AD23632 - Framework for Data and Visual Analytics

EXP.NO:1	Python in Jupyter Notebook

AIM:

Setting up the Python environment and libraries-Juypter Notebook to:

i)Create a new notebook for Python

ii)Write and execute Python code

iii)Create new cells for code and Markdown

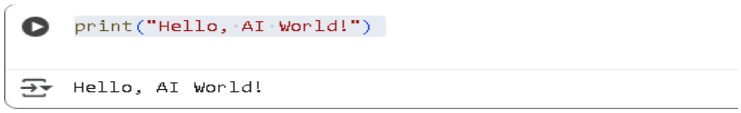
iv)Demonstrate the application of Jupyter Widgets, Jupyter AI

SOURCE CODE:

i)

```
print("Hello, AI World!")
```

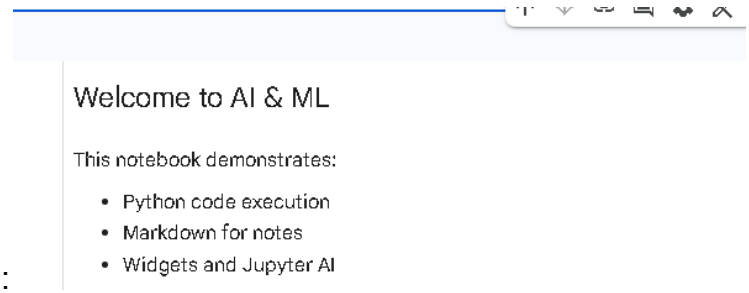
OUTPUT:



ii)

```
## Welcome to AI & ML
This notebook demonstrates:
- Python code execution
- Markdown for notes
- Widgets and Jupyter AI
```

OUTPUT:



:

iii)

```
import ipywidgets as widgets
from IPython.display import display

slider = widgets.IntSlider(value=5, min=0, max=10, step=1, description='Slider:')
display(slider)

def on_change(change):
    print(f'Slider changed to {change["new"]}')

slider.observe(on_change, names='value')
```

OUTPUT:

RESULT:

Thus the program has been successfully executed in python

EXP.NO:2	EDA-DATA IMPORT AND EXPORT

AIM:

To:

i)Import data from CSV, Excel, SQL databases, and web scraping

ii)Handling different data formats

iii) Export a DataFrame to an Excel file.

SOURCE CODE:

```
import pandas as pd
import requests
from google.colab import files
uploaded = files.upload()
df_csv = pd.read_csv(next(iter(uploaded)))
print("CSV loaded successfully")

print(df_csv.head())
print(df_csv.shape)
print(df_csv.columns)
print(df_csv.describe())
print(df_csv.isnull().sum())
df_excel = df_csv.copy()
print("New DataFrame created from CSV")
print(df_excel.head())
df_csv.to_excel("college_placement_export.xlsx", index=False)
print("Excel file saved successfully")
url = "https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)"
response = requests.get(url)
if response.status_code == 200:
    print("Successfully retrieved the webpage content.")
    # print(response.text) # Uncomment this line to see the HTML content
    try:
        tables = pd.read_html(response.text)
        # Assuming the table is still the 3rd one (index 2) as in the original code
        df_web = tables[2]

        # Clean column names safely
        df_web.columns = [str(col).strip() for col in df_web.columns]

        print("Scraped table:")
        display(df_web.head())
```

```
        # 5. Export scraped table to Excel (runtime only)
        df_web.to_excel("scraped_gdp_table.xlsx", index=False)
        print("Scraped table saved to Excel")

    except ValueError as e:
        print(f"Error reading HTML tables: {e}")
        print("It seems the structure of the Wikipedia page might have changed, or the table is not in the expected format.")
else:
    print(f"Failed to retrieve the webpage. Status code: {response.status_code}")
```

OUTPUT:

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving college_student_placement_dataset.csv to college_student_placement_dataset (3).csv
CSV loaded successfully
  College_ID   IQ  Prev_Sem_Result  CGPA  Academic_Performance  \
0    CLG0030  107             6.61  6.28                    8
1    CLG0061   97             5.52  5.37                    8
2    CLG0036  109             5.36  5.83                    9
3    CLG0055  122             5.47  5.75                    6
4    CLG0004   96             7.91  7.69                    7

