Experiment -1

Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation

`statistics` Library

Overview:

- **Purpose:** Simplifies descriptive statistical calculations such as measures of central tendency (mean, median, mode) and measures of dispersion (variance, standard deviation).
- **Key Functions:**
  - `mean(data)`: Computes the average of the data.
  - `median(data)`: Finds the middle value in a sorted dataset.
  - `mode(data)`: Determines the most frequent value in the dataset.
  - `variance(data)`: Measures the spread of the dataset from the mean.
  - `stdev(data)`: Calculates the standard deviation, quantifying data spread.

Use Case:

- Used for exploratory data analysis to summarize key metrics of a dataset.

Example Insights:

- In the program, we used **both manual calculations and `statistics` functions** to showcase the ease and accuracy the library provides. The library eliminates the need to write complex loops for basic statistical tasks.

Program:

```
import statistics

# Dynamic input: Accept dataset from the user
data = list(map(float, input("Enter the dataset values separated by spaces: ").split()))

# Mean
mean = sum(data) / len(data)
print("\nMean (manual):", mean)
print("Mean (library):", statistics.mean(data))

# Median
sorted_data = sorted(data)
n = len(data)
median = (sorted_data[n // 2] + sorted_data[(n - 1) // 2]) / 2 if n % 2 == 0 else sorted_data[n // 2]
print("\nMedian (manual):", median)
print("Median (library):", statistics.median(data))
```

```
# Mode
    mode = statistics.mode(data)
    print("\nMode (library):", mode)


# Variance
mean_square_diff = sum((x - mean) ** 2 for x in data) / len(data)
print("\nVariance (manual):", mean_square_diff)
print("Variance (library):", statistics.variance(data))

# Standard Deviation
std_dev = mean_square_diff ** 0.5
print("\nStandard Deviation (manual):", std_dev)
print("Standard Deviation (library):", statistics.stdev(data))
```

Experiment-2

`Math` Library

Overview:

- **Purpose:** Provides foundational mathematical functions to perform operations like trigonometry, logarithms, power functions, and constants like π and e.
- **Key Functions:**
  - `sqrt(x)`: Computes the square root of x.
  - `factorial(x)`: Finds the factorial of an integer x.
  - `pow(x, y)`: Raises x to the power of y.
  - `log(x, base)`: Computes the logarithm of x to the given base.
  - `ceil(x)` and `floor(x)`: Round numbers up or down.

Use Case:

- Ideal for mathematical modeling, geometry, physics simulations, and simple calculations.

Example Insights:

- The program demonstrates basic operations like square root and factorial to show the versatility of the library in numerical computations. It bridges the gap between basic arithmetic and advanced calculations.

Program:

```
import math

# Dynamic input for basic math operations
num = float(input("Enter a number for mathematical operations: "))

# Arithmetic Operations
print("\nBasic Operations:")
print(f"Square root of {num}: {math.sqrt(num)}")
print(f"Factorial of {int(num)} (if integer): {math.factorial(int(num)) if num.is_integer() and num >= 0 else 'Factorial not defined for this input.'}")
print(f"{num} raised to the power 3: {math.pow(num, 3)}")

# Trigonometric Functions
print("\nTrigonometric Functions:")
angle = float(input("Enter an angle (in degrees): "))
radians = math.radians(angle)
print(f"Sine of {angle} degrees: {math.sin(radians)}")
print(f"Cosine of {angle} degrees: {math.cos(radians)}")
print(f"Tangent of {angle} degrees: {math.tan(radians)}")

# Logarithmic and Exponential Functions
print("\nLogarithmic and Exponential Functions:")
```

```
log_base = float(input(f"Enter a base for logarithm of {num}: "))
print(f"Logarithm of {num} to base {log_base}: {math.log(num, log_base)}")
print(f"Natural logarithm (ln) of {num}: {math.log(num)}")
print(f"Exponential of {num}: {math.exp(num)}")

# Rounding Functions
print("\nRounding Functions:")
print(f"Ceiling of {num}: {math.ceil(num)}")
print(f"Floor of {num}: {math.floor(num)}")
```

