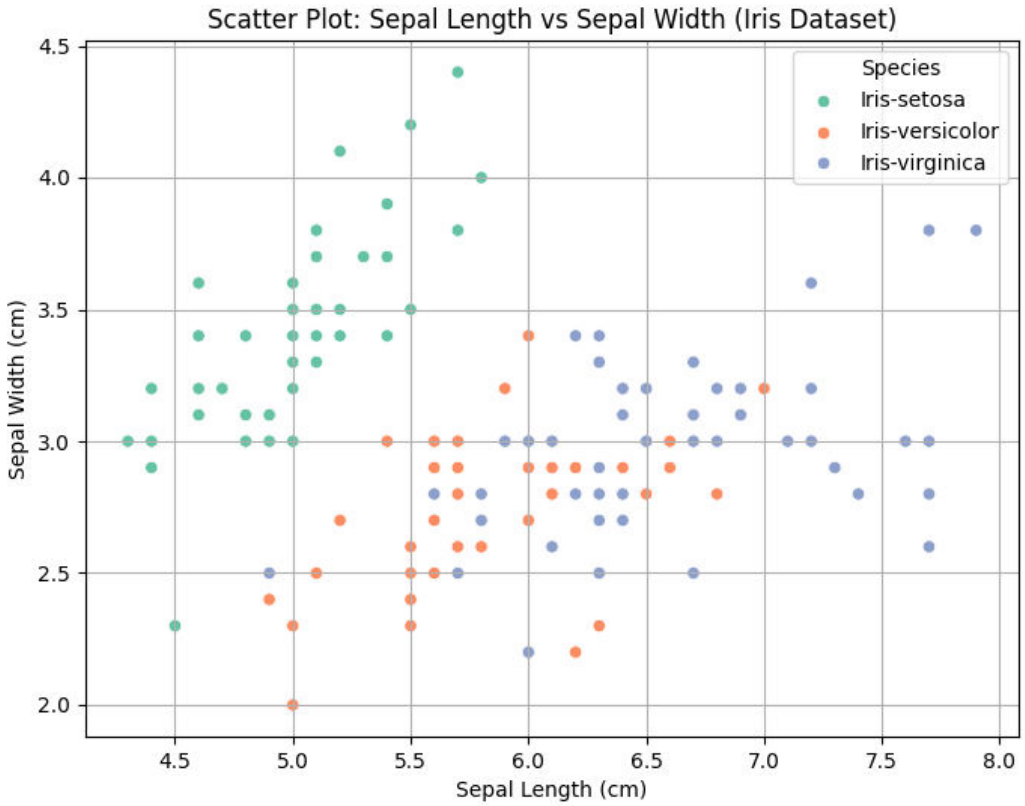
Assignment 1)Write a python program to Prepare Scatter Plot (Use Forge Dataset / Iris Dataset)

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Step 1: Load the Iris dataset
column_names = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']
iris=load_iris()
df=pd.DataFrame(data=iris.data,columns=column_names)
df['species']=iris.target

# Step 2: Display the first 5 rows
print("First 5 rows of the Iris dataset:")
print(df.head())


# Step 3: Create a scatter plot: Sepal Length vs Sepal Width
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='sepal_length', y='sepal_width',
hue='species', palette='Set2')
plt.title('Scatter Plot: Sepal Length vs Sepal Width (Iris Dataset)')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Species')
plt.grid(True)
plt.show()
```

output:

First 5 rows of the Iris dataset:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa



Assignment 2)Write a python program to find all null values in a given data set and remove them.
Create your own dataset.

```
import pandas as pd
import numpy as np

# Creating a sample DataFrame with missing values
data = {
    'School ID': [101, 102, 103, np.nan, 105, 106, 107, 108],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank', 'Grace',
'Henry'],
    'Address': ['123 Main St', '456 Oak Ave', '789 Pine Ln', '101 Elm St',
np.nan, '222 Maple Rd', '444 Cedar Blvd', '555 Birch Dr'],
    'City': ['Los Angeles', 'New York', 'Houston', 'Los Angeles', 'Miami',
np.nan, 'Houston', 'New York'],
    'Subject': ['Math', 'English', 'Science', 'Math', 'History', 'Math',
'Science', 'English'],
    'Marks': [85, 92, 78, 89, np.nan, 95, 80, 88],
    'Rank': [2, 1, 4, 3, 8, 1, 5, 3],
    'Grade': ['B', 'A', 'C', 'B', 'D', 'A', 'C', 'B']
}

