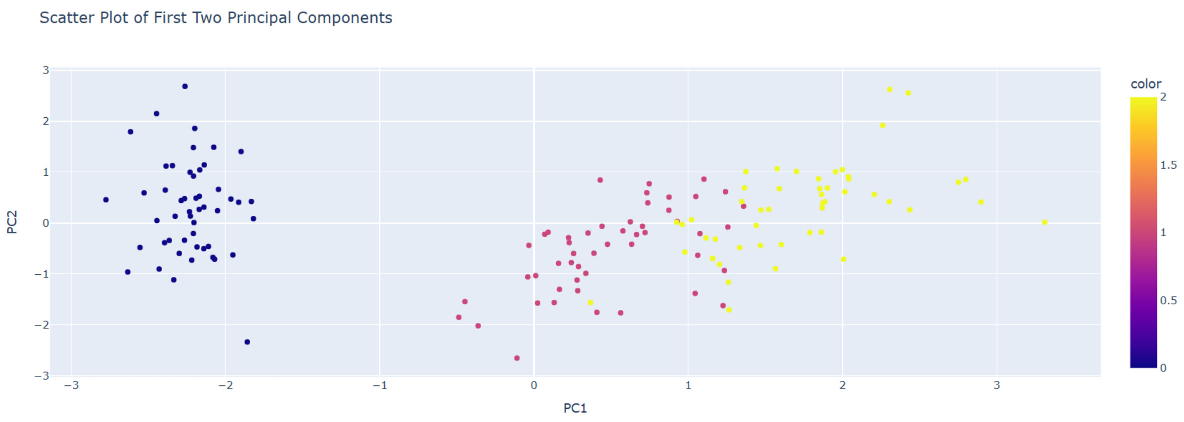
```
-  import pandas as pd
-
-  # Assuming x_pca is your PCA transformed data
-  df_pca = pd.DataFrame(x_pca, columns=['PC1', 'PC2', 'PC3',
   'PC4'])
-
-  # Get the loadings (contribution of original features to each PC)
-  loadings = pd.DataFrame(pca.components_.T, columns=['PC1', 'PC2',
   'PC3', 'PC4'], index=iris.feature_names)
-
-  # Print the loadings for PC1 and PC2
-  print("Loadings for PC1 and PC2:")
-  print(loadings[['PC1', 'PC2']])
```

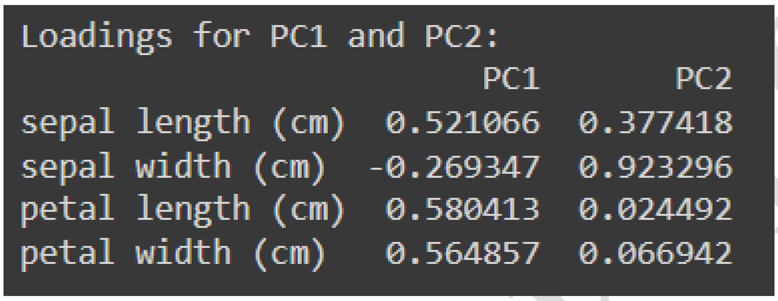
Result:



```
Explained Variance Ratio: [0.72962445 0.22850762 0.03668922 0.00517871]
Cumulative Explained Variance: [0.72962445 0.95813207 0.99482129 1.        ]
```







```
Loadings for PC1 and PC2:
                        PC1       PC2
sepal length (cm)  0.521066  0.377418
sepal width (cm)  -0.269347  0.923296
petal length (cm)  0.580413  0.024492
petal width (cm)   0.564857  0.066942
```

Conclusion:

- In this task, Principal Component Analysis (PCA) was performed to reduce the dimensionality of the dataset while retaining the most important features that explain the variance in the data. PCA helped simplify the dataset by transforming it into a set of orthogonal components, making it easier to visualize and analyze. This dimensionality reduction technique is essential in improving computational efficiency and reducing the risk of overfitting, especially in high-dimensional datasets. By retaining the most informative features, PCA enhances model performance and helps in better understanding the underlying structure of the data.

Practical 7

K-Means Clustering

Objective:

- The objective of this practical is to implement the K-Means clustering algorithm to partition a dataset into distinct clusters based on feature similarity.

