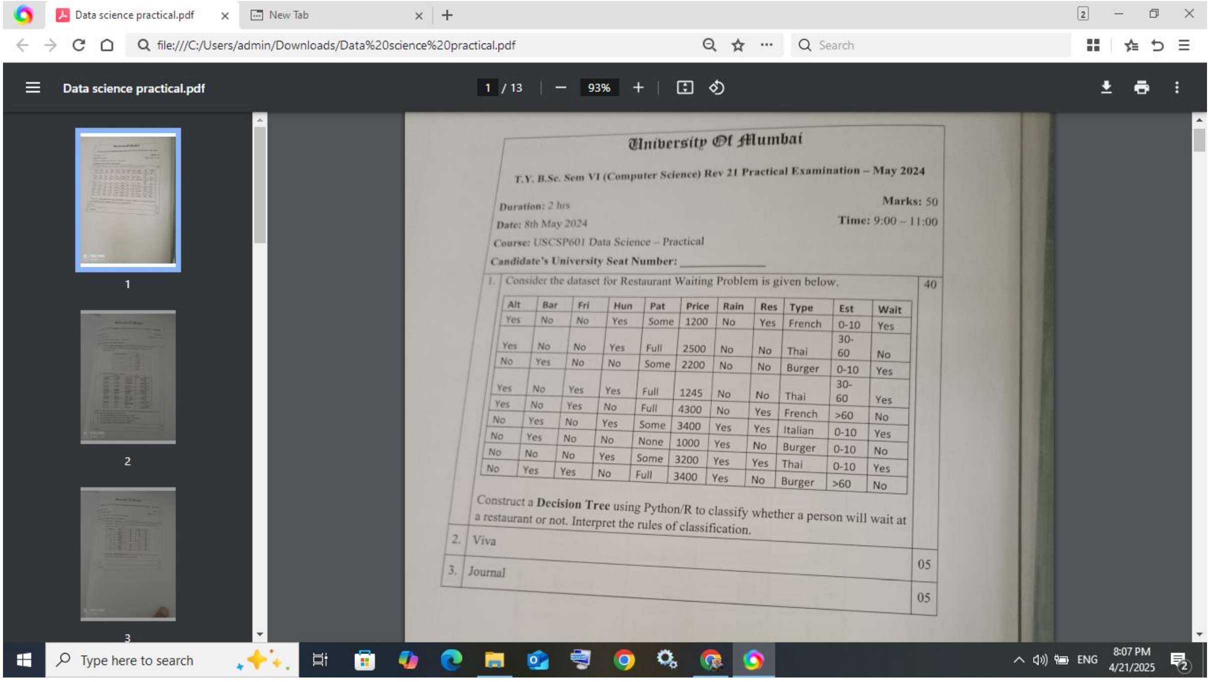


University Of Mumbai

T.Y. B.Sc. Sem VI (Computer Science) Rev 21 Practical Examination – May 2024

Duration: 2 hrs Marks: 50

Date: 8th May 2024 Time: 9:00 – 11:00

Course: USCSP601 Data Science – Practical

Candidate's University Seat Number: ____________

1. Consider the dataset for Restaurant Waiting Problem is given below. 40

| Alt | Bar | Fri | Hun | Pat | Price | Rain | Res | Type | Est | Wait |
|---|---|---|---|---|---|---|---|---|---|---|
| Yes | No | No | Yes | Some | 1200 | No | Yes | French | 0-10 | Yes |
| Yes | No | No | Yes | Full | 2500 | No | No | Thai | 30-60 | No |
| No | Yes | No | No | Some | 2200 | No | No | Burger | 0-10 | Yes |
| Yes | No | Yes | Yes | Full | 1245 | No | No | Thai | 30-60 | Yes |
| Yes | No | Yes | No | Full | 4300 | No | Yes | French | >60 | No |
| No | Yes | No | Yes | Some | 3400 | Yes | Yes | Italian | 0-10 | Yes |
| No | Yes | No | No | None | 1000 | Yes | No | Burger | 0-10 | No |
| No | No | No | Yes | Some | 3200 | Yes | Yes | Thai | 0-10 | Yes |
| No | Yes | Yes | No | Full | 3400 | Yes | No | Burger | >60 | No |

Construct a **Decision Tree** using Python/R to classify whether a person will wait at a restaurant or not. Interpret the rules of classification.

