HONDA CARS SALES DATA ANALYSIS USING PYTHON

AUTHORS

DHWANI VAISHNAV

MANIMOZHI NEETHINAYAGAM

CIS 5270 BUSINESS INTELLIGENCE PROJECT

SPRING 2023

Contents

A. Introduction

Founded in 1948, Honda Motor Co., Ltd. has emerged as a global leader in the automotive industry. Honda's unwavering commitment to quality, reliability, and innovation has propelled its success and earned the trust of millions of customers worldwide. With a rich heritage of engineering excellence and a passion for creating exceptional vehicles, Honda has consistently delivered cutting-edge solutions and redefined mobility.

The automotive industry has witnessed remarkable advancements over the years, and Honda, a renowned name in the field, has consistently set the bar high with its exceptional range of cars. Honda's commitment to excellence, innovation, and customer satisfaction has made it a preferred choice for car enthusiasts around the globe. Recognizing the importance of environmental sustainability, Honda has made significant strides in developing hybrid and electric vehicles.

In this comprehensive sales report analysis, weather you are a first-time car buyer or a seasoned automobile enthusiast, this guide will provide valuable insights into the diverse Honda lineup, highlighting their unique selling points and why they stand out in the competitive market.

A statistical summary provides a concise and organized presentation of the data related to Honda car sales. It includes key numerical measures such as total sales, average sales, sales by region, sales by model, and other relevant metrics. Understanding the correlation between consumer ratings, reviews, and pricing variations can support marketing and product differentiation strategies. Positive consumer ratings and reviews can be highlighted in marketing campaigns to build trust and credibility, while pricing variations based on transmission and powertrain can be used to differentiate Honda models from competitors.

By examining key metrics such as total sales, sales growth rates, market share, and regional sales distribution, stakeholders can assess the success and effectiveness of Honda's sales strategies. This evaluation provides insights into areas of strength, identifies opportunities for improvement, and helps refine future sales approaches [2]. Analyzing price variations based on transmission and powertrain helps Honda assess its competitive positioning in the market.

Buyers often consider mileage as an indicator of a used car's condition and potential future maintenance costs. By examining the relationship between mileage and prices, buyers can assess whether the asking price aligns with the expected wear and tear associated with the car's mileage. This analysis builds buyer confidence and transparency in the purchasing process. By understanding how mileage affects prices, sellers can make strategic decisions regarding when to sell a vehicle to maximize its value.

Studying the price spread across fuel types provides insights into market trends and shifts in consumer preferences. Changes in fuel prices and environmental consciousness can influence buyer behavior and the demand for certain fuel types. Analyzing the price spread over time helps identify any shifts in consumer preferences towards more fuel-efficient or environmentally friendly options.

Analyzing Honda car sales is essential for evaluating market performance, understanding consumer preferences, forecasting sales, managing inventory, assessing competition, refining marketing strategies, and enhancing customer satisfaction [1]. This analysis allows sellers and buyers to make informed decisions and ensures fair transactions in the Used or New car market.

B. Dataset URL and Dataset Description

URL: https://www.kaggle.com/datasets/omartorres25/honda-data

The dataset presents information about used Honda cars for sale in the United States.

The dataset includes details such as the car's make and model, year, mileage, engine type, transmission type, and fuel type. It also includes information on the car's price, location, and seller type, whether it's a dealership or a private seller.

The dataset contains around 10,000 rows of data and can be used to analyze the trends in used Honda car sales, pricing, and features across different regions in the US.

Column Name	Description	Sample Value
Year	The year the car was manufactured	2023
Make	The make or brand of the car	Honda
Model	The model's name or number of the car	CR-V Hybrid Sport
Condition	The overall condition of the car (New, Used)	New
Price	The price of the car in USD	$46,370
Consumer_Rating	The average rating given by consumers	4.8
Consumer_Review_#	The number of consumer reviews for the car	9
Exterior_Color	The color of the car's exterior	Platinum White Pearl
Interior_Color	The color of the car's interior	Beige

Drivetrain	The type of drivetrain the car has	All-wheel Drive
MPG	The average miles per gallon the car gets	19-25
Fuel_Type	The type of fuel the car uses	Gasoline
Transmission	The type of transmission the car has	Automatic
Engine	The size or type of engine the car has	2.0L I4 16V GDI DOHC Hybrid
VIN	The vehicle identification number	2HKRS5H5XPH702953
Stock_#	The stock number assigned by the seller	6402953
Mileage	The total number of miles the car has been driven	10
Comfort_Rating	The rating given by consumers for the car's comfort	5
Interior_Design_Rating	The rating given by consumers for the car's interior design	4.8
Performance_Rating	The rating given by consumers for the car's performance	4.8
Value_For_Money_Rating	The rating given by consumers for the car's value for money	4.2
Exterior_Styling_Rating	The rating given by consumers for the car's exterior styling	5

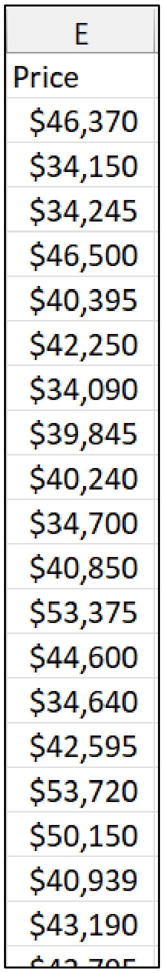
Reliability_Rating	The rating given by consumers for the car's reliability	5
State	The state where the car is located	CA
Seller_Type	The type of seller (e.g., Dealer, Individual)	Dealer

C. Data Cleaning

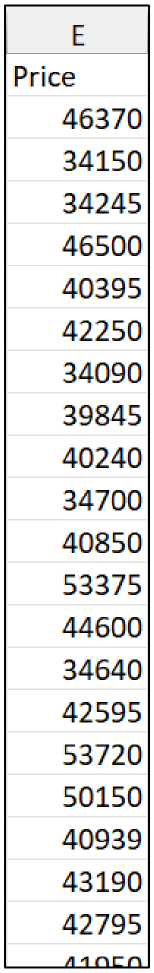
1. Handling Datatypes

Column: Price

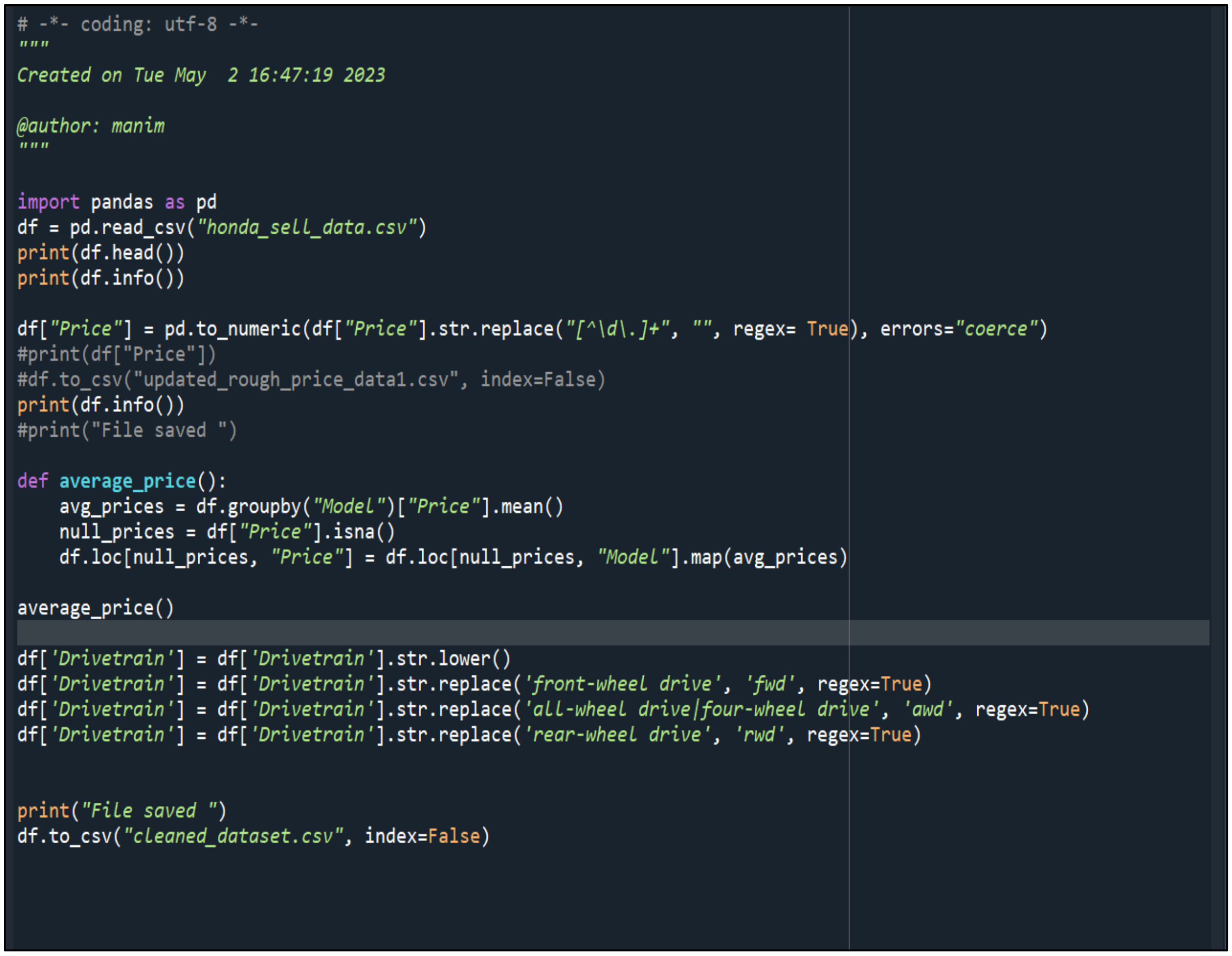
Before Cleaning: **After Cleaning:**



E
Price
$46,370
$34,150
$34,245
$46,500
$40,395
$42,250
$34,090
$39,845
$40,240
$34,700
$40,850
$53,375
$44,600
$34,640
$42,595
$53,720
$50,150
$40,939
$43,190



E
Price
46370
34150
34245
46500
40395
42250
34090
39845
40240
34700
40850
53375
44600
34640
42595
53720
50150
40939
43190
42795



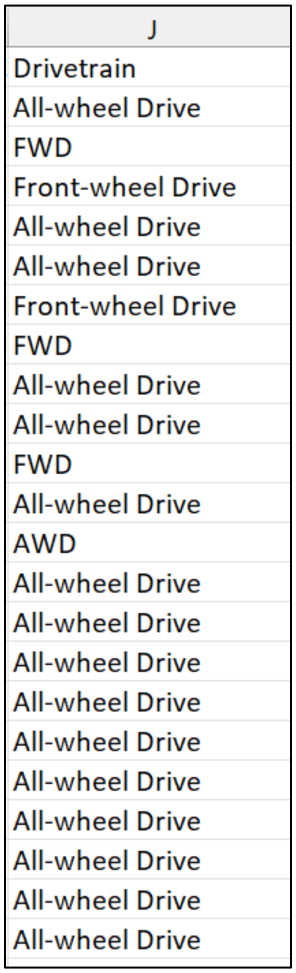
```python
# -*- coding: utf-8 -*-
"""
Created on Tue May  2 16:47:19 2023

@author: manim
"""

import pandas as pd
df = pd.read_csv("honda_sell_data.csv")
print(df.head())
print(df.info())

df["Price"] = pd.to_numeric(df["Price"].str.replace("[^\d\.]+", "", regex= True), errors="coerce")
#print(df["Price"])
#df.to_csv("updated_rough_price_data1.csv", index=False)
print(df.info())
#print("File saved ")

def average_price():
    avg_prices = df.groupby("Model")["Price"].mean()
    null_prices = df["Price"].isna()
    df.loc[null_prices, "Price"] = df.loc[null_prices, "Model"].map(avg_prices)

average_price()

df['Drivetrain'] = df['Drivetrain'].str.lower()
df['Drivetrain'] = df['Drivetrain'].str.replace('front-wheel drive', 'fwd', regex=True)
df['Drivetrain'] = df['Drivetrain'].str.replace('all-wheel drive|four-wheel drive', 'awd', regex=True)
df['Drivetrain'] = df['Drivetrain'].str.replace('rear-wheel drive', 'rwd', regex=True)


print("File saved ")
df.to_csv("cleaned_dataset.csv", index=False)
```

i) Functions Used: Handling Datatype, Applying Delimiters.

The code initially uses the **pd.to_numeric()** function to change the "Price" column in the dataframe from **a string to a numeric data type**. All non-numeric characters **(“,” “$”)** are eliminated from each text in the column using a regular expression. The errors option is configured to "coerce" the replacement of any non-numeric values with NaN.

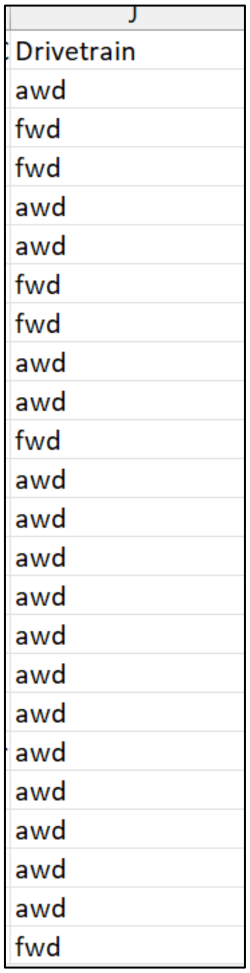
By grouping the data by "Model" and calculating the mean of the "Price" column, the average_price() method determines the average price of each model. The.isna() method is then used to find any rows with a "Price" value that is missing (NaN). Using the.map() method, it fills in the missing value for these rows with the average cost of the relevant model.

Before Cleaning:



J
Drivetrain
All-wheel Drive
FWD
Front-wheel Drive
All-wheel Drive
All-wheel Drive
Front-wheel Drive
FWD
All-wheel Drive
All-wheel Drive
FWD
All-wheel Drive
AWD
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive
All-wheel Drive

After Cleaning:



J
Drivetrain
awd
fwd
fwd
awd
awd
fwd
fwd
awd
awd
fwd
awd
awd
awd
awd
awd
awd
awd
awd
awd
awd
awd
awd
fwd

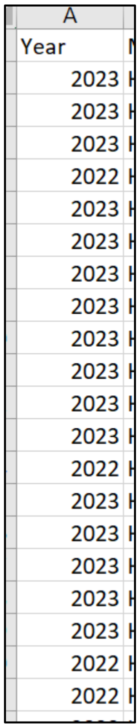
ii) Function Used: Group by, Replace, to lower.

The values in the dataframe's 'Drivetrain' column are being standardized and cleaned up by this code. All values in the column are being converted to lowercase in the first line's.**str.lower()** function.

The **str.replace() function** is used in the following three lines of code to replace certain values with predefined abbreviations. For instance, "front-wheel drive" is being replaced with "fwd", "all-wheel drive" and "four-wheel drive" are being replaced with "awd", and "rear-wheel drive" is being replaced with "rwd". This is done to make sure that values in the 'Drivetrain' column are consistent and won't confuse the analysis.

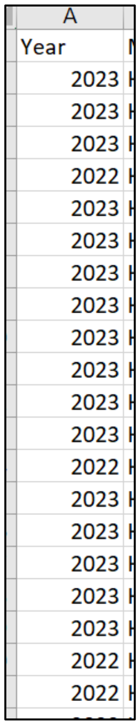
Column: Year

Before Cleaning:

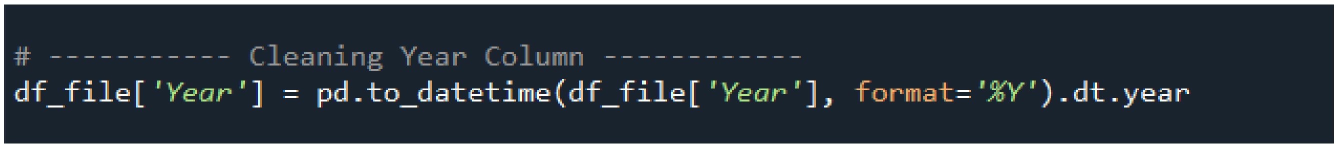


A
Year
2023
2023
2023
2022
2023
2023
2023
2023
2023
2023
2023
2023
2022
2023
2023
2023
2023
2023
2022
2022

After Cleaning:



A
Year
2023
2023
2023
2022
2023
2023
2023
2023
2023
2023
2023
2023
2022
2023
2023
2023
2023
2023
2022
2022



```
# ----------- Cleaning Year Column ------------
df_file['Year'] = pd.to_datetime(df_file['Year'], format='%Y').dt.year
```

Function Used: to_datetime()

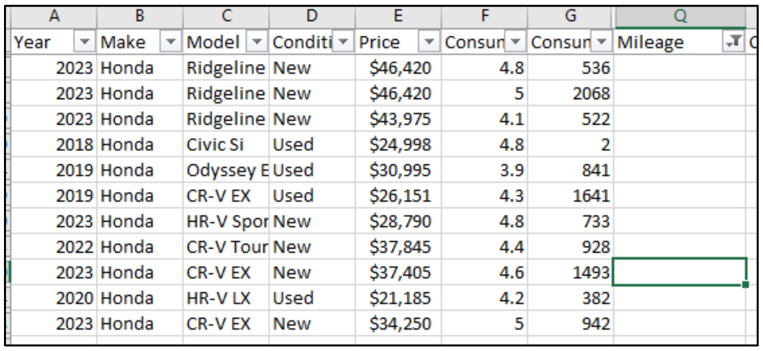
The data in column ‘Year’ is in String format. To use the ‘Year’ to plot the graphs, it needs to be converted to the ‘datetime’ format.

This code cleans the 'Year' column by converting the values in the column to datetime objects. The format parameter is used to specify the format of the date string in the 'Year' column, which is '%Y', indicating that the year is specified in 4-digit format.

