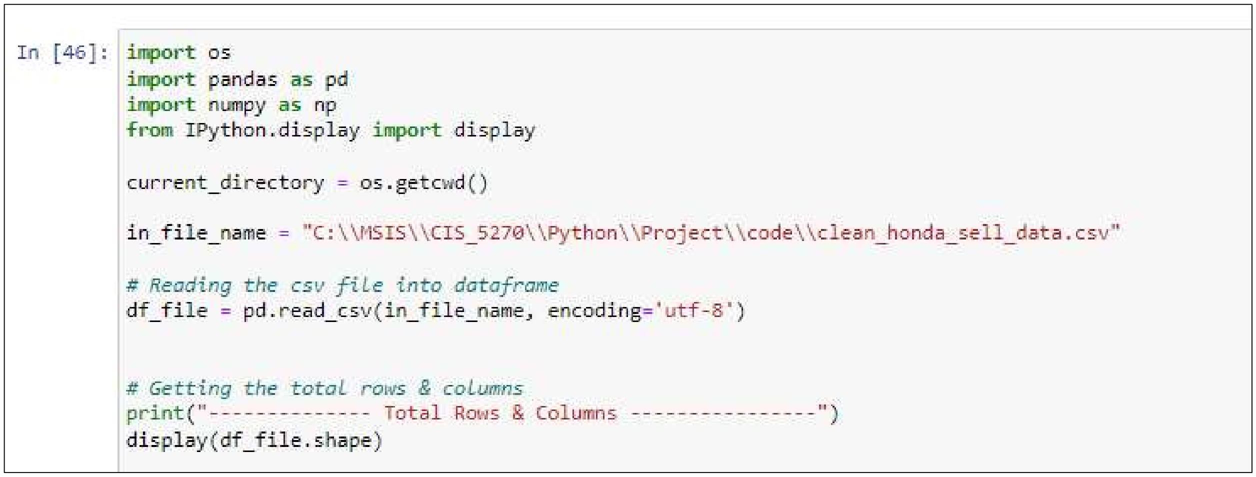
D. Statistical Summary

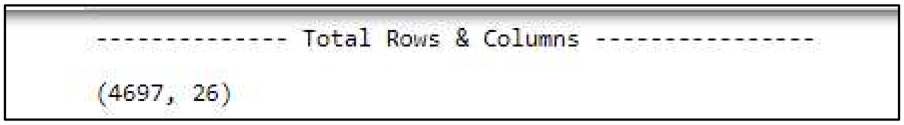
1. Getting the Total Rows & Columns

Functions used: pandas -> shape()



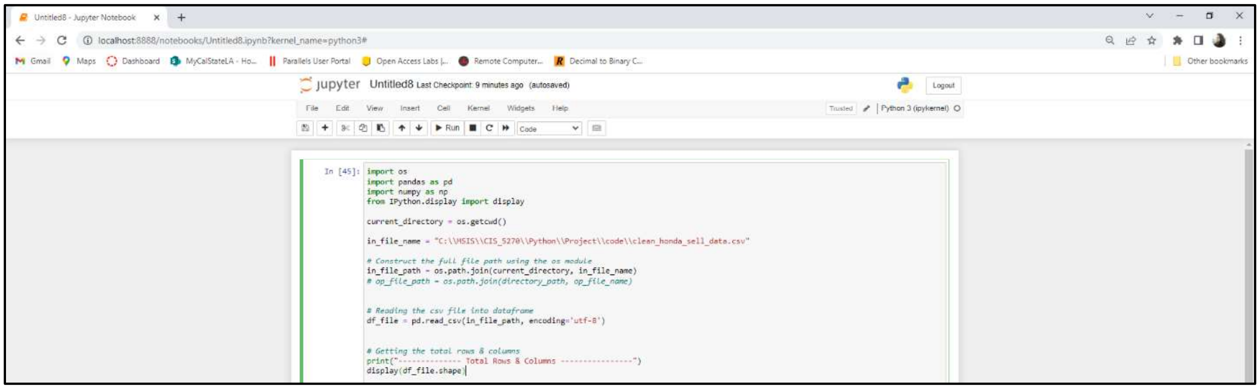
```
In [46]: import os
         import pandas as pd
         import numpy as np
         from IPython.display import display

         current_directory = os.getcwd()

         in_file_name = "C:\\MSIS\\CIS_5270\\Python\\Project\\code\\clean_honda_sell_data.csv"

         # Reading the csv file into dataframe
         df_file = pd.read_csv(in_file_name, encoding='utf-8')


         # Getting the total rows & columns
         print("------------- Total Rows & Columns ---------------")
         display(df_file.shape)
```

Output:



```
------------- Total Rows & Columns ---------------

(4697, 26)
```

Overall Screenshot:



Untitled8 - Jupyter Notebook
localhost:8888/notebooks/Untitled8.ipynb?kernel_name=python3#
Gmail
Maps
Dashboard
MyCalStateLA - Ho...
Parallels User Portal
Open Access Labs |...
Remote Computer...
Decimal to Binary C...
Other bookmarks
jupyter Untitled8 Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

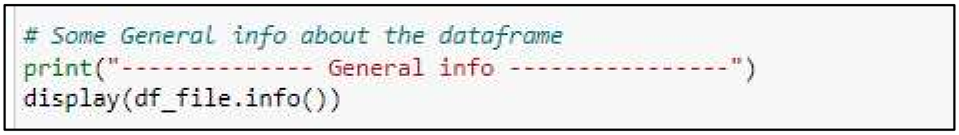
```
In [45]: import os
         import pandas as pd
         import numpy as np
         from IPython.display import display

         current_directory = os.getcwd()

         in_file_name = "C:\\MSIS\\CIS_5270\\Python\\Project\\code\\clean_honda_sell_data.csv"

         # Construct the full file path using the os module
         in_file_path = os.path.join(current_directory, in_file_name)
         # op_file_path = os.path.join(directory_path, op_file_name)


         # Reading the csv file into dataframe
         df_file = pd.read_csv(in_file_path, encoding='utf-8')


         # Getting the total rows & columns
         print("------------- Total Rows & Columns ---------------")
         display(df_file.shape)
```

(4697, 26), here 4697 represents the row and 26 represents the column. By this we can understand that there are 4697 rows and 26 columns.

Getting the total number of rows and columns in a dataset can provide valuable insights into the data's size, completeness, quality, and structure, and for this we can use **Shape()** function. The **shape** attribute of a pandas Data Frame returns a tuple representing the number of rows and columns in the Data Frame, respectively.

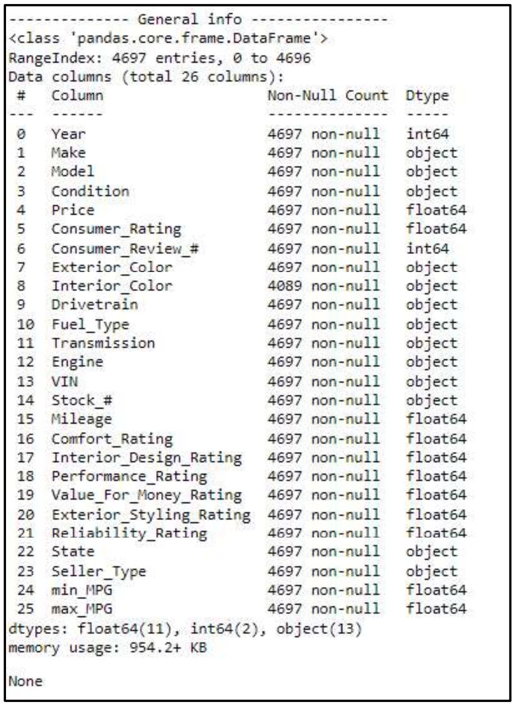
2. Getting concise Summary of a Data Frame

Functions used: pandas -> info()



```
# Some General info about the dataframe
print("-------------- General info ----------------")
display(df_file.info())
```

Output:



```
-------------- General info ----------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4697 entries, 0 to 4696
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Year                     4697 non-null   int64  
 1   Make                     4697 non-null   object 
 2   Model                    4697 non-null   object 
 3   Condition                4697 non-null   object 
 4   Price                    4697 non-null   float64
 5   Consumer_Rating          4697 non-null   float64
 6   Consumer_Review_#        4697 non-null   int64  
 7   Exterior_Color           4697 non-null   object 
 8   Interior_Color           4089 non-null   object 
 9   Drivetrain               4697 non-null   object 
 10  Fuel_Type                4697 non-null   object 
 11  Transmission             4697 non-null   object 
 12  Engine                   4697 non-null   object 
 13  VIN                      4697 non-null   object 
 14  Stock_#                  4697 non-null   object 
 15  Mileage                  4697 non-null   float64
 16  Comfort_Rating           4697 non-null   float64
 17  Interior_Design_Rating   4697 non-null   float64
 18  Performance_Rating       4697 non-null   float64
 19  Value_For_Money_Rating   4697 non-null   float64
 20  Exterior_Styling_Rating  4697 non-null   float64
 21  Reliability_Rating       4697 non-null   float64
 22  State                    4697 non-null   object 
 23  Seller_Type              4697 non-null   object 
 24  min_MPG                  4697 non-null   float64
 25  max_MPG                  4697 non-null   float64
dtypes: float64(11), int64(2), object(13)
memory usage: 954.2+ KB

None
```

From the above output, we can see column names and their data types. We also can see the non-null values, null values in the dataset.

Int64: Year, Consumer_Review_#.

