Phase 1 Project: Data Science

Project Details

- **Start Date:** 1 November 2023
- **Deadline:** 15 November 2023

Project Steps

1. Accessing Public API and Creating a Dataset

Used Python with requests library to access a public API (e.g., OpenWeatherMap API) and saved the obtained data as a CSV file.

In [1]:

```python
import pandas as pd
import requests

# URL for JSONPlaceholder API to fetch dummy data (example: posts)
api_url = 'https://jsonplaceholder.typicode.com/posts'

# Make a GET request to the API
response = requests.get(api_url)

if response.status_code == 200:
    data = response.json()

    # Creating a Pandas DataFrame from the retrieved data
    df = pd.DataFrame(data)

    # Saving the DataFrame to a CSV file
    df.to_csv('data.csv', index=False)
    print("CSV file 'data.csv' has been created with the fetched data.")
else:
    print("Failed to retrieve data. Status code:", response.status_code)
```

```
CSV file 'data.csv' has been created with the fetched data.
```

In [2]:

```python
print(df.head())
```

```
   userId  id                                              title  \
0       1   1  sunt aut facere repellat provident occaecati e...   
1       1   2                                       qui est esse   
2       1   3  ea molestias quasi exercitationem repellat qui...   
3       1   4                               eum et est occaecati   
4       1   5                                 nesciunt quas odio   

                                                body  
0  quia et suscipit\nsuscipit recusandae consequu...  
1  est rerum tempore vitae\nsequi sint nihil repr...  
2  et iusto sed quo iure\nvoluptatem occaecati om...  
3  ullam et saepe reiciendis voluptatem adipisci\...  
4  repudiandae veniam quaerat sunt sed\nalias aut...  
```

Data Cleaning with Pandas

This Python code snippet demonstrates the process of cleaning a dataset using Pandas, focusing on handling missing values and removing outliers.

Importing Necessary Libraries

The code begins by importing the required Python libraries, mainly `pandas` for data manipulation and handling.

Loading the Dataset

It retrieves the dataset from the specified URL using `pd.read_csv()` and stores it in a Pandas DataFrame.

Displaying Basic Information

It prints basic information about the dataset using `df.info()` , which includes column names, data types, and missing values.

Handling Missing Values

Missing values in numerical columns are filled with the mean of the respective column using `df.fillna(df.mean(), inplace=True)` . This replaces NaN values with the mean.

Removing Outliers

A common approach to removing outliers is applied. The code uses a threshold (in this case, 3 standard deviations from the mean) to filter out rows where a specific column's values are considered outliers.

The mean and standard deviation of the column are calculated, and then rows where the column value is beyond the threshold are filtered using boolean indexing.

Displaying the Cleaned Dataset

The head of the cleaned dataset is printed to display the first few rows after handling missing values and removing outliers.

Saving the Cleaned Dataset

Finally, the cleaned dataset is saved to a new CSV file named `cleaned_data.csv` using `df.to_csv()` .

Please note: The code is a basic example. Adjustments are necessary based on the characteristics of the dataset and specific requirements for handling missing values and outliers.

