DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE
(AUTONOMOUS)
(Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai)
Re-Accredited with 'A' Grade By NAAC, Accredited by TCS.
Accredited by NBA (AERO, CSE, IT & MECH)
Re-Accredited by NBA (BME, ECE, EEE)
PERAMBALUR - 621212.

U23AIP42/ DATA SCIENCE LABORATORY

B.E/B.Tech., Degree Practical Examination

LAB RECORD MANUAL

TABLEOF CONTENTS

Experiment 1: Study of Basic Functions in Excel

AIM

To understand and apply the basic built-in functions in Microsoft Excel such as SUM, AVERAGE, IF, COUNT, and others for performing basic data analysis tasks.

ALGORITHM:

1. **Open Microsoft Excel** and create a new worksheet.
2. **Enter sample data** into cells (e.g., names and numerical values like marks or sales).
3. **Use the** SUM() **function** to calculate the total of numerical data in a range.
4. **Use the** AVERAGE() **function** to find the mean of the numbers in the data range.
5. **Apply the** IF() **function** to implement conditional logic (e.g., pass/fail based on marks).
6. **Use the** COUNT() **function** to count the number of numeric entries in a range.
7. **Use the** COUNTA() **function** to count the number of non-empty cells.
8. **Use the** MAX() **and** MIN() **functions** to find the highest and lowest values in the dataset.
9. **Review and interpret the results** obtained from the applied functions.

Functions to Study:

Function	Description
SUM()	Adds all the numbers in a range of cells.
AVERAGE()	Calculates the average (arithmetic mean) of the selected range.
IF()	Performs a logical test and returns one value for TRUE and another for FALSE.
COUNT()	Counts the number of numeric values in a range.
COUNTA()	Counts the number of non-empty cells.
MAX()	Returns the maximum value in a range.
MIN()	Returns the minimum value in a range.

Procedure:

1. Open Microsoft Excel and create a new spreadsheet.
2. Enter sample data in rows and columns. For example

```
A       B
-----------------
Name    Marks
John    78
Alice   82
Bob     90
Emma    66
```

3. Apply the following functions:
    - =SUM(B2:B5) to calculate total marks.
    - =AVERAGE(B2:B5) to calculate average marks.
    - =MAX(B2:B5) and =MIN(B2:B5) to find highest and lowest marks.
    - =IF(B2>75, "Pass", "Fail") to determine pass/fail status.
    - =COUNT(B2:B5) and =COUNTA(A2:A5) to count numeric and non-empty cells.

Observation Table:

Name	Marks	Result (IF Function)
John	78	Pass
Alice	82	Pass
Bob	90	Pass
Emma	66	Fail

RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 2: STUDY OF BASIC DATA SCIENCE LIBRARIES IN PYTHON

AIM:

To explore and understand the fundamental Python libraries used in data science — NumPy**,** Pandas**,** Matplotlib**, and** Seaborn **—** and demonstrate basic operations such as array manipulation, data handling, and data visualization.

ALGORITHM:

1. Initialize the Python environment by importing required libraries: `numpy`, `pandas`, `matplotlib.pyplot`, and `seaborn`.
2. Work with NumPy:
    - Create a NumPy array.
    - Perform basic mathematical operations such as mean, standard deviation, and sum.
3. Work with Pandas:
    - Create a DataFrame using a dictionary or external data source.
    - Display the DataFrame.
    - Compute descriptive statistics such as mean and count.
4. Work with Matplotlib:
    - Generate a simple line plot or bar chart using sample data.
    - Label the axes and add a title to the plot.
    - Display the plot.
5. Work with Seaborn:
    - Use Seaborn to create a more visually appealing statistical plot (e.g., barplot, scatterplot).
    - Customize the plot as needed (titles, axis labels).
    - Display the plot

Introduction to Libraries:

Library	Description
NumPy	Provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions.
Pandas	Used for data manipulation and analysis; provides DataFrame and Series structures.
Matplotlib	A plotting library for creating static, interactive, and animated visualizations.
Seaborn	Built on Matplotlib; provides a high-level interface for attractive and informative statistical graphics.