2. Viva 05

3. Journal 05

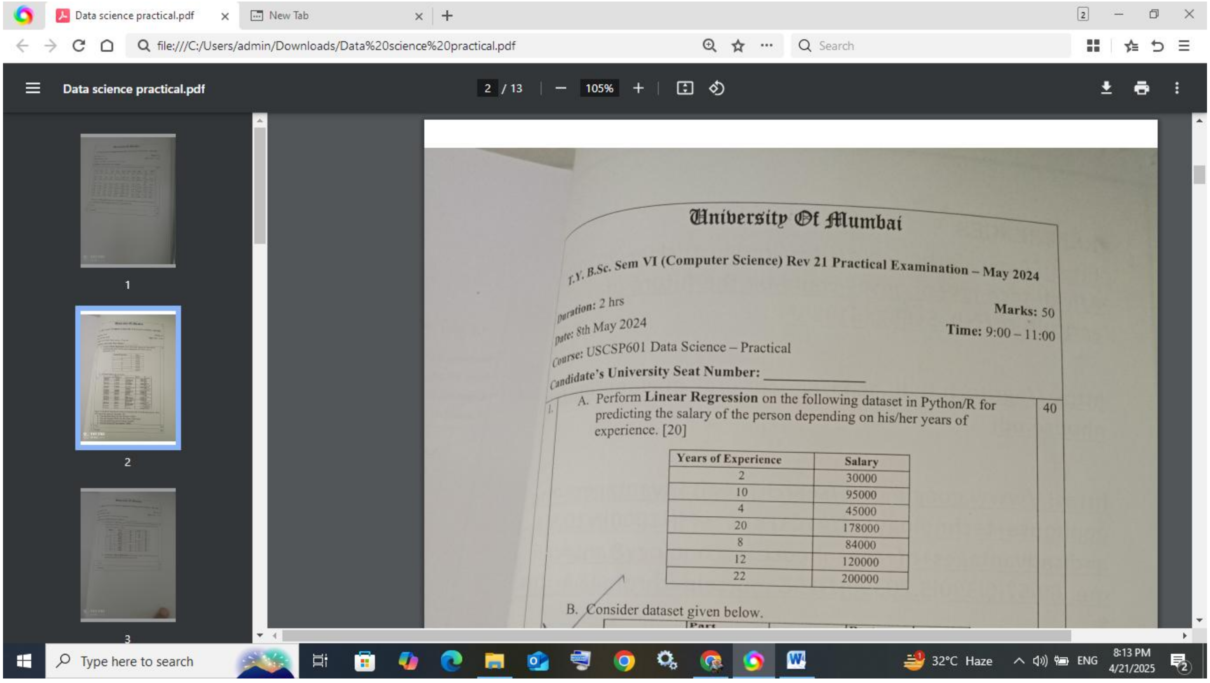
Code:

```
import pandas as pd

from sklearn.tree import DecisionTreeClassifier, export_text

from sklearn.preprocessing import LabelEncoder

# Dataset from the image

data = {

    'Alt': ['Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'Yes'],

    'Bar': ['No', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes'],

    'Fri': ['No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes'],

    'Hun': ['Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes'],

    'Pat': ['Some', 'Full', 'Some', 'Full', 'Full', 'Some', 'None', 'Some', 'Some', 'Full'],

    'Price': [1200, 2500, 2200, 4300, 4300, 3400, 1000, 3400, 3200, 3400],

    'Rain': ['No', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes'],

    'Res': ['Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No'],

    'Type': ['French', 'Thai', 'Burger', 'Thai', 'French', 'Italian', 'Burger', 'Thai', 'Thai', 'Burger'],

    'Est': ['0-10', '30-60', '0-10', '30-60', '>60', '0-10', '0-10', '>60', '0-10', '>60'],
```

```
    'Wait': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No']
}
df = pd.DataFrame(data)
# Encode categorical variables
le = LabelEncoder()
for column in df.columns:
    df[column] = le.fit_transform(df[column])
# Split into features and target
X = df.drop('Wait', axis=1)
y = df['Wait']
# Train Decision Tree
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X, y)
# Display the tree
tree_rules = export_text(clf, feature_names=list(X.columns))
print(tree_rules)
```