`Numpy` Library

Overview:

- **Purpose:** Provides support for large multi-dimensional arrays, matrices, and a collection of mathematical functions to operate on them efficiently.
- **Key Features:**
  - **Arrays and Broadcasting:** Efficient operations on large datasets without explicit loops.
  - **Mathematical Operations:** Supports functions like `mean`, `sum`, `min`, `max`, and element-wise operations.
  - **Linear Algebra:** Offers matrix operations, eigenvalues, and vector manipulations.
  - **Random Sampling:** Generates random numbers or samples datasets.

Use Case:

- Foundational library for numerical computations in scientific computing and machine learning.

Example Insights:

- In the program, `np.array()` was used to create an array. Operations like `np.sum()` and `np.mean()` demonstrate how **operations** eliminate manual iteration for performance optimization.

Why It's Important:

- It's the backbone of Python's data science stack and integrates seamlessly with libraries like `pandas`, `scipy`, and `matplotlib`.

Program:

```
import numpy as np

# Create a NumPy array from user input
data = np.array(list(map(float, input("Enter numbers to create a NumPy array (space-separated):
").split())))

# Basic Array Operations
print("\nBasic Array Operations:")
print("Array:", data)
```

```
print("Shape of array:", data.shape)
print("Size of array:", data.size)
print("Mean:", np.mean(data))
print("Sum:", np.sum(data))
print("Maximum:", np.max(data))
print("Minimum:", np.min(data))

# Element-wise Operations
print("\nElement-wise Operations:")
print("Array squared:", data ** 2)
print("Array multiplied by 2:", data * 2)

# Linear Algebra
print("\nLinear Algebra Operations:")
matrix = data.reshape(-1, 1) if len(data) % 2 == 0 else np.append(data, 0).reshape(-1, 2)
print("Matrix (reshaped data):\n", matrix)
if matrix.shape[0] == matrix.shape[1]:  # Square matrix required
   print("Matrix Determinant:", np.linalg.det(matrix))

# Random Sampling
print("\nRandom Sampling:")
rand_arr = np.random.randint(0, 100, 5)
print("Random array of 5 integers:", rand_arr)
```

scipy Library

Overview:

- **Purpose:** Built on top of `numpy`, it extends functionality to include advanced algorithms for scientific and technical computing.
- **Modules:**
  - `stats`: For statistical functions like Z-scores, t-tests, and distributions.
  - `integrate`: For numerical integration.
  - `optimize`: For solving optimization problems.
  - `signal`: For signal processing tasks.
  - `interpolate`: For interpolation methods.

Use Case:

- Used in scenarios requiring precise scientific computations like engineering simulations, physics models, or statistical testing.

Advanced Applications:

- Multivariate statistical analysis, curve fitting, and optimization problems in research domains.

Program:

```
from scipy import stats, integrate
import numpy as np

# Dataset for statistical analysis
data = np.array(list(map(float, input("Enter numbers for statistical analysis (space-separated): ").split())))

# Statistical Functions
print("\nStatistical Analysis:")
print("Mean:", stats.tmean(data))
print("Variance:", stats.tvar(data))
print("Standard Deviation:", stats.tstd(data))
print("Mode:", stats.mode(data, keepdims=False))
print("Z-Scores:", stats.zscore(data))

# Integration
print("\nIntegration Example:")
result, _ = integrate.quad(lambda x: x**2, 0, 1)  # ∫(x^2) dx from 0 to 1
print("Definite integral of x^2 from 0 to 1:", result)

# Probability Distribution
print("\nProbability Distribution:")
mean, std = np.mean(data), np.std(data)
print(f"PDF of normal distribution at mean ({mean}):", stats.norm.pdf(mean, loc=mean, scale=std))
print(f"CDF of normal distribution at mean ({mean}):", stats.norm.cdf(mean, loc=mean, scale=std))
```

Key Functionalities :

