



# ML Record - Google Docs - Machine Learning Lab practical solutions

Machine Learning (Jawaharlal Nehru Technological University, Hyderabad)



Scan to open on Studocu

<b>S.No</b>	<b>List of Experiments</b>	<b>Page No.</b>
<b>1.</b>	<b>Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation</b>	<b>2</b>
<b>2.</b>	<b>Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy</b>	<b>2-5</b>
<b>3.</b>	<b>Study of Python Libraries for ML application such as Pandas and Matplotlib</b>	<b>5-8</b>
<b>4.</b>	<b>Write a Python program to implement Simple Linear Regression</b>	<b>9</b>
<b>5.</b>	<b>Implementation of Multiple Linear Regression for House Price Prediction using sklearn</b>	<b>10</b>
<b>6.</b>	<b>Implementation of Decision tree using sklearn and its parameter tuning</b>	<b>11-12</b>
<b>7.</b>	<b>Implementation of KNN using sklearn</b>	<b>12</b>
<b>8.</b>	<b>Implementation of Logistic Regression using sklearn</b>	<b>12-13</b>
<b>9.</b>	<b>Implementation of K-Means Clustering</b>	<b>13-14</b>
<b>10.</b>	<b>Performance analysis of Classification Algorithms on a specific dataset (Mini Project)</b>	<b>14-16</b>

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

### **Program:**

```
import numpy as np
import statistics as stats
a=[1,2,3,4,5,9,3,9]
Mean=np.mean(a)
print("Mean is:",Mean)
Median=np.median(a)
print("Median is:",Median)
Mode=stats.mode(a)
print("Mode is:",Mode)
Standard_deviation=np.std(a)
print("Standard deviation is:",Standard_deviation)
Variance=np.var(a)
print("Variance is:",Variance)
```

### **Output:**

```
Mean is: 4.5
Median is: 3.5
Mode is: 3
Standard deviation is: 2.8284271247461903
Variance is: 8.0
```

## 2. Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

### **a) Statistics Library**

The statistics library in Python provides functions for mathematical statistics calculations, like mean, median, mode, variance, and standard deviation.

### **Functions:**

- ❖ **mean()**: Returns the arithmetic mean of data.
- ❖ **median()**: Returns the median (middle value) of data.
- ❖ **mode()**: Returns the most common data point.
- ❖ **variance()**: Calculates the variance of the data.
- ❖ **stdev()**: Calculates the standard deviation of data.

### **Program:**

```
import statistics
data = [10, 20, 30, 40, 50]
```

```
print("Mean:", statistics.mean(data))
print("Median:", statistics.median(data))
print("Mode:", statistics.mode([1, 2, 2, 3]))
print("Variance:", statistics.variance(data))
print("Standard Deviation:", statistics.stdev(data))
```

**Output:**

Mean: 30  
Median: 30  
Mode: 2  
Variance: 250  
Standard Deviation: 15.811388300841896

**b) Math Library:**

The math library provides functions for performing basic mathematical operations such as logarithms, trigonometry, factorials, powers, and constants like pi.

**Functions:**

- ❖ **sqrt()**: Returns the square root of a number.
- ❖ **log()**: Returns the natural logarithm (log base e).
- ❖ **pow()**: Returns the value of x raised to the power y.
- ❖ **factorial()**: Returns the factorial of a number.
- ❖ **sin(), cos(), tan()**: Trigonometric functions.

**Program:**

```
import math
print("Square root of 25:", math.sqrt(25))
print("Log of 100:", math.log(100))
angle = math.pi / 4
print("Sine of 45 degrees:", math.sin(angle))
print("Cosine of 45 degrees:", math.cos(angle))
print("Pi value:", math.pi)
```

**Output:**

Square root of 25: 5.0  
Log of 100: 4.605170185988092  
Sine of 45 degrees: 0.7071067811865475  
Cosine of 45 degrees: 0.7071067811865476  
Pi value: 3.141592653589793

### c) NumPy Library:

NumPy (Numerical Python) is a powerful library for performing numerical and matrix operations. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

#### **Features:**

- ❖ **Arrays:** Supports multi-dimensional arrays (1D, 2D, etc.).
- ❖ **Linear Algebra:** Matrix operations, determinants, eigenvalues, etc.
- ❖ **Random Sampling:** Provides methods to generate random numbers.