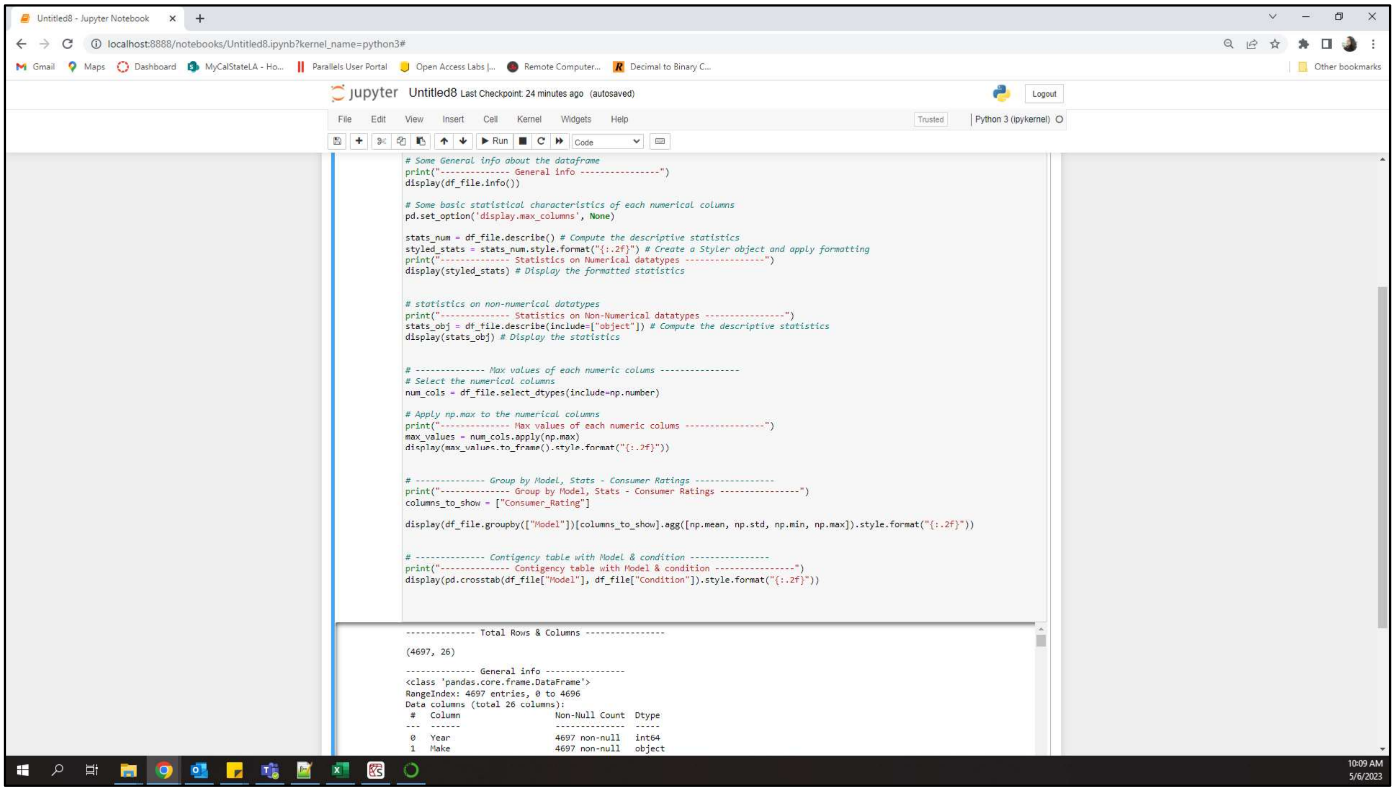
Float64: Price, Consumer_Rating,Milage, Comfort_Rating, Interior_Design_Rating, Performance_Rating, Value_For_Money_Rating, Exterior_Styling_Rating, Reliablity_Rating.

Object: Make, Model, Condition, Exterior_Color, Interior_Color, Drivetrain, Transmission, Fuel_Type, Engine, VIN, Stock_#, State, Seller_Type.

Through the output, we identified the following data types and their columns, and all the columns are non- null.

The function display(df_file.info()) gives a summary of the metadata and information about a pandas Data Frame df_file.

Overall Screenshot:

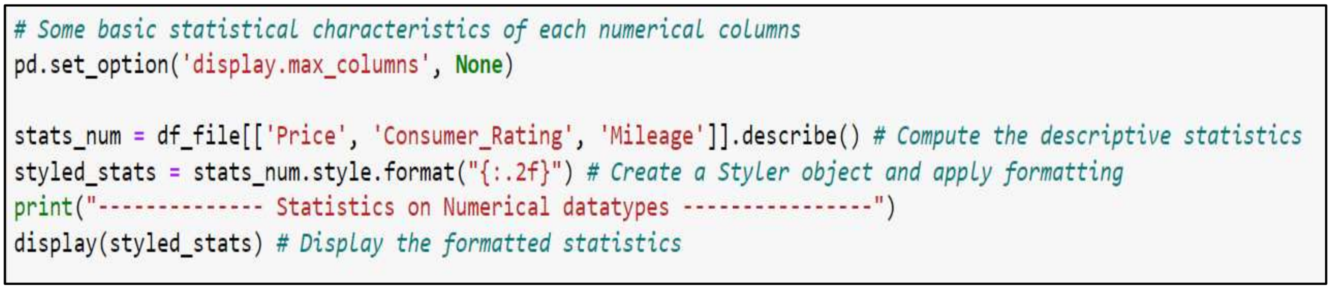


```
# Some General info about the dataframe
print("------------- General info ---------------")
display(df_file.info())

# Some basic statistical characteristics of each numerical columns
pd.set_option('display.max_columns', None)

stats_num = df_file.describe() # Compute the descriptive statistics
styled_stats = stats_num.style.format("{:.2f}") # Create a Styler object and apply formatting
print("------------- Statistics on Numerical datatypes ---------------")
display(styled_stats) # Display the formatted statistics


# statistics on non-numerical datatypes
print("------------- Statistics on Non-Numerical datatypes ---------------")
stats_obj = df_file.describe(include=["object"]) # Compute the descriptive statistics
display(stats_obj) # Display the statistics


# ------------- Max values of each numeric colums ---------------
# Select the numerical columns
num_cols = df_file.select_dtypes(include=np.number)

# Apply np.max to the numerical columns
print("------------- Max values of each numeric colums ---------------")
max_values = num_cols.apply(np.max)
display(max_values.to_frame().style.format("{:.2f}"))


# ------------- Group by Model, Stats - Consumer Ratings ---------------
print("------------- Group by Model, Stats - Consumer Ratings ---------------")
columns_to_show = ["Consumer_Rating"]

display(df_file.groupby(["Model"])[columns_to_show].agg([np.mean, np.std, np.min, np.max]).style.format("{:.2f}"))


# ------------- Contigency table with Model & condition ---------------
print("------------- Contigency table with Model & condition ---------------")
display(pd.crosstab(df_file["Model"], df_file["Condition"]).style.format("{:.2f}"))
```

```
------------- Total Rows & Columns ---------------

(4697, 26)

------------- General info ---------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4697 entries, 0 to 4696
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Year                   4697 non-null   int64
 1   Make                   4697 non-null   object
```

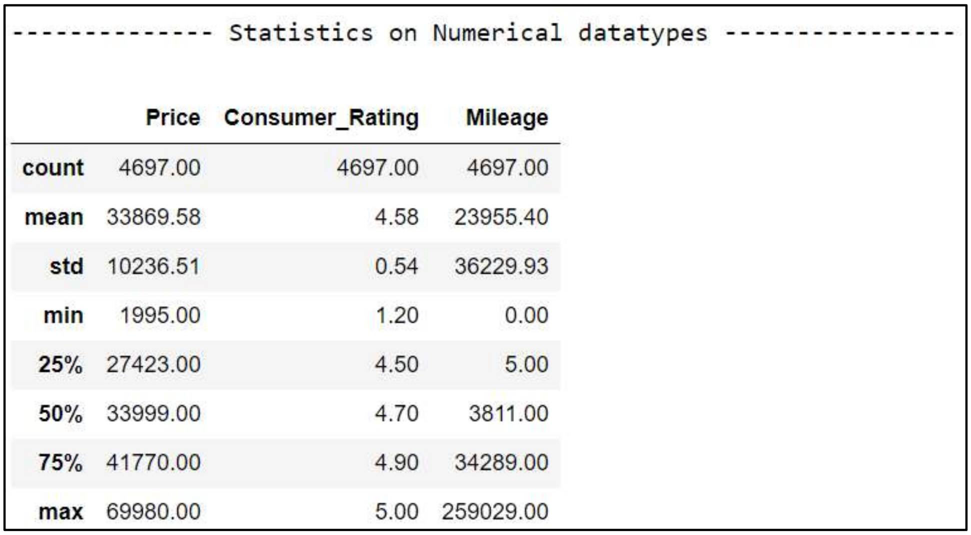
3. Some basic statistical characteristics of each numerical columns

Functions used: pandas -> describe(), set_option(), IPython.display -> display(), style()



```python
# Some basic statistical characteristics of each numerical columns
pd.set_option('display.max_columns', None)

stats_num = df_file[['Price', 'Consumer_Rating', 'Mileage']].describe() # Compute the descriptive statistics
styled_stats = stats_num.style.format("{:.2f}") # Create a Styler object and apply formatting
print("------------- Statistics on Numerical datatypes ---------------")
display(styled_stats) # Display the formatted statistics
```

Output:



```
------------- Statistics on Numerical datatypes ---------------
```

	Price	Consumer_Rating	Mileage
count	4697.00	4697.00	4697.00
mean	33869.58	4.58	23955.40
std	10236.51	0.54	36229.93
min	1995.00	1.20	0.00
25%	27423.00	4.50	5.00
50%	33999.00	4.70	3811.00
75%	41770.00	4.90	34289.00
max	69980.00	5.00	259029.00

The above code block is used to compute and display the descriptive statistics of specific columns in a pandas Data Frame df_file. By formatting the output, it is easier to read and interpret.

