untitled

November 21, 2024

[4]:
```
input_string = input("Enter String :")
words = input_string.split()

word_freq={}

for word in words:
    word= word.lower()
    if word in word_freq:
        word_freq[word]+=1
    else:
        word_freq[word]=1
print("Word Freq - ")
for word,count in word_freq.items():
    print(f"{word} : {count}")
```

```
Enter String : Hello Hello how are you you
```

```
Word Freq -
hello : 2
how : 1
are : 1
you : 2
```

[11]:
```
def count_word_in_file(filepath):
    try:
        with open(filepath, 'r') as file:
            contents = file.read()
            words = contents.split()
            word_count=len(words)
            print(f"The file contains {word_count} words")

    except Exception as e:
        print(f"An error occurred: {e}")

filepath = "freq_sample.txt"
count_word_in_file(filepath)
```

```
The file contains 2 words
```

[17]:
```python
import pandas as pd

data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve"],
    "Age": [24, 27, 22, 32, 29],
    "Score": [85, 90, 78, 88, 92]
}
df = pd.DataFrame(data)
print("\nOriginal datafarme: \n",df)

print("\nSelecting Column: \n", df["Name"])
print("\nSelecting Rows: \n", df.loc[df["Age"]>30])

# Filter rows where Age is between 25 and 30
filtered_df = df[(df["Age"] >= 25) & (df["Age"] <= 30)]
print("\nRows where 25 <= Age <= 30:\n", filtered_df)

print("\nSorting Rows based on score\n", df.sort_values(by="Score"))
```

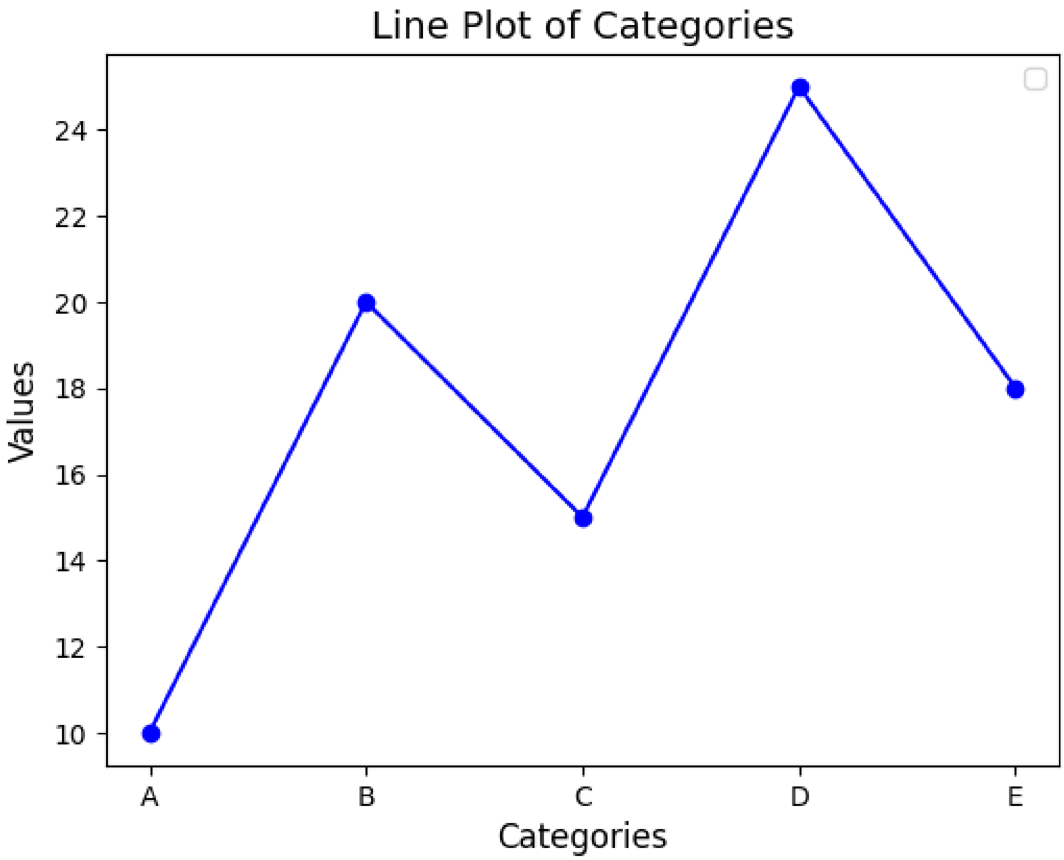
```
Original datafarme:
       Name  Age  Score
0    Alice   24     85
1      Bob   27     90
2  Charlie   22     78
3    David   32     88
4      Eve   29     92

Selecting Column:
 0      Alice
1        Bob
2    Charlie
3      David
4        Eve
Name: Name, dtype: object

Selecting Rows:
     Name  Age  Score
3  David   32     88

Rows where 25 <= Age <= 30:
   Name  Age  Score
1  Bob   27     90
4  Eve   29     92

Sorting Rows based on score
       Name  Age  Score
```

```
2  Charlie   22     78
0    Alice   24     85
3    David   32     88
1      Bob   27     90
4      Eve   29     92
```

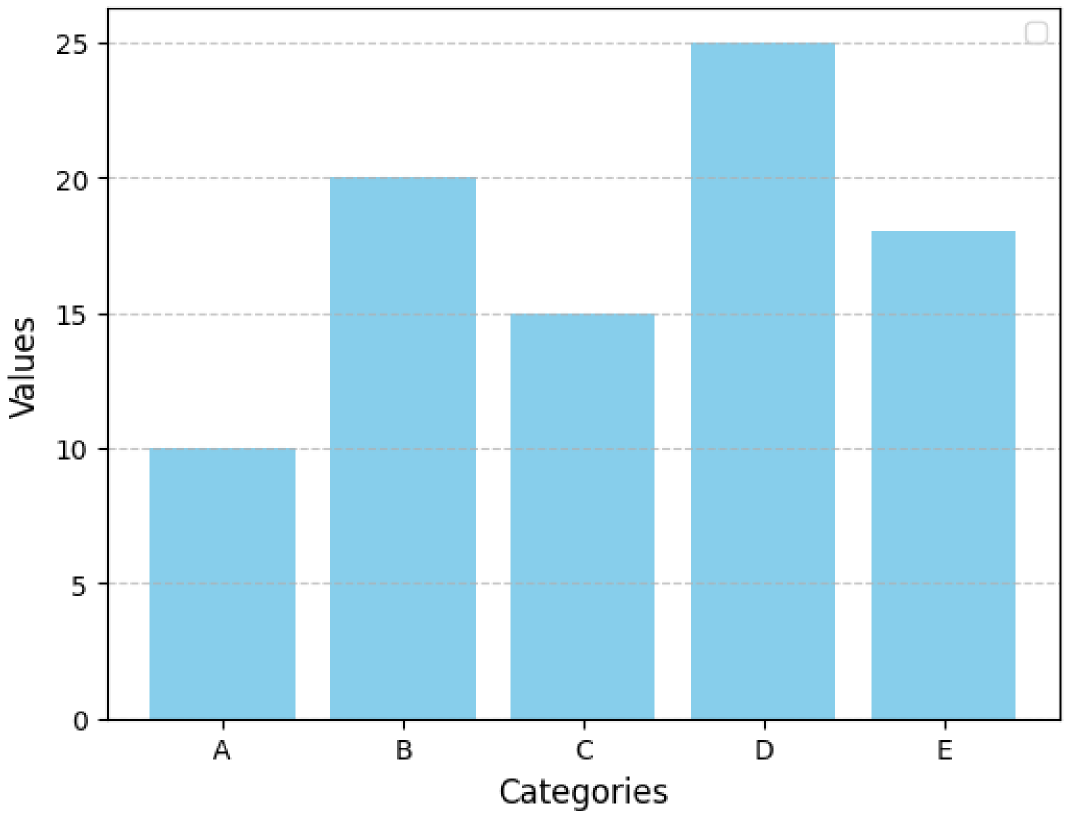
[25]:

```python
import matplotlib.pyplot as plt

categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 25, 18]

plt.plot(categories, values, marker='o', color='b')

plt.title("Line Plot of Categories", fontsize=14)
plt.xlabel("Categories", fontsize=12)
plt.ylabel("Values", fontsize=12)

plt.legend()
plt.show()

plt.bar(categories, values, color='skyblue')
plt.xlabel("Categories", fontsize=12)
plt.ylabel("Values", fontsize=12)

plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend()

plt.show()
```

```
C:\Users\91762\AppData\Local\Temp\ipykernel_7852\4162411356.py:12: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```




