EDA Lab

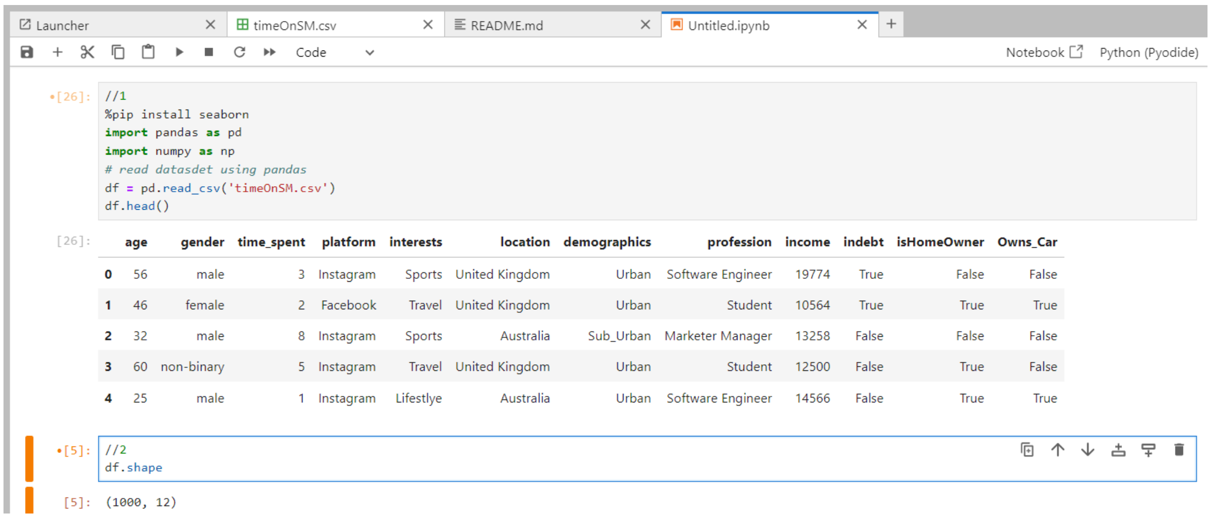
By L00158553

The dataset I used for my EDA lab was ‘Average time spent from a user on social media. This data shows how much a user spends time on their devices using social media.

The description for each column is as follows:

- **age:** The age of the user.
- **gender:** The gender identity of the user (Male, Female, Non-binary).
- **demographics:** The type of area the user resides in (Urban, Suburban, Rural).
- **interests:** The user's primary area of interest or hobby.
- **device_type:** The type of device used by the user (Mobile).
- **location:** The country of residence for the user.
- **platform:** The social media platform where the user spends time.
- **profession:** The user's occupation or professional status.
- **income:** The yearly income of the user.
- **indebt:** Indicates whether the user is in debt (True or False).
- **homeowner:** Indicates whether the user owns a home (True or False).
- **owns_cars:** Indicates whether the user owns cars (True or False).

This code installs the Seaborn library, imports the Pandas and NumPy libraries for data manipulation, reads the dataset named 'timeOnSM.csv' into a Data Frame using Pandas, and displays the first few rows of the dataset.



```
•[26]: //1
%pip install seaborn
import pandas as pd
import numpy as np
# read datasdet using pandas
df = pd.read_csv('timeOnSM.csv')
df.head()
```

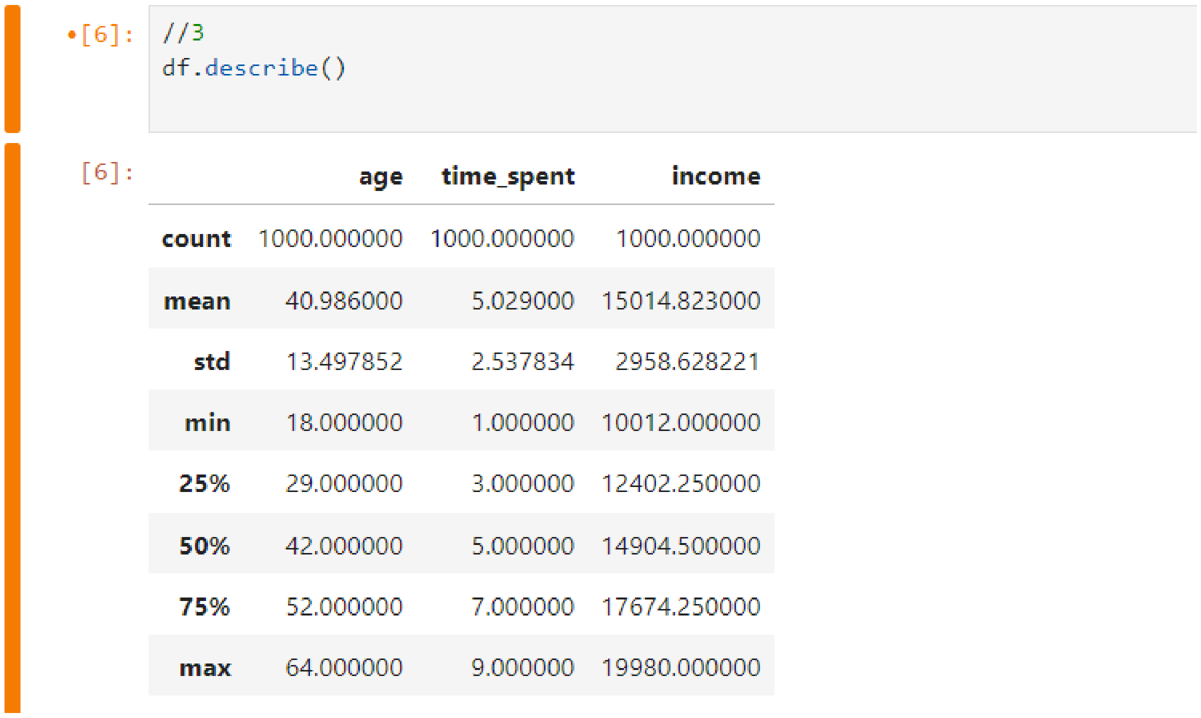
[26]:

	age	gender	time_spent	platform	interests	location	demographics	profession	income	indebt	isHomeOwner	Owns_Car
0	56	male	3	Instagram	Sports	United Kingdom	Urban	Software Engineer	19774	True	False	False
1	46	female	2	Facebook	Travel	United Kingdom	Urban	Student	10564	True	True	True
2	32	male	8	Instagram	Sports	Australia	Sub_Urban	Marketer Manager	13258	False	False	False
3	60	non-binary	5	Instagram	Travel	United Kingdom	Urban	Student	12500	False	True	False
4	25	male	1	Instagram	Lifestlye	Australia	Urban	Software Engineer	14566	False	True	True

```
•[5]: //2
df.shape
```

```
[5]: (1000, 12)
```

The code **`df.shape`** returns the dimensions of the dataset as a tuple, indicating the number of rows and columns, while **`df.describe()`** generates descriptive statistics that summarize the central tendency, dispersion, and shape of the dataset's numerical columns.

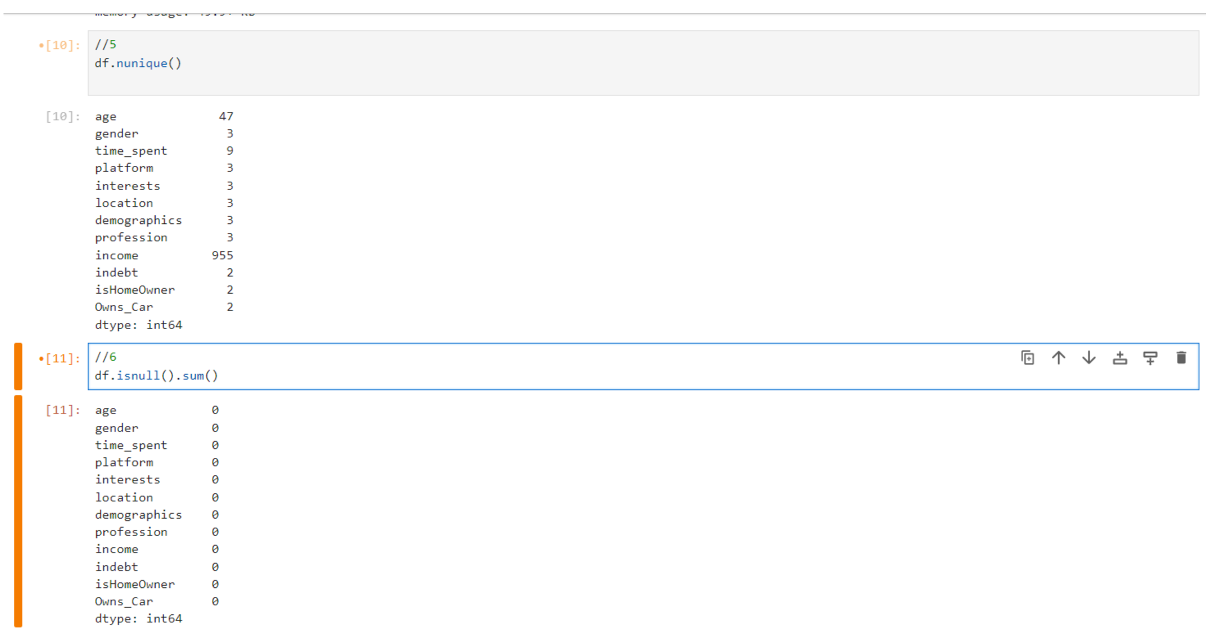


```
•[6]: //3
df.describe()
```

[6]:

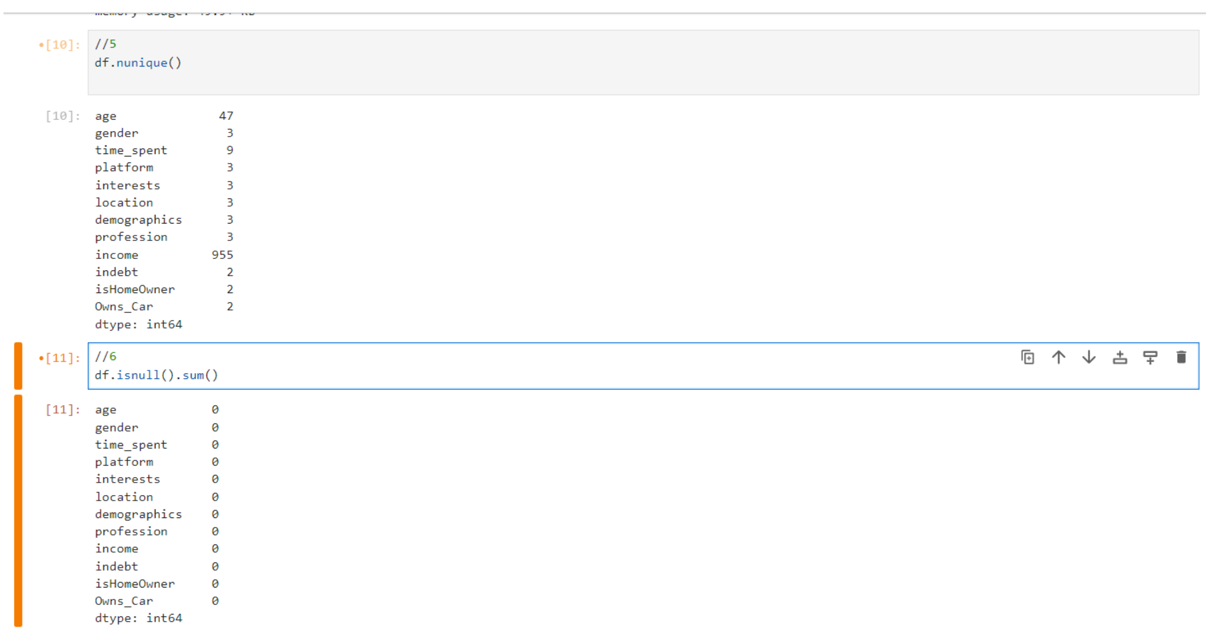
	age	time_spent	income
count	1000.000000	1000.000000	1000.000000
mean	40.986000	5.029000	15014.823000
std	13.497852	2.537834	2958.628221
min	18.000000	1.000000	10012.000000
25%	29.000000	3.000000	12402.250000
50%	42.000000	5.000000	14904.500000
75%	52.000000	7.000000	17674.250000
max	64.000000	9.000000	19980.000000

The code **df.info()** displays a concise summary of the Data Frame, including the number of non-null entries for each column, the column data types, and memory usage of the dataset.



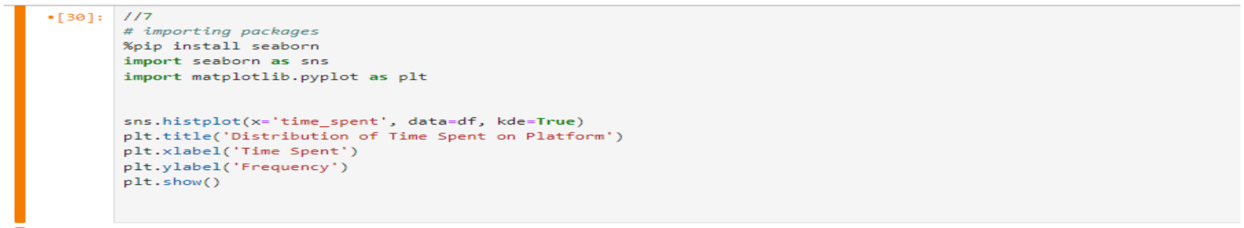
```
•[10]: //5
       df.nunique()

 [10]: age              47
       gender            3
       time_spent        9
       platform          3
       interests         3
       location          3
       demographics      3
       profession        3
       income          955
       indebt            2
       isHomeOwner       2
       Owns_Car          2
       dtype: int64

•[11]: //6
       df.isnull().sum()

 [11]: age             0
       gender          0
       time_spent      0
       platform        0
       interests       0
       location        0
       demographics    0
       profession      0
       income          0
       indebt          0
       isHomeOwner     0
       Owns_Car        0
       dtype: int64
```

The code **df.nunique()** calculates the number of unique values in each column of the Data Frame, helping to understand the diversity of data in each field and **df.isnull().sum()** calculates the number of missing values in each column of the Data Frame by checking for null entries and summing them up.