df = pd.DataFrame(data)
print("Sample DataFrame:")
print(df)
```

df_cleaned = df.dropna() **#this method is used to remove null values from given dataset**

 # Displaying the DataFrame after removing missing values
print("\nDataFrame after removing rows with missing values:")
print(df_cleaned)

output:

Sample DataFrame:

	School ID	Name	Address	City	Subject	Marks	Rank	Grade
0	101.0	Alice	123 Main St	Los Angeles	Math	85.0	2	B
1	102.0	Bob	456 Oak Ave	New York	English	92.0	1	A
2	103.0	Charlie	789 Pine Ln	Houston	Science	78.0	4	C
3	NaN	David	101 Elm St	Los Angeles	Math	89.0	3	B
4	105.0	Eva	NaN	Miami	History	NaN	8	D
5	106.0	Frank	222 Maple Rd	NaN	Math	95.0	1	A
6	107.0	Grace	444 Cedar Blvd	Houston	Science	80.0	5	C
7	108.0	Henry	555 Birch Dr	New York	English	88.0	3	B

DataFrame after removing rows with missing values:

	School ID	Name	Address	City	Subject	Marks	Rank	Grade
0	101.0	Alice	123 Main St	Los Angeles	Math	85.0	2	B
1	102.0	Bob	456 Oak Ave	New York	English	92.0	1	A
2	103.0	Charlie	789 Pine Ln	Houston	Science	78.0	4	C
6	107.0	Grace	444 Cedar Blvd	Houston	Science	80.0	5	C
7	108.0	Henry	555 Birch Dr	New York	English	88.0	3	B

Assignment 3) Write a python program the Categorical values in numeric format for a given dataset.(iris dataset)

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import LabelEncoder

# Step 1: Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Step 2: Add the species column using human-readable names
df['species'] = [iris.target_names[i] for i in iris.target]

# Step 3: Label Encode the 'species' column
le = LabelEncoder()
df['species_encoded'] = le.fit_transform(df['species'])

# Step 4: Display the original and encoded data
print("Original DataFrame with Categorical 'species':")
print(df[['species']].head())

print("\nDataFrame after Label Encoding:")
print(df[['species', 'species_encoded']].head())

# Step 5: Show label mapping
print("\nLabel Encoding Mapping:")
for class_label, encoded_val in zip(le.classes_,
le.transform(le.classes_)):
```

```
    print(f"{class_label} → {encoded_val}")
```

output:

```
Original DataFrame with Categorical 'species':
  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa

DataFrame after Label Encoding:
  species  species_encoded
0  setosa               0
1  setosa               0
2  setosa               0
3  setosa               0
4  setosa               0

Label Encoding Mapping:
setosa → 0
versicolor → 1
virginica → 2
```

Assignment 4)Write a python program to implement simple Linear Regression for predicting house price .