```
C:\Users\91762\AppData\Local\Temp\ipykernel_7852\4162411356.py:20: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```



[29]:
```
import sqlite3

conn = sqlite3.connect('sample.db')
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS Employees (
    ID INTEGER PRIMARRY KEY,
    Name TEXT,
    Age INTEGER,
    Salary Real
)
''')

sample_data = [
    (1, 'Alice', 30, 60000),
    (2, 'Bob', 35, 80000),
    (3, 'Charlie', 28, 50000),
    (4, 'David', 40, 95000),
    (5, 'Eve', 25, 55000)
]
```

```python
cursor.executemany('''
INSERT INTO Employees (ID, Name, Age, Salary)
VALUES (?,?,?,?)
''', sample_data)

conn.commit()
```

[33]:
```python
import pandas as pd

df = pd.read_sql_query("SELECT * FROM Employees", conn)

print("\nDataFrame: \n",df)

selected_col = df[['Name','Salary']]

print("\nSelected Columns (Name and Salary):\n", selected_col)

filtered_row = df[df["Age"]>30]

print("\nEmployees with Age > 30:\n", filtered_row)
```

```
DataFrame:
    ID     Name  Age   Salary
0   1    Alice   30  60000.0
1   2      Bob   35  80000.0
2   3  Charlie   28  50000.0
3   4    David   40  95000.0
4   5      Eve   25  55000.0

Selected Columns (Name and Salary):
       Name   Salary
0    Alice  60000.0
1      Bob  80000.0
2  Charlie  50000.0
3    David  95000.0
4      Eve  55000.0

Employees with Age > 30:
    ID   Name  Age   Salary
1   2    Bob   35  80000.0
3   4  David   40  95000.0
```

[53]:
```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

```
# Step 1: Send an HTTP request to fetch the webpage
url = "https://en.wikipedia.org/wiki/List_of_countries_by_population"
response = requests.get(url)

# Step 2: Parse the webpage content using BeautifulSoup
soup = BeautifulSoup(response.content, 'html.parser')

# Step 3: Find the table on the page
table = soup.find('table', {'class': 'wikitable'})

# Step 4: Extract table headers
headers = []
for th in table.find_all('th'):
    headers.append(th.get_text(strip=True))

# Step 5: Extract table rows
rows = []
for tr in table.find_all('tr')[1:]:  # Skipping the header row
    row = []
    for td in tr.find_all('td'):
        row.append(td.get_text(strip=True))
    rows.append(row)

# Step 6: Convert the extracted data into a DataFrame
df = pd.DataFrame(rows, columns=headers)

# Display the first few rows of the DataFrame
print(df.head())
```

```
                Location      Population  % ofworld                     Date  \
0       -          World  8,119,000,000       100%               1 Jul 2024
1  1/2[b]          China  1,409,670,000      17.3%              31 Dec 2023
2   India  1,404,910,000          17.3%  1 Jul 2024  Official projection[6]
3       3  United States    335,893,238       4.1%               1 Jan 2024
4       4      Indonesia    282,477,584       3.5%              31 Jun 2024

  Source (official or fromtheUnited Nations) Notes
0                       UN projection[1][3]
1                       Official estimate[5]   [c]
2                                        [d]  None
3                     Official projection[7]   [e]
4               National annual projection[8]
```

[66]:
```
# 1. Select specific columns
selected_columns = df[['Location', 'Population']]
```

```
print("\nSelected Columns (Location and Population):\n", selected_columns.
 ↪head())

# 2. Filter rows where population > 100 million
filtered_data = df[df['Population'] > 100000000]
print("\nCountries with Population > 100 Million:\n",␣
 ↪filtered_data[['Location', 'Population']].head())

# 3. Sort by Population (Descending)
sorted_by_population = df.sort_values(by='Population', ascending=False)
print("\nCountries Sorted by Population (Descending):\n",␣
 ↪sorted_by_population[['Location', 'Population']].head())

# 4. Calculate the average population
average_population = df['Population'].mean()
print("\nAverage Population of Countries:\n", average_population)

# 5. Exclude the "World" row and find the country with the highest population
df_filtered = df[df['Location'] != 'World']  # Exclude 'World' row
max_population_country = df_filtered.loc[df_filtered['Population'].idxmax()]

print("\nCountry with the Highest Population (Excluding 'World'):\n",␣
 ↪max_population_country)
```

```
Selected Columns (Location and Population):
         Location    Population
0           World  8.119000e+09
1           China  1.409670e+09
2  1,404,910,000           NaN
3   United States  3.358932e+08
4       Indonesia  2.824776e+08

Countries with Population > 100 Million:
         Location    Population
0           World  8.119000e+09
1           China  1.409670e+09
3   United States  3.358932e+08
4       Indonesia  2.824776e+08
5        Pakistan  2.414994e+08

Countries Sorted by Population (Descending):
         Location    Population
0           World  8.119000e+09
1           China  1.409670e+09
3   United States  3.358932e+08
4       Indonesia  2.824776e+08
```

```
5       Pakistan  2.414994e+08

Average Population of Countries:
 61335961.820083685

Country with the Highest Population (Excluding 'World'):
                                                          1/2[b]
Location                                                   China
Population                                          1409670000.0
% ofworld                                                  17.3%
Date                                                 31 Dec 2023
Source (official or fromtheUnited Nations)    Official estimate[5]
Notes                                                        [c]
Name: 1, dtype: object
```

[70]:
```python
import pandas as pd
import requests
api_key='4b83d897e9844fddfbd4f9a272ee294c'
city = "London"
url=f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}'

response = requests.get(url)
data = response.json()

weather_data = {
    "City": [data["name"]],
    "Temperature (°C)": [data["main"]["temp"]],
    "Humidity (%)": [data["main"]["humidity"]],
    "Pressure (hPa)": [data["main"]["pressure"]],
    "Weather": [data["weather"][0]["description"]],
    "Wind Speed (m/s)": [data["wind"]["speed"]],
    "Coordinates": [data["coord"]]
}

# Create a DataFrame from the weather data
df = pd.DataFrame(weather_data)

# Step 4: Display the DataFrame
print("\nWeather Data for the City:\n", df)
```

```
Weather Data for the City:
      City  Temperature (°C)  Humidity (%)  Pressure (hPa)          Weather  \
0  London            273.21            91            1001  overcast clouds

   Wind Speed (m/s)                        Coordinates
0              0.45  {'lon': -0.1257, 'lat': 51.5085}
```

[76]:
```python
# 1. Select specific columns (e.g., Temperature and Wind Speed)
selected_columns = df[["Temperature (°C)", "Wind Speed (m/s)"]]
print("\nSelected Columns (Temperature and Wind Speed):\n", selected_columns)

# 2. Check if the temperature is greater than 20°C
if df["Temperature (°C)"].loc[0] > 20:
    print("\nTemperature is greater than 20°C.")
else:
    print("\nTemperature is not greater than 20°C. Current Temperature:",␣
 ↪df["Temperature (°C)"].iloc[0])

# 3. Calculate the average temperature (useful even with a single city)
average_temperature = df["Temperature (°C)"].mean()
print("\nAverage Temperature (for London):", average_temperature)

# 4. Extract Latitude and Longitude into separate columns
df[['Latitude', 'Longitude']] = pd.json_normalize(df['Coordinates']).iloc[:, 0:
 ↪2]
print("\nData with Separate Latitude and Longitude Columns:\n", df[['City',␣
 ↪'Latitude', 'Longitude']])

# 5. Display the weather description (to show textual data from the 'Weather'␣
 ↪column)
weather_description = df["Weather"].loc[0]
print("\nWeather Description:", weather_description)
```

```
Selected Columns (Temperature and Wind Speed):
    Temperature (°C)  Wind Speed (m/s)
0            273.21              0.45

Temperature is greater than 20°C.