  Internship_Experience  Extra_Curricular_Score  Communication_Skills  \
0                    No                       8                     8
1                    No                       7                     8
2                    No                       3                     1
3                   Yes                       1                     6
4                    No                       8                    10

   Projects_Completed Placement
0                   4        No
1                   0        No
2                   1        No
3                   1        No
4                   2        No
(10000, 10)
Index(['College_ID', 'IQ', 'Prev_Sem_Result', 'CGPA', 'Academic_Performance',
       'Internship_Experience', 'Extra_Curricular_Score',
       'Communication_Skills', 'Projects_Completed', 'Placement'],
      dtype='object')
                IQ  Prev_Sem_Result          CGPA  Academic_Performance  \
count  10000.000000     10000.000000  10000.000000          10000.000000
mean      99.471800         7.535673      7.532379              5.546400
std       15.053101         1.447519      1.470141              2.873477
min       41.000000         5.000000      4.540000              1.000000
25%       89.000000         6.290000      6.290000              3.000000
50%       99.000000         7.560000      7.550000              6.000000
75%      110.000000         8.790000      8.770000              8.000000
```

```
max      158.000000        10.000000     10.460000             10.000000

       Extra_Curricular_Score  Communication_Skills  Projects_Completed
count            10000.000000          10000.000000        10000.000000
mean                 4.970900              5.561800            2.513400
std                  3.160103              2.900866            1.715959
min                  0.000000              1.000000            0.000000
25%                  2.000000              3.000000            1.000000
50%                  5.000000              6.000000            3.000000
75%                  8.000000              8.000000            4.000000
max                 10.000000             10.000000            5.000000
College_ID                0
IQ                        0
Prev_Sem_Result           0
CGPA                      0
Academic_Performance      0
Internship_Experience     0
Extra_Curricular_Score    0
Communication_Skills      0
Projects_Completed        0
Placement                 0
dtype: int64
New DataFrame created from CSV
  College_ID   IQ  Prev_Sem_Result  CGPA  Academic_Performance  \
0    CLG0030  107             6.61  6.28                     8
1    CLG0061   97             5.52  5.37                     8
2    CLG0036  109             5.36  5.83                     9
3    CLG0055  122             5.47  5.75                     6
4    CLG0004   96             7.91  7.69                     7

  Internship_Experience  Extra_Curricular_Score  Communication_Skills  \
0                    No                       8                     8
1                    No                       7                     8
2                    No                       3                     1
3                   Yes                       1                     6
4                    No                       8                    10

   Projects_Completed Placement
0                   4        No
1                   0        No
2                   1        No
3                   1        No
4                   2        No
Excel file saved successfully
```

RESULT:

The data has been imported and exported successfully.

EXP.NO:3	EDA-Data Cleaning

AIM:

To implement Handling missing values: detection, filling, and dropping,Removing duplicates and unnecessary data.Data type conversion and ensuring consistency Normalize data (e.g., standardization, min-max scaling).

SOURCE CODE:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from google.colab import files
uploaded = files.upload()
df = pd.read_csv(next(iter(uploaded)))
print("CSV loaded successfully")
print(" Dataset Preview:")
print(df.iloc[:, :4].head())
print("\n Missing Values:")
print(df.isnull().sum())

if 'Age' in df.columns:
    df['Age'].fillna(df['Age'].median(), inplace=True)

# Fill Embarked with mode
if 'Embarked' in df.columns:
    df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Fill Cabin with 'Unknown' or drop if sparse
if 'Cabin' in df.columns:
    missing_ratio = df['Cabin'].isnull().mean()
    if missing_ratio > 0.8:
        df.drop('Cabin', axis=1, inplace=True)
    else:
        df['Cabin'].fillna('Unknown', inplace=True)

# Drop rows with any remaining missing values (if needed)
df.dropna(inplace=True)

# Step 6: Remove duplicates
duplicates = df.duplicated().sum()
print(f"\n Duplicates Found: {duplicates}")
```

```
df.drop_duplicates(inplace=True)

# Step 7: Drop unnecessary columns (optional)
drop_cols = ['Name', 'Ticket']  # Add more if needed
df.drop([col for col in drop_cols if col in df.columns], axis=1, inplace=True)

# Step 8: Convert data types
if 'Survived' in df.columns:
    df['Survived'] = df['Survived'].astype('category')
if 'Pclass' in df.columns:
    df['Pclass'] = df['Pclass'].astype('category')

# Step 9: Ensure consistency in categorical values
if 'Sex' in df.columns:
    df['Sex'] = df['Sex'].str.lower().str.strip()
if 'Embarked' in df.columns:
    df['Embarked'] = df['Embarked'].str.upper().str.strip()

# Step 10: Normalize numeric columns
# Choose either Min-Max or Standardization
numeric_cols = ['Age', 'Fare']
available_numeric_cols = [col for col in numeric_cols if col in df.columns]

# Min-Max Scaling
scaler = MinMaxScaler()
df[available_numeric_cols] = scaler.fit_transform(df[available_numeric_cols])

# Final info
print("\n Cleaned Data Info:")
print(df.info())

print("\n Summary Statistics:")
print(df.describe(include='all'))
```