Code:

- Import libraries & load the dataset

```
-   import numpy as np
-
-   from sklearn.datasets import load_digits
-
-   data, labels = load_digits(return_X_y=True)
-   (n_samples, n_features), n_digits = data.shape,
    np.unique(labels).size
-
-   print(f"# digits: {n_digits}; # samples: {n_samples}; # features
    {n_features}")
```

- Function to Benchmark K-Means Clustering Performance

```
-   from time import time
-
-   from sklearn import metrics
-   from sklearn.pipeline import make_pipeline
-   from sklearn.preprocessing import StandardScaler
-
-   def bench_k_means(kmeans, name, data, labels):
-
-       t0 = time()
-       estimator = make_pipeline(StandardScaler(), kmeans).fit(data)
-       fit_time = time() - t0
-       results = [name, fit_time, estimator[-1].inertia_]
-
-       # Define the metrics which require only the true labels and
    estimator
-       # labels
-       clustering_metrics = [
-           metrics.homogeneity_score,
-           metrics.completeness_score,
-           metrics.v_measure_score,
-           metrics.adjusted_rand_score,
-           metrics.adjusted_mutual_info_score,
-       ]
-       results += [m(labels, estimator[-1].labels_) for m in
    clustering_metrics]
-
-       # The silhouette score requires the full dataset
-       results += [
-           metrics.silhouette_score(
-               data,
-               estimator[-1].labels_,
-               metric="euclidean",
-               sample_size=300,
-           )
-       ]
-
-       # Show the results
-       formatter_result = (
-           "{:9s}\t{:.3f}s\t{:.0f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}\t
    {:.3f}\t{:.3f}"
-       )
-       print(formatter_result.format(*results))
```

- Benchmarking Different Initialization Methods for K-Means Clustering

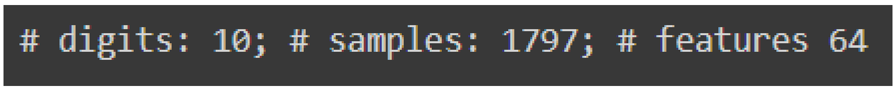
```
-   from sklearn.cluster import KMeans
-   from sklearn.decomposition import PCA
-
-   print(82 * "_")
-   print("init\t\ttime\tinertia\thomo\tcompl\tv-
    meas\tARI\tAMI\tsilhouette")
-
-   kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4,
    random_state=0)
-   bench_k_means(kmeans=kmeans, name="k-means++", data=data,
    labels=labels)
-
-   kmeans = KMeans(init="random", n_clusters=n_digits, n_init=4,
    random_state=0)
-   bench_k_means(kmeans=kmeans, name="random", data=data,
    labels=labels)
-
-   pca = PCA(n_components=n_digits).fit(data)
-   kmeans = KMeans(init=pca.components_, n_clusters=n_digits,
    n_init=1)
-   bench_k_means(kmeans=kmeans, name="PCA-based", data=data,
    labels=labels)
-
-   print(82 * "_")
```

- Visualizing K-Means Clustering Results on PCA-Reduced Data

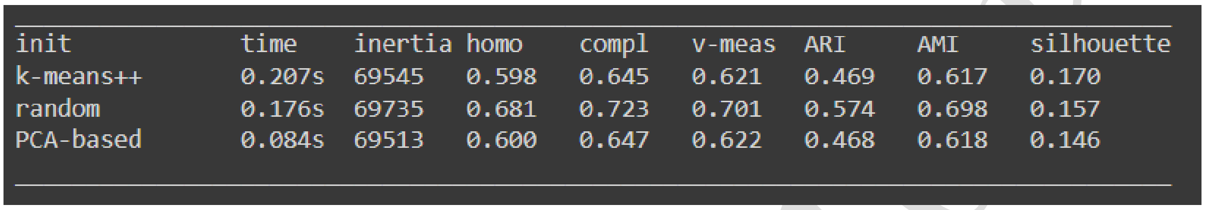
```
-   import matplotlib.pyplot as plt
-
-   reduced_data = PCA(n_components=2).fit_transform(data)
-   kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4)
-   kmeans.fit(reduced_data)
-
```

```
-   # Step size of the mesh. Decrease to increase the quality of the
    VQ.
-   h = 0.02  # point in the mesh [x_min, x_max]x[y_min, y_max].
-
-   # Plot the decision boundary. For that, we will assign a color to
    each
-   x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:,
    0].max() + 1
-   y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:,
    1].max() + 1
-   xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
    y_max, h))
-
-   # Obtain labels for each point in mesh. Use last trained model.
-   Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
-
-   # Put the result into a color plot
-   Z = Z.reshape(xx.shape)
-   plt.figure(1)
-   plt.clf()
-   plt.imshow(
-       Z,
-       interpolation="nearest",
-       extent=(xx.min(), xx.max(), yy.min(), yy.max()),
-       cmap=plt.cm.Paired,
-       aspect="auto",
-       origin="lower",
-   )
-
-   plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.",
    markersize=2)
-   # Plot the centroids as a white X
-   centroids = kmeans.cluster_centers_
-   plt.scatter(
-       centroids[:, 0],
-       centroids[:, 1],
-       marker="x",
-       s=169,
-       linewidths=3,
-       color="w",
-       zorder=10,
-   )
-   plt.title(
-       "K-means clustering on the digits dataset (PCA-reduced
    data)\n"
-       "Centroids are marked with white cross"
-   )
-   plt.xlim(x_min, x_max)
-   plt.ylim(y_min, y_max)
-   plt.xticks(())
-   plt.yticks(())
-   plt.show()
```