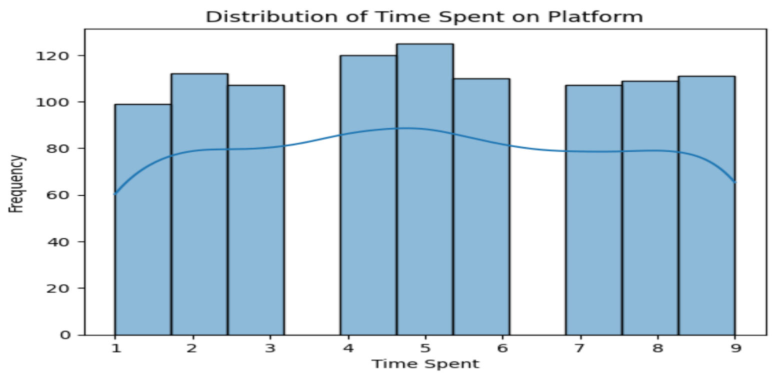


```
•[10]: //5
       df.nunique()

 [10]: age              47
       gender            3
       time_spent        9
       platform          3
       interests         3
       location          3
       demographics      3
       profession        3
       income          955
       indebt            2
       isHomeOwner       2
       Owns_Car          2
       dtype: int64

•[11]: //6
       df.isnull().sum()

 [11]: age             0
       gender          0
       time_spent      0
       platform        0
       interests       0
       location        0
       demographics    0
       profession      0
       income          0
       indebt          0
       isHomeOwner     0
       Owns_Car        0
       dtype: int64
```

This code block installs the Seaborn library, imports Seaborn and Matplotlib's pyplot for data visualization, and creates a histogram with a Kernel Density Estimate (KDE) overlay for the 'time_spent' column from the Data Frame **df**. It visualizes the distribution of time spent on a platform, labelling the x-axis as 'Time Spent', the y-axis as 'Frequency', and titles the plot 'Distribution of Time Spent on Platform'. Finally, it displays the plot.

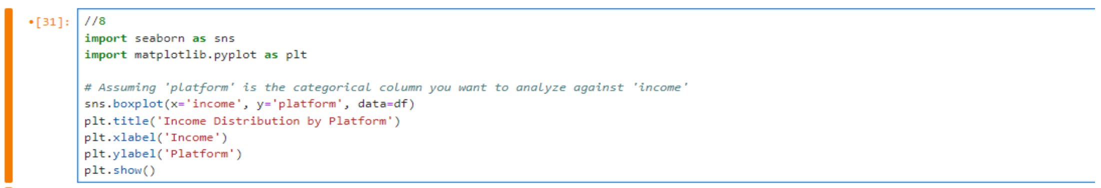


```
•[30]: //7
       # importing packages
       %pip install seaborn
       import seaborn as sns
       import matplotlib.pyplot as plt

       sns.histplot(x='time_spent', data=df, kde=True)
       plt.title('Distribution of Time Spent on Platform')
       plt.xlabel('Time Spent')
       plt.ylabel('Frequency')
       plt.show()
```

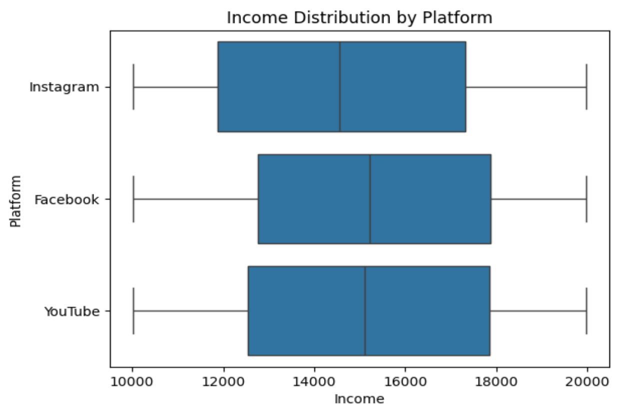


This code imports Seaborn and Matplotlib's pyplot for data visualization, and then uses Seaborn to create a boxplot that shows the distribution of 'income' across different categories in the 'platform' column of the Data Frame **df**. It sets 'Income' as the x-axis and 'Platform' as the y-axis, titles the plot 'Income Distribution by Platform', and displays the visualization. This boxplot can help identify differences in income distributions across various platforms, including medians, quartiles, and potential outliers.