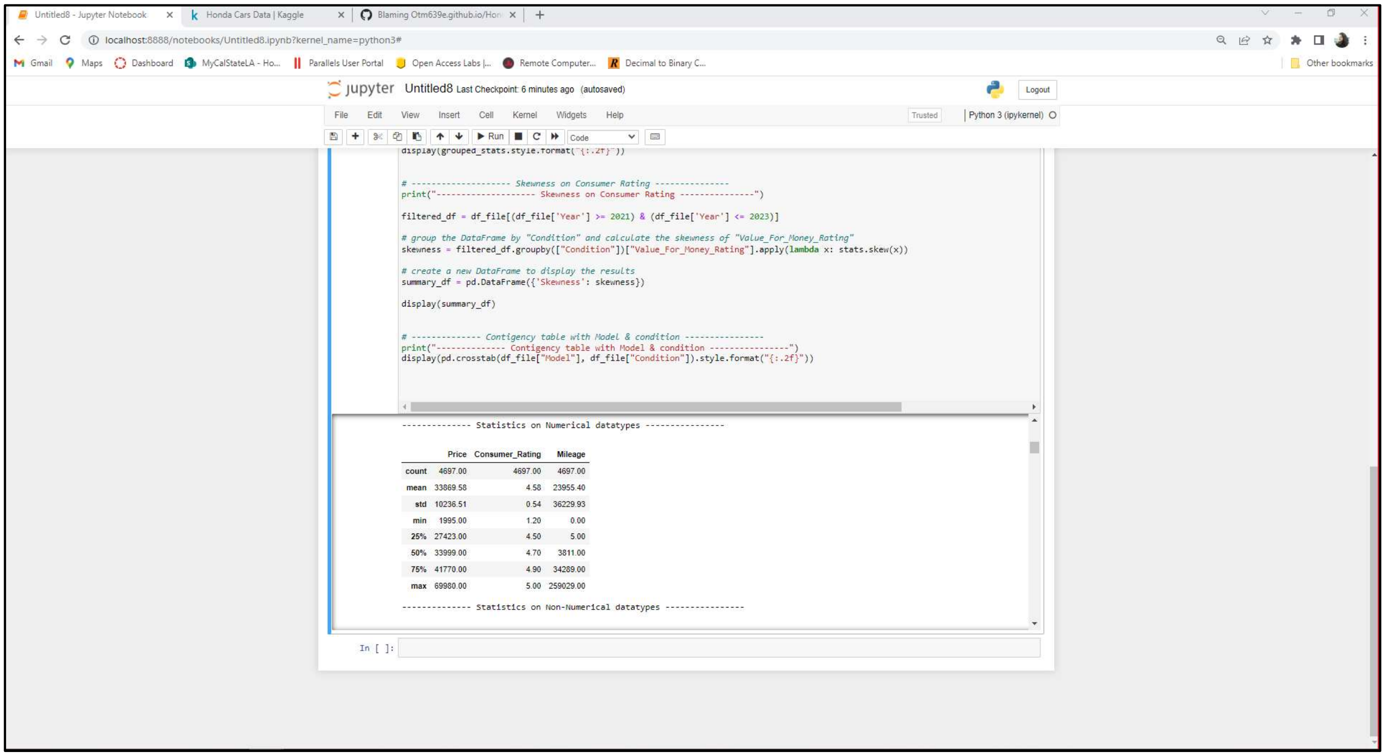
The **columns** in which we performed the summary statics are **Price, Consumer_Rating, Mileage.**

Through this Summary statics analysis, we got the following output:

- The **Count** of Price is 4697.00, for rating also it is 4697.00 and for Mileage is 4697.00
- The **mean** value of Price is 33869.58, for rating the mean is 4.58 and for Mileage it is 23955.40.
- The **standard deviation** indicates the amount of variability or dispersion in the data. For Price its SD is 10236.51, for rating it is 0.54 and for Mileage it is 36229.93.

- The **minimum** value of Price is 1995.00, rating is 1.20 and for Mileage is 0.00.
- **Quartiles**: The output shows the 25th, 50th (median), and 75th percentiles of the data.
- **25%** of Price value is 27423.00, rating value is 4.50 and Mileage value is 5.00.
- **50%** of Price value is 33999.00, rating value is 4.70 and Mileage value is 3811.00.
- **75%** of Price value is 41770.00, rating value is 4.90 and Mileage value is 34289.00.
- The **maximum** value of Price is 69980.00, rating is 5.00 and for Mileage is 259029.00.

Overall Screenshot:



```
display(grouped_stats.style.format("{:.2f}"))


# ------------------- Skewness on Consumer Rating --------------
print("------------------- Skewness on Consumer Rating --------------")

filtered_df = df_file[(df_file['Year'] >= 2021) & (df_file['Year'] <= 2023)]

# group the DataFrame by "Condition" and calculate the skewness of "Value_For_Money_Rating"
skewness = filtered_df.groupby(["Condition"])["Value_For_Money_Rating"].apply(lambda x: stats.skew(x))

# create a new DataFrame to display the results
summary_df = pd.DataFrame({'Skewness': skewness})

display(summary_df)


# ------------- Contigency table with Model & condition ---------------
print("------------- Contigency table with Model & condition ---------------")
display(pd.crosstab(df_file["Model"], df_file["Condition"]).style.format("{:.2f}"))
```

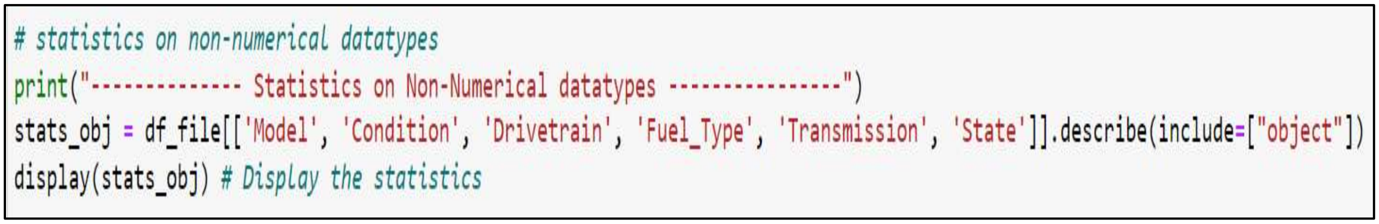
```
------------- Statistics on Numerical datatypes ---------------
```

	Price	Consumer_Rating	Mileage
count	4697.00	4697.00	4697.00
mean	33869.58	4.58	23955.40
std	10236.51	0.54	36229.93
min	1995.00	1.20	0.00
25%	27423.00	4.50	5.00
50%	33999.00	4.70	3811.00
75%	41770.00	4.90	34289.00
max	69980.00	5.00	259029.00

```
------------- Statistics on Non-Numerical datatypes ---------------
```

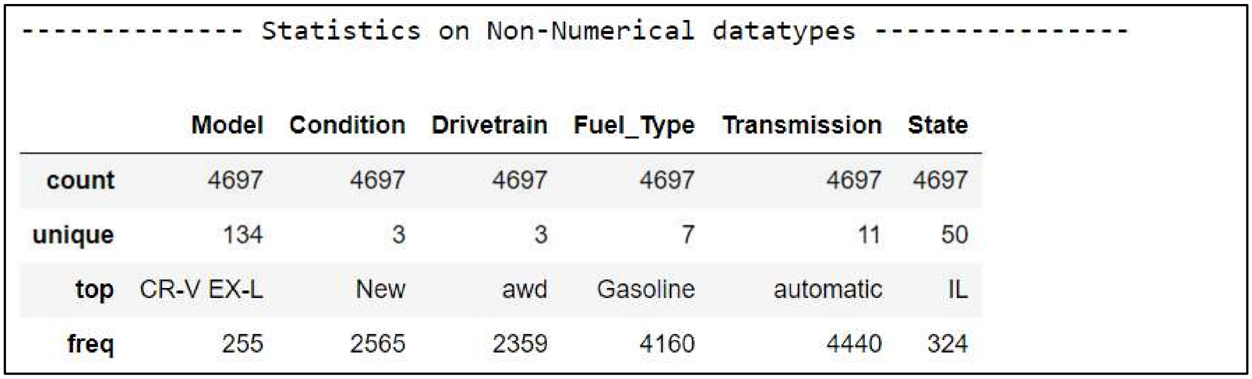
4. Statistics on non-numerical datatypes

Functions used: pandas -> describe(), set_option(), IPython.display -> display()



```
# statistics on non-numerical datatypes
print("------------- Statistics on Non-Numerical datatypes ---------------")
stats_obj = df_file[['Model', 'Condition', 'Drivetrain', 'Fuel_Type', 'Transmission', 'State']].describe(include=["object"])
display(stats_obj) # Display the statistics
```

Output:



```
------------- Statistics on Non-Numerical datatypes ---------------
```

	Model	Condition	Drivetrain	Fuel_Type	Transmission	State
count	4697	4697	4697	4697	4697	4697
unique	134	3	3	7	11	50
top	CR-V EX-L	New	awd	Gasoline	automatic	IL
freq	255	2565	2359	4160	4440	324

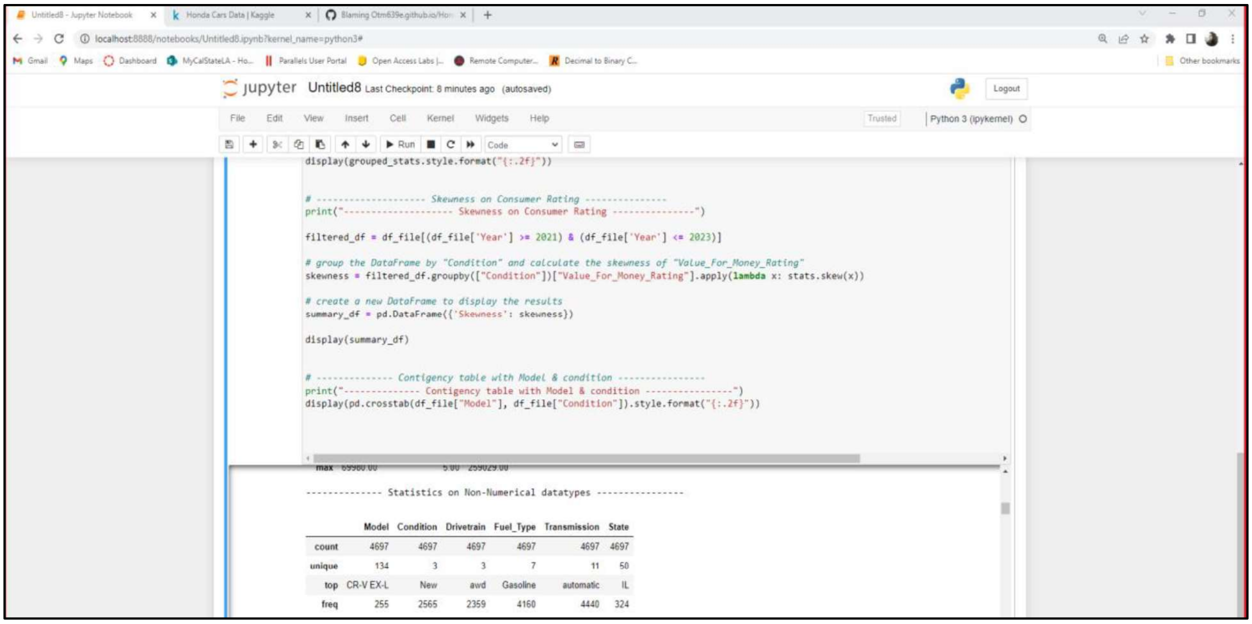
The above code displays the descriptive statistics of six columns in a pandas Data Frame **df_file**.