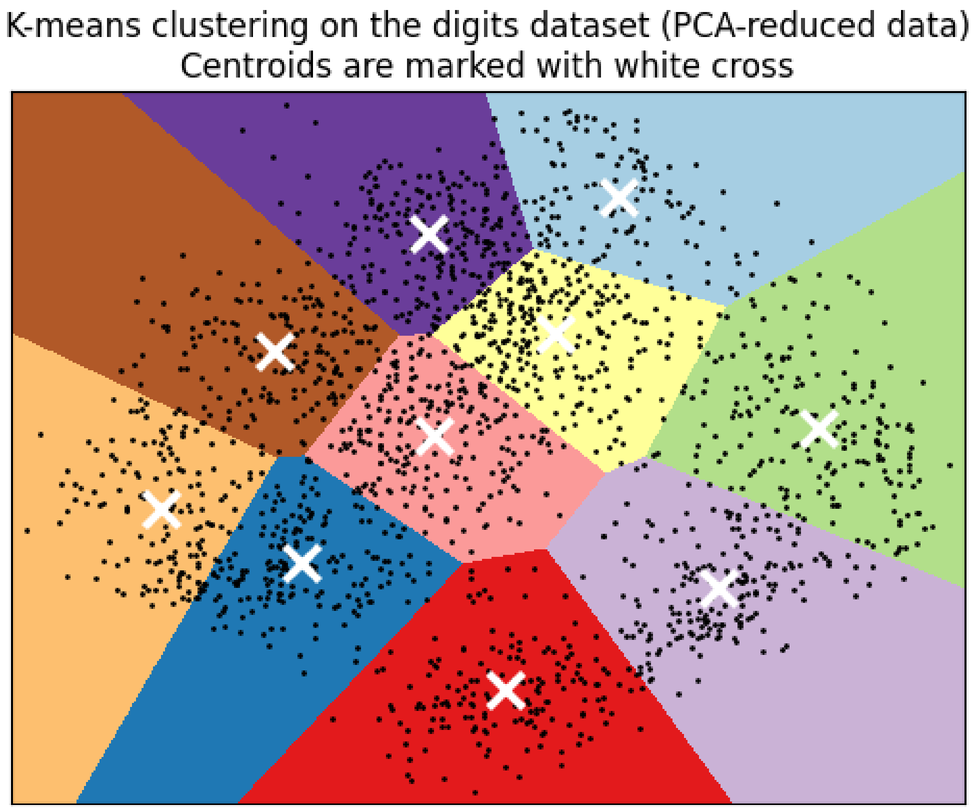
Result:



```
# digits: 10; # samples: 1797; # features 64
```



```
init            time    inertia homo    compl   v-meas  ARI     AMI     silhouette
k-means++       0.207s  69545   0.598   0.645   0.621   0.469   0.617   0.170
random          0.176s  69735   0.681   0.723   0.701   0.574   0.698   0.157
PCA-based       0.084s  69513   0.600   0.647   0.622   0.468   0.618   0.146
```



Conclusion:

- This project successfully demonstrates the application of K-Means clustering on the digits dataset, utilizing PCA for dimensionality reduction. Various initialization methods for the K-Means algorithm were benchmarked, highlighting differences in performance through metrics such as inertia and silhouette scores. Visualizations effectively illustrate the clustering results, showcasing decision boundaries and centroids. This analysis underscores the importance of integrating dimensionality reduction techniques with clustering algorithms, enabling deeper insights into complex datasets and facilitating improved data-driven decision-making.