#### **Program:**

```
import numpy as np
array1 = np.array([1, 2, 3, 4, 5])
print("1D Array:", array1)
array2 = np.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:\n", array2)
print("Sum of array1:", np.sum(array1)) # 15
print("Mean of array2:", np.mean(array2)) # 3.5
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
print("Matrix multiplication:\n", np.dot(matrix1, matrix2))
```

#### **Output:**

```
1D Array: [1 2 3 4 5]
2D Array:
[[1 2 3]
 [4 5 6]]
Sum of array1: 15
Mean of array2: 3.5
Matrix multiplication:
[[19 22]
 [43 50]]
```

### d) SciPy Library:

SciPy (Scientific Python) builds on NumPy and provides additional tools for scientific computing. It is commonly used for optimization, integration, interpolation, eigenvalue problems, and statistics.

**Features:**

- ❖ **Optimization:** Functions for minimizing or maximizing.
- ❖ **Linear Algebra:** Solving linear systems, eigenvalues.
- ❖ **Signal Processing:** FFT, filtering.
- ❖ **Statistics:** PDF, CDF, statistical tests.

**Program:**

```
from scipy import stats
import numpy as np
data = np.random.normal(0, 1, 1000)
mean, variance = stats.norm.fit(data)
print("Mean of the data:", mean)
print("Variance of the data:", variance)
t_statistic, p_value = stats.ttest_1samp(data, 0)
print("T-test statistic:", t_statistic)
print("P-value:", p_value)
```

**Output:**

```
Mean of the data: 0.020227221229950007
Variance of the data: 1.0132553470704482
T-test statistic: 0.6309574379565216
P-value: 0.5282126755151515
```

### 3. Study of Python Libraries for ML application such as Pandas and Matplotlib

#### a) Pandas Library

Pandas is a powerful library used for data manipulation and analysis. It provides data structures like DataFrame and Series for handling and analyzing structured data, which is essential in Machine Learning workflows for loading, processing, and transforming datasets.