Average Temperature (for London): 273.21

Data with Separate Latitude and Longitude Columns:
      City  Latitude  Longitude
0  London   -0.1257    51.5085

Weather Description: overcast clouds
```

[85]:
```python
import pandas as pd
df = pd.read_csv('heart.csv')

# Check for missing values
missing_values = df.isnull().sum()
```

```python
# Display the count of missing values for each column
print("Missing values in each column:")
print(missing_values)

# Step 2: Drop rows with missing values
df_dropped = df.dropna()
print("\nDataFrame after Dropping Rows with Missing Values:")
print(df_dropped)

# Step 3: Fill missing values with a specific value (e.g., 0 or 'Unknown')
df_filled_value = df.fillna(0)  # Replace missing values with 0 (you can choose␣
↪other values)
print("\nDataFrame after Filling Missing Values with 0:")
print(df_filled_value)

# Step 4: Fill missing values with the mean of the column
df_filled_mean = df.fillna(df.mean())  # Replace missing values with the mean␣
↪of the column
print("\nDataFrame after Filling Missing Values with Mean of Columns:")
print(df_filled_mean)
```

```
Missing values in each column:
age         0
gender      0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64

DataFrame after Dropping Rows with Missing Values:
     age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63       1   3       145   233    1        0      150      0      2.3
1     37       1   2       130   250    0        1      187      0      3.5
2     41       0   1       130   204    0        0      172      0      1.4
3     56       1   1       120   236    0        1      178      0      0.8
4     57       0   0       120   354    0        1      163      1      0.6
..   …      …  ..       …    …   …       …       …     …
298  57       0   0       140   241    0        1      123      1      0.2
```

```
299   45        1   3       110   264    0        1       132       0       1.2
300   68        1   0       144   193    1        1       141       0       3.4
301   57        1   0       130   131    0        1       115       1       1.2
302   57        0   1       130   236    0        0       174       0       0.0

     slope   ca  thal  target
0        0    0     1       1
1        0    0     2       1
2        2    0     2       1
3        2    0     2       1
4        2    0     2       1
..     …   ..    …      …
298      1    0     3       0
299      1    0     3       0
300      1    2     3       0
301      1    1     3       0
302      1    1     2       0

[303 rows x 14 columns]

DataFrame after Filling Missing Values with 0:
     age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63       1   3       145   233    1        0      150      0      2.3
1     37       1   2       130   250    0        1      187      0      3.5
2     41       0   1       130   204    0        0      172      0      1.4
3     56       1   1       120   236    0        1      178      0      0.8
4     57       0   0       120   354    0        1      163      1      0.6
..   …      …  ..       …    …  …       …       …     …       …
298   57       0   0       140   241    0        1      123      1      0.2
299   45       1   3       110   264    0        1      132      0      1.2
300   68       1   0       144   193    1        1      141      0      3.4
301   57       1   0       130   131    0        1      115      1      1.2
302   57       0   1       130   236    0        0      174      0      0.0

     slope   ca  thal  target
0        0    0     1       1
1        0    0     2       1
2        2    0     2       1
3        2    0     2       1
4        2    0     2       1
..     …   ..    …      …
298      1    0     3       0
299      1    0     3       0
300      1    2     3       0
301      1    1     3       0
302      1    1     2       0

[303 rows x 14 columns]
```

```
DataFrame after Filling Missing Values with Mean of Columns:
     age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63       1   3       145   233    1        0      150      0      2.3
1     37       1   2       130   250    0        1      187      0      3.5
2     41       0   1       130   204    0        0      172      0      1.4
3     56       1   1       120   236    0        1      178      0      0.8
4     57       0   0       120   354    0        1      163      1      0.6
..   …      … ..       …   …  …       …       …     …       …
298   57       0   0       140   241    0        1      123      1      0.2
299   45       1   3       110   264    0        1      132      0      1.2
300   68       1   0       144   193    1        1      141      0      3.4
301   57       1   0       130   131    0        1      115      1      1.2
302   57       0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     …  ..   …     …
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 14 columns]
```

[94]:

```python
import pandas as pd
from scipy.stats import zscore
import numpy as np

df = pd.read_csv('heart.csv')

z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))

outliers = z_scores>3

df_cleaned = df[~outliers.any(axis=1)]

# Step 5: Display the cleaned DataFrame
print("Cleaned DataFrame (after removing outliers):")
print(df_cleaned)
```

```
Cleaned DataFrame (after removing outliers):
     age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
```

```
0     63      1   3     145   233    1        0      150      0      2.3
1     37      1   2     130   250    0        1      187      0      3.5
2     41      0   1     130   204    0        0      172      0      1.4
3     56      1   1     120   236    0        1      178      0      0.8
4     57      0   0     120   354    0        1      163      1      0.6
..   ...    ...  ..   ...   ...  ...    ...    ...    ...    ...
298   57      0   0     140   241    0        1      123      1      0.2
299   45      1   3     110   264    0        1      132      0      1.2
300   68      1   0     144   193    1        1      141      0      3.4
301   57      1   0     130   131    0        1      115      1      1.2
302   57      0   1     130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     ...  ..   ...     ...
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[287 rows x 14 columns]
```

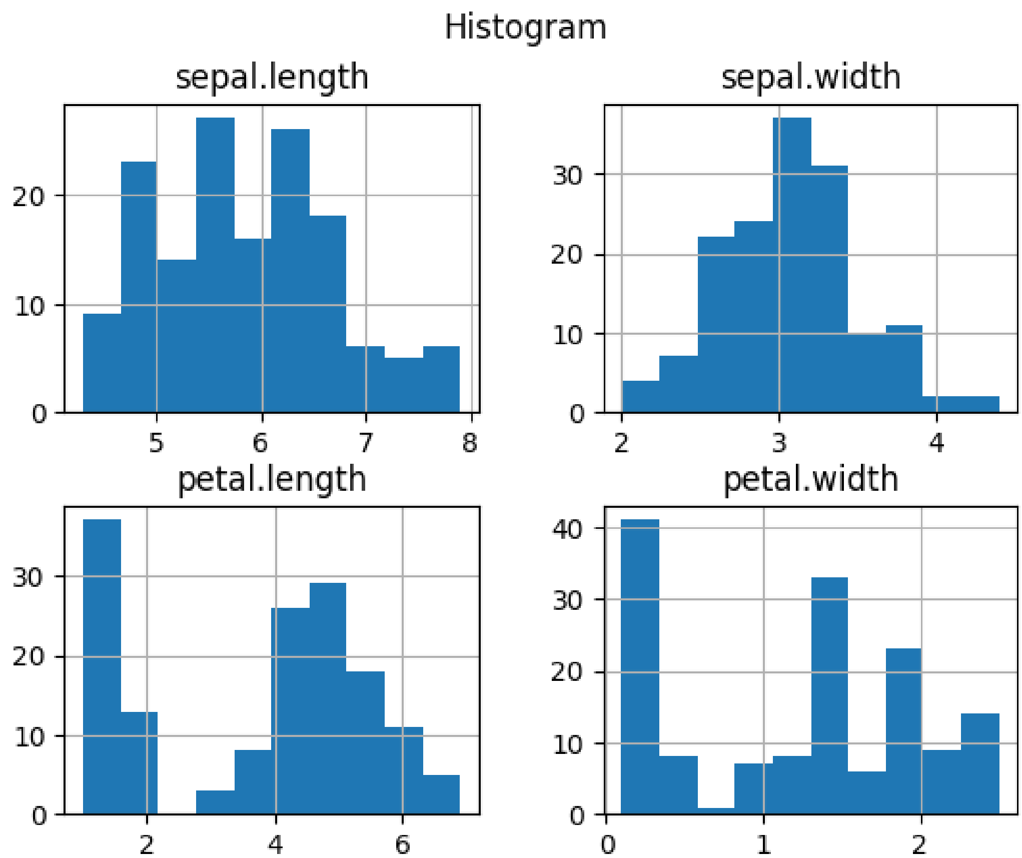
[106]:

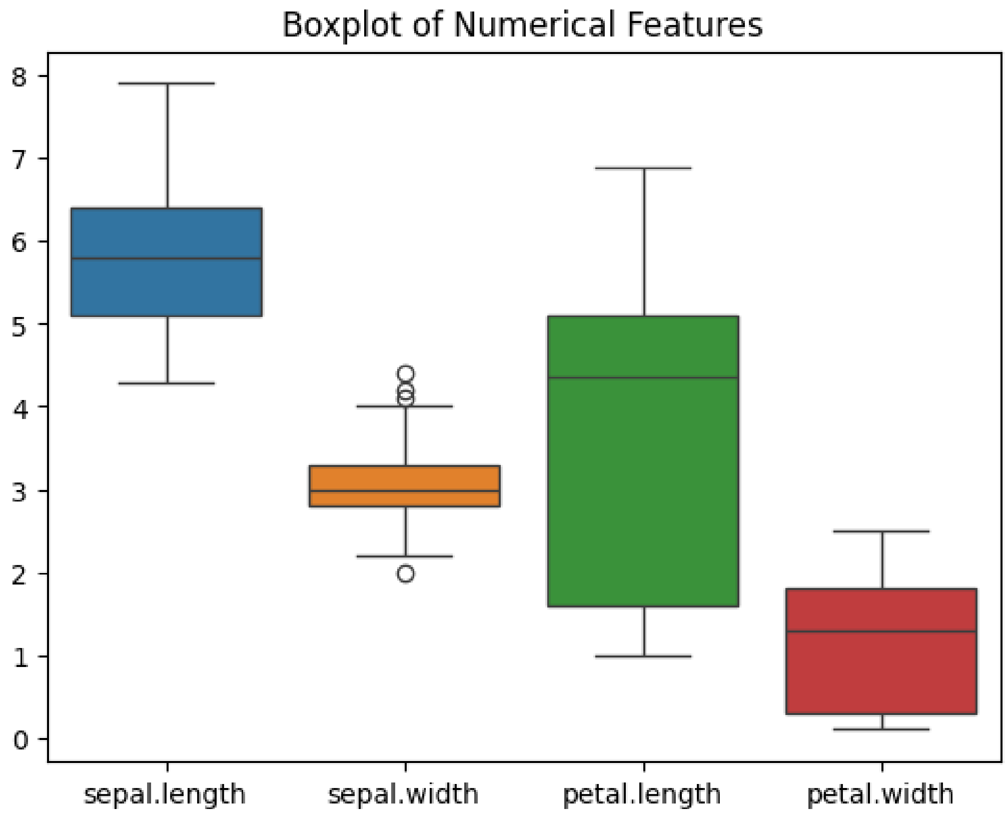
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('iris.csv')

print("\nHead : \n",df.head())
print("\nInfo : \n",df.info())
print("\nDescribe : \n",df.describe())
print("\nNull Values : \n",df.isnull().sum())

df.hist(bins=10)
plt.suptitle('Histogram')
plt.show()

sns.boxplot(data=df)
plt.title('Boxplot of Numerical Features')
plt.show()

sns.scatterplot(data=df, x='petal.length', y='petal.width')
```