Library	Feature	Description
math	Arithmetic Operations	Square root, factorial, power functions
math	Trigonometric Functions	sin, cos, tan with angle conversion from degrees to radians
math	Logarithms and Exponentials	Base n logarithm, natural logarithm, and exponentials
numpy	Array Manipulation	Array reshaping, basic operations (sum, mean, max, min)
numpy	Element-wise Operations	Square, scalar multiplication
numpy	Linear Algebra	Determinant of square matrices
numpy	Random Sampling	Generating random integers
scipy	Statistical Analysis	mode, zscore, and variance calculations
scipy	Integration	Numerical integration of functions
scipy	Probability Distribution	PDF and CDF of normal distributions

EXPERIMENT 3 :

Study of Python Libraries for ML application such as Pandas and Matplotlib ?

PANDAS LIBRARY

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

1. **Series**:
   - One-dimensional labeled array.
   - Acts like a NumPy array but with labels.

```
import pandas as pd
s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
print(s['a'])  # Access by label
```

2. **DataFrame**:
   - Two-dimensional, tabular data structure.
   - Each column is a **Series**.

```
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df = pd.DataFrame(data)
print(df['Name'])  # Access column
```

3. **Index**:
   - Immutable labels for rows and columns.
   - Can be customized for hierarchical indexing.

```
df = pd.DataFrame(data, index=['Row1', 'Row2'])
```

Data Selection and Slicing:

1. **Label-based selection (`.loc`)**:
   - Access rows/columns by label.

```
df.loc['Row1', 'Name']  # Access specific cell
```

2. **Integer-location based (`.iloc`)**:
    - Access rows/columns by position.

```
df.iloc[0, 0]  # First row, first column
```

Data Transformation:

- **Apply custom functions** to data:

```
df['Age'] = df['Age'].apply(lambda x: x + 5)
```

- **Vectorized operations** for efficiency:

```
df['NewCol'] = df['Age'] * 2
```

Merging and Joining:

- Combine datasets:
    - **Concatenation**:

```
pd.concat([df1, df2], axis=0)  # Stack rows
```

    - **Merge** on a common column:

```
pd.merge(df1, df2, on='key')
```

Handling Time-Series Data:

- Built-in support for time-based indexing and resampling:

```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df.resample('M').mean()  # Monthly aggregation
```

Data Loading:

    - Supports various formats like CSV, Excel, SQL, JSON, and more.

```
import pandas as pd
data = pd.read_csv('data.csv')
```

Data Cleaning:

    - Handling missing values:

```
data.fillna(value=0, inplace=True)  # Replace NaN with 0
data.dropna(inplace=True)          # Drop rows with NaN
```

    - Removing duplicates:

```
data.drop_duplicates(inplace=True)
```

Data Transformation:

- Renaming columns:

```
data.rename(columns={'OldName': 'NewName'}, inplace=True)
```

- Filtering rows:

```
filtered_data = data[data['column'] > threshold]
```

Data Aggregation:

- Grouping data:

```
grouped = data.groupby('column').mean()
```

Applications in ML:

- **Feature Engineering**: Handle categorical and numerical transformations.
- **Data Preprocessing**: Clean and prepare datasets before feeding them into ML models.
- **Integration**: Works seamlessly with other ML libraries like **NumPy** and **Scikit-learn**.

MATPLOT LIBRARY

Matplotlib is a versatile library for creating static, animated, and interactive visualizations in Python. It helps visualize patterns, trends, and distributions in data.

Key Features:

Line Plots:

- Used to visualize trends over time.

```
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot')
plt.show()
```

Bar Charts:

- Useful for categorical data comparisons.

```
plt.bar(categories, values)
```

Scatter Plots:

- Ideal for exploring relationships between two variables.

```
plt.scatter(x, y)
```

Histograms:

- Useful for visualizing the distribution of a variable.

```
plt.hist(data['column'], bins=10)
```

Customization:

- Add grid lines, legends, colors, and more:

```
plt.grid(True)
plt.legend(['Line 1', 'Line 2'])
plt.show()
```