**Features:**

**DataFrame:** A 2-dimensional labeled data structure similar to an Excel table or SQL table.

**Data Loading:** Functions to load data from CSV, Excel, databases, and more.

**Data Manipulation:** Tools to clean, transform, and aggregate data.

**Handling Missing Data:** Functions to identify and handle missing data.

**Descriptive Statistics:** Quick statistics on the data (mean, median, mode, etc.).

**Program:**

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'Salary': [50000, 60000, 70000, 80000]}
df = pd.DataFrame(data)
print("DataFrame:\n", df)
print("\nDescriptive Statistics:\n", df.describe())
print("\nAges:\n", df['Age'])
print("\nRows where salary > 60000:\n", df[df['Salary'] > 60000])
df['Bonus'] = [5000, 6000, None, 8000]
print("\nData with missing values:\n", df)
df['Bonus'].fillna(0, inplace=True)
print("\nData after filling missing values:\n", df)
```

**Output:**

DataFrame:

	Name	Age	Salary
0	Alice	25	50000
1	Bob	30	60000
2	Charlie	35	70000
3	David	40	80000

Descriptive Statistics:

	Age	Salary
count	4.000000	4.000000
mean	32.500000	65000.000000
std	6.454972	12909.944487
min	25.000000	50000.000000
25%	28.750000	57500.000000
50%	32.500000	65000.000000
75%	36.250000	72500.000000
max	40.000000	80000.000000

Ages:

0	25
1	30
2	35
3	40

Name: Age, dtype: int64

Rows where salary > 60000:

	Name	Age	Salary
2	Charlie	35	70000
3	David	40	80000

Data with missing values:

	Name	Age	Salary	Bonus
0	Alice	25	50000	5000.0
1	Bob	30	60000	6000.0
2	Charlie	35	70000	NaN
3	David	40	80000	8000.0

Data after filling missing values:

	Name	Age	Salary	Bonus
0	Alice	25	50000	5000.0
1	Bob	30	60000	6000.0
2	Charlie	35	70000	0.0
3	David	40	80000	8000.0

## b) Matplotlib Library

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It's particularly useful in Machine Learning applications for visualizing data distributions, model performance, and more.

### Features:

- ❖ **Plotting:** Line plots, scatter plots, histograms, bar charts, and more.
- ❖ **Customization:** Control over labels, titles, gridlines, legends, colors, etc.
- ❖ **Subplots:** Create multiple plots in the same figure.
- ❖ **Interactivity:** Capabilities for interactive plotting with `plt.show()`.

### Program:

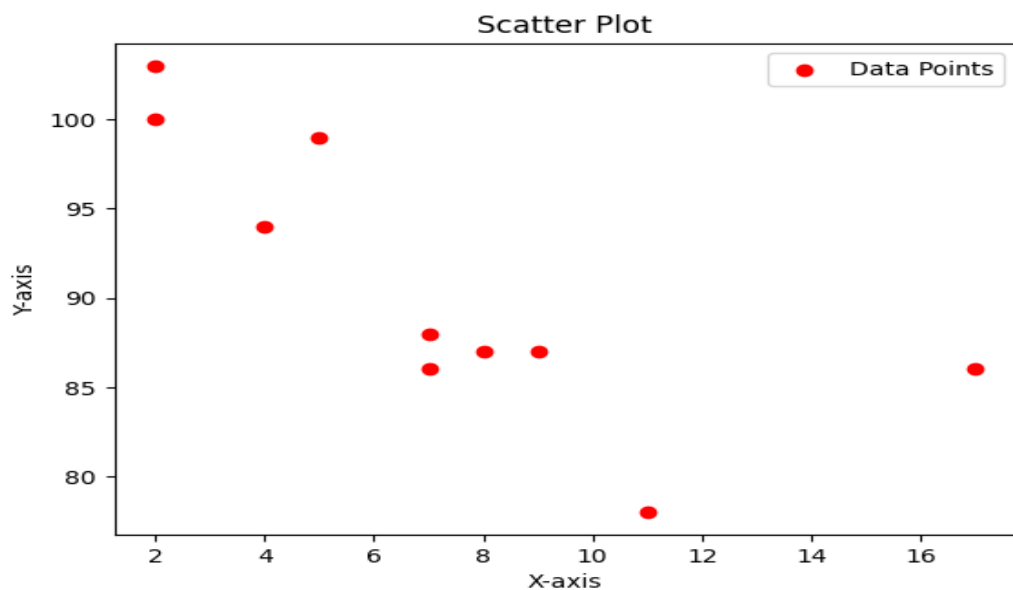
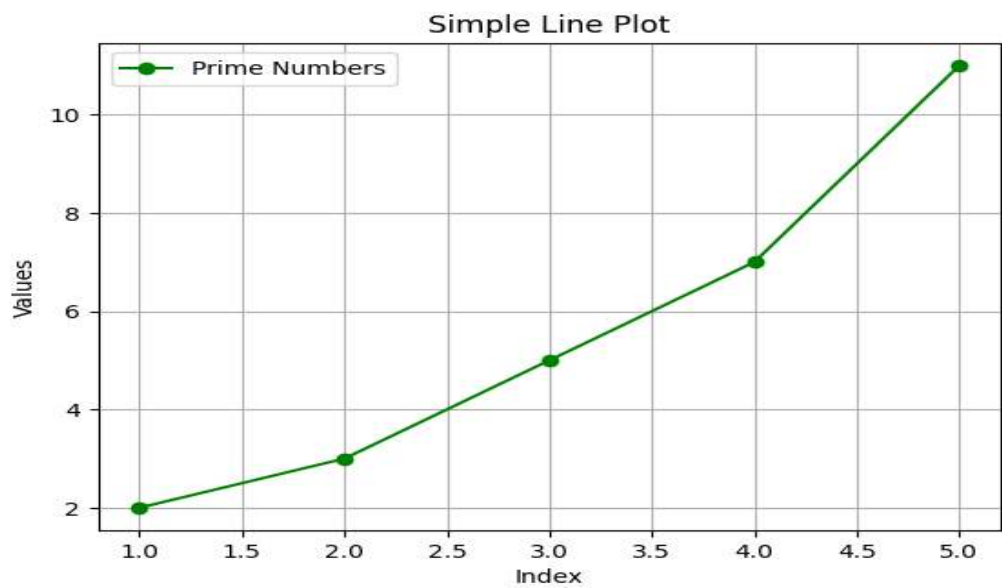
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.plot(x, y, label='Prime Numbers', color='green', marker='o')
plt.xlabel('Index')
plt.ylabel('Values')
plt.title('Simple Line Plot')
plt.grid(True)
plt.legend()
```

```

plt.show()
x_scatter = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11]
y_scatter = [99, 86, 87, 88, 100, 86, 103, 87, 94, 78]
plt.scatter(x_scatter, y_scatter, color='red', label='Data Points')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')
plt.legend()
plt.show()