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

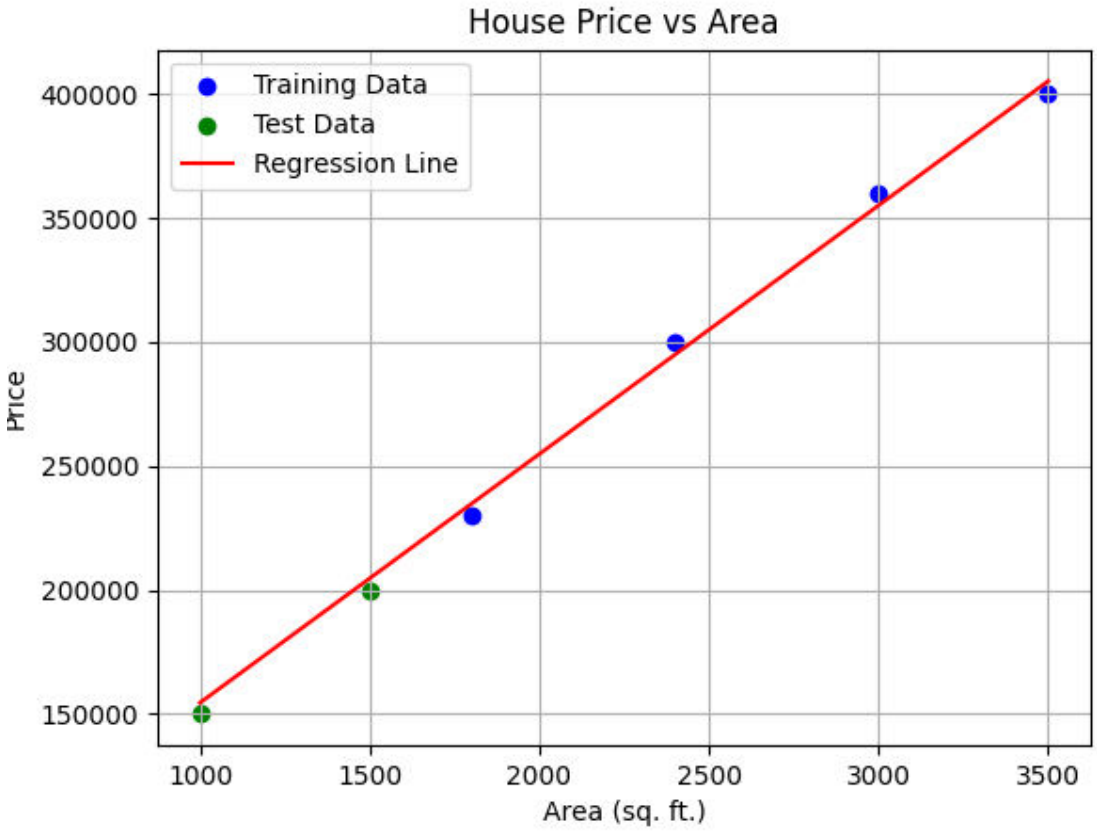
```
from sklearn.metrics import r2_score

# 1. Manually creating the dataset
data = {
    'Area': [1000, 1500, 1800, 2400, 3000, 3500],
    'Price': [150000, 200000, 230000, 300000, 360000, 400000]
}
df = pd.DataFrame(data)

# 2. Define features and target
X = df[['Area']]
y = df['Price']

# 3. Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 4. Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Predict price for 2000 sq. ft.
predicted_price = model.predict(pd.DataFrame([[2000]],
columns=['Area']))
print("Predicted Price for 2000 sq. ft.:", predicted_price[0])

# 6. Predict on the test set and compute R²
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R-squared (R²) Score:", r2)

# 7. Plot the training data, regression line, and test data
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
```

```
plt.xlabel('Area (sq. ft.)')
plt.ylabel('Price')
plt.title('House Price vs Area')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:

Predicted Price for 2000 sq. ft.: 254792.6267281106

R-squared (R^2) Score: 0.9666885165443214



Assignment 5) Write a python program to implement multiple Linear Regression for a given dataset.
(Use 50Startups.csv dataset)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import r2_score

# Step 1: Load the dataset
dataset = pd.read_csv("D:/Teaching Material/Machine
Learning/Assignment 5/50_Startups.csv")
print(dataset.head())

# Step 2: Separate features (X) and target (y)
X = dataset.iloc[:, :-1].values   # All columns except the last
y = dataset.iloc[:, -1].values    # Only the last column (Profit)

# Step 3: Encode categorical data (State column)
ct = ColumnTransformer(
   transformers=[('encoder', OneHotEncoder(), [3])],  # Column 3 is
'State'
   remainder='passthrough'
)
```