**output:-**

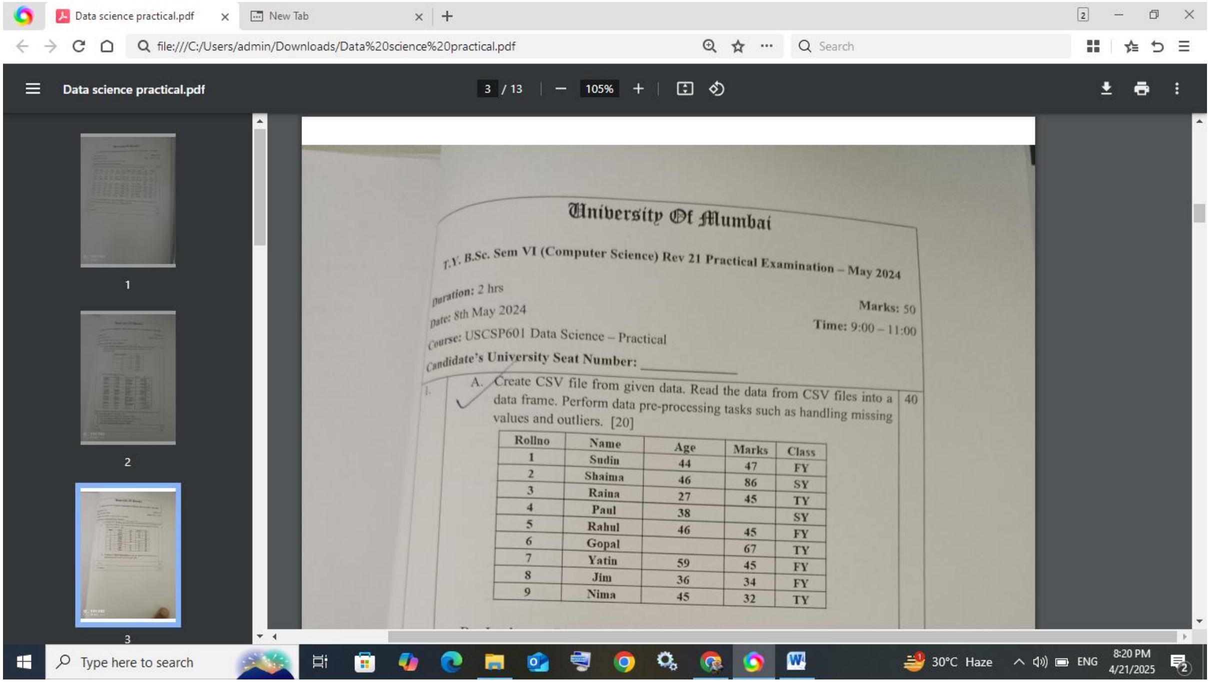
```
|--- Est <= 1.50
|   |--- Pat <= 1.50
|   |   |--- Fri <= 0.50
|   |   |   |--- class: 0        -> Will NOT wait
|   |   |--- Fri >  0.50
|   |   |   |--- class: 1        -> Will wait
|   |--- Pat >  1.50
|   |   |--- class: 1            -> Will wait
|--- Est >  1.50
|   |--- class: 0                -> Will NOT wait
```




Code:-

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression


# Dataset

data = {

    'YearsExperience': [2, 10, 4, 20, 8, 12, 22],

    'Salary': [30000, 95000, 45000, 178000, 84000, 120000, 200000]

}


df = pd.DataFrame(data)


# Features and Target

X = df[['YearsExperience']]

y = df['Salary']
```

```
# Model

model = LinearRegression()

model.fit(X, y)

# Predict salary for 15 years experience (optional)

predicted_salary = model.predict([[15]])

print(f"Predicted salary for 15 years of experience: ₹{predicted_salary[0]:,.2f}")

# Show equation

print("Regression Equation: Salary = {:.2f} * YearsExperience + {:.2f}".format(model.coef_[0],
model.intercept_))

# Plotting

plt.scatter(X, y, color='blue')

plt.plot(X, model.predict(X), color='red')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Salary vs Experience')

plt.grid(True)

plt.show()
```


University Of Mumbai

T.Y. B.Sc. Sem VI (Computer Science) Rev 21 Practical Examination – May 2024

Duration: 2 hrs

Date: 8th May 2024 Marks: 50

Course: USCSP601 Data Science – Practical Time: 9:00 – 11:00

Candidate's University Seat Number: ____________

1. A. Create CSV file from given data. Read the data from CSV files into a data frame. Perform data pre-processing tasks such as handling missing values and outliers. [20] 40

| Rollno | Name | Age | Marks | Class |
|---|---|---|---|---|
| 1 | Sudin | 44 | 47 | FY |
| 2 | Shaima | 46 | 86 | SY |
| 3 | Raina | 27 | 45 | TY |
| 4 | Paul | 38 | | SY |
| 5 | Rahul | 46 | 45 | FY |
| 6 | Gopal | | 67 | TY |
| 7 | Yatin | 59 | 45 | FY |
| 8 | Jim | 36 | 34 | FY |
| 9 | Nima | 45 | 32 | TY |

Code:

```
import pandas as pd

import numpy as np


# Step 1: Create Data and Save to CSV

data = {

   'Rollno': [1, 2, 3, 4, 5, 6, 7, 8, 9],

   'Name': ['Sudin', 'Shaima', 'Raina', 'Paul', 'Rahul', 'Gopal', 'Yatin', 'Jim', 'Nima'],

   'Age': [44, 46, 27, 38, 46, 67, 59, 36, 45],

   'Marks': [47, 86, 45, np.nan, 45, np.nan, 45, 34, 32],

   'Class': ['FY', 'SY', 'TY', 'SY', 'FY', 'TY', 'FY', 'FY', 'TY']