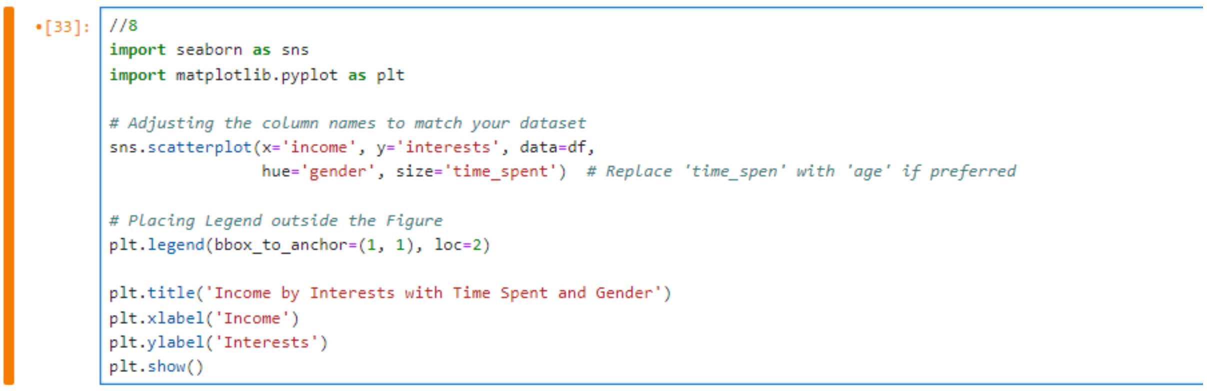
Time Spent



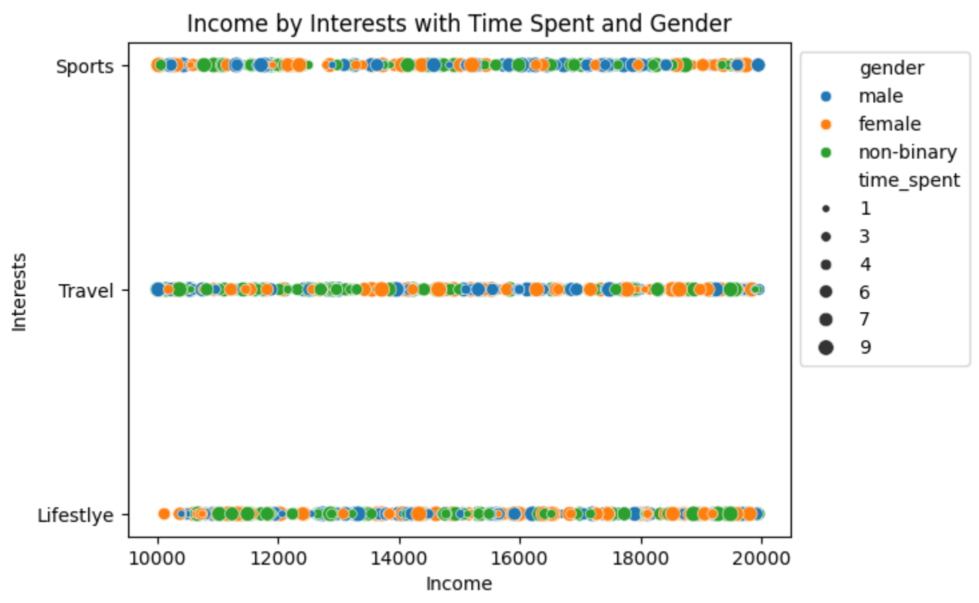
```
•[31]: //8
       import seaborn as sns
       import matplotlib.pyplot as plt

       # Assuming 'platform' is the categorical column you want to analyze against 'income'
       sns.boxplot(x='income', y='platform', data=df)
       plt.title('Income Distribution by Platform')
       plt.xlabel('Income')
       plt.ylabel('Platform')
       plt.show()
```



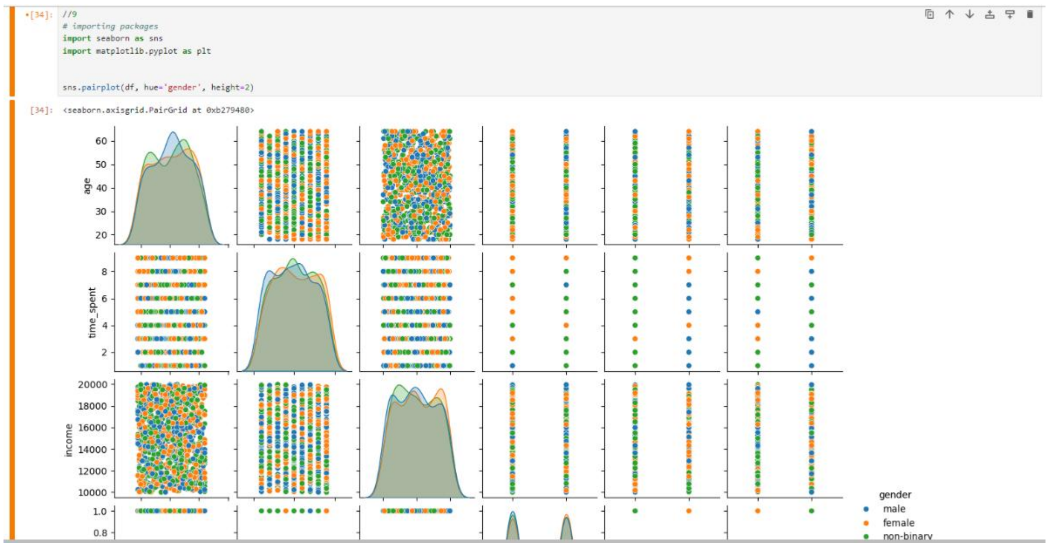
This code imports Seaborn and Matplotlib's pyplot for data visualization and creates a scatter plot using the Data Frame **df** to visualize the relationship between 'income' and 'interests'. It differentiates the data points by 'gender' using different colours (hue) and sizes the points based on 'time_spent' to indicate the amount of time spent on a platform. The legend, which helps to interpret the hues and sizes, is placed outside the figure for clarity. The plot is titled 'Income by Interests with Time Spent and Gender', with 'Income' on the x-axis and 'Interests' on the y-axis, and is displayed to the user.



```
•[33]: //8
import seaborn as sns
import matplotlib.pyplot as plt

# Adjusting the column names to match your dataset
sns.scatterplot(x='income', y='interests', data=df,
                hue='gender', size='time_spent')  # Replace 'time_spen' with 'age' if preferred

# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.title('Income by Interests with Time Spent and Gender')
plt.xlabel('Income')
plt.ylabel('Interests')
plt.show()
```