In [3]:

```python
import pandas as pd

# Load the downloaded dataset
file_path = 'dataset - netflix1.csv'  # Replace with the actual file path
```

```
df = pd.read_csv(file_path)

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

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

# Handle missing values (example: replacing missing values in 'director' column wi
df['director'].fillna('Unknown', inplace=True)

# Convert date columns to datetime format
df['date_added'] = pd.to_datetime(df['date_added'])
df['release_year'] = pd.to_datetime(df['release_year'], format='%Y')  # Adjust for

# Clean up 'duration' column (example: extract numeric values)
df['duration'] = df['duration'].str.extract('(\d+)').astype(float)  # Extracting nu

# Display the head of the cleaned dataset
print(df.head())

# Save the cleaned dataset to a new CSV file
df.to_csv('cleaned_data.csv', index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8790 entries, 0 to 8789
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   show_id       8790 non-null   object
 1   type          8790 non-null   object
 2   title         8790 non-null   object
 3   director      8790 non-null   object
 4   country       8790 non-null   object
 5   date_added    8790 non-null   object
 6   release_year  8790 non-null   int64 
 7   rating        8790 non-null   object
 8   duration      8790 non-null   object
 9   listed_in     8790 non-null   object
dtypes: int64(1), object(9)
memory usage: 686.8+ KB
None
show_id         0
type            0
title           0
director        0
country         0
date_added      0
release_year    0
rating          0
duration        0
listed_in       0
dtype: int64
  show_id     type                             title         director  \
0      s1    Movie              Dick Johnson Is Dead  Kirsten Johnson   
1      s3  TV Show                         Ganglands  Julien Leclercq   
2      s6  TV Show                     Midnight Mass    Mike Flanagan   
3     s14    Movie  Confessions of an Invisible Girl    Bruno Garotti   
4      s8    Movie                           Sankofa     Haile Gerima   

         country date_added release_year rating  duration  \
0  United States 2021-09-25   2020-01-01  PG-13      90.0   
1         France 2021-09-24   2021-01-01  TV-MA       1.0   
2  United States 2021-09-24   2021-01-01  TV-MA       1.0   
3         Brazil 2021-09-22   2021-01-01  TV-PG      91.0   
4  United States 2021-09-24   1993-01-01  TV-MA     125.0   

                                           listed_in  
0                                      Documentaries  
1  Crime TV Shows, International TV Shows, TV Act...  
2                 TV Dramas, TV Horror, TV Mysteries  
3                 Children & Family Movies, Comedies  
4   Dramas, Independent Movies, International Movies  
```

In [4]:

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

# Identify and remove outliers from the 'duration' column
column_name = 'duration'  # Replace with the column name containing outliers

# Calculate mean and standard deviation
mean = df[column_name].mean()
std_dev = df[column_name].std()

# Define a threshold (e.g., 3 standard deviations from the mean)
threshold = 3 * std_dev

# Filter out rows where the column value is considered an outlier
```

```
df = df[abs(df[column_name] - mean) < threshold]

# Display the head of the cleaned dataset after outlier removal
print(df.head())

# Save the cleaned dataset to a new CSV file
df.to_csv('cleaned_data_without_outliers.csv', index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8790 entries, 0 to 8789
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   show_id       8790 non-null   object        
 1   type          8790 non-null   object        
 2   title         8790 non-null   object        
 3   director      8790 non-null   object        
 4   country       8790 non-null   object        
 5   date_added    8790 non-null   datetime64[ns]
 6   release_year  8790 non-null   datetime64[ns]
 7   rating        8790 non-null   object        
 8   duration      8790 non-null   float64       
 9   listed_in     8790 non-null   object        
dtypes: datetime64[ns](2), float64(1), object(7)
memory usage: 686.8+ KB
None
  show_id     type                             title         director  \
0      s1    Movie              Dick Johnson Is Dead  Kirsten Johnson   
1      s3  TV Show                         Ganglands  Julien Leclercq   
2      s6  TV Show                     Midnight Mass    Mike Flanagan   
3     s14    Movie  Confessions of an Invisible Girl    Bruno Garotti   
4      s8    Movie                           Sankofa     Haile Gerima   

         country date_added release_year rating  duration  \
0  United States 2021-09-25   2020-01-01  PG-13      90.0   
1         France 2021-09-24   2021-01-01  TV-MA       1.0   
2  United States 2021-09-24   2021-01-01  TV-MA       1.0   
3         Brazil 2021-09-22   2021-01-01  TV-PG      91.0   
4  United States 2021-09-24   1993-01-01  TV-MA     125.0   

                                           listed_in  
0                                      Documentaries  
1  Crime TV Shows, International TV Shows, TV Act...  
2                 TV Dramas, TV Horror, TV Mysteries  
3                 Children & Family Movies, Comedies  
4   Dramas, Independent Movies, International Movies  
```

In [5]:

```
# outilers as been removed(8790-8781=9)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8781 entries, 0 to 8789
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   show_id       8781 non-null   object
 1   type          8781 non-null   object
 2   title         8781 non-null   object
 3   director      8781 non-null   object
 4   country       8781 non-null   object
 5   date_added    8781 non-null   datetime64[ns]
 6   release_year  8781 non-null   datetime64[ns]
 7   rating        8781 non-null   object
 8   duration      8781 non-null   float64
 9   listed_in     8781 non-null   object
dtypes: datetime64[ns](2), float64(1), object(7)
memory usage: 754.6+ KB
None
```

Phase -2

Analysis of Dataset and Graph Generation

Project Details

- **Start Date:** 16 November 2023
- **Deadline:** 1 December 2023
- **Task:** Analyzing the provided dataset and creating graphs using Seaborn and Matplotlib.

Dataset

- The dataset used for analysis can be accessed here.

Methodology

1. **Loading the Dataset:**

    - The dataset was imported into a Pandas DataFrame for analysis using Python.
    - Python libraries used: Pandas, Seaborn, Matplotlib.

2. **Data Exploration:**

    - Performed data exploration to understand the structure, columns, and types of data present in the dataset.
    - Utilized commands like `df.head()` , `df.info()` , and `df.describe()` to gain insights into the data.

3. **Data Visualization:**

    - Utilized Seaborn and Matplotlib to generate visualizations and insights from the dataset.
    - Created various types of plots such as:

        - Histograms
        - Scatter Plots
        - Bar Charts
        - Pair Plots

4. **Findings and Analysis:**

    - Interpreted the visualizations and derived insights from the generated graphs.
    - Noted any trends, patterns, or anomalies observed in the data.

Code Snippets

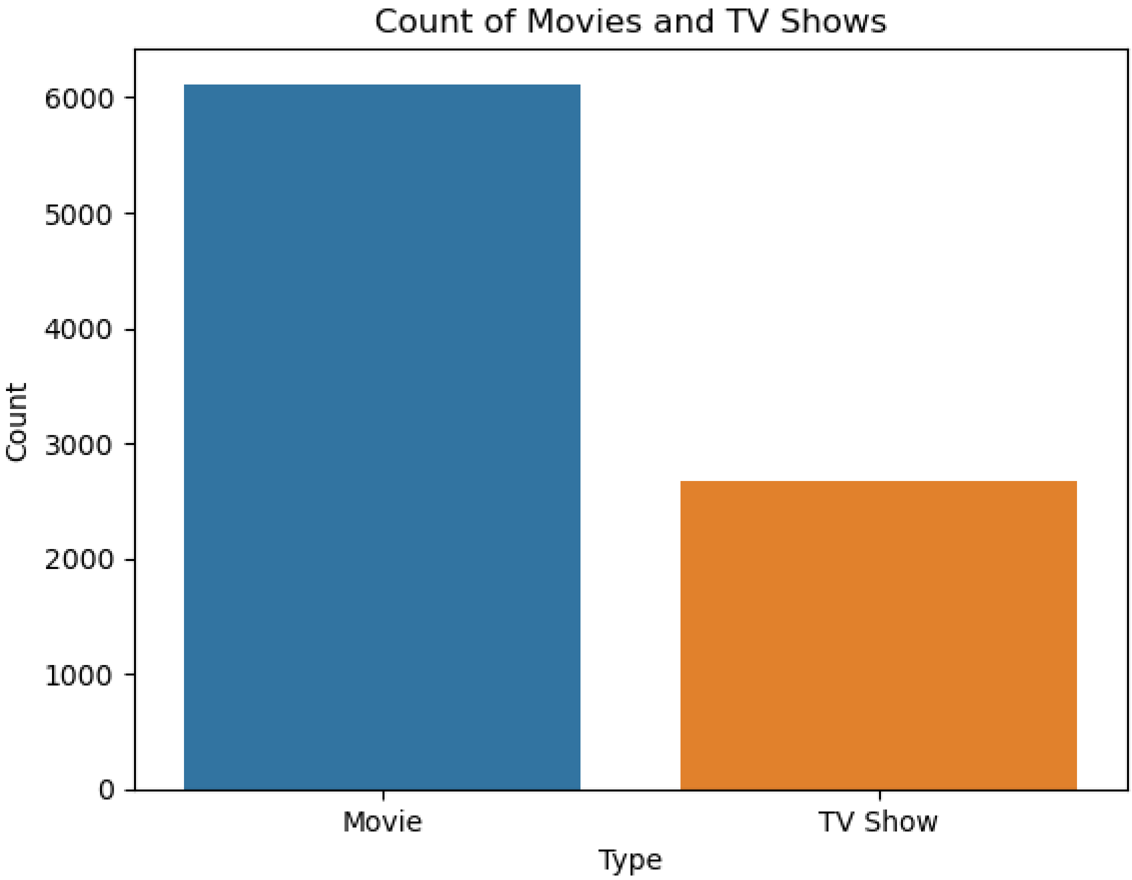
Example code used for generating graphs:

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

# Load the dataset

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

# Creating a histogram

sns.histplot(data=df, x='column_name', kde=True) plt.title('Histogram of column_name') plt.show()