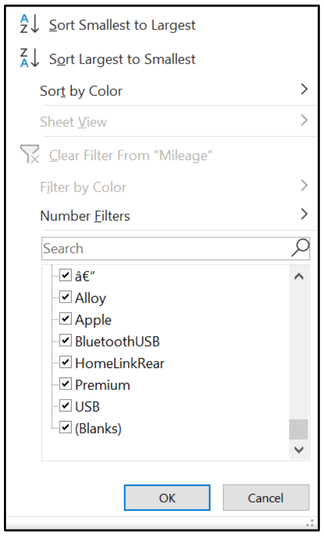
2. Handling Missing & Invalid Values

Column: Mileage

Before Cleaning:



Year	Make	Model	Conditi	Price	Consun	Consun	Mileage
2023	Honda	Ridgeline	New	$46,420	4.8	536	
2023	Honda	Ridgeline	New	$46,420	5	2068	
2023	Honda	Ridgeline	New	$43,975	4.1	522	
2018	Honda	Civic Si	Used	$24,998	4.8	2	
2019	Honda	Odyssey E	Used	$30,995	3.9	841	
2019	Honda	CR-V EX	Used	$26,151	4.3	1641	
2023	Honda	HR-V Spor	New	$28,790	4.8	733	
2022	Honda	CR-V Tour	New	$37,845	4.4	928	
2023	Honda	CR-V EX	New	$37,405	4.6	1493	
2020	Honda	HR-V LX	Used	$21,185	4.2	382	
2023	Honda	CR-V EX	New	$34,250	5	942	

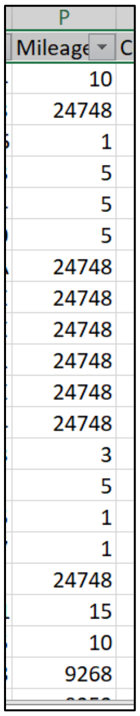


Sort Smallest to Largest
Sort Largest to Smallest
Sort by Color
Sheet View
Clear Filter From "Mileage"
Filter by Color
Number Filters
Search

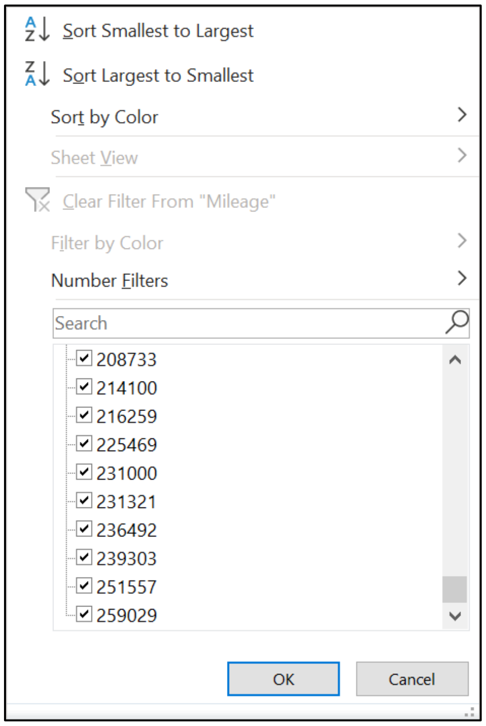
- ✓ â€"
- ✓ Alloy
- ✓ Apple
- ✓ BluetoothUSB
- ✓ HomeLinkRear
- ✓ Premium
- ✓ USB
- ✓ (Blanks)

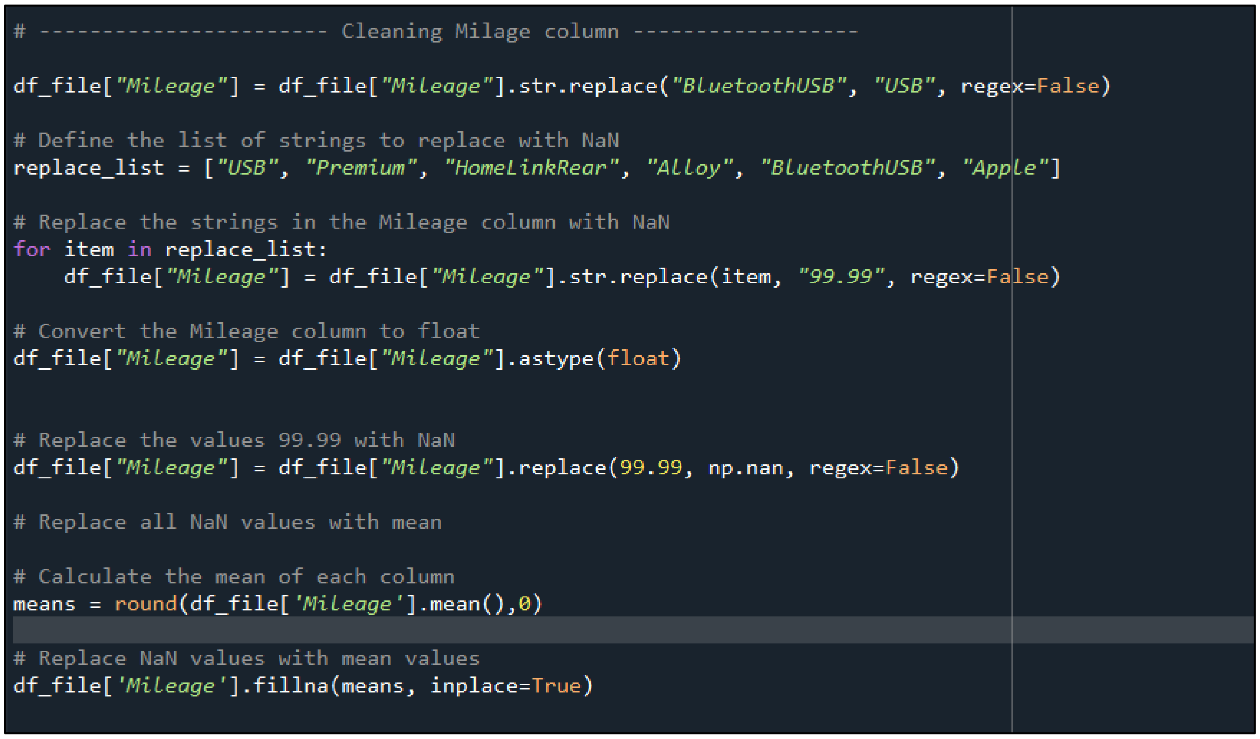
OK Cancel

After Cleaning:



P
Mileage
10
24748
1
5
5
5
24748
24748
24748
24748
24748
24748
3
5
1
1
24748
15
10
9268





```
# ----------------------- Cleaning Milage column ------------------

df_file["Mileage"] = df_file["Mileage"].str.replace("BluetoothUSB", "USB", regex=False)

# Define the list of strings to replace with NaN
replace_list = ["USB", "Premium", "HomeLinkRear", "Alloy", "BluetoothUSB", "Apple"]

# Replace the strings in the Mileage column with NaN
for item in replace_list:
    df_file["Mileage"] = df_file["Mileage"].str.replace(item, "99.99", regex=False)

# Convert the Mileage column to float
df_file["Mileage"] = df_file["Mileage"].astype(float)


# Replace the values 99.99 with NaN
df_file["Mileage"] = df_file["Mileage"].replace(99.99, np.nan, regex=False)

# Replace all NaN values with mean

# Calculate the mean of each column
means = round(df_file['Mileage'].mean(),0)

# Replace NaN values with mean values
df_file['Mileage'].fillna(means, inplace=True)
```

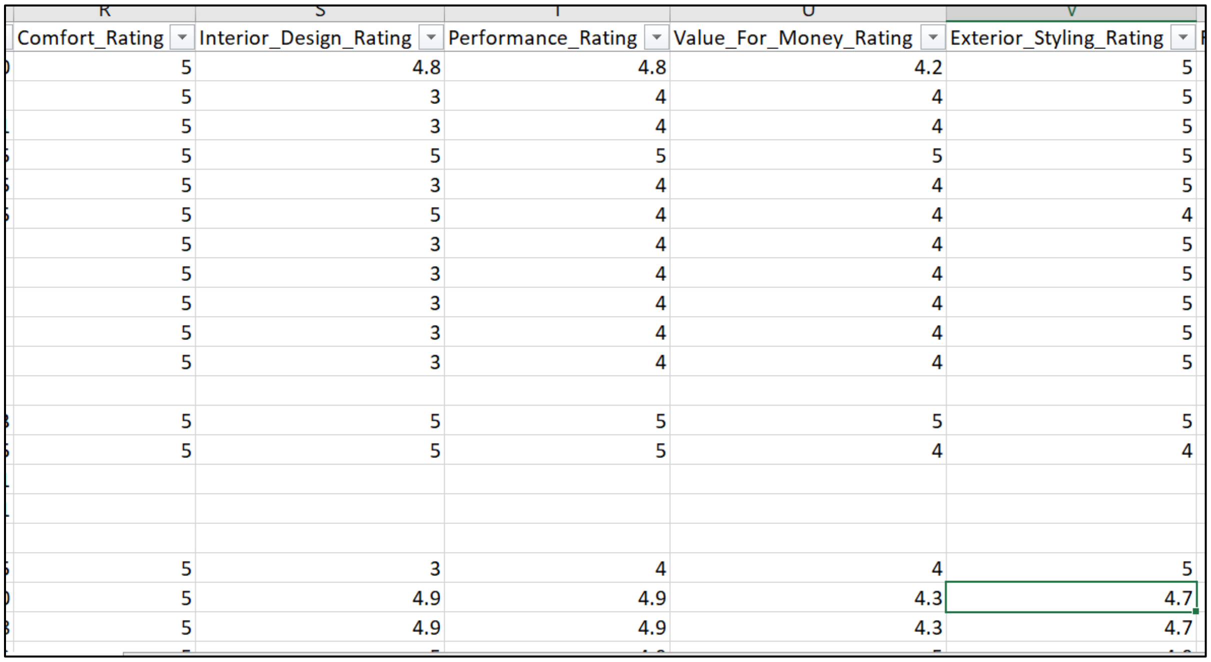
Functions Used: mean(), replace(), fillna()

The 'Mileage' column in the dataset contains missing values as well as non-numeric invalid values such as 'USB', 'Alloy', 'Apple', etc.

To clean this column, the invalid values are replaced with a value of 99.99 which is later converted to NaN. After that, the column is converted to a float data type. Finally, all the NaN values in the column are replaced with the mean mileage of all cars in the dataset using the mean() function.

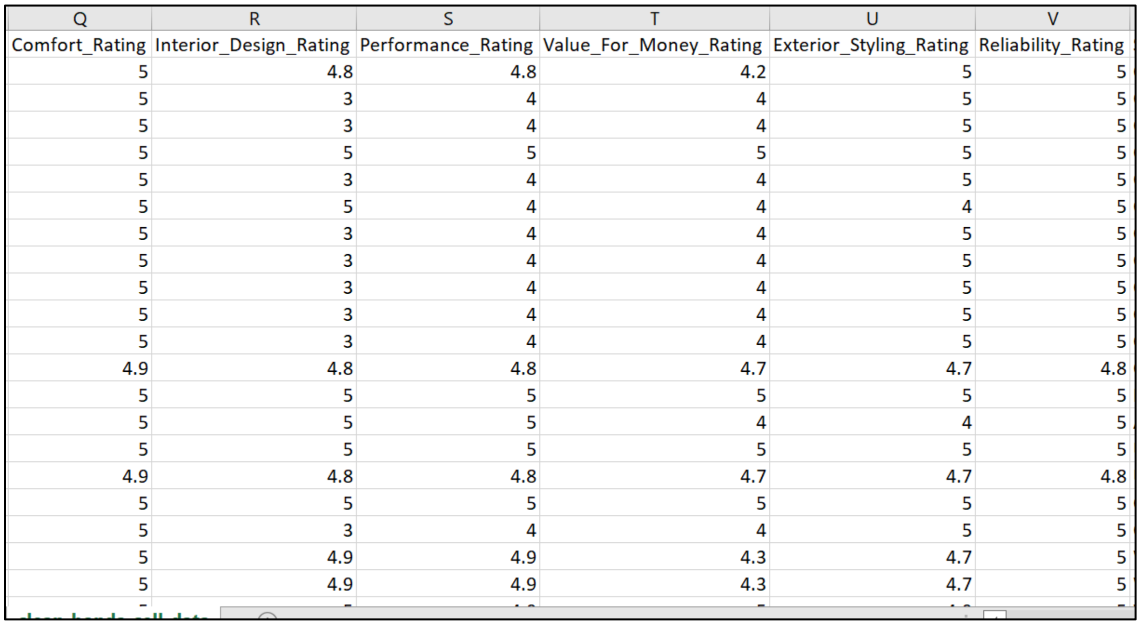
Columns: Comfort_Rating, Interior_Design_Rating, Performance_Rating, Value_For_Money_Rating, Exterior_Styling_Rating, Reliability_Rating

Before Cleaning:

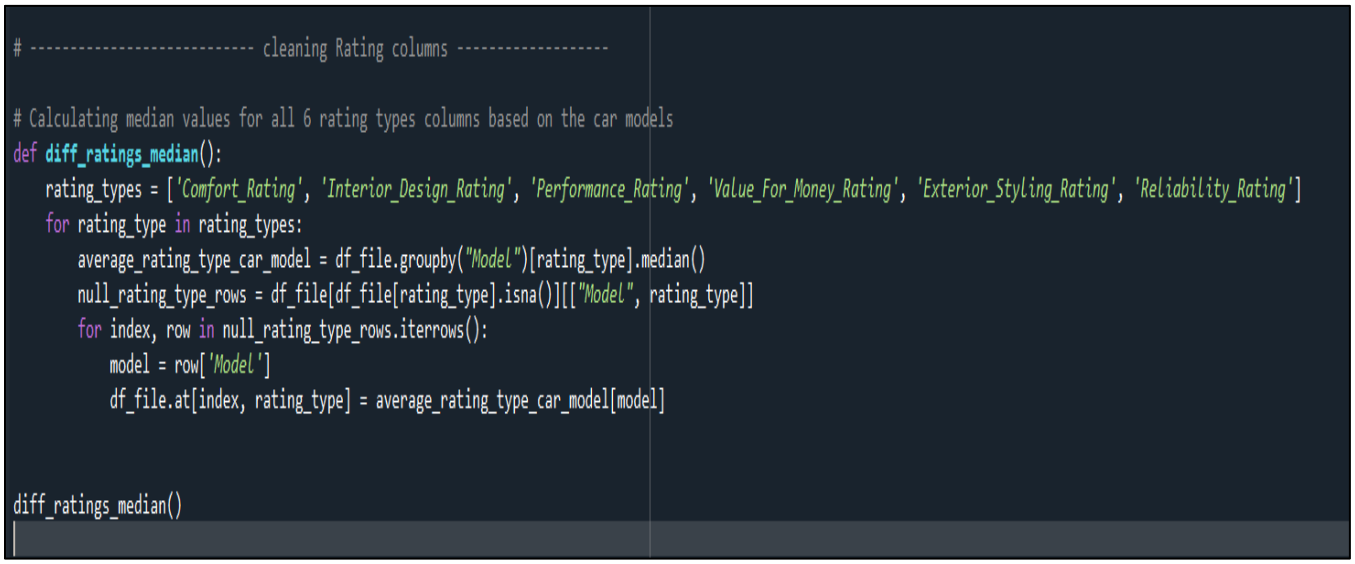


R	S	T	U	V
Comfort_Rating	Interior_Design_Rating	Performance_Rating	Value_For_Money_Rating	Exterior_Styling_Rating
5	4.8	4.8	4.2	5
5	3	4	4	5
5	3	4	4	5
5	5	5	5	5
5	3	4	4	5
5	5	4	4	4
5	3	4	4	5
5	3	4	4	5
5	3	4	4	5
5	3	4	4	5
5	3	4	4	5
5	5	5	5	5
5	5	5	4	4
5	3	4	4	5
5	4.9	4.9	4.3	4.7
5	4.9	4.9	4.3	4.7

After Cleaning:



Q	R	S	T	U	V
Comfort_Rating	Interior_Design_Rating	Performance_Rating	Value_For_Money_Rating	Exterior_Styling_Rating	Reliability_Rating
5	4.8	4.8	4.2	5	5
5	3	4	4	5	5
5	3	4	4	5	5
5	5	5	5	5	5
5	3	4	4	5	5
5	5	4	4	4	5
5	3	4	4	5	5
5	3	4	4	5	5
5	3	4	4	5	5
5	3	4	4	5	5
5	3	4	4	5	5
4.9	4.8	4.8	4.7	4.7	4.8
5	5	5	5	5	5
5	5	5	4	4	5
5	5	5	5	5	5
4.9	4.8	4.8	4.7	4.7	4.8
5	5	5	5	5	5
5	3	4	4	5	5
5	4.9	4.9	4.3	4.7	5
5	4.9	4.9	4.3	4.7	5



```python
# -------------------------- cleaning Rating columns -----------------

# Calculating median values for all 6 rating types columns based on the car models
def diff_ratings_median():
    rating_types = ['Comfort_Rating', 'Interior_Design_Rating', 'Performance_Rating', 'Value_For_Money_Rating', 'Exterior_Styling_Rating', 'Reliability_Rating']
    for rating_type in rating_types:
        average_rating_type_car_model = df_file.groupby("Model")[rating_type].median()
        null_rating_type_rows = df_file[df_file[rating_type].isna()][["Model", rating_type]]
        for index, row in null_rating_type_rows.iterrows():
            model = row['Model']
            df_file.at[index, rating_type] = average_rating_type_car_model[model]


diff_ratings_median()
```

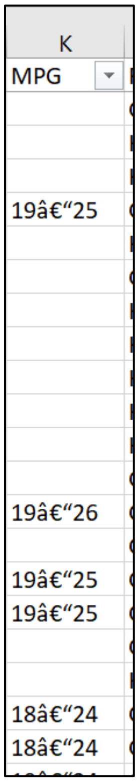
Functions Used: groupby(), median()

These different rating columns have many missing values; hence it is required to handle the missing values for these columns to make use of the data.

To handle the missing values, the “median” rating value is calculated for each car model using ‘Group By’ and all missing values are replaced by calculated median values for corresponding car models.