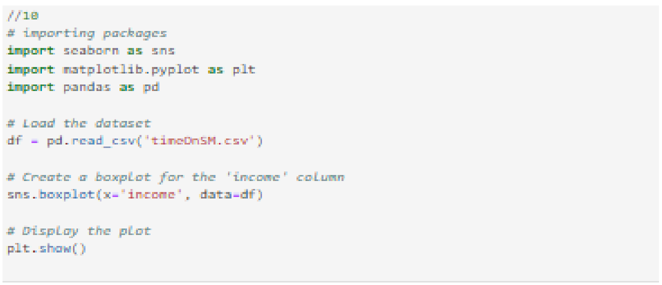
This code imports Seaborn and Matplotlib's pyplot for data visualization and uses Seaborn's **pairplot** function to create a grid of scatter plots. Each plot represents pairwise relationships in the DataFrame **df** across all numerical columns, with different colours (hue) representing different 'gender' categories. The **height=2** parameter specifies the height (in inches) of each subplot in the grid. This visualization helps in understanding the relationships between pairs of variables in the dataset, differentiated by gender.



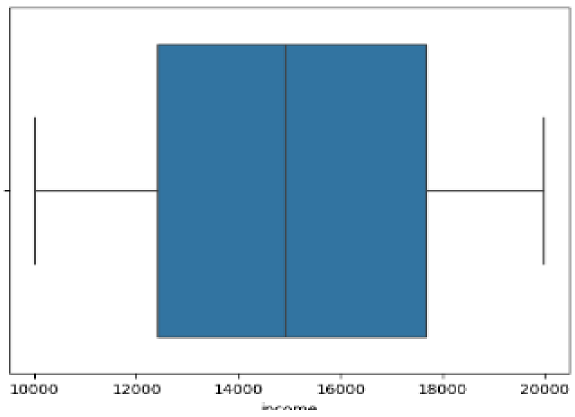
```
•[34]: //9
       # importing packages
       import seaborn as sns
       import matplotlib.pyplot as plt


       sns.pairplot(df, hue='gender', height=2)
```

```
[34]: <seaborn.axisgrid.PairGrid at 0xb279480>
```

This code loads the dataset named 'timeOnSM.csv' into a pandas Data Frame, then uses Seaborn to create a boxplot for the 'income' column of that Data Frame. The boxplot visualizes the distribution of income, highlighting the median, quartiles, and any outliers. Finally, it displays the plot using Matplotlib's **plt.show()** function, allowing you to see the spread and central tendency of income among the dataset's observations.

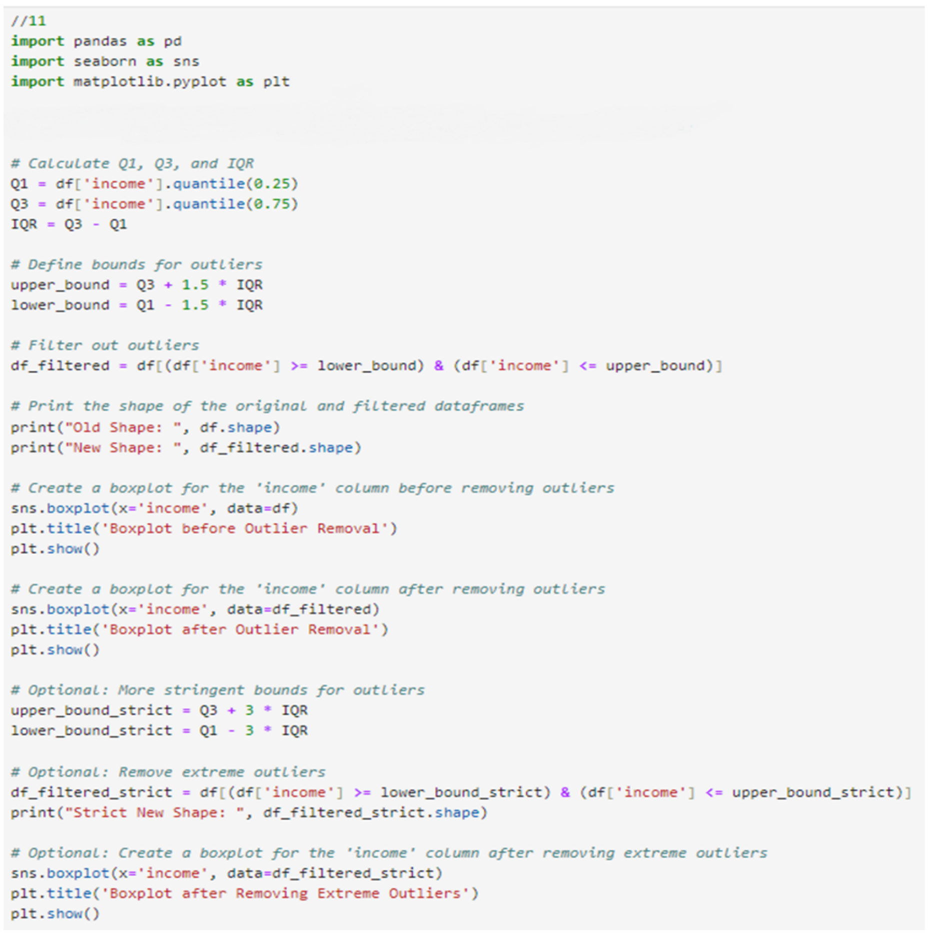


```
//10
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

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

# Create a boxplot for the 'income' column
sns.boxplot(x='income', data=df)

# Display the plot
plt.show()
```