Procedure:

1. **Import Required Libraries:**

   ```
   import numpy as np
   import pandas as pd
   import matplotlib.pyplot as plt
   import seaborn as sns
   ```

2. **Using NumPy:**

   ```
   arr = np.array([10, 20, 30, 40])
   print("Array:", arr)
   print("Mean:", np.mean(arr))
   print("Standard Deviation:", np.std(arr))
   ```

3. **Using Pandas:**

   ```
   data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Marks': [85, 90, 78]}
   df = pd.DataFrame(data)
   print(df)
   print("Average Marks:", df['Marks'].mean())
   ```

4. **Using Matplotlib:**

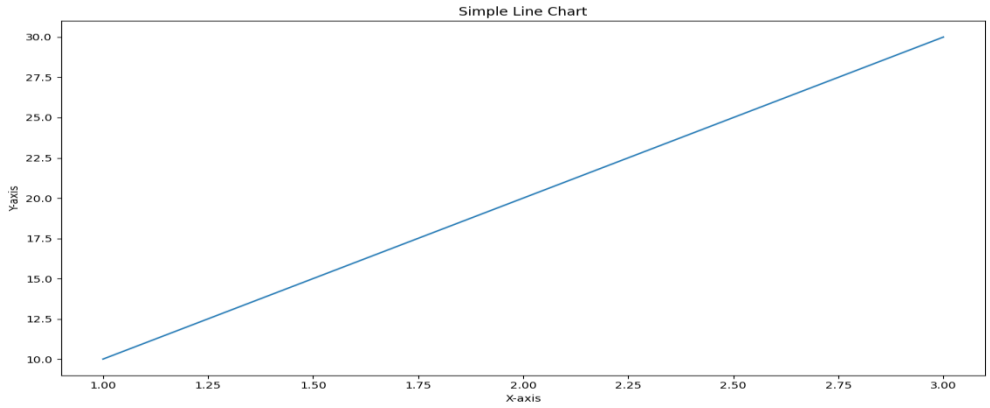
   ```
   plt.plot([1, 2, 3], [10, 20, 30])
   plt.title("Simple Line Chart")
   plt.xlabel("X-axis")
   plt.ylabel("Y-axis")
   plt.show()
   ```

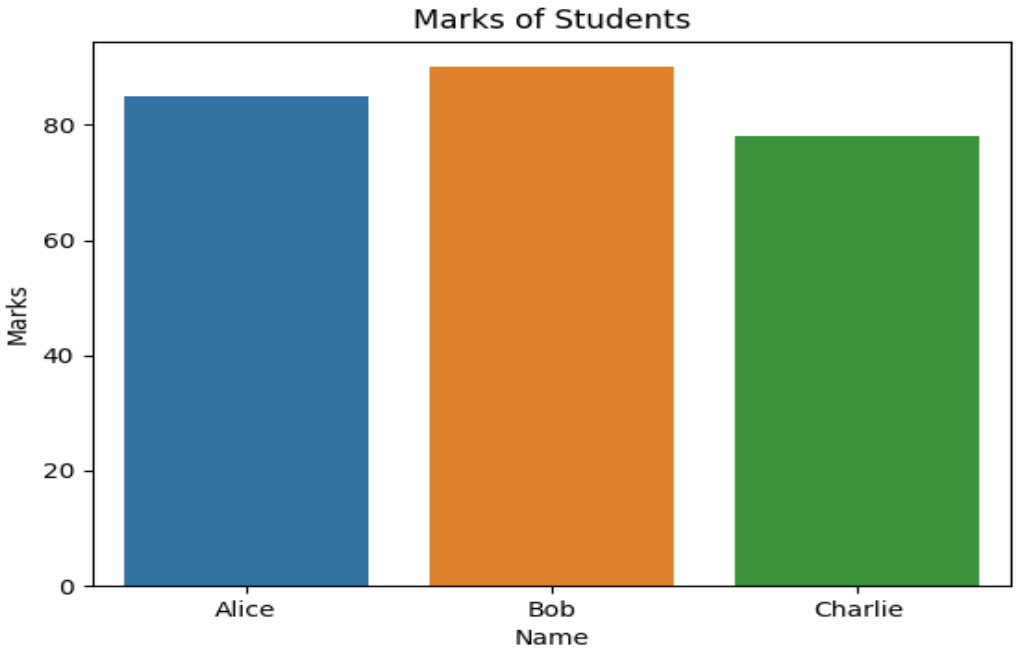
5. **Using Seaborn:**

   ```
   sns.barplot(x='Name', y='Marks', data=df)
   plt.title("Marks of Students")
   plt.show()
   ```

OUTPUT:

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/HI/AppData/Local/Programs/Python/Python37/EX.NP.PY =====
Array: [10 20 30 40]
Mean: 25.0
Standard Deviation: 11.180339887498949
      Name  Marks
0    Alice     85
1      Bob     90
2  Charlie     78
Average Marks: 84.33333333333333
>>>
```





RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 3: WORKING WITH RANGE NAMES AND TABLES IN PYTHON

AIM:

To understand how to work with named data ranges and tables in Python using the Pandas library for effective data management and analysis.

ALGORITHM:

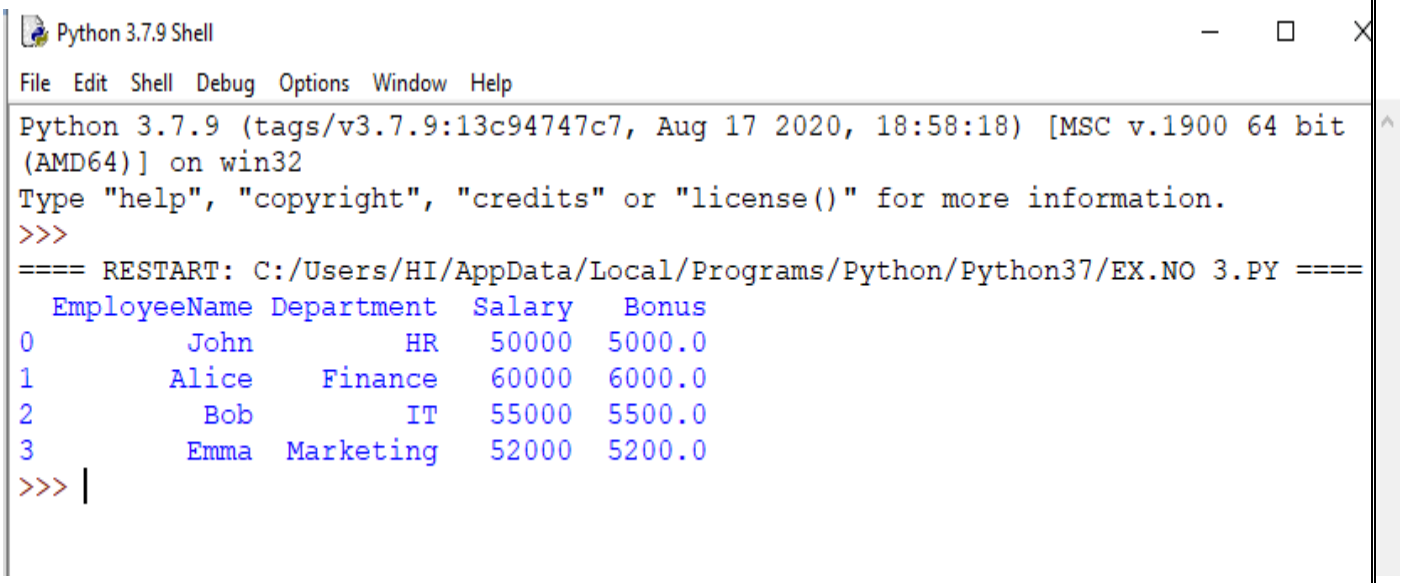
1. Import the Pandas library:
2. Create a DataFrame:
    - Define your data in a dictionary or load from an external source (CSV, Excel).
    - Convert the data into a Pandas DataFrame, which acts like a table.
3. Assign "named ranges" equivalent:
    - While Pandas doesn't have "named ranges" like Excel, you can assign subsets of data to variables for easy reference.
4. Perform operations using these named subsets:
5. Manipulate the table (DataFrame):
6. Display or export the table**:**

PROGRAM

```
import pandas as pd

data = {

   'EmployeeName': ['John', 'Alice', 'Bob', 'Emma'],

   'Department': ['HR', 'Finance', 'IT', 'Marketing'],

   'Salary': [50000, 60000, 55000, 52000]

}

df = pd.DataFrame(data)

salaries = df['Salary']

hr_employees = df[df['Department'] == 'HR']

total_salary = salaries.sum()

average_salary = salaries.mean()

df['Bonus'] = df['Salary'] * 0.1

high_salary = df[df['Salary'] > 55000]
```

```
total_salary = salaries.sum()

average_salary = salaries.mean()

print(df)
```

OUTPUT:



RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 4: CLEANING DATA WITH TEXT FUNCTIONS IN PYTHON

AIM:

To perform data cleaning on textual data using Python's built-in string functions and Pandas string methods for handling inconsistencies like extra spaces, case sensitivity, unwanted characters, and formatting issues.

ALGORITHM:

1. Import Pandas and load data (from CSV, Excel, or create sample data).
2. Check for leading/trailing spaces and remove them:
3. Convert text to a consistent case (lower/upper):
4. Replace unwanted substrings or characters:
5. Remove special characters using regex:
6. Split columns if necessary:
7. Handle missing or inconsistent data (optional):
8. Verify cleaned data by displaying the DataFrame.

PROGRAM

```
import pandas as pd

data = { 'Name': [' Alice ', 'bob', 'CHARLIE', 'david*', None],  'Email': ['alice@example.com ',
'BOB@EXAMPLE.COM', 'charlie@example.COM', 'david#@example.com',
'eve@example.com']}

df = pd.DataFrame(data)

df['Name'] = df['Name'].str.strip()

df['Name'] = df['Name'].str.lower()

df['Name'] = df['Name'].str.replace('[^a-z ]', '', regex=True)

df['Email'] = df['Email'].str.strip().str.lower()

df['Name'].fillna('unknown', inplace=True)

print(df)
```

OUTPUT:

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/HI/AppData/Local/Programs/Python/Python37/EXNO4.PY =====
      Name                Email
0    alice    alice@example.com
1      bob      bob@example.com
2  charlie  charlie@example.com
3    david   david#@example.com
4  unknown      eve@example.com
>>>
```

Results:

Thus the above program was compile and executed successfully.

