# Practicle 2

# • Read data from CSV and JSON files into a data frame.

print("Read data from CSV and JSON files into a data frame.")

import pandas as pd

df\_csv=pd.read\_csv('C:\\Users\\VEDANT RAJPUT\\pract\\ds\\p1\\student\_data.csv')

print (df\_csv)

df\_json=pd.read\_json('C:\\Users\\VEDANT RAJPUT\\pract\\ds\\p1\\student\_data.json')

print (df\_json)

# • Perform basic data pre-processing tasks such as handling missing values and outliers.

print('• Perform basic data pre-processing tasks such as handling missing values and outliers.')

'removing rows with missing values'

# df\_csv\_clean=df\_csv.dropna()

# print(df\_csv\_clean)

# # Or fill missing values with a default (e.g., 0)

df\_csv\_filled = df\_csv.fillna(0)

# "sir's method"

df\_csv['Age'].fillna(df\_csv['Age'].mean(), inplace=True)

print(df\_csv)

'removing outliers'

df\_csv\_no\_outliers = df\_csv[(df\_csv['Age']>=18)&(df\_csv['Age']<=100)]

print(df\_csv\_no\_outliers)

# • Manipulate and transform data using functions like filtering, sorting, and grouping

print('\n')

print("• Manipulate and transform data using functions like filtering, sorting, and grouping")

filtered=df\_csv\_no\_outliers[(df\_csv\_no\_outliers["Marks"]>=30)]

print(filtered)

sorted = df\_csv\_no\_outliers.sort\_values(by='Marks', ascending=True)

print(sorted)

group=df\_csv\_no\_outliers.groupby('Gender').size()

print(group)

# Practical 3

# • Apply feature-scaling techniques like standardization and normalization to numerical features.

# • Perform feature dummification to convert categorical variables into numerical representations.

import pandas as pd

from sklearn.preprocessing import StandardScaler,MinMaxScaler

# import pandas as pd

# from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Sample dataset

data = {

    "Age": [25, 30, 35, 40, 28, 32, 38, 45],

    "Income": [50000, 60000, 70000, 80000, 55000, 65000, 75000, 90000],

    "Education": ["Bachelor", "Master", "PhD", "Master", "Bachelor", "Master", "PhD", "Master"],

    "Marital\_Status": ["Single", "Married", "Single", "Married", "Single", "Married", "Single", "Married"]

}

df = pd.DataFrame(data)

# Standardization (Z-score scaling)

scaler\_std = StandardScaler()

df[['Age', 'Income']] = scaler\_std.fit\_transform(df[['Age', 'Income']])

print('\n')

print(df)

# Normalization (Min-Max scaling)

scaler\_norm = MinMaxScaler()

df[['Age', 'Income']] = scaler\_norm.fit\_transform(df[['Age', 'Income']])

print('\n')

print(df)

# Dummification (Convert categorical 'Gender' column)

df = pd.get\_dummies(df, columns=['Education', 'Marital\_Status'], drop\_first=True)

print('\n')

print(df)

# Practical 4

# Hypothesis Testing

# • Formulate null and alternative hypotheses for a given problem.

# • Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chisquare test).

# • Interpret the results and draw conclusions based on the test outcomes.

import pandas as pd

from scipy.stats import chi2\_contingency

# Sample Data

data = {

    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female'],

    'Preferred\_Subject': ['Math', 'Math', 'Science', 'Math', 'Science', 'Science']

}

df = pd.DataFrame(data)

# Create contingency table

contingency\_table = pd.crosstab(df['Gender'], df['Preferred\_Subject'])

print (contingency\_table)

# Hypothesis:

# H₀: Gender and subject preference are independent.

# H₁: Gender and subject preference are NOT independent.

chi2, p, dof, expected = chi2\_contingency(contingency\_table)

print("Chi-square value:", chi2)

print("P-value:", p)

print(f"Degrees of Freedom: {dof}")

print("Expected Frequencies:")

print(expected)

if p < 0.05:

    print(" Reject H₀: Gender and subject preference are related.")

else:

    print(" Fail to reject H₀: Gender and preference are independent.")