- Change **line style** and **colors**:

```
plt.plot(x, y, linestyle='--', color='red', linewidth=2)
```

- Modify **ticks**:

```
ax.set_xticks([0, 5, 10])
ax.set_xticklabels(['Low', 'Medium', 'High'])
```

Subplots:

- Create multiple plots within a figure.

```
fig, axs = plt.subplots(2, 2)  # 2x2 grid
axs[0, 0].plot(x, y)
axs[1, 1].bar(categories, values)
```

Interactive Plots:

- Use **Pan/Zoom** with interactive backends.

```
plt.ion()  # Interactive mode on
```

Applications in ML:

- **EDA**: Gain insights into the dataset through visual exploration.
- **Model Evaluation**: Visualize performance metrics like accuracy, loss curves, or confusion matrices.
- **Feature Relationships**: Understand how features interact.

LAB-4

Write a Python program to implement Simple Linear Regression

#Importing Required Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split;
from sklearn.linear_model import LinearRegression;
import matplotlib.pyplot as pt;
from sklearn.metrics import r2_score;
```

#Importing Student Performance Dataset

```
data = pd.read_csv("/content/Student_Performance.csv");
df_data=pd.DataFrame(data)

print(df_data.head())
```

#Data Cleaning

```
print(df_data.isnull().sum())
#df_data.fillna(df_data[1].mean,inplace=True)


print(df_data.head())
```

#Splitting dataset for Training and Testing Phrase

```
x=df_data[["Hours Studied"]]
print(x)
y=df_data[["Performance Index"]]
print(y)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=42);
```

#Data Visualization

```
pt.scatter(x,y)
pt.xlabel("Hours Studied")
pt.ylabel("Performance Index")
pt.show()
```

#Training Phrase

```
model=LinearRegression();
model.fit(x_train,y_train);
```

#Testing Phrase

```
prediction=model.predict(x_test)
print(prediction)
```

#Model Evaluation with r2Score

```
eval=r2_score(y_test,prediction)
print(eval)
```

LAB-5

Implementation of Multiple Linear Regression for House Price Prediction using sklearn

#Importing Required Libraries

```
import numpy as np;
import pandas as pd
from sklearn.model_selection import train_test_split;
from sklearn.linear_model import LinearRegression;
import matplotlib.pyplot as pt;
from sklearn.metrics import r2_score;
from sklearn.preprocessing import LabelEncoder;
```

#Importing House Price Prediction Dataset

```
data = pd.read_csv("/content/DOC-20241116-WA0004_.csv");
df_data=pd.DataFrame(data)

print(df_data.head())
```

#Cleaning the data

```
print(df_data.isnull().sum())
#df_data.fillna(df_data[1].mean,inplace=True)
le = LabelEncoder();
df_data["furnishingstatus"]=le.fit_transform(df_data["furnishingstatus"]);
df_data["mainroad"]=le.fit_transform(df_data["mainroad"]);
df_data["guestroom"]=le.fit_transform(df_data["guestroom"]);
df_data["basement"]=le.fit_transform(df_data["basement"]);
df_data['hotwaterheating']=le.fit_transform(df_data['hotwaterheating']);
df_data["airconditioning"]=le.fit_transform(df_data["airconditioning"]);
df_data["prefarea"]=le.fit_transform(df_data["prefarea"]);

print(df_data.head())
```

#Spliting the dataset for Training and Testing Phrase

```
x=df_data.iloc[:,1:]
print(x)
y=df_data["price"]
print(y)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=42);
```

#Training the model

```
model=LinearRegression();
model.fit(x_train,y_train);
```

#Testing the model

```
prediction=model.predict(x_test)
print(prediction)
```

#Evaluating the model with r2Score

```
eval=r2_score(y_test,prediction)
print(eval)
```

LAB-6

Implementation of Decision tree using sklearn and its parameter tuning

Using Online Fraud Detection Dataset

#Importing Required Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
```

#Importing House Price Prediction Dataset

```
dataset=pd.read_csv('/content/onlinefraud.csv')
df_dataset=pd.DataFrame(dataset);
print(df_dataset)
```