This code performs outlier detection and removal on the 'income' column of a DataFrame **df**, using the Interquartile Range (IQR) method. It calculates the first and third quartiles (Q1, Q3), the IQR, and uses these to define upper and lower bounds for outliers. It then filters the Data Frame to remove these outliers and prints the shape of the original and filtered Data Frames to show how many entries were removed.

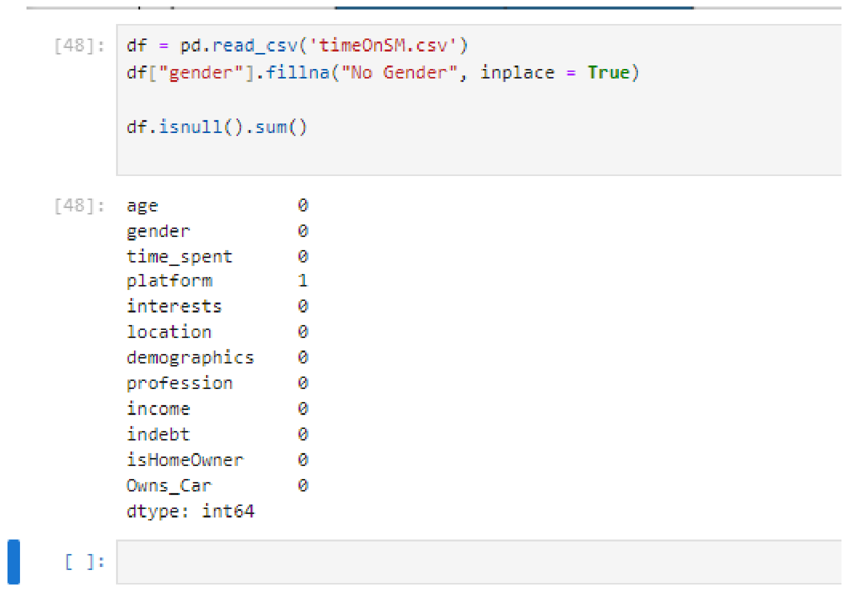
The code visualizes the effect of outlier removal by creating boxplots of the 'income' column before and after removing outliers. It also introduces a more stringent criterion for identifying extreme outliers (using 3 times the IQR) and displays the impact of removing these extreme values through another boxplot, highlighting the difference in data distribution and outlier presence.



```
//11
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# Calculate Q1, Q3, and IQR
Q1 = df['income'].quantile(0.25)
Q3 = df['income'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
upper_bound = Q3 + 1.5 * IQR
lower_bound = Q1 - 1.5 * IQR

# Filter out outliers
df_filtered = df[(df['income'] >= lower_bound) & (df['income'] <= upper_bound)]

# Print the shape of the original and filtered dataframes
print("Old Shape: ", df.shape)
print("New Shape: ", df_filtered.shape)

# Create a boxplot for the 'income' column before removing outliers
sns.boxplot(x='income', data=df)
plt.title('Boxplot before Outlier Removal')
plt.show()

# Create a boxplot for the 'income' column after removing outliers
sns.boxplot(x='income', data=df_filtered)
plt.title('Boxplot after Outlier Removal')
plt.show()

# Optional: More stringent bounds for outliers
upper_bound_strict = Q3 + 3 * IQR
lower_bound_strict = Q1 - 3 * IQR

# Optional: Remove extreme outliers
df_filtered_strict = df[(df['income'] >= lower_bound_strict) & (df['income'] <= upper_bound_strict)]
print("Strict New Shape: ", df_filtered_strict.shape)

# Optional: Create a boxplot for the 'income' column after removing extreme outliers
sns.boxplot(x='income', data=df_filtered_strict)
plt.title('Boxplot after Removing Extreme Outliers')
plt.show()
```