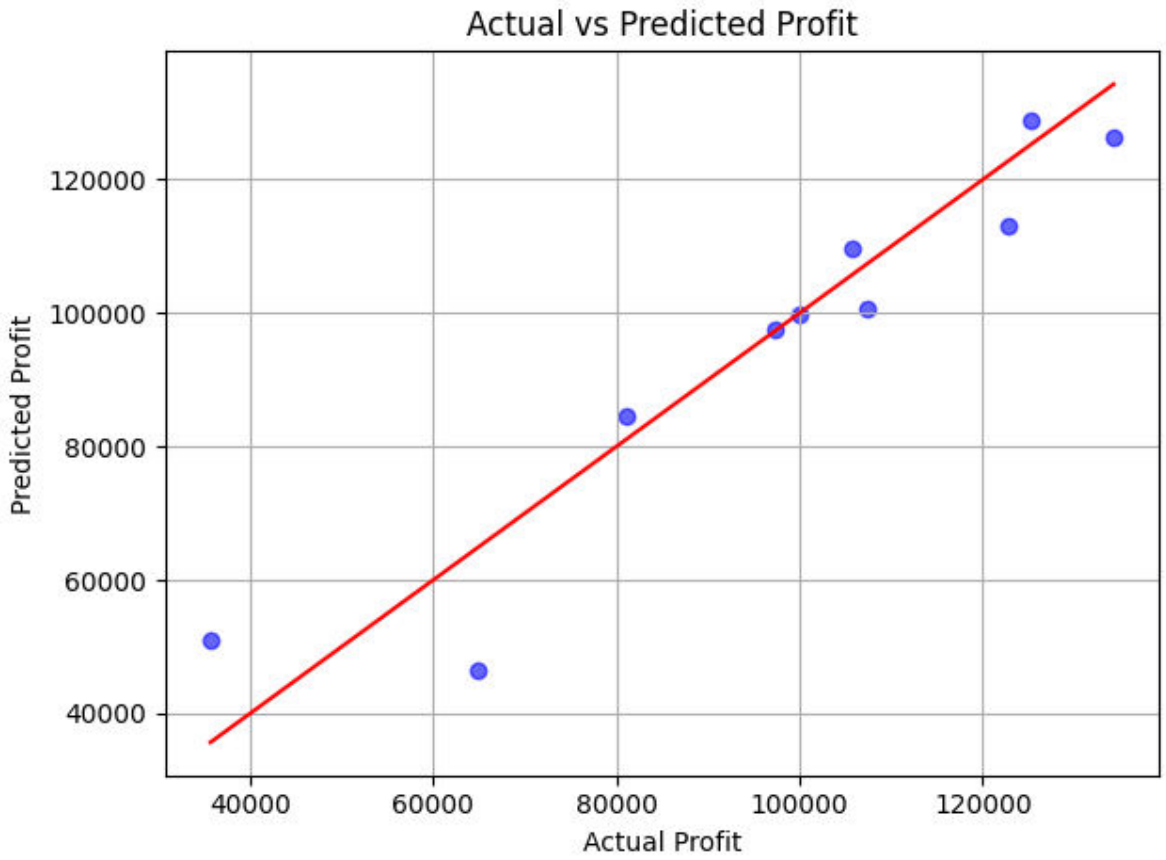
```
X = np.array(ct.fit_transform(X))  # Convert to NumPy array after encoding

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Step 5: Train the Multiple Linear Regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Step 6: Predict the results on the test set
y_pred = regressor.predict(X_test)

# Step 7: Compare predicted and actual values
comparison_df = pd.DataFrame({
    'Actual Profit': y_test,
    'Predicted Profit': y_pred
})
print(comparison_df)

#Step 8: Evaluate model performance using R² score
print("\nR² Score:", r2_score(y_test, y_pred))

# Optional: Plot predicted vs actual
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Profit')
plt.ylabel('Predicted Profit')
plt.title('Actual vs Predicted Profit')
plt.grid(True)
plt.tight_layout()
plt.show()
```

output:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

	Actual Profit	Predicted Profit
0	134307.35	126362.879083
1	81005.76	84608.453836
2	99937.59	99677.494252
3	64926.08	46357.460686
4	125370.37	128750.482885
5	35673.41	50912.417419
6	105733.54	109741.350327
7	107404.34	100643.242816
8	97427.84	97599.275746
9	122776.86	113097.425244

R^2 Score: 0.8987266414319911



Assignment 6) Write a python program to implement Polynomial Regression for given dataset. (use salary_positions.csv dataset)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