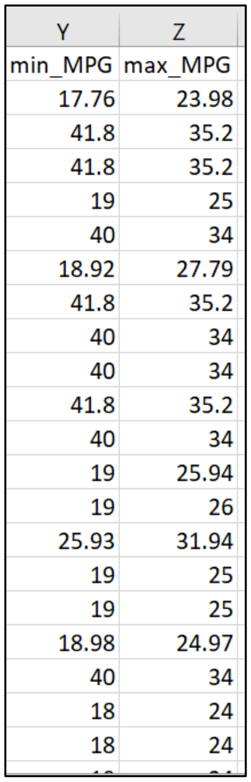
Column: MPG

Before Cleaning:



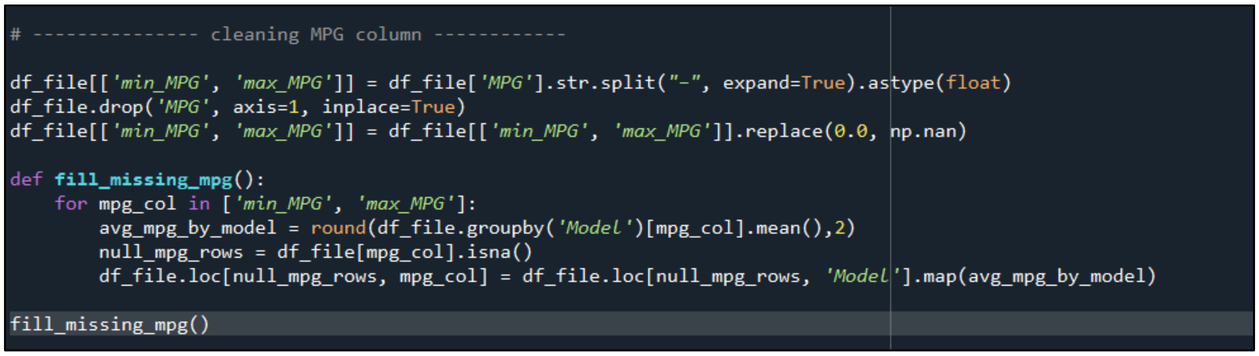
K
MPG
19â€“25
19â€“26
19â€“25
19â€“25
18â€“24
18â€“24

After Cleaning:



Y	Z
min_MPG	max_MPG
17.76	23.98
41.8	35.2
41.8	35.2
19	25
40	34
18.92	27.79
41.8	35.2
40	34
40	34
41.8	35.2
40	34
19	25.94
19	26
25.93	31.94
19	25
19	25
18.98	24.97
40	34
18	24
18	24

NOTE: â€“ is a “-”



```
# --------------- cleaning MPG column ------------

df_file[['min_MPG', 'max_MPG']] = df_file['MPG'].str.split("-", expand=True).astype(float)
df_file.drop('MPG', axis=1, inplace=True)
df_file[['min_MPG', 'max_MPG']] = df_file[['min_MPG', 'max_MPG']].replace(0.0, np.nan)

def fill_missing_mpg():
    for mpg_col in ['min_MPG', 'max_MPG']:
        avg_mpg_by_model = round(df_file.groupby('Model')[mpg_col].mean(),2)
        null_mpg_rows = df_file[mpg_col].isna()
        df_file.loc[null_mpg_rows, mpg_col] = df_file.loc[null_mpg_rows, 'Model'].map(avg_mpg_by_model)

fill_missing_mpg()
```

Functions Used: split(), astype(), replace(), round(), mean(), isna(), loc(), drop()

The 'MPG' column in the dataset represents the range of miles per gallon values for each car, separated by a hyphen (-). To prepare this column for analysis, it is necessary to split it into two separate columns: 'min_MPG' and 'max_MPG'.

Furthermore, the 'MPG' column contains many missing values that need to be addressed.

To clean the column, the 'MPG' column is split into 'min_MPG' and 'max_MPG' columns using a hyphen (-) as a separator. The original 'MPG' column is then removed from the dataset.

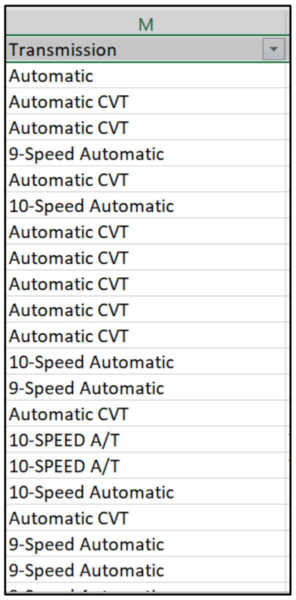
Next, any '0.0' values in the new columns are replaced with NaN to signify missing values.

To fill in the missing values, the code calculates the average MPG for each car model using the 'Group By' operation. If any MPG values are missing for a particular car model, the code fills those missing values with the calculated average MPG value specific to that car model.

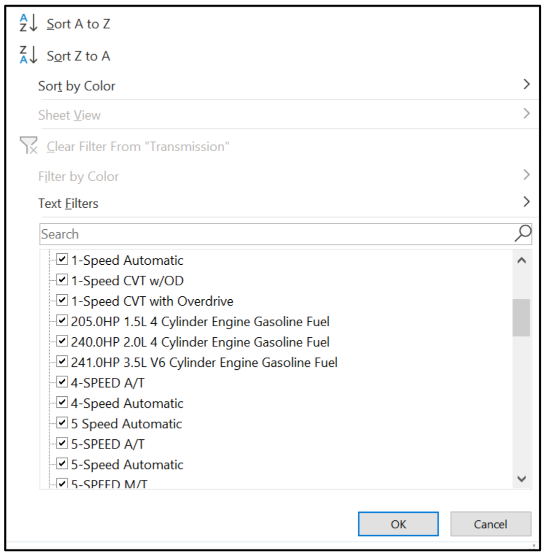
3. Data Standardization

Column: Transmission

Before Cleaning:

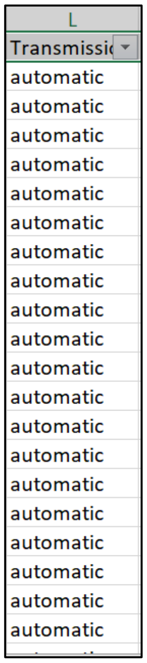


M
Transmission
Automatic
Automatic CVT
Automatic CVT
9-Speed Automatic
Automatic CVT
10-Speed Automatic
Automatic CVT
Automatic CVT
Automatic CVT
Automatic CVT
Automatic CVT
10-Speed Automatic
9-Speed Automatic
Automatic CVT
10-SPEED A/T
10-SPEED A/T
10-Speed Automatic
Automatic CVT
9-Speed Automatic
9-Speed Automatic

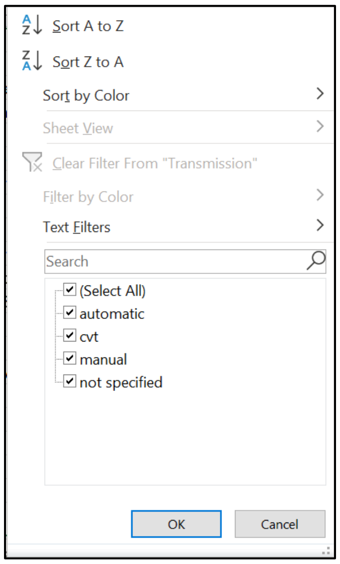


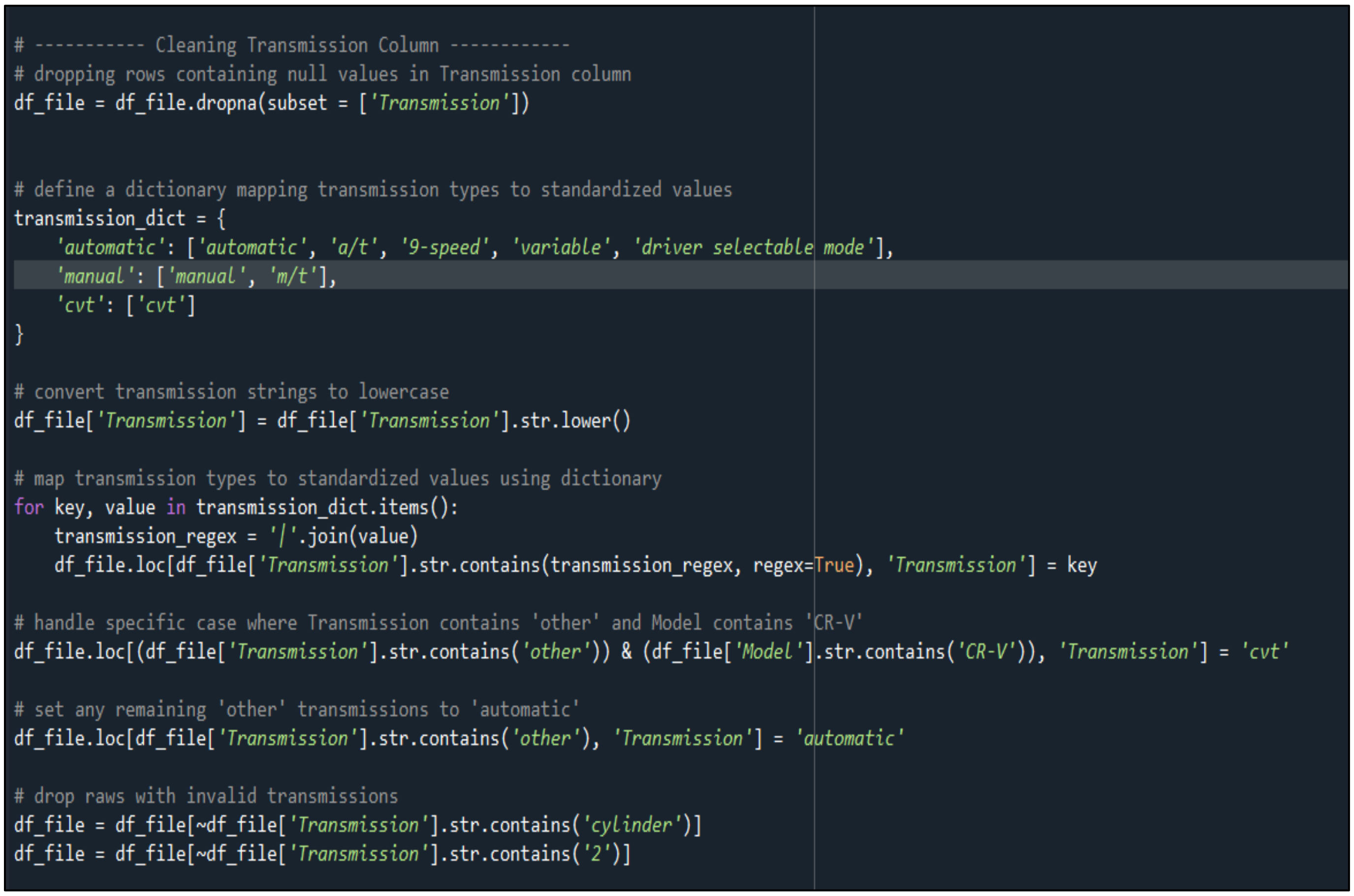
☑ 1-Speed Automatic
☑ 1-Speed CVT w/OD
☑ 1-Speed CVT with Overdrive
☑ 205.0HP 1.5L 4 Cylinder Engine Gasoline Fuel
☑ 240.0HP 2.0L 4 Cylinder Engine Gasoline Fuel
☑ 241.0HP 3.5L V6 Cylinder Engine Gasoline Fuel
☑ 4-SPEED A/T
☑ 4-Speed Automatic
☑ 5 Speed Automatic
☑ 5-SPEED A/T
☑ 5-Speed Automatic
☑ 5-SPEED M/T

After Cleaning:



L
Transmissic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic
automatic





```
# ----------- Cleaning Transmission Column ------------
# dropping rows containing null values in Transmission column
df_file = df_file.dropna(subset = ['Transmission'])


# define a dictionary mapping transmission types to standardized values
transmission_dict = {
    'automatic': ['automatic', 'a/t', '9-speed', 'variable', 'driver selectable mode'],
    'manual': ['manual', 'm/t'],
    'cvt': ['cvt']
}

# convert transmission strings to lowercase
df_file['Transmission'] = df_file['Transmission'].str.lower()

# map transmission types to standardized values using dictionary
for key, value in transmission_dict.items():
    transmission_regex = '|'.join(value)
    df_file.loc[df_file['Transmission'].str.contains(transmission_regex, regex=True), 'Transmission'] = key

# handle specific case where Transmission contains 'other' and Model contains 'CR-V'
df_file.loc[(df_file['Transmission'].str.contains('other')) & (df_file['Model'].str.contains('CR-V')), 'Transmission'] = 'cvt'

# set any remaining 'other' transmissions to 'automatic'
df_file.loc[df_file['Transmission'].str.contains('other'), 'Transmission'] = 'automatic'

# drop raws with invalid transmissions
df_file = df_file[~df_file['Transmission'].str.contains('cylinder')]
df_file = df_file[~df_file['Transmission'].str.contains('2')]
```

Functions Used: lower(), contains(), loc()

The 'Transmission' column of the dataset contains values that refer to similar types of transmissions, but in different variations. To make it easier to group similar types and gain insights, the values are standardized by replacing them with appropriate labels.

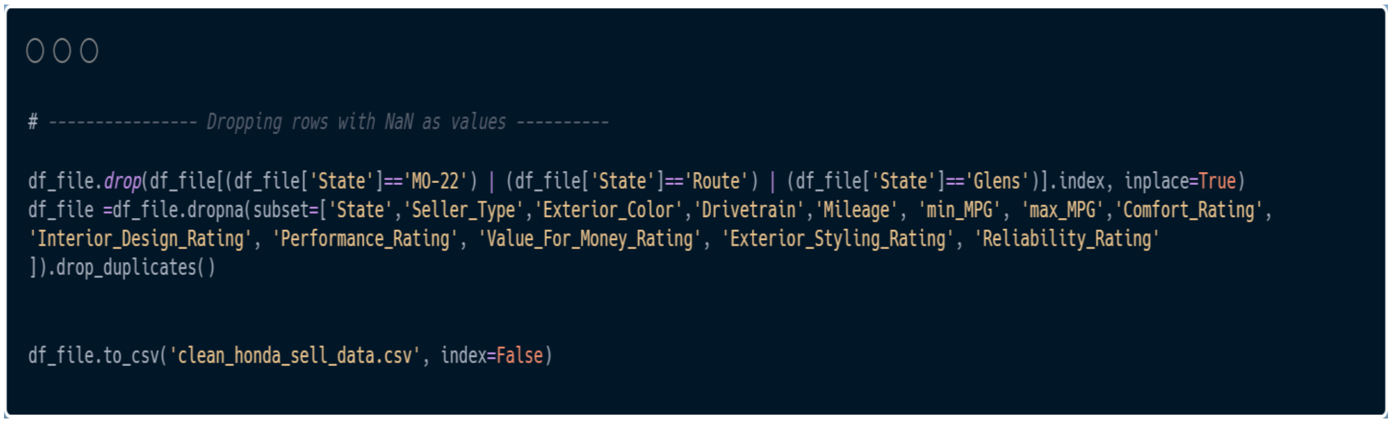
The values that contain 'automatic', 'a/t', '9-speed', 'variable', or 'driver selectable mode' are replaced by 'automatic', since they all belong to automatic transmission. Similarly, values that contain 'manual' and 'm/t' are replaced by 'manual', since they all belong to manual transmission.

In some cases, the 'Transmission' column contains the value 'Others'. For all models except CR-V, 'Others' is replaced with 'automatic'. For CR-V, it is replaced with 'cvt' because Honda offers CVT in several car models, including Civic, Accord, Insight, HR-V, CR-V, Fit, and Clarity. The data doesn't contain 'Others' in any of these models except CR-V.

Moreover, the 'Transmission' column also contains invalid values that don't fall into transmission categories, such as values like ‘141.0HP 1.8L 4 Cylinder Engine Gasoline Fuel’ or '2'. These rows have been dropped from the dataset.

Standardizing the 'Transmission' column makes it possible to plot the proper visualization and gain insights into the data.

Note: Since the piece of code is too long for spyder window, and the screenshot won’t help due to very small size of fonts, we have pasted this code to ‘https://carbon.now.sh’, to beautify the code and pasted here.



```
# ---------------- Dropping rows with NaN as values ----------

df_file.drop(df_file[(df_file['State']=='MO-22') | (df_file['State']=='Route') | (df_file['State']=='Glens')].index, inplace=True)
df_file =df_file.dropna(subset=['State','Seller_Type','Exterior_Color','Drivetrain','Mileage', 'min_MPG', 'max_MPG','Comfort_Rating',
'Interior_Design_Rating', 'Performance_Rating', 'Value_For_Money_Rating', 'Exterior_Styling_Rating', 'Reliability_Rating'
]).drop_duplicates()


df_file.to_csv('clean_honda_sell_data.csv', index=False)
```

Functions Used: drop, dropna()

The above code drops rows from the DataFrame where the 'State' column value is 'MO-22', 'Route', or 'Glens'. This step helps to remove any irrelevant or erroneous data from the dataset.Then, it removes rows that have missing values (NaN) in specific columns such as 'State', 'Seller_Type', 'Exterior_Color', 'Drivetrain', 'Mileage', 'min_MPG', 'max_MPG', 'Comfort_Rating', 'Interior_Design_Rating', 'Performance_Rating', 'Value_For_Money_Rating', 'Exterior_Styling_Rating', and 'Reliability_Rating'. By dropping these rows, the code ensures that only rows with complete information in these essential columns are retained.

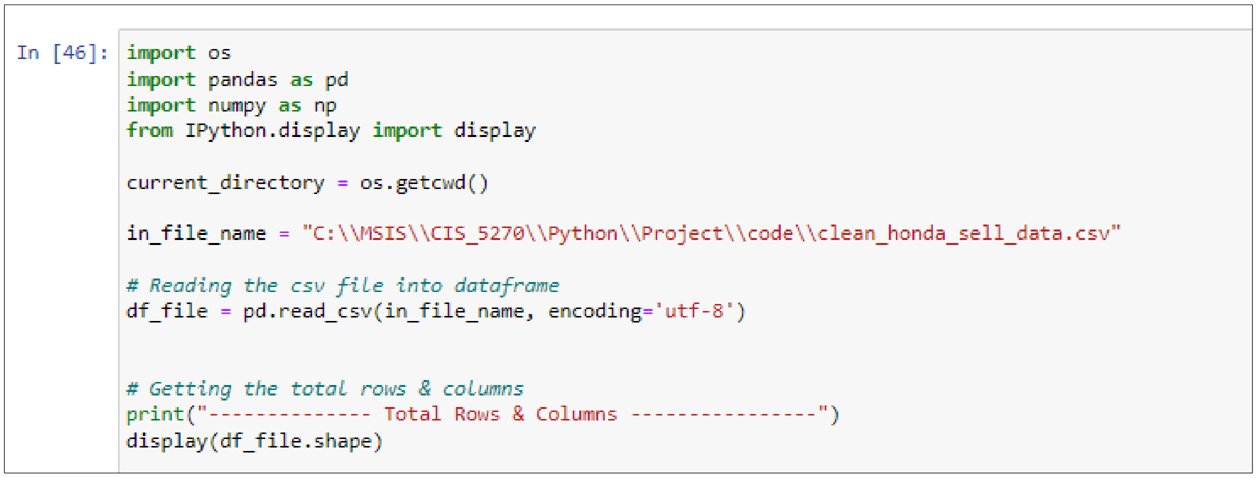
Later, after removing all the irrelevant values in the dataset, it eliminates any duplicate rows from the DataFrame, ensuring that each row in the dataset is unique.

The cleaned DataFrame is saved to a CSV file named 'clean_honda_sell_data.csv'. The resulting CSV file will contain the cleaned data, excluding the index column.