OUTPUT:

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Titanic-Dataset.csv to Titanic-Dataset.csv
CSV loaded successfully
 Dataset Preview:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name
0                            Braund, Mr. Owen Harris
1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2                             Heikkinen, Miss. Laina
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                           Allen, Mr. William Henry

 Missing Values:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

 Duplicates Found: 0

 Cleaned Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    category
 2   Pclass       891 non-null    category
 3   Sex          891 non-null    object
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Cabin        891 non-null    object
 9   Embarked     891 non-null    object
dtypes: category(2), float64(2), int64(3), object(3)
```

```
memory usage: 57.8+ KB
None

 Summary Statistics:
        PassengerId  Survived  Pclass   Sex         Age       SibSp  \
count    891.000000     891.0   891.0   891  891.000000  891.000000
unique          NaN       2.0     3.0     2         NaN         NaN
top             NaN       0.0     3.0  male         NaN         NaN
freq            NaN     549.0   491.0   577         NaN         NaN
mean     446.000000       NaN     NaN   NaN    0.363679    0.523008
std      257.353842       NaN     NaN   NaN    0.163605    1.102743
min        1.000000       NaN     NaN   NaN    0.000000    0.000000
25%      223.500000       NaN     NaN   NaN    0.271174    0.000000
50%      446.000000       NaN     NaN   NaN    0.346569    0.000000
75%      668.500000       NaN     NaN   NaN    0.434531    1.000000
max      891.000000       NaN     NaN   NaN    1.000000    8.000000

             Parch        Fare    Cabin Embarked
count   891.000000  891.000000      891      891
unique         NaN         NaN      148        3
top            NaN         NaN  Unknown        S
freq           NaN         NaN      687      646
mean      0.381594    0.062858      NaN      NaN
std       0.806057    0.096995      NaN      NaN
min       0.000000    0.000000      NaN      NaN
25%       0.000000    0.015440      NaN      NaN
50%       0.000000    0.028213      NaN      NaN
75%       0.000000    0.060508      NaN      NaN
max       6.000000    1.000000      NaN      NaN
```

RESULT:

The data has been cleaned successfully.

EXP.NO:4	EDA-Data Inspection and Analysis

AIM:

To implement Viewing and inspecting DataFrames Filtering and subsetting data using conditions Descriptive statistics: measures of central tendency (mean, median, mode) and measures of dispersion (range, variance, standard deviation)

SOURCE CODE:

```
import pandas as pd
import numpy as np

from google.colab import files

# 1. Upload CSV File
uploaded = files.upload()
df = pd.read_csv(next(iter(uploaded)))
print("CSV loaded successfully")

# 1. Viewing and Inspecting DataFrames
print(" Shape of dataset:", df.shape)

print("\n Data Types and Null Values:")
print(df.info())

print("\n First 5 Rows:")
print(df.head())

print("\n Missing values in each column:")
print(df.isnull().sum())

# 2. Filtering and Subsetting Data

# Applicants with income > 5000
high_income = df[df['ApplicantIncome'] > 5000]
print(f"\n Number of high income applicants (>5000): {high_income.shape[0]}")