Practical 8

Decision Tree

Objective:

- The objective of this practical is to implement a Decision Tree classifier to model and predict outcomes based on a set of input features.

Code:

- Import libraries & load the dataset

```
- from sklearn.datasets import load_iris
- iris = load_iris()
```

- Train & test Split

```
- from sklearn.model_selection import train_test_split
- X_train, X_test, y_train, y_test = train_test_split(iris.data,
  iris.target, test_size=0.2)
```

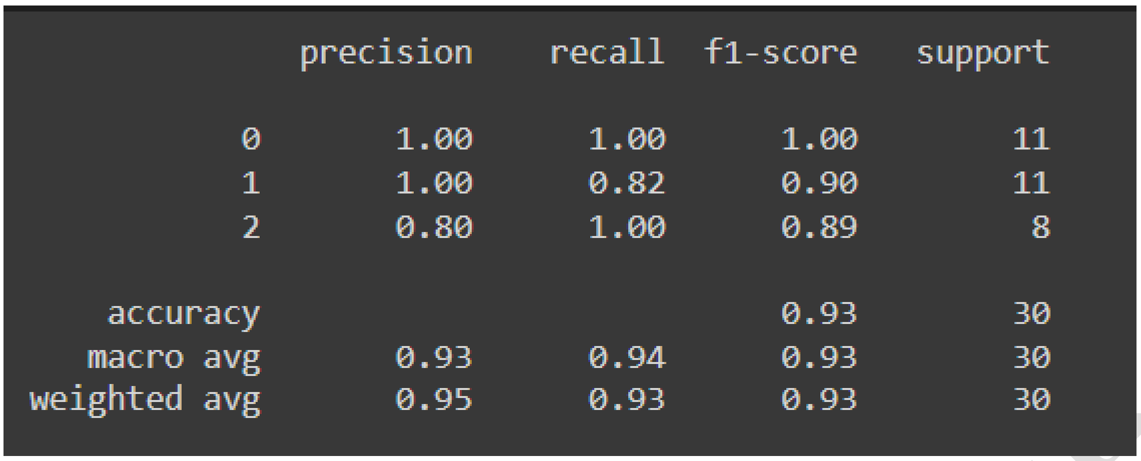
- Model training & evaluation

```
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.metrics import classification_report
-
- # Create a Decision Tree Classifier
- clf = DecisionTreeClassifier(random_state=42)
-
- # Train the classifier
- clf.fit(X_train, y_train)
-
- # Make predictions on the test set
- y_pred = clf.predict(X_test)
-
- # Generate the classification report
- report = classification_report(y_test, y_pred)
-
- print(report)
```