D. Statistical Summary

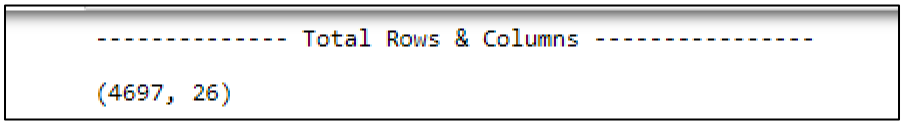
1. Getting the Total Rows & Columns

Functions used: pandas -> shape()



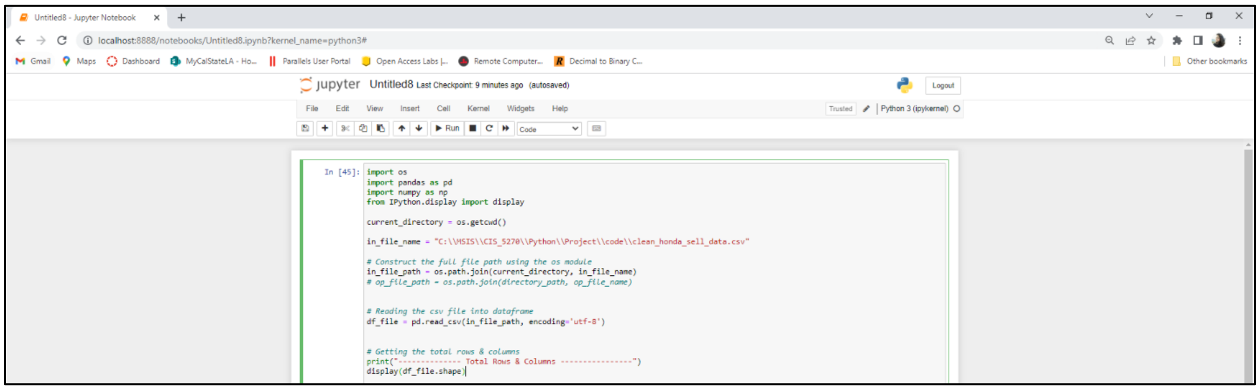
```
In [46]: import os
         import pandas as pd
         import numpy as np
         from IPython.display import display

         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:



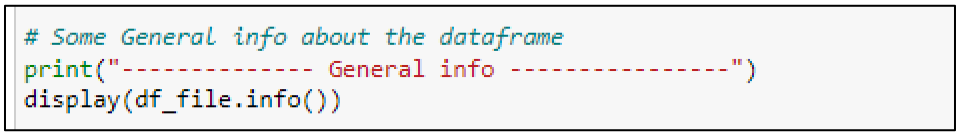
```
In [45]: import os
         import pandas as pd
         import numpy as np
         from IPython.display import display

         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.

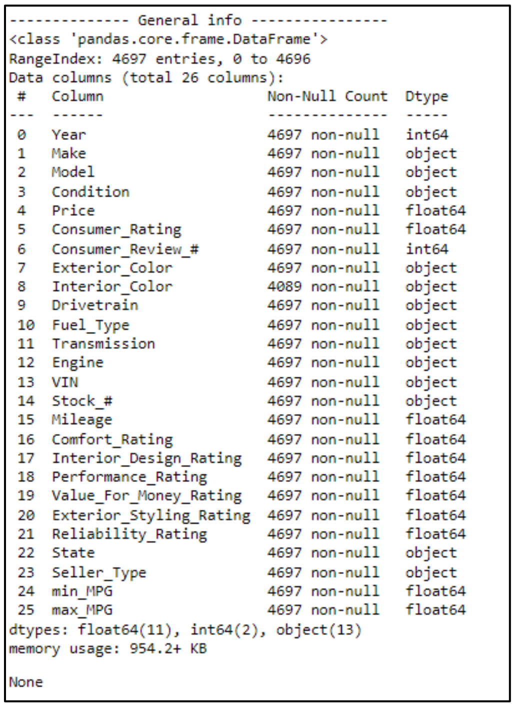
2. Getting concise Summary of a Data Frame

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_#.

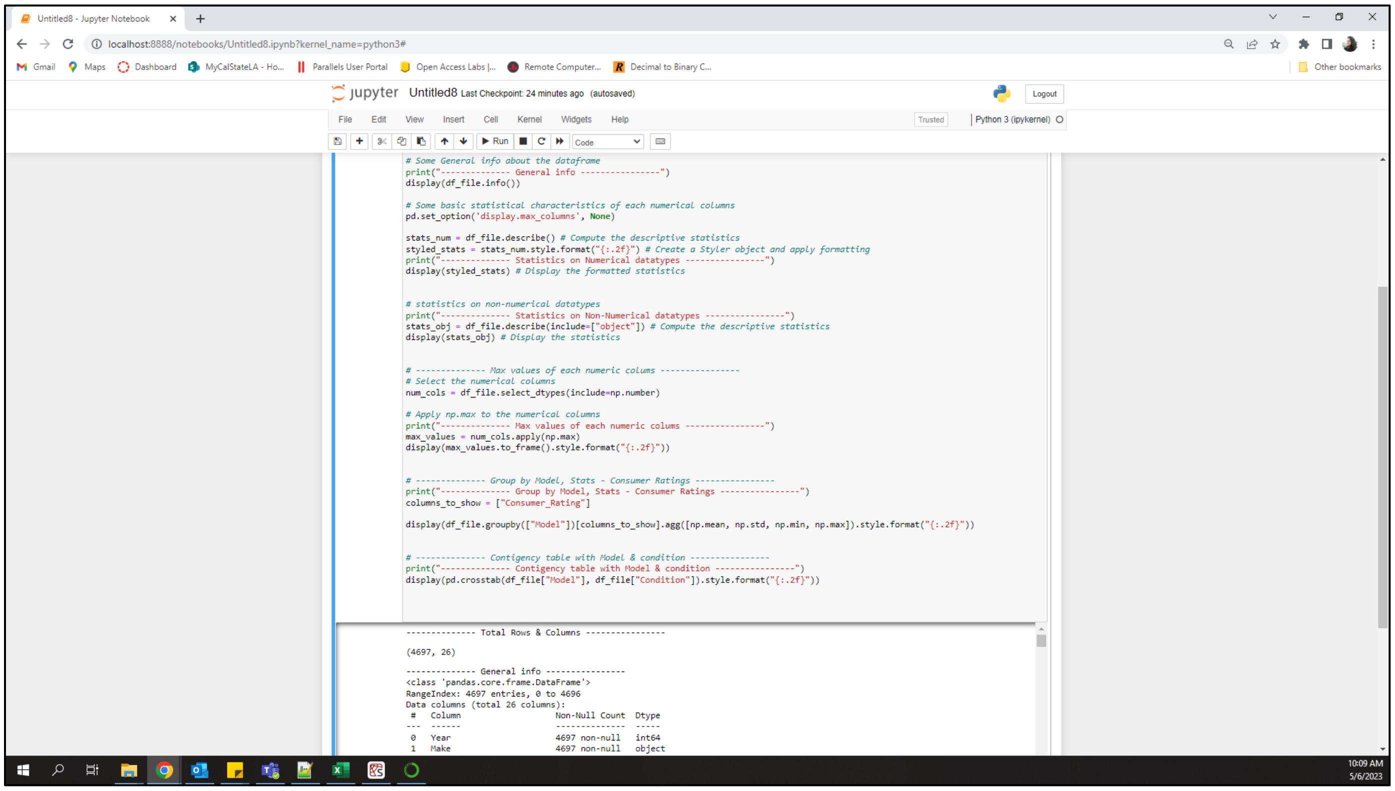
Float64: Price, Consumer_Rating,Milage, Comfort_Rating, Interior_Design_Rating, Performance_Rating, Value_For_Money_Rating, Exterior_Styling_Rating, Reliablity_Rating.

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:

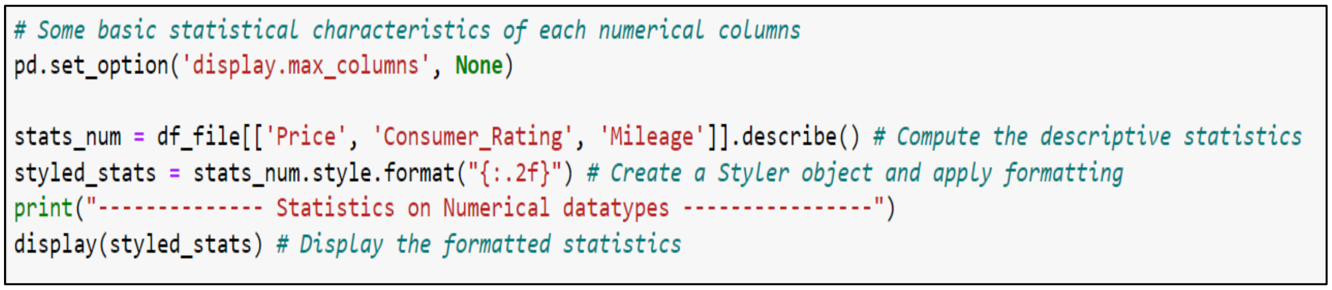

Untitled8 - Jupyter Notebook
localhost:8888/notebooks/Untitled8.ipynb?kernel_name=python3#
jupyter Untitled8 Last Checkpoint: 24 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
# 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
```

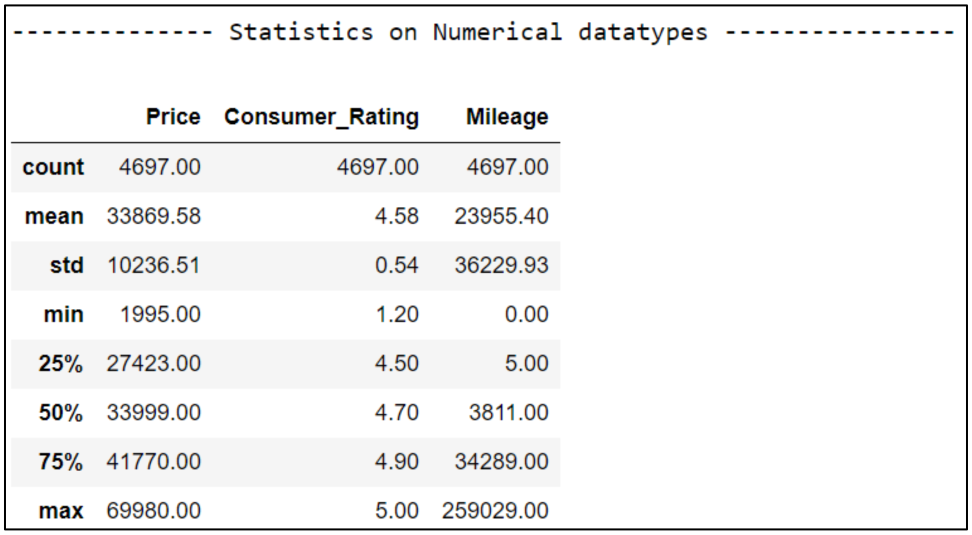
3. Some basic statistical characteristics of each numerical columns

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



```
# 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.

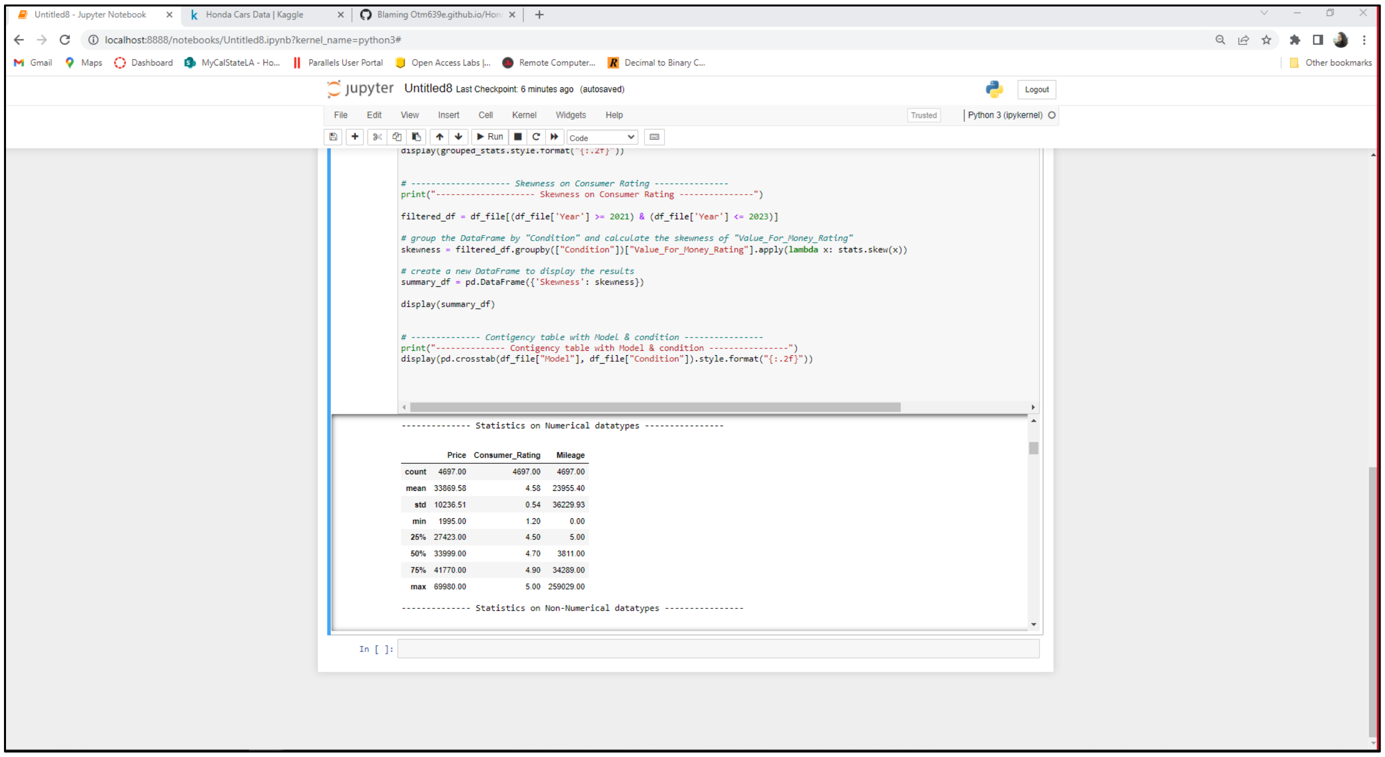
The **columns** in which we performed the summary statics are **Price, Consumer_Rating, Mileage.**

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

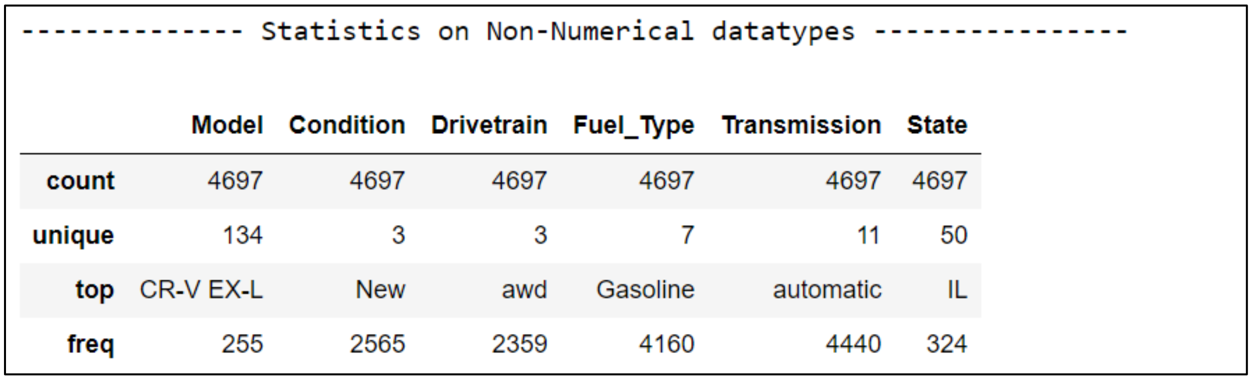
```
------------- Statistics on Non-Numerical datatypes ---------------
```

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

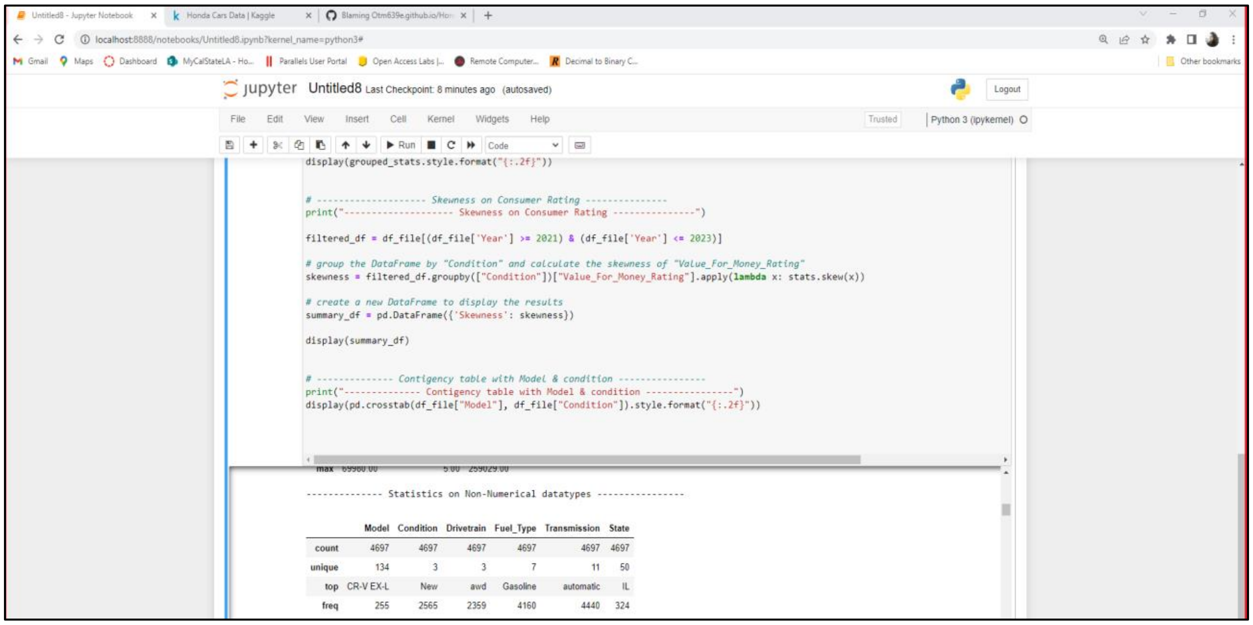
The above code displays the descriptive statistics of six columns in a pandas Data Frame **df_file**.