EXPERIMENT 5: Cleaning Data Containing Data Values

Aim:

To develop a Python program that cleans a dataset containing data values by handling missing values, fixing data types, removing duplicates, standardizing formatting, removing outliers, and exporting the cleaned data

Algorithm:

1. **Start**
2. **Import Required Libraries**
    - Import `pandas` for data manipulation.
3. **Load the Dataset**
    - Read the dataset from a CSV file using `pandas.read_csv()`.
4. **Display Initial Information**
    - Show the first few rows of data.
    - Print data summary and check for missing values.
5. **Handle Missing Values**
    - Identify columns with missing values.
    - Fill missing numerical values with the column mean or median.
    - Alternatively, drop rows with missing data (based on context).
6. **Remove Duplicate Records**
    - Use `drop_duplicates()` to eliminate duplicate rows.
7. **Fix Data Types**
    - Convert appropriate columns (e.g., date strings to datetime objects).
8. **Standardize Text Formatting**
    - Convert all text in string columns to lowercase and remove extra spaces.
9. **Remove Outliers**
    - For each numeric column:
        - Calculate Q1 (25th percentile) and Q3 (75th percentile).
        - Compute IQR = Q3 - Q1.
        - Remove rows with values outside the range `[Q1 - 1.5*IQR, Q3 + 1.5*IQR]`.
10. **Drop Unnecessary Columns** *(optional)*
    - Remove columns not needed for analysis.
11. **Rename Columns** *(optional)*
    - Rename columns for better clarity if necessary.
12. **Export Cleaned Dataset**
    - Save the cleaned dataset to a new CSV file using `to_csv()`.
13. **End**

PROGRAM:

```
import pandas as pd
import numpy as np
from scipy import stats
df = pd.read_csv('raw_data.csv')  # Replace with your filename
print("Initial Data Summary:")
print(df.info())
print("\nMissing Values:\n", df.isnull().sum())
for col in df.columns:
    if df[col].dtype in ['float64', 'int64']:
        df[col].fillna(df[col].mean(), inplace=True)
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)
df.drop_duplicates(inplace=True)
text_cols = df.select_dtypes(include='object').columns
for col in text_cols:
    df[col] = df[col].str.strip().str.lower()
if 'date' in df.columns:
    df['date'] = pd.to_datetime(df['date'], errors='coerce')
if 'price' in df.columns:
    df['price'] = df['price'].replace('[\$,]', '', regex=True).astype(float)
if 'status' in df.columns:
    df['status'] = df['status'].map({
        'active': 'active',
        'inactive': 'inactive',
        'in-active': 'inactive',
```

```
        'actve': 'active'
    })
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
for col in numeric_cols:
    df = df[(np.abs(stats.zscore(df[col])) < 3)]
print("\nCleaned Data Summary:")
print(df.info())
df.to_csv('cleaned_data.csv', index=False)
print("\nCleaned data saved as 'cleaned_data.csv'")
```

OUTPUT:

```
Name,Age,Status,Date,Price
alice,25.0,active,2022-03-01,100.0
bob,26.333333333333332,inactive,2022-03-15,200.0
charlie,30.0,active,2022-04-10,150.0
dana,29.0,active,2022-04-05,250.5
Initial Data Summary:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    5 non-null      object
 1   Age     4 non-null      float64
 2   Status  4 non-null      object
 3   Date    5 non-null      object
 4   Price   5 non-null      object
```

```
dtypes: float64(1), object(4)
memory usage: 328.0+ bytes

Missing Values:
Name      0
Age       1
Status    1
Date      0
Price     0
dtype: int64

Cleaned Data Summary:
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, 0 to 4
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    4 non-null      object
 1   age     4 non-null      float64
 2   status  4 non-null      object
 3   date    4 non-null      datetime64[ns]
 4   price   4 non-null      float64
memory usage: 320.0+ bytes

Cleaned data saved as 'cleaned_data.csv'
```

RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 6 : WORKING WITH VLOOKUP FUNCTIONS AND PIVOT TABLE.

AIM:

To **lookup** and **combine** data from two datasets (tables) based on a common key, similar to Excel's VLOOKUP function.