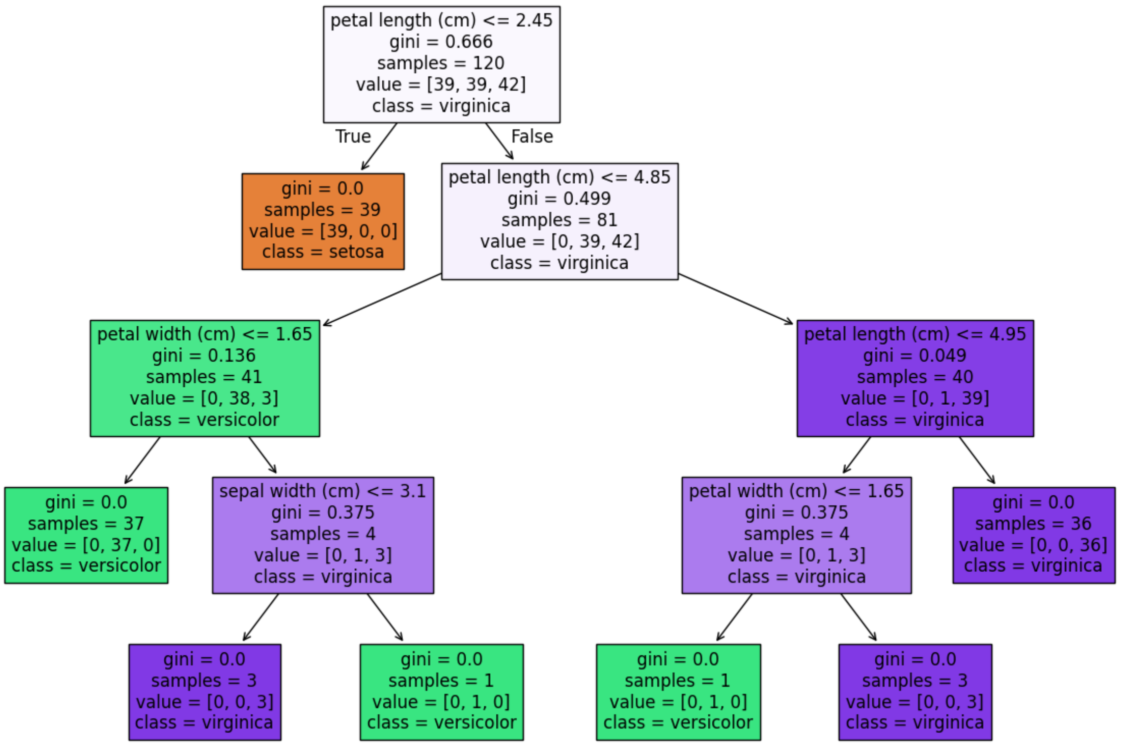
- Plot the decision tree

```
- from sklearn.tree import plot_tree
- import matplotlib.pyplot as plt
-
- plt.figure(figsize=(15,10))
- plot_tree(clf, filled=True, feature_names=iris.feature_names,
  class_names=iris.target_names)
- plt.show()
```

Result:



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.82      0.90        11
           2       0.80      1.00      0.89         8

    accuracy                           0.93        30
   macro avg       0.93      0.94      0.93        30
weighted avg       0.95      0.93      0.93        30
```



Conclusion:

- In this task, a decision tree classifier was applied to the Iris dataset to categorize the different species of iris flowers based on their features. The decision tree algorithm provided a clear and interpretable model that delineates the classification process through a series of decision rules derived from the dataset's attributes. The performance of the model was evaluated using classification metrics such as accuracy, precision, recall, and F1-score, demonstrating the effectiveness of decision trees in handling multi-class classification problems. This task underscores the importance of decision trees in machine learning, offering a transparent approach to classification that is particularly valuable in exploratory data analysis and when interpretability is crucial.

Practical 9

Random Forest

Objective:

- The objective of this practical is to implement a Random Forest classifier to improve predictive accuracy and control overfitting by utilizing an ensemble of decision trees.

Code:

- Import libraries & load the dataset

```
-   import pandas as pd
-   from sklearn.datasets import load_iris
-   iris = load_iris()
-   df = pd.DataFrame(iris.data, columns=iris.feature_names)
-   df['target'] = iris.target
-   df.head()
```

- Verify class imbalance

```
-   df['target'].value_counts()
```

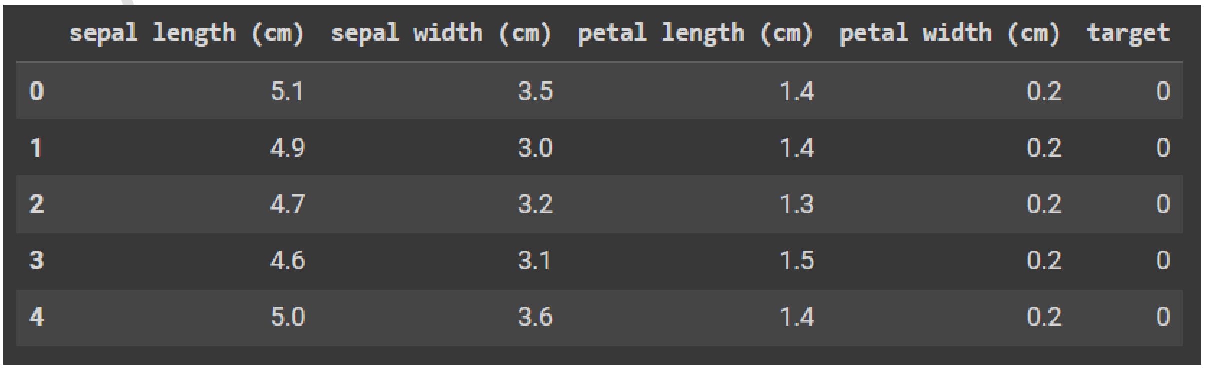
- Train & test split