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:



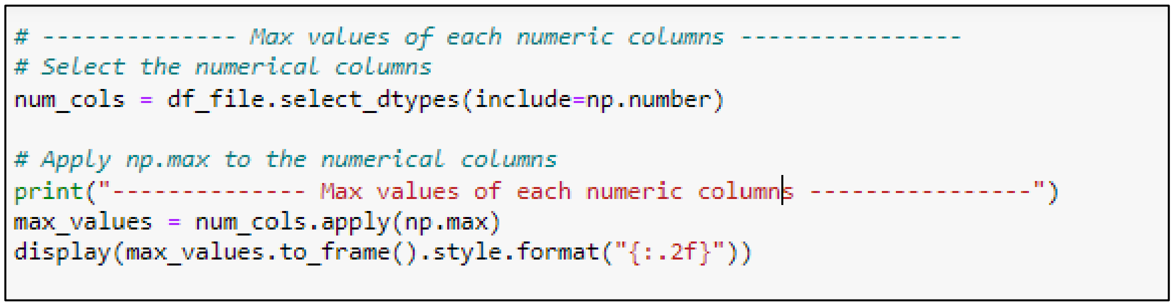
```
display(grouped_stats.style.format("{:.2f}"))


# ------------------- 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

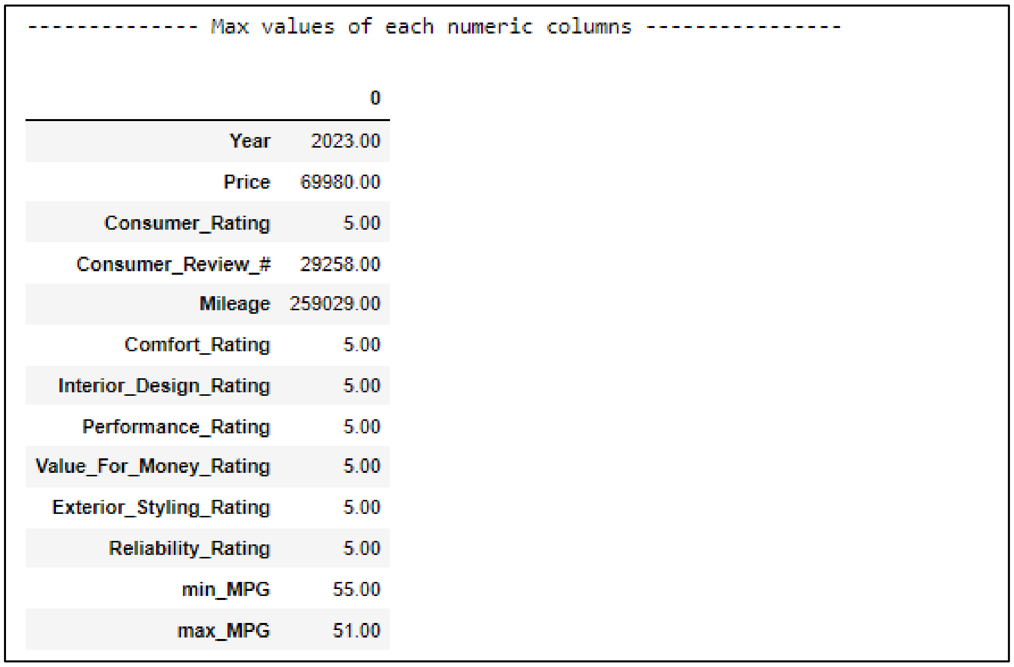
5. Showing Max values of each numeric columns

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}"))
```

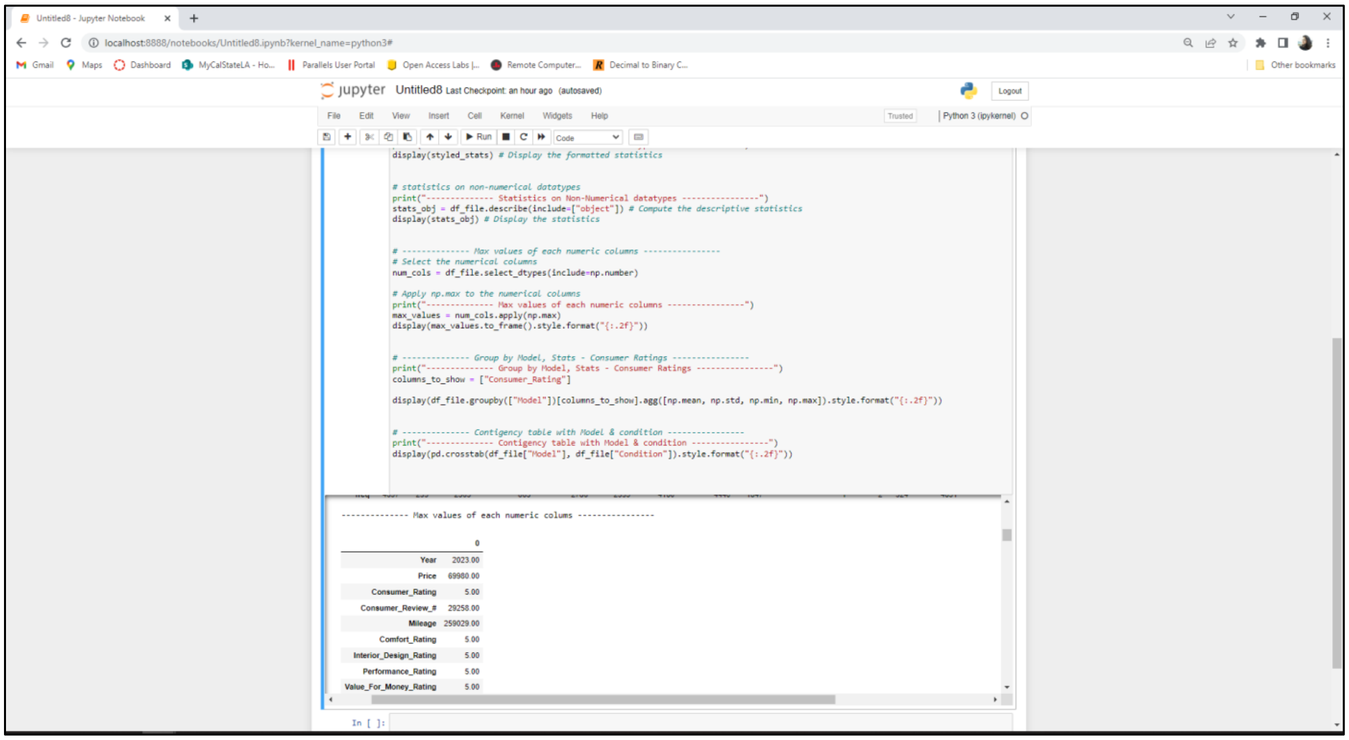
Output:



```
-------------- 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
```

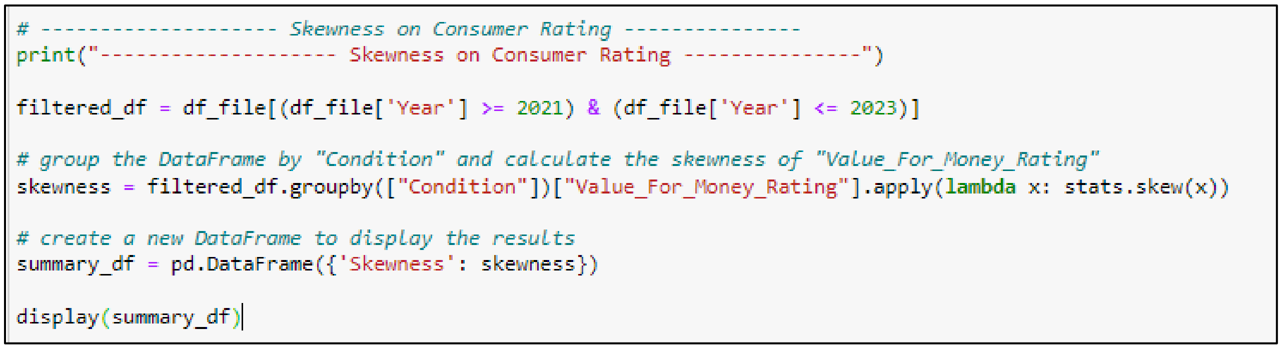
This code performs an analysis on a dataset, specifically looking at the maximum value of each column that contains numerical data. Np.max() function is applied to each of the numerical columns, which finds the maximum value in each column.

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

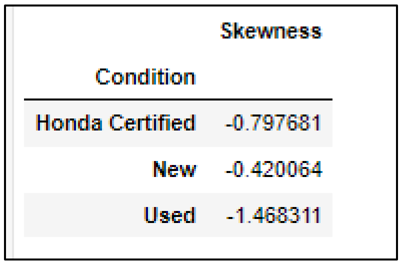
6. Skewness of Consumer ratings on ‘Value for money Ratings’ for last 3 years based on Condition (Used/New/Certified)

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



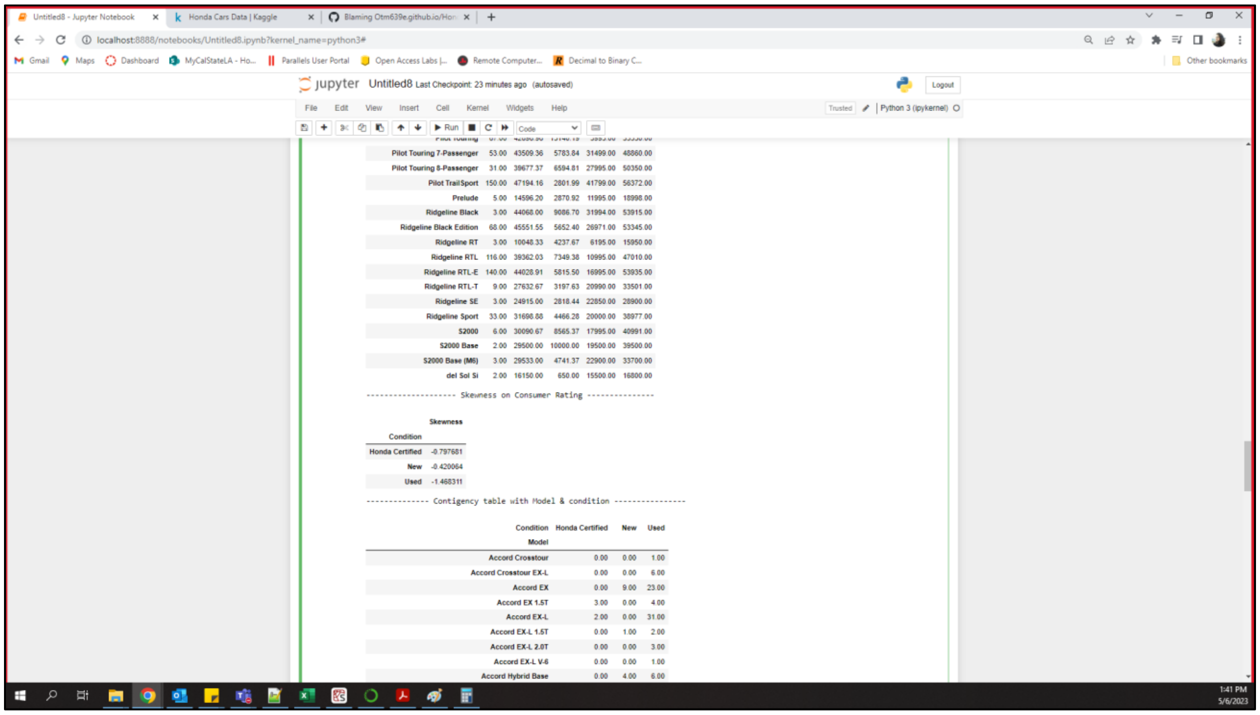
```python
# -------------------- Skewness on Consumer Rating ---------------
print("-------------------- Skewness on Consumer Rating ---------------")

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:



jupyter Untitled8 Last Checkpoint: 23 minutes ago (autosaved)

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

```
-------------------- 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".

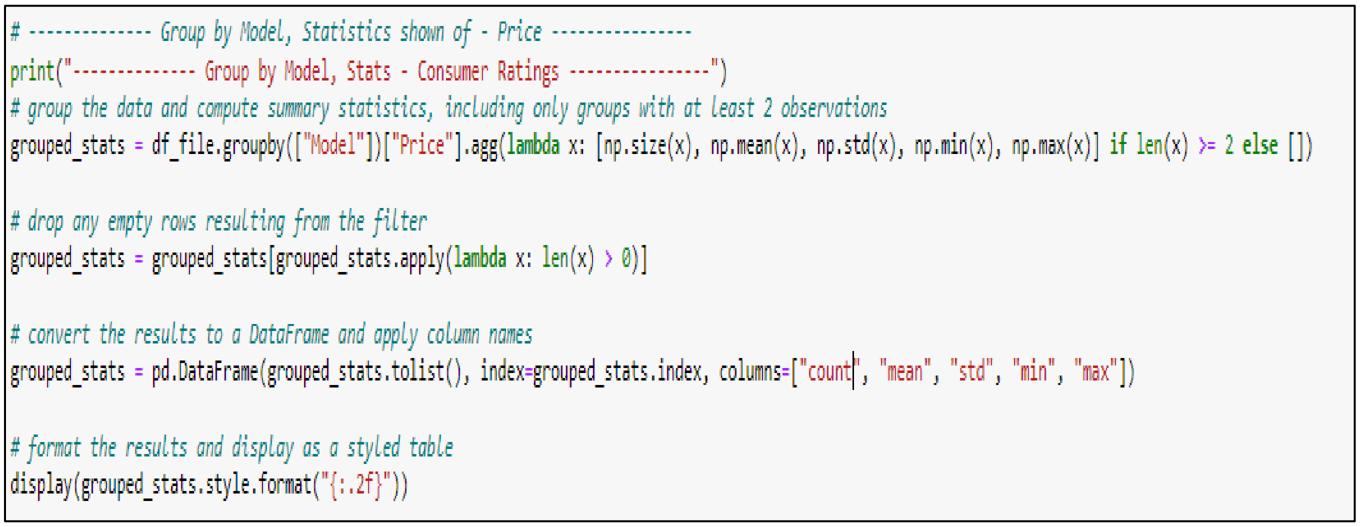
For the "Honda Certified" group, the skewness value is -0.797681. This suggests that the distribution of the "Value_For_Money_Rating" values for this group is slightly skewed to the left, meaning that there are more ratings on the higher end of the scale.

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.

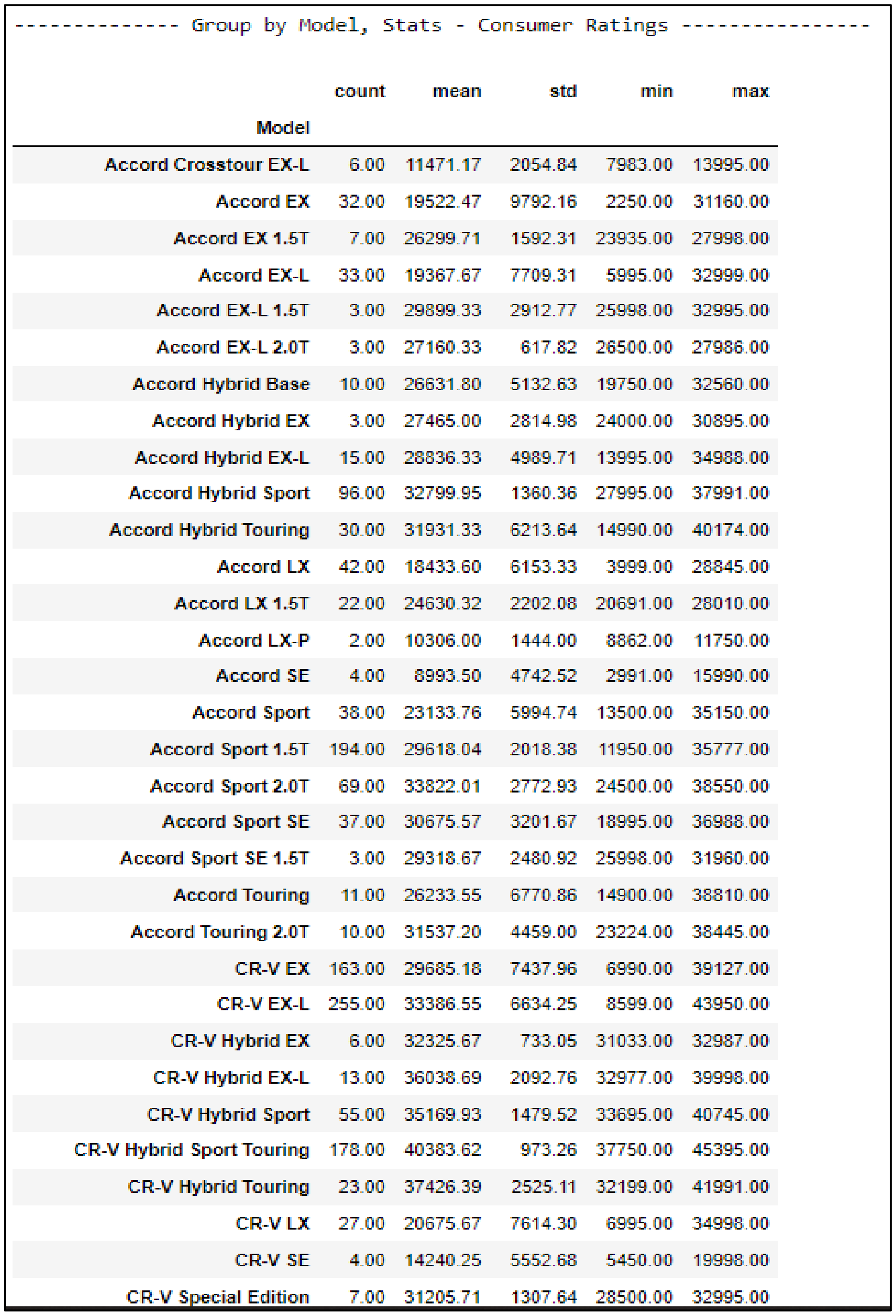
7. Statistics Summary for Price for each Car Model