 # Load dataset

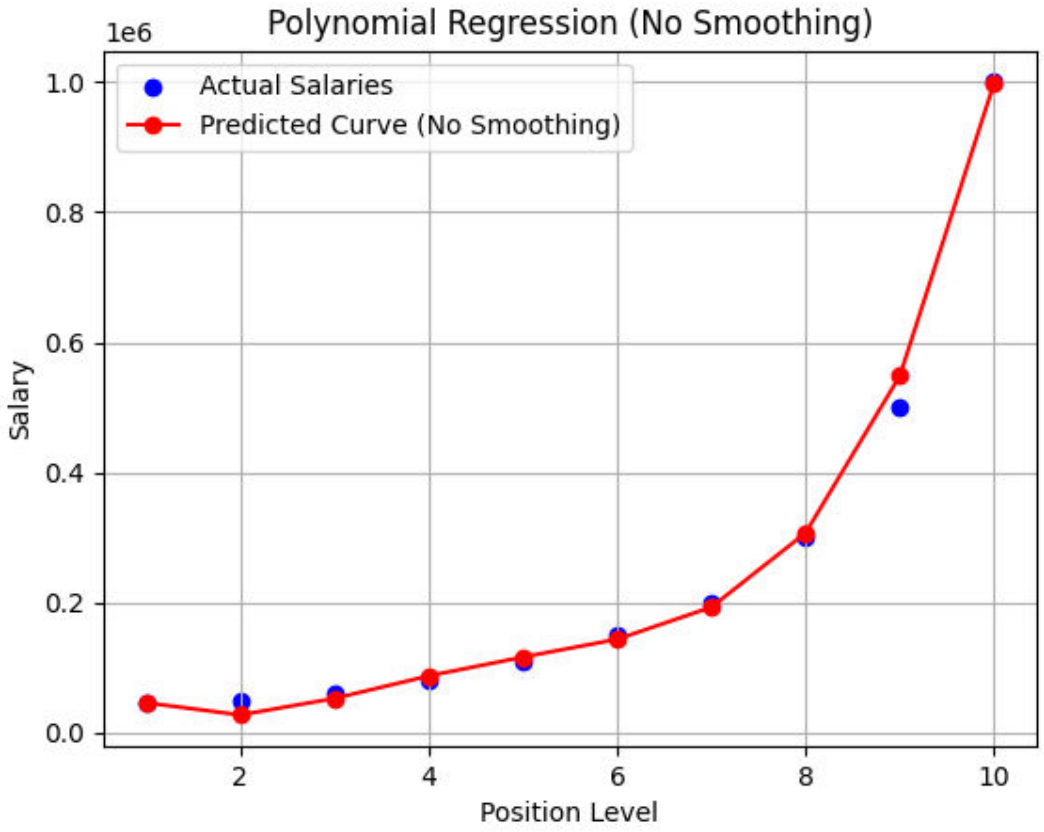
```
data = pd.read_csv('D:\\Teaching Material\\Machine
Learning\\Assignment 6 Polynomial
Regression\\salary_positions.csv')
X = data[['Level']].values
y = data['Salary'].values
```

```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Polynomial features (degree 4)
poly = PolynomialFeatures(degree=4)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Train model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Predict on test and all data
y_test_pred = model.predict(X_test_poly)
y_all_pred = model.predict(poly.transform(X))

# Evaluation
print("R² Score:", r2_score(y_test, y_test_pred))

# Predict for Level 11 and 12
salary_11 = model.predict(poly.transform([[11]]))[0]
salary_12 = model.predict(poly.transform([[12]]))[0]
print("Predicted Salary for Level 11:", round(salary_11, 2))
print("Predicted Salary for Level 12:", round(salary_12, 2))

# Plot without smoothing
plt.scatter(X, y, color='blue', label='Actual Salaries')
plt.plot(X, y_all_pred, color='red', marker='o', label='Predicted Curve
(No Smoothing)')
plt.title('Polynomial Regression (No Smoothing)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

Output:

R^2 Score: 0.9714666803843249
Predicted Salary for Level 11: 1759103.68
Predicted Salary for Level 12: 2949328.94



Assignment 7(a): Write a python program to implement Logistic Regression for Iris dataset

 # Logistic Regression on Iris Dataset

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

```
# 1. Load the dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

# 2. Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Create and train Logistic Regression model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# 4. Make predictions
y_pred = model.predict(X_test)

# 5. Evaluate accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Print classification report
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred,
target_names=iris.target_names))


# 6. Test prediction on a sample
sample = [[5.1, 3.5, 1.4, 0.2]]  # Example flower features
pred_class = iris.target_names[model.predict(sample)[0]]
print("\nSample Prediction:", pred_class)
```

Output:

Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	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

Sample Prediction: setosa

Assignment 7(b): Write a python program to Implement Naïve Bayes.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
columns = ["Pregnancies", "Glucose", "BloodPressure",
"SkinThickness",
        "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
"Outcome"]
```

```
data = pd.read_csv("D:\Teaching Material\Machine Learning\Assignment7(b)\diabetes_dataset.csv",header=None,names=columns)

# Show first 5 rows
print("First 5 rows of dataset:")
print(data.head())

X = data.drop("Outcome", axis=1)   # Drop Outcome column → features
y = data["Outcome"]

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Gaussian Naive Bayes model
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Sample prediction:
# Format: [Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age]
sample = [[6,148,72,35,0,33.6,0.627,50]]
```

```
prediction = model.predict(sample)

if prediction[0] == 1:
    print("\nThe sample input is predicted to HAVE diabetes.")
else:
    print("\nThe sample input is predicted to NOT have diabetes.")
```

OUTPUT:

First 5 rows of dataset:

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

Accuracy: 0.7662337662337663

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.80	0.81	99
1	0.66	0.71	0.68	55
accuracy			0.77	154
macro avg	0.75	0.75	0.75	154
weighted avg	0.77	0.77	0.77	154

The sample input is predicted to HAVE diabetes.