```

### **Output:**



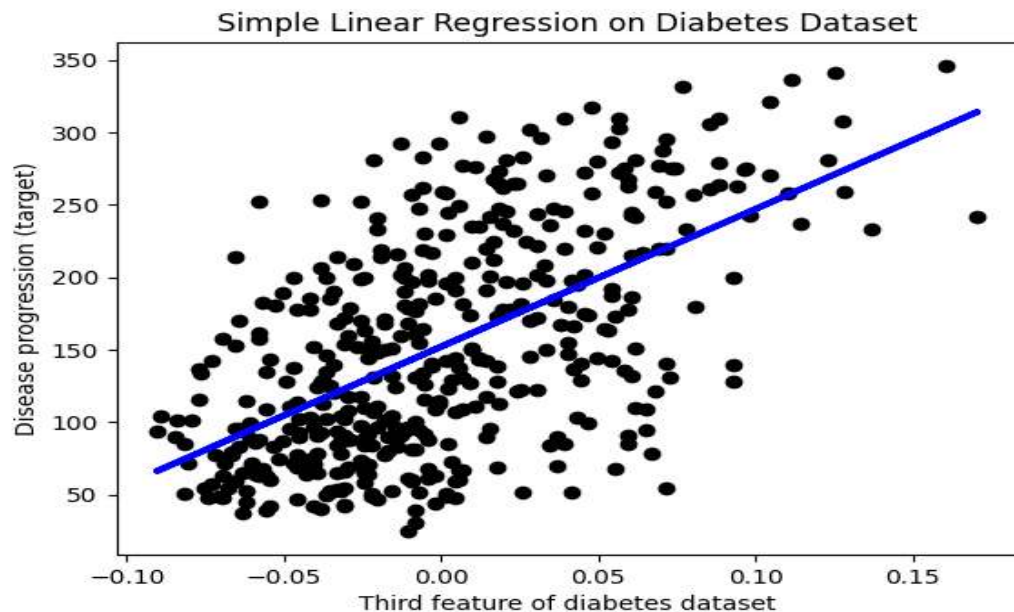
#### 4. Write a Python program to implement Simple Linear Regression

##### Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
diabetes = datasets.load_diabetes()
diabetes_X = diabetes.data[:, np.newaxis, 2]
diabetes_Y = diabetes.target
model = LinearRegression()
model.fit(diabetes_X, diabetes_Y)
diabetes_Y_pred = model.predict(diabetes_X)
mse=mean_squared_error(diabetes_Y,diabetes_Y_pred)
print(mse)
plt.scatter(diabetes_X, diabetes_Y, color='black')
plt.plot(diabetes_X, diabetes_Y_pred, color='blue', linewidth=3)
plt.xlabel("Third feature of diabetes dataset")
plt.ylabel("Disease progression (target)")
plt.title("Simple Linear Regression on Diabetes Dataset")
plt.show()
```

##### Output:

3890.456585461273



## 5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn

### **Program:**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = {
    "Size (sq ft)": [1500, 1700, 1800, 2400, 3000, 3500, 4000, 4500, 5000, 5500],
    "Number of Bedrooms": [3, 3, 4, 4, 5, 5, 5, 6, 6, 7],
    "Age of House (years)": [10, 15, 20, 5, 8, 12, 18, 5, 10, 3],
    "Price ($)": [300000, 350000, 370000, 450000, 600000, 700000, 720000, 850000, 950000, 1100000],
}
df = pd.DataFrame(data)
X = df[["Size (sq ft)", "Number of Bedrooms", "Age of House (years)"]]
y = df["Price ($)"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"\nMean Squared Error (MSE): {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")
print("\nPredicted vs Actual Prices:")
comparison = pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
print(comparison)
```

### **Output:**

Mean Squared Error (MSE): 1194787342.65  
R^2 Score: 0.99

Predicted vs Actual Prices:

	Actual	Predicted
8	950000	940248.545635
1	350000	302099.229411

## 6. Implementation of Decision tree using sklearn and its parameter tuning

### **Program:**

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}
grid_search = GridSearchCV(estimator=dt_classifier,
param_grid=param_grid, cv=5, scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
best_dt_classifier = grid_search.best_estimator_
y_pred = best_dt_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