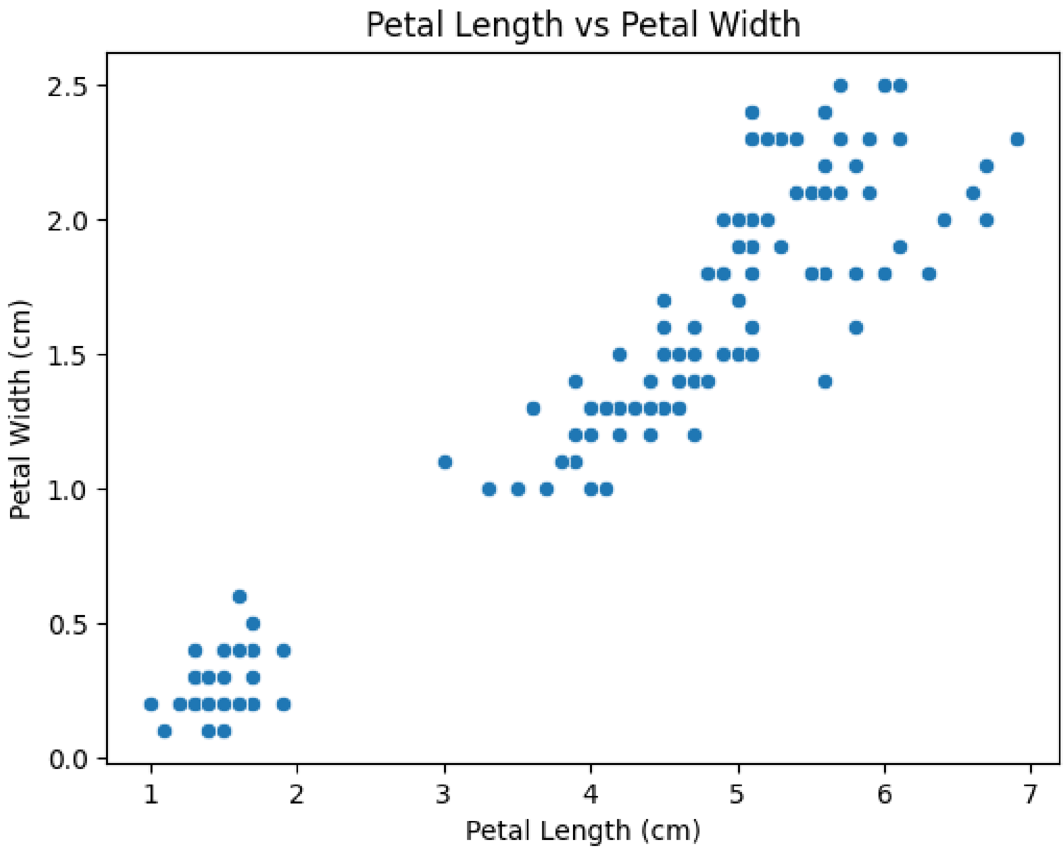
```python
plt.title('Petal Length vs Petal Width')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.show()

correlation_matrix = df.drop(columns=['species']).corr()

sns.heatmap(data=correlation_matrix,  cmap="coolwarm")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

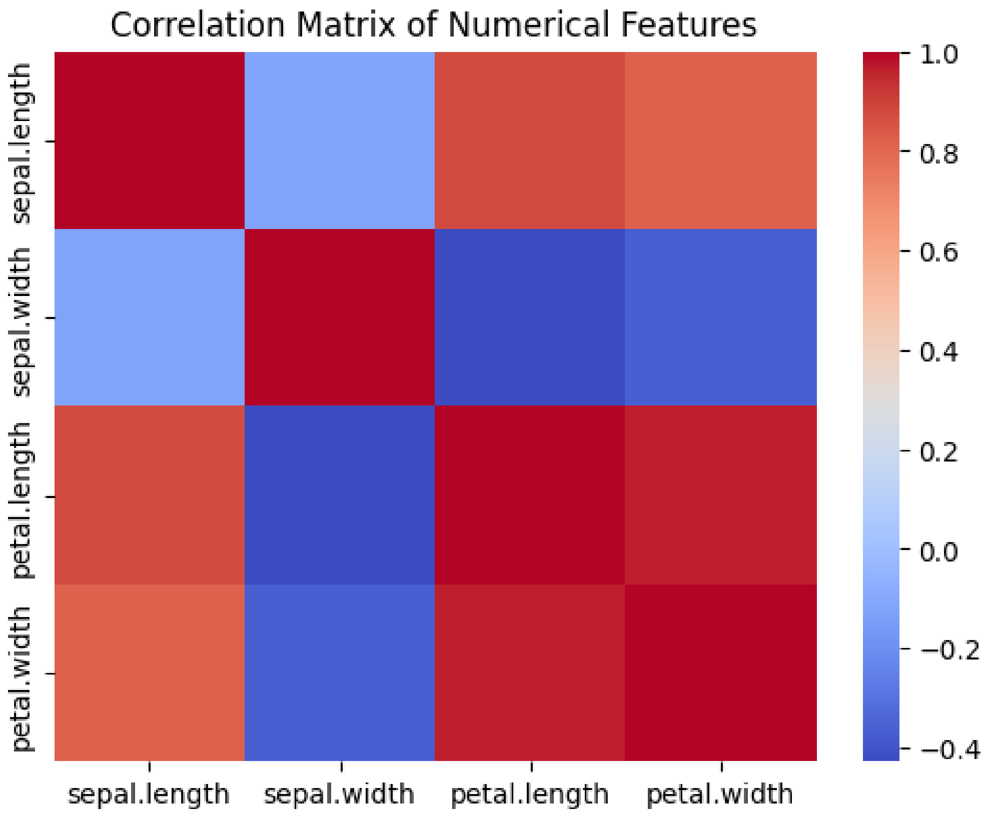
```
Head :
    sepal.length  sepal.width  petal.length  petal.width species
0            5.1          3.5           1.4          0.2  Setosa
1            4.9          3.0           1.4          0.2  Setosa
2            4.7          3.2           1.3          0.2  Setosa
3            4.6          3.1           1.5          0.2  Setosa
4            5.0          3.6           1.4          0.2  Setosa
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB

Info :
 None

Describe :
        sepal.length  sepal.width  petal.length  petal.width
count     150.000000   150.000000    150.000000   150.000000
mean        5.843333     3.057333      3.758000     1.199333
std         0.828066     0.435866      1.765298     0.762238
min         4.300000     2.000000      1.000000     0.100000
25%         5.100000     2.800000      1.600000     0.300000
50%         5.800000     3.000000      4.350000     1.300000
75%         6.400000     3.300000      5.100000     1.800000
max         7.900000     4.400000      6.900000     2.500000

Null Values :
```

```
 sepal.length    0
sepal.width     0
petal.length    0
petal.width     0
species         0
dtype: int64
```




Boxplot of Numerical Features
8
7
6
5
4
3
2
1
0
sepal.length
sepal.width
petal.length
petal.width


Petal Length vs Petal Width
Petal Width (cm)
Petal Length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7



[124]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset
df = pd.read_csv('Orange_Telecom_Churn_Data.csv')

# Display the first few rows of the dataset
print(df.head())

# Basic information about the dataset
print(df.info())

# Check for missing values
print(df.isnull().sum())

# Summary statistics for numerical columns
print(df.describe())

sns.histplot(df['total_day_minutes'], bins=20, color='blue')
```

```python
plt.title('total_day_minutes')
plt.xlabel('total_day_minutes')
plt.ylabel('Frequency')
plt.show()

sns.boxplot(y=df['total_day_charge'], color='green')
plt.title('Box Plot of total_day_charge ')
plt.xlabel('total_day_charge ')
plt.show()

# Calculate the correlation matrix
correlation_matrix = df.select_dtypes(include=[np.number]).corr()

sns.heatmap(correlation_matrix, cmap='coolwarm', linewidths=1)
plt.title('Correlation Heatmap')
plt.show()
```

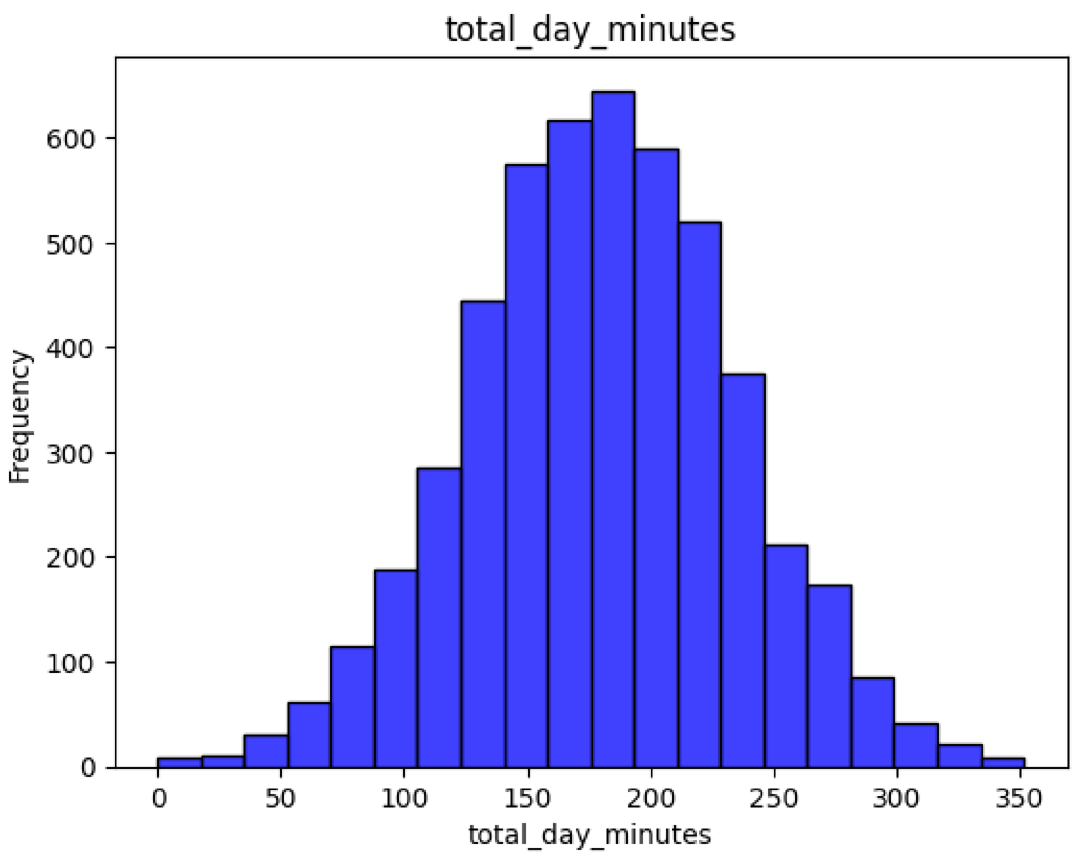
```
  state  account_length  area_code phone_number intl_plan voice_mail_plan  \
0    KS             128        415     382-4657        no             yes
1    OH             107        415     371-7191        no             yes
2    NJ             137        415     358-1921        no              no
3    OH              84        408     375-9999       yes              no
4    OK              75        415     330-6626       yes              no

   number_vmail_messages  total_day_minutes  total_day_calls  \
0                     25              265.1              110
1                     26              161.6              123
2                      0              243.4              114
3                      0              299.4               71
4                      0              166.7              113

   total_day_charge  …  total_eve_calls  total_eve_charge  \
0             45.07  …               99             16.78
1             27.47  …              103             16.62
2             41.38  …              110             10.30
3             50.90  …               88              5.26
4             28.34  …              122             12.61

   total_night_minutes  total_night_calls  total_night_charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total_intl_minutes  total_intl_calls  total_intl_charge  \
0                10.0                 3               2.70
```