```
-   from sklearn.model_selection import train_test_split
-   X_train, X_test, y_train, y_test = train_test_split(iris.data,
    iris.target, test_size=0.3)
```

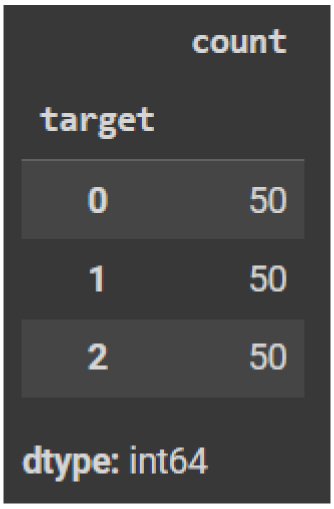
- Model training & evaluation

```
-   from sklearn.ensemble import RandomForestClassifier
-
-   # Create a Random Forest Classifier with random_state=42
-   rf_classifier = RandomForestClassifier(random_state=42)
-
-   # Train the model
-   rf_classifier.fit(X_train, y_train)
-   from sklearn.metrics import classification_report
-   y_pred = rf_classifier.predict(X_test)
-   print(classification_report(y_test, y_pred))
```

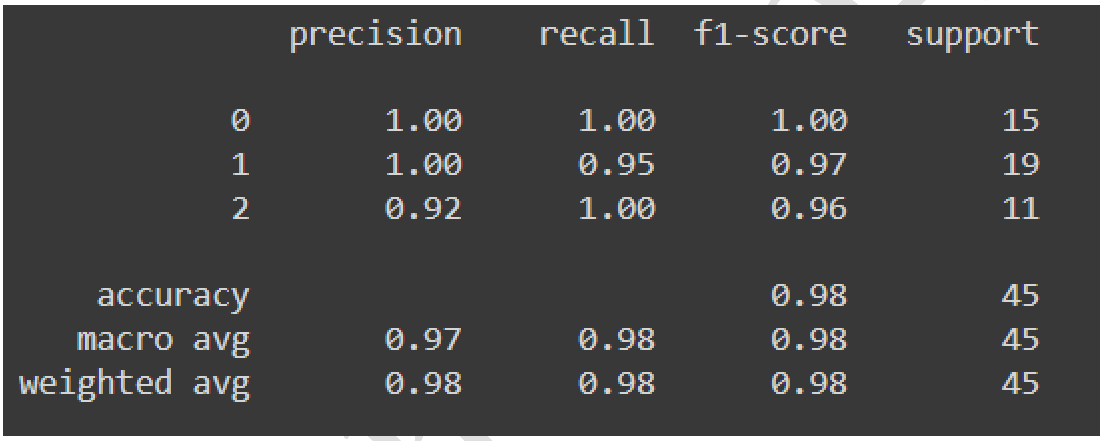
Result:



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0



```
        count
target
     0     50
     1     50
     2     50

dtype: int64
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.95      0.97        19
           2       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Conclusion:

- In this task, a Random Forest classifier was implemented to classify the Iris dataset based on its features. Random Forest, an ensemble learning method, combines multiple decision trees to enhance predictive accuracy and control overfitting. By aggregating the predictions from several trees, this algorithm provides a robust classification model that is less sensitive to noise and variance in the data. The model's performance was evaluated using classification metrics such as accuracy, precision, recall, and F1-score, illustrating its effectiveness in handling multi-class classification tasks. This task highlights the advantages of using Random Forest in machine learning, particularly its ability to improve model reliability and interpretability while maintaining high classification performance.

Practical 10

Gradient Boosting

Objective:

- The objective of this practical is to implement a Gradient Boosting model to enhance predictive accuracy through sequential ensemble learning, where each new tree corrects the errors of its predecessor.