The columns taken into consideration for the above code is **Model, Condition, Drivetrain, Fuel_Type, Transmission, and State.**

The output of this code block include:

- The **Count** of Model is 4697, for Condition it is 4697, Drivetrain it is 4697, Fuel_type it is 4697and for transmission we have 4697 and State it is 4697.
- The **Unique** represents the number of unique values in each column, for Model is 134, for Condition it is New, Drivetrain it is awd, Fuel_type it is 7 and for transmission we have11 and State it is 15.
- **Top** represents the most frequent value (mode) in each column for Model is CRV-EX-L, for Condition it is 3, Drivetrain it is 3, Fuel_type it is Gasoline and for transmission it is automatic and State it is IL.
- The **frequency** (count) of the top value in each column for Model is 255, for Condition it is 2565, Drivetrain it is 2359, Fuel_type it is 4160 and for transmission it is 4440 and State it is 324.

Overall Screenshot:



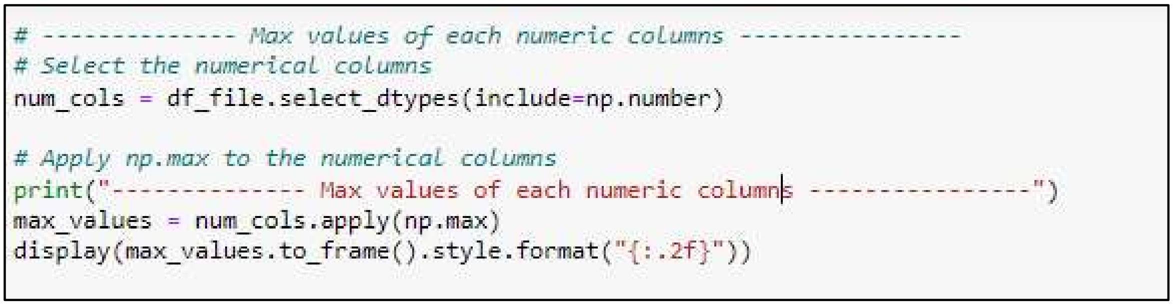
```
display(grouped_stats.style.format("{:.2f}"))


# ------------------- Skewness on Consumer Rating --------------
print("------------------- Skewness on Consumer Rating --------------")

filtered_df = df_file[(df_file['Year'] >= 2021) & (df_file['Year'] <= 2023)]

# group the DataFrame by "Condition" and calculate the skewness of "Value_For_Money_Rating"
skewness = filtered_df.groupby(["Condition"])["Value_For_Money_Rating"].apply(lambda x: stats.skew(x))

# create a new DataFrame to display the results
summary_df = pd.DataFrame({'Skewness': skewness})

display(summary_df)


# ------------- Contigency table with Model & condition ---------------
print("------------- Contigency table with Model & condition ---------------")
display(pd.crosstab(df_file["Model"], df_file["Condition"]).style.format("{:.2f}"))
```

```
  max  69980.00            5.00  259029.00

------------- Statistics on Non-Numerical datatypes ---------------
```

	Model	Condition	Drivetrain	Fuel_Type	Transmission	State
count	4697	4697	4697	4697	4697	4697
unique	134	3	3	7	11	50
top	CR-V EX-L	New	awd	Gasoline	automatic	IL
freq	255	2565	2359	4160	4440	324

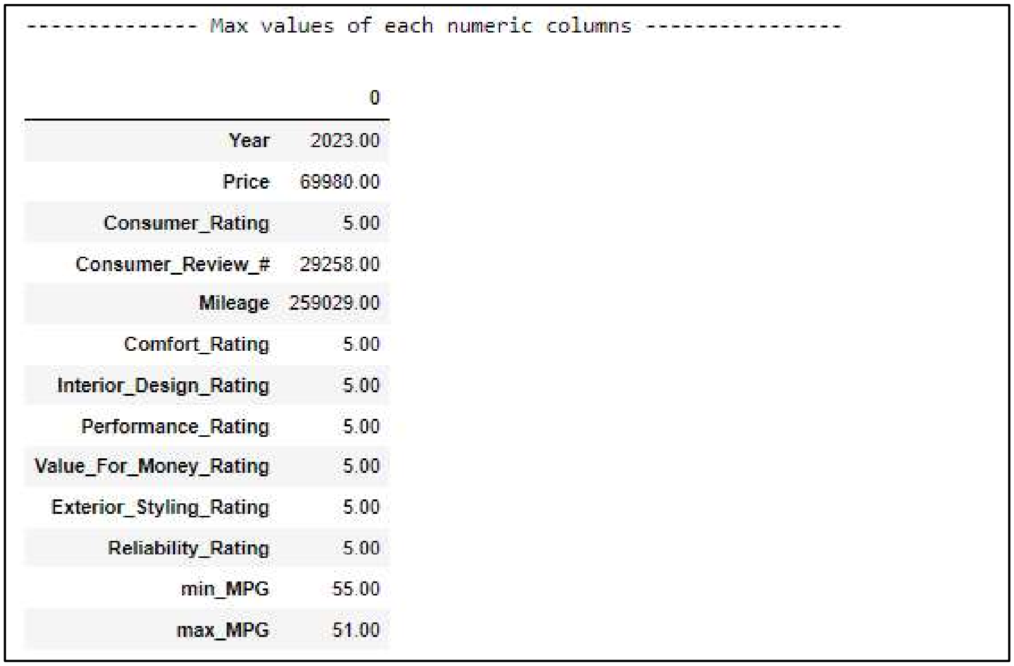
5. Showing Max values of each numeric columns

Functions used: numpy -> max(), pandas -> select_dtypes(), appy(), IPython.display -> display()



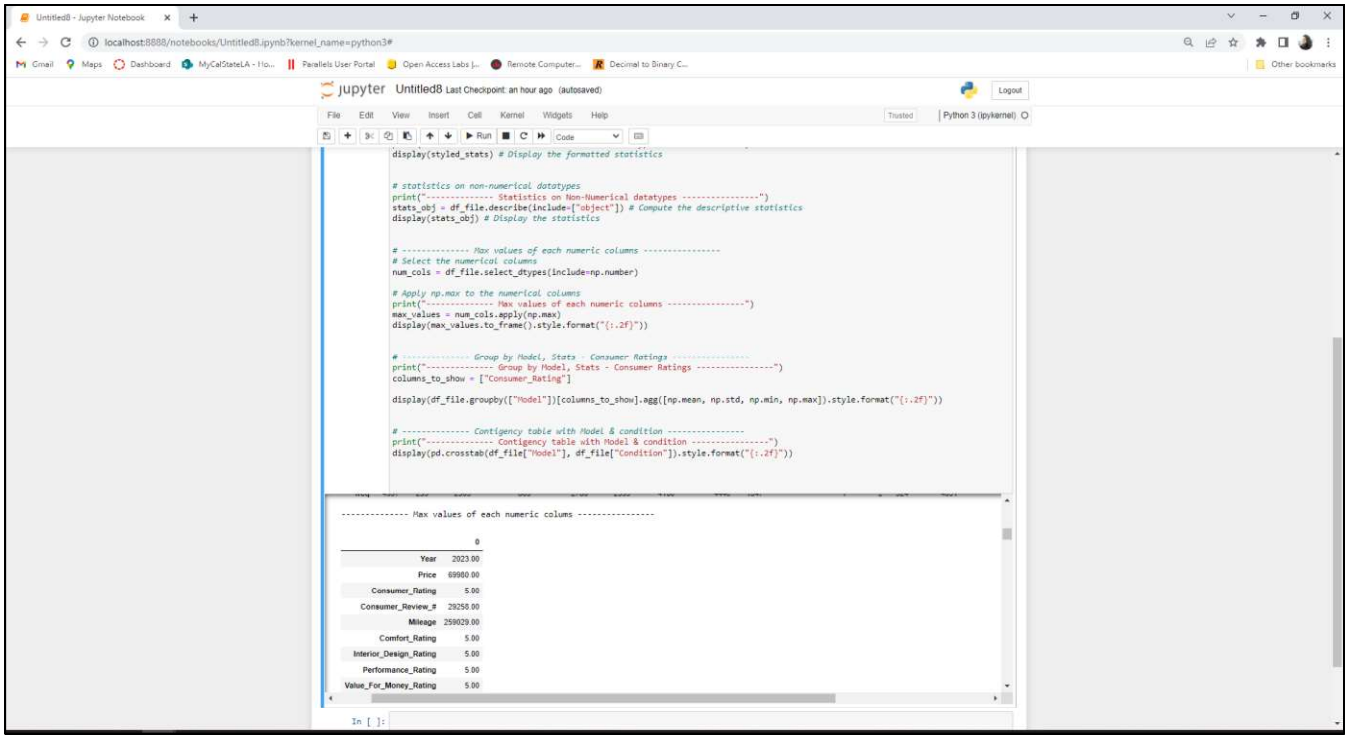
```
# -------------- Max values of each numeric columns ----------------
# Select the numerical columns
num_cols = df_file.select_dtypes(include=np.number)

# Apply np.max to the numerical columns
print("-------------- Max values of each numeric columns ----------------")
max_values = num_cols.apply(np.max)
display(max_values.to_frame().style.format("{:.2f}"))
```

Output:



```
-------------- Max values of each numeric columns ----------------

                              0
                 Year   2023.00
                Price  69980.00
      Consumer_Rating      5.00
    Consumer_Review_#  29258.00
              Mileage 259029.00
       Comfort_Rating      5.00
Interior_Design_Rating     5.00
   Performance_Rating      5.00
Value_For_Money_Rating     5.00
 Exterior_Styling_Rating   5.00
   Reliability_Rating      5.00
              min_MPG     55.00
              max_MPG     51.00
```