#Cleaning the data

```
le=LabelEncoder();
df_dataset['type']=le.fit_transform(df_dataset['type']);
df_dataset['nameOrig'] = le.fit_transform(df_dataset['nameOrig'])
df_dataset['nameDest'] = le.fit_transform(df_dataset['nameDest'])
print(df_dataset)
print("Missing Values:\n", df_dataset.isnull().sum())
df_dataset.dropna(inplace=True)
print(df_dataset.isnull().sum())
```

#Spliting the dataset for Training and Testing Phrase

```
x=df_dataset.iloc[:,:-1]
y=df_dataset['isFlaggedFraud'];
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=42)
```

#Training the model

```
model=DecisionTreeClassifier();
model.fit(X_train, y_train)
```

#Visualize Decision Tree

```
plt.figure(figsize=(10, 10))
plot_tree(model,filled=True,feature_names=X_train.columns, )
plt.show()
```

#Testing the model

```
predictions=model.predict(X_test)
print(predictions)
```

#Evaluating the model

```
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

#Tuning with grid search Method

```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid,
scoring='accuracy', cv=5)
```

#Performing grid search tuning

```
grid_search.fit(X_train, y_train)
best_clf = grid_search.best_estimator_
```

#Training the dataset with best model obtained from grid search

```
best_clf.fit(X_train, y_train)
```

#Visualizing the decision tree

```
plt.figure(figsize=(10, 10))
plot_tree(model,filled=True,feature_names=X_train.columns, )
plt.show()
```

#Testing

```
y_pred_optimized = best_clf.predict(X_test)
print(y_pred_optimized)
```

#Evaluating the model

```
print(accuracy_score(y_test, y_pred_optimized))
print(classification_report(y_test, y_pred_optimized))
```

LAB-7

Implementation of KNN using sklearn
Using Diabetics Dataset

#Importing Required Libraries

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, Classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

#Importing Diabetics Prediction Dataset

```
dataset = pd.read_csv("/content/diabetics.csv");
df_dataset=pd.DataFrame(dataset)

print(df_dataset.head())
```

#Cleaning the data

```
df_dataset.isnull().sum()
print(df_dataset)
```

#Spliting the dataset for Training and Testing Phrase

```
x=df_dataset.iloc[:,:-1]
y=df_dataset['Outcome'];
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

#Training Phrase with neighbors=5

```
model=KNeighborsClassifier(n_neighbors=5)
model.fit(xtrain,ytrain)
```

#Testing Phrase

```
predictions=model.predict(xtest)
```

#Evaluating the model

```
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

#Again Training with neighbors=8

```
model=KNeighborsClassifier(n_neighbors=8)
model.fit(xtrain,ytrain)
```

#Testing Phrase

```
predictions=model.predict(xtest)
```

#Evaluating the model

```
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

LAB-8

Implementation of Logistic Regression using sklearn
Using Chronic Heart Disease Dataset

#Importing Required Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,accuracy_score
from sklearn.preprocessing import LabelEncoder
```

#Importing Chronic Heart Disease Prediction Dataset

```
dataset = pd.read_csv("/content/CHD.csv");
df_dataset=pd.DataFrame(dataset)
df_dataset.shape
print(df_dataset.head())
```

#Cleaning the data

```
Print(df_dataset.isnull().sum())
df_dataset['education']=df_dataset['education'].fillna(df_dataset['education'].mean())
df_dataset['cigsPerDay']=df_dataset['cigsPerDay'].fillna(df_dataset['cigsPerDay'].median
())
df_dataset['BPMeds']=df_dataset['BPMeds'].fillna(df_dataset['BPMeds'].mean())
df_dataset['BMI']=df_dataset['BMI'].fillna(df_dataset['BMI'].mean())
df_dataset['totChol']=df_dataset['totChol'].fillna(df_dataset['totChol'].median())
df_dataset['heartRate']=df_dataset['heartRate'].fillna(df_dataset['heartRate'].mean())
df_dataset['glucose']=df_dataset['glucose'].fillna(df_dataset['glucose'].mean())

print(df_dataset)
```

#Spliting the dataset for Training and Testing Phrase

```
x=df_dataset.iloc[:,:-1]
y=df_dataset['CHD'];
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