# Approved loans for self-employed applicants
approved_self_employed = df[(df['Self_Employed'] == 'Yes') & (df['Loan_Status'] == 'Y')]
print(f" Approved self-employed loans: {approved_self_employed.shape[0]}")
```

```
# Urban applicants with coapplicants
urban_with_coapp = df[(df['Property_Area'] == 'Urban') & (df['CoapplicantIncome'] > 0)]
print(f" Urban applicants with coapplicants: {urban_with_coapp.shape[0]}")

# 3. Descriptive Statistics

# LoanAmount column (drop NaNs)
loan_amt = df['LoanAmount'].dropna()

mean_loan = loan_amt.mean()
median_loan = loan_amt.median()
mode_loan = loan_amt.mode()[0]
range_loan = loan_amt.max() - loan_amt.min()
variance_loan = loan_amt.var()
std_loan = loan_amt.std()

print("\n LoanAmount Statistics:")
print(f"Mean: {mean_loan:.2f}")
print(f"Median: {median_loan}")
print(f"Mode: {mode_loan}")
print(f"Range: {range_loan}")
print(f"Variance: {variance_loan:.2f}")
print(f"Standard Deviation: {std_loan:.2f}")

# ApplicantIncome column
income = df['ApplicantIncome']

mean_income = income.mean()
median_income = income.median()
mode_income = income.mode()[0]
range_income = income.max() - income.min()
variance_income = income.var()
std_income = income.std()

print("\n ApplicantIncome Statistics:")
print(f"Mean: {mean_income:.2f}")
print(f"Median: {median_income}")
print(f"Mode: {mode_income}")
print(f"Range: {range_income}")
print(f"Variance: {variance_income:.2f}")
print(f"Standard Deviation: {std_income:.2f}")

# 4. Summary Table
print("\n Summary Statistics for All Numeric Columns:")
```

```
print(df.describe())


# 5. Group Analysis: Mean LoanAmount by Education
loan_by_education = df.groupby('Education')['LoanAmount'].mean()
print("\n Mean LoanAmount by Education:")
print(loan_by_education)
```

OUTPUT:

```
Saving Loan_data.csv to Loan_data.csv
CSV loaded successfully
 Shape of dataset: (614, 13)

 Data Types and Null Values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
None

 First 5 Rows:
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
```

```
0           1.0       Urban          Y
1           1.0       Rural          N
2           1.0       Urban          Y
3           1.0       Urban          Y
4           1.0       Urban          Y

 Missing values in each column:
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64

 Number of high income applicants (>5000): 191
 Approved self-employed loans: 56
 Urban applicants with coapplicants: 107

 LoanAmount Statistics:
Mean: 146.41
Median: 128.0
Mode: 120.0
Range: 691.0
Variance: 7325.19
Standard Deviation: 85.59

 ApplicantIncome Statistics:
Mean: 5403.46
Median: 3812.5
Mode: 2500
Range: 80850
Variance: 37320390.17
Standard Deviation: 6109.04

 Summary Statistics for All Numeric Columns:
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       614.000000         614.000000  592.000000         600.00000
mean       5403.459283        1621.245798  146.412162         342.00000
std        6109.041673        2926.248369   85.587325          65.12041
min         150.000000           0.000000    9.000000          12.00000
25%        2877.500000           0.000000  100.000000         360.00000
50%        3812.500000        1188.500000  128.000000         360.00000
75%        5795.000000        2297.250000  168.000000         360.00000
max       81000.000000       41667.000000  700.000000         480.00000