In [6]:

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

print(df.info())

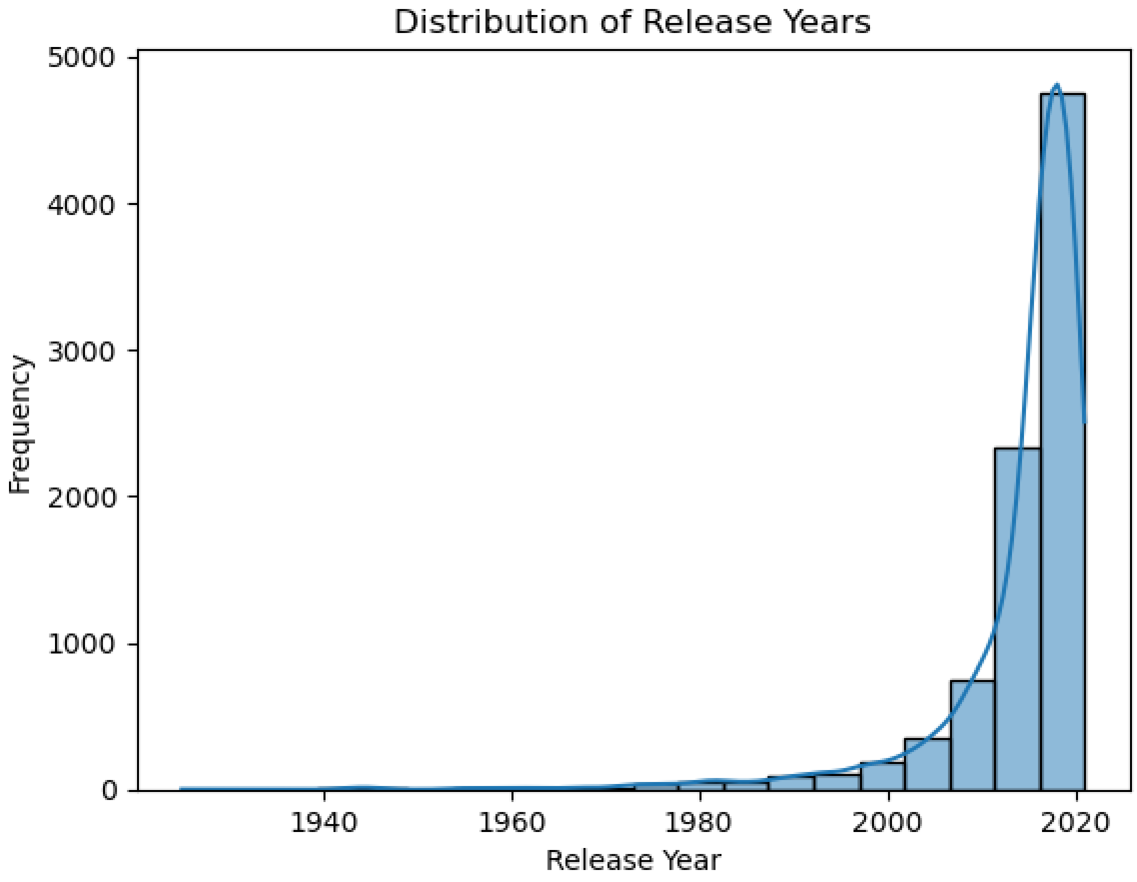
# Example visualizations
# Plotting count of 'type' (Movie vs TV Show)
sns.countplot(x='type', data=df)
plt.title('Count of Movies and TV Shows')
plt.xlabel('Type')
plt.ylabel('Count')
plt.show()
```
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8781 entries, 0 to 8789
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   show_id       8781 non-null   object        
 1   type          8781 non-null   object        
 2   title         8781 non-null   object        
 3   director      8781 non-null   object        
 4   country       8781 non-null   object        
 5   date_added    8781 non-null   datetime64[ns]
 6   release_year  8781 non-null   datetime64[ns]
 7   rating        8781 non-null   object        
 8   duration      8781 non-null   float64       
 9   listed_in     8781 non-null   object        
dtypes: datetime64[ns](2), float64(1), object(7)
memory usage: 754.6+ KB
None
```

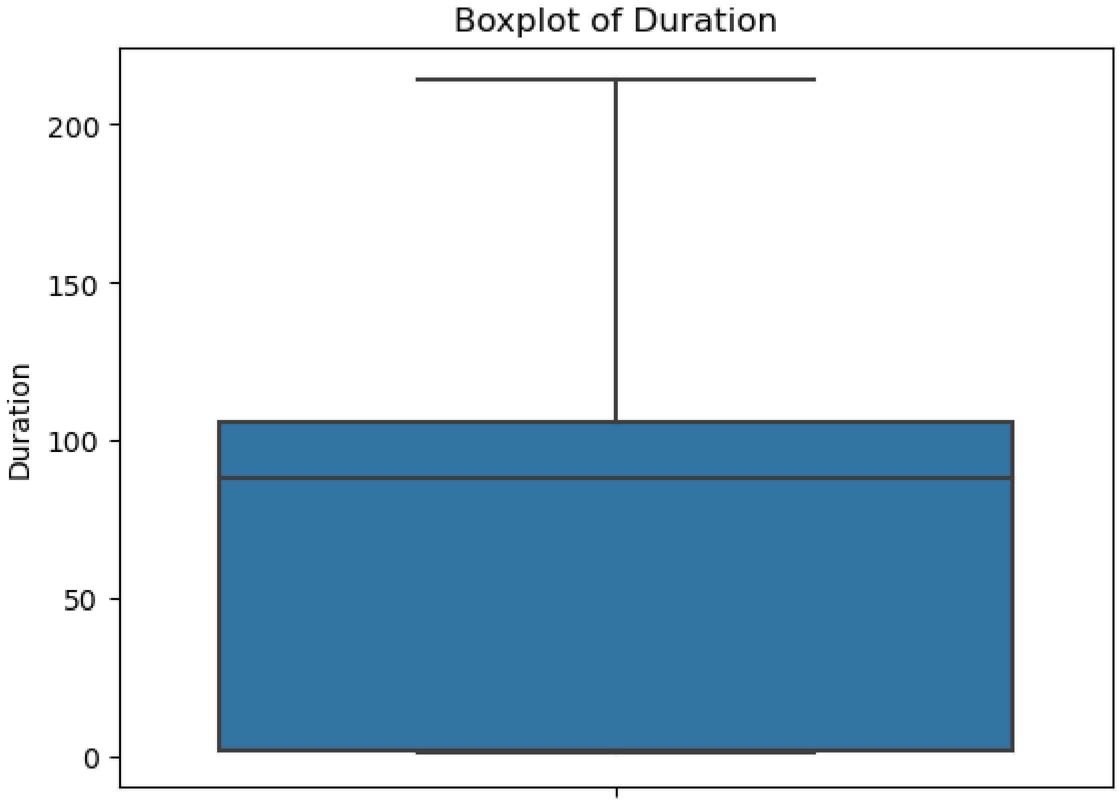


In [7]:

```
# Creating a histogram of 'release_year'
sns.histplot(data=df, x='release_year', bins=20, kde=True)
plt.title('Distribution of Release Years')
plt.xlabel('Release Year')
plt.ylabel('Frequency')
plt.show()
```



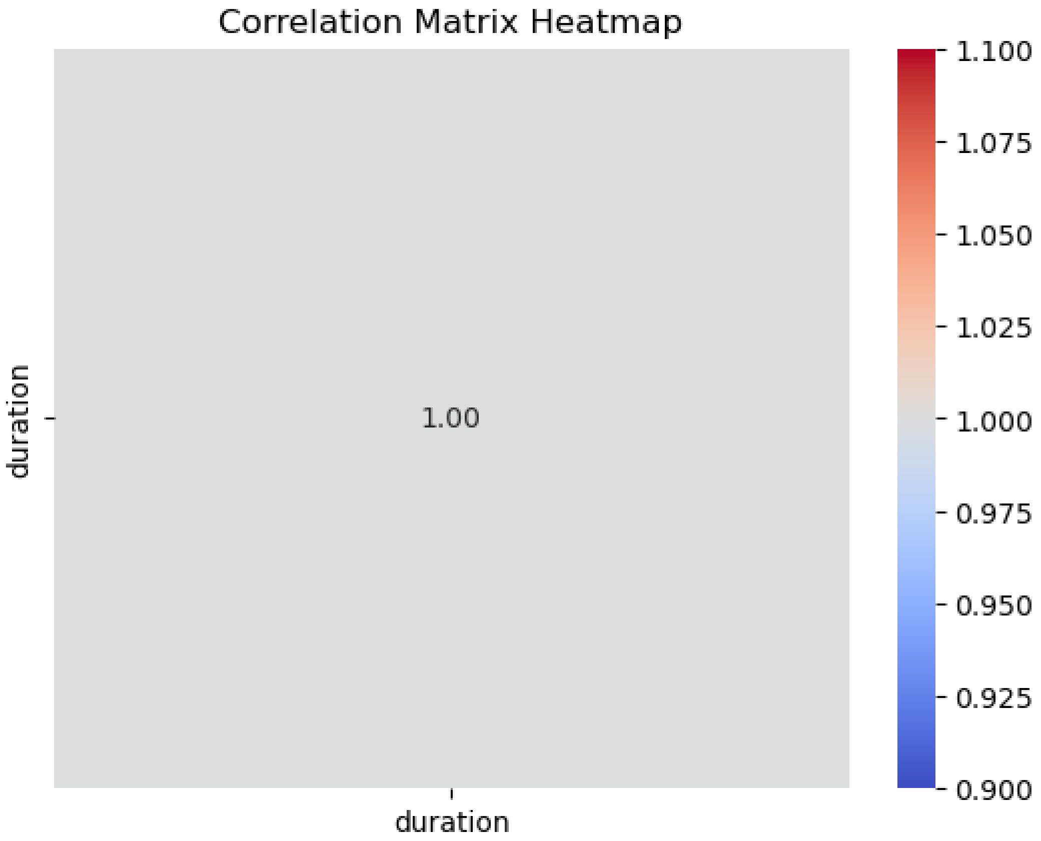
```
In [8]: # Create a boxplot for 'duration' to detect outliers
        sns.boxplot(data=df, y='duration')
        plt.title('Boxplot of Duration')
        plt.ylabel('Duration')
        plt.show()
```