}


df = pd.DataFrame(data)

df.to_csv("students.csv", index=False)  # Save to CSV


# Step 2: Read from CSV
```

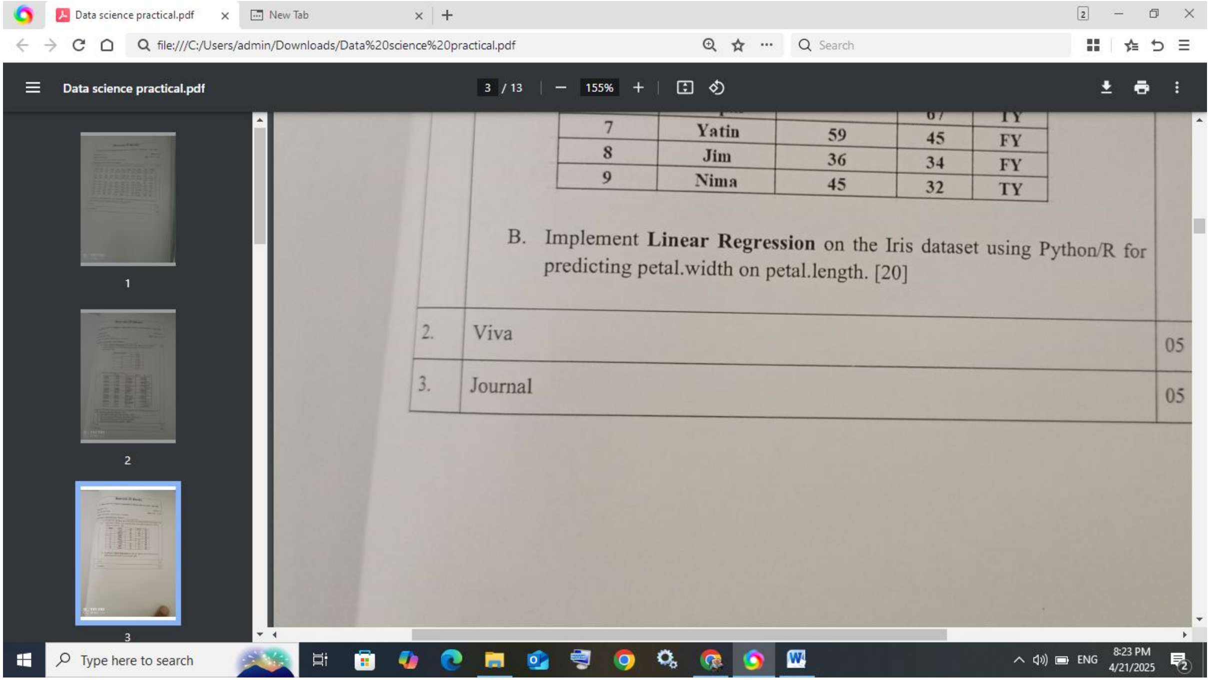
```
df = pd.read_csv("students.csv")

print("Original Data:\n", df)


# Step 3: Handle Missing Values

df['Marks'] = df['Marks'].fillna(df['Marks'].mean())


# Step 4: Detect and Remove Outliers in Age

Q1 = df['Age'].quantile(0.25)

Q3 = df['Age'].quantile(0.75)

IQR = Q3 - Q1

lower = Q1 - 1.5 * IQR

upper = Q3 + 1.5 * IQR


outliers = df[(df['Age'] < lower) | (df['Age'] > upper)]

print("\nOutliers in Age:\n", outliers)


df_cleaned = df[(df['Age'] >= lower) & (df['Age'] <= upper)]