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}"))
```

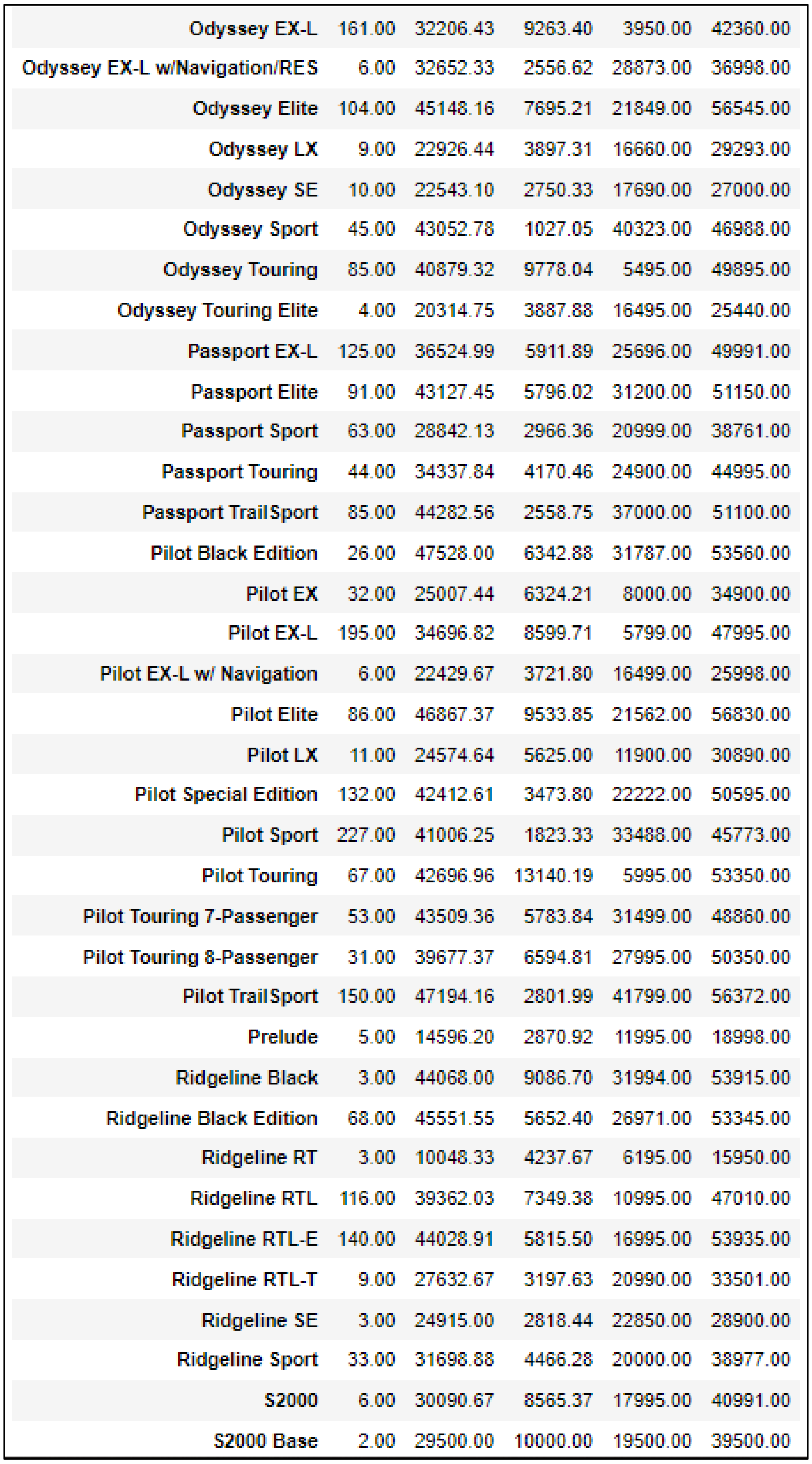
Output:



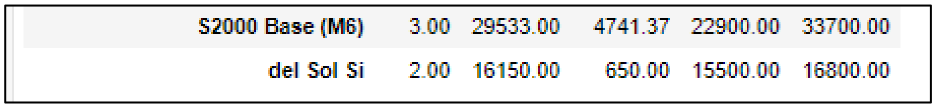
```
-------------- Group by Model, Stats - Consumer Ratings ----------------
```

Model	count	mean	std	min	max
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

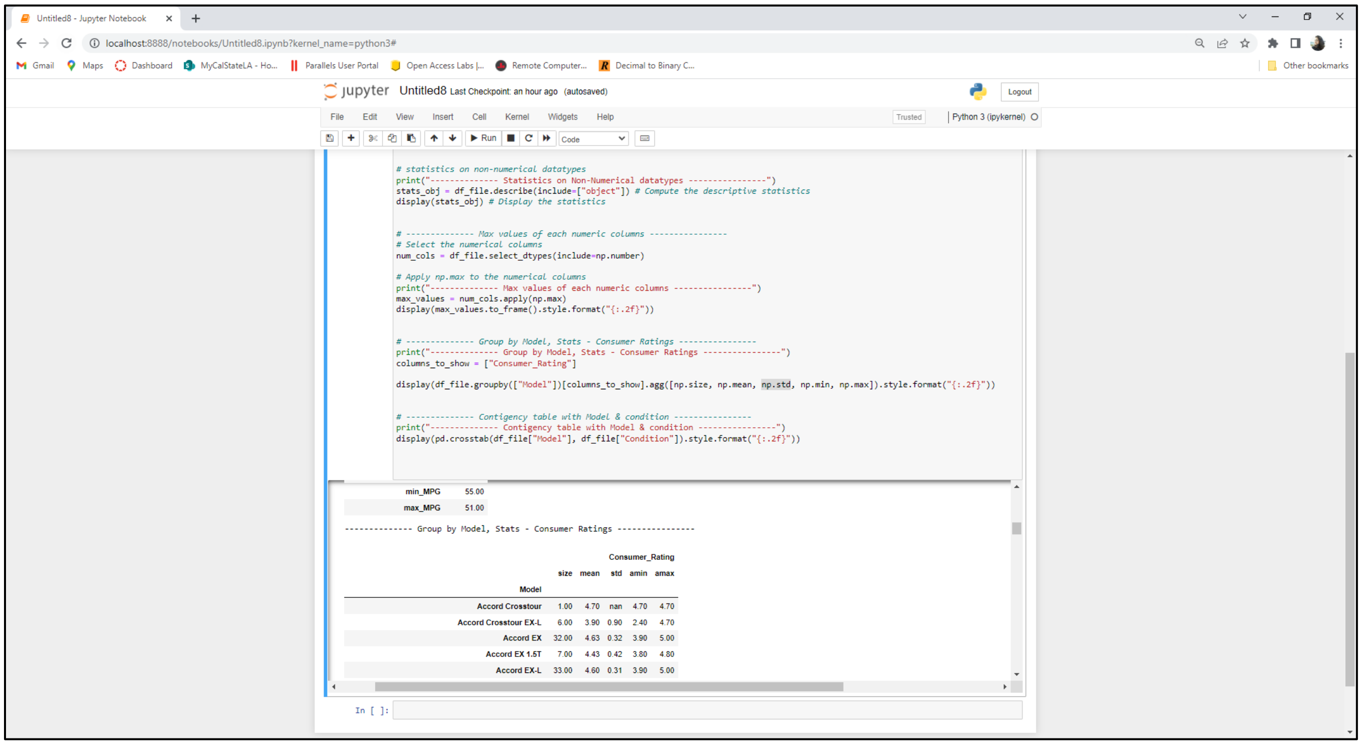


Odyssey EX-L	161.00	32206.43	9263.40	3950.00	42360.00
Odyssey EX-L w/Navigation/RES	6.00	32652.33	2556.62	28873.00	36998.00
Odyssey Elite	104.00	45148.16	7695.21	21849.00	56545.00
Odyssey LX	9.00	22926.44	3897.31	16660.00	29293.00
Odyssey SE	10.00	22543.10	2750.33	17690.00	27000.00
Odyssey Sport	45.00	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**.

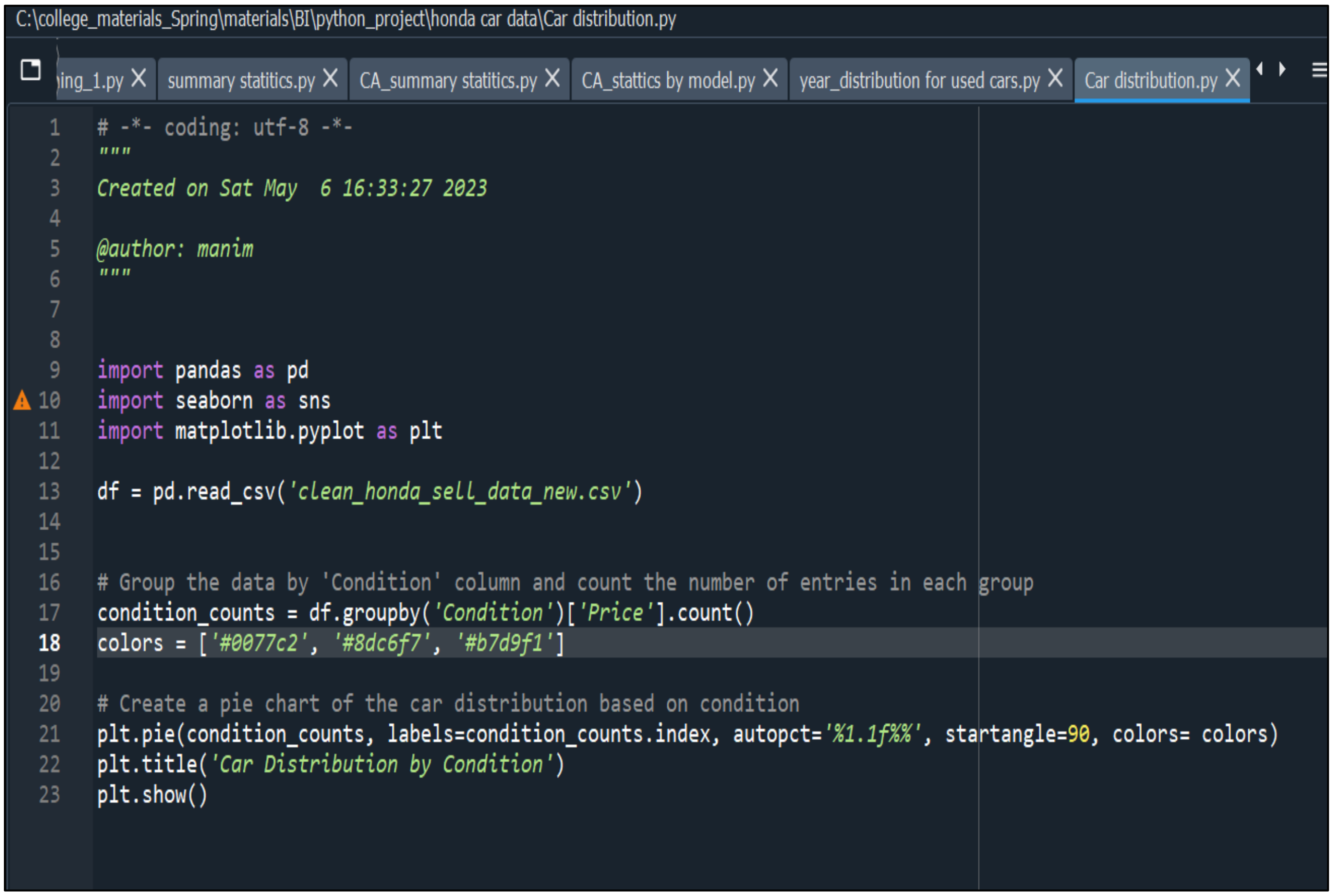
Here are some summary statistics:

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

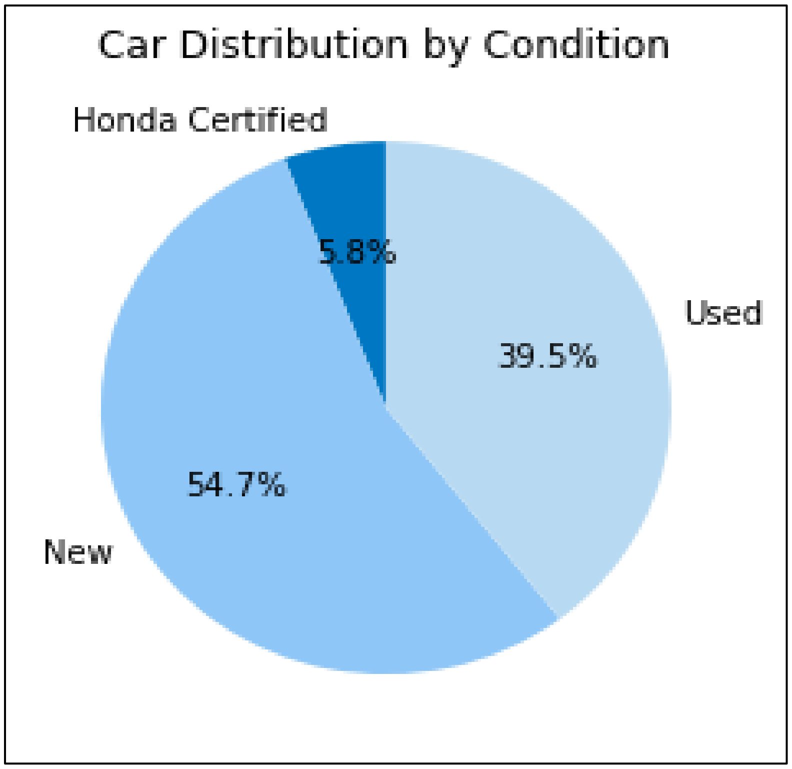
- The **number of cars sold per model** ranges from 2 to 255, with an average of 41.7 and a standard deviation of 60.5.
- The **average price** of the cars ranges from $8,996.20 to $40,383.62, with an overall **average** of **$26,267.27** and a **standard deviation** of $**6,726.28**.
- The **minimum** 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.

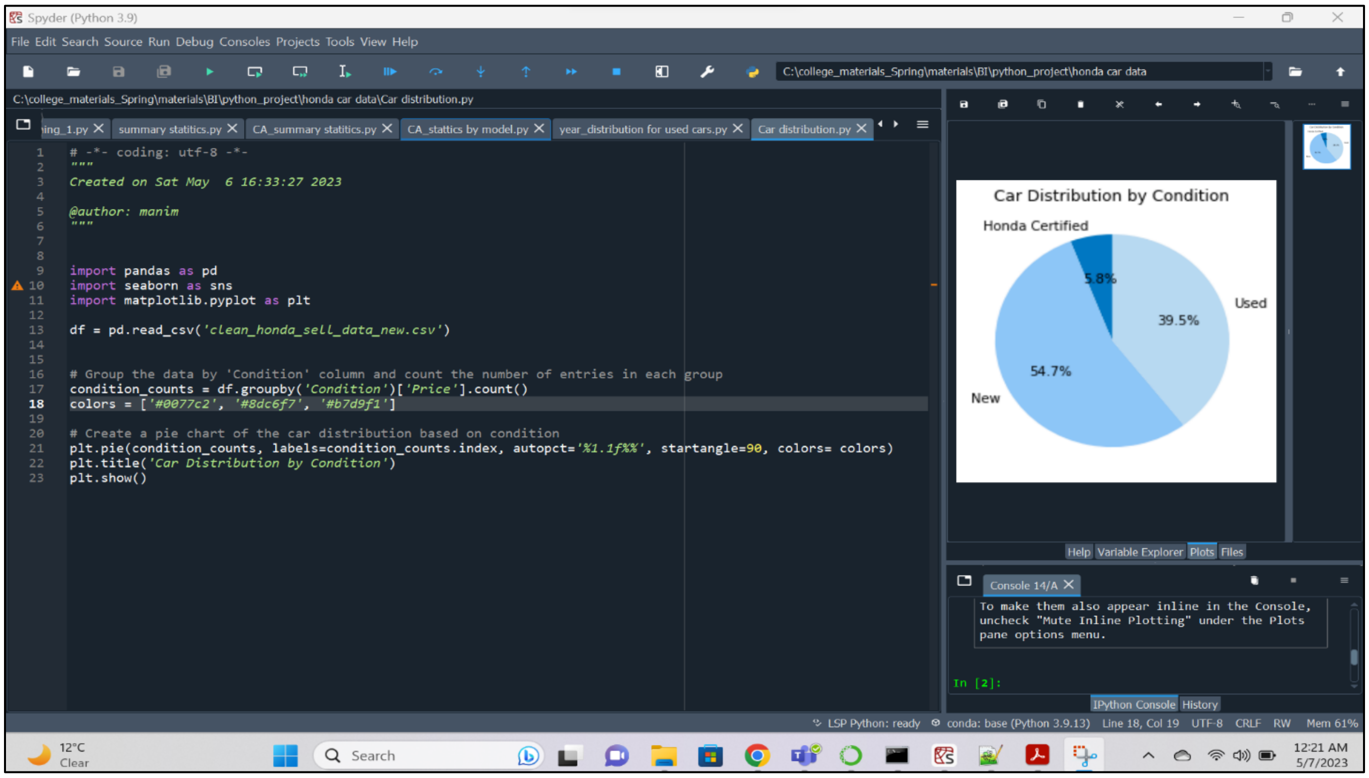
E. Visualizations

1. Show the car distribution based on condition (New, used, certified)


C:\college_materials_Spring\materials\BI\python_project\honda car data\Car distribution.py
ing_1.py
summary statitics.py
CA_summary statitics.py
CA_stattics by model.py
year_distribution for used cars.py
Car distribution.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 16:33:27 2023

@author: manim
"""


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

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


# Group the data by 'Condition' column and count the number of entries in each group
condition_counts = df.groupby('Condition')['Price'].count()
colors = ['#0077c2', '#8dc6f7', '#b7d9f1']

# Create a pie chart of the car distribution based on condition
plt.pie(condition_counts, labels=condition_counts.index, autopct='%1.1f%%', startangle=90, colors= colors)
plt.title('Car Distribution by Condition')
plt.show()
```




Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\college_materials_Spring\materials\BI\python_project\honda car data
C:\college_materials_Spring\materials\BI\python_project\honda car data\Car distribution.py
ing_1.py
summary statitics.py
CA_summary statitics.py
CA_stattics by model.py
year_distribution for used cars.py
Car distribution.py
Car Distribution by Condition
Honda Certified
5.8%
Used
39.5%
54.7%
New
Help
Variable Explorer
Plots
Files
Console 14/A
IPython Console
History
LSP Python: ready
conda: base (Python 3.9.13)
Line 18, Col 19
UTF-8
CRLF
RW
Mem 61%
12°C
Clear
Search
12:21 AM
5/7/2023

```python
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 16:33:27 2023

@author: manim
"""


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

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


# Group the data by 'Condition' column and count the number of entries in each group
condition_counts = df.groupby('Condition')['Price'].count()
colors = ['#0077c2', '#8dc6f7', '#b7d9f1']

# Create a pie chart of the car distribution based on condition
plt.pie(condition_counts, labels=condition_counts.index, autopct='%1.1f%%', startangle=90, colors= colors)
plt.title('Car Distribution by Condition')
plt.show()
```

```
To make them also appear inline in the Console,
uncheck "Mute Inline Plotting" under the Plots
pane options menu.

In [2]:
```

Insights:

The analysis provides a distribution of cars based on their condition. The three conditions being considered are 'New', 'Used', and 'Honda Certified'.

Most of the cars in the dataset are in the 'New' condition, accounting for 54.7% of the total cars. 'Used' cars account for 39.5%, and 'Honda Certified ' cars account for 5.8%.

