Data science homework 8 - Mayank Sharma - ms14662

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, auc
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Logistic Regression Section
print("=== Logistic Regression ===")

# Example Dataset Placeholder
data = pd.DataFrame({
    'feature1': np.random.randn(1000),
    'feature2': np.random.choice([0, 1], size=1000),
    'feature3': np.random.choice(['A', 'B', 'C'], size=1000),
    'target': np.random.choice([0, 1], size=1000)
})

# Preprocess Data
numeric_features = ['feature1']
categorical_features = ['feature3']

target = data['target']
X = data.drop(columns=['target'])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size=0.2, random_state=42)

# Logistic Regression Pipeline
logreg = Pipeline([
    ('preprocessor', preprocessor),
```

```
    ('classifier', LogisticRegression())
])

logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_proba = logreg.predict_proba(X_test)[:, 1]

# 1. Try Different Thresholds
print("\n1. Threshold Impact Analysis")
thresholds = [0.3, 0.5, 0.7]  # Keeping it simple with fewer thresholds
for thresh in thresholds:
    y_thresh = (y_proba >= thresh).astype(int)
    acc = accuracy_score(y_test, y_thresh)
    prec = precision_score(y_test, y_thresh)
    rec = recall_score(y_test, y_thresh)
    print(f"Threshold: {thresh:.1f} | Accuracy: {acc:.2f} | Precision: {prec:.2f} | Recall: {rec:.2f}")

# 2. ROC Curve
print("\n2. ROC Curve")
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

# Clustering Section
print("\n=== Clustering ===")

# Using scaled numeric features
print("\n1. K-Means Analysis")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(data[['feature1', 'feature2']])

# Try a few values of k
for k in [2, 3, 4]:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)
    inertia = kmeans.inertia_
    silhouette = silhouette_score(X_scaled, labels)
```

```
    print(f"k: {k} | Inertia: {inertia:.2f} | Silhouette Score: {silhouette:.2f}")

# Plot for Understanding k
k_values = [2, 3, 4]
inertias = []
silhouettes = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)
    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(X_scaled, labels))

plt.plot(k_values, inertias, marker='o', label='Inertia')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Inertia for Different k')
plt.show()

print("\n2. Food Nutrients Dataset Placeholder")
print("You can repeat the same clustering process with the food nutrients dataset.")
```