print("\nCleaned DataFrame:\n", df_cleaned)
```

**output:-**

```
  Rollno    Name  Age  Marks Class
0      1   Sudin   44   47.0    FY
1      2  Shaima   46   86.0    SY
2      3   Raina   27   45.0    TY
3      4    Paul   38    NaN    SY
4      5   Rahul   46   45.0    FY
5      6   Gopal   67    NaN    TY
6      7   Yatin   59   45.0    FY
```

**7 8 Jim 36 34.0 FY**

**8 9 Nima 45 32.0 TY**




| | | | | |
|---|---|---|---|---|
| 7 | Yatin | 59 | 45 | FY |
| 8 | Jim | 36 | 34 | FY |
| 9 | Nima | 45 | 32 | TY |

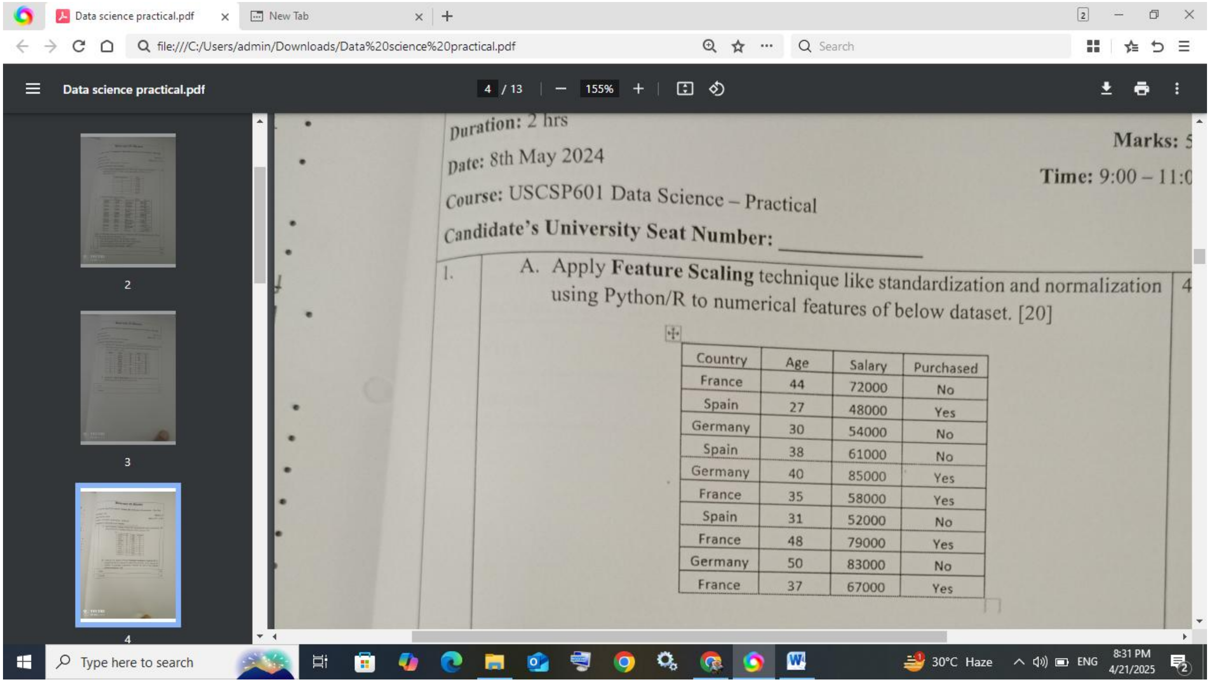
B. Implement **Linear Regression** on the Iris dataset using Python/R for predicting petal.width on petal.length. [20]

| | | |
|---|---|---|
| 2. | Viva | 05 |
| 3. | Journal | 05 |

Code:-

```
from sklearn.linear_model import LinearRegression

from sklearn.datasets import load_iris

import pandas as pd

import matplotlib.pyplot as plt


# Load Iris dataset

iris = load_iris()

iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)


# Extract petal length and width

X = iris_df[['petal length (cm)']]

y = iris_df['petal width (cm)']


# Train model
```

```
model = LinearRegression()
model.fit(X, y)

# Predict
predicted = model.predict(X)

# Output equation
print("Linear Regression Equation:")
print(f"petal.width = {model.coef_[0]:.2f} * petal.length + {model.intercept_:.2f}")

# Plot
plt.scatter(X, y, color='blue')
plt.plot(X, predicted, color='red')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Linear Regression on Iris Dataset')
plt.grid(True)
plt.show()
```



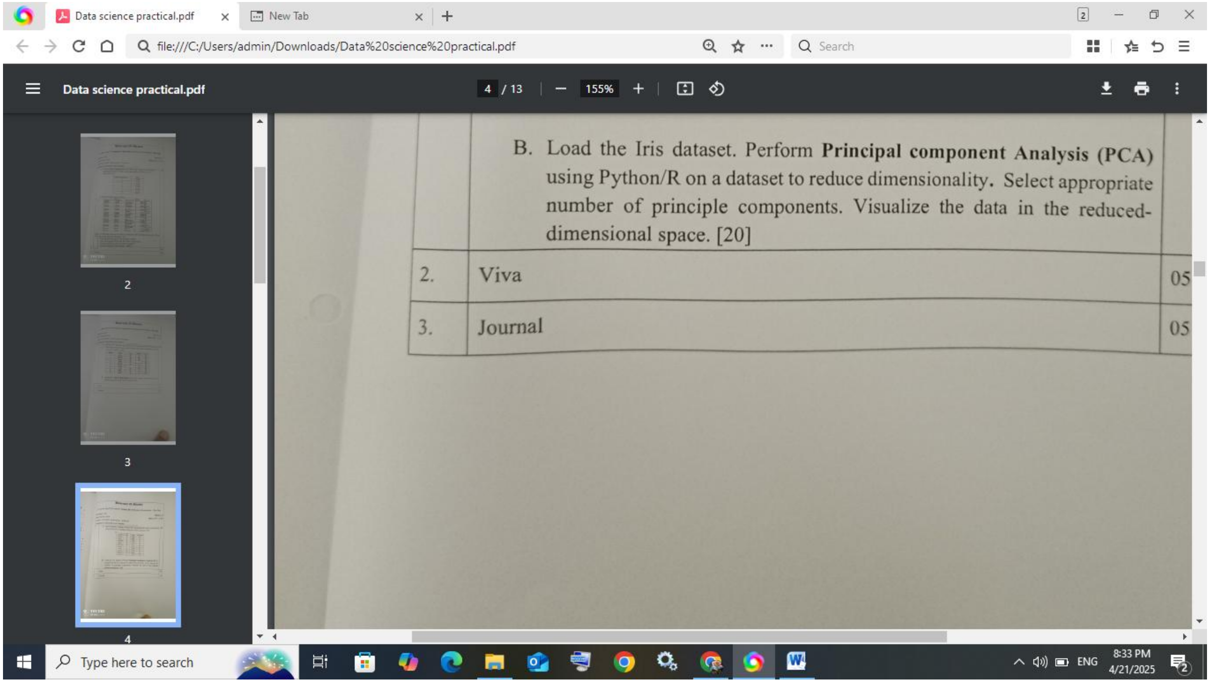

Code:-