Overall Screenshot:



```
display(styled_stats) # Display the formatted statistics


# statistics on non-numerical datatypes
print("------------- Statistics on Non-Numerical datatypes ---------------")
stats_obj = df_file.describe(include=["object"]) # Compute the descriptive statistics
display(stats_obj) # Display the statistics


# ------------- Max values of each numeric columns ---------------
# Select the numerical columns
num_cols = df_file.select_dtypes(include=np.number)

# Apply np.max to the numerical columns
print("------------- Max values of each numeric columns ---------------")
max_values = num_cols.apply(np.max)
display(max_values.to_frame().style.format("{:.2f}"))


# ------------- Group by Model, Stats - Consumer Ratings ---------------
print("------------- Group by Model, Stats - Consumer Ratings ---------------")
columns_to_show = ["Consumer_Rating"]

display(df_file.groupby(["Model"])[columns_to_show].agg([np.mean, np.std, np.min, np.max]).style.format("{:.2f}"))


# ------------- Contigency table with Model & condition ---------------
print("------------- Contigency table with Model & condition ---------------")
display(pd.crosstab(df_file["Model"], df_file["Condition"]).style.format("{:.2f}"))
```

```
-------------- Max values of each numeric colums ----------------

                              0
                 Year   2023.00
                Price  69980.00
      Consumer_Rating      5.00
    Consumer_Review_#  29258.00
              Mileage 259029.00
       Comfort_Rating      5.00
Interior_Design_Rating     5.00
   Performance_Rating      5.00
Value_For_Money_Rating     5.00
```

```
In [ ]:
```

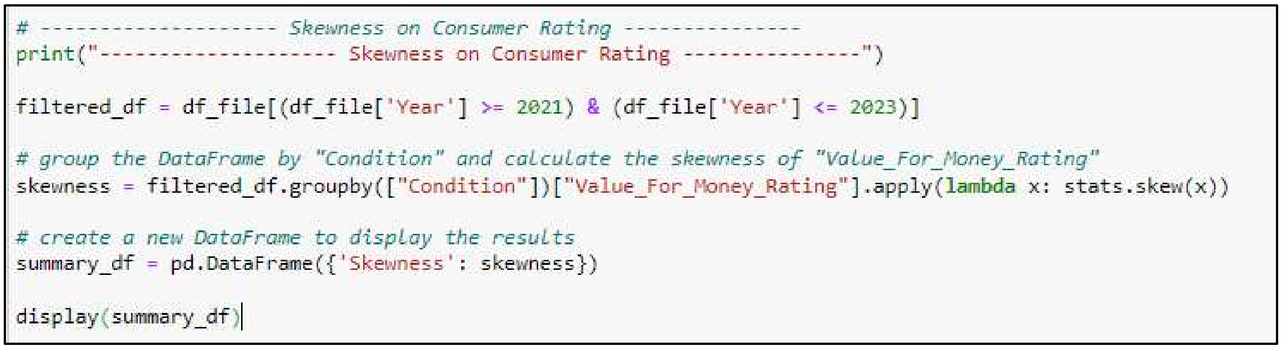
This code performs an analysis on a dataset, specifically looking at the maximum value of each column that contains numerical data. Np.max() function is applied to each of the numerical columns, which finds the maximum value in each column.

The results are displayed as the output of the code:

- The maximum value found in the **Year** column was 2023.
- The highest value found in the **Price** column was 69980.
- The **Consumer_Rating** column had a maximum value of 5.00.
- The **Consumer_Review_#** column had the highest value of 29258.
- The **Mileage** column had a maximum value of 259029.
- The **Comfort_Rating** column had the highest value of 5.00.
- The **Interior_Design_Rating** column had the maximum value of 5.00.
- The **Performance_Rating** column had the highest value of 5.00.
- The **Value_For_Money_Rating** column had the maximum value of 5.00.
- The **Exterior_Styling_Rating** column had the maximum value of 5.00.
- The **Reliability_Rating** column had the maximum value of 5.00.
- The **min_MPG** column had the highest value of 55.00.
- The **max_MPG** column had the highest value of 51.00

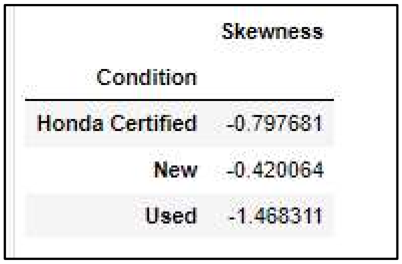
6. Skewness of Consumer ratings on ‘Value for money Ratings’ for last 3 years based on Condition (Used/New/Certified)

Functions used: scipy.stats -> skew(), pandas -> apply()



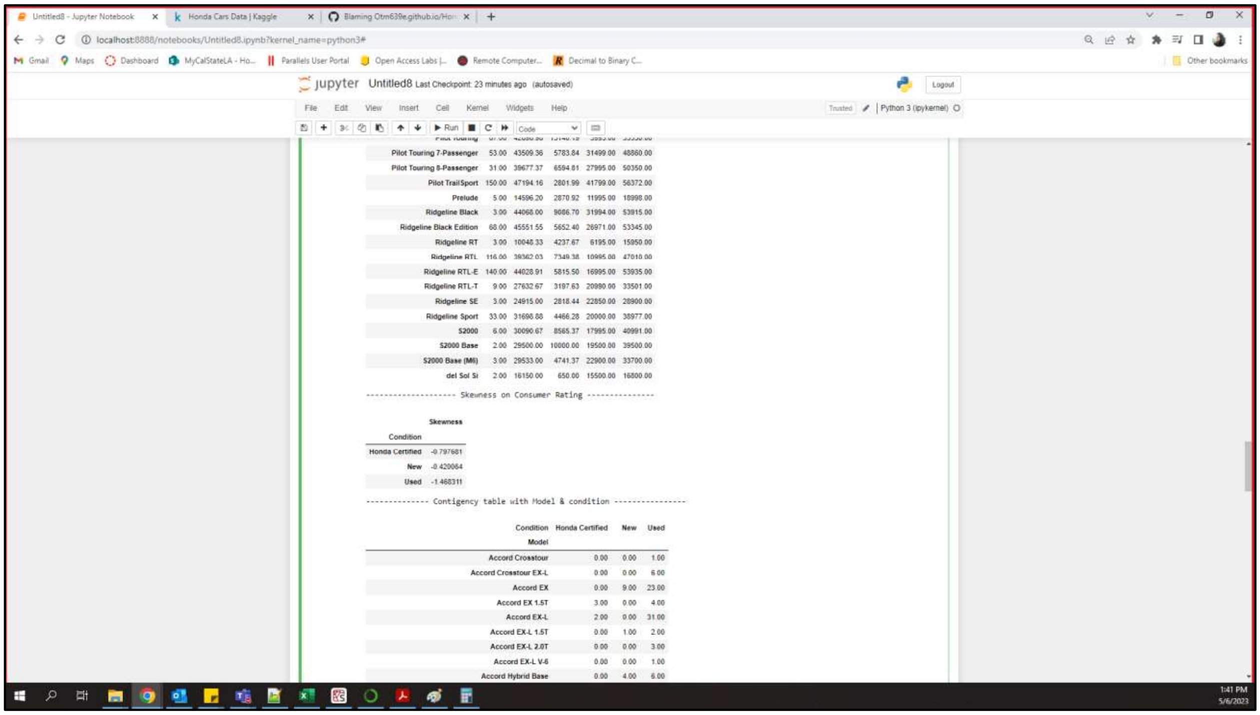
```
# -------------------- Skewness on Consumer Rating ---------------
print("-------------------- Skewness on Consumer Rating ---------------")

filtered_df = df_file[(df_file['Year'] >= 2021) & (df_file['Year'] <= 2023)]

# group the DataFrame by "Condition" and calculate the skewness of "Value_For_Money_Rating"
skewness = filtered_df.groupby(["Condition"])["Value_For_Money_Rating"].apply(lambda x: stats.skew(x))

# create a new DataFrame to display the results
summary_df = pd.DataFrame({'Skewness': skewness})

display(summary_df)
```

Output:



	Skewness
Condition	
Honda Certified	-0.797681
New	-0.420064
Used	-1.468311

Overall Screenshot:



```
   Pilot Touring 7-Passenger   53.00  43509.36   5783.84  31499.00  48860.00
   Pilot Touring 8-Passenger   31.00  39677.37   6594.81  27995.00  50350.00
           Pilot TrailSport   150.00  47194.16   2801.99  41799.00  56372.00
                    Prelude     5.00  14596.20   2870.92  11995.00  18998.00
            Ridgeline Black     3.00  44068.00   9086.70  31994.00  53915.00
    Ridgeline Black Edition    68.00  45551.55   5652.40  26971.00  53345.00
               Ridgeline RT     3.00  10048.33   4237.67   6195.00  15950.00
              Ridgeline RTL   116.00  38362.03   7349.38  10995.00  47010.00
            Ridgeline RTL-E   140.00  44028.91   5815.50  16995.00  53935.00
            Ridgeline RTL-T     9.00  27632.67   3197.63  20980.00  33501.00
               Ridgeline SE     3.00  24915.00   2818.44  22850.00  28900.00
            Ridgeline Sport    33.00  31696.88   4466.28  20000.00  38977.00
                      S2000     6.00  30090.67   8565.37  17995.00  40991.00
                 S2000 Base     2.00  29500.00  10000.00  19500.00  39500.00
            S2000 Base (Mi)     3.00  29533.00   4741.37  22900.00  33700.00
                 del Sol Si     2.00  16150.00    650.00  15500.00  16800.00
-------------------- Skewness on Consumer Rating ---------------

                  Skewness
        Condition
  Honda Certified  -0.797681
              New  -0.420064
             Used  -1.468311
------------- Contigency table with Model & condition ---------------

                     Condition  Honda Certified   New   Used
                         Model
              Accord Crosstour             0.00  0.00   1.00
         Accord Crosstour EX-L             0.00  0.00   6.00
                     Accord EX             0.00  9.00  23.00
                Accord EX 1.5T             3.00  0.00   4.00
                   Accord EX-L             2.00  0.00  31.00
              Accord EX-L 1.5T             0.00  1.00   2.00
              Accord EX-L 2.0T             0.00  0.00   3.00
               Accord EX-L V-6             0.00  0.00   1.00
            Accord Hybrid Base             0.00  4.00   6.00
```

The output shows the skewness values of the "Value_For_Money_Rating" column for each group of the "Condition" column. The three groups are "Honda Certified", "New", and "Used".