Code:

- Import the libraries & load dataset

```
-   from sklearn.datasets import load_breast_cancer
-
-   data = load_breast_cancer()
-   X, y = data.data, data.target
-   import pandas as pd
-   df = pd.DataFrame(X, columns=data.feature_names)
-   df['target'] = y
-   df
```

- Verify imbalance

```
-   df['target'].value_counts()
```

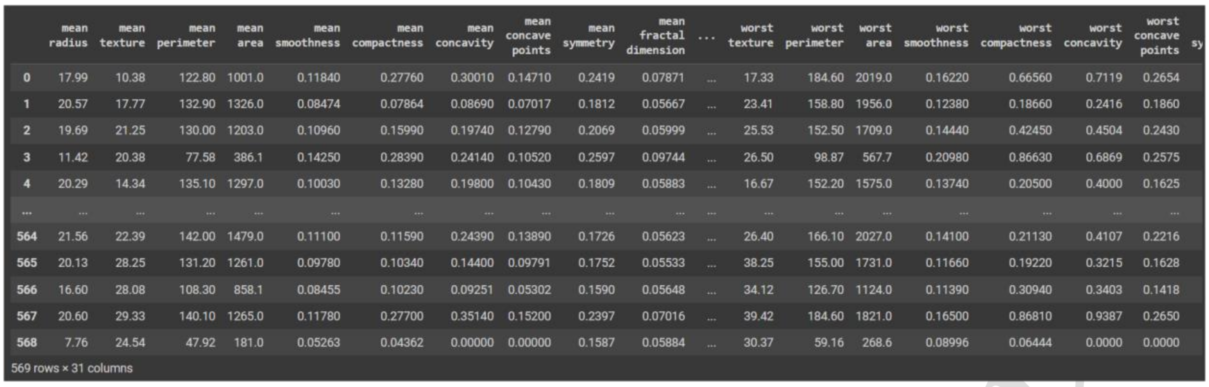
- Train & test split

```
-   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=42)
```

- Model training & evaluation

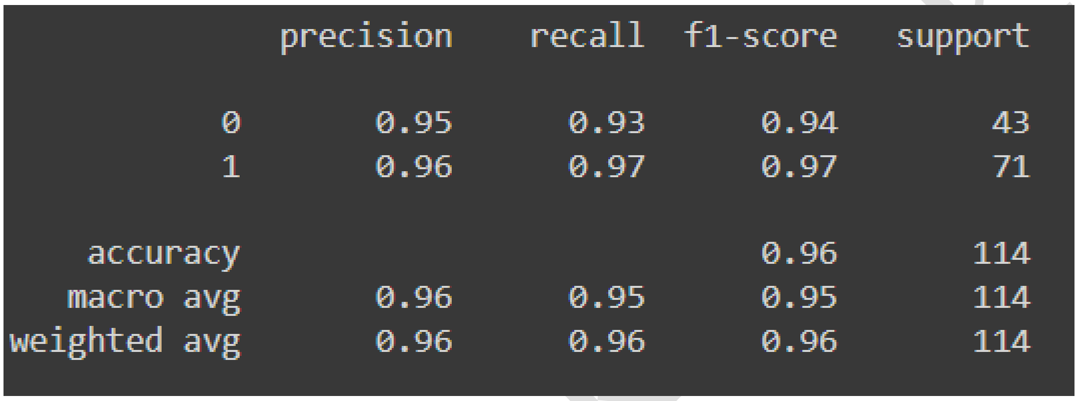
```
-   from sklearn.ensemble import GradientBoostingClassifier
-   from sklearn.metrics import classification_report
-
-   # Initialize the Gradient Boosting Classifier
-   gb_classifier = GradientBoostingClassifier()
-
-   # Train the model
-   gb_classifier.fit(X_train, y_train)
-
-   # Make predictions on the test set
-   y_pred = gb_classifier.predict(X_test)
-
-   # Evaluate the model using classification report
-   print(classification_report(y_test, y_pred))
```

Result:



	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	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	

569 rows × 31 columns



```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.96      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Conclusion:

- In this task, Gradient Boosting was applied to the breast cancer dataset to classify whether tumours were malignant or benign based on various features. This ensemble learning technique builds multiple weak learners sequentially, with each new model focused on correcting the errors of the previous ones. The final model aggregates these predictions to enhance accuracy and reduce the likelihood of overfitting. The model's performance was evaluated using a classification report, providing a comprehensive overview of key metrics such as precision, recall, F1-score, and support for each class. This evaluation highlights the effectiveness of Gradient Boosting in improving classification accuracy in critical applications like breast cancer diagnosis, where reliable predictions are essential for patient care. The task emphasizes the value of Gradient Boosting as a powerful tool in machine learning for handling complex classification problems.