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()
```