       Credit_History
```

```
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000

 Mean LoanAmount by Education:
Education
Graduate        154.060215
Not Graduate    118.409449
Name: LoanAmount, dtype: float64
```

RESULT:

The python code for data inspection and data analysis has been executed successfully.

EXP.NO:5	EDA-Data Visualization with Matplotlib

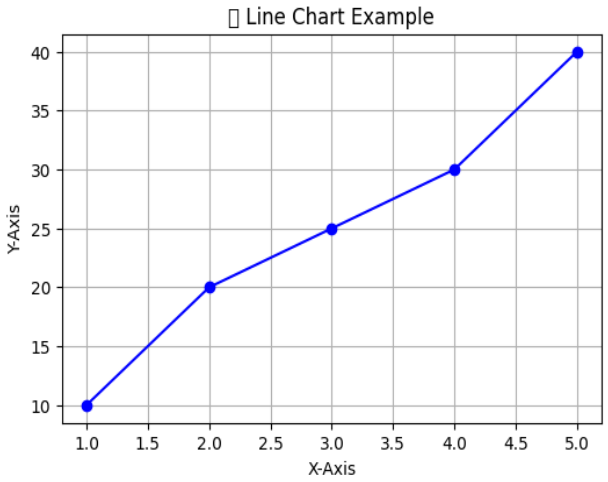
AIM:

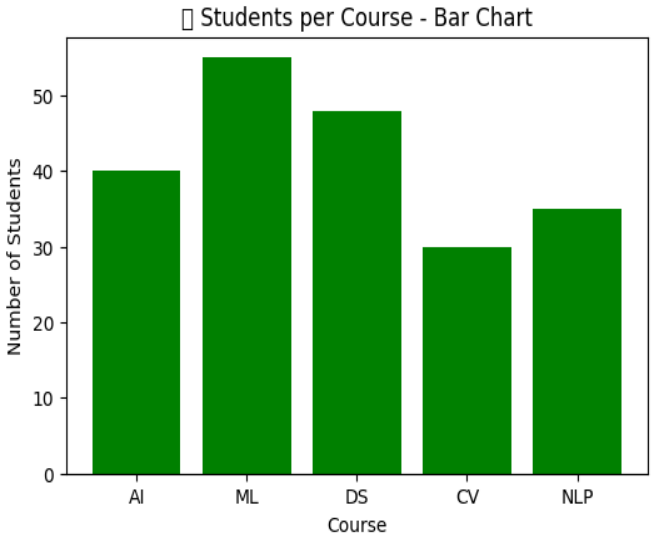
To plot the Basic plotting: line charts, bar charts, histograms

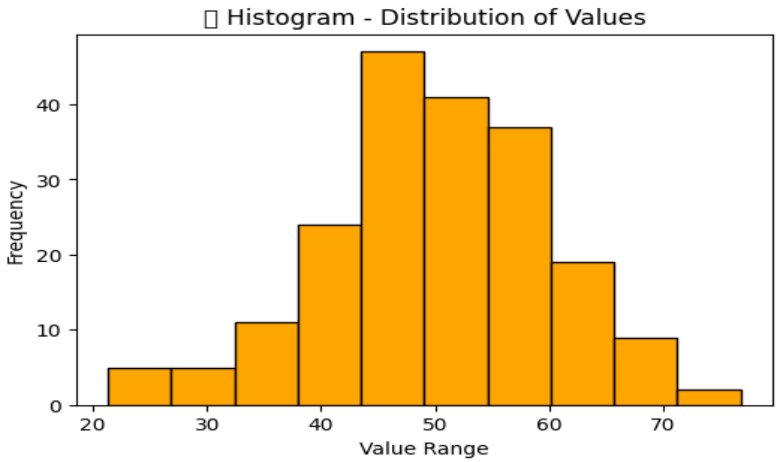
SOURCE CODE:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.figure(figsize=(6, 4))
plt.plot(x, y, color='blue', marker='o', linestyle='-')
plt.title("📈 Line Chart Example")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.grid(True)
plt.show()
courses = ['AI', 'ML', 'DS', 'CV', 'NLP']
students = [40, 55, 48, 30, 35]
plt.figure(figsize=(6, 4))
plt.bar(courses, students, color='green')
plt.title("📊 Students per Course - Bar Chart")
plt.xlabel("Course")
plt.ylabel("Number of Students")
plt.show()
data = np.random.normal(loc=50, scale=10, size=200)  # Mean=50, Std=10

plt.figure(figsize=(6, 4))
plt.hist(data, bins=10, color='orange', edgecolor='black')
plt.title("📉 Histogram - Distribution of Values")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```

OUTPUT:







RESULT:

Thus the visualization has been executed successfully.