```
import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler


# Step 1: Create the dataset

data = {

  'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'Germany', 'France'],

  'Age': [44, 27, 30, 38, 40, 35, 31, 48, 50, 37],

  'Salary': [72000, 48000, 54000, 61000, 85000, 58000, 52000, 79000, 83000, 67000],

  'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']

}


df = pd.DataFrame(data)

print("Original Data:\n", df)


# Step 2: Standardization (Z-score scaling)
```

```
scaler_std = StandardScaler()

df_std = df.copy()

df_std[['Age', 'Salary']] = scaler_std.fit_transform(df_std[['Age', 'Salary']])

print("\nStandardized Data:\n", df_std)


# Step 3: Normalization (Min-Max scaling)

scaler_norm = MinMaxScaler()

df_norm = df.copy()

df_norm[['Age', 'Salary']] = scaler_norm.fit_transform(df_norm[['Age', 'Salary']])

print("\nNormalized Data:\n", df_norm)
```

**output:-**

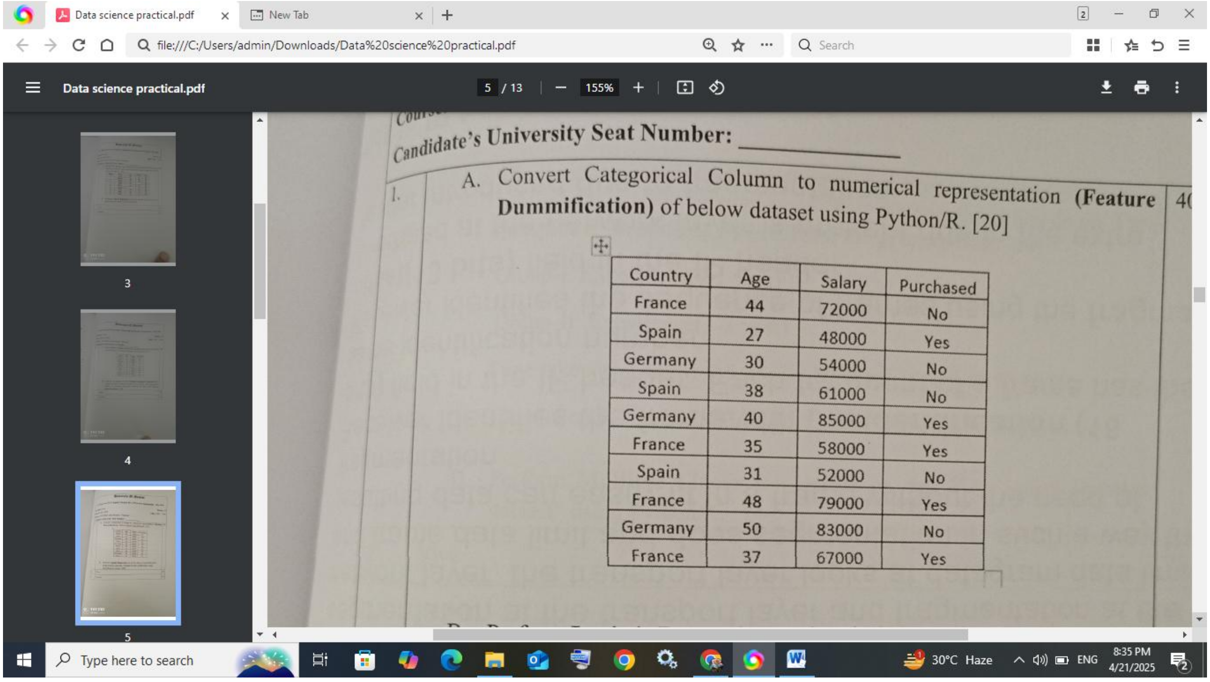
```
   Country  Age  Salary Purchased
0   France   44   72000        No
1    Spain   27   48000       Yes
2  Germany   30   54000        No
3    Spain   38   61000        No
4  Germany   40   85000       Yes
5   France   35   58000       Yes
6    Spain   31   52000        No
7   France   48   79000       Yes
8  Germany   50   83000        No
9   France   37   67000       Yes
```




**Code:-**

```
# Step 1: Import libraries
import pandas as pd
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


# Step 2: Load the Iris dataset
df = sns.load_dataset('iris')


# Step 3: Separate features and target
X = df.drop('species', axis=1)
y = df['species']


# Step 4: Standardize the features
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)


# Step 5: Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)


# Step 6: Explained variance
explained_variance = pca.explained_variance_ratio_
print("Explained Variance Ratio:\n", explained_variance)


