**# EXP 2: DATA CLEANING**

import pandas as pd

import seaborn as sns

# Load Titanic dataset

df = sns.load\_dataset('titanic')

print(f"\nShape before cleaning: {df.shape}")

print(df.head(1))

# 1. Check for missing values

print("\nMissing values in each column:")

print(df.isnull().sum())

# 2. Drop columns with too many missing values (optional)

df = df.drop(['deck'], axis=1)  # 'deck' has lots of nulls

# 3. Fill missing values (example: age and embarked)

df['age'] = df['age'].fillna(df['age'].median())

df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])

# 4. Check and remove duplicates

duplicate\_count = df.duplicated().sum()

print(f"\nDuplicate rows found: {duplicate\_count}")

df = df.drop\_duplicates()

# 5. Check data types

print("\nData types:")

print(df.dtypes)

# 6. Final cleaned data preview

print("\nCleaned Data Preview:")

print(df.head(1))

print(f"\nFinal shape: {df.shape}")

**# EXP 3: DATA VISUALIZATION**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = {'Year': [2010, 2011, 2012, 2013, 2014, 2015, 2016],

        'Sales': [200, 240, 300, 350, 420, 500, 600],

        'Profit': [50, 70, 80, 100, 120, 150, 180]}

df = pd.DataFrame(data)

plt.figure(figsize=(4,3))

plt.plot(df['Year'], df['Sales'], marker='o', color='b', label='Sales')

plt.plot(df['Year'], df['Profit'], marker='s', color='g', label='Profit')

plt.title('Sales and Profit Over Time')

plt.xlabel('Year')

plt.ylabel('Amount')

plt.legend()

plt.grid(True)

plt.show()

plt.figure(figsize=(4,3))

df.plot(kind='bar', x='Year', y=['Sales', 'Profit'], color=['blue', 'green'], figsize=(4,3))

plt.title('Sales and Profit Comparison (2010-2016)')

plt.ylabel('Amount')

plt.xlabel('Year')

plt.show()

plt.figure(figsize=(4,3))

sns.scatterplot(data=df, x='Sales', y='Profit', hue='Profit', palette='viridis', size='Profit', sizes=(50, 200))

plt.title('Sales vs Profit')

plt.xlabel('Sales')

plt.ylabel('Profit')

plt.show()

plt.figure(figsize=(4,3))

sns.histplot(df['Sales'], kde=True, color='purple')

plt.title('Sales Distribution')

plt.xlabel('Sales')

plt.ylabel('Frequency')

plt.show()

**# EXP 4: LOGISTIC REGRESSION AND EXPLORE PERFORMANCE EVALUATION METRICS**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import (

    accuracy\_score,

    precision\_score,

    recall\_score,

    f1\_score,

    confusion\_matrix,

    classification\_report,

  )

# Load data

iris = load\_iris()

X = iris.data

y = iris.target

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train logistic regression model

model = LogisticRegression(max\_iter=200)

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluation metrics

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

# Output results

print("Logistic Regression Evaluation Metrics:")

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision (Weighted): {precision:.2f}")

print(f"Recall (Weighted): {recall:.2f}")

print(f"F1 Score (Weighted): {f1:.2f}")

print("\nClassification Report:\n", report)

# Confusion Matrix Visualization

plt.figure(figsize=(4, 3))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

**# EXP 5 To generate synthetic data for class imbalance using the SMOTE**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Load dataset

iris = load\_iris()

X = iris.data

y = iris.target

df = pd.DataFrame(X, columns=iris.feature\_names)

df['target'] = y

# Simulate class imbalance

df\_imbalanced = pd.concat([

    df[df['target'] == 0],

    df[df['target'] == 1].sample(25, random\_state=42),

    df[df['target'] == 2].sample(10, random\_state=42)

])

X\_imbalanced = df\_imbalanced.drop('target', axis=1)

y\_imbalanced = df\_imbalanced['target']

# Visualize class distribution before SMOTE

plt.figure(figsize=(3, 2))

sns.countplot(x=y\_imbalanced, palette='pastel')

plt.title("Class Distribution BEFORE SMOTE")

plt.xlabel("Class")

plt.ylabel("Count")

plt.xticks(ticks=[0, 1, 2], labels=iris.target\_names)

plt.show()

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X\_imbalanced, y\_imbalanced, test\_size=0.3, random\_state=42, stratify=y\_imbalanced)

# Apply SMOTE

smote = SMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(X\_train, y\_train)

# Visualize class distribution after SMOTE

plt.figure(figsize=(3, 2))

sns.countplot(x=y\_resampled, palette='Set2')

plt.title("Class Distribution AFTER SMOTE")

plt.xlabel("Class")

plt.ylabel("Count")

plt.xticks(ticks=[0, 1, 2], labels=iris.target\_names)

plt.show()

# Train model

model = RandomForestClassifier(random\_state=42)

model.fit(X\_resampled, y\_resampled)

# Predictions

y\_pred = model.predict(X\_test)

# Evaluation

print("Classification Report:\n")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(3, 2))

sns.heatmap(cm, annot=True, fmt='d', cmap='BuPu', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

**# EXP 8 : OUTLIER DETECTION**

import numpy as np

from sklearn.neighbors import NearestNeighbors

import matplotlib.pyplot as plt

# Example data points (with a potential outlier at [20, 20])

X = np.array([[5, 5], [6, 5], [5, 6], [6, 6], [5, 7], [7, 5], [8, 6], [5, 8], [6, 7], [6, 8], [20, 20]])

# Define number of neighbors (k)

k = 2

# Fit the k-NN model

neigh = NearestNeighbors(n\_neighbors=k)

neigh.fit(X)

# Calculate distances and indices of k-nearest neighbors

distances, indices = neigh.kneighbors(X)

# Calculate average distance for each point to its k-nearest neighbors

avg\_distances = np.mean(distances, axis=1)

# Print the average distances for debugging

print("Average distances for each point to its k-nearest neighbors:")

print(avg\_distances)

# Set a threshold for outlier detection (e.g., 90th percentile of the average distance)

threshold = np.percentile(avg\_distances, 90)  # Set to 90th percentile for better sensitivity

# Identify outliers based on the threshold

outliers = np.where(avg\_distances > threshold)

# Print outliers detected

print("Outliers detected (indices of outliers):", outliers[0])

# Visualize the data points and outliers

plt.figure(figsize=(4, 3))

plt.scatter(X[:, 0], X[:, 1], label='Data Points', c='blue')

plt.scatter(X[outliers[0], 0], X[outliers[0], 1], color='red', label='Outliers')

plt.legend()

plt.title('k-NN Outlier Detection')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.grid(True)

plt.show()