Assignment 8: Write a python program to Implement Decision Tree whether or not to play tennis

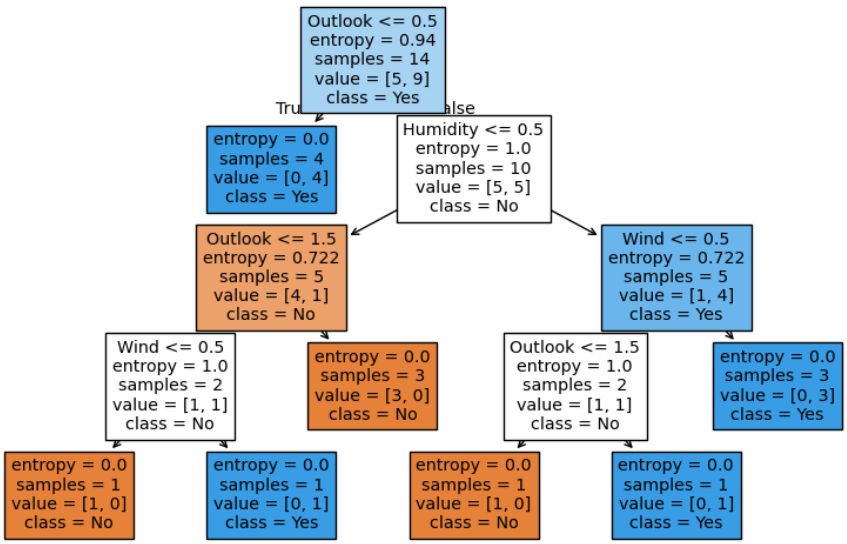
```
# Step 1: Import libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Step 2: Load dataset from local CSV file
df = pd.read_csv("D:\Teaching Material\Machine Learning\Assignment 8\Play_tennis.csv")

# Step 3: Encode categorical variables
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Step 4: Split features (X) and target (y)
X = df[['Outlook','Temperature','Humidity','Wind']]
y = df['Play Tennis']

# Step 5: Train Decision Tree
model = DecisionTreeClassifier(criterion='entropy')
```

```
model.fit(X, y)
```

 # Step 6: Visualize Decision Tree

```
plt.figure(figsize=(10,6))
plot_tree(model,
          feature_names=['Outlook','Temperature','Humidity','Wind'],
          class_names=['No','Yes'],
          filled=True)
plt.show()
```

OUTPUT:



Assignment(9): Write a python program to Implement Random Forest for iris Datset

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# 1. Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# 2. Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 3. Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 4. Predictions
y_pred = model.predict(X_test)
```

```
# 5. Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# 6. Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)

# 7. Prediction for a new flower
sample = [[5.0, 3.6, 1.4, 0.2]]
print("\nPrediction for sample:",
iris.target_names[model.predict(sample)][0])
```

output:

```
Accuracy: 1.0

Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Prediction for sample: setosa
```

Assignment (10): Write a python program to implement linear SVM. (use wine dataset)

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix

# 1. Load dataset
wine = datasets.load_wine()
X = wine.data
y = wine.target

# 2. Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 3. Train Linear SVM
model = SVC(kernel='linear', C=1.0, random_state=42)
model.fit(X_train, y_train)

# 4. Predictions
y_pred = model.predict(X_test)

# 5. Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
# 6. Example prediction
sample = [X_test[0]]
print("\nPrediction for sample:",
wine.target_names[model.predict(sample)][0])
```

OUTPUT:

```
Accuracy: 0.9814814814814815

Confusion Matrix:
 [[19  0  0]
 [ 0 20  1]
 [ 0  0 14]]

Prediction for sample: class_0
```

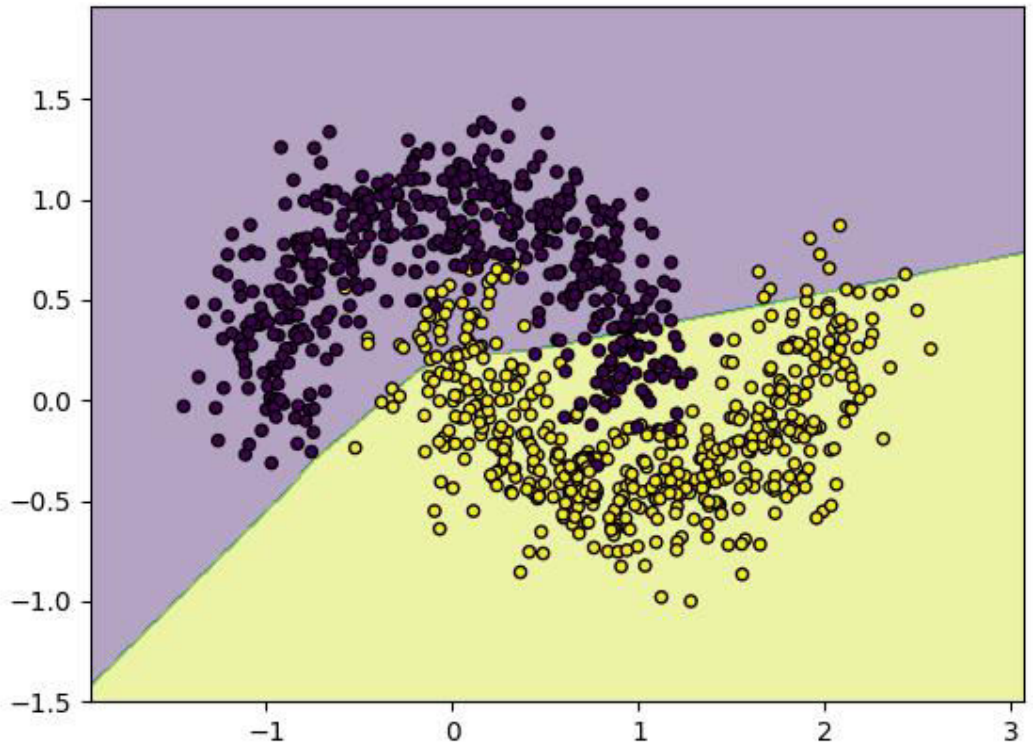
Assignment(11): Write a python program to find Decision boundary by using a neural network with 10 hidden units on two moons dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

```
# 1. Generate two moons dataset
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 2. Define and train Neural Network with 10 hidden units
model = MLPClassifier(hidden_layer_sizes=(10,), activation='relu', solver='adam', max_iter=2000, random_state=42)
model.fit(X_train, y_train)

# 3. Plot decision boundary
def plot_decision_boundary(model, X, y):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    h = 0.01
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
```

```
    plt.show()

plot_decision_boundary(model, X, y)
```