```
In [9]: # Drop non-numeric or irrelevant columns for correlation
        non_numeric_cols = ['show_id']  # Adjust with other non-numeric columns if needed
```

```
df_numeric = df.select_dtypes(include=['int64', 'float64'])

# Display basic information about the remaining numeric columns
print(df_numeric.info())

# Correlation matrix heatmap for numeric columns
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8781 entries, 0 to 8789
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   duration  8781 non-null   float64
dtypes: float64(1)
memory usage: 137.2 KB
None
```



Linear Regression Using Provided Datasets

Objective

The goal is to demonstrate the process of training a simple linear regression model using the provided datasets. The training dataset consists of 'x' and 'y' values, and the test dataset contains 'x' values for which predictions need to be made.

Steps

Step 1: Load the Training Data

- Utilize the Pandas library to fetch data from the provided Google Sheets URLs for the training dataset.
- Display the initial rows to understand the data structure.

```python import pandas as pd

train_data_url = 'Training_Data_Google_Sheets_URL' train_data = pd.read_csv(train_data_url) print(train_data.head())

In [10]:

~~~
import pandas as pd

# Load the training dataset
train_data_url = 'https://docs.google.com/spreadsheets/d/e/2PACX-1vRTK2NvcndgPX41C
train_data = pd.read_csv(train_data_url)

# Display the first few rows of the training data
print(train_data.head())
~~~

~~~
      x          y
0  24.0  21.549452
1  50.0  47.464463
2  15.0  17.218656
3  38.0  36.586398
4  87.0  87.288984
~~~

In [14]:

~~~
# Check for NaN values in the 'y' column of the training data
nan_values = train_data['y'].isnull().sum()
print("Number of NaN values in 'y' column:", nan_values)

train_data = train_data.dropna(subset=['y'])

train_data['y'].fillna(train_data['y'].mean(), inplace=True)
~~~

~~~
Number of NaN values in 'y' column: 0
~~~

## Step 2: Train a Simple Linear Regression Model

Use a machine learning library like Scikit-learn to create and train a linear regression model.

In [15]:

~~~
from sklearn.linear_model import LinearRegression

# Create the linear regression model
model = LinearRegression()

# Fit the model using the training data
X_train = train_data[['x']]  # Features
y_train = train_data['y']    # Target variable
model.fit(X_train, y_train)
~~~

Out[15]:

~~~
▾ LinearRegression
LinearRegression()
~~~

## Step 3: Load the Test Data and Make Predictions

Repeat a similar process for the test dataset.
```

In [16]:

```
# Load the test dataset
test_data_url = 'https://docs.google.com/spreadsheets/d/e/2PACX-1vRyvZ7lknwiSghK9a
test_data = pd.read_csv(test_data_url)