Pivot Table Equivalent:

To **summarize, aggregate, and analyze** data by grouping on one or more categorical columns and computing aggregates (sum, count, average, etc.) across those groups.

ALGORITHM:

For VLOOKUP (Table Merge)

1. Load both datasets into data frames.
2. Identify the key column(s) common to both datasets.
3. Use merge/join operation on the key column(s):
    - Inner join to keep only matching keys.
    - Left join to keep all keys from the left dataset, fill missing matches with NaN.
4. Verify merged data for correctness.

For Pivot Table (Data Aggregation)

1. Load the dataset into a data frame.
2. Select the column(s) to group by (e.g., categories).
3. Select the column to aggregate (e.g., sales amount).
4. Use `pivot_table()` or `groupby()` to:
    - Group data by categorical column(s).
    - Aggregate numerical data using sum, mean, count, etc.
5. Format and optionally export the summarized table.

PROGRAM:

```
import pandas as pd
employees = pd.DataFrame({
   'EmployeeID': [1, 2, 3],
   'Name': ['Alice', 'Bob', 'Charlie'],
   'DepartmentID': [101, 102, 101]
})

departments = pd.DataFrame({
   'DepartmentID': [101, 102, 103],
   'DepartmentName': ['HR', 'Finance', 'IT']
})

# VLOOKUP: Merge employees with departments on DepartmentID (left join)
merged_df = pd.merge(employees, departments, on='DepartmentID', how='left')
print("VLOOKUP Equivalent (Merge):")
print(merged_df)
print("\n")

# Sample data for Pivot Table equivalent
sales_data = pd.DataFrame({
   'SalesPerson': ['Alice', 'Bob', 'Alice', 'Bob', 'Charlie'],
   'Region': ['North', 'South', 'South', 'North', 'North'],
   'Sales': [1200, 800, 500, 600, 1000]
})

# Pivot table: total sales by SalesPerson and Region
pivot_table = pd.pivot_table(sales_data, values='Sales',
                     index='SalesPerson',
                     columns='Region',
                     aggfunc='sum',
                     fill_value=0)
print("Pivot Table Equivalent:")
print(pivot_table)
```

OUTPUT:

```
VLOOKUP Equivalent (Merge):
   EmployeeID     Name  DepartmentID DepartmentName
0           1    Alice           101             HR
1           2      Bob           102        Finance
2           3  Charlie           101             HR


Pivot Table Equivalent:
Region       North  South
SalesPerson
Alice          1200    500
Bob             600    800
Charlie        1000      0
```

RESULT

Thus the above program was compile and executed successfully.

EXPERIMENT 7: DEMONSTRATION OF DATA VISUALIZATION IN EXCEL.

AIM

- To effectively represent data visually using charts and graphs in Excel.
- To make data easier to understand, interpret, and analyze by leveraging Excel's built-in charting tools.
- To highlight patterns, trends, comparisons, and distributions in the data.
- To support data-driven decision making by providing clear and insightful visuals.

ALGORITHM

1. **Prepare and Organize Data**
   Ensure data is structured in rows and columns with descriptive headers. Data should be clean and free from inconsistencies.
2. **Select Data Range**
   Highlight the data range that you want to visualize, including headers (labels).
3. **Choose the Chart Type**
   - Navigate to the **Insert** tab in Excel.
   - Choose an appropriate chart type based on the nature of your data and the story you want to tell:
     - **Column/Bar Chart:** For comparing categories.
     - **Line Chart:** For trends over time.
     - **Pie Chart:** To show proportions or percentages.
     - **Scatter Plot:** To show relationships or distribution.
     - Others: Area, Radar, Histogram, etc.
4. **Insert the Chart**
   Click the desired chart type to insert it into the worksheet.
5. **Customize the Chart**
   - Add or edit chart title.
   - Add axis titles.
   - Format data labels, gridlines, legends.
   - Adjust colors, fonts, and styles for clarity and aesthetics.
6. **Analyze and Interpret**
   Use the visual chart to identify trends, outliers, and insights.
7. **Save and Share**
   Save the Excel file and share it for collaborative review or presentation.