OUTPUT:



Assignment(12): Write a python program to generate frequent itemset and association rule by applying apriori algorithm on Market basket dataset

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
```

```
from mlxtend.preprocessing import TransactionEncoder

# Sample Market Basket Dataset (replace with your actual data)
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

# Convert the dataset into a one-hot encoded DataFrame
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Generate frequent itemsets using Apriori
frequent_itemsets = apriori(df, min_support=0.6,
use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.7,num_itemsets=5)

# Display frequent itemsets
print("Frequent Itemsets:")
```

```
print(frequent_itemsets)
```

 # Display association rules

```
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

OUTPUT:

Frequent Itemsets:

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Eggs, Onion)
7	0.6	(Milk, Kidney Beans)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Kidney Beans, Yogurt)
10	0.6	(Kidney Beans, Eggs, Onion)

Association Rules:

	antecedents	consequents	support	confidence	lift
0	(Kidney Beans)	(Eggs)	0.8	0.80	1.00

1	(Eggs)	(Kidney Beans)	0.8	1.00	1.00
2	(Eggs)	(Onion)	0.6	0.75	1.25
3	(Onion)	(Eggs)	0.6	1.00	1.25
4	(Milk)	(Kidney Beans)	0.6	1.00	1.00
5	(Onion)	(Kidney Beans)	0.6	1.00	1.00
6	(Yogurt)	(Kidney Beans)	0.6	1.00	1.00
7	(Kidney Beans, Eggs)	(Onion)	0.6	0.75	1.25
8	(Kidney Beans, Onion)	(Eggs)	0.6	1.00	1.25
9	(Eggs, Onion)	(Kidney Beans)	0.6	1.00	1.00
10	(Eggs)	(Kidney Beans, Onion)	0.6	0.75	1.25
11	(Onion)	(Kidney Beans, Eggs)	0.6	1.00	1.25

Assignment(13): Write a python program to implement k-nearest Neighbours ML algorithm to build prediction model (Use Forge Dataset)

```
import matplotlib.pyplot as plt
import mglearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# 1. Load the Forge dataset
X, y = mglearn.datasets.make_forge()
```

```
# 2. Split dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)

# 3. Create KNN model (k=3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# 4. Predictions
y_pred = knn.predict(X_test)

# 5. Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# 6. Confusion Matrix
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# 7. Visualization of decision boundary
mglearn.plots.plot_2d_separator(knn, X, fill=True, eps=0.5,
alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1"], loc=4)
plt.xlabel("Feature 1")
```

```
plt.ylabel("Feature 2")
plt.title("KNN Decision Boundary (k=3)")
plt.show()
```