# Step 7: Choose number of components (let's use 2 for visualization)
pca_2 = PCA(n_components=2)
X_reduced = pca_2.fit_transform(X_scaled)


# Step 8: Visualize in 2D
pca_df = pd.DataFrame(X_reduced, columns=['PC1', 'PC2'])
pca_df['species'] = y


plt.figure(figsize=(8, 5))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='species', palette='Set1', s=100)
plt.title('PCA of Iris Dataset (2D Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.legend()
plt.show()
```


Candidate's University Seat Number: ___________

1. A. Convert Categorical Column to numerical representation **(Feature Dummification)** of below dataset using Python/R. [20]

| Country | Age | Salary | Purchased |
|---|---|---|---|
| France | 44 | 72000 | No |
| Spain | 27 | 48000 | Yes |
| Germany | 30 | 54000 | No |
| Spain | 38 | 61000 | No |
| Germany | 40 | 85000 | Yes |
| France | 35 | 58000 | Yes |
| Spain | 31 | 52000 | No |
| France | 48 | 79000 | Yes |
| Germany | 50 | 83000 | No |
| France | 37 | 67000 | Yes |

**Code:-**

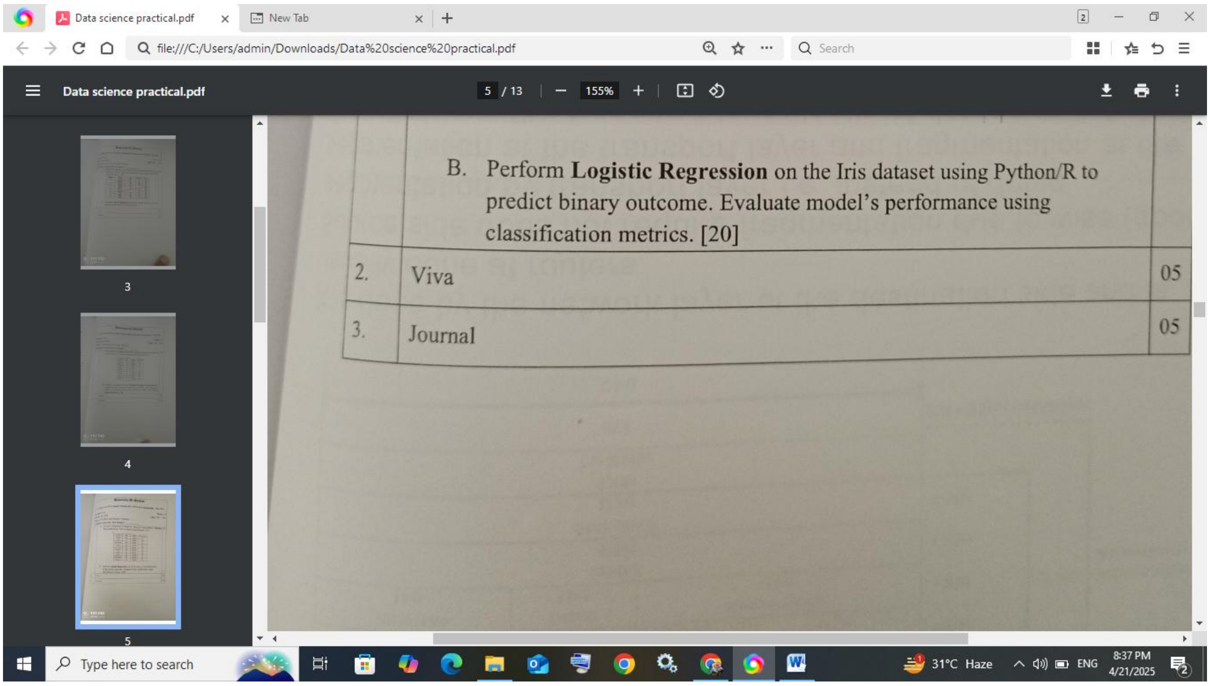
```
import pandas as pd


# Step 1: Create the dataset
data = {
    'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'Germany', 'France'],
    'Age': [44, 27, 30, 38, 40, 35, 31, 48, 50, 37],
    'Salary': [72000, 48000, 54000, 61000, 85000, 58000, 52000, 79000, 83000, 67000],
    'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']
}


df = pd.DataFrame(data)


# Step 2: Convert categorical columns using get_dummies
df_encoded = pd.get_dummies(df, columns=['Country', 'Purchased'], drop_first=True)
```

# Step 3: Display the result

print("Encoded DataFrame:\n", df_encoded)

**output:-**