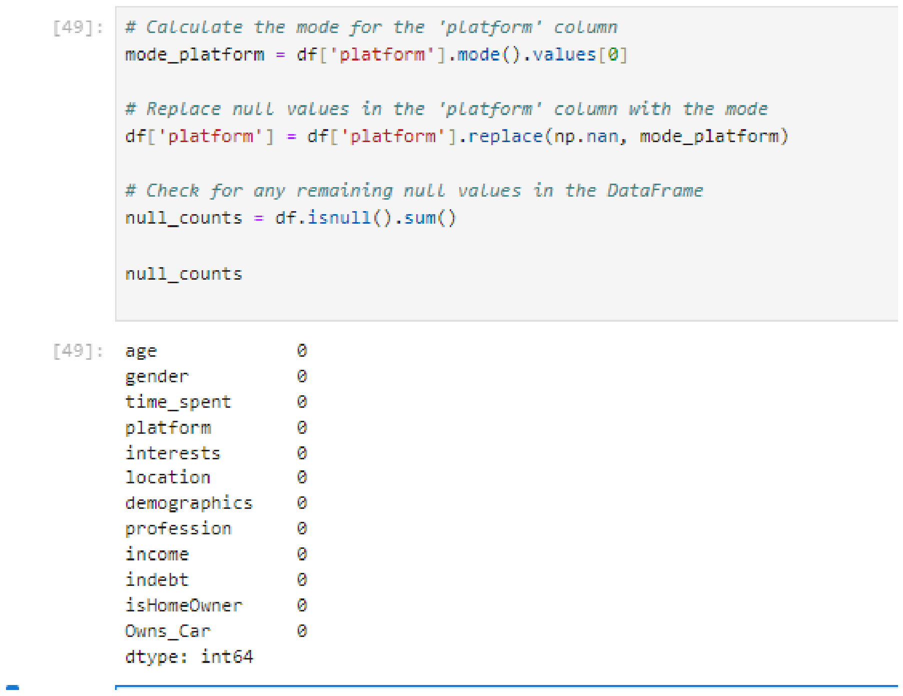
I did not have any null values in my .csv so I modified it to contain null values.

The code reads the CSV file into a Data Frame, replaces missing 'gender' values with "No Gender," and then counts null values across all columns.



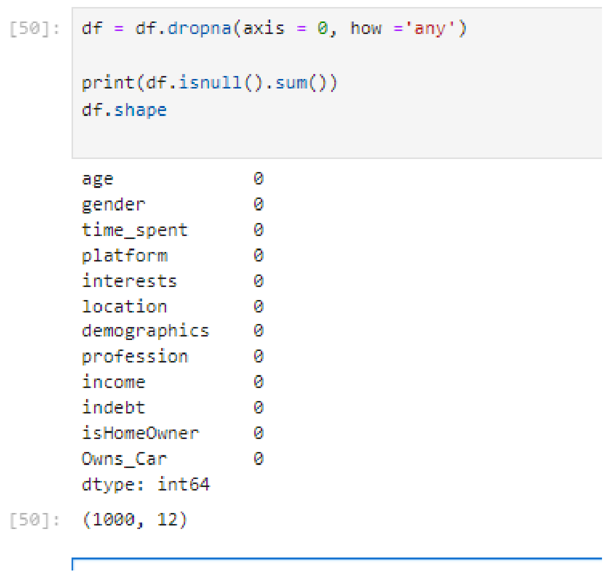
```
[48]:  df = pd.read_csv('timeOnSM.csv')
       df["gender"].fillna("No Gender", inplace = True)

       df.isnull().sum()

[48]:  age             0
       gender          0
       time_spent      0
       platform        1
       interests       0
       location        0
       demographics    0
       profession      0
       income          0
       indebt          0
       isHomeOwner     0
       Owns_Car        0
       dtype: int64

[ ]:
```

The code calculates the most common value ('mode') in the 'platform' column, fills in missing values in that column with this mode, and then reports any remaining nulls in the Data Frame.



```
[49]:  # Calculate the mode for the 'platform' column
       mode_platform = df['platform'].mode().values[0]

       # Replace null values in the 'platform' column with the mode
       df['platform'] = df['platform'].replace(np.nan, mode_platform)

       # Check for any remaining null values in the DataFrame
       null_counts = df.isnull().sum()

       null_counts

[49]:  age             0
       gender          0
       time_spent      0
       platform        0
       interests       0
       location        0
       demographics    0
       profession      0
       income          0
       indebt          0
       isHomeOwner     0
       Owns_Car        0
       dtype: int64
```

The code removes all rows with any missing values from the Data Frame, then prints the count of nulls in each column (expected to be zero) and outputs the Data Frame’s new dimensions.



```
[50]: df = df.dropna(axis = 0, how ='any')

      print(df.isnull().sum())
      df.shape
```

```
age             0
gender          0
time_spent      0
platform        0
interests       0
location        0
demographics    0
profession      0
income          0
indebt          0
isHomeOwner     0
Owns_Car        0
dtype: int64
```

```
[50]: (1000, 12)
```

Conclusion

From doing the EDA lab report and applying these commands to the 'timeOnSM.csv' dataset, I effectively handled missing values, analysed data distributions, and visualized relationships, leading to cleaner and more insightful data for further analysis.