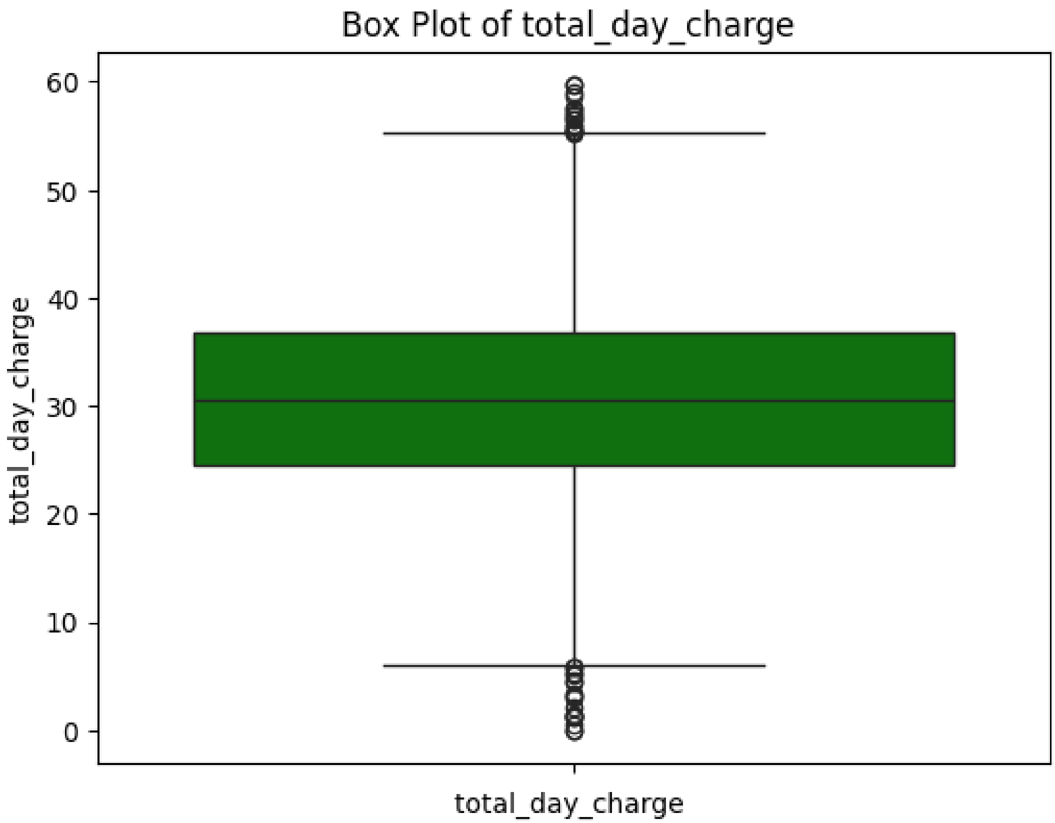
```
1               13.7                 3               3.70
2               12.2                 5               3.29
3                6.6                 7               1.78
4               10.1                 3               2.73

   number_customer_service_calls  churned
0                              1    False
1                              1    False
2                              0    False
3                              2    False
4                              3    False

[5 rows x 21 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   state                          5000 non-null   object
 1   account_length                 5000 non-null   int64
 2   area_code                      5000 non-null   int64
 3   phone_number                   5000 non-null   object
 4   intl_plan                      5000 non-null   object
 5   voice_mail_plan                5000 non-null   object
 6   number_vmail_messages          5000 non-null   int64
 7   total_day_minutes              5000 non-null   float64
 8   total_day_calls                5000 non-null   int64
 9   total_day_charge               5000 non-null   float64
 10  total_eve_minutes              5000 non-null   float64
 11  total_eve_calls                5000 non-null   int64
 12  total_eve_charge               5000 non-null   float64
 13  total_night_minutes            5000 non-null   float64
 14  total_night_calls              5000 non-null   int64
 15  total_night_charge             5000 non-null   float64
 16  total_intl_minutes             5000 non-null   float64
 17  total_intl_calls               5000 non-null   int64
 18  total_intl_charge              5000 non-null   float64
 19  number_customer_service_calls  5000 non-null   int64
 20  churned                        5000 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 786.3+ KB
None
state                            0
account_length                   0
area_code                        0
phone_number                     0
intl_plan                        0
voice_mail_plan                  0
```

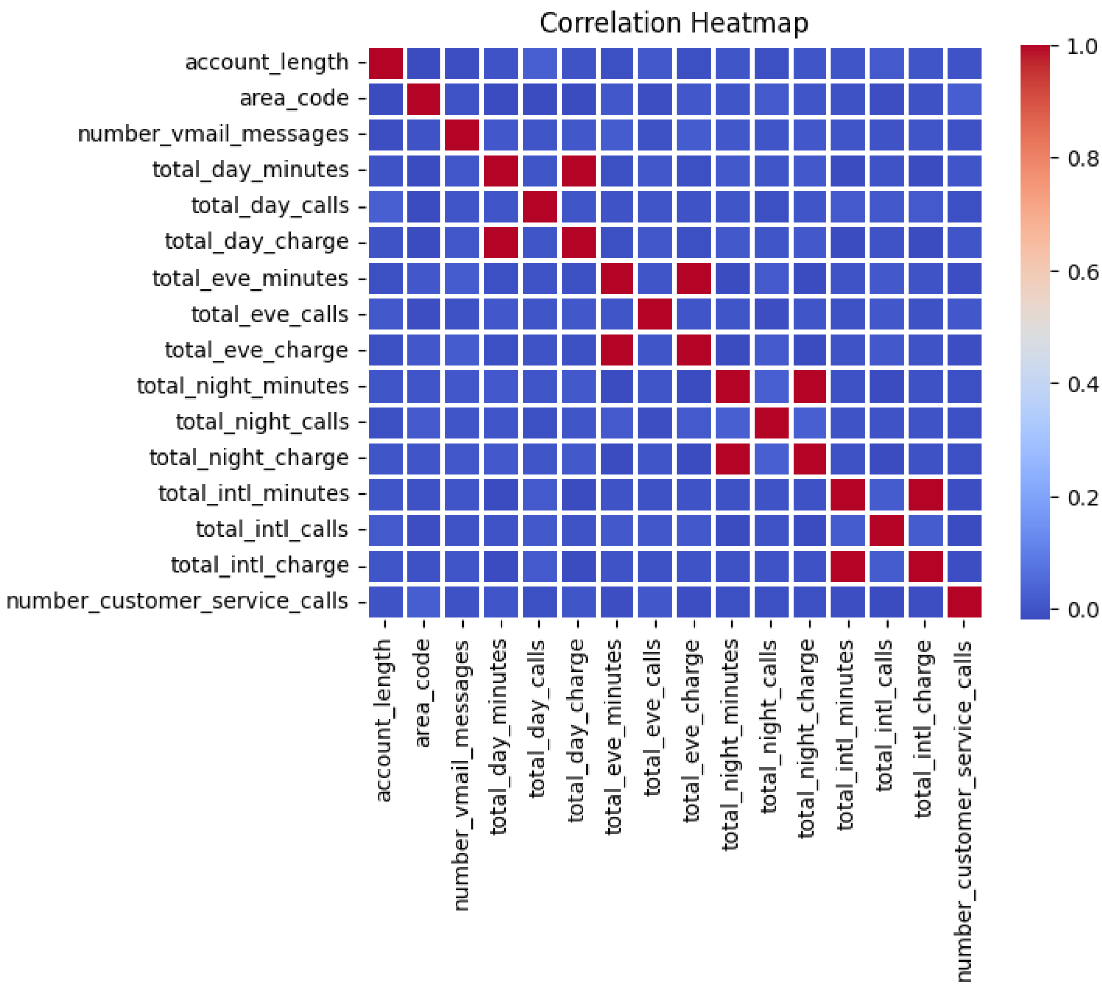
```
number_vmail_messages            0
total_day_minutes                0
total_day_calls                  0
total_day_charge                 0
total_eve_minutes                0
total_eve_calls                  0
total_eve_charge                 0
total_night_minutes              0
total_night_calls                0
total_night_charge               0
total_intl_minutes               0
total_intl_calls                 0
total_intl_charge                0
number_customer_service_calls    0
churned                          0
dtype: int64
       account_length    area_code  number_vmail_messages  total_day_minutes  \
count      5000.00000  5000.000000            5000.000000        5000.000000
mean        100.25860   436.911400               7.755200         180.288900
std          39.69456    42.209182              13.546393          53.894699
min           1.00000   408.000000               0.000000           0.000000
25%          73.00000   408.000000               0.000000         143.700000
50%         100.00000   415.000000               0.000000         180.100000
75%         127.00000   415.000000              17.000000         216.200000
max         243.00000   510.000000              52.000000         351.500000

       total_day_calls  total_day_charge  total_eve_minutes  total_eve_calls  \
count      5000.000000       5000.000000        5000.000000      5000.000000
mean        100.029400         30.649668         200.636560       100.191000
std          19.831197          9.162069          50.551309        19.826496
min           0.000000          0.000000           0.000000         0.000000
25%          87.000000         24.430000         166.375000        87.000000
50%         100.000000         30.620000         201.000000       100.000000
75%         113.000000         36.750000         234.100000       114.000000
max         165.000000         59.760000         363.700000       170.000000

       total_eve_charge  total_night_minutes  total_night_calls  \
count       5000.000000          5000.000000        5000.000000
mean          17.054322           200.391620          99.919200
std            4.296843            50.527789          19.958686
min            0.000000             0.000000           0.000000
25%           14.140000           166.900000          87.000000
50%           17.090000           200.400000         100.000000
75%           19.900000           234.700000         113.000000
max           30.910000           395.000000         175.000000

       total_night_charge  total_intl_minutes  total_intl_calls  \
count         5000.000000         5000.000000       5000.000000
```