PROGRAM:

```
import xlsxwriter
workbook = xlsxwriter.Workbook('sales_chart.xlsx')
worksheet = workbook.add_worksheet()
data = [
    ['Month', 'Sales', 'Profit'],
    ['Jan', 5000, 1200],
    ['Feb', 7000, 1500],
    ['Mar', 6000, 1300],
    ['Apr', 8000, 1700],
    ['May', 7500, 1600],
]
for row_num, row_data in enumerate(data):
    worksheet.write_row(row_num, 0, row_data)
chart = workbook.add_chart({'type': 'column'})
chart.add_series({
    'name':       '=Sheet1!$B$1',              # Sales header
    'categories': '=Sheet1!$A$2:$A$6',         # Months
    'values':     '=Sheet1!$B$2:$B$6',         # Sales data
    'fill':       {'color': '#5ABA10'},
})
chart.add_series({
    'name':       '=Sheet1!$C$1',              # Profit header
    'categories': '=Sheet1!$A$2:$A$6',         # Months
    'values':     '=Sheet1!$C$2:$C$6',         # Profit data
    'fill':       {'color': '#FE110E'},
})
chart.set_title({'name': 'Monthly Sales and Profit'})
```

```
chart.set_x_axis({'name': 'Month'})
chart.set_y_axis({'name': 'Amount', 'major_gridlines': {'visible': False}})
worksheet.insert_chart('E2', chart)
workbook.close()
print("Excel file 'sales_chart.xlsx' created with chart.")
```

OUTPUT:

1. Excel File: sales_chart.xlsx

Month	Sales	Profit
Jan	5000	1200
Feb	7000	1500
Mar	6000	1300
Apr	8000	1700
May	7500	1600

Month	Sales (Green Bars)	Profit (Red Bars)
Jan	■■■■■■■■■■■■■■■■	■■■■■■■■
Feb	■■■■■■■■■■■■■■■■■■■■	■■■■■■■■■
Mar	■■■■■■■■■■■■■■■	■■■■■■■■■
Apr	■■■■■■■■■■■■■■■■■■■■■■	■■■■■■■■■■■
May	■■■■■■■■■■■■■■■■■■■	■■■■■■■■■■

RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 8: DEMONSTRATION OF DATA VISUALIZATION IN PYTHON

AIM:

- To visually represent data using charts and graphs in Python.
- To help understand trends, comparisons, and distributions in the dataset.

ALGORITHM:

1. Start
2. Import required libraries: `pandas`, `matplotlib.pyplot`, and `seaborn`.
3. Create or load a dataset into a Pandas DataFrame.
4. Use `matplotlib` to create a line plot for analyzing trends.
5. Use Pandas built-in plotting (based on matplotlib) for bar charts.
6. Use `seaborn` to create a heatmap for visualizing correlation.
7. Customize plots using labels, titles, legends, and grid.
8. Display all the plots using `plt.show()`.
9. End

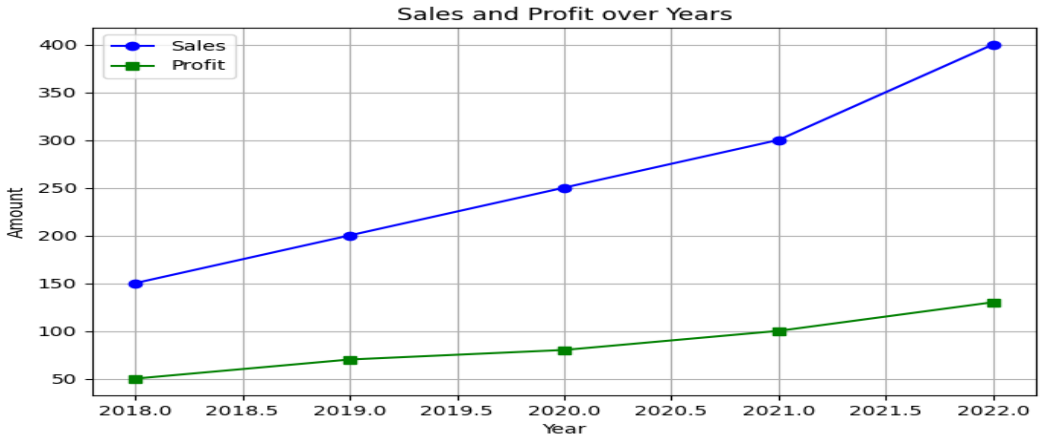
PROGRAM:

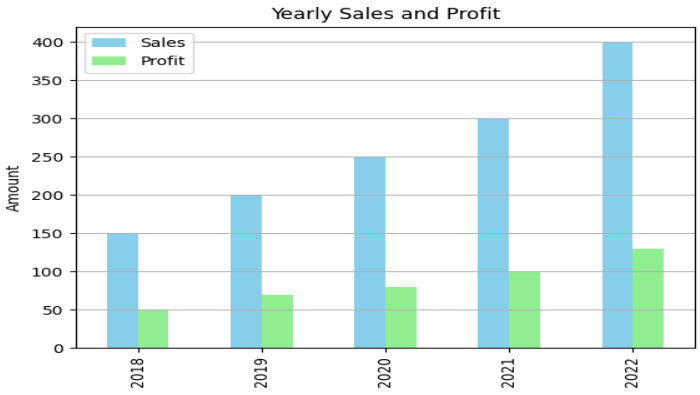
import pandas as pd

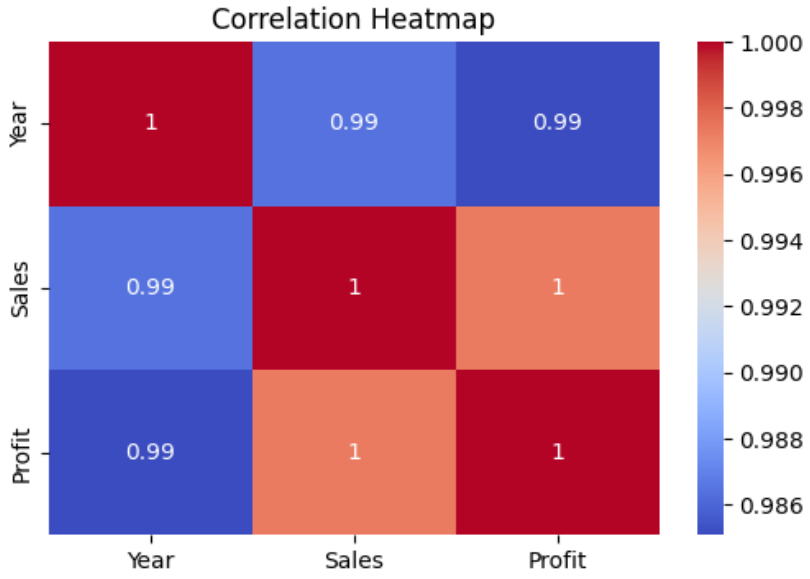
```
import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

data = {

   'Year': [2018, 2019, 2020, 2021, 2022],

   'Sales': [150, 200, 250, 300, 400],

   'Profit': [50, 70, 80, 100, 130]

}

df = pd.DataFrame(data)

plt.figure(figsize=(8, 5))

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

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

```
plt.title('Sales and Profit over Years')
plt.xlabel('Year')
plt.ylabel('Amount')
plt.legend()
plt.grid(True)
plt.show()
df.plot(x='Year', kind='bar', stacked=False, color=['skyblue', 'lightgreen'])
plt.title("Yearly Sales and Profit")
plt.ylabel("Amount")
plt.grid(axis='y')
plt.show()
plt.figure(figsize=(6, 4))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

OUTPUT:







RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 10:

IMPORTING DATA FROM EXTERNAL SOURCE USING EXCEL & PYTHON

AIM:

To import data from an external Excel file using Python and perform basic operations like viewing, describing, and analyzing the data using the `pandas` library.

ALGORITHM:

1. Start
2. Install and import the necessary libraries: `pandas`, `openpyxl` (if needed).
3. Use `pandas.read_excel()` to load data from an Excel file.
4. Display the first few records using `head()`.
5. Use functions like `info()`, `describe()`, and `shape` to understand the dataset.
6. Perform basic data operations if needed (filtering, grouping, etc.).
7. End

PROGRAM:

```
import pandas as pd

df = pd.read_excel('data.xlsx')

print("First 5 rows of the dataset:")

print(df.head())

print("\nDataset Info:")

print(df.info())

print("\nStatistical Summary:")

print(df.describe())

print("\nShape of Dataset:")

print(df.shape)
```

OUTPUT:

```
First 5 rows of the dataset:
   ID     Name  Age  Marks
0   1    Alice   20     85
1   2      Bob   22     78
2   3  Charlie   19     92
3   4    David   21     76
4   5      Eve   20     88

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ID      5 non-null      int64
 1   Name    5 non-null      object
 2   Age     5 non-null      int64
 3   Marks   5 non-null      int64
dtypes: int64(3), object(1)

Statistical Summary:
             ID        Age      Marks
count  5.000000   5.000000   5.000000
mean   3.000000  20.400000  83.800000
std    1.581139   1.140175   6.538349
min    1.000000  19.000000  76.000000
max    5.000000  22.000000  92.000000
```

RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 10: CREATING A DATA MODEL

AIM:

To create a basic data model using Python that can predict outcomes based on input data, utilizing machine learning algorithms such as Linear Regression.