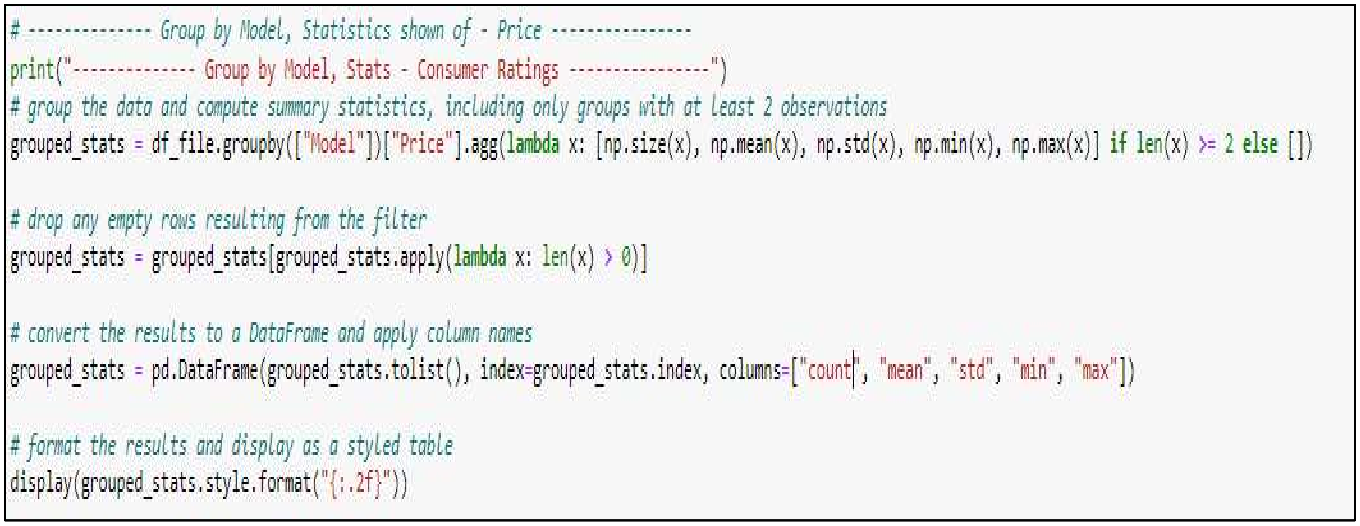
For the "Honda Certified" group, the skewness value is -0.797681. This suggests that the distribution of the "Value_For_Money_Rating" values for this group is slightly skewed to the left, meaning that there are more ratings on the higher end of the scale.

For the "New" group, the skewness value is -0.421407. This also indicates a slightly left-skewed distribution, which means that there are more ratings on the higher end of the scale for this group as well.

Finally, for the "Used" group, the skewness value is -1.468311. This suggests a more heavily left-skewed distribution, which indicates that there are more ratings on the higher end of the scale for this group, but there are also more extremely low ratings in this group compared to the other groups.

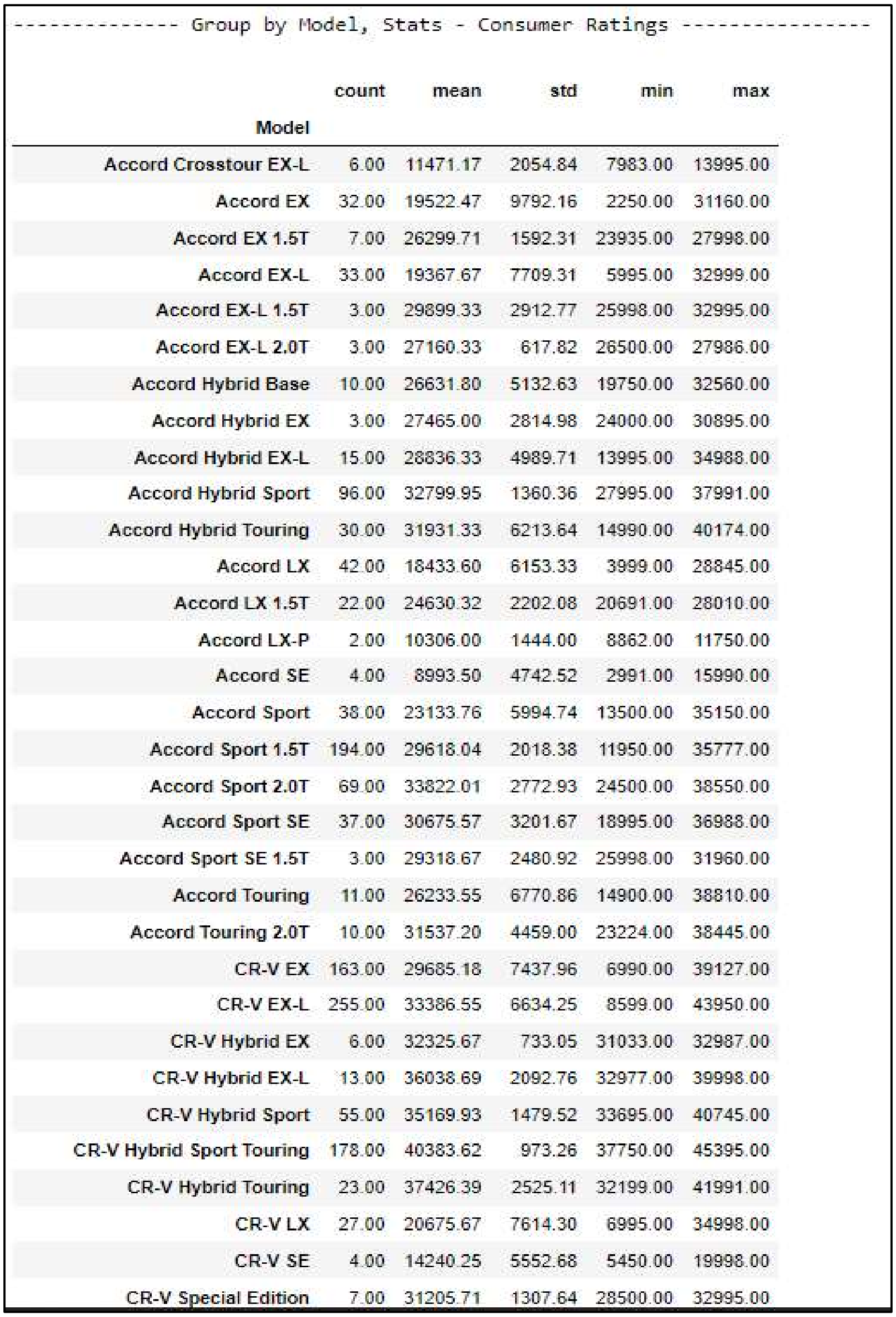
7. Statistics Summary for Price for each Car Model

Functions used: numpy -> mean(), std(), min(),max(), pandas -> groupby(), agg()



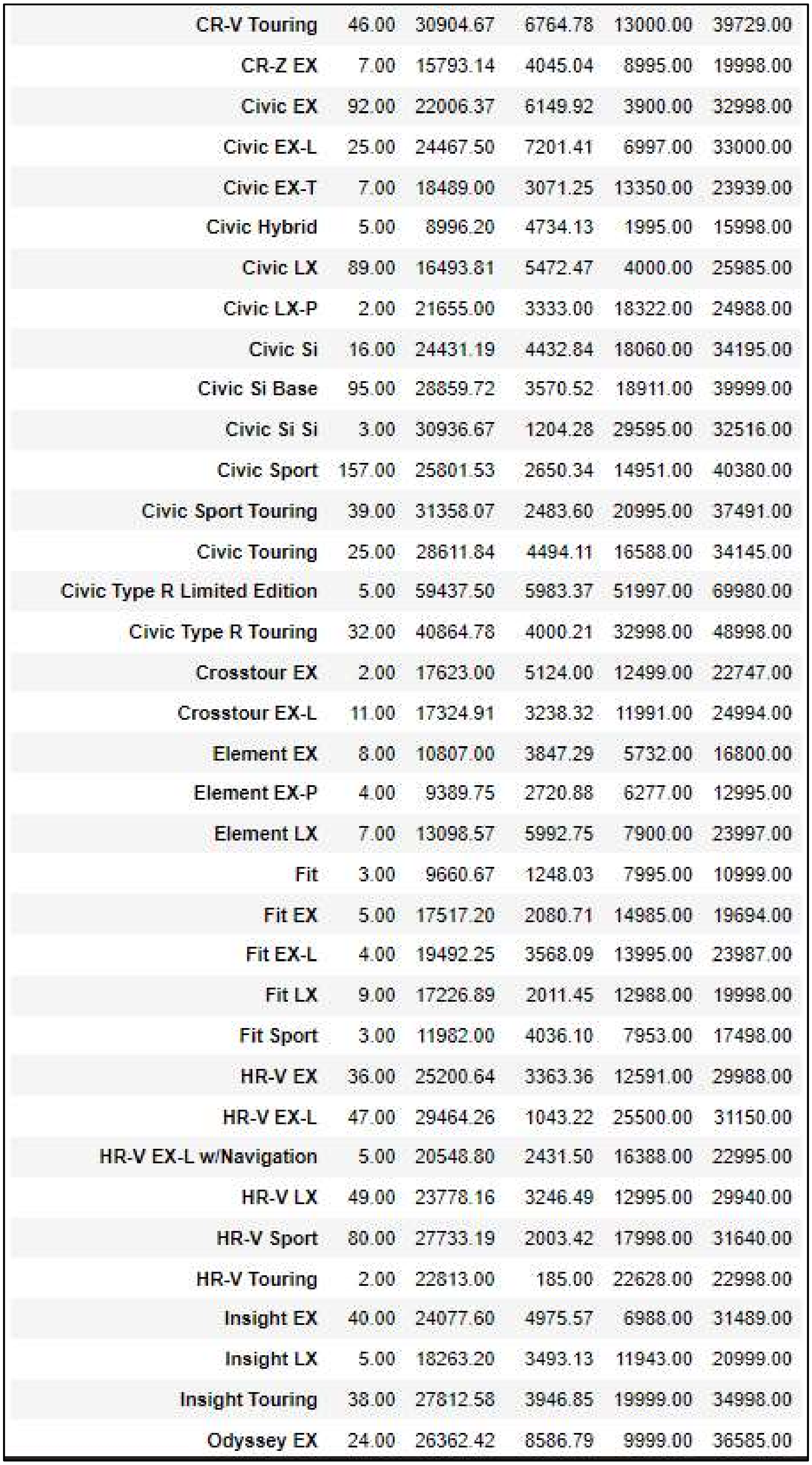
```
# ------------- Group by Model, Statistics shown of - Price ---------------
print("------------- Group by Model, Stats - Consumer Ratings ---------------")
# group the data and compute summary statistics, including only groups with at least 2 observations
grouped_stats = df_file.groupby(["Model"])["Price"].agg(lambda x: [np.size(x), np.mean(x), np.std(x), np.min(x), np.max(x)] if len(x) >= 2 else [])

# drop any empty rows resulting from the filter
grouped_stats = grouped_stats[grouped_stats.apply(lambda x: len(x) > 0)]

# convert the results to a DataFrame and apply column names
grouped_stats = pd.DataFrame(grouped_stats.tolist(), index=grouped_stats.index, columns=["count", "mean", "std", "min", "max"])

# format the results and display as a styled table
display(grouped_stats.style.format("{:.2f}"))
```