# Display the first few rows of the test data
print(test_data.head())

# Perform predictions using the trained model
X_test = test_data[['x']]  # Features
predictions = model.predict(X_test)

# Display the predictions
print(predictions)
```

```
    x          y
0  77  79.775152
1  21  23.177279
2  22  25.609262
3  20  17.857388
4  36  41.849864
[76.94327594 20.90651855 21.90717494 19.90586217 35.91636428 14.90258026
 61.93343021 94.95509081 19.90586217  4.89601644  3.89536006 18.90520579
 95.95574719 61.93343021 35.91636428 14.90258026 64.93539936 13.90192388
 86.94983976 68.93802488 88.95115252 50.92621001 88.95115252 26.91045685
 96.95640358 57.93080468 78.9445887  20.90651855 92.95377805 26.91045685
 98.95771634 30.91308237 32.91439514 79.94524508 27.91111323 46.92358448
 52.92752277 68.93802488 27.91111323 32.91439514 90.95246528 70.93933765
 49.92555363 75.94261956  3.89536006 36.91702066 69.93868127 67.9373685
 39.91898981 34.9157079  93.95443443 87.95049614 51.92686639 30.91308237
 58.93146107 -0.10726546 38.91833343 63.93474297 68.93802488 56.9301483
 12.9012675  71.93999403 75.94261956 60.93277383 81.94655785 17.90454941
 40.91964619 49.92555363 54.92883554 12.9012675  45.9229281  12.9012675
 78.9445887  52.92752277 14.90258026 27.91111323 80.94590147 68.93802488
 51.92686639 83.94787061 67.9373685  26.91045685 55.92949192 47.92424086
 39.91898981 38.91833343 81.94655785 99.95837272 58.93146107 42.92095896
 66.93671212 37.91767705 62.93408659 90.95246528 59.93211745 13.90192388
 20.90651855 86.94983976 72.94065041 31.91373876  1.8940473  81.94655785
 18.90520579 73.94130679 41.92030257 11.90061112  0.89339092 89.9518089
 88.95115252 -0.10726546 40.91964619 15.90323665 93.95443443 96.95640358
 65.93605574 23.9084877  16.90389303 89.9518089  12.9012675  -0.10726546
 63.93474297 95.95574719 97.95705996 11.90061112 40.91964619 46.92358448
 77.94393232 19.90586217 88.95115252 28.91176961 63.93474297 74.94196317
 11.90061112 24.90914408 27.91111323 29.91242599 64.93539936 58.93146107
 63.93474297 52.92752277 70.93933765 96.95640358 72.94065041  8.89864197
 11.90061112 62.93408659 98.95771634 59.93211745 34.9157079   1.8940473
 59.93211745 31.91373876 93.95443443 83.94787061 62.93408659 21.90717494
 80.94590147 92.95377805 32.91439514  6.89732921 41.92030257 45.9229281
 53.92817916 15.90323665 48.92489725 42.92095896 94.95509081 65.93605574
 20.90651855 34.9157079  79.94524508 36.91702066 53.92817916 55.92949192
  0.89339092 31.91373876 57.93080468 31.91373876 45.9229281  71.93999403
 16.90389303 96.95640358 92.95377805 90.95246528 36.91702066  3.89536006
 53.92817916 50.92621001 26.91045685 45.9229281  91.95312167 72.94065041
 76.94327594 90.95246528 60.93277383 98.95771634  3.89536006 71.93999403
 18.90520579 56.9301483  77.94393232 25.90980046 73.94130679 89.9518089
 65.93605574 12.9012675  39.91898981 76.94327594 66.93671212 74.94196317
 22.90783132 44.92227172 58.93146107 43.92161534 22.90783132 54.92883554
 54.92883554 94.95509081 11.90061112  3.89536006  6.89732921 99.95837272
 47.92424086 41.92030257 95.95574719 38.91833343 99.95837272 86.94983976
 13.90192388 13.90192388 36.91702066  4.89601644 87.95049614 90.95246528
 64.93539936 73.94130679 55.92949192 15.90323665  4.89601644 27.91111323
 91.95312167 45.9229281  53.92817916 38.91833343 43.92161534 30.91308237
 67.9373685  85.94918338 89.9518089  37.91767705 20.90651855 94.95509081
 55.92949192 59.93211745 64.93539936 77.94393232 88.95115252  5.89667283
 66.93671212 35.91636428 15.90323665 99.95837272 44.92227172 72.94065041
 56.9301483  19.90586217 75.94261956 33.91505152 54.92883554 71.93999403
 54.92883554  7.89798559 55.92949192 71.93999403 57.93080468  5.89667283
 95.95574719 22.90783132 57.93080468 22.90783132 18.90520579 24.90914408
 63.93474297 20.90651855 58.93146107 18.90520579 15.90323665 41.92030257
 42.92095896 60.93277383 91.95312167 10.89995474 40.91964619  0.89339092
  7.89798559 70.93933765 45.9229281  54.92883554 61.93343021 46.92358448]
```

```
In [ ]:
```