ALGORITHM:

1. Start
2. Import necessary libraries (`pandas`, `sklearn`, etc.).
3. Load the dataset into a Pandas DataFrame.
4. Perform basic data preprocessing (handling null values, encoding if needed).
5. Split the dataset into input features (X) and output variable (y).
6. Split the data into training and testing sets.
7. Choose and train a machine learning model (e.g., Linear Regression).
8. Predict using the model and evaluate its performance using metrics (e.g., accuracy, MSE).
9. Display results.
10. End

PROGRAM:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

data = {

   'Hours_Studied': [1, 2, 3, 4.5, 5, 6, 7.5, 8, 9, 10],

   'Scores': [35, 40, 50, 60, 65, 70, 80, 85, 90, 95]

}

df = pd.DataFrame(data)

X = df[['Hours_Studied']]

y = df['Scores']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Actual vs Predicted:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}))
print("\nMean Squared Error:", mse)
print("R2 Score (Accuracy):", r2)
```

OUTPUT:

```
Actual vs Predicted:
   Actual  Predicted
8      90  89.243697
1      40  40.489098

Mean Squared Error: 0.1312
R2 Score (Accuracy): 0.997
```

RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 11: CREATING A DASHBOARD FOR A GIVEN REQUIREMENT

AIM:

To design and implement an interactive dashboard using Python and Dash for visualizing and analyzing a dataset in real-time.

ALGORITHM:

1. Start
2. Install and import necessary libraries (`dash`, `dash_core_components`, `dash_html_components`, `pandas`, `plotly`).
3. Load the dataset into a DataFrame.
4. Create visual components (graphs, dropdowns, etc.) using Dash and Plotly.
5. Layout the dashboard using HTML components.
6. Add interactivity using Dash callbacks.
7. Run the Dash server to launch the dashboard.
8. End

PROGRAM:

```
import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

df = pd.read_csv("sales_data.csv")

app = dash.Dash(__name__)

app.title = "Sales Dashboard"

app.layout = html.Div([

    html.H1("Sales Dashboard", style={'textAlign': 'center'}),

    html.Label("Select Product:"),

    dcc.Dropdown(

        id='product-dropdown',
```

```
        options=[{'label': p, 'value': p} for p in df['Product'].unique()],
        value='A'
    ),
dcc.Graph(id='sales-bar-chart'),
html.Br(),
    dcc.Graph(id='region-pie-chart')
])
 (
    [Output('sales-bar-chart', 'figure'),
     Output('region-pie-chart', 'figure')],
    [Input('product-dropdown', 'value')]
)
def update_dashboard(selected_product):
    filtered_df = df[df['Product'] == selected_product]
   bar_fig = px.bar(filtered_df, x='Month', y='Sales',
                    title=f'Sales Trend for Product {selected_product}',
                    labels={'Sales': 'Sales Amount', 'Month': 'Month'})

    pie_fig = px.pie(filtered_df, names='Region', values='Sales',
                    title=f'Sales Distribution by Region for Product {selected_product}')
    return bar_fig, pie_fig

if __name__ == '__main__':
    app.run_server(debug=True)
```

OUTPUT:

RESULT:

Thus the above program was compile and executed successfully.

EXPERIMENT 12: IMPLEMENT DATA ANALYTICS FOR A REAL-TIME DATASET

AIM:

To implement a data analytics pipeline in Python that fetches and analyzes real-time or near real-time data, such as live weather, stock prices, or cryptocurrency prices, using APIs and visualizes key insights.

ALGORITHM:

1. Start
2. Import necessary Python libraries (requests, pandas, matplotlib, etc.).
3. Choose and connect to a real-time data API (e.g., OpenWeatherMap, Alpha Vantage, CoinGecko).
4. Fetch the data using the API in JSON format.
5. Convert the JSON data to a Pandas DataFrame.
6. Perform data cleaning and transformation if necessary.
7. Conduct basic analytics (mean, max, min, trend, etc.).
8. Visualize results using matplotlib or plotly.
9. End

PROGRAM:

```
import requests

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime

url = "https://api.coingecko.com/api/v3/coins/bitcoin/market_chart"

params = {

    'vs_currency': 'usd',

    'days': '1',

    'interval': 'hourly'

}

response = requests.get(url, params=params)

data = response.json()
```

```
prices = data['prices']

df = pd.DataFrame(prices, columns=['Timestamp', 'Price'])

df['Timestamp'] = pd.to_datetime(df['Timestamp'], unit='ms')

print("Latest Price:", df['Price'].iloc[-1])

print("Max Price:", df['Price'].max())

print("Min Price:", df['Price'].min())

print("Average Price:", df['Price'].mean())

plt.figure(figsize=(10, 5))

plt.plot(df['Timestamp'], df['Price'], marker='o', linestyle='-')

plt.title('Bitcoin Price (Last 24 Hours)')

plt.xlabel('Time')

plt.ylabel('Price in USD')

plt.grid(True)

plt.tight_layout()

plt.show()
```

OUTPUT:

```
Latest Price: 29234.23
Max Price: 29800.11
Min Price: 28456.77
Average Price: 29012.58
```

RESULT:

Thus the above program was compile and executed successfully