Output:

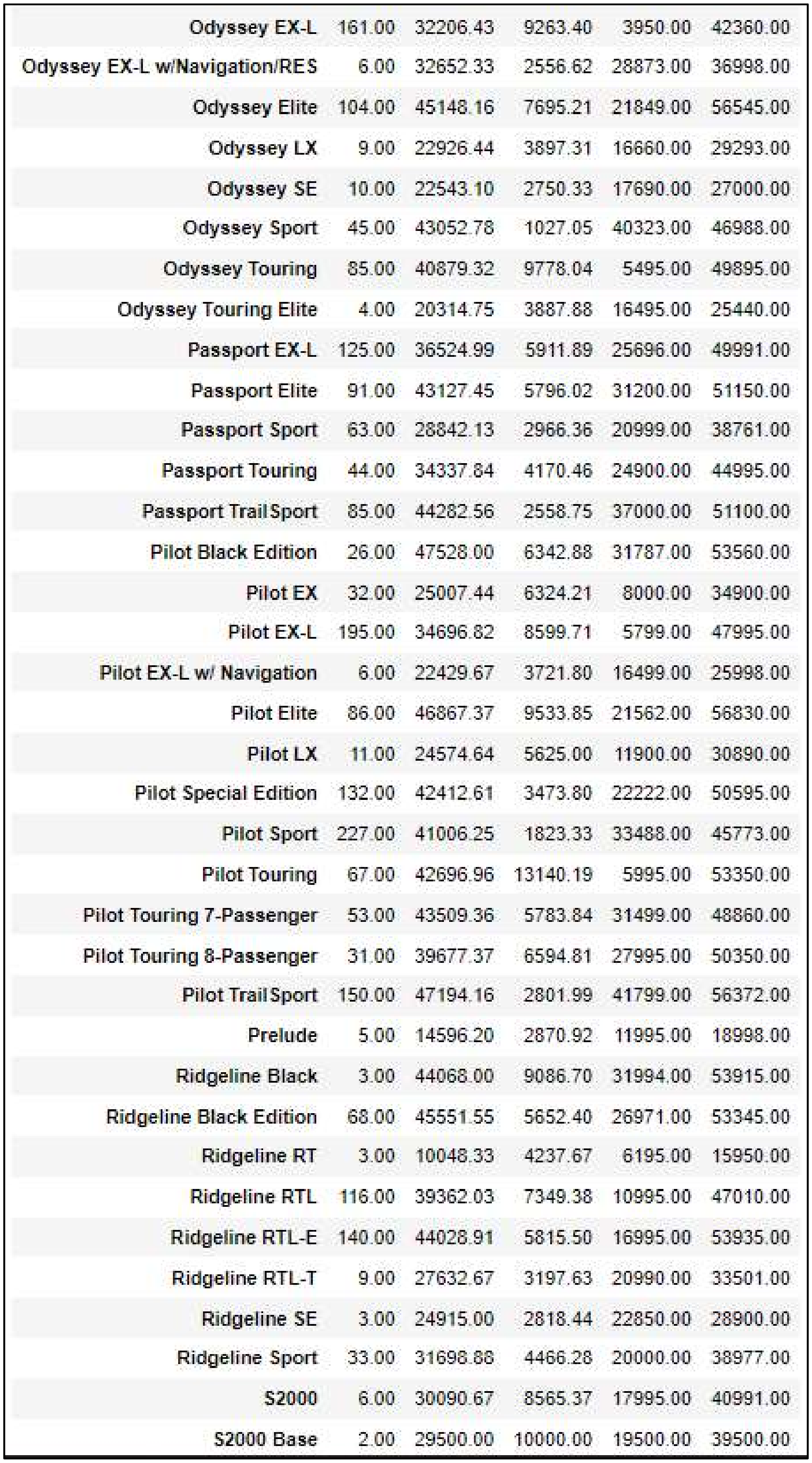


```
-------------- Group by Model, Stats - Consumer Ratings ----------------
```

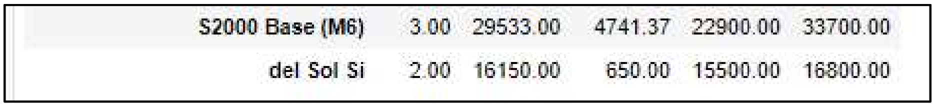
	count	mean	std	min	max
Model					
Accord Crosstour EX-L	6.00	11471.17	2054.84	7983.00	13995.00
Accord EX	32.00	19522.47	9792.16	2250.00	31160.00
Accord EX 1.5T	7.00	26299.71	1592.31	23935.00	27998.00
Accord EX-L	33.00	19367.67	7709.31	5995.00	32999.00
Accord EX-L 1.5T	3.00	29899.33	2912.77	25998.00	32995.00
Accord EX-L 2.0T	3.00	27160.33	617.82	26500.00	27986.00
Accord Hybrid Base	10.00	26631.80	5132.63	19750.00	32560.00
Accord Hybrid EX	3.00	27465.00	2814.98	24000.00	30895.00
Accord Hybrid EX-L	15.00	28836.33	4989.71	13995.00	34988.00
Accord Hybrid Sport	96.00	32799.95	1360.36	27995.00	37991.00
Accord Hybrid Touring	30.00	31931.33	6213.64	14990.00	40174.00
Accord LX	42.00	18433.60	6153.33	3999.00	28845.00
Accord LX 1.5T	22.00	24630.32	2202.08	20691.00	28010.00
Accord LX-P	2.00	10306.00	1444.00	8862.00	11750.00
Accord SE	4.00	8993.50	4742.52	2991.00	15990.00
Accord Sport	38.00	23133.76	5994.74	13500.00	35150.00
Accord Sport 1.5T	194.00	29618.04	2018.38	11950.00	35777.00
Accord Sport 2.0T	69.00	33822.01	2772.93	24500.00	38550.00
Accord Sport SE	37.00	30675.57	3201.67	18995.00	36988.00
Accord Sport SE 1.5T	3.00	29318.67	2480.92	25998.00	31960.00
Accord Touring	11.00	26233.55	6770.86	14900.00	38810.00
Accord Touring 2.0T	10.00	31537.20	4459.00	23224.00	38445.00
CR-V EX	163.00	29685.18	7437.96	6990.00	39127.00
CR-V EX-L	255.00	33386.55	6634.25	8599.00	43950.00
CR-V Hybrid EX	6.00	32325.67	733.05	31033.00	32987.00
CR-V Hybrid EX-L	13.00	36038.69	2092.76	32977.00	39998.00
CR-V Hybrid Sport	55.00	35169.93	1479.52	33695.00	40745.00
CR-V Hybrid Sport Touring	178.00	40383.62	973.26	37750.00	45395.00
CR-V Hybrid Touring	23.00	37426.39	2525.11	32199.00	41991.00
CR-V LX	27.00	20675.67	7614.30	6995.00	34998.00
CR-V SE	4.00	14240.25	5552.68	5450.00	19998.00
CR-V Special Edition	7.00	31205.71	1307.64	28500.00	32995.00



CR-V Touring	46.00	30904.67	6764.78	13000.00	39729.00
CR-Z EX	7.00	15793.14	4045.04	8995.00	19998.00
Civic EX	92.00	22006.37	6149.92	3900.00	32998.00
Civic EX-L	25.00	24467.50	7201.41	6997.00	33000.00
Civic EX-T	7.00	18489.00	3071.25	13350.00	23939.00
Civic Hybrid	5.00	8996.20	4734.13	1995.00	15998.00
Civic LX	89.00	16493.81	5472.47	4000.00	25985.00
Civic LX-P	2.00	21655.00	3333.00	18322.00	24988.00
Civic Si	16.00	24431.19	4432.84	18060.00	34195.00
Civic Si Base	95.00	28859.72	3570.52	18911.00	39999.00
Civic Si Si	3.00	30936.67	1204.28	29595.00	32516.00
Civic Sport	157.00	25801.53	2650.34	14951.00	40380.00
Civic Sport Touring	39.00	31358.07	2483.60	20995.00	37491.00
Civic Touring	25.00	28611.84	4494.11	16588.00	34145.00
Civic Type R Limited Edition	5.00	59437.50	5983.37	51997.00	69980.00
Civic Type R Touring	32.00	40864.78	4000.21	32998.00	48998.00
Crosstour EX	2.00	17623.00	5124.00	12499.00	22747.00
Crosstour EX-L	11.00	17324.91	3238.32	11991.00	24994.00
Element EX	8.00	10807.00	3847.29	5732.00	16800.00
Element EX-P	4.00	9389.75	2720.88	6277.00	12995.00
Element LX	7.00	13098.57	5992.75	7900.00	23997.00
Fit	3.00	9660.67	1248.03	7995.00	10999.00
Fit EX	5.00	17517.20	2080.71	14985.00	19694.00
Fit EX-L	4.00	19492.25	3568.09	13995.00	23987.00
Fit LX	9.00	17226.89	2011.45	12988.00	19998.00
Fit Sport	3.00	11982.00	4036.10	7953.00	17498.00
HR-V EX	36.00	25200.64	3363.36	12591.00	29988.00
HR-V EX-L	47.00	29464.26	1043.22	25500.00	31150.00
HR-V EX-L w/Navigation	5.00	20548.80	2431.50	16388.00	22995.00
HR-V LX	49.00	23778.16	3246.49	12995.00	29940.00
HR-V Sport	80.00	27733.19	2003.42	17998.00	31640.00
HR-V Touring	2.00	22813.00	185.00	22628.00	22998.00
Insight EX	40.00	24077.60	4975.57	6988.00	31489.00
Insight LX	5.00	18263.20	3493.13	11943.00	20999.00
Insight Touring	38.00	27812.58	3946.85	19999.00	34998.00
Odyssey EX	24.00	26362.42	8586.79	9999.00	36585.00