```
   Age  Salary  Country_Germany  Country_Spain  Purchased_Yes
0   44   72000                0              0              0
1   27   48000                0              1              1
2   30   54000                1              0              0
3   38   61000                0              1              0
4   40   85000                1              0              1
5   35   58000                0              0              1
6   31   52000                0              1              0
7   48   79000                0              0              1
8   50   83000                1              0              0
9   37   67000                0              0              1
```




**Code:-**

# Step 1: Import libraries

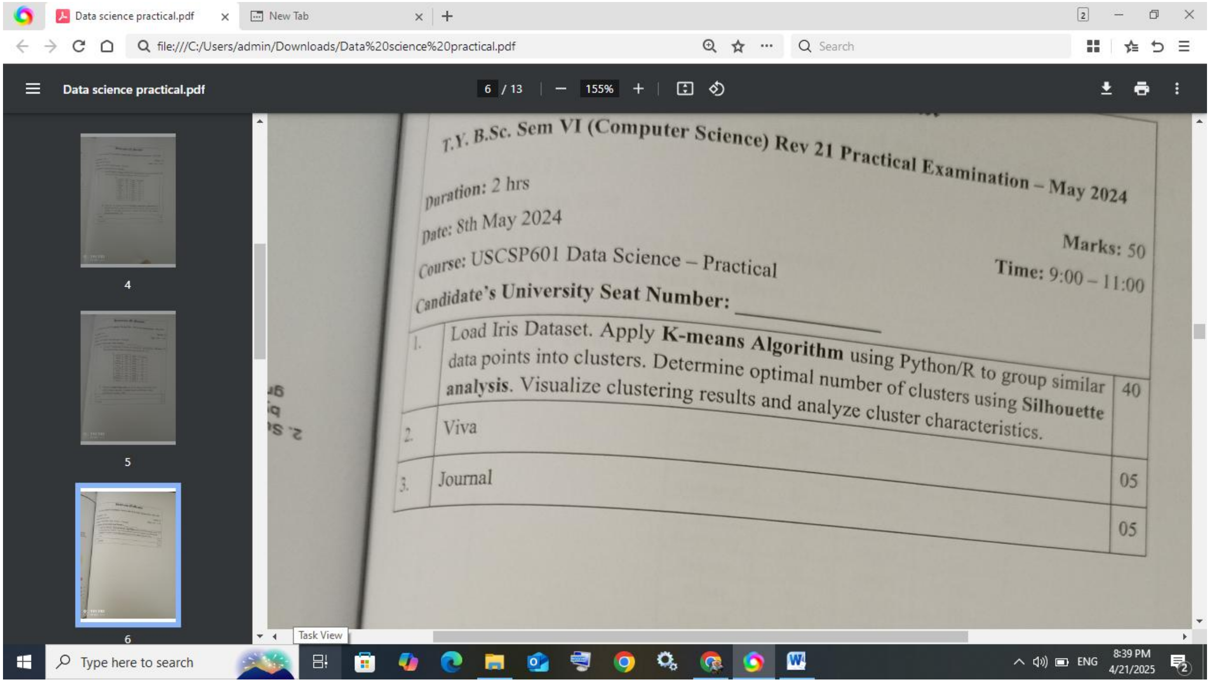
import pandas as pd

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from sklearn.preprocessing import StandardScaler

import seaborn as sns


# Step 2: Load dataset

iris = sns.load_dataset('iris')


# Step 3: Create a binary classification target (1 if setosa, else 0)

iris['target'] = (iris['species'] == 'setosa').astype(int)


# Step 4: Select features and target

X = iris.drop(['species', 'target'], axis=1)

y = iris['target']


# Step 5: Feature scaling (optional but improves performance)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 6: Split into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)


# Step 7: Train logistic regression model

model = LogisticRegression()

model.fit(X_train, y_train)
```

```
# Step 8: Predict on test data
y_pred = model.predict(X_test)


# Step 9: Evaluate performance
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
```

**output:-**

```
Confusion Matrix:
[[29  0]
 [ 0 16]]


Classification Report:
              precision    recall  f1-score   support


           0       1.00      1.00      1.00        29
           1       1.00      1.00      1.00        16


    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Accuracy Score: 1.0
```




**Code:-**

```
# Suppress warnings
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load Iris dataset
```

```
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)

# Determine the optimal number of clusters using silhouette score
silhouette_scores = []
K_range = range(2, 10)
for k in K_range:
   kmeans = KMeans(n_clusters=k, random_state=42)
   kmeans.fit(X)
   score = silhouette_score(X, kmeans.labels_)
   silhouette_scores.append(score)

# Plot silhouette scores
plt.figure(figsize=(8, 4))
plt.plot(K_range, silhouette_scores, marker='o')
plt.title("Silhouette Score vs Number of Clusters")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")
plt.grid(True)
plt.show()

# Apply KMeans with optimal k (e.g., 3)
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)

# Add labels to DataFrame
X['Cluster'] = labels
```

```
# Visualize Clustering
sns.pairplot(X, hue='Cluster', palette='Set1', corner=True)
plt.suptitle("K-Means Clustering of Iris Data", y=1.02)
plt.show()
```