# Practical 5

# ANOVA (Analysis of Variance)

# • Perform one-way ANOVA to compare means across multiple groups.

# • Conduct post-hoc tests to identify significant differences between group means

import pandas as pd

from scipy.stats import f\_oneway

from statsmodels.stats.multicomp import pairwise\_tukeyhsd

# Sample data: 3 different classes with marks

data = {

    'Class': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'],

    'Marks': [85, 88, 90, 78, 80, 79, 92, 95, 94]

}

df = pd.DataFrame(data)

# Split the marks into groups

group\_A = df[df['Class'] == 'A']['Marks']

group\_B = df[df['Class'] == 'B']['Marks']

group\_C = df[df['Class'] == 'C']['Marks']

# Perform One-Way ANOVA

f\_stat, p\_value = f\_oneway(group\_A, group\_B, group\_C)

print("F-statistic:", f\_stat)

print("P-value:", p\_value)

if p\_value < 0.05:

    print("❌ Reject H₀: At least one class has a different average.")

    # Perform Tukey's HSD post-hoc test

    posthoc = pairwise\_tukeyhsd(df['Marks'], df['Class'], alpha=0.05)

    print(posthoc)

else:

    print("✅ Fail to reject H₀: No significant difference among the classes.")

# practical 6

# Regression and Its Types

# • Implement simple linear regression using a dataset.

# • Explore and interpret the regression model coefficients and goodness-of-fit measures.

# • Extend the analysis to multiple linear regression and assess the impact of additional predictors

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

"Simple Linear Regression (One input ➝ One output)"

# Sample data: Hours studied vs Marks scored

data = {

    'Hours\_Studied': [1, 2, 3, 4, 5, 6, 7, 8],

    'Marks': [35, 40, 50, 55, 60, 65, 70, 80]

}

df = pd.DataFrame(data)

# Split X (input) and y (output)

X = df[['Hours\_Studied']]

y = df['Marks']

# Create and train the model

model = LinearRegression()

model.fit(X, y)

# Predict and evaluate

predictions = model.predict(X)

print("Intercept (b₀):", model.intercept\_)

print("Coefficient (b₁):", model.coef\_[0])

print("R² Score:", r2\_score(y, predictions))  # Goodness of fit

print("Mean Squared Error:", mean\_squared\_error(y, predictions))

" Multiple Linear Regression (More inputs ➝ One output)"

# More complex data

data = {

    'Hours\_Studied': [1, 2, 3, 4, 5, 6, 7, 8],

    'Sleep\_Hours': [6, 7, 6, 5, 7, 6, 8, 7],

    'Attendance': [70, 75, 80, 82, 85, 88, 90, 95],

    'Marks': [35, 40, 50, 55, 60, 65, 70, 80]

}

df = pd.DataFrame(data)

X = df[['Hours\_Studied', 'Sleep\_Hours', 'Attendance']]

y = df['Marks']

model = LinearRegression()

model.fit(X, y)

predictions = model.predict(X)

print("\nIntercept (b₀):", model.intercept\_)

print("Coefficients (b₁, b₂, b₃):", model.coef\_)

print("R² Score:", r2\_score(y, predictions))

# Practical 7

# Logistic Regression and Decision Tree

# • Build a logistic regression model to predict a binary outcome.

# • Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).

# • Construct a decision tree model and interpret the decision rules for classification.

import pandas as pd

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, classification\_report

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import matplotlib.pyplot as plt

# Sample dataset

data = {

    'Marks': [40, 50, 60, 30, 90, 85, 20, 75],

    'Attendance': [60, 70, 80, 50, 95, 90, 40, 85],

    'Pass': [0, 1, 1, 0, 1, 1, 0, 1]  # Target variable (binary)

}

df = pd.DataFrame(data)

# Step 1: Split input (X) and output (y)

X = df[['Marks', 'Attendance']]

y = df['Pass']

# Step 2: Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Step 3: Build model

model = LogisticRegression()

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

# Step 4: Predict and Evaluate

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

print("Predictions:", y\_pred)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Build decision tree model

tree\_model = DecisionTreeClassifier()

tree\_model.fit(X\_train, y\_train)

# Predict

y\_pred\_tree = tree\_model.predict(X\_test)

# Evaluate

print("Accuracy (Decision Tree):", accuracy\_score(y\_test, y\_pred\_tree))