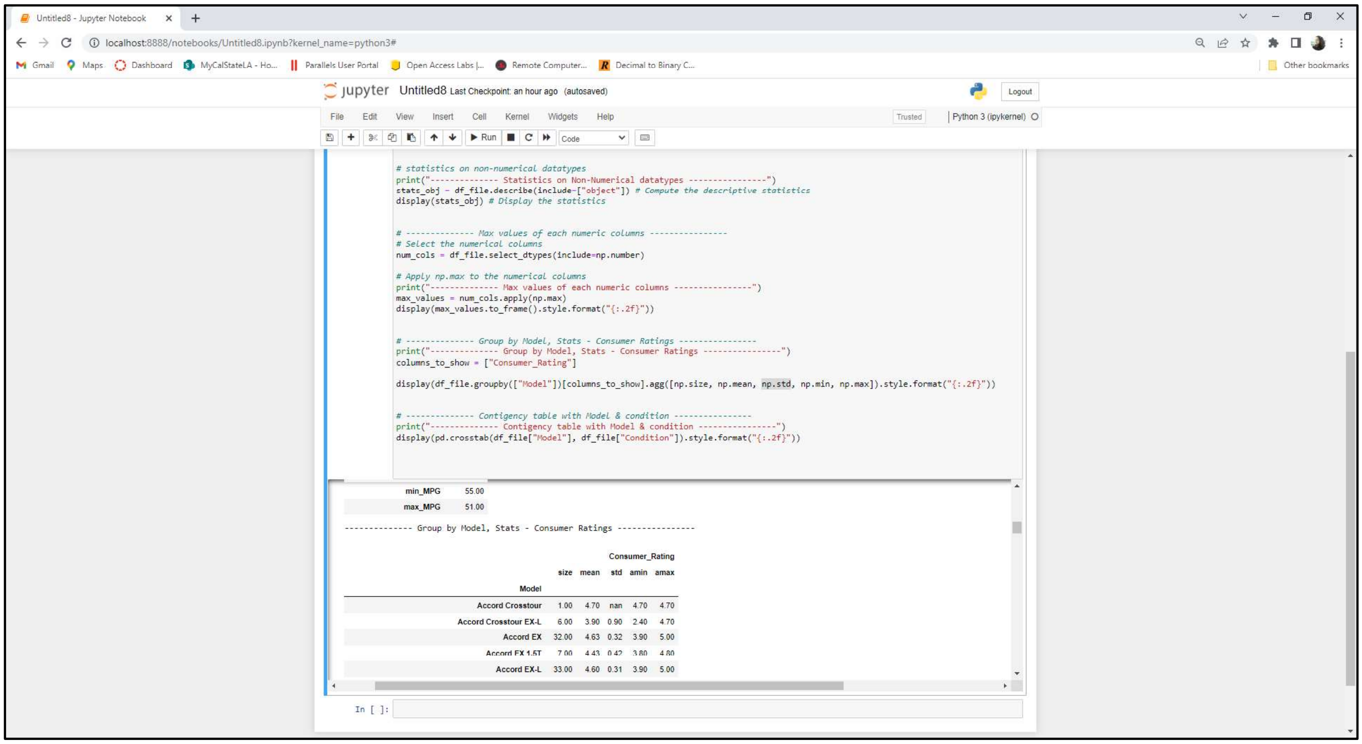


Odyssey EX-L	161.00	32206.43	9263.40	3950.00	42360.00
Odyssey EX-L w/Navigation/RES	6.00	32652.33	2556.62	28873.00	36998.00
Odyssey Elite	104.00	45148.16	7695.21	21849.00	56545.00
Odyssey LX	9.00	22926.44	3897.31	16660.00	29293.00
Odyssey SE	10.00	22543.10	2750.33	17690.00	27000.00
Odyssey Sport	45.00	43052.78	1027.05	40323.00	46988.00
Odyssey Touring	85.00	40879.32	9778.04	5495.00	49895.00
Odyssey Touring Elite	4.00	20314.75	3887.88	16495.00	25440.00
Passport EX-L	125.00	36524.99	5911.89	25696.00	49991.00
Passport Elite	91.00	43127.45	5796.02	31200.00	51150.00
Passport Sport	63.00	28842.13	2966.36	20999.00	38761.00
Passport Touring	44.00	34337.84	4170.46	24900.00	44995.00
Passport TrailSport	85.00	44282.56	2558.75	37000.00	51100.00
Pilot Black Edition	26.00	47528.00	6342.88	31787.00	53560.00
Pilot EX	32.00	25007.44	6324.21	8000.00	34900.00
Pilot EX-L	195.00	34696.82	8599.71	5799.00	47995.00
Pilot EX-L w/ Navigation	6.00	22429.67	3721.80	16499.00	25998.00
Pilot Elite	86.00	46867.37	9533.85	21562.00	56830.00
Pilot LX	11.00	24574.64	5625.00	11900.00	30890.00
Pilot Special Edition	132.00	42412.61	3473.80	22222.00	50595.00
Pilot Sport	227.00	41006.25	1823.33	33488.00	45773.00
Pilot Touring	67.00	42696.96	13140.19	5995.00	53350.00
Pilot Touring 7-Passenger	53.00	43509.36	5783.84	31499.00	48860.00
Pilot Touring 8-Passenger	31.00	39677.37	6594.81	27995.00	50350.00
Pilot TrailSport	150.00	47194.16	2801.99	41799.00	56372.00
Prelude	5.00	14596.20	2870.92	11995.00	18998.00
Ridgeline Black	3.00	44068.00	9086.70	31994.00	53915.00
Ridgeline Black Edition	68.00	45551.55	5652.40	26971.00	53345.00
Ridgeline RT	3.00	10048.33	4237.67	6195.00	15950.00
Ridgeline RTL	116.00	39362.03	7349.38	10995.00	47010.00
Ridgeline RTL-E	140.00	44028.91	5815.50	16995.00	53935.00
Ridgeline RTL-T	9.00	27632.67	3197.63	20990.00	33501.00
Ridgeline SE	3.00	24915.00	2818.44	22850.00	28900.00
Ridgeline Sport	33.00	31698.88	4466.28	20000.00	38977.00
S2000	6.00	30090.67	8565.37	17995.00	40991.00
S2000 Base	2.00	29500.00	10000.00	19500.00	39500.00



S2000 Base (M6)	3.00	29533.00	4741.37	22900.00	33700.00
del Sol Si	2.00	16150.00	650.00	15500.00	16800.00

Overall Screenshot:



```
# statistics on non-numerical datatypes
print("------------- Statistics on Non-Numerical datatypes ---------------")
stats_obj = df_file.describe(include=["object"]) # Compute the descriptive statistics
display(stats_obj) # Display the statistics


# ------------- Max values of each numeric columns ---------------
# Select the numerical columns
num_cols = df_file.select_dtypes(include=np.number)

# Apply np.max to the numerical columns
print("------------- Max values of each numeric columns ---------------")
max_values = num_cols.apply(np.max)
display(max_values.to_frame().style.format("{:.2f}"))


# ------------- Group by Model, Stats - Consumer Ratings ---------------
print("------------- Group by Model, Stats - Consumer Ratings ---------------")
columns_to_show = ["Consumer_Rating"]

display(df_file.groupby(["Model"])[columns_to_show].agg([np.size, np.mean, np.std, np.min, np.max]).style.format("{:.2f}"))


# ------------- Contigency table with Model & condition ---------------
print("------------- Contigency table with Model & condition ---------------")
display(pd.crosstab(df_file["Model"], df_file["Condition"]).style.format("{:.2f}"))
```

```
          min_MPG    55.00
          max_MPG    51.00

------------- Group by Model, Stats - Consumer Ratings ---------------

                                          Consumer_Rating
                                size   mean   std   amin   amax
                      Model
          Accord Crosstour      1.00   4.70   nan   4.70   4.70
     Accord Crosstour EX-L      6.00   3.90  0.90   2.40   4.70
                 Accord EX     32.00   4.63  0.32   3.90   5.00
            Accord EX 1.5T      7.00   4.43  0.42   3.80   4.80
               Accord EX-L     33.00   4.60  0.31   3.90   5.00
```

```
In [ ]:
```

This code is performing an analysis on a dataset. It groups the data by the "Model" column and calculates summary statistics for the "Price" column for each group.

The summary statistics calculated include the count, mean, standard deviation, minimum, and maximum values of the "Price" column for each group.

Groups with fewer than two observations are excluded from the results to avoid getting null output for std() functions.

The data provided includes the **name of the car model** and its **sub-model, number of cars sold, average price, standard deviation, minimum price,** and **maximum price**.

Here are some summary statistics:

- There are 37 **car models** in the dataset.

- The **number of cars sold per model** ranges from 2 to 255, with an average of 41.7 and a standard deviation of 60.5.
- The **average price** of the cars ranges from $8,996.20 to $40,383.62, with an overall **average** of **$26,267.27** and a **standard deviation** of $**6,726.28**.
- The **minimum** and **maximum prices** of the cars range from $1,995 to $45,395.
- The **most popular car model** in the dataset is the CR-V EX-L, with 255 cars sold.
- The **most expensive car model** in the dataset is the NSX, with an average price of $160,000 and a standard deviation of $0.
- The **least expensive car model** in the dataset is the Civic DX, with an average price of $8,740.90 and a standard deviation of $1,233.73.