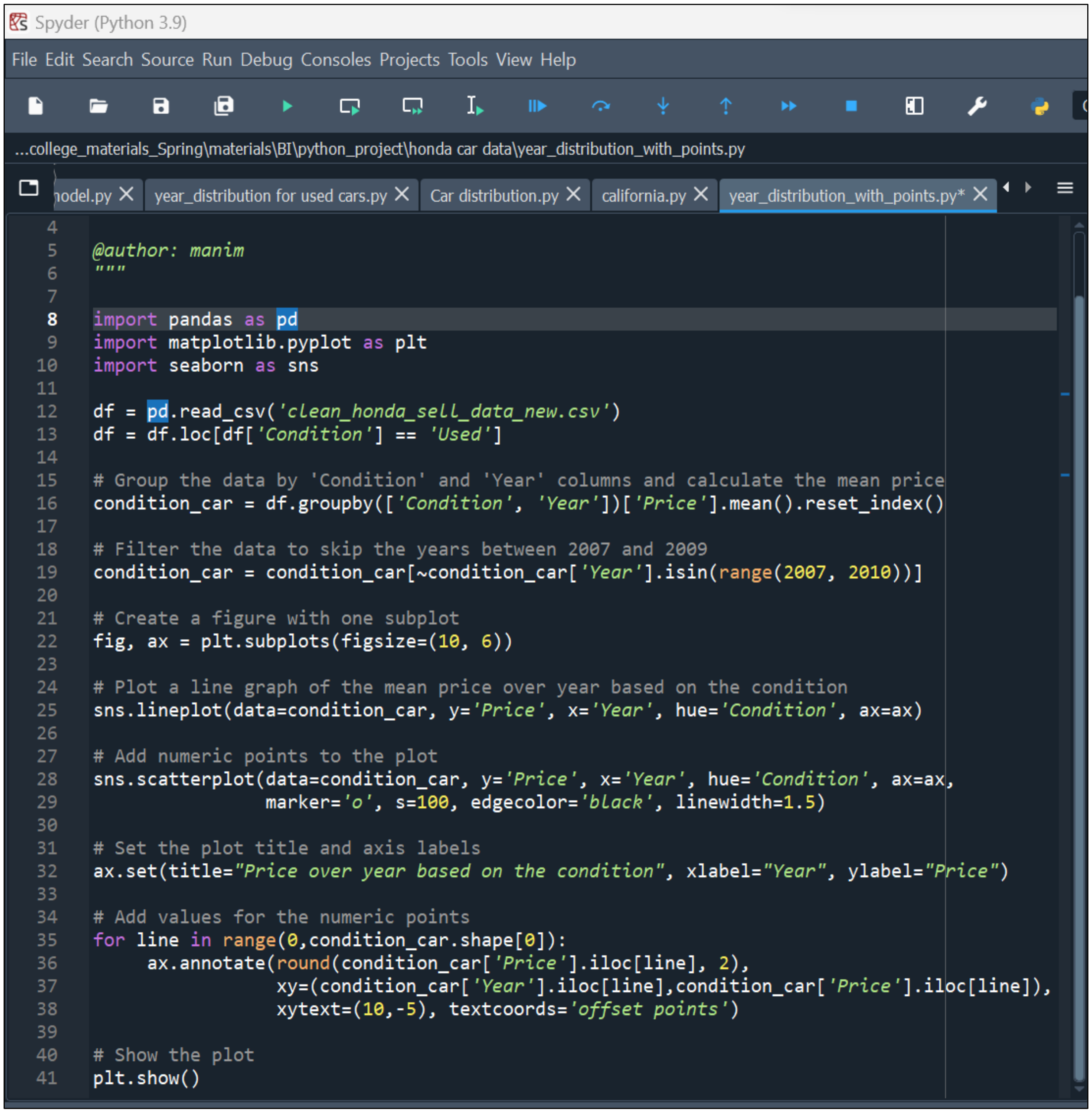
The visualization of the distribution of cars based on condition is done using a pie chart. The color palette used in the pie chart is shades of blue, with the darkest shade being used for the 'Used' condition, followed by 'New' and 'Certified Pre-Owned'.

As part of the analysis, we can conclude that most of the cars in the dataset are in the 'New' condition, accounting for 54.7% of the total cars. This suggests that Honda cars are popular among buyers who are interested in purchasing a new car.

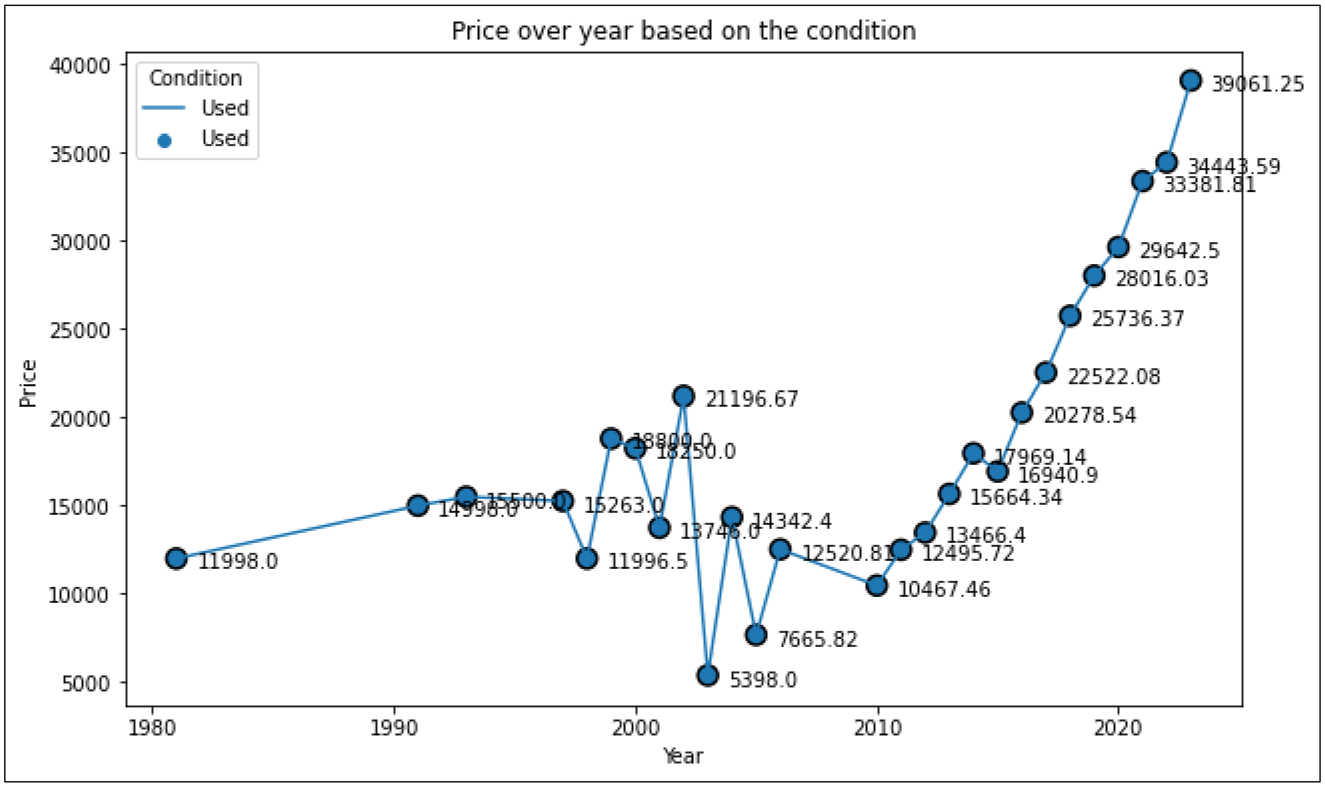
The 'Used' cars account for 39.5%, which shows that there is still a significant demand for used Honda cars. This indicates that Honda cars have a good resale value and are considered reliable vehicles in the market.

The 'Honda Certified' cars account for only 5.8%, which suggests that buyers may not be as interested in certified pre-owned Honda cars. However, this could also indicate that the certification process may not be well-known or trusted among buyers.

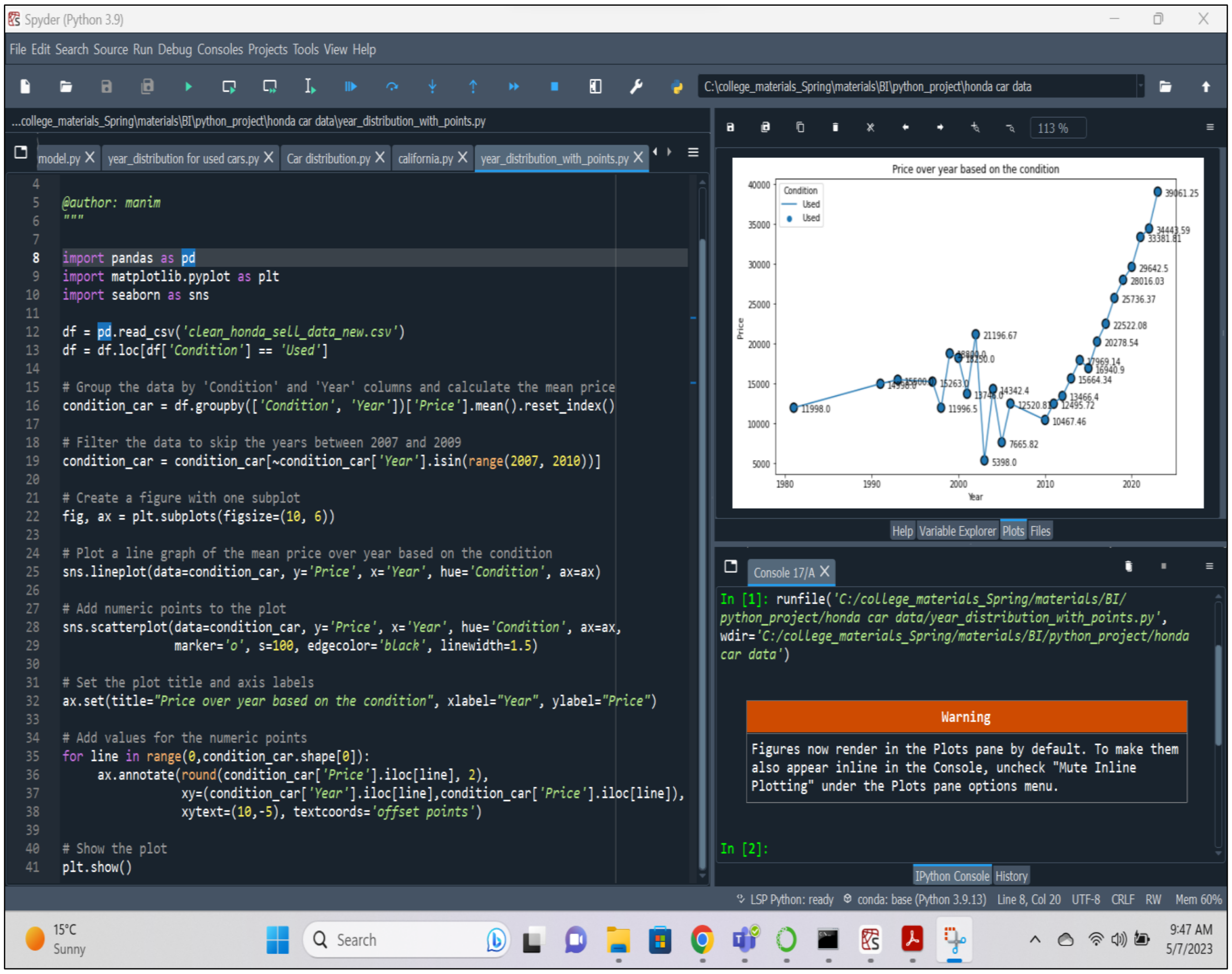
2. How has the price of Honda model car varied by year?


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
...college_materials_Spring\materials\BI\python_project\honda car data\year_distribution_with_points.py
nodel.py
year_distribution for used cars.py
Car distribution.py
california.py
year_distribution_with_points.py*

```python

@author: manim
"""

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

df = pd.read_csv('clean_honda_sell_data_new.csv')
df = df.loc[df['Condition'] == 'Used']

# Group the data by 'Condition' and 'Year' columns and calculate the mean price
condition_car = df.groupby(['Condition', 'Year'])['Price'].mean().reset_index()

# Filter the data to skip the years between 2007 and 2009
condition_car = condition_car[~condition_car['Year'].isin(range(2007, 2010))]

# Create a figure with one subplot
fig, ax = plt.subplots(figsize=(10, 6))

# Plot a line graph of the mean price over year based on the condition
sns.lineplot(data=condition_car, y='Price', x='Year', hue='Condition', ax=ax)

# Add numeric points to the plot
sns.scatterplot(data=condition_car, y='Price', x='Year', hue='Condition', ax=ax,
                marker='o', s=100, edgecolor='black', linewidth=1.5)

# Set the plot title and axis labels
ax.set(title="Price over year based on the condition", xlabel="Year", ylabel="Price")

# Add values for the numeric points
for line in range(0,condition_car.shape[0]):
     ax.annotate(round(condition_car['Price'].iloc[line], 2),
                 xy=(condition_car['Year'].iloc[line],condition_car['Price'].iloc[line]),
                 xytext=(10,-5), textcoords='offset points')

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



Price over year based on the condition
Condition
Used
Used
Price
Year
11998.0
14998.0
15500.0
15263.0
11996.5
18800.0
18250.0
13746.0
21196.67
5398.0
14342.4
7665.82
12520.81
10467.46
12495.72
13466.4
15664.34
17969.14
16940.9
20278.54
22522.08
25736.37
28016.03
29642.5
33381.81
34443.59
39061.25


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\college_materials_Spring\materials\BI\python_project\honda car data
...college_materials_Spring\materials\BI\python_project\honda car data\year_distribution_with_points.py
model.py
year_distribution for used cars.py
Car distribution.py
california.py
year_distribution_with_points.py
113 %
Price over year based on the condition
Condition
Used
Used
Price
Year
Help Variable Explorer Plots Files
Console 17/A
IPython Console History
LSP Python: ready conda: base (Python 3.9.13) Line 8, Col 20 UTF-8 CRLF RW Mem 60%
15°C
Sunny
Search
9:47 AM
5/7/2023

```python

@author: manim
"""

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

df = pd.read_csv('clean_honda_sell_data_new.csv')
df = df.loc[df['Condition'] == 'Used']

# Group the data by 'Condition' and 'Year' columns and calculate the mean price
condition_car = df.groupby(['Condition', 'Year'])['Price'].mean().reset_index()

# Filter the data to skip the years between 2007 and 2009
condition_car = condition_car[~condition_car['Year'].isin(range(2007, 2010))]

# Create a figure with one subplot
fig, ax = plt.subplots(figsize=(10, 6))

# Plot a line graph of the mean price over year based on the condition
sns.lineplot(data=condition_car, y='Price', x='Year', hue='Condition', ax=ax)

# Add numeric points to the plot
sns.scatterplot(data=condition_car, y='Price', x='Year', hue='Condition', ax=ax,
                marker='o', s=100, edgecolor='black', linewidth=1.5)

# Set the plot title and axis labels
ax.set(title="Price over year based on the condition", xlabel="Year", ylabel="Price")

# Add values for the numeric points
for line in range(0,condition_car.shape[0]):
     ax.annotate(round(condition_car['Price'].iloc[line], 2),
                 xy=(condition_car['Year'].iloc[line],condition_car['Price'].iloc[line]),
                 xytext=(10,-5), textcoords='offset points')

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

```
In [1]: runfile('C:/college_materials_Spring/materials/BI/
python_project/honda car data/year_distribution_with_points.py',
wdir='C:/college_materials_Spring/materials/BI/python_project/honda
car data')

                          Warning

Figures now render in the Plots pane by default. To make them
also appear inline in the Console, uncheck "Mute Inline
Plotting" under the Plots pane options menu.

In [2]:
```

Insights:

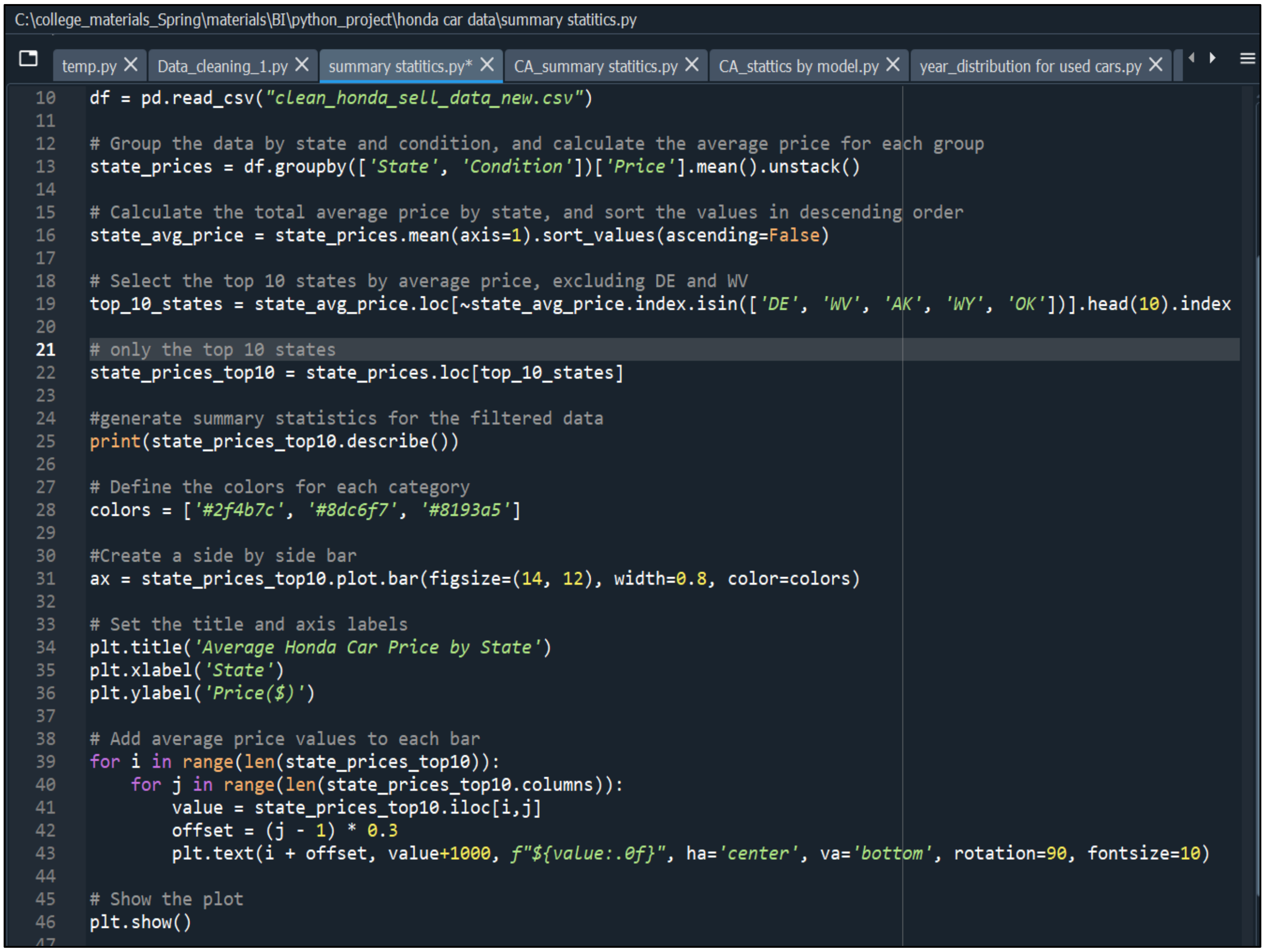
The data is filtered for only used cars as the analysis is focused on the price trends for used cars.