#Training Phrase

```
lr=LogisticRegression(max_iter=400)
lr.fit(xtrain,ytrain)
```

#Testing Phrase

```
predictions=lr.predict(xtest)
predictions
```

#Evaluating the model

```
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

LAB-9

Implementation of K-Means Clustering
Using Mall-Customer Segmentation Dataset

#Importing Required Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report,accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
```

#Importing Mall-Customer Segmentation Dataset

```
dataset=pd.read_csv("/content/Mall_Customers.csv")
df_dataset=pd.DataFrame(dataset)
df_dataset.shape
```

#Cleaning and Pre-Processing the data

```
df_dataset.isnull().sum()
LE=LabelEncoder()
df_dataset["Gender"]=LE.fit_transform(df_dataset["Gender"])
print(df_dataset)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_dataset)
print(X_scaled)
```

#Creating Elbow Curve

```
inertias = []
for k in range(1, 21):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

print(inertias)
```

#Visualizing the Elbow Curve

```
plt.figure(figsize=(8, 6))
plt.plot(range(1, 21), inertias, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```

#Training the Model with Optimal K

```
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k)
kmeans.fit(X_scaled)
```

#Reducing Dimensionality to Visualize the Data

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

#Visualizing the Clusters

```
plt.figure(figsize=(8, 6))

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans.labels_, cmap='viridis', s=100,
alpha=0.7, edgecolor='black')

centers = pca.transform(kmeans.cluster_centers_)
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X', label='Cluster
Centers')

plt.title(f'K-Means Clustering with k={optimal_k}')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

LAB-10

Performance analysis of Classification Algorithms on a specific dataset (Mini Project)

Air Pollution Dataset used for these Experiment:

#Importing Required 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, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

#Importing Air Pollution Dataset

```
dataset=pd.read_csv('/content/updated_pollution_dataset.csv')
df_dataset=pd.DataFrame(dataset);
df_dataset
```

#Cleaning and Pre-Processing the data

```
LE=LabelEncoder();
df_dataset['Air Quality'] = LE.fit_transform(df_dataset['Air Quality']);
print(df_dataset.isnull().sum())
df_dataset
```

#Spliting the dataset for Training and Testing Phrase

```
x=df_dataset.iloc[:,:-1];
y=df_dataset['Air Quality'];
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Defining the classification models to compare

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=200),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier()
}

results = {}
```

Training and evaluating each model

```
for model_name, model in models.items():
    # Training the model
    model.fit(X_train_scaled, y_train)

    # Making predictions on the test set
    y_pred = model.predict(X_test_scaled)

    # Evaluating the performance
    accuracy = accuracy_score(y_test, y_pred)
    classification_rep = classification_report(y_test, y_pred)
    confusion_mat = confusion_matrix(y_test, y_pred)

    # Storing results
    results[model_name] = {
        "accuracy": accuracy,
        "classification_report": classification_rep,
        "confusion_matrix": confusion_mat
    }

    # Printing the results
    print(f"Model: {model_name}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Classification Report:\n{classification_rep}")
    print(f"Confusion Matrix:\n{confusion_mat}\n")
```