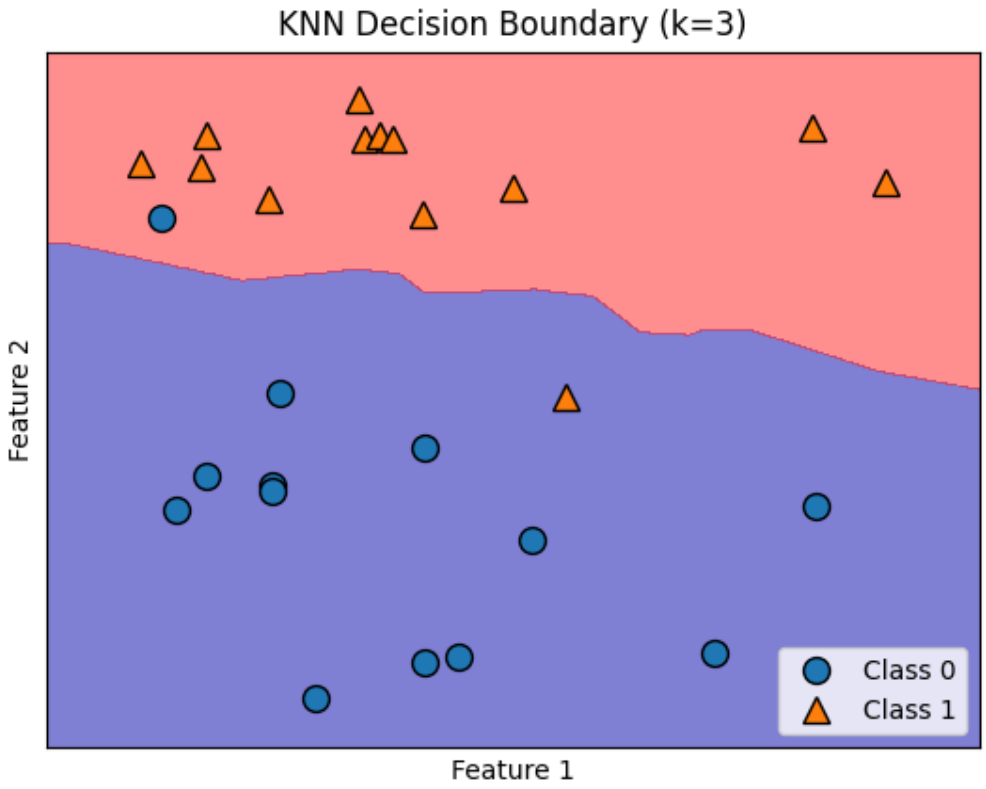
OUTPUT:

Accuracy: 0.8571428571428571

Confusion Matrix:

[[1 0]

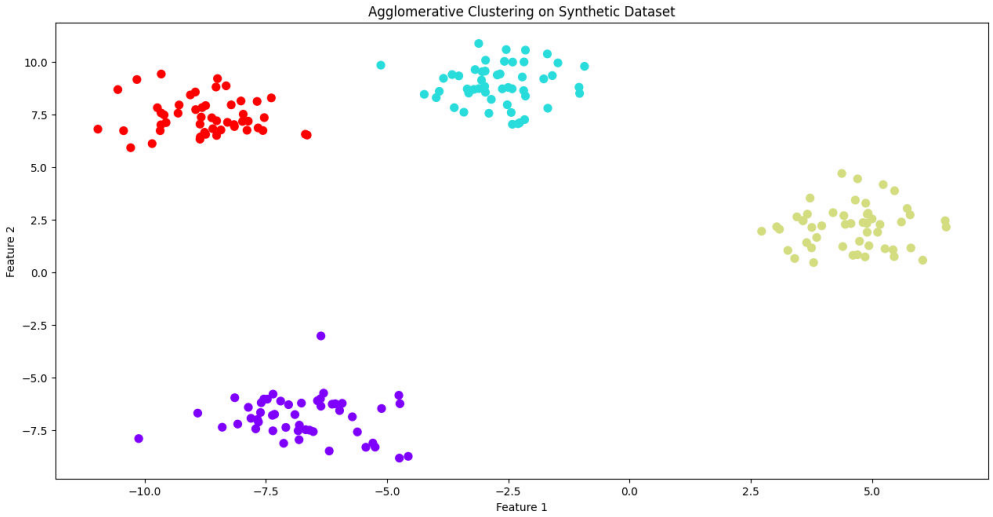
[1 5]]



Assignment(14): Write a python program to implement Agglomerative clustering on a synthetic dataset.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

# Step 1: Generate synthetic dataset
X, y_true = make_blobs(n_samples=200, centers=4, cluster_std=1.0,
random_state=42)

# Step 2: Apply Agglomerative Clustering
agg_clust = AgglomerativeClustering(n_clusters=4, linkage='ward')
y_pred = agg_clust.fit_predict(X)

# Step 3: Plot the clusters
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='rainbow', s=50)
plt.title("Agglomerative Clustering on Synthetic Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

OUTPUT:


Agglomerative Clustering on Synthetic Dataset
Feature 2
Feature 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0