# Visualize the tree

plt.figure(figsize=(10,6))

tree.plot\_tree(tree\_model, feature\_names=['Marks', 'Attendance'], class\_names=['Fail', 'Pass'], filled=True)

plt.title("Decision Tree")

plt.show()

# Practical 8

# K-Means Clustering

# • Apply the K-Means algorithm to group similar data points into clusters.

# • Determine the optimal number of clusters using elbow method or silhouette analysis.

# • Visualize the clustering results and analyze the cluster characteristics

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

# 1. Sample Data

df = pd.DataFrame({

    'Income': [15, 16, 15.5, 17, 80, 85, 82, 88, 55, 53, 58, 60],

    'Spending': [40, 42, 38, 45, 90, 85, 87, 88, 60, 62, 63, 65]

})

# 2. Standardize the Data

scaled = StandardScaler().fit\_transform(df)

# 3. Elbow & Silhouette Method

for k in range(2, 6):

    model = KMeans(n\_clusters=k, random\_state=0)

    labels = model.fit\_predict(scaled)

    print(f"k={k} → Silhouette Score: {silhouette\_score(scaled, labels):.2f}")

# 4. Apply K-Means (Best k=3)

model = KMeans(n\_clusters=3, random\_state=0)

df['Cluster'] = model.fit\_predict(scaled)

print(df)

# 5. Plot the Clusters

colors = ['red', 'green', 'blue']

for i in range(3):

    plt.scatter(df[df['Cluster']==i]['Income'], df[df['Cluster']==i]['Spending'],

                color=colors[i], label=f'Cluster {i}')

plt.title("K-Means Clusters")

plt.xlabel("Income")

plt.ylabel("Spending")

plt.legend()

plt.grid(True)

plt.show()

# Practical 9

# Principal Component Analysis (PCA)

# • Perform PCA on a dataset to reduce dimensionality.

# • Evaluate the explained variance and select the appropriate number of principal components.

# • Visualize the data in the reduced-dimensional space.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# 1. Sample Data

df = pd.DataFrame({

    'Marks': [45, 55, 65, 75, 85, 95, 40, 60, 70, 90],

    'Attendance': [60, 65, 70, 80, 85, 95, 55, 75, 78, 93],

    'Study\_Hours': [2, 3, 3.5, 4, 5, 6, 1.5, 3.8, 4.5, 5.5]

})

# 2. Standardize

scaled = StandardScaler().fit\_transform(df)

# 3. Apply PCA (2 components)

pca = PCA(n\_components=2)

reduced = pca.fit\_transform(scaled)

# 4. Explained Variance

print("Explained Variance:", pca.explained\_variance\_ratio\_)

# 5. Plot

plt.scatter(reduced[:, 0], reduced[:, 1], color='orchid', s=100)

plt.title("PCA Result (2D View)")

plt.xlabel("PC1")

plt.ylabel("PC2")

plt.grid(True)

plt.show()

# Practical 10

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Sample dataset

data = {

    'Name': ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'],

    'Gender': ['Male', 'Female', 'Female', 'Male', 'Female', 'Male', 'Male', 'Female'],

    'Study\_Hours': [2, 4, 3.5, 5, 6, 1, 3, 4.5],

    'Marks': [50, 75, 70, 85, 90, 40, 60, 80]

}

df = pd.DataFrame(data)

# Step 2: Plot 1 - Study Hours vs Marks

plt.figure(figsize=(6, 4))

sns.scatterplot(data=df, x='Study\_Hours', y='Marks', hue='Gender', style='Gender', s=100)

plt.title("📚 More Study = Better Marks?")

plt.grid(True)

plt.show()

# Step 3: Plot 2 - Gender-wise average marks

plt.figure(figsize=(6, 4))

sns.barplot(data=df, x='Gender', y='Marks', ci=None, palette='pastel')

plt.title(" Who Scores Better on Average?")

plt.grid(True)

plt.show()

# Step 4: Plot 3 - Study Hours Distribution

plt.figure(figsize=(6, 4))

sns.histplot(data=df, x='Study\_Hours', hue='Gender', kde=True, palette='muted')

plt.title(" Study Hour Patterns")

plt.grid(True)

plt.show()