DSBD LAB

Assignment 6

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load dataset
url = "D:\DEVENDRA\VS Code\Assignment\TY\sem 6\dsbdcode\iris.data.csv"
cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
df = pd.read_csv(url, header=None, names=cols)

# Standardize features and apply PCA
x = StandardScaler().fit_transform(df[cols[:-1]])
pca = PCA(n_components=2).fit_transform(x)
df_pca = pd.DataFrame(pca, columns=['PC1', 'PC2'])
df_pca['class'] = df['class']

# Plot
colors = {'Iris-setosa': 'r', 'Iris-versicolor': 'g', 'Iris-virginica': 'b'}
for label, color in colors.items():
    subset = df_pca[df_pca['class'] == label]
    plt.scatter(subset['PC1'], subset['PC2'], c=color, label=label, s=50)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.grid()
plt.show()
```

Assignment 7

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset from local CSV
df = pd.read_csv(r"D:\DEVENDRA\VS Code\Assignment\TY\sem
6\dsbdcode\HousingData.csv")

# Drop missing values
df = df.dropna()

# Prepare X and y using DataFrame and Series
X = pd.DataFrame(df.iloc[:, :-1], columns=df.columns[:-1])
y = pd.Series(df.iloc[:, -1])

# Split data 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)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")

# Plot actual vs predicted
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Median House Value")
plt.ylabel("Predicted Median House Value")
plt.title("Actual vs Predicted Median House Value")
plt.show()
```

Assignment 8

```
import pandas as pd, numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# Load & preprocess
url = "D:\DEVENDRA\VS Code\Assignment\TY\sem
6\dsbdcode\iris.data.csv"
cols = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width', 'species']
df = pd.read_csv(url, header=None, names=cols)
le = LabelEncoder()
df['species'] = le.fit_transform(df['species'])  # setosa=0,
versicolor=1, virginica=2

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, :-1],
df['species'], test_size=0.3, random_state=42)
model = KNeighborsClassifier(n_neighbors=3).fit(X_train, y_train)
y_pred = model.predict(X_test)

# Treat class 0 (setosa) as positive, others as negative
target_class = 0
y_test_binary = (y_test == target_class).astype(int)
y_pred_binary = (y_pred == target_class).astype(int)

# Compute confusion matrix
cm = confusion_matrix(y_test_binary, y_pred_binary)
TP = cm[1, 1]
FP = cm[0, 1]
TN = cm[0, 0]
FN = cm[1, 0]

# Print results
print("Confusion Matrix:\n", cm)
print("\nTrue Positives (TP):", TP)
print("False Positives (FP):", FP)
print("True Negatives (TN):", TN)
print("False Negatives (FN):", FN)

acc = accuracy_score(y_test_binary, y_pred_binary)
print(f"\nAccuracy: {acc:.1f}")
print(f"Error Rate: {1 - acc:.1f}")
print("Precision:", round(precision_score(y_test_binary,
y_pred_binary), 1))
```

```
print("Recall:", round(recall_score(y_test_binary, y_pred_binary), 1))
print("F1-Score:", round(f1_score(y_test_binary, y_pred_binary), 1))
```

Assignment 9

```
import numpy as np, matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(load_iris().data)

# Elbow method
inertia = [KMeans(n_clusters=k, random_state=42, n_init=10).fit(X).inertia_ for k in range(1, 11)]
plt.plot(range(1, 11), inertia, marker='o'), plt.xlabel('k'), plt.ylabel('Inertia'), plt.title('Elbow Method'), plt.show()

# KMeans with k=3
labels = KMeans(n_clusters=3, random_state=42, n_init=10).fit_predict(X)

# Cluster visualization (first 2 features)
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolor='k')
plt.xlabel('Feature 1'), plt.ylabel('Feature 2'), plt.title('K-Means Clustering (k=3)'), plt.show()
```