The Libyan Academy
School of Applied Sciences and Engineering
Department of Electrical and Computer Engineering
Branch: Information Technology

Special Topics - ITE697

IoT Big Data Analytics - Labs

By:
Fooz Barakat

Labs:

Chapter 1: Data and Internet of Things

Chapter 2: Fundamentals of Data Analysis

Chapter 3: Data Analysis

Chapter 4: Advanced Analytics and machine learning

Chapter 5: Storytelling with Data

Chapter 6: Architecture for Big Data and Data Engineering

Chapter 1: Data and Internet of Things

- 1.1.2.3 - Demonstrate the Growth of Data with VNI
- 1.2.2.3 - Explore Sources of Open Data
- 1.2.2.4 - Where is My Data?
- 1.3.2.6 - Demonstrate Spreadsheet Limitations in Data Analysis
- 1.3.2.8 - Introduction to PL-App
- 1.3.2.9 - Take the Python Challenge

Chapter 2: Fundamentals of Data Analysis

- **2.2.2.5 - Basic Data Analysis**
- 2.2.4.5 - San Francisco Crime
- 2.3.2.7 - Preparing Data
- 2.4.1.5 - Big Data Ethics
- 2.5.1.4 - Internet Meter Data Analysis
- 2.5.2.4 - Working with Python and SQLite
- 2.5.2.5 - Internet Meter SQL

Objectives:

Use very simple methods to describe existing data, fill-in missing data values and to make simple predictions

2.2.2.5 - Basic Data Analysis _ Worksheet 1

Used for data manipulation and analysis,including reading CSV files and working with data frames

Used for creating visualizations, such as plots and charts, to help understand the data

2.2.2.5 - Lab

```python
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV file
worksheet1 = pd.read_csv("worksheet1.csv")

# Read each column
median_age_males = worksheet1['median_age_males']
median_age_females = worksheet1['median_age_females']

# Calculate the minimum and maximum median ages for males and females
male_min_age = median_age_males.min()
male_max_age = median_age_males.max()
female_min_age = median_age_females.min()
female_max_age = median_age_females.max()
```

Year	Median Age of Males	Median Age of Females
1890	26.1	22.0
1900	25.9	21.9
1910	25.1	21.6
1920	24.6	21.2
1930	24.3	21.3
1940	24.3	21.5
1950	22.8	20.3
1960	22.8	20.3
1970	23.2	20.8
1980	24.7	22.0
1990	26.1	23.9
2000	26.8	25.1

Console:

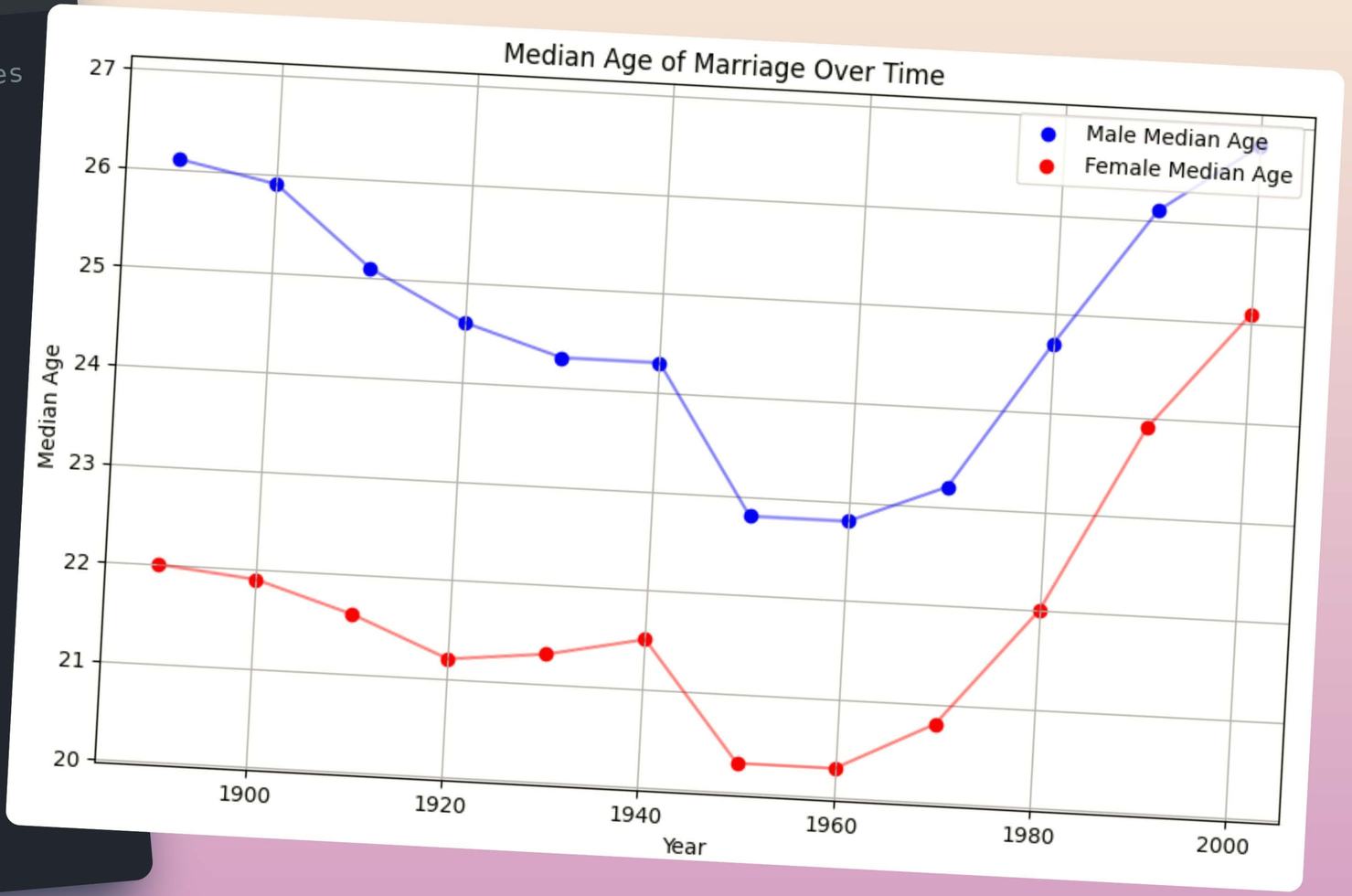
Male median age:
22.8 to 26.8

Female median age:
20.3 to 25.1

2.2.2.5 - Basic Data Analysis _ Worksheet 1

2.2.2.5 - Lab

```python
# Plotting the median age of marriage over time for males
#and females
plt.figure(figsize=(10, 6))
plt.scatter(year, median_age_males, color='blue',
label='Male Median Age', marker='o')
plt.plot(year, median_age_males, color='blue',
linestyle='-', alpha=0.5)
plt.scatter(year, median_age_females, color='red',
label='Female Median Age', marker='o')
plt.plot(year, median_age_females, color='red',
linestyle='-', alpha=0.5)

# Adding labels and title
plt.xlabel('Year')
plt.ylabel('Median Age')
plt.title('Median Age of Marriage Over Time')
plt.legend(loc='upper right')

# Display plot
plt.grid(True)
plt.show()
```



2.2.2.5 - Basic Data Analysis _ Worksheet 2

Used for numerical operations and array manipulation, often used for data preparation and calculations in data analysis and machine learning

Used for machine learning tasks like linear regression in this case, helping to analyze relationships in the data

Year	Men hours housework week	Woman hours housework week
1965	4.4	31.9
1975	6	23.6
1985	10.2	20.7
1995	10.2	18.9
2000	10	18.6
2005	9.2	19.1
2010	10	17.4
2015	9.8	17.8

2.2.2.5 - Lab

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Read the CSV file
worksheet2 = pd.read_csv("worksheet2.csv")

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

# Plot original data for men
plt.scatter(worksheet2['year'], worksheet2['men_hours_housework_week'], color='blue', label='Men (Original)', marker='o')
plt.plot(worksheet2['year'], worksheet2['men_hours_housework_week'], color='blue', linestyle='-', alpha=0.5)
```

Same code is implemented for the women data

2.2.2.5 - Basic Data Analysis _ Worksheet 2

Interpolated data fills in missing values within the known range, providing a smoother trend representation.

2.2.2.5 - Lab

```python
# Add missing years
missing_years = [1970, 1980, 1990]
for year in missing_years:
    worksheet2.loc[len(worksheet2)] = [year, np.nan, np.nan]

# Sort the dataframe again
worksheet2.sort_values('year', inplace=True)

# Interpolate missing values
worksheet2['men_hours_housework_week'] = worksheet2['men_hours_housework_week'].interpolate(method='linear')
worksheet2['women_hours_housework_week'] = worksheet2['women_hours_housework_week'].interpolate(method='linear')

# Plot interpolated data for men
plt.scatter(worksheet2['year'], worksheet2['men_hours_housework_week'], color='blue', label='Men (Interpolated)', marker='x')
plt.plot(worksheet2['year'], worksheet2['men_hours_housework_week'], color='blue', linestyle='--', alpha=0.5)
```

This pandas function is used to fill in missing data (NaN values) in a DataFrame

Year	Men hours housework week	Woman hours housework week
1965	4.4	31.9
1975	6	23.6
1985	10.2	20.7
1995	10.2	18.9
2000	10	18.6
2005	9.2	19.1
2010	10	17.4
2015	9.8	17.8

'linear' interpolation means that the missing values are estimated based on a linear interpolation between the two nearest data points

2.2.2.5 - Basic Data Analysis _ Worksheet 2

Extrapolated data estimates values outside the known range, projecting future trends based on existing data.

2.2.2.5 - Lab

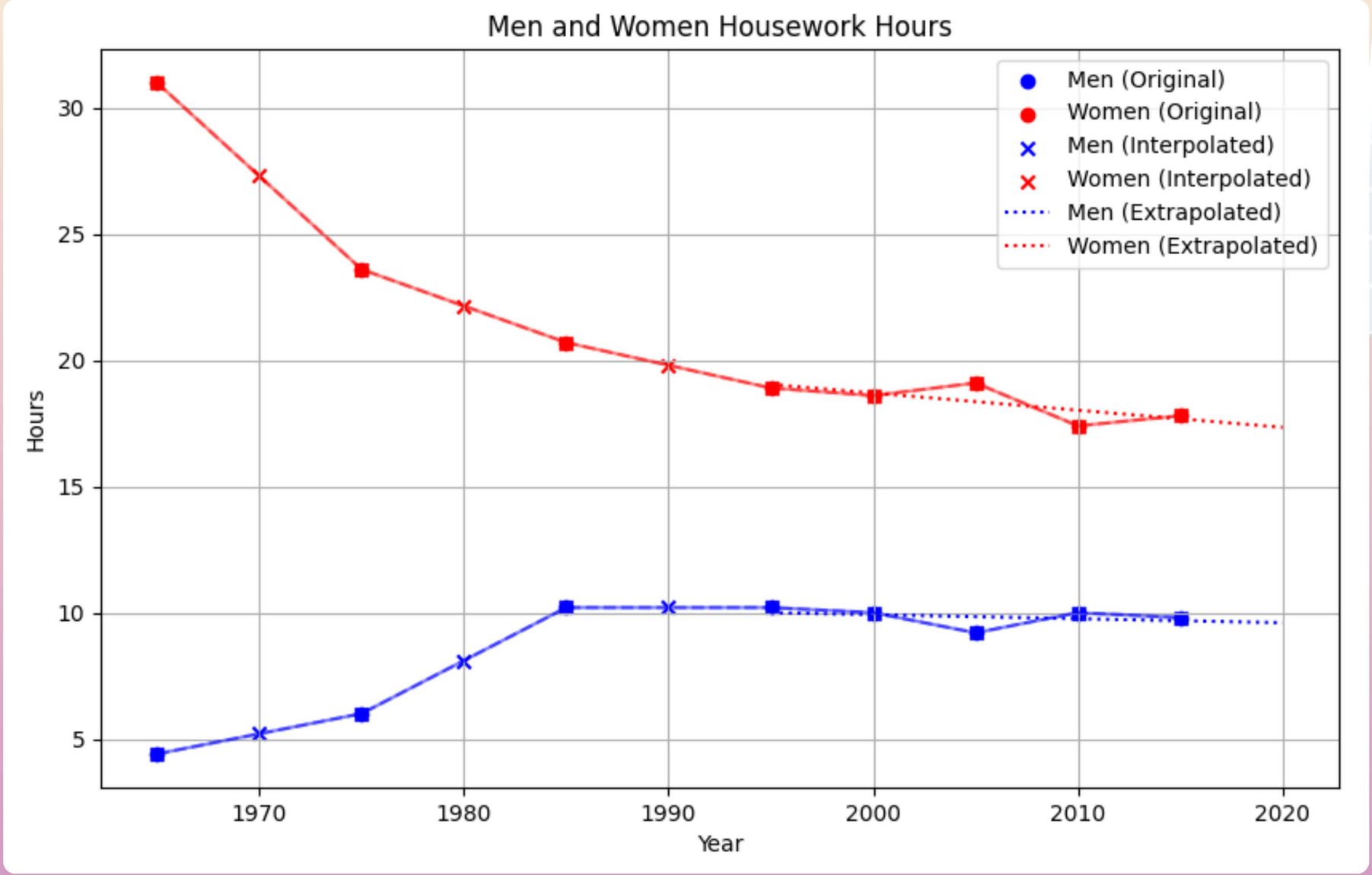
```
# Get the data for the previous five periods and reshaping it into a 2D array
X = worksheet2['year'][-5:].values.reshape(-1, 1)
#  Extracting the corresponding values for men and women for the previous five periods
y_men = worksheet2['men_hours_housework_week'][-5:].values

# Fit a linear regression model for men
model_men = LinearRegression()
model_men.fit(X, y_men)

# Extrapolate the values for the year 2020 for men and wo
year_2020 = 2020
# Creating an array containing the year 2020 for extrapo
X_2020 = np.array([[year_2020]])
extrapolated_value_men = model_men.predict(X_2020)
extrapolated_value_women = model_women.predict(X_2020)

# Plot extrapolated line for men
plt.plot([X[0][0], year_2020], [model_men.predict([[X[0][0]]])[0], extrapolated_value_men[0]], color='blue',
linestyle=':', label='Men (Extrapolated)')
```

Same code is implemented for the women data

'predict' function is used to predict the target variable (in this case, housework hours) based on the input features (in this case, the year) using the trained linear regression model. It takes the input features (X) as input and returns the predicted target variable values

Year	Men hours housework week	Woman hours housework week
1965	4.4	31.9
1975	6	23.6
1985	10.2	20.7
1995	10.2	18.9
2000	10	18.6
2005	9.2	19.1
2010	10	17.4
2015	9.8	17.8

2.2.2.5 - Basic Data Analysis _ Worksheet 2



Year	Men hours housework week	Woman hours housework week
1965	4.4	31.9
1975	6	23.6
1985	10.2	20.7
1995	10.2	18.9
2000	10	18.6
2005	9.2	19.1
2010	10	17.4
2015	9.8	17.8

Console:

Extrapolated Value Men: 9.6

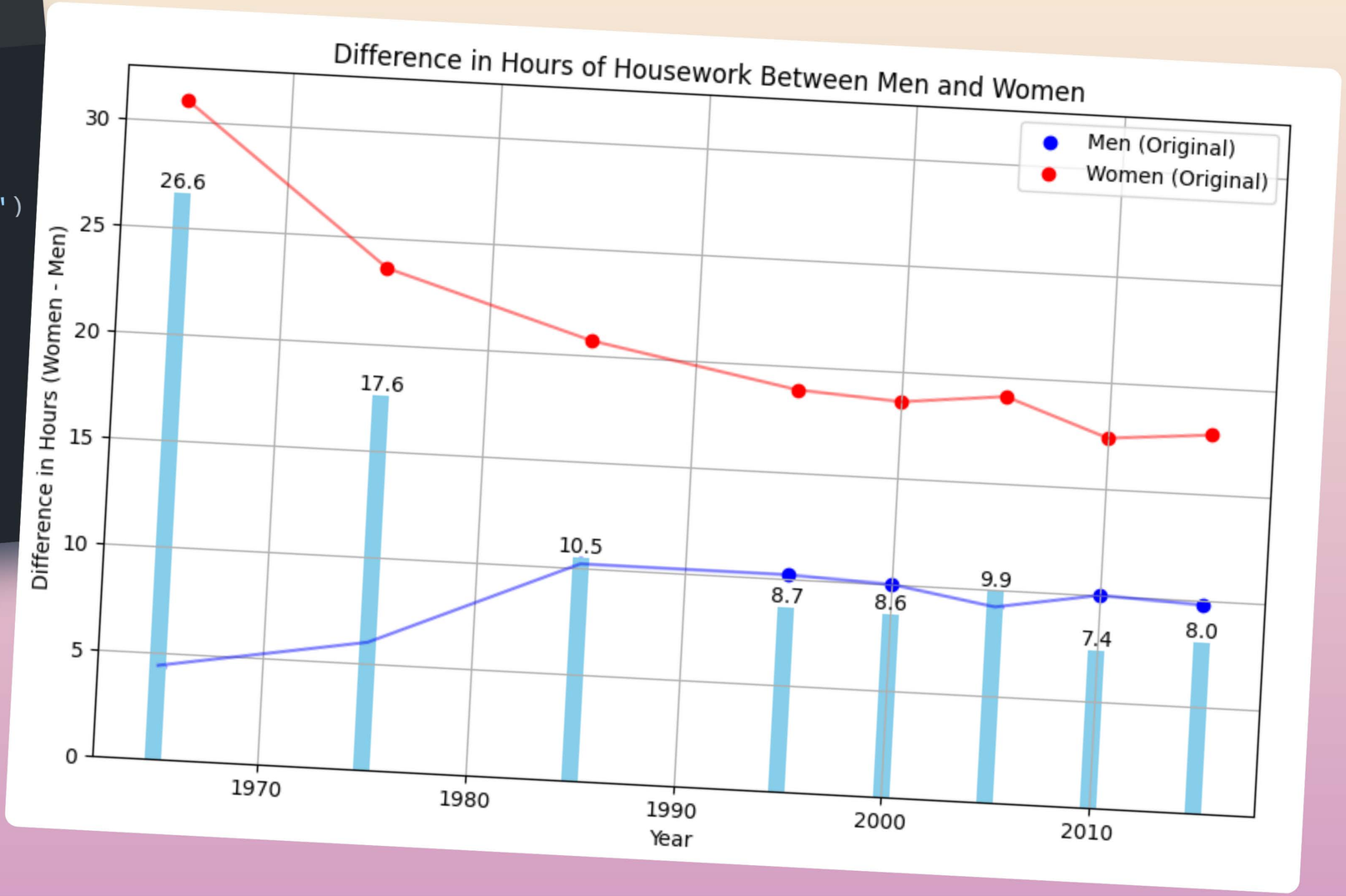
Extrapolated Value Women: 17.43

2.2.2.5 - Basic Data Analysis _ Worksheet 3

2.2.2.5 - Lab

```python
# Plot original data for men
plt.scatter(worksheet2['year'],
worksheet2['men_hours_housework_week'],
color='blue', label='Men (Original)', marker='o')

plt.plot(worksheet2['year'],
worksheet2['men_hours_housework_week'],
color='blue', linestyle='-', alpha=0.5)

plt.bar(worksheet3['year'],
worksheet3['women-men_hours'],
color='skyblue')
```

Year	Men hours housework week	Woman hours housework week	Woman Hours – Men Hours
1965	4.4	31.9	
1975	6	23.6	
1985	10.2	20.7	
1995	10.2	18.9	
2000	10	18.6	
2005	9.2	19.1	
2010	10	17.4	
2015	9.8	17.8	



Chapter 2: Fundamentals of Data Analysis

- 2.2.2.5 - Basic Data Analysis
- **2.2.4.5 - San Francisco Crime**
- 2.3.2.7 - Preparing Data
- 2.4.1.5 - Big Data Ethics
- 2.5.1.4 - Internet Meter Data Analysis
- 2.5.2.4 - Working with Python and SQLite
- 2.5.2.5 - Internet Meter SQL

Objectives:

Demonstrate your knowledge of the Data Analysis Lifecycle using a given set of data and the tools, Python and Jupyter Notebook

2.2.4.5 - San Francisco

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	IncidntNun	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution	Address	X	Y	Location			
2		LARCENY/T	GRAND THE	Sunday	08/31/2014	20:30	CENTRAL	NONE	HYDE ST / C	-122.417	37.79097	(37.7909741243888, -122.417392830334)			
3		LARCENY/T	GRAND THE	Sunday	08/31/2014	14:30	CENTRAL	NONE	COLUMBUS	-122.404	37.7963	(37.7963018736036, -122.404417620748)			
4		LARCENY/T	GRAND THE	Sunday	08/31/2014	11:30	CENTRAL	NONE	SUTTER ST /	-122.407	37.78943	(37.7894347630337, -122.406958660602)			
5		DRUG/NAR	POSSESSIC	Sunday	08/31/2014	17:49	MISSION	ARREST, BC	16TH ST / M	-122.42	37.76505	(37.7650501214965, -122.419671780296)			
6		DRUG/NAR	POSSESSIC	Sunday	08/31/2014	18:05	NORTHERN	ARREST, BC	LARKIN ST /	-122.418	37.78517	(37.7851670875814, -122.417903977564)			
7		LARCENY/T	GRAND THE	Sunday	08/31/2014	0:01	CENTRAL	NONE	GRANT AV /	-122.405	37.78963	(37.7896302267231, -122.405402610955)			
8		WARRANTS	WARRANT /	Sunday	08/31/2014	18:51	RICHMOND	ARREST, BC	MARTIN LU	-122.482	37.76677	(37.7667702165662, -122.482372245817)			
9		VEHICLE TH	STOLEN AU	Sunday	08/31/2014	10:30	INGLESIDE	NONE	200.0 Block	-122.427	37.72345	(37.7234534751554, -122.427016818213)			
10		LARCENY/T	PETTY THEF	Sunday	08/31/2014	16:30	CENTRAL	NONE	100.0 Block	-122.399	37.79698	(37.7969838270893, -122.398859251506)			

Allows to create interactive maps and visualize geospatial data

'apply' function then applies this 'lambda' function to each row in the 'Date' column

'lambda' function is used to create an inline function that takes each value in the 'Date' column (row in this context) and extracts the first two characters (the month digits) as a substring

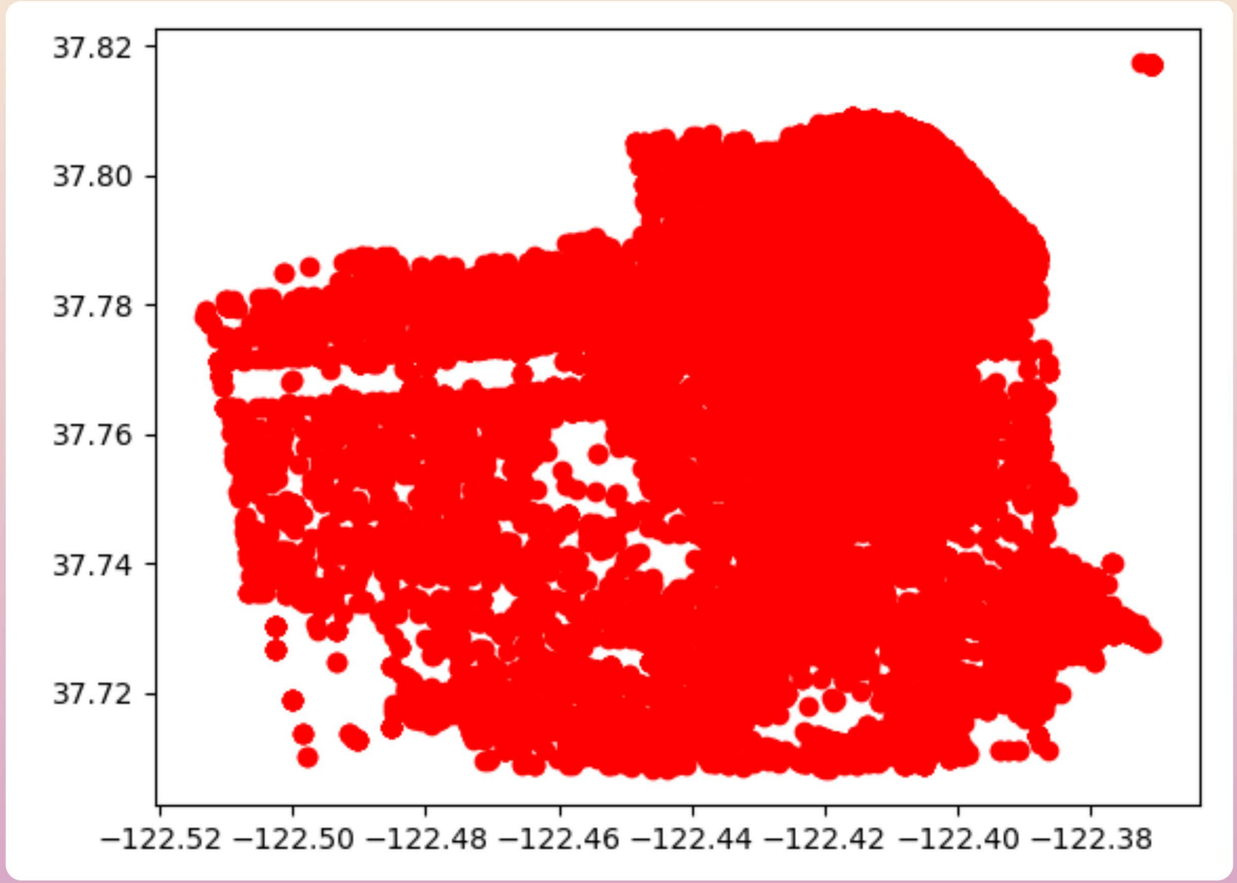
2.2.4.5 - Lab

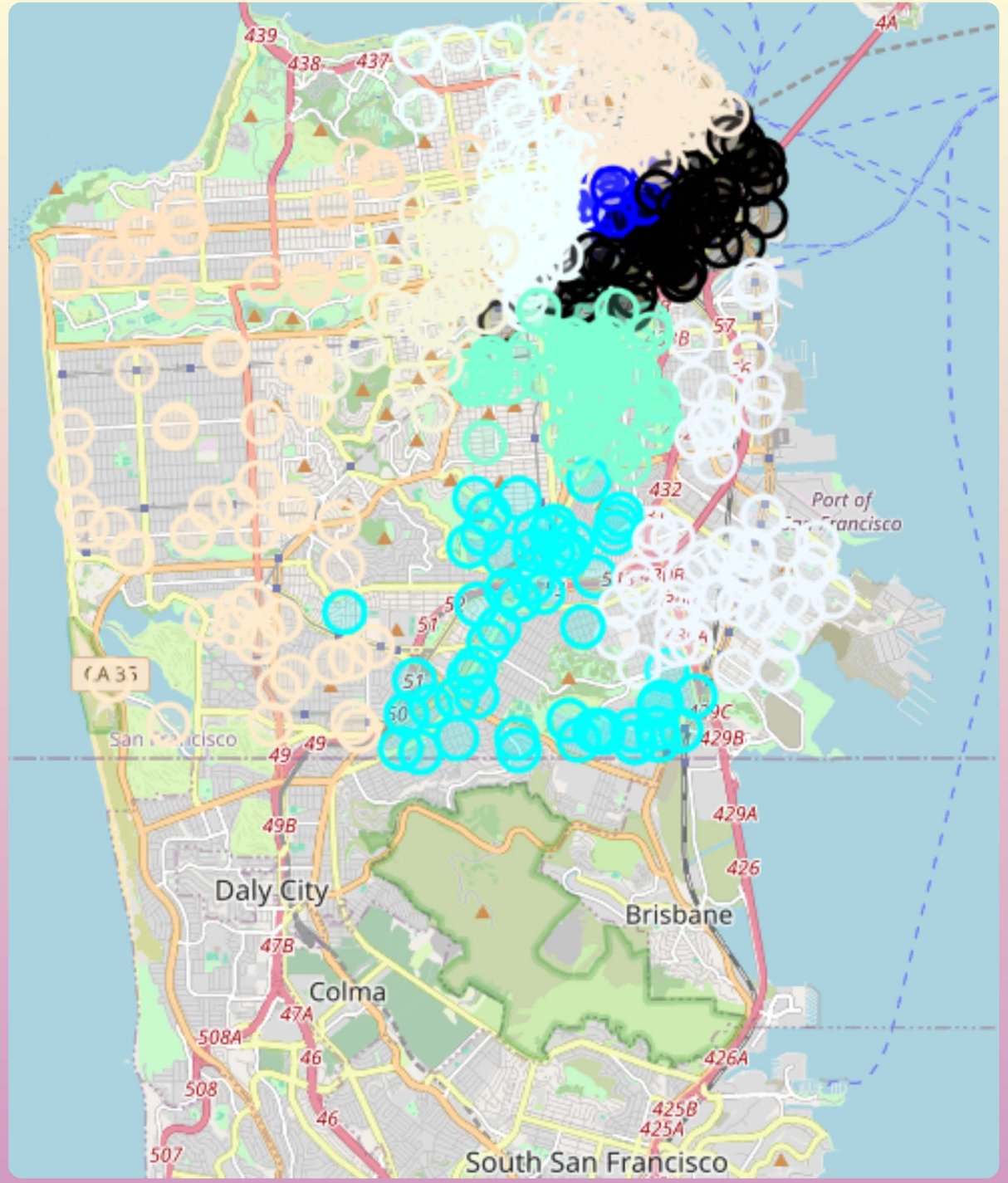
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import folium

# Read the data
SF = pd.read_csv("Map-Crime_Incidents-Previous_Three_Months.csv")
# Extracting Month and Day from Date
SF['Month'] = SF['Date'].apply(lambda row: int(row[0:2]))
SF['Day'] = SF['Date'].apply(lambda row: int(row[3:5]))

# Summarizing Crime Categories
CountCategory = SF['Category'].value_counts()
# Crimes Committed in August
AugustCrimes = SF[SF['Month'] == 8]
# Plotting Crime Locations
plt.plot(SF['X'], SF['Y'], 'ro')
```

2.2.4.5 - Lab

```python
# Mapping Police Department Districts
map_osm = folium.Map(location=[SF['Y'].mean(),
SF['X'].mean()], zoom_start=12)
for el in obs[0:-1:plotEvery]:
    folium.CircleMarker(el[0:2], color=color_dict[el[2]],
     fill_color=el[2],
     radius=10).add_to(map_osm)
map_osm.save('crime_map.html')
```

2.2.4.5 - San Francisco Crime





Chapter 2: Fundamentals of Data Analysis

- 2.2.2.5 - Basic Data Analysis
- 2.2.4.5 - San Francisco Crime
- **2.3.2.7 - Preparing Data**
- 2.4.1.5 - Big Data Ethics
- 2.5.1.4 - Internet Meter Data Analysis
- 2.5.2.4 - Working with Python and SQLite
- 2.5.2.5 - Internet Meter SQL

Objectives:

Part 1: Import the Data into Microsoft Excel
Part 2: Get the Month out of a Date
Part 3: Split Full Name Column into Different Columns
Part 4: Concatenate Columns

Chapter 2: Fundamentals of Data Analysis

- 2.2.2.5 - Basic Data Analysis
- 2.2.4.5 - San Francisco Crime
- 2.3.2.7 - Preparing Data
- **2.4.1.5 - Big Data Ethics**
- 2.5.1.4 - Internet Meter Data Analysis
- 2.5.2.4 - Working with Python and SQLite
- 2.5.2.5 - Internet Meter SQL

Objectives:

Use the discussion boards to start, or contribute to, a topic based on the ethics of using Big Data

Chapter 2: Fundamentals of Data Analysis

- 2.2.2.5 - Basic Data Analysis
- 2.2.4.5 - San Francisco Crime
- 2.3.2.7 - Preparing Data
- 2.4.1.5 - Big Data Ethics
- **2.5.1.4 - Internet Meter Data Analysis**
- 2.5.2.4 - Working with Python and SQLite
- 2.5.2.5 - Internet Meter SQL

Objectives:

Part 1: Collect and Store Data

Part 2: Manipulate Data

2.5.1.4 - Internet Meter Data Analysis

2.5.1.4 - Lab

```python
import datetime
import subprocess
import csv
import pandas as pd

# Define column names
column_names = ['Date & Time', 'Ping (ms)', 'Download (Mbit/s)', 'Upload (Mbit/s)']

# Create an empty DataFrame with the specified column names
df = pd.DataFrame(columns=column_names)
# Save the DataFrame to a CSV file with headers
df.to_csv('./internet_speed_data.csv', sep='\t', index=False)

def speedtest():
    speedtest_cmd = "speedtest-cli --simple"
    process = subprocess.Popen(speedtest_cmd.split(), stdout=subprocess
    process_output = process.communicate()[0].split()
    date_time = datetime.datetime.now().strftime("%d/%m/%Y %H:%M")
    ping = process_output[1].decode('utf-8')
    download = process_output[4].decode('utf-8')
    upload = process_output[7].decode('utf-8')
    return date_time, ping, download, upload
```

Provides classes for manipulating dates and times. It allows you to create, format, and perform operations on dates and times

It provides a way to execute system commands from within Python scripts

Provides functionality for reading and writing CSV

2.5.1.4 - Lab

```python
def save_to_csv(data, filename):
    with open(filename + '.csv', 'a', newline='') as f:
        wr = csv.writer(f, delimiter='\t')
        wr.writerow(data)

for i in range(5):  # Generate 5 rows of data
    speedtest_output = speedtest()
    save_to_csv(speedtest_output, 'internet_speed_data')
```

2.5.1.4 - Internet Meter Data Analysis

A1 | Date & Time | Ping (ms) | Download (Mbit/s) | Upload (Mbit/s)

	A	B	C	D	E	F	G	H	I
1	Date & Time	Ping (ms)	Download (Mbit/s)	Upload (Mbit/s)					
2	23/02/2024 21:37	56.497	20.332	7.27					
3	23/02/2024 21:37	54.588	18.012	1.83					
4	23/02/2024 21:37	50.006	18.255	.73					
5	23/02/2024 21:38	57.972	25.512	2.78					
6	23/02/2024 21:38	64.178	22.267	.60					

2.5.1.4 - Lab

```python
# Read the CSV file with the specified column names
df = pd.read_csv('./internet_speed_data.csv', sep='\t')

# Split the 'Date & Time' column into separate 'Date' and 'Time' columns
df[['Date', 'Time']] = df['Date & Time'].str.split(' ', n=1, expand=True)

# Drop the original 'Date & Time' column
df.drop(columns=['Date & Time'], inplace=True)

# Reorder the columns to place 'Date' and 'Time' as the first columns
cols = df.columns.tolist()
cols = cols[-2:] + cols[:-2]
df = df[cols]

# Save the modified DataFrame to a new CSV file
df.to_csv('./updated_internet_speed_data.csv', index=False)
```

Indicates that the values in the CSV file are separated by tabs

'n=1' specifies that the split operation should be performed only once. 'expand=True' parameter indicates that the split elements should be expanded into separate columns in the DataFrame

Used to modify the DataFrame in place, meaning it will remove the specified column directly from the DataFrame without the need to assign the result back to a variable

2.5.1.4 - Internet Meter Data Analysis

A1 | Date & Time | Ping (ms) | Download (Mbit/s) | Upload (Mbit/s)

	A	B	C	D	E	F	G	H	I
1	Date & Time	Ping (ms)	Download (Mbit/s)	Upload (Mbit/s)					
2	23/02/2024 21:37	56.497	20.33	22.27					
3	23/02/2024 21:37	54.588	18.01	21.83					
4	23/02/2024 21:37	50.006	18.25	5.73					
5	23/02/2024 21:38	57.972	25.51	22.78					
6	23/02/2024 21:38	64.178	22.26	7.6					

A1 | Date

	A	B	C	D	E	F
1	Date	Time	Ping (ms)	Download (	Upload (Mbit/s)	
2	23/02/2024	21:37	56.497	20.33	22.27	
3	23/02/2024	21:37	54.588	18.01	21.83	
4	23/02/2024	21:37	50.006	18.25	5.73	
5	23/02/2024	21:38	57.972	25.51	22.78	
6	23/02/2024	21:38	64.178	22.26	7.6	

Chapter 2: Fundamentals of Data Analysis

- 2.2.2.5 - Basic Data Analysis
- 2.2.4.5 - San Francisco Crime
- 2.3.2.7 - Preparing Data
- 2.4.1.5 - Big Data Ethics
- 2.5.1.4 - Internet Meter Data Analysis
- **2.5.2.4 - Working with Python and SQLite** ✅
- 2.5.2.5 - Internet Meter SQL

Objectives:

Part 1: Create the Database

Part 2: Enter Data and Query the Database

Part 3: Automate the Process with Python

2.5.2.4 - Working with Python and SQLite

Provides a simple way to work with SQLite databases, allowing you to create, connect to, and manipulate SQLite database files

2.5.2.4 - Lab

```python
import sqlite3
import subprocess
import datetime

# Function to create the database and table
def create_database():
    # Connect to the database file
    conn = sqlite3.connect('internet_speed_data.db')
    cursor = conn.cursor()

    # Create the 'data' table if it doesn't exist
    cursor.execute('''CREATE TABLE IF NOT EXISTS data
                      (id INTEGER PRIMARY KEY, date TEXT, time TEXT, ping REAL
                      download REAL, upload REAL)''')

    # Commit changes and close the connection
    conn.commit()
    conn.close()
```

2.5.2.4 - Lab

```python
# Function to insert data into the database
def insert_data(date, time, ping, download, upload):
    conn = sqlite3.connect('internet_speed_data.db')
    cursor = conn.cursor()

    # Insert data into the 'data' table
    cursor.execute(
        "INSERT INTO data
        (date, time, ping, download, upload)
        VALUES (?, ?, ?, ?, ?)",
        (date, time, ping, download, upload))

    # Commit changes and close the connection
    conn.commit()
    conn.close()
```

2.5.2.4 - Working with Python and SQLite

2.5.2.4 - Lab

```python
# Function to query and display all data from the database
def query_data():
    conn = sqlite3.connect('internet_speed_data.db')
    cursor = conn.cursor()

    # Retrieve all rows from the 'data' table
    cursor.execute("SELECT * FROM data")
    rows = cursor.fetchall()

    # Print each row
    print("Query data from the database:")
    for row in rows:
        print(row)

    # Close the connection
    conn.close()
```

2.5.2.4 - Lab

```python
# Main function to run speed tests and store results in the database
def main():
    # Create the database and table if they don't exist
    create_database()

    # Perform speed tests and insert results into the database
    for i in range(2):  # Perform 2 speed tests
        date_time, ping, download, upload = speedtest()
        date, time = date_time.split(' ')
        insert_data(date, time, ping, download, upload)

    # Query and display all data from the database
    query_data()

# Run the main function
if __name__ == "__main__":
    main()
```

2.5.2.4 - Working with Python and SQLite

DB Browser for SQLite - C:\Users\reach\OneDrive\Desktop\IoT\CH 2\3\internet_speed_data.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: data

	id	date	time	ping	download	upload
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	21/02/2024	15:59	31.699	12.27	4.76
2	2	21/02/2024	15:59	47.92	15.62	4.65
3	3	23/02/2024	22:40	53.238	15.69	11.45
4	4	23/02/2024	22:41	45.891	20.32	14.37
5	5	23/02/2024	22:41	52.49	17.14	14.78
6	6	23/02/2024	22:42	49.573	11.86	14.6
7	7	23/02/2024	22:42	50.673	18.52	3.89
8	8	23/02/2024	22:43	50.664	24.02	23.89
9	9	23/02/2024	22:43	54.277	23.33	26.56

```
C:\Users\reach\OneDrive\Desktop\IoT\CH 2\3>py 2.5.2.4.py
Query data from the database:
(1, '21/02/2024', '15:59', 31.699, 12.27, 4.76)
(2, '21/02/2024', '15:59', 47.92, 15.62, 4.65)
(3, '23/02/2024', '22:40', 53.238, 15.69, 11.45)
(4, '23/02/2024', '22:41', 45.891, 20.32, 14.37)
(5, '23/02/2024', '22:41', 52.49, 17.14, 14.78)
(6, '23/02/2024', '22:42', 49.573, 11.86, 14.6)
(7, '23/02/2024', '22:42', 50.673, 18.52, 3.89)
(8, '23/02/2024', '22:43', 50.664, 24.02, 23.89)
(9, '23/02/2024', '22:43', 54.277, 23.33, 26.56)
```

data		CREATE TABLE data (id INTEGER PRIMARY KEY, date TEXT, time TEXT, ping REAL, download REAL, upload REAL)
id	INTEGER	"id" INTEGER
date	TEXT	"date" TEXT
time	TEXT	"time" TEXT
ping	REAL	"ping" REAL
download	REAL	"download" REAL
upload	REAL	"upload" REAL

Chapter 2: Fundamentals of Data Analysis

- 2.2.2.5 - Basic Data Analysis
- 2.2.4.5 - San Francisco Crime
- 2.3.2.7 - Preparing Data
- 2.4.1.5 - Big Data Ethics
- 2.5.1.4 - Internet Meter Data Analysis
- 2.5.2.4 - Working with Python and SQLite
- **2.5.2.5 - Internet Meter SQL** ✅

Objectives:

Part 1: Set up your Database
Part 2: Connect to the Database
Part 3: Manipulate the Data with Pandas
Part 4: Join Tables with Pandas

2.5.2.5 - Internet Meter SQL

2.5.2.5 - Lab

```python
import sqlite3
import pandas as pd
import random
import string

# Connect to the existing database
conn = sqlite3.connect('internet_speed_data.db')

# Read the existing table into a DataFrame
df_existing = pd.read_sql_query("SELECT * FROM data", conn)

# Create a new table
conn.execute('''CREATE TABLE IF NOT EXISTS users
               (id INTEGER PRIMARY KEY, name TEXT, country TEXT, speed_id INTEGER)''')

# Generate random names and countries
names = [''.join(random.choices(string.ascii_uppercase, k=5)) for _ in range(len(df_existing))]
countries = [''.join(random.choices(string.ascii_uppercase, k=5)) for _ in range(len(df_existing))]

# Insert data into the new table
for i in range(len(df_existing)):
    conn.execute(f"INSERT INTO users (name, country, speed_id) VALUES ('{names[i]}', '{countries[i]}', {i + 1})")

# Commit changes to the database
conn.commit()
```

Provides functions to generate pseudo-random numbers, text, or strings

2.5.2.5 - Lab

```python
# Read the new table into a DataFrame
df_new = pd.read_sql_query("SELECT * FROM users", conn)

# Join the tables
df_merged = pd.merge(df_existing, df_new,
left_on='id', right_on='speed_id', how='inner')

# Display the merged DataFrame
print(df_merged.head())

# Close the connection
conn.close()
```

2.5.2.5 - Internet Meter SQL

users		CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, country TEXT, speed_id INTEGER)
id	INTEGER	"id" INTEGER
name	TEXT	"name" TEXT
country	TEXT	"country" TEXT
speed_id	INTEGER	"speed_id" INTEGER

```
C:\Users\reach\OneDrive\Desktop\IoT\CH 2\3>py 2.5.2.5.py
   id_x        date   time    ping  download  upload  id_y   name country  speed_id
0     1  21/02/2024  15:59  31.699     12.27    4.76     1  Alice     USA         1
1     1  21/02/2024  15:59  31.699     12.27    4.76     3  YKUUF   PAUQZ         1
2     1  21/02/2024  15:59  31.699     12.27    4.76     5  XHELZ   KTGIZ         1
3     2  21/02/2024  15:59  47.920     15.62    4.65     2    Bob  Canada         2
4     2  21/02/2024  15:59  47.920     15.62    4.65     4  SVTCU   VOPJI         2
```

Table: users

	id	name	country	speed_id
	Filter	Filter	Filter	Filter
1	1	Alice	USA	1
2	2	Bob	Canada	2
3	3	YKUUF	PAUQZ	1
4	4	SVTCU	VOPJI	2
5	5	XHELZ	KTGIZ	1
6	6	RRKGH	VQDAO	2
7	7	AFFYK	ZYZTR	3
8	8	SBYLB	EIQOT	4
9	9	KVPEM	ITNXX	5
10	10	CNPUD	XSLOP	6
11	11	ESXPG	LUWBN	7
12	12	YJBAZ	TJSRX	8
13	13	VQBJB	DGMNG	9

Chapter 3: Data Analysis

- **3.1.4.6 Descriptive Statistics in Python** ✅
- 3.1.5.5 Correlation Analysis in Python
- 3.2.1.6 Internet Meter Visualization

Objectives:

Part 1: Analyzing the Data

Part 2: Visualizing the Data

3.1.4.6 Descriptive Statistics in Python

3.1.4.6 - Lab

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the CSV file into a DataFrame
df = pd.read_csv('rpi_describe.csv')
# Use the describe method of the new dataframe to view the table of statistics
print(df.describe())
# Add a new column to the dataframe and populate it with rounded weights
df['rounded'] = df['weight'].round()

# Calculate descriptive statistics for the 'rounded' column
count = df['rounded'].count()
mean = df['rounded'].mean()
median = df['rounded'].median()
std = df['rounded'].std()
rng = df['rounded'].max() - df['rounded'].min()

# Create strings reporting the descriptive statistics
countstring = "The count of the dataset is {}.".format(count)
meanstring = "The mean of the dataset is {:.2f}.".format(mean)
stdstring = "The standard deviation of the dataset is {:.2f}.".format(std)
rangestring = "The range of the dataset is {:.2f} (min: {:.2f}, max: {:.2f}).".format(rng, df['rounded'].min(), df['rounded'].max())
```

Console:

```
          weight
count  10000.000000
mean      20.499212
std        0.199874
min       19.752000
25%       20.365000
50%       20.500000
75%       20.635000
max       21.171000
```

	A
1	weight
2	20.44
3	20.244
4	20.549
5	20.755
6	20.74
7	20.847
8	20.063
9	20.453
10	20.719

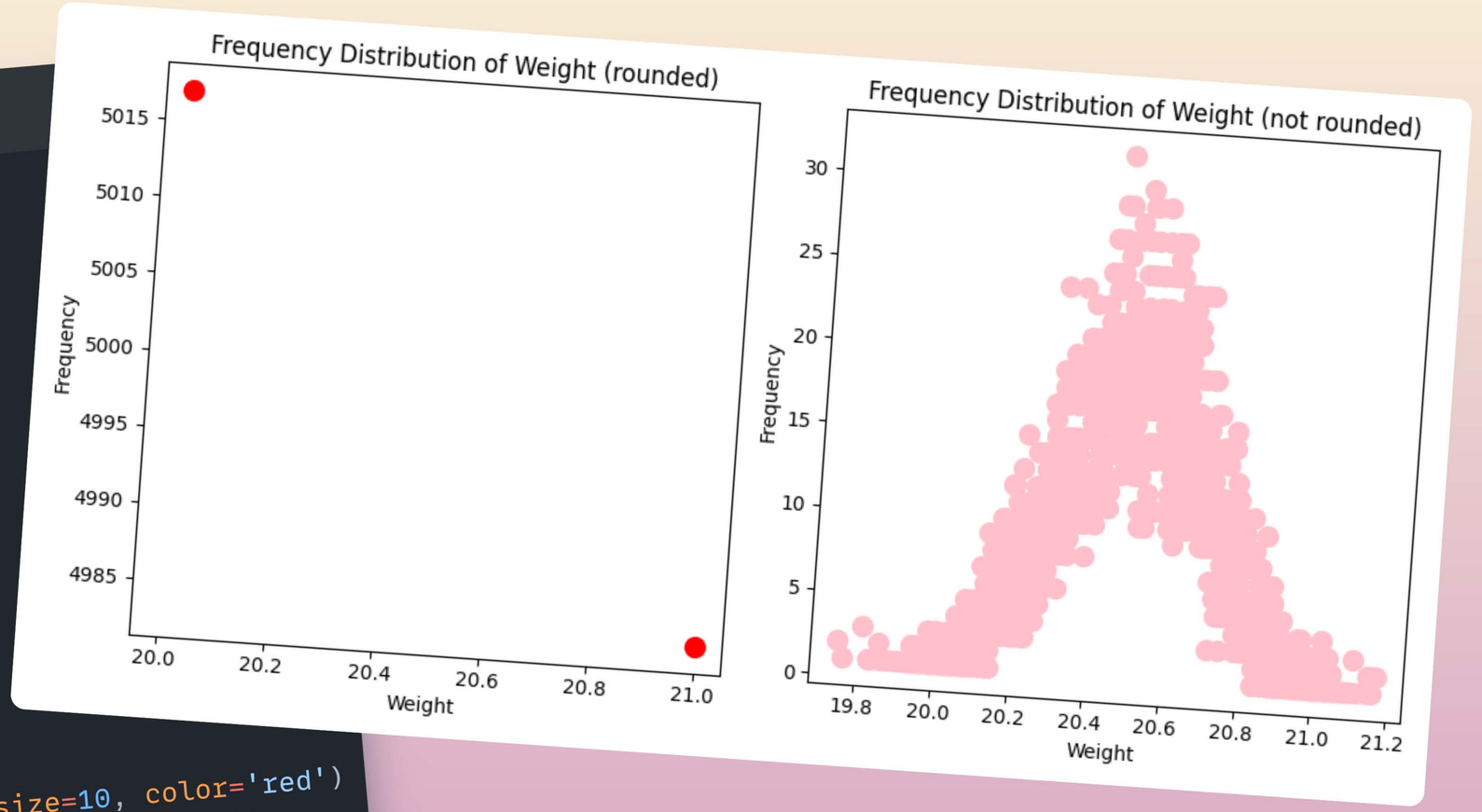
Console:

```
The count of the dataset is 10000.
The mean of the dataset is 20.50.
The standard deviation of the dataset is 0.50.
The range of the dataset is 1.00 (min: 20.00, max: 21.00).
```

3.1.4.6 Descriptive Statistics in Python

3.1.4.6 - Lab

```python
# Create a variable called 'freq' to
#hold the weight values and their frequencies
freq = df['rounded'].value_counts()

# Convert the freq object to a data
#frame. Use to_frame().
freq = freq.to_frame().reset_index()

# Rename the columns to 'value' and 'frequency'
freq.columns = ['value', 'freq']

# Create a new figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Plot the first graph on the left subplot
axs[0].plot(freq.value, freq.freq, "o", markersize=10, color='red')
axs[0].set_title('Frequency Distribution of Weight (rounded)')
axs[0].set_xlabel('Weight')
axs[0].set_ylabel('Frequency')
```



Chapter 3: Data Analysis

- 3.1.4.6 Descriptive Statistics in Python
- **3.1.5.5 Correlation Analysis in Python**
- 3.2.1.6 Internet Meter Visualization

Objectives:

Part 1: The Dataset
Part 2: Scatterplot Graphs and Correlatable Variables
Part 3: Calculating Correlation with Python
Part 4: Visualizing

3.1.5.5 Correlation Analysis in Python

```
Gender,FSIQ,VIQ,PIQ,Weight,Height,MRI_Count
Female,133,132,124,118,64.5,816932
Male,140,150,124,NaN,72.5,1001121
Male,139,123,150,143,73.3,1038437
Male,133,129,128,172.0,68.8,965353
Female,137,132,134,147.0,65.0,951545
```

Visualization library based on matplotlib that provides a high-level interface for creating informative and attractive statistical graphics

3.1.5.5 - Lab

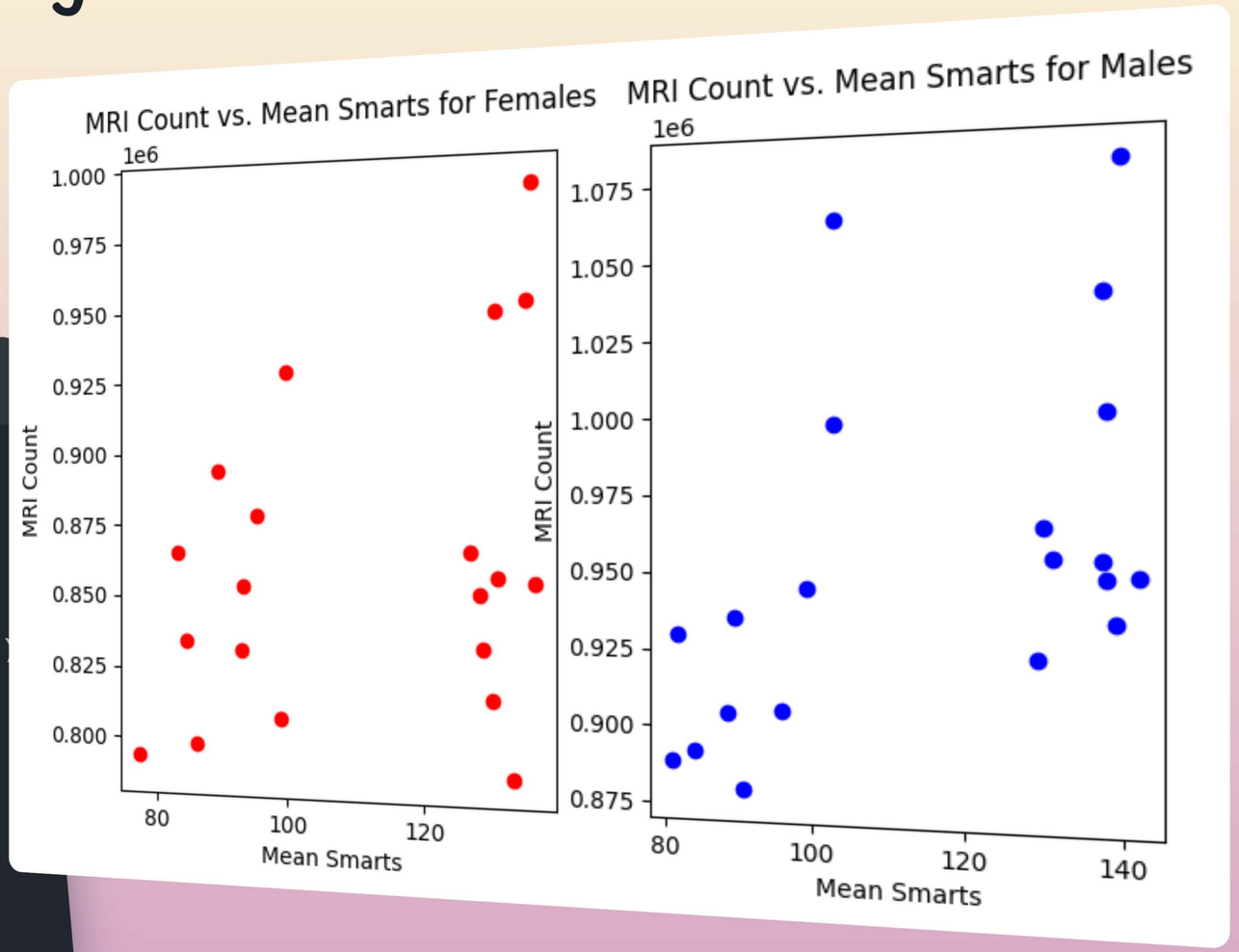
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

brainFile = './brainsize.txt'
brainFrame = pd.read_csv(brainFile)

print(brainFrame.head())
print(brainFrame.describe())

# Filter data for males and females
menDf = brainFrame[(brainFrame.Gender == 'Male')]
womenDf = brainFrame[(brainFrame.Gender == 'Female')]

# Calculate mean smarts for males and females
menMeanSmarts = menDf[["PIQ", "FSIQ", "VIQ"]].mean(axis=1)
womenMeanSmarts = womenDf[['PIQ', 'FSIQ', 'VIQ']].mean(axis=1)
```

```
C:\Users\reach\OneDrive\Desktop\IoT\CH 3\2>py 3.1.5.5.py
   Gender  FSIQ  VIQ  PIQ  Weight  Height  MRI_Count
0  Female   133  132  124   118.0    64.5     816932
1    Male   140  150  124     NaN    72.5    1001121
2    Male   139  123  150   143.0    73.3    1038437
3    Male   133  129  128   172.0    68.8     965353
4  Female   137  132  134   147.0    65.0     951545
             FSIQ         VIQ        PIQ      Weight     Height     MRI_Count
count   40.000000   40.000000  40.00000   38.000000  39.000000  4.000000e+01
mean   113.450000  112.350000 111.02500  151.052632  68.525641  9.087550e+05
std     24.082071   23.616107  22.47105   23.478509   3.994649  7.228205e+04
min     77.000000   71.000000  72.00000  106.000000  62.000000  7.906190e+05
25%     89.750000   90.000000  88.25000  135.250000  66.000000  8.559185e+05
50%    116.500000  113.000000 115.00000  146.500000  68.000000  9.053990e+05
75%    135.500000  129.750000 128.00000  172.000000  70.500000  9.500780e+05
max    144.000000  150.000000 150.00000  192.000000  77.000000  1.079549e+06
```

3.1.5.5 Correlation Analysis in Python

3.1.5.5 - Lab

```python
# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(8, 5))

# Plot the scatter plot for females on the left subplot
axs[0].scatter(womenMeanSmarts, womenDf['MRI_Count'], color="red")
axs[0].set_title('MRI Count vs. Mean Smarts for Females')
axs[0].set_xlabel('Mean Smarts')
axs[0].set_ylabel('MRI Count')

# Plot the scatter plot for males on the right subplot
axs[1].scatter(menMeanSmarts, menDf["MRI_Count"], color="blue")
axs[1].set_title('MRI Count vs. Mean Smarts for Males')
axs[1].set_xlabel('Mean Smarts')
axs[1].set_ylabel('MRI Count')
```



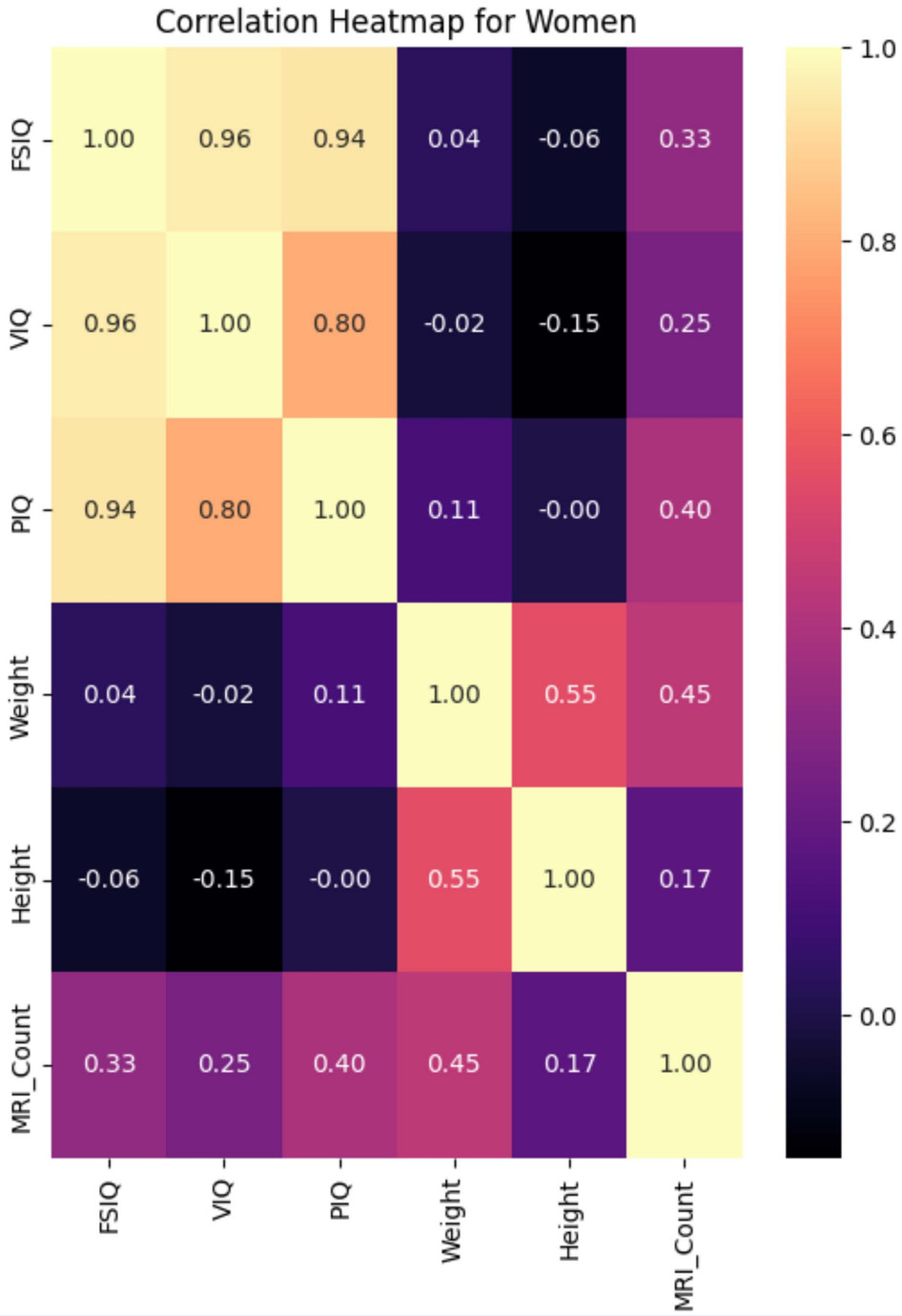
3.1.5.5 Correlation Analysis in Python

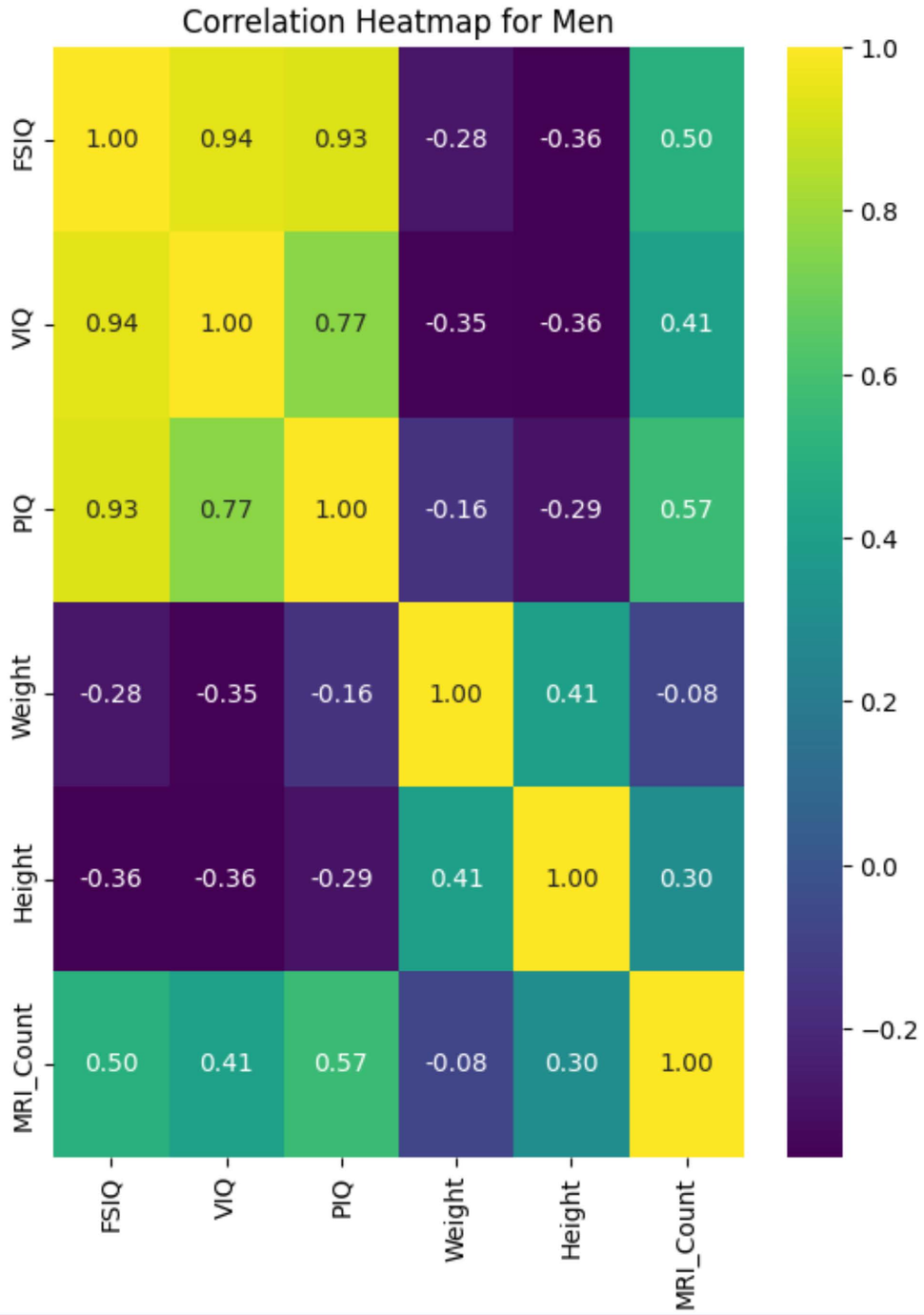
3.1.5.5 - Lab

```python
# Drop the 'Gender' column from the menDf dataframe to exclude non-numeric data
menDf_numeric = menDf.drop(columns=['Gender'])
# Calculate the correlation matrix for the remaining numeric columns using Pearson method
menCorr = menDf_numeric.corr(method='pearson')

# Drop the 'Gender' column from the womenDf dataframe to exclude non-numeric data
womenDf_numeric = womenDf.drop(columns=['Gender'])
# Calculate the correlation matrix for the remaining numeric columns using Pearson method
womenCorr = womenDf_numeric.corr(method='pearson')

# Set up the subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 8))

# Plot the heatmap for women on the left
sns.heatmap(womenCorr, ax=axs[0], annot=True, cmap='magma', fmt=".2f")
axs[0].set_title('Correlation Heatmap for Women')

# Plot the heatmap for men on the right
sns.heatmap(menCorr, ax=axs[1], annot=True, cmap='viridis', fmt=".2f")
axs[1].set_title('Correlation Heatmap for Men')
```

A ***positive correlation*** means that as one variable increases, the other variable also tends to increase

A ***negative correlation*** means that as one variable increases, the other variable tends to decrease


Correlation Heatmap for Women
FSIQ
VIQ
PIQ
Weight
Height
MRI_Count
1.00 0.96 0.94 0.04 -0.06 0.33
0.96 1.00 0.80 -0.02 -0.15 0.25
0.94 0.80 1.00 0.11 -0.00 0.40
0.04 -0.02 0.11 1.00 0.55 0.45
-0.06 -0.15 -0.00 0.55 1.00 0.17
0.33 0.25 0.40 0.45 0.17 1.00
1.0
0.8
0.6
0.4
0.2
0.0


Correlation Heatmap for Men
FSIQ
VIQ
PIQ
Weight
Height
MRI_Count
1.00 0.94 0.93 -0.28 -0.36 0.50
0.94 1.00 0.77 -0.35 -0.36 0.41
0.93 0.77 1.00 -0.16 -0.29 0.57
-0.28 -0.35 -0.16 1.00 0.41 -0.08
-0.36 -0.36 -0.29 0.41 1.00 0.30
0.50 0.41 0.57 -0.08 0.30 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Chapter 3: Data Analysis

- 3.1.4.6 Descriptive Statistics in Python
- 3.1.5.5 Correlation Analysis in Python
- **3.2.1.6 Internet Meter Visualization**

Sales Data

Objectives:

Part 1: Clean and Analyze Data
Part 2: Visualize Data

3.2.1.6 Sales Data Visualization

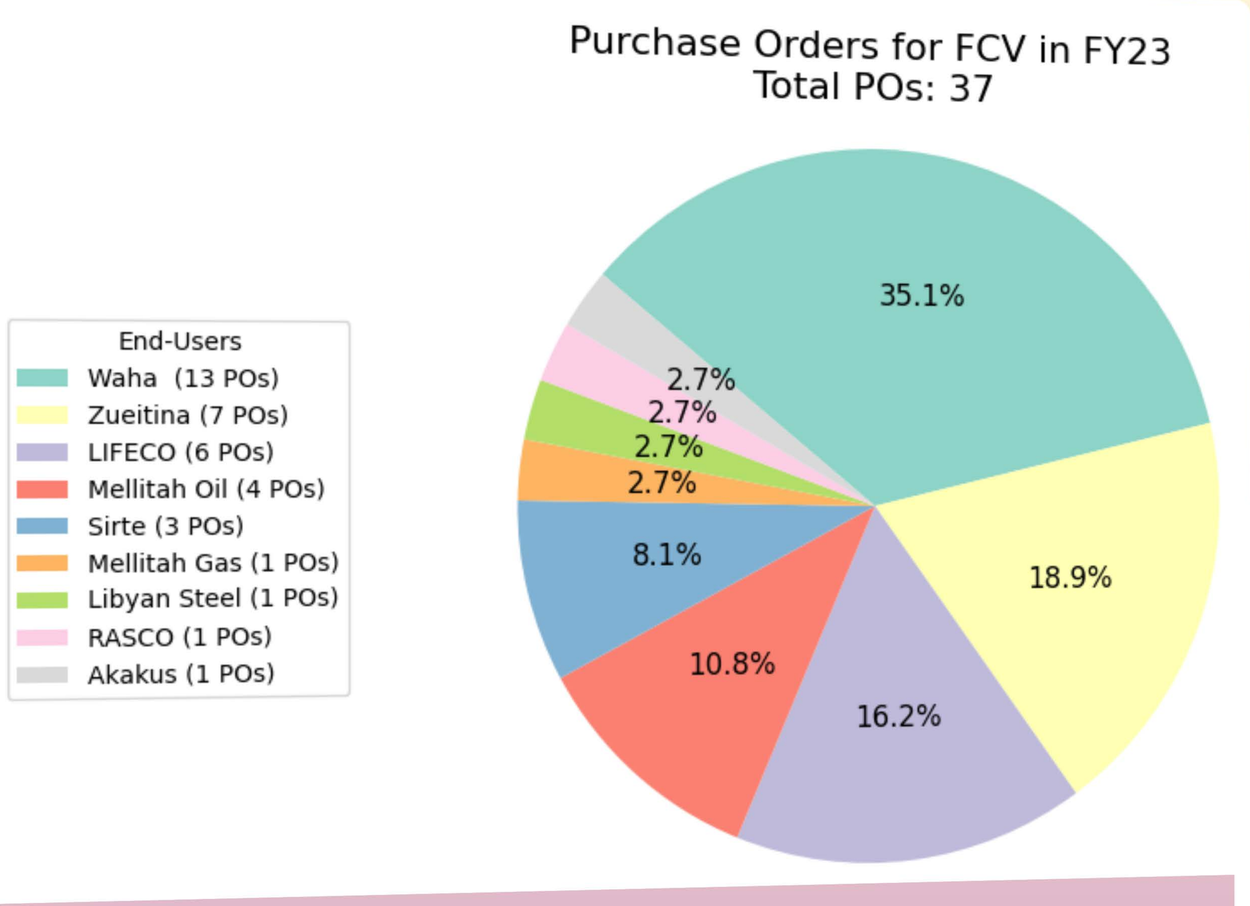
#	Customer	End-User	RFQ #	PO #	PO Release	PO Booking	SO#	BU/Vendor	Vendor Pric	BG Price	Payment Te	Delivery Tin	Expected D	PO handled	Vendor Deli	Client Deliv	Actual/Sch
1	Black Gold	Mellitah Oil	26-203928	PO - 02004	25-Oct-22	P 01	70135822	FCV	6,790	$9,360.00	Net 30 Days	19	7-Mar-23	Ali/Fooz	CPT - Tripol	DDP - Tripo	27-Mar-23
3	Black Gold	Waha	9963	131573 - F(	22-Nov-22	P 02	70136960	FCV	70,778	$188,126.50	CAD	26	23-May-23	Fooz	CPT - Tripol	DDP - Tripo	31-May-23
10	Black Gold	Zueitina	ZOC-21125	ZFP-22/10(	16-Nov-22	P 02	70136772	FCV	35,460	$41,558.00	CAD	26	17-May-23	Wael/Fooz	CPT - Tripol	CPT - Tripol	18-May-23
14	Black Gold	Zueitina	ZOC-22113	ZFP-22/104	17-Nov-22	P 02	70136967	FCV	25,286	$34,056.00	Net 30 Days	30	15-Jun-23	Fooz	CPT - Tripol	DDP - Tripo	14-Jun-23
16	Black Gold	Mellitah Ga	MELL-2585	LPO-25855	27-Dec-22	P 03	70138472	FCV	4,231	$9,395.00	Net 30 Days	10	7-Mar-23	Enaam/Foo	CPT - Tripol	DDP - Tripo	24-May-23
18	Black Gold	Zueitina	ZOC-2114(	ZFP-22/104	29-Dec-22	P 03	70138604	FCV	7,596	$11,224.00	Net 30 Days	32	10-Aug-23	Fooz	CPT - Tripol	DDP - T	
19	Black Gold	Waha	8026	134598 - F(	26-Dec-23	P 03	70138501	FCV	167,276	$341,803.00	LC	22	28-May-24	Fooz			
22	Black Gold	Mellitah Oil	10-013940	PO - 02093	13-Feb-23	P 06	70142372	FCV	15,844	$23,450.00	Net 45 Days						
23	Black Gold	Libyan Stee	ST/213082/	PO 230086	19-Feb-23	P 06	70142391	FCV	3,858	$6,142.00							
24	Black Gold	LIFECO	LIF-009030	LF-004869	27-Jan-23	P 04	70139758	FCV									
25	Black Gold	LIFECO	LIF-009070	LF-004856	21-Mar-23	P 06											

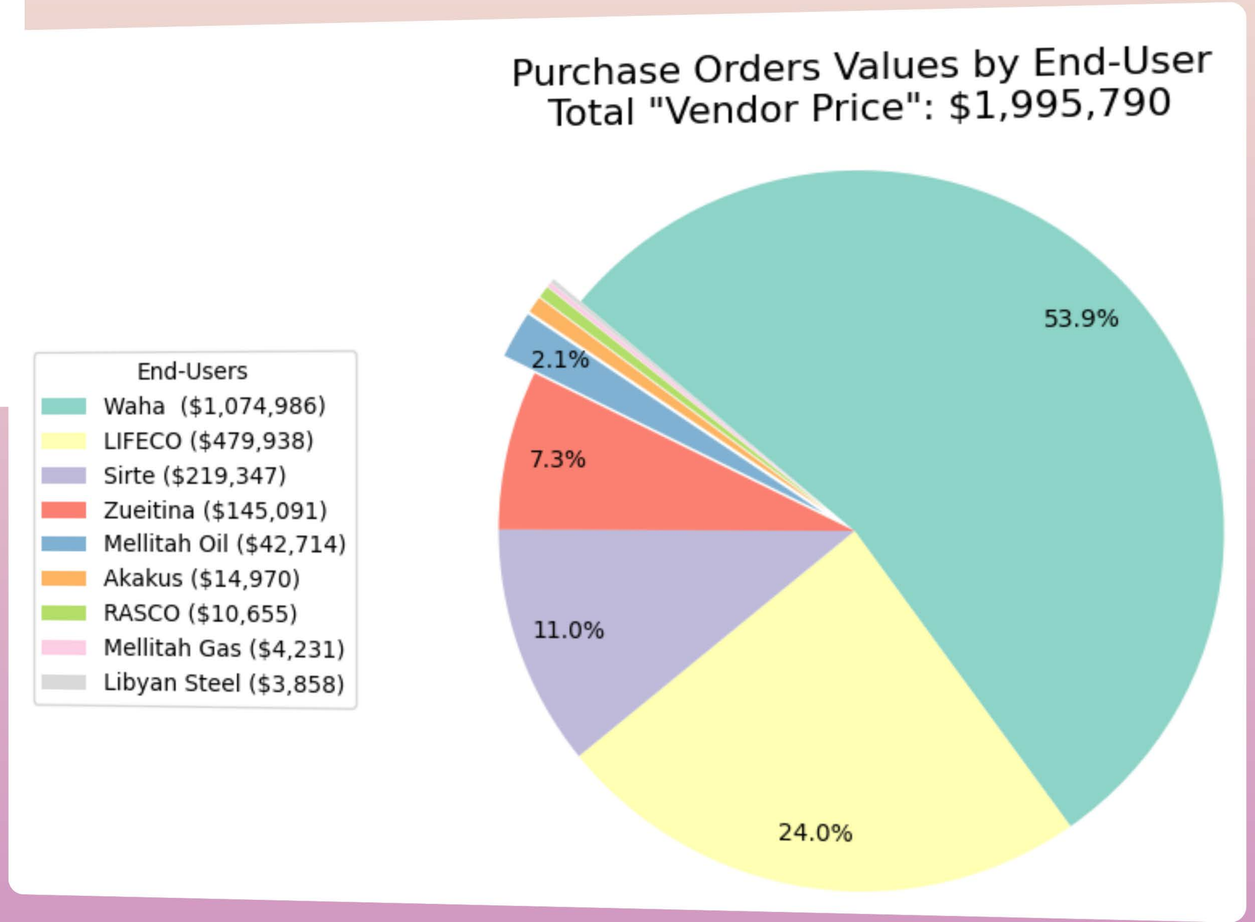
3.2.1.6 - Lab

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load data from the CSV file
df = pd.read_csv('purchase_orders.csv')

# Group the data by End-User and calculate
po_counts = df['End-User'].value_counts()
# Calculate the total number of POs
total_pos = po_counts.sum()

# Create a pie chart
fig, ax = plt.subplots(figsize=(10, 6))
# Generate a color map for the slices
colors = plt.cm.Set3(range(len(po_counts)))

# Add the total number of POs to the title
title = f'Purchase Orders for FCV in FY23\n
ax.set_title(title, fontsize=16)
```

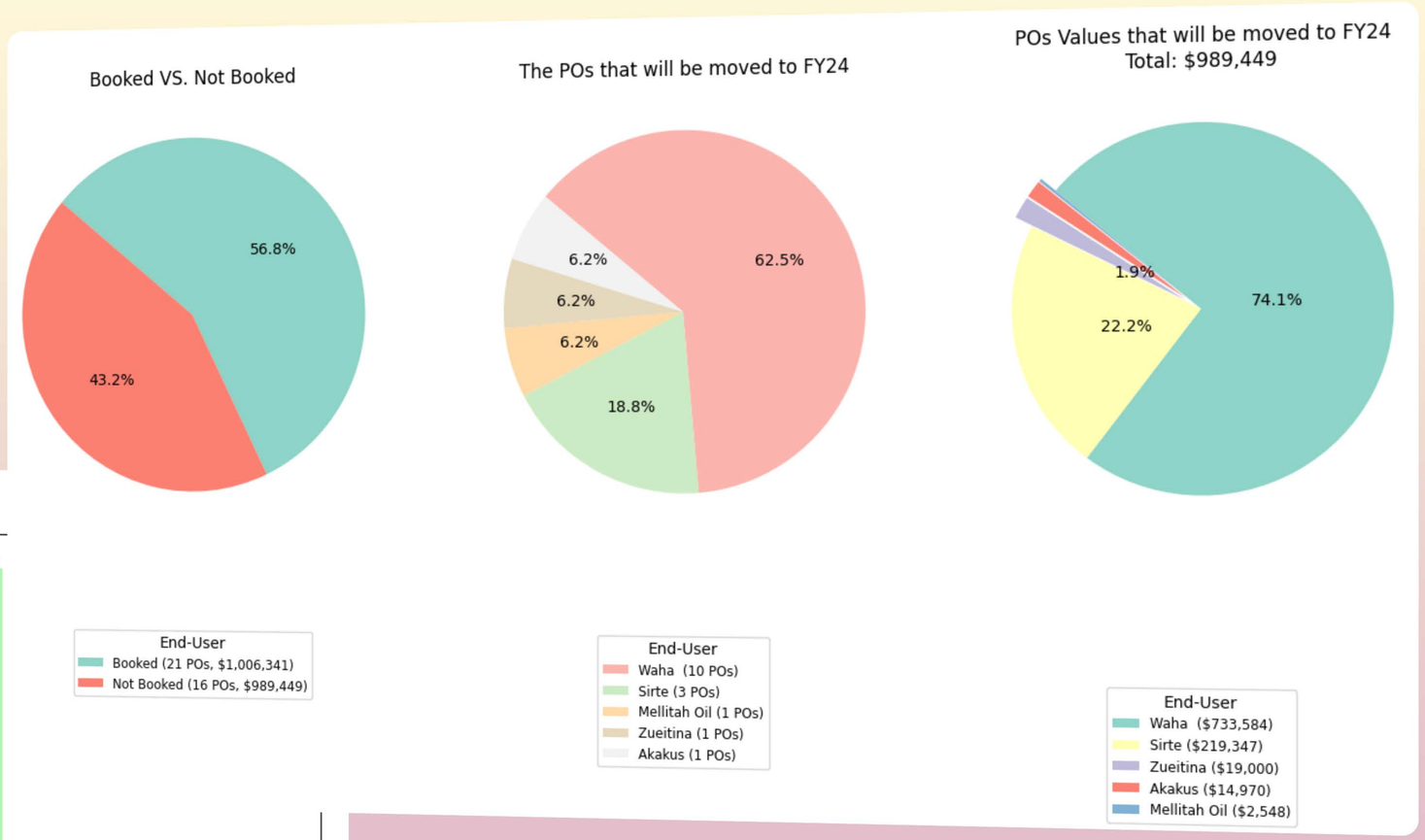
3.2.1.6 - Lab

```python
# Remove commas from "Vendor Price" column and convert it to integer
df['Vendor Price'] = df['Vendor Price'].str.replace(',', '', regex=True).astype(int)

# Group the data by End-User and calculate the total "Vendor Price"
vendor_prices = df.groupby('End-User')['Vendor Price'].sum()

# Sort the End-Users by their total "Vendor Price" in descending order
vendor_prices = vendor_prices.sort_values(ascending=False)

# Calculate the total "Vendor Price" for all End-Users
total_vendor_price = vendor_prices.sum()

# Calculate the percentage of "Vendor Price" for each End-User
percentages = (vendor_prices / total_vendor_price) * 100
```


Purchase Orders for FCV in FY23
Total POs: 37
End-Users
Waha (13 POs)
Zueitina (7 POs)
LIFECO (6 POs)
Mellitah Oil (4 POs)
Sirte (3 POs)
Mellitah Gas (1 POs)
Libyan Steel (1 POs)
RASCO (1 POs)
Akakus (1 POs)
35.1%
18.9%
16.2%
10.8%
8.1%
2.7%
2.7%
2.7%
2.7%
2.7%


Purchase Orders Values by End-User
Total "Vendor Price": $1,995,790
End-Users
Waha ($1,074,986)
LIFECO ($479,938)
Sirte ($219,347)
Zueitina ($145,091)
Mellitah Oil ($42,714)
Akakus ($14,970)
RASCO ($10,655)
Mellitah Gas ($4,231)
Libyan Steel ($3,858)
53.9%
24.0%
11.0%
7.3%
2.1%

Vendor Price by Period and Quarter
Total for All Quarters: $1,006,341
Vendor Price
Quarter
0
50000
100000
150000
200000
6,790
131,524
179,103
207,067
123,427
17,421
228,260
87,330
25,419
End-User
Booked (21 POs, $1,006,341)
Not Booked (16 POs, $989,449)


Booked VS. Not Booked
56.8%
43.2%
The POs that will be moved to FY24
62.5%
18.8%
6.2%
6.2%
6.2%
End-User
Waha (10 POs)
Sirte (3 POs)
Mellitah Oil (1 POs)
Zueitina (1 POs)
Akakus (1 POs)
POs Values that will be moved to FY24
Total: $989,449
74.1%
22.2%
1.9%
End-User
Waha ($733,584)
Sirte ($219,347)
Zueitina ($19,000)
Akakus ($14,970)
Mellitah Oil ($2,548)

Chapter 4: Advanced Analytics and machine learning

- **4.1.2.4 Simple Linear Regression in Python** ✅
- 4.1.3.5 Decision Tree Classification
- 4.2.2.5 Evaluating Fit Errors in Linear Regression
- 4.3.1.4 Internet Meter Linear Regression
- 4.3.2.4 Internet Meter Anomaly Detection

Objectives:

Part 1: Import the Libraries and Data
Part 2: Plot the Data
Part 3: Perform Simple Linear Regression

4.1.2.4 Simple Linear Regression in Python

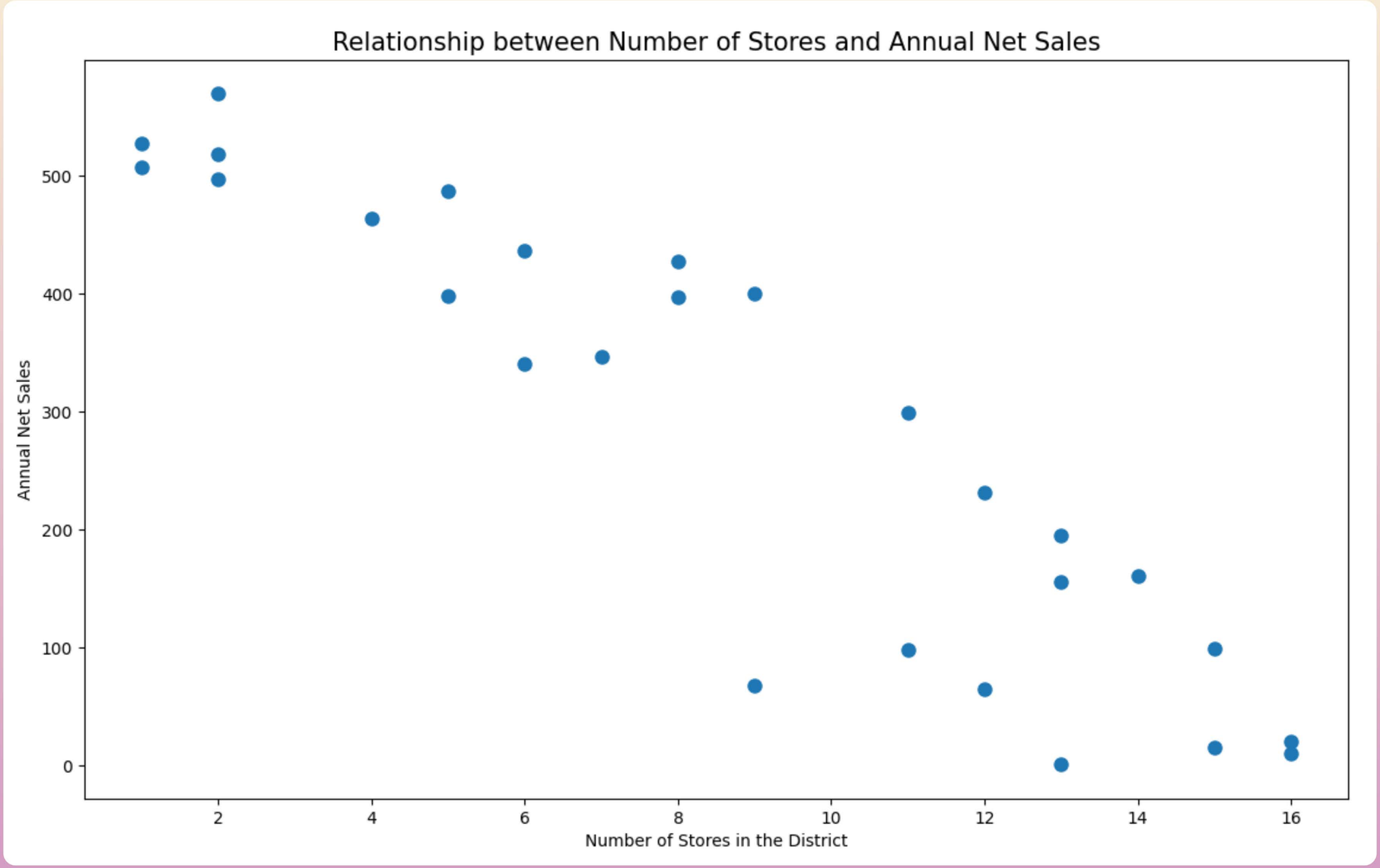
	A	B	C	D
1	district	annual net	number of stores in district	
2	1	231	12	
3	2	156	13	
4	3	10	16	
5	4	519	2	
6	5	437	6	
7	6	487	5	

3.2.1.6 - Lab

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# The district column has no relevance at this time, so it can be dropped.
salesDist = salesDist.rename(columns={'annual net sales':'sales','number of stores in district':'stores'})

# Size of the plot
plt.figure(figsize=(10, 8))

# Create a scatter plot: Number of stores in the District vs. Annual Net Sales
plt.plot(x, y, 'o', markersize = 8)

# Add axis labels and increase the font size
plt.ylabel('Annual Net Sales')
plt.xlabel('Number of Stores in the District')
plt.title('Relationship between Number of Stores and Annual Net Sales', fontsize = 15)
```

```
   district  sales  stores
0         1  231.0      12
1         2  156.0      13
2         3   10.0      16
3         4  519.0       2
4         5  437.0       6
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   district  27 non-null     int64
 1   sales     27 non-null     float64
 2   stores    27 non-null     int64
dtypes: float64(1), int64(2)
```

4.1.2.4 Simple Linear Regression in Python



4.1.2.4 Simple Linear Regression in Python

Fits a polynomial of degree 1 (a straight line) to the data points (x, y) and returns the slope m and y-intercept b of the best-fit line.

3.2.1.6 - Lab

```python
y = sales['sales']
x = sales.stores
# ⇒ Linear Regression
m, b = np.polyfit(x, y, 1)
print ('The slope of line is {:.2f}.'.format(m))
print ('The y-intercept is {:.2f}.'.format(b))
print ('The best fit simple linear regression line is {:.2f}x + {:.2f}.'.format(m, b))
y_mean = y.mean()
x_mean = x.mean()
print ('The centroid for this dataset is x = {:.2f} and y = {:.2f}.'.format(x_mean, y_mean))

# ⇒ Prediction
def predict(query):
    if query ⩾ 1:
        predict = m * query + b
        return predict
    else:
        print ("You must have at least 1 store in the district to predict the annual net sales.")
print('predict(3): ', predict(3))
print('predict(15): ', predict(15))
```

```
The slope of line is -35.79.
The y-intercept is 599.38.
The best fit simple linear regression line is -35.79x + 599.38.
The centroid for this dataset is x = 8.74 and y = 286.57.
predict(3):  492.0184535837394
predict(7):  348.8701117318434
predict(12):  169.93468441697337
predict(15):  62.57342802805135
```

Polynomial regression can approximate a ***nonlinear*** relationship between the independent and dependent variables by transforming the data into a ***linear*** form through the use of polynomial functions

Relationship between Number of Stores and Annual Net Sales
Annual Net Sales
Simple Linear Regression Line
Centroid
(8.74, 286.57)
Annual Net Sales
Number of Stores in District
0
100
200
300
400
500
2
4
6
8
10
12
14
16

Chapter 4: Advanced Analytics and machine learning

- 4.1.2.4 Simple Linear Regression in Python
- **4.1.3.5 Decision Tree Classification** ✅
- 4.2.2.5 Evaluating Fit Errors in Linear Regression
- 4.3.1.4 Internet Meter Linear Regression
- 4.3.2.4 Internet Meter Anomaly Detection

Objectives:

Part 1: Create a Decision Tree Classifier
Part 2: Apply the Decison Tree Model
Part 3: Evaluate the Decison Tree Model

4.1.3.5 Decision Tree Classification

Decision tree classifier that uses a tree-like graph of decisions and their possible consequences to make predictions

PassengerI	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr.	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, M	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen,	female	26	0	0	STON/O2. 3	7.925		S
4	1	1	Futrelle, Mr	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. W	male	35	0	0	373450	8.05		S

4.1.3.5 - Lab

```python
import pandas as pd
from sklearn import tree
from io import StringIO
import subprocess
import numpy as np
from sklearn.model_selection import train_test_split

# Load the training data from a CSV file into a pandas DataFrame
training = pd.read_csv("./titanic_train.csv")

# Replace string data with numeric labels for gender
training["Gender"] = training["Gender"].apply(lambda toLabel: 0 if toLabel == 'male' else 1)

# Handle missing values for the age variable
training["Age"].fillna(training["Age"].mean(), inplace=True)

# Create arrays for the target and input variables
y_target = training["Survived"].values
columns = ["Fare", "Pclass", "Gender", "Age", "SibSp"]
X_input = training[list(columns)].values
```

Sklearn library for splitting data into training and testing sets

1) Loading
2) Preprocessing
3) Training
4) Testing the model

4.1.3.5 Decision Tree Classification

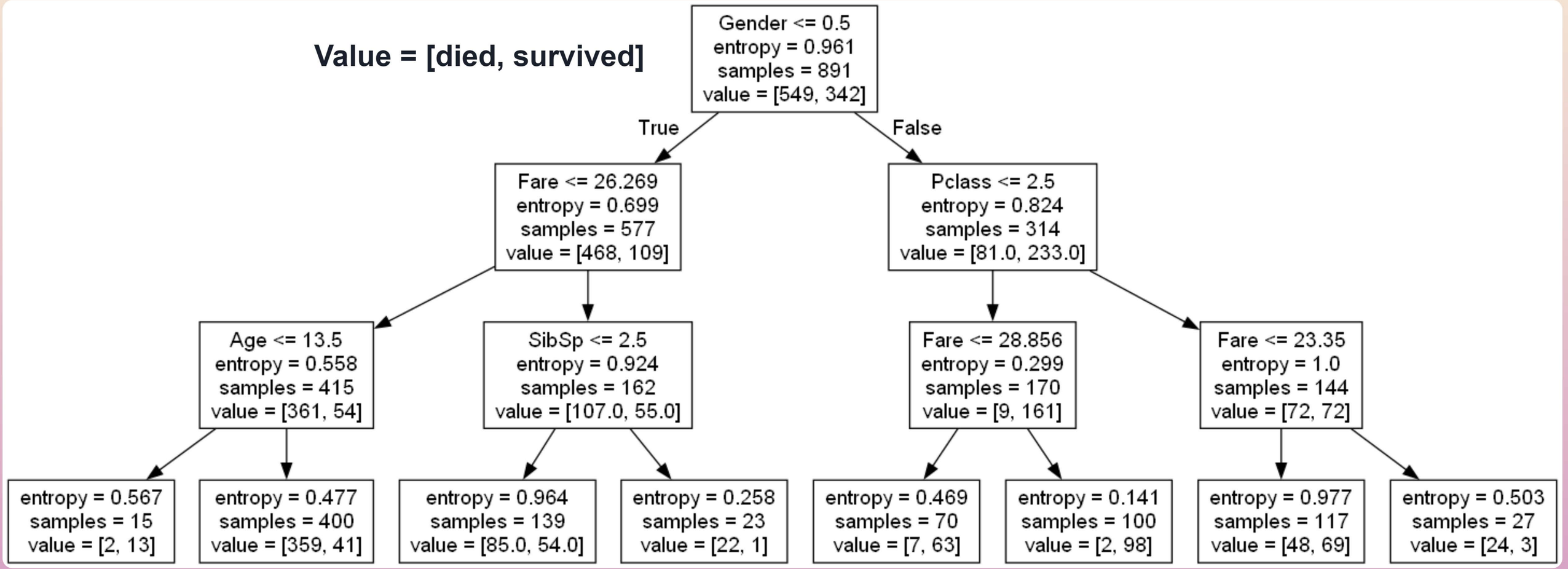
PassengerI	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
892	3	Kelly, Mr. Ja	male	34.5	0	0	330911	7.8292		Q
893	3	Wilkes, Mrs	female	47	1	0	363272	7		S
894	2	Myles, Mr. T	male	62	0	0	240276	9.6875		Q
895	3	Wirz, Mr. Al	male	27	0	0	315154	8.6625		S

4.1.3.5 - Lab

```python
# Train a decision tree classifier using the training data
clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf_train = clf_train.fit(X_input, y_target)

# Evaluate the model's accuracy on the training data
score = clf_train.score(X_input, y_target) * 100

# Load the testing data from a CSV file into a pandas DataFrame
testing = pd.read_csv("./titanic_test.csv")

# Replace gender labels and handle missing age values
testing["Gender"] = testing["Gender"].apply(lambda toLabel: 0 if toLabel == 'male' else 1)
testing["Age"].fillna(testing["Age"].mean(), inplace=True)

# Apply the trained model to the testing data and compute the accuracy
X_input = testing[list(columns)].values
target_labels = clf_train.predict(X_input)
target_labels = pd.DataFrame({'Est_Survival': target_labels, 'Name': testing['Name']})
```

1) Loading
2) Preprocessing
3) Training
4) Testing the model

4.1.3.5 Decision Tree Classification



Chapter 4: Advanced Analytics and machine learning

- 4.1.2.4 Simple Linear Regression in Python
- 4.1.3.5 Decision Tree Classification
- **4.2.2.5 Evaluating Fit Errors in Linear Regression**
- 4.3.1.4 Internet Meter Linear Regression
- 4.3.2.4 Internet Meter Anomaly Detection

Objectives:

In this lab, you will become familiar with the concepts of evaluating fit errors in linear regression
Part 1: Import the Libraries and Data
Part 2: Calculate the Errors

4.2.2.5 Evaluating Fit Errors in Linear Regression

Linear regression is a linear approach to modeling the relationship between a dependent variable and one or more independent variables

4.2.2.5 - Lab

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
# Import the data
salesDist = pd.read_csv('./stores-dist.csv')
salesDist.columns = ['district','sales','stores']
# Prepare the data
sales = salesDist.drop('district', axis=1)
y = sales.sales
x = sales.stores
# Fit a linear regression model
model = LinearRegression()
model.fit(x.values.reshape(-1, 1), y)
# Calculate predictions
predictions = model.predict(x.values.reshape(-1, 1))

# Calculate evaluation metrics
r2 = r2_score(y, predictions)
mse = mean_squared_error(y, predictions)
mae = mean_absolute_error(y, predictions)
```

R-squared (R2): tells you how well the linear regression model fits the data. It's a value between 0 and 1, where 1 means the model fits the data perfectly

Mean Squared Error (MSE): measures the average squared difference between the actual and predicted values. It gives you an idea of how close the predicted values are to the actual values. A lower MSE indicates that the model is better at predicting the data

Mean Absolute Error (MAE): is similar to MSE but uses the absolute differences between actual and predicted values instead of squared differences. It represents the average magnitude of the errors in the predictions. Like MSE, a lower MAE indicates a better model

Console:

r2 = 0.83217523508888
mse = 5961.386465941158
mae = 61.223261178687

Chapter 4: Advanced Analytics and machine learning

- 4.1.2.4 Simple Linear Regression in Python
- 4.1.3.5 Decision Tree Classification
- 4.2.2.5 Evaluating Fit Errors in Linear Regression
- **4.3.1.4 Internet Meter Linear Regression** ✓
- 4.3.2.4 Internet Meter Anomaly Detection

Predicting Forest Fire

Objectives:

Become familiar with the concepts of regression and prediction and the concept of working with data

Part 1: Import the Libraries
Part 2: Visualize the Historical Data
Part 3: Create a Simple Linear Regression Model: First Order Polynomial
Part 4: Create a Linear Regression Model - Higher Order Polynomials
Part 5: Calculate the Errors
Part 6: Create a Non-Linear Regression Model
Part 7: Compare the Model

4.3.1.4 Predicting Forest Fire Linear Regression

Linear Support Vector Classifier (LinearSVC): This is a linear model, meaning it assumes a linear relationship between the input features and the output. It is considered a "first order" model because it deals with linear relationships

4.2.2.5 - Lab

```
# Regular EDA (exploring data analysis) and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# Models from Scikit-Learn
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Model Evaluation
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score, roc_curve, auc
```

K-Nearest Neighbors Classifier (KNeighborsClassifier): This is a non-linear model that makes predictions based on the "k" nearest neighbors of a data point. It is considered a "higher order" model because it can capture non-linear relationships between the input features and the output

Random Forest Classifier (RandomForestClassifier): This is also a non-linear model that uses an ensemble of decision trees to make predictions. Like the K-Nearest Neighbors Classifier, it is considered a "higher order" model because it can capture non-linear relationships

4.3.1.4 Predicting Forest Fire Linear Regression

1. Problem definition
2. Data
3. Features
4. **Data Exploration**
5. **Modelling**

- Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012)
- Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42
- RH : Relative Humidity in %: 21 to 90
- Ws :Wind speed in km/h: 6 to 29
- Rain: total day in mm: 0 to 16.8
- FFMC: Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5
- DMC: Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9
- DC: Drought Code (DC) index from the FWI system: 7 to 220.4
- ISI: Initial Spread Index (ISI) index from the FWI system: 0 to 18.5
- BUI: Buildup Index (BUI) index from the FWI system: 1.1 to 68
- FWI: Fire Weather Index (FWI) Index: 0 to 31.1
- Classes: two classes, namely fire and no fire.

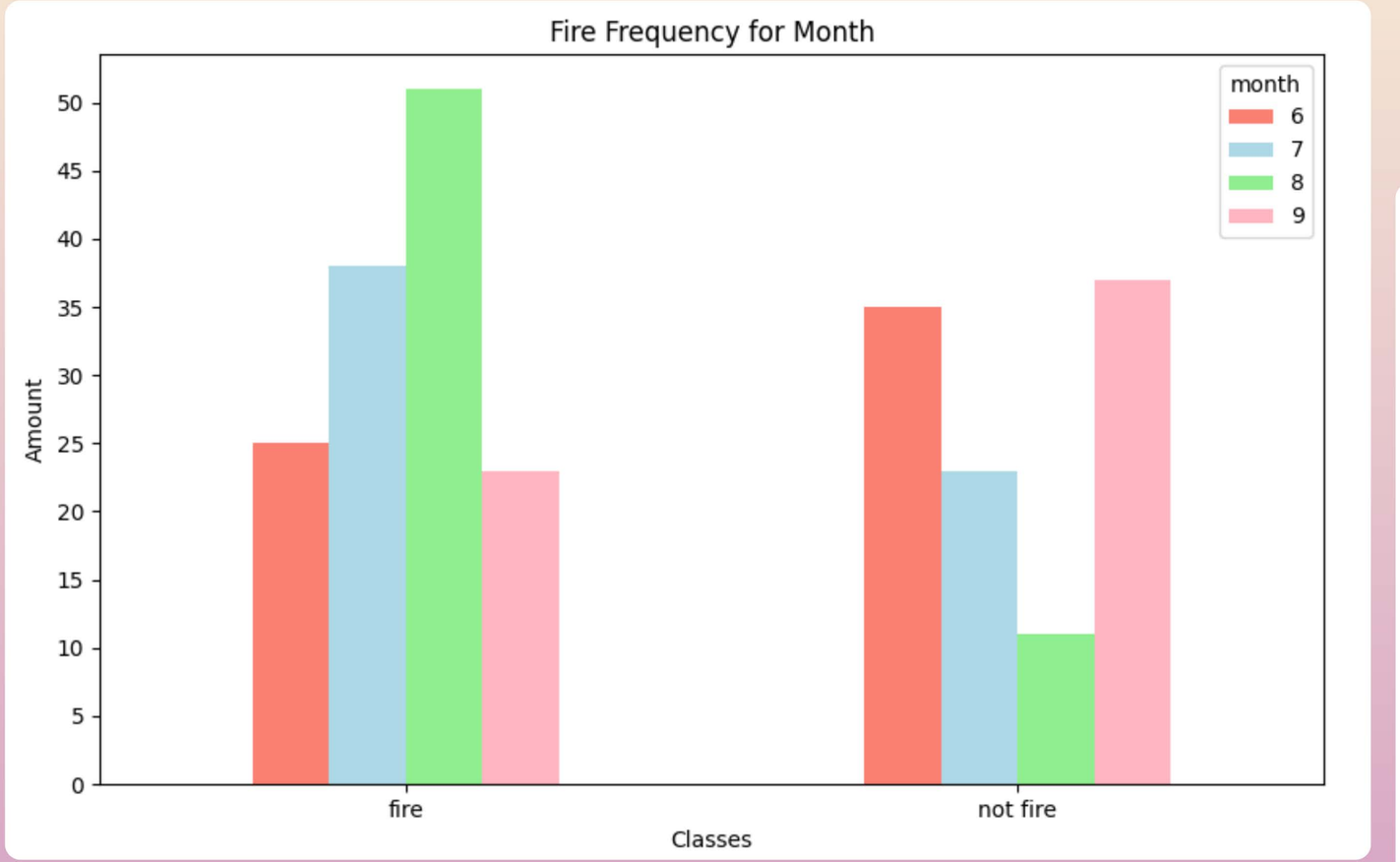
Problem Definition

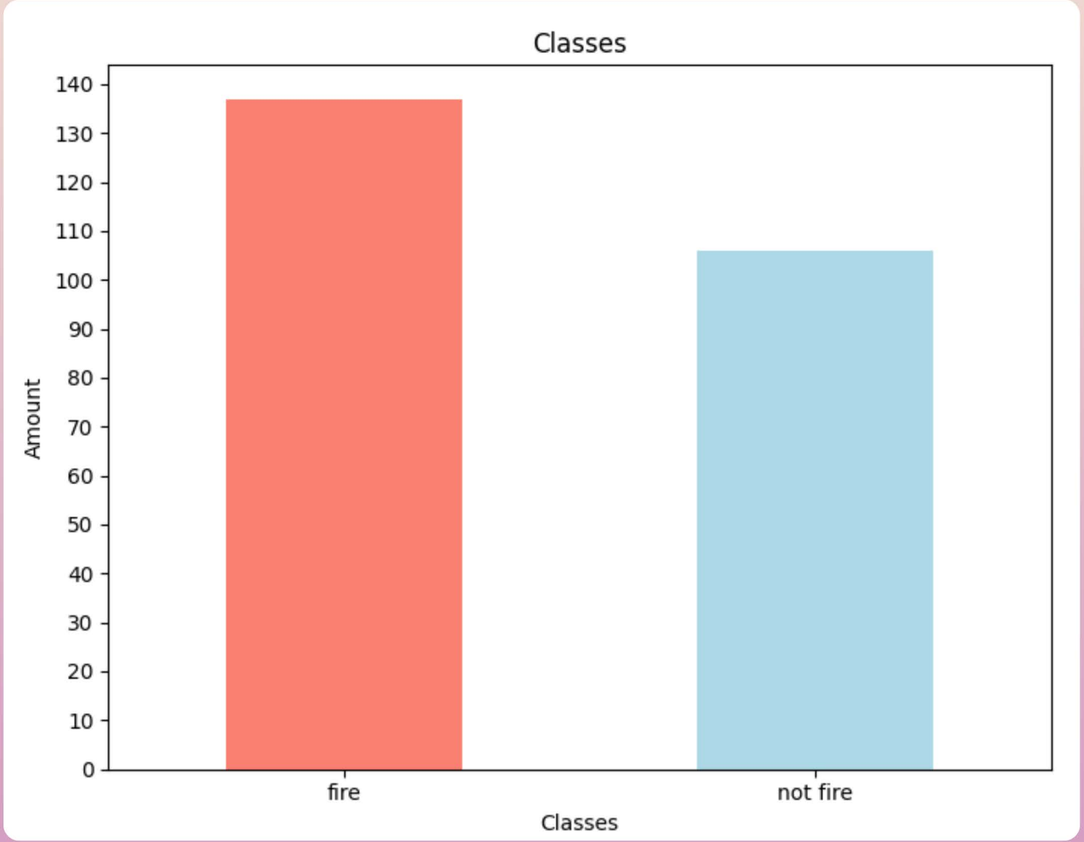
Given parameters about a fire, can we predict whether or not the fire happened?

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	not fire
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire

4.3.1.4 Predicting Forest Fire Linear Regression

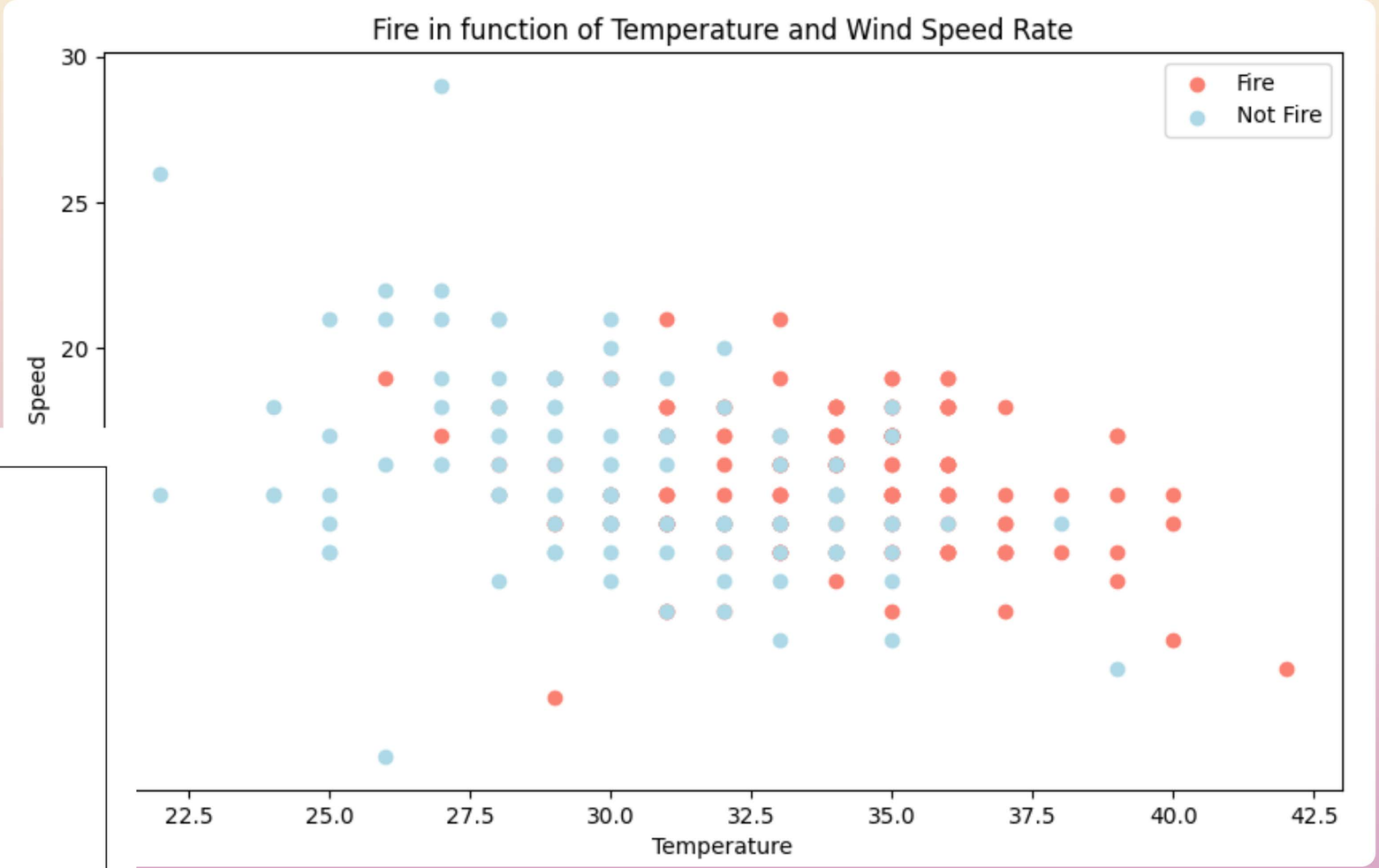
1. Problem definition
2. Data
3. Features
4. Data Exploration
5. Modelling

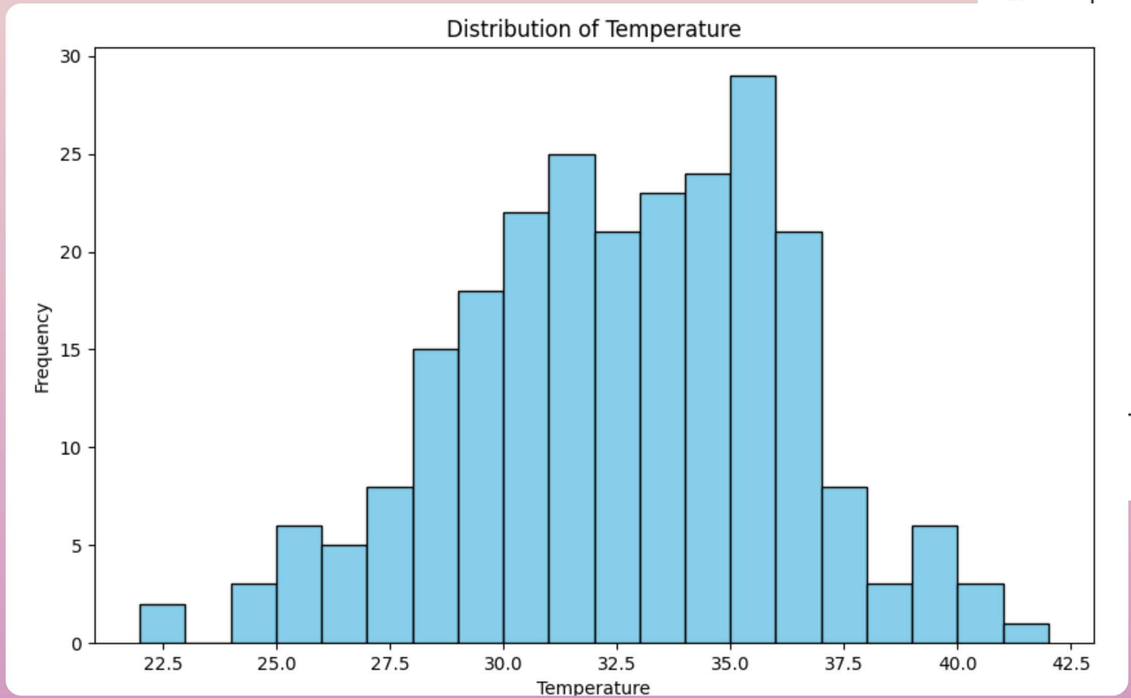




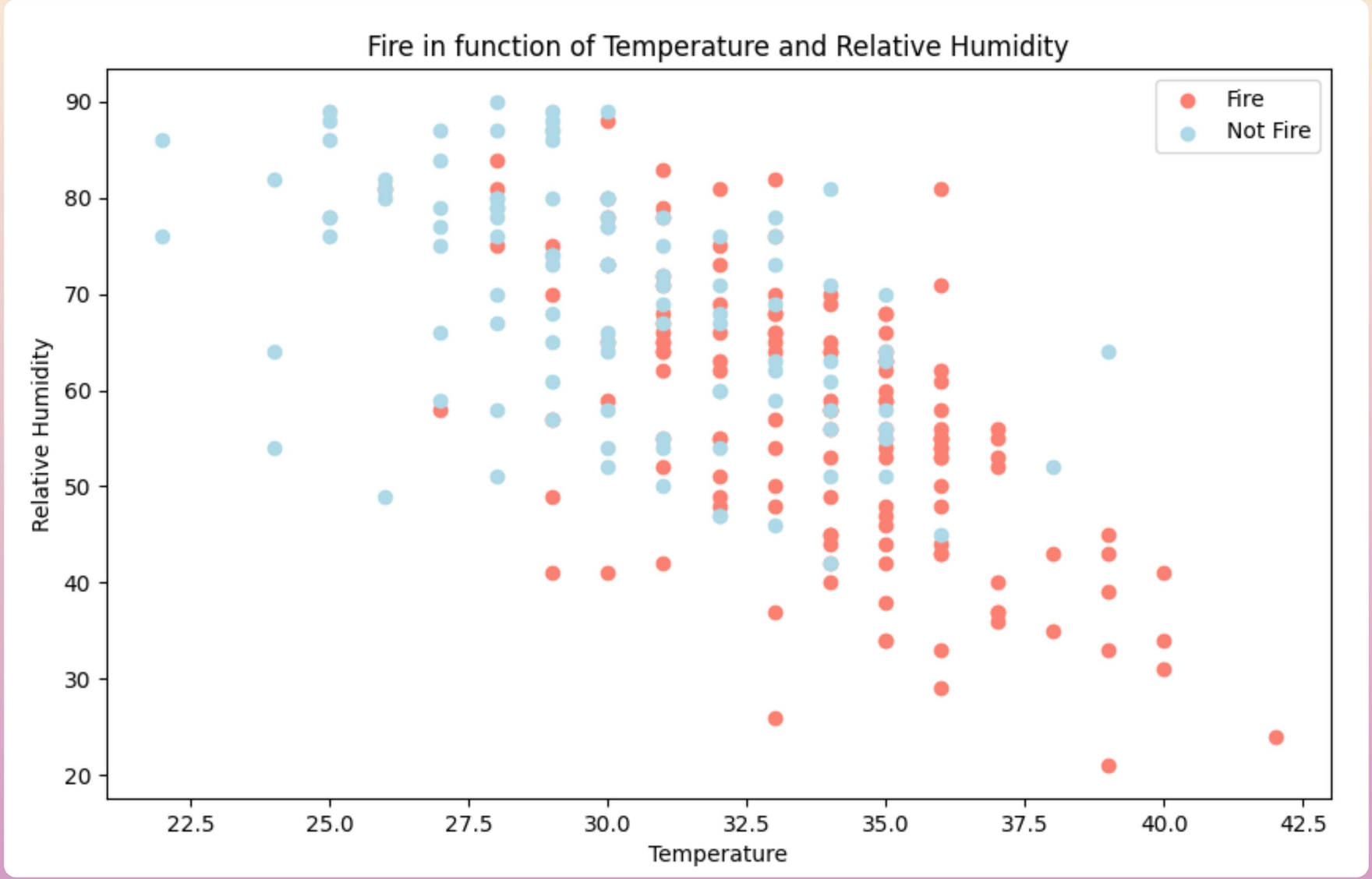
4.3.1.4 Predicting Forest Fire Linear Regression

1. **Problem definition**
2. **Data**
3. **Features**
4. **Data Exploration**
5. **Modelling**



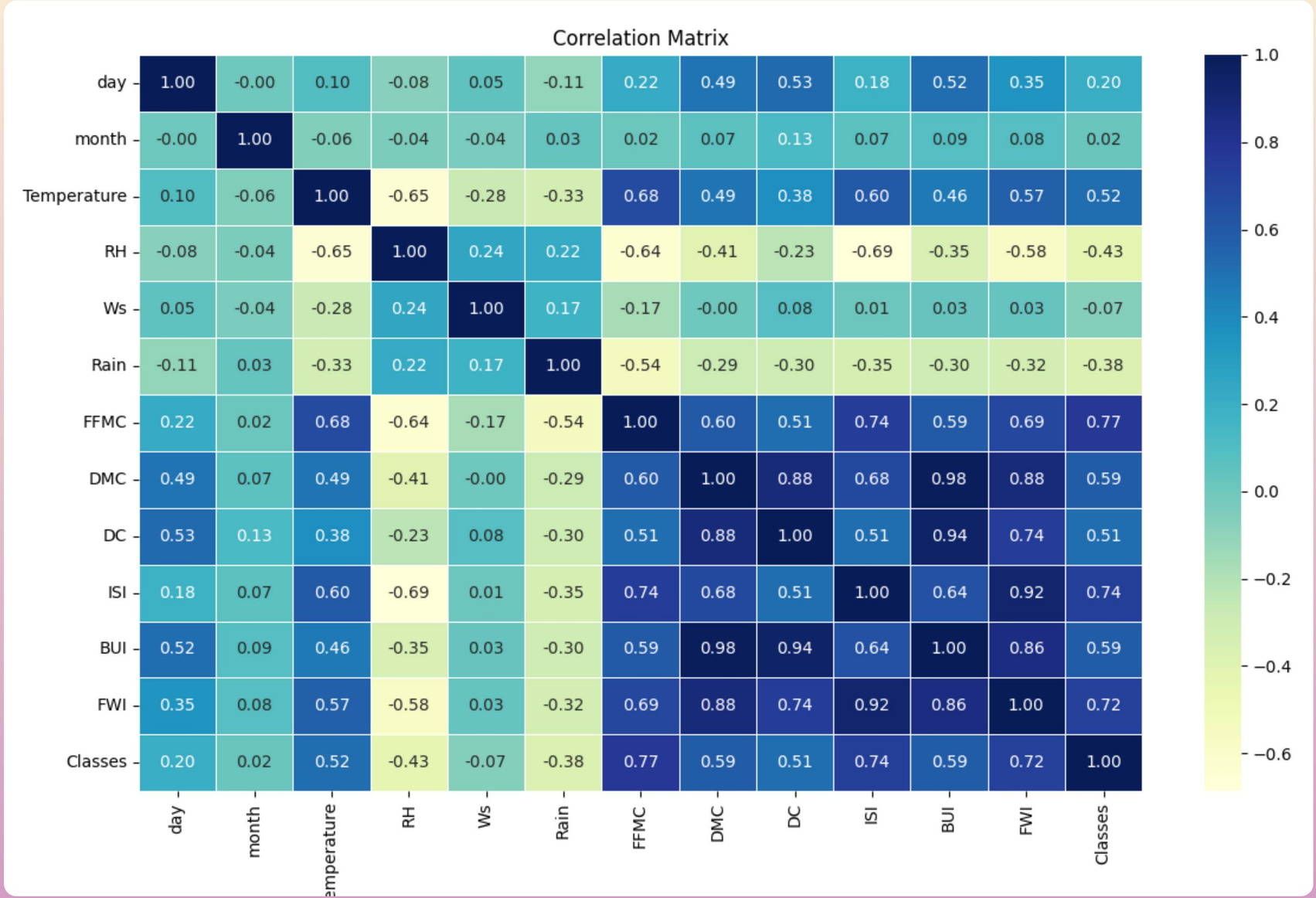


4.3.1.4 Predicting Forest Fire Linear Regression



1. **Problem definition**
2. **Data**
3. **Features**
4. **Data Exploration**
5. **Modelling**

4.3.1.4 Predicting Forest Fire Linear Regression



1. **Problem definition**
2. **Data**
3. **Features**
4. **Data Exploration**
5. **Modelling**

4.3.1.4 Predicting Forest Fire Linear Regression

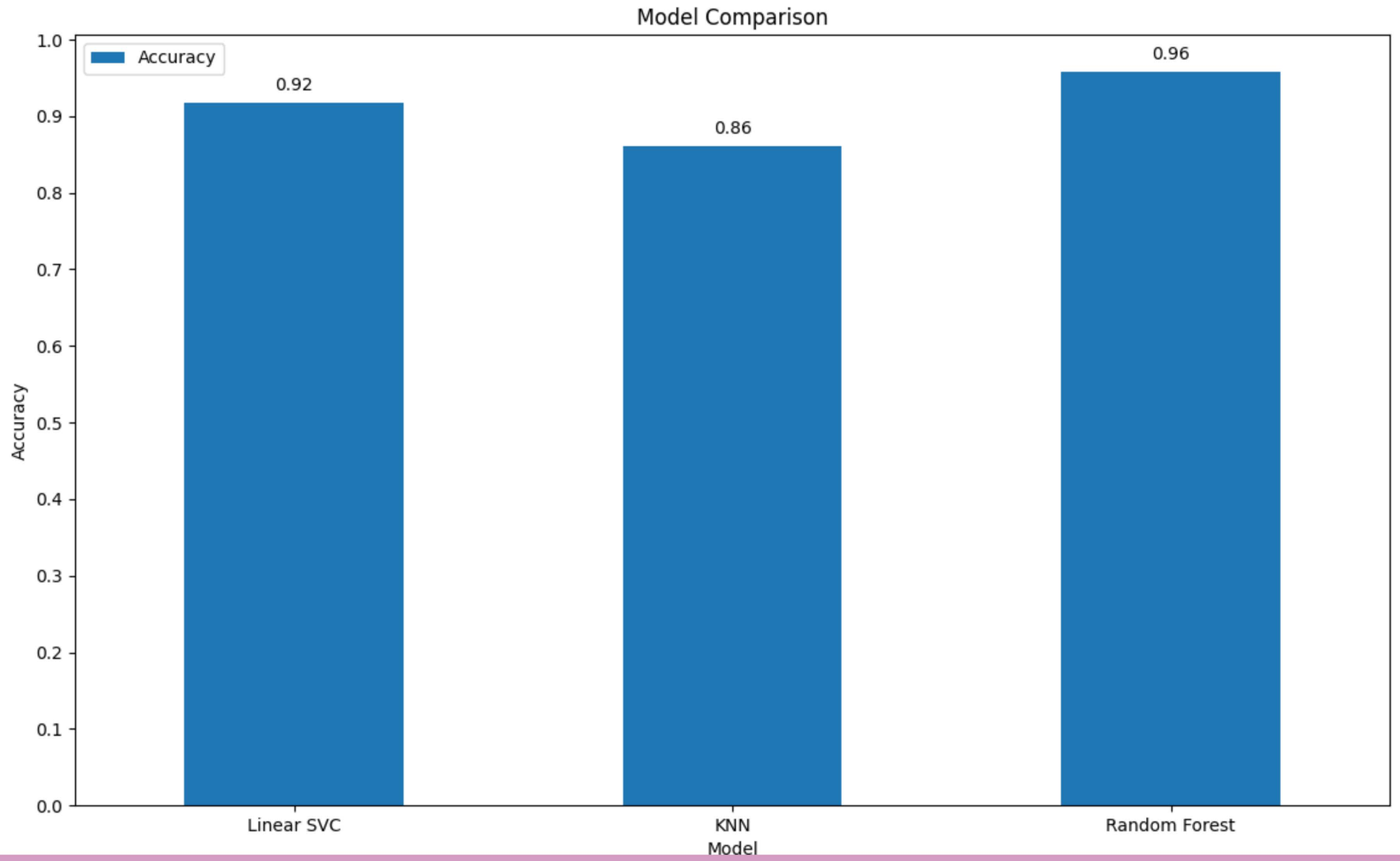
1. **Problem definition**
2. **Data**
3. **Features**
4. **Data Exploration**
5. **Modelling**

4.2.2.5 - Lab

```python
# Put models in a dictionary
models = {"Linear SVC": LinearSVC(dual='auto', max_iter=10000),
"KNN": KNeighborsClassifier(), "Random Forest": RandomForestClassifier()}

# Create a function to fit and score models
def fit_and_score(models, x_train, x_test, y_train, y_test):
    # Set random seed
    np.random.seed(5)
    # Make a dictionary to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(x_train, y_train)
        # Evaluate the model and append its score to model_scores
        y_pred = model.predict(x_test)
        fpr, tpr, _ = roc_curve(y_test, y_pred)
        roc_auc = auc(fpr, tpr)
        model_scores[name] = roc_auc
    return model_scores

model_scores = fit_and_score(models=models, x_train=x_train, x_test=x_test, y_train=y_train, y_test=y_test)
```

Fits and evaluates given machine learning models
models : a dict of different Scikit-Learn machine learning models
x_train : training data (no labels)
x_test : testing data (no labels)
y_train : training labels
y_test : test labels

4.3.1.4 Predicting Forest Fire Linear Regression



1. Problem definition
2. Data
3. Features
4. Data Exploration
5. Modelling

Chapter 4: Advanced Analytics and machine learning

- 4.1.2.4 Simple Linear Regression in Python
- 4.1.3.5 Decision Tree Classification
- 4.2.2.5 Evaluating Fit Errors in Linear Regression
- 4.3.1.4 Internet Meter Linear Regression
- **4.3.2.4 Internet Meter Anomaly Detection** ✅

Objectives:

Part 1: Feature Engineering

Part 2: Euclidean Anomaly Detection

4.3.2.4 Internet Meter Anomaly Detection

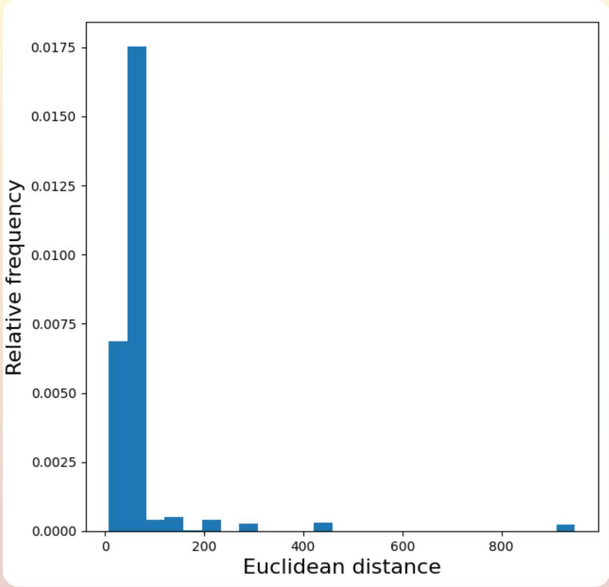
Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. In simpler terms, it is the length of the line segment connecting two points. In the context of the lab, Euclidean distance is used to quantify the similarity or dissimilarity between data points in a multidimensional space

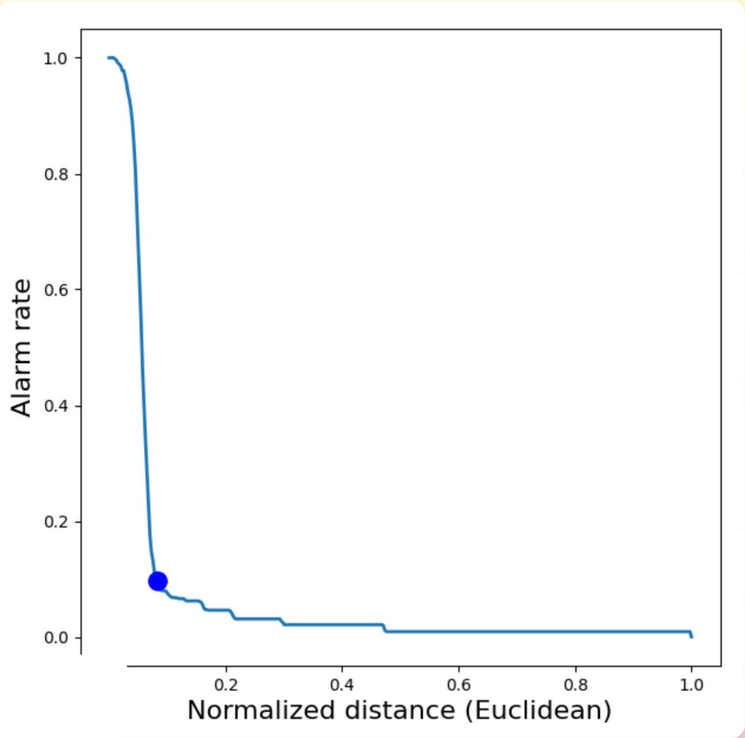
4.2.2.5 - Lab

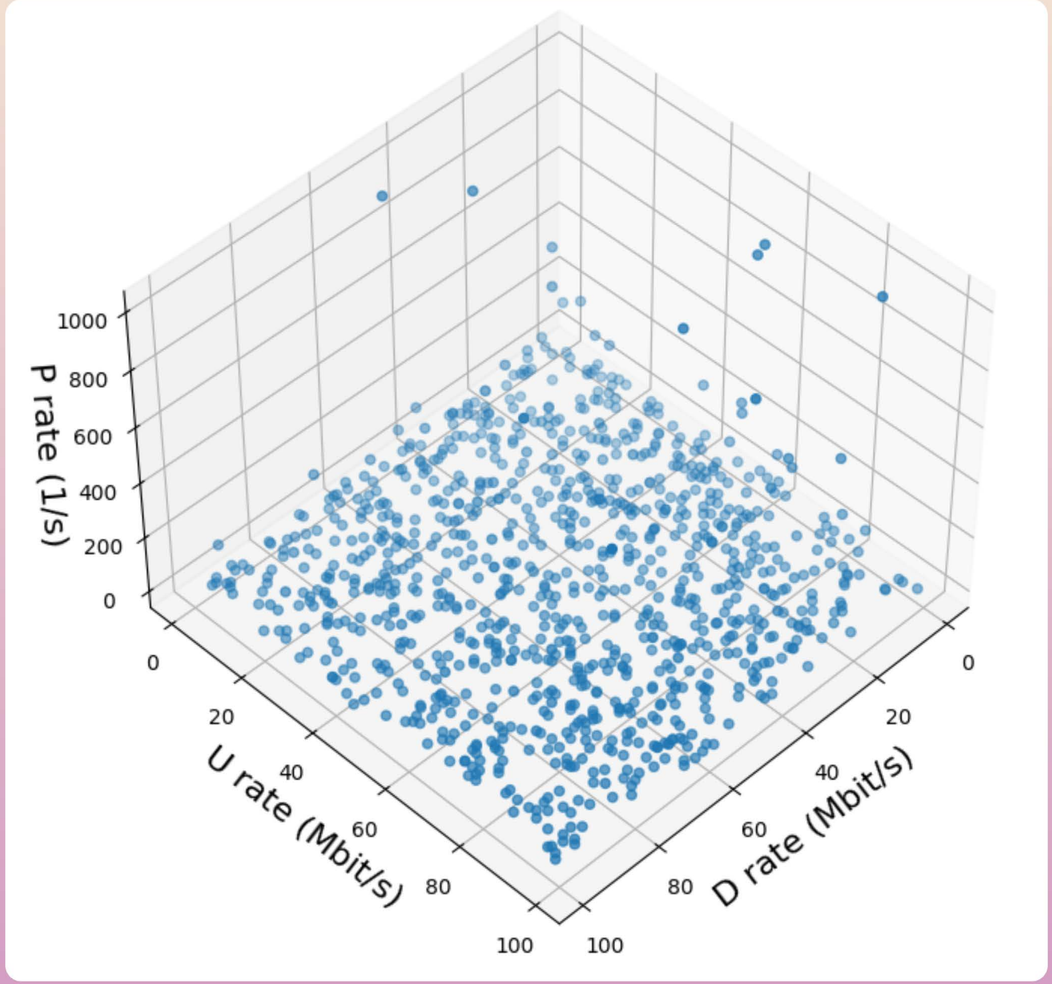
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

# Step 8: Visualize the decision boundary in 3D
radius = ecl_decision * max_euclid
phi = np.linspace(0, 2 * np.pi, 300)
theta = np.linspace(0, 2 * np.pi, 300)
xs = radius * np.outer(np.sin(theta), np.cos(phi))
ys = radius * np.outer(np.sin(theta), np.sin(phi))
zs = radius * np.outer(np.cos(theta), np.ones(np.size(phi)))
ecl_xd = xs + df_rates['download_rate'].mean()
ecl_yd = ys + df_rates['upload_rate'].mean()
ecl_zd = zs + df_rates['ping_rate'].mean()
```

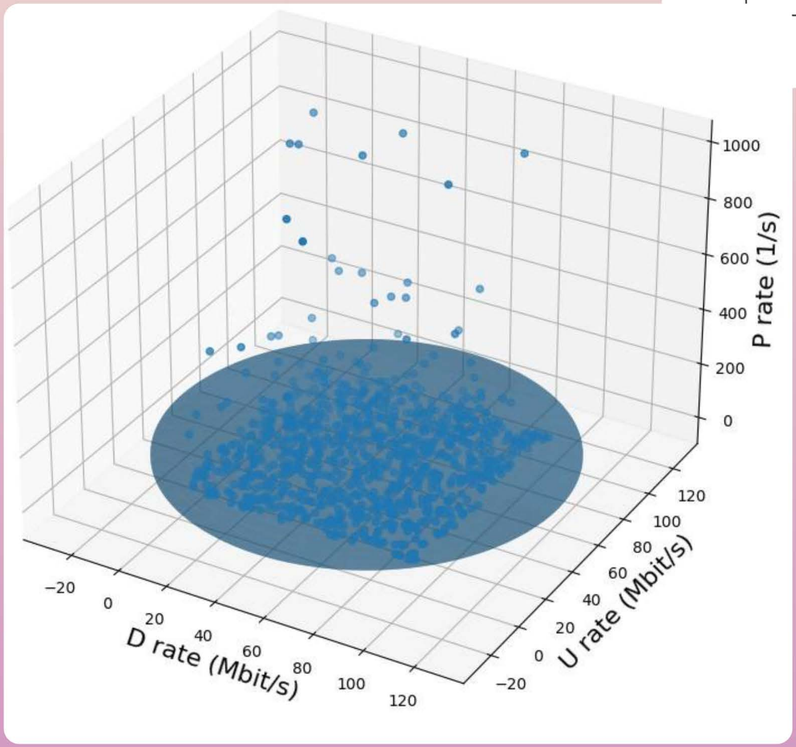
Decision boundary is a line, surface, or hypersurface that separates different classes in a classification problem. It's like a boundary line drawn on a map that separates one area from another. In machine learning, the decision boundary separates different groups or categories in a dataset based on their features.

Relative frequency is high cause most of the data is inside the **Decision boundary**









Chapter 5: Storytelling with Data

- **5.2.3.8 Visualization Data in Excel**
- 5.3.1.4 Advanced Data Visulization
- 5.3.1.5 Intrnet Speed Compliance

Objectives:

Part 1: Prepare Data for Visualization
Part 2: Create Excel Charts for Visualizing Data

Chapter 5: Storytelling with Data

- 5.2.3.8 Visualization Data in Excel
- **5.3.1.4 Advanced Data Visulization**
- 5.3.1.5 Intrnet Speed Compliance

Objectives:

Part 1: Prepare the data

Part 2: Visualize the data

5.3.1.4 Advanced Data Visulization

Allows to create interactive maps and visualize geospatial data

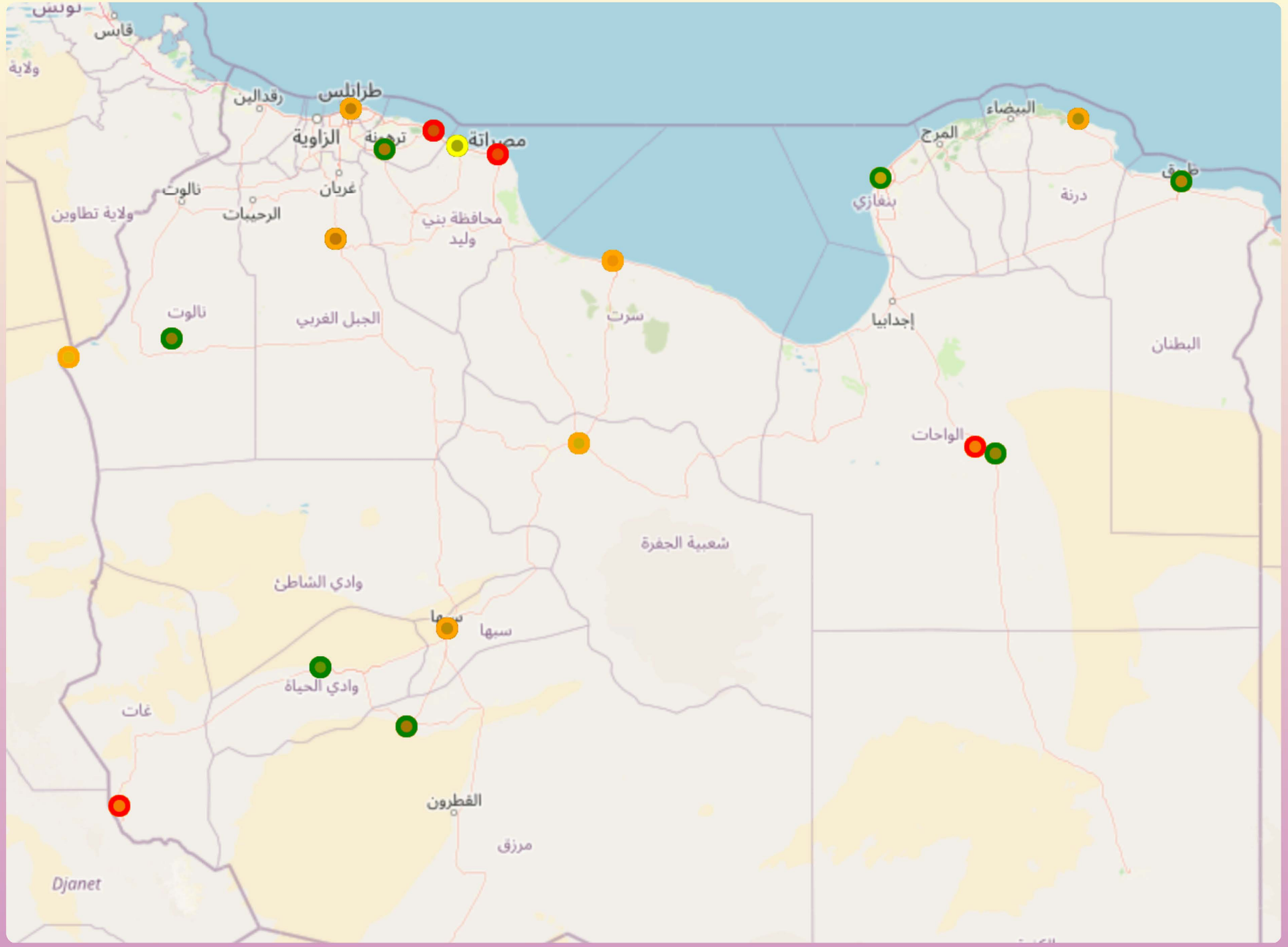
used for encoding and decoding JSON data

5.3.1.4 - Lab

```python
import pandas as pd
import numpy as np
import folium
import json

# Read the data from the CSV file into a DataFrame
df = pd.read_csv('libyan_cities_data.csv')
# Read the JSON file into a dictionary
with open('libyan_cities_coordinates.json') as f:
    coordinates = json.load(f)
# Define a function to assign colors based on value ranges
def color_producer(val):
    if val < 20:
        return 'red'
    elif val < 40:
        return 'orange'
    elif val < 60:
        return 'yellow'
    else:
        return 'green'
```

5.3.1.4 - Lab

```python
# Create a map centered at a specific location
mymap = folium.Map(location=[25.0, 17.0], zoom_start=6)

# Add markers for each city
for index, row in df.iterrows():
    city = row['City']
    value = row['Value']
    if city in coordinates:
        lat = coordinates[city]['latitude']
        lon = coordinates[city]['longitude']
        color = color_producer(value)
        folium.CircleMarker(location=[lat, lon],
        radius=5, popup=f'{city}: {value}',
        color=color, fill=True).add_to(mymap)

# Display the map
mymap.save('intrnet_speed_map.html')
```


تونس
قابس
ولاية
رقدالين
طرابلس
الزاوية
ترهونة
مصراتة
نالوت
غريان
الرحيبات
ولاية تطاوين
محافظة بني وليد
نالوت
الجبل الغربي
سرت
البيضاء
المرج
بنغازي
درنة
طبرق
إجدابيا
البطنان
الواحات
شعبية الجفرة
وادي الشاطئ
سبها
سبها
وادي الحياة
غات
القطرون
مرزق
Djanet

Chapter 5: Storytelling with Data

- 5.2.3.8 Visualization Data in Excel
- 5.3.1.4 Advanced Data Visulization
- **5.3.1.5 Intrnet Speed Compliance**

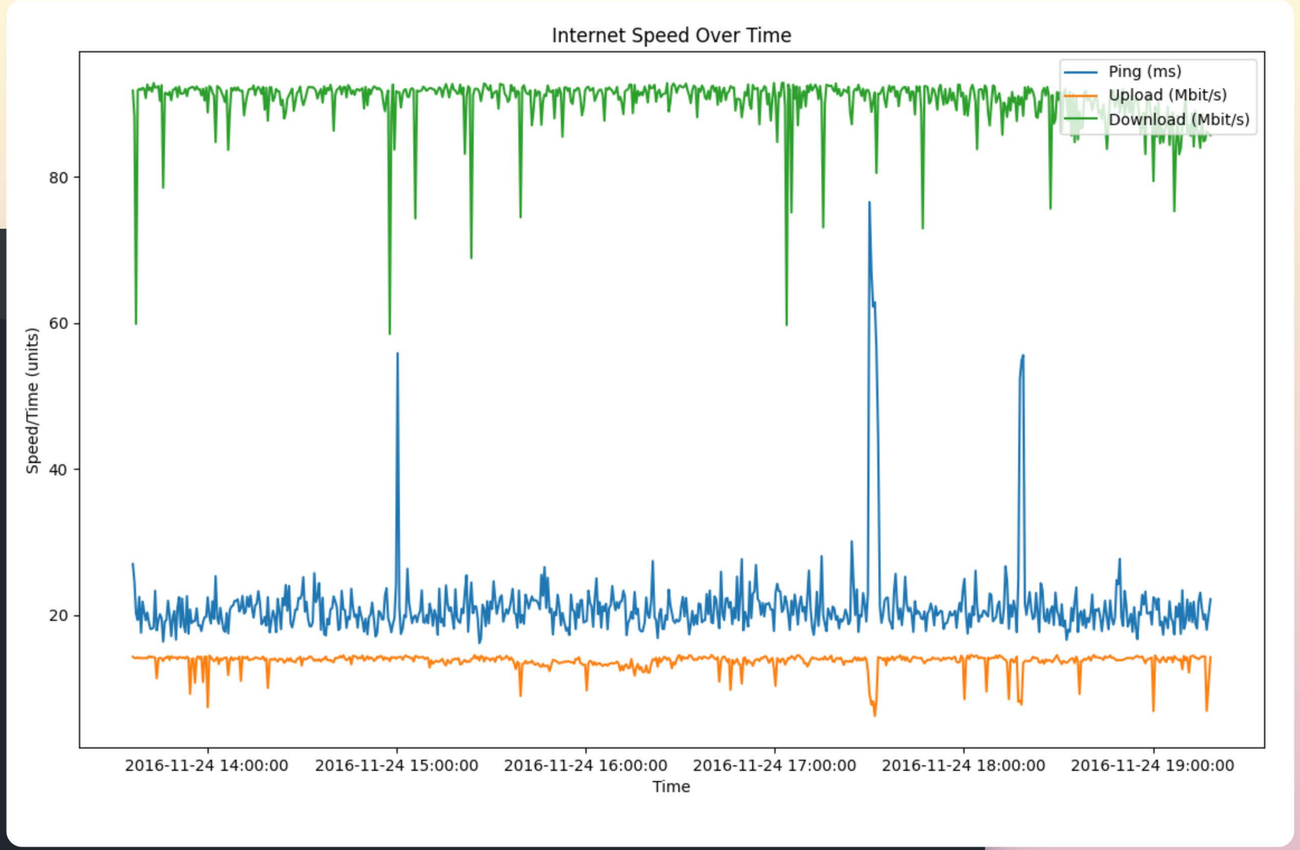
Objectives:

- Part 1: Determine if the Delivered Internet Speed Complies with the Provider's Contract
- Part 2: Write a Report to the Internet Provider (Optional)

5.3.1.5 Intrnet Speed

5.3.1.5 - Lab

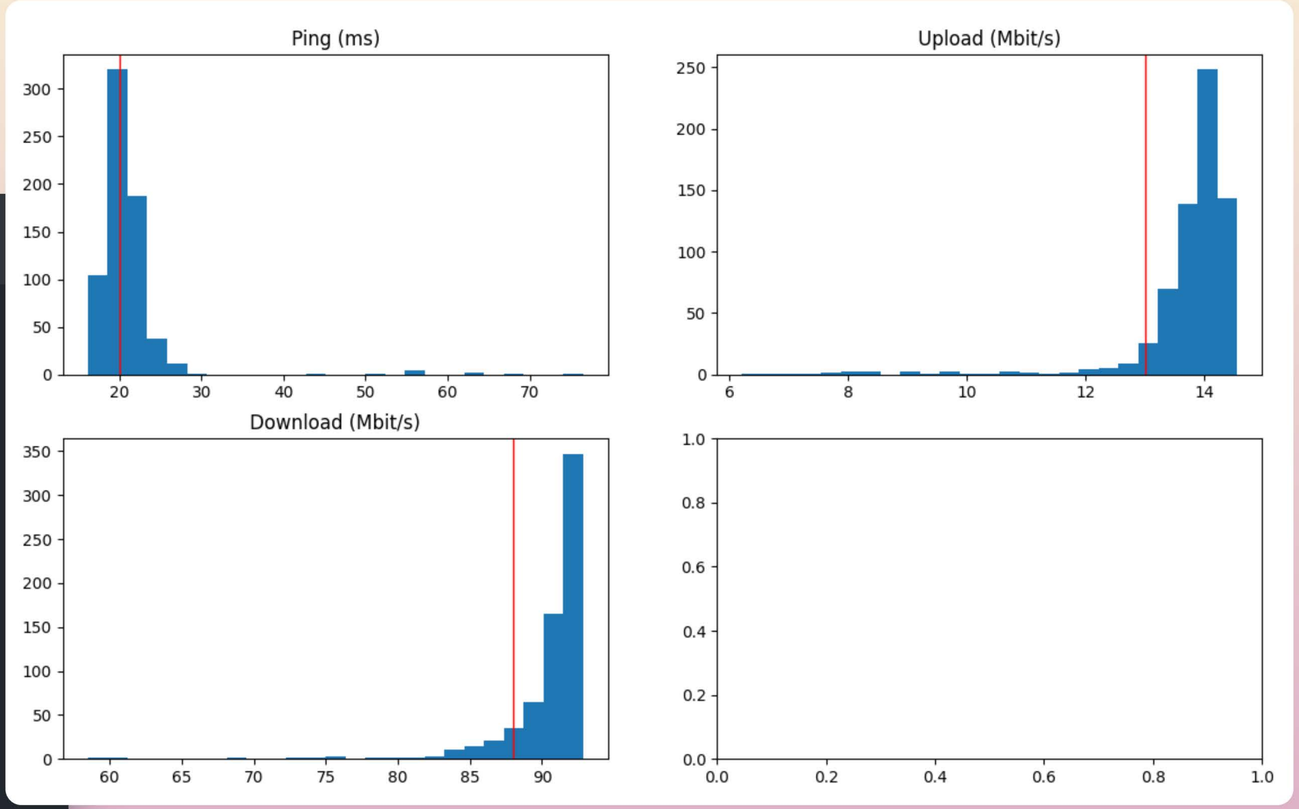
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Load the data from the CSV file
df_compact = pd.read_csv('rpi_data_processed.csv')

# Remove any rows with NaN values
df_compact = df_compact.dropna()

# Convert 'Date' and 'Time' columns to datetime objects
df_compact['Time'] = pd.to_datetime(df_compact['Time'])
df_compact['Date'] = pd.to_datetime(df_compact['Date'] + ' ' + df_compact['Time'].dt.strftime('%H:%M:%S'))

# Plot the data
fig, ax = plt.subplots(figsize=(8, 8))
ax.plot(df_compact['Date'], df_compact['Ping (ms)'], label='Ping (ms)')
ax.plot(df_compact['Date'], df_compact['Upload (Mbit/s)'], label='Upload (Mbit/s)')
ax.plot(df_compact['Date'], df_compact['Download (Mbit/s)'], label='Download (Mbit/s)')
ax.legend(loc='upper right')
# Set x-axis format to include date
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d %H:%M:%S'))
```



5.3.1.5 Intrnet Speed Compliance

5.3.1.5 - Lab

```
# Define the acceptable values
acceptable_upload = 13  # Mbit/s
acceptable_download = 88  # Mbit/s
acceptable_ping = 20  # ms

# Plot histograms
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
ax[0, 0].hist(df_compact['Ping (ms)'], bins=25)
ax[0, 0].axvline(acceptable_ping, color='red', linewidth=1)
ax[0, 0].set_title('Ping (ms)')

ax[0, 1].hist(df_compact['Upload (Mbit/s)'], bins=25)
ax[0, 1].axvline(acceptable_upload, color='red', linewidth=1)
ax[0, 1].set_title('Upload (Mbit/s)')

ax[1, 0].hist(df_compact['Download (Mbit/s)'], bins=25)
ax[1, 0].axvline(acceptable_download, color='red', linewidth=1)
ax[1, 0].set_title('Download (Mbit/s)')
```



Chapter 6: Architecture for Big Data and Data Engineering

- **6.1.3.4 Install a Virtual Machine on a Personal Computer**
- 6.4.1.2 Install and Test the Raspberry Pi Camer
- 6.4.1.3 Image Processing Change Detection
- 6.4.1.4 Smile Detection

Objectives:

Part 1: Download and Install VirtualBox
Part 2: Download the Virtual Machine Image File
Part 3: Create the Virtual Machine
Part 4: Install the Operating System
Part 5: Guest-Host Network Communication
Part 6: Challenge: Install a Second Virtual Machine (Optional)

Chapter 6: Architecture for Big Data and Data Engineering

- 6.1.3.4 Install a Virtual Machine on a Personal Computer
- **6.4.1.2 Install and Test the Raspberry Pi Camer**
- 6.4.1.3 Image Processing Change Detection
- 6.4.1.4 Smile Detection

Objectives:

Part 1: Setup the Raspberry Pi camera.
Part 2: Test the Raspberry Pi camera.
Part 3: Capture images with Python.

Chapter 6: Architecture for Big Data and Data Engineering

- 6.1.3.4 Install a Virtual Machine on a Personal Computer
- 6.4.1.2 Install and Test the Raspberry Pi Camer
- **6.4.1.3 Image Processing Change Detection** ✅
- 6.4.1.4 Smile Detection

Objectives:

Part 1: Learn About Computer Imaging
Part 2: Capture Images
Part 3: Change Detection
Part 4: Explanation and Application

6.4.1.3 Image Processing Change Detection

5.3.1.5 - Lab

```python
import cv2
from matplotlib import pyplot as plt

# Initialize the webcam with the index for the back camera
cap = cv2.VideoCapture(1)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Display the resulting frame using matplotlib
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

    # Break the loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam
cap.release()
```

Is an open-source computer vision and machine learning software library. It provides a wide range of functionalities for image and video processing, including real-time computer vision applications.

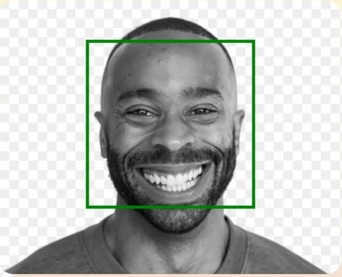
Chapter 6: Architecture for Big Data and Data Engineering

- 6.1.3.4 Install a Virtual Machine on a Personal Computer
- 6.4.1.2 Install and Test the Raspberry Pi Camer
- 6.4.1.3 Image Processing Change Detection
- **6.4.1.4 Smile Detection**

Objectives:

Part 1: Detect Faces
Part 2: Data Processing: Face Standardization
Part 3: Load a Pre-trained Moel
Part 4: Real-time Smile Detection

6.4.1.4 Smile Detection



Is a submodule of SciPy, a scientific computing library in Python. It provides functions for performing various operations on n-dimensional arrays, including image processing tasks such as filtering, interpolation, and morphology

6.4.1.4 - Lab

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy.ndimage import zoom
import pickle as pkl

# Load test images
smile = cv2.imread('smile.jpg')
no_smile = cv2.imread('no_smile.jpg')

# Load face detection classifier
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath)

# Detect faces in smile image
gray_smile = cv2.cvtColor(smile, cv2.COLOR_BGR2GRAY)
faces_smile = faceCascade.detectMultiScale(gray_smile, scaleFactor=1.1, minNeighbors=6, minSize=(100, 100))
```

Is used for serializing and deserializing Python objects. It allows you to convert complex objects into a byte stream, which can be saved to a file or sent over a network, and then reconstructed back into their original form later. pickle is commonly used for saving and loading machine learning models

Note: I couldn't complete this lab because:

- I didn't find the pre-trained model.
- Also, it uses a real-time camera using

You can find All the labs at the following link:

https://github.com/FoozBarakat/Special_Topics_ITE697