mean	9.017732	10.261780	4.435200
std	2.273763	2.761396	2.456788
min	0.000000	0.000000	0.000000
25%	7.510000	8.500000	3.000000
50%	9.020000	10.300000	4.000000
75%	10.560000	12.000000	6.000000
max	17.770000	20.000000	20.000000

	total_intl_charge	number_customer_service_calls
count	5000.000000	5000.000000
mean	2.771196	1.570400
std	0.745514	1.306363
min	0.000000	0.000000
25%	2.300000	1.000000
50%	2.780000	1.000000
75%	3.240000	2.000000
max	5.400000	9.000000




Box Plot of total_day_charge
60
50
40
30
20
10
0
total_day_charge
total_day_charge

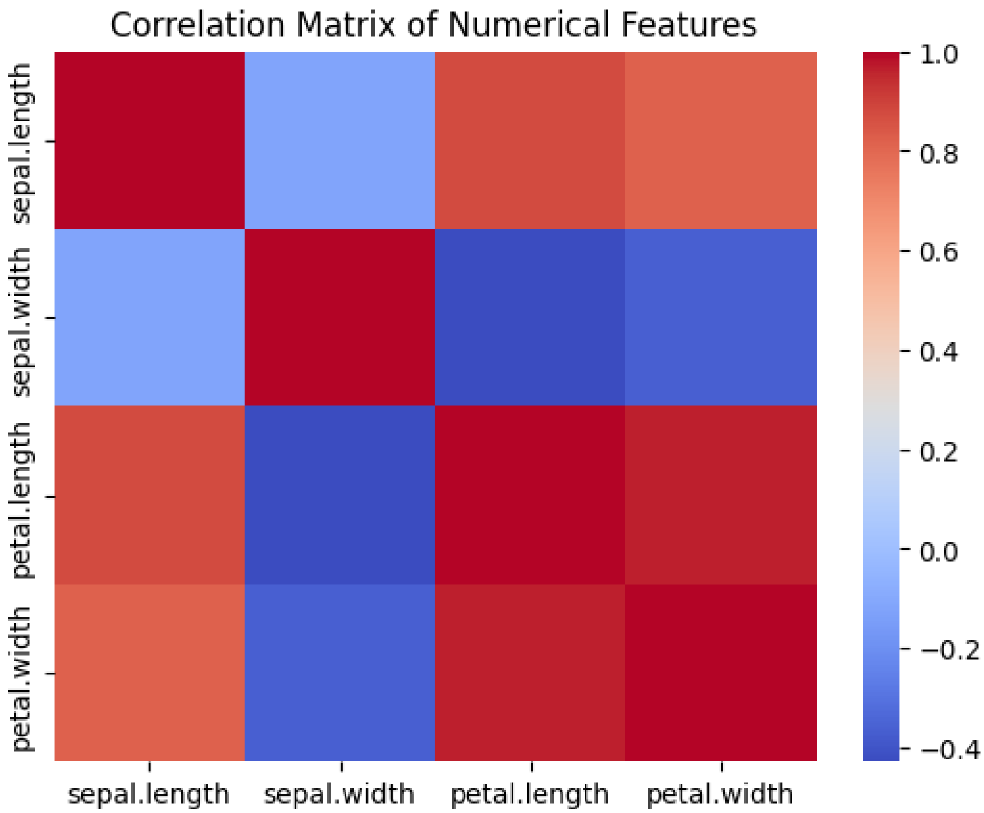


[125]:

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

df = pd.read_csv('iris.csv')

print("\nHead : \n",df.head())
print("\nInfo : \n",df.info())
print("\nDescribe : \n",df.describe())
print("\nNull Values : \n",df.isnull().sum())


correlation_matrix = df.drop(columns=['species']).corr()

sns.heatmap(data=correlation_matrix,  cmap="coolwarm")
plt.title('Correlation Matrix of Numerical Features')
```

```python
plt.show()
```

```
Head :
   sepal.length  sepal.width  petal.length  petal.width species
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB

Info :
 None

Describe :
        sepal.length  sepal.width  petal.length  petal.width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Null Values :
 sepal.length    0
sepal.width     0
petal.length    0
petal.width     0
species         0
dtype: int64
```



[131]:
```python
from scipy.stats import ttest_1samp

# Sample data: weights of apples
sample_weights = [152, 149, 153, 151, 148, 150, 155, 147]

# Known population mean
population_mean = 150

t_stats,p_value = ttest_1samp(sample_weights, population_mean)

# Output the results
print("T-statistic:", t_statistic)
print("P-value:", p_value)

threshold = 0.05

if p_value<=threshold:
    print("Reject the null hypothesis: The sample mean significantly differs␣
 ↪from the population mean.")
else:
```

```
    print("Fail to reject the null hypothesis: The sample mean does not␣
↪significantly differ from the population mean.")
```

```
T-statistic: 0.6622661785325219
P-value: 0.5289936281086355
Fail to reject the null hypothesis: The sample mean does not significantly
differ from the population mean.
```

[156]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report

df = pd.read_csv('heart.csv')

print("Original DataFrame: \n",df.head())
```

```
Original DataFrame:
    age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0   63       1   3       145   233    1        0      150      0      2.3
1   37       1   2       130   250    0        1      187      0      3.5
2   41       0   1       130   204    0        0      172      0      1.4
3   56       1   1       120   236    0        1      178      0      0.8
4   57       0   0       120   354    0        1      163      1      0.6

   slope  ca  thal  target
0      0   0     1       1
1      0   0     2       1
2      2   0     2       1
3      2   0     2       1
4      2   0     2       1
```

[157]:
```
from scipy.stats import zscore
print(df.isnull().sum())
print(df.shape)

z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))

outliers = z_scores>3
df_cleaned = df[~outliers.any(axis=1)]
print("\nCleaned Data: \n", df_cleaned.shape)

x = df_cleaned.drop('target', axis=1)
y = df_cleaned['target']

X_train, X_test, Y_train, Y_test=train_test_split(x,y)
```

```python
model = LogisticRegression(max_iter=1000)

model.fit(X_train, Y_train)
```

```
age         0
gender      0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
(303, 14)

Cleaned Data:
 (287, 14)
```

[157]:

```
LogisticRegression(max_iter=1000)
```

[158]:

```python
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", accuracy)

# Display the classification report
print("Classification Report:\n", classification_report(Y_test, y_pred))

# Display the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(Y_test, y_pred))
```

```
Accuracy: 0.8333333333333334
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.71      0.79        31
           1       0.81      0.93      0.86        41

    accuracy                           0.83        72
   macro avg       0.84      0.82      0.82        72
```

```
weighted avg       0.84      0.83      0.83        72

Confusion Matrix:
 [[22  9]
 [ 3 38]]
```

[170]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("housing.csv")

# Display the first few rows of the dataset
print(df.head())

# Display basic info to understand the dataset
print(df.info())
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
```

```
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
```

[187]:
```python
# Check for missing values
print(df.isnull().sum())

df["total_bedrooms"] = df["total_bedrooms"].fillna(df["total_bedrooms"].mean())

# Separate features (X) and target variable (y)
X = df.drop(["median_house_value","ocean_proximity"], axis=1)  # Replace␣
 ↪"price" with the actual column name for the target
Y = df["median_house_value"]