### **Output:**

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Parameters: {'criterion': 'gini', 'max_depth': None,
'min_samples_leaf': 5, 'min_samples_split': 2}
Accuracy: 1.0
Confusion Matrix:
[[10 0 0]
 [ 0 9 0]
 [ 0 0 11]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

## 7. Implementation of KNN using sklearn

**Program:**

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
features = iris.data
labels = iris.target
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.2, random_state=42)
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
new_sample = [[3, 1.5, 4, 5]]
prediction = classifier.predict(new_sample)
print(f'Predicted class for the new sample: {prediction}')
y_pred = classifier.predict(X_test)
```

**Output:**

Predicted class for the new sample: [2]

## 8. Implementation of Logistic Regression using sklearn

**Program:**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

```

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
logistic_model = LogisticRegression(max_iter=200)
logistic_model.fit(X_train, y_train)
y_pred = logistic_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred,
target_names=iris.target_names)
print(f'Accuracy: {accuracy:.2f}')
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_rep)

```

### **Output:**

Accuracy: 100.00%

Confusion Matrix:

```

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

## **9. Implementation of K-Means Clustering**

### **Program:**

```

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

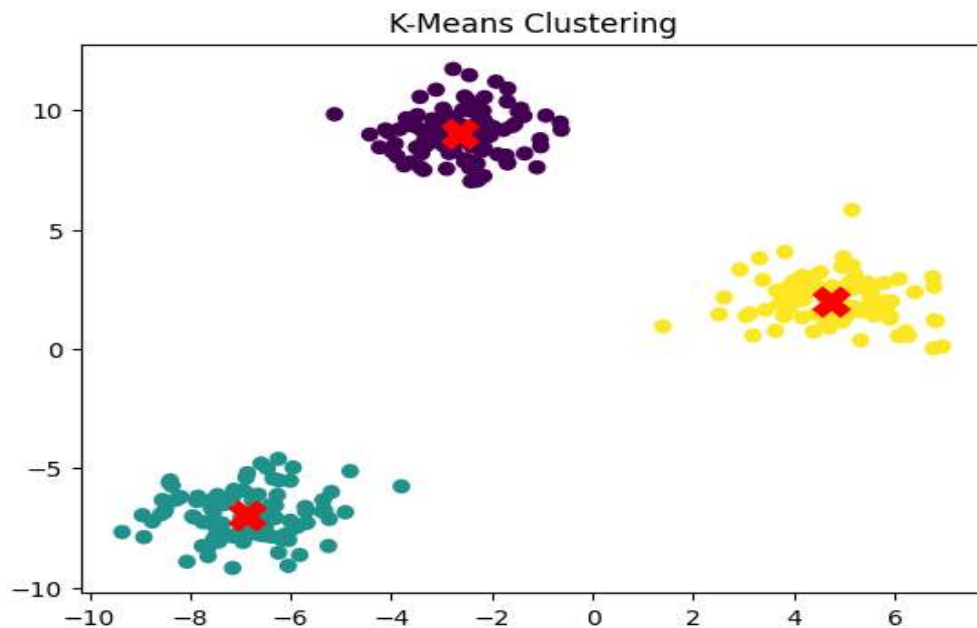
```

```

X, _ = make_blobs(n_samples=300, centers=3, random_state=42)
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s=200, c='red', marker='X')
plt.title("K-Means Clustering")
plt.show()

```

### **Output:**



## **10. Performance analysis of Classification Algorithms on a specific dataset (Mini Project)**

### **Program:**

```

import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
iris = load_iris()

```

```

X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "k-NN": KNeighborsClassifier() }
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\nClassification Report for {name}:")
    print(classification_report(y_test, y_pred,
target_names=iris.target_names))

```

### **Output:**

Classification Report for Logistic Regression:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	0.90	0.90	0.90	10
virginica	0.90	0.90	0.90	10
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

Classification Report for Decision Tree:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.90	0.95	10
virginica	0.91	1.00	0.95	10
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for k-NN:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	0.83	1.00	0.91	10
virginica	1.00	0.80	0.89	10
accuracy			0.93	30
macro avg	0.94	0.93	0.93	30
weighted avg	0.94	0.93	0.93	30