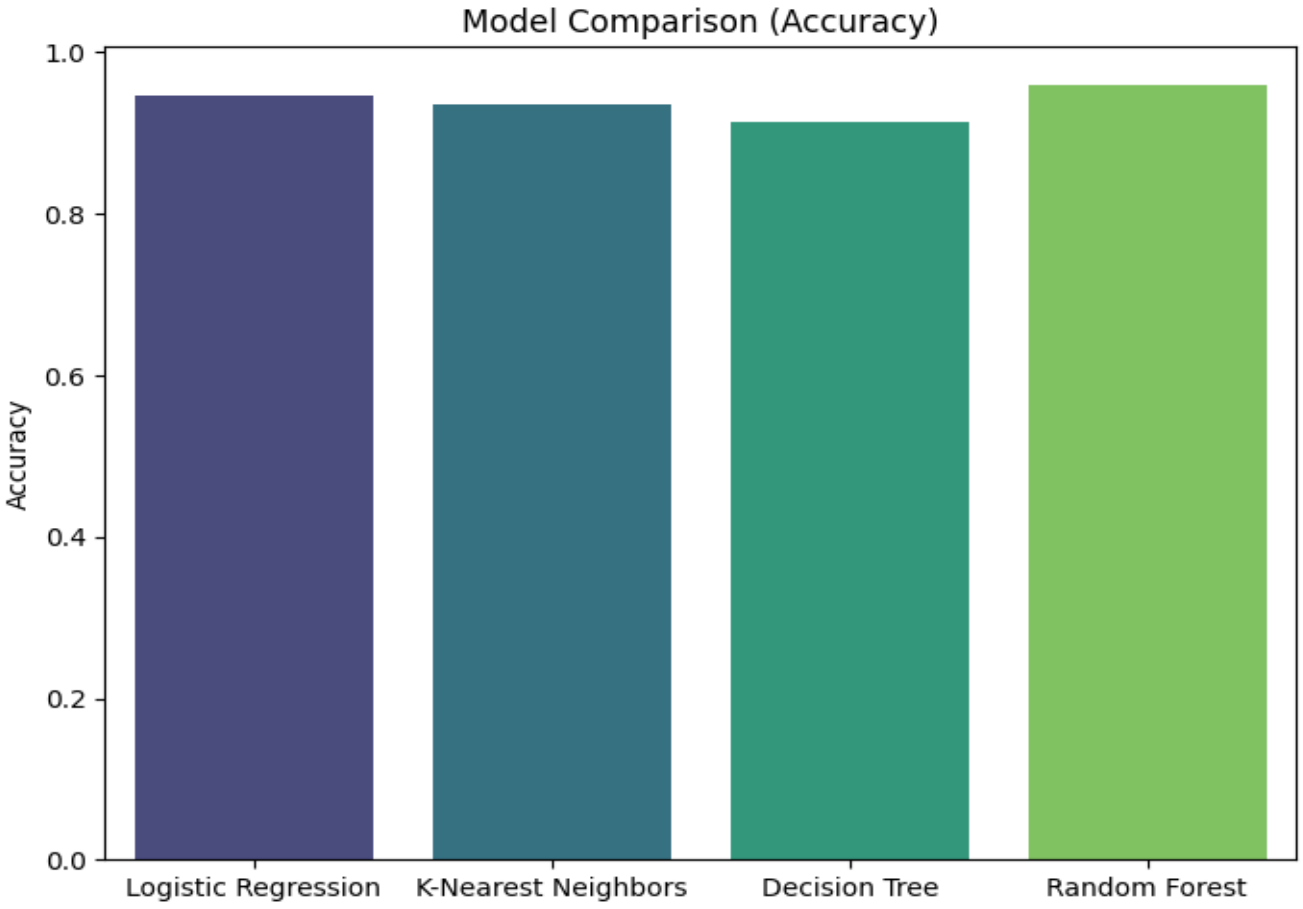
Comparing performance of models
Plotting the accuracy of all models

```
model_names = list(results.keys())
accuracies = [results[model]["accuracy"] for model in model_names]

plt.figure(figsize=(8, 6))
sns.barplot(x=model_names, y=accuracies, palette="viridis")
plt.title("Model Comparison (Accuracy)")
plt.ylabel("Accuracy")
plt.show()

# If needed, we can save the results to a CSV or text file
results_df = pd.DataFrame({
    "Model": model_names,
    "Accuracy": accuracies
})
results_df.to_csv("model_comparison_results.csv", index=False)
```

OUTPUT:


Model Comparison (Accuracy)
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
K-Nearest Neighbors
Decision Tree
Random Forest