The data is grouped by condition and year, and the mean price of the used cars is calculated. This is done to indicate that the analysis is focused on the price trends of used cars. The resulting line graph shows how the average price of used Honda cars has fluctuated over time.

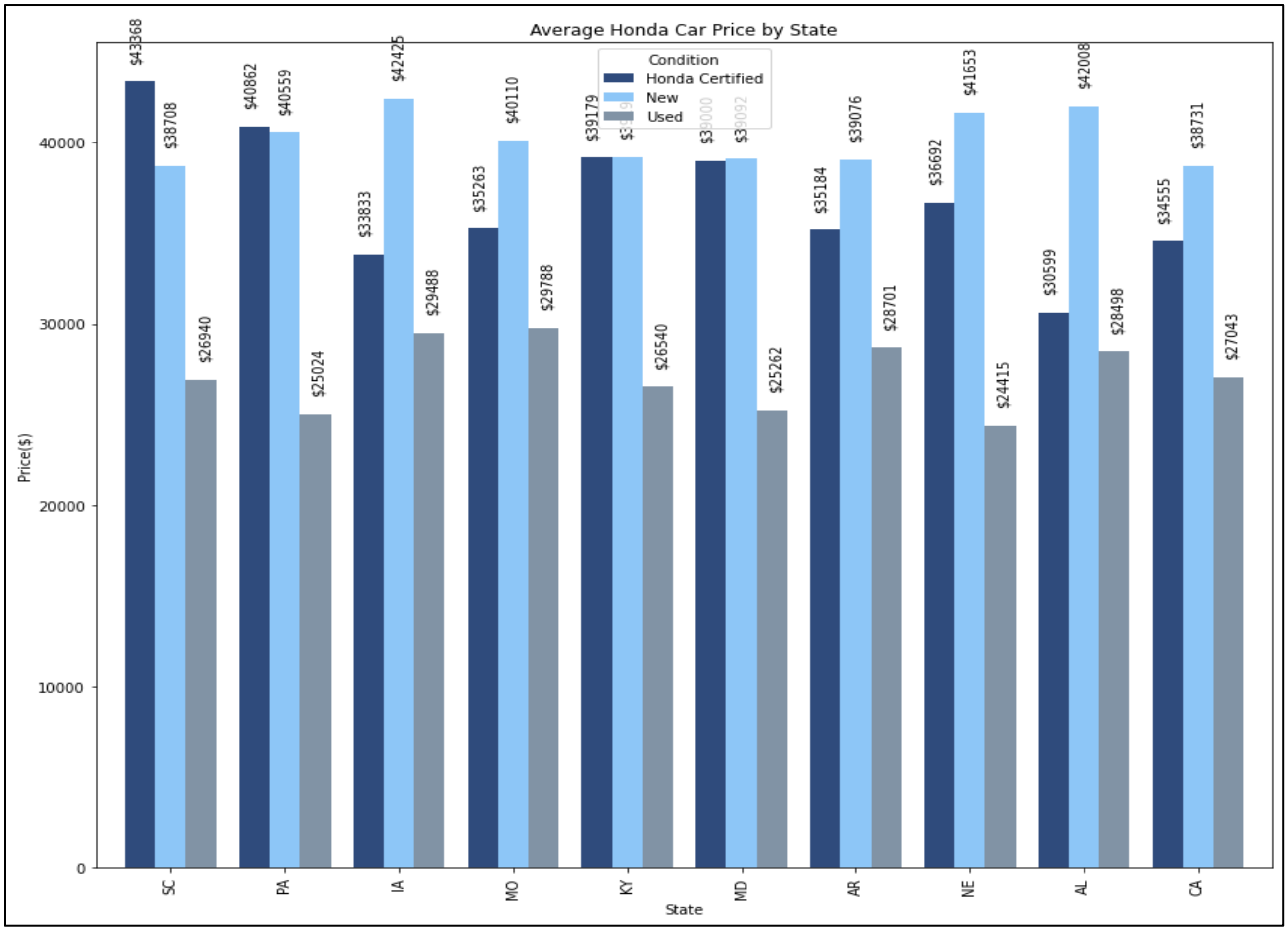
Through the line plot we visualize the mean price of the used cars over the years based on their condition. The x-axis shows the Year, and the Y-axis shows the Average Price. This visualization helps in understanding the trend in price changes for used cars over time.

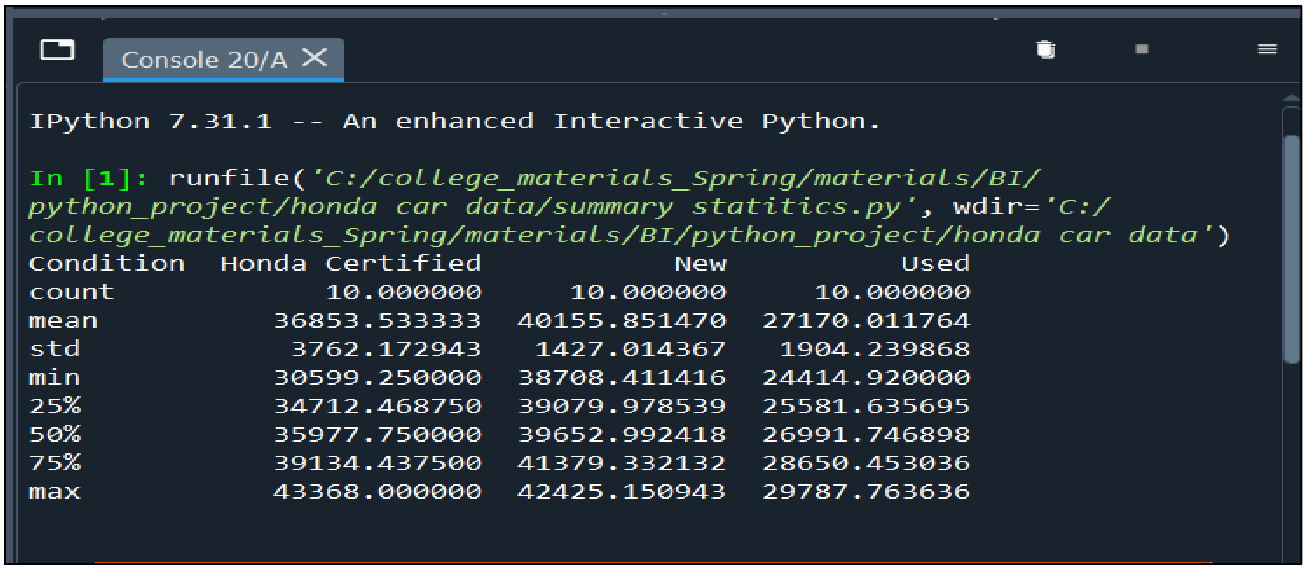
The plot indicates that the price of used cars has increased over the years. We can see that after the year 2000 till 2010 year, the price of used cars dropped drastically and after 2010, the average price of used cars shoots up very high. This indicates that the demand for used cars has increased over time, leading to an increase in their prices.

3. Show the Average Honda cars Price for top 10 states

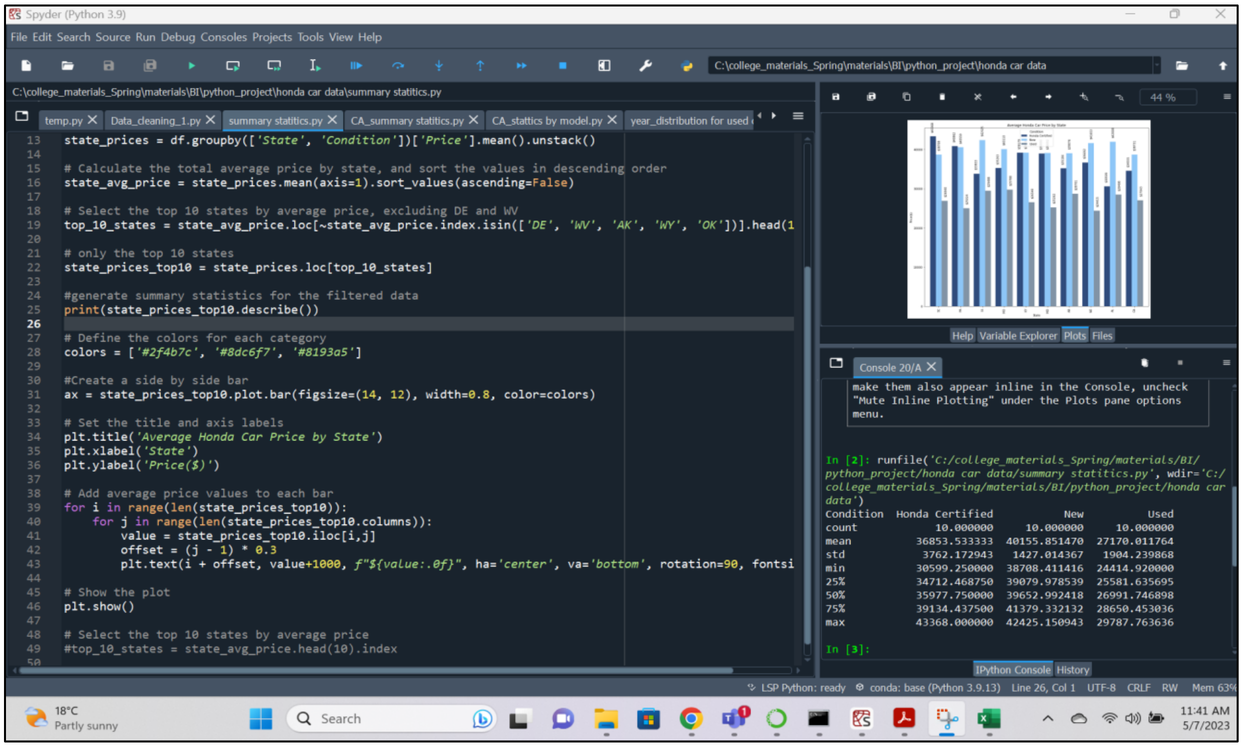

C:\college_materials_Spring\materials\BI\python_project\honda car data\summary statitics.py

```python
df = pd.read_csv("clean_honda_sell_data_new.csv")

# Group the data by state and condition, and calculate the average price for each group
state_prices = df.groupby(['State', 'Condition'])['Price'].mean().unstack()

# Calculate the total average price by state, and sort the values in descending order
state_avg_price = state_prices.mean(axis=1).sort_values(ascending=False)

# Select the top 10 states by average price, excluding DE and WV
top_10_states = state_avg_price.loc[~state_avg_price.index.isin(['DE', 'WV', 'AK', 'WY', 'OK'])].head(10).index

# only the top 10 states
state_prices_top10 = state_prices.loc[top_10_states]

#generate summary statistics for the filtered data
print(state_prices_top10.describe())

# Define the colors for each category
colors = ['#2f4b7c', '#8dc6f7', '#8193a5']

#Create a side by side bar
ax = state_prices_top10.plot.bar(figsize=(14, 12), width=0.8, color=colors)

# Set the title and axis labels
plt.title('Average Honda Car Price by State')
plt.xlabel('State')
plt.ylabel('Price($)')

# Add average price values to each bar
for i in range(len(state_prices_top10)):
    for j in range(len(state_prices_top10.columns)):
        value = state_prices_top10.iloc[i,j]
        offset = (j - 1) * 0.3
        plt.text(i + offset, value+1000, f"${value:.0f}", ha='center', va='bottom', rotation=90, fontsize=10)

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





```
IPython 7.31.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/college_materials_Spring/materials/BI/
python_project/honda car data/summary statitics.py', wdir='C:/
college_materials_Spring/materials/BI/python_project/honda car data')
Condition   Honda Certified           New          Used
count             10.000000     10.000000     10.000000
mean           36853.533333  40155.851470  27170.011764
std             3762.172943   1427.014367   1904.239868
min            30599.250000  38708.411416  24414.920000
25%            34712.468750  39079.978539  25581.635695
50%            35977.750000  39652.992418  26991.746898
75%            39134.437500  41379.332132  28650.453036
max            43368.000000  42425.150943  29787.763636
```



```
state_prices = df.groupby(['State', 'Condition'])['Price'].mean().unstack()

# Calculate the total average price by state, and sort the values in descending order
state_avg_price = state_prices.mean(axis=1).sort_values(ascending=False)

# Select the top 10 states by average price, excluding DE and WV
top_10_states = state_avg_price.loc[~state_avg_price.index.isin(['DE', 'WV', 'AK', 'WY', 'OK'])].head(1

# only the top 10 states
state_prices_top10 = state_prices.loc[top_10_states]

#generate summary statistics for the filtered data
print(state_prices_top10.describe())

# Define the colors for each category
colors = ['#2f4b7c', '#8dc6f7', '#8193a5']

#Create a side by side bar
ax = state_prices_top10.plot.bar(figsize=(14, 12), width=0.8, color=colors)

# Set the title and axis labels
plt.title('Average Honda Car Price by State')
plt.xlabel('State')
plt.ylabel('Price($)')

# Add average price values to each bar
for i in range(len(state_prices_top10)):
    for j in range(len(state_prices_top10.columns)):
        value = state_prices_top10.iloc[i,j]
        offset = (j - 1) * 0.3
        plt.text(i + offset, value+1000, f"${value:.0f}", ha='center', va='bottom', rotation=90, fontsi

# Show the plot
plt.show()

# Select the top 10 states by average price
#top_10_states = state_avg_price.head(10).index
```

```
make them also appear inline in the Console, uncheck
"Mute Inline Plotting" under the Plots pane options
menu.

In [2]: runfile('C:/college_materials_Spring/materials/BI/
python_project/honda car data/summary statitics.py', wdir='C:/
college_materials_Spring/materials/BI/python_project/honda car
data')
Condition  Honda Certified           New          Used
count            10.000000     10.000000     10.000000
mean          36853.533333  40155.851470  27170.011764
std            3762.172943   1427.014367   1904.239868
min           30599.250000  38708.411416  24414.920000
25%           34712.468750  39079.978539  25581.635695
50%           35977.750000  39652.992418  26991.746898
75%           39134.437500  41379.332132  28650.453036
max           43368.000000  42425.150943  29787.763636

In [3]:
```

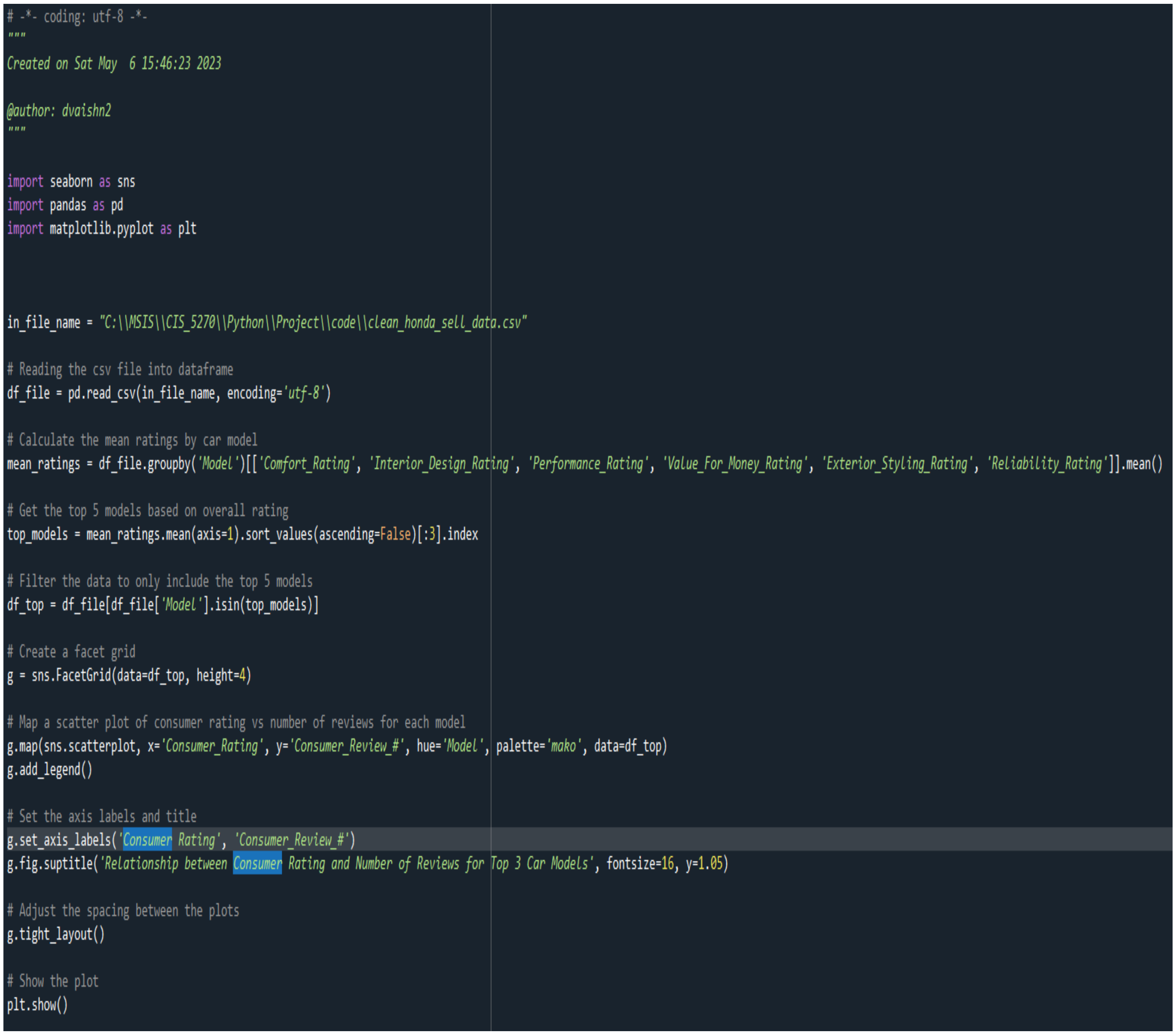
Insights:

The code groups the data by state and condition, calculates the average price for each group, and creates a side-by-side bar chart showing the average Honda car price by state. Only the data for these top 10 states is used to generate the bar chart. The bar chart showed us that the average price of Honda cars varied widely by state and condition, with some states having much higher prices than others.

Through this analysis we can understand that the state SC (South Carolina) has highest price for Honda certified cars and in the state IA (IOWA) Average