X_train, x_test, Y_train, y_test = train_test_split(X,Y, test_size=0.2)

model = LinearRegression()

model.fit(X_train,Y_train)
```

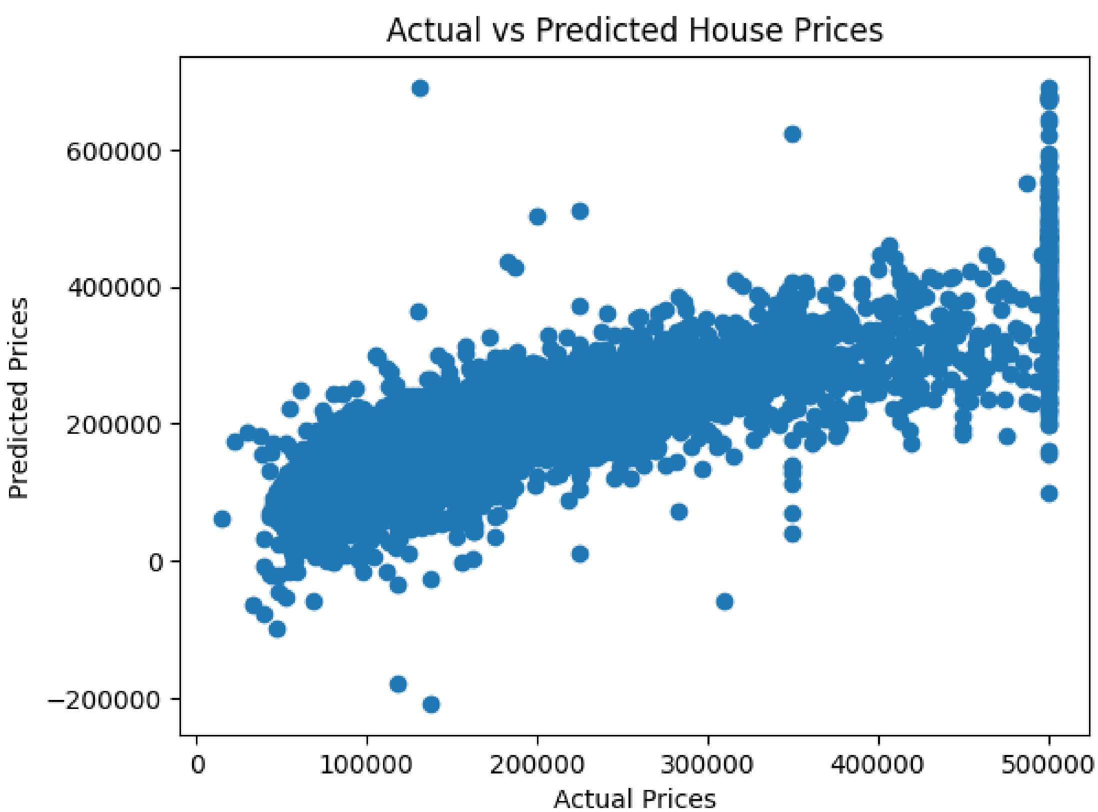
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

[187]: LinearRegression()

[188]:
```python
y_pred = model.predict(x_test)

# Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared (R²):", r2)

# Plot actual vs predicted house prices
plt.scatter(y_test, y_pred)
```

```
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
plt.show()
```

```
Mean Squared Error (MSE): 4886893640.625827
R-squared (R²): 0.6318032491833379
```



[191]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
import seaborn as sns

# Load the dataset
df = pd.read_csv("Admission_Predict.csv")

# Display the first few rows of the dataset
print(df.head())
```

```
# Display basic info about the dataset
print(df.info())
```

```
   Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0           1        337          118                  4  4.5   4.5  9.65
1           2        324          107                  4  4.0   4.5  8.87
2           3        316          104                  3  3.0   3.5  8.00
3           4        322          110                  3  3.5   2.5  8.67
4           5        314          103                  2  2.0   3.0  8.21

   Research  Chance of Admit
0         1             0.92
1         1             0.76
2         1             0.72
3         1             0.80
4         0             0.65
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         400 non-null    int64
 1   GRE Score          400 non-null    int64
 2   TOEFL Score        400 non-null    int64
 3   University Rating  400 non-null    int64
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
None
```

[214]:

```
from scipy.stats import zscore

# Binning the target variable into two classes: 'Admitted' and 'Not Admitted'
df['Admit Category'] = pd.cut(df['Chance of Admit '], bins=[0, 0.5, 1],␣
 ↪labels=['Not Admitted', 'Admitted'])

# Check for missing values
print(df.isnull().sum())

# Check the shape of the data
print(df.shape)
```

```
# Remove outliers using Z-score
z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))
outliers = z_scores > 3
df_cleaned = df[~outliers.any(axis=1)]
print("\nCleaned Data: \n", df_cleaned.shape)

X = df_cleaned.drop(["Chance of Admit ", "Admit Category"], axis=1)
y = df_cleaned["Admit Category"]  # Categorical target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize the Decision Tree Classifier model
model = DecisionTreeClassifier()

# Train the model
model.fit(X_train, y_train)
```

```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
Admit Category       0
dtype: int64
(400, 10)

Cleaned Data:
 (399, 10)
```

[214]:
```
DecisionTreeClassifier()
```

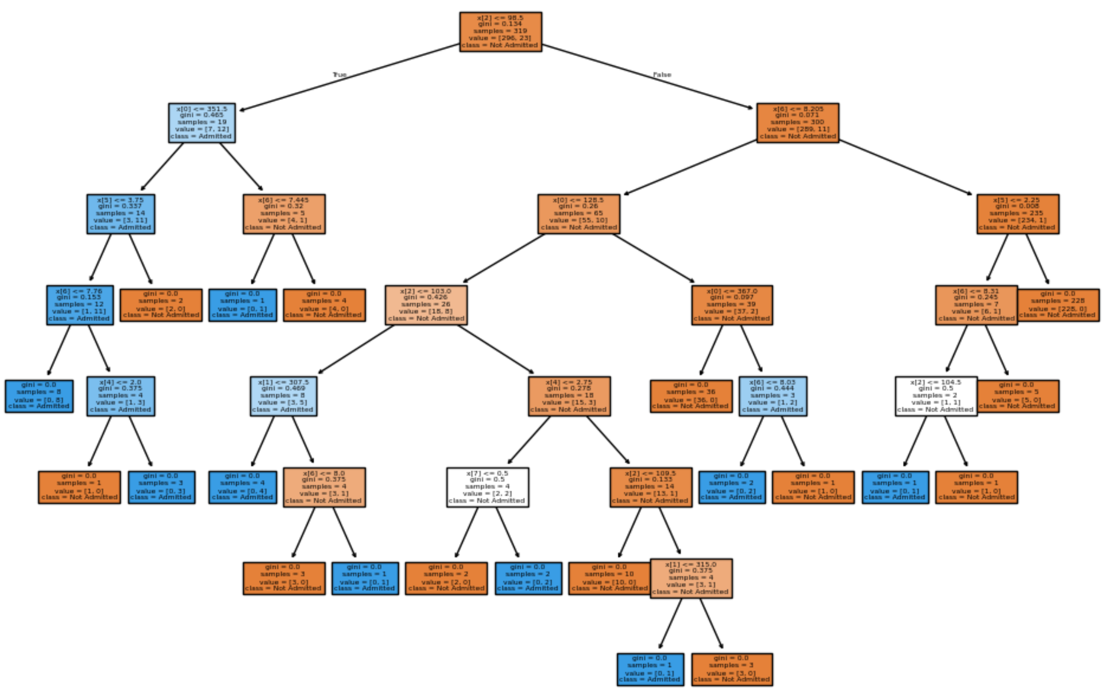
[215]:
```
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display the classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Display the confusion matrix
```

```python
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9125
Classification Report:
               precision    recall  f1-score   support

    Admitted       0.94      0.96      0.95        69
Not Admitted       0.70      0.64      0.67        11

    accuracy                           0.91        80
   macro avg       0.82      0.80      0.81        80
weighted avg       0.91      0.91      0.91        80

Confusion Matrix:
 [[66  3]
 [ 4  7]]
```

[216]:

```python
from sklearn.tree import plot_tree

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True, class_names=["Not Admitted", "Admitted"])
plt.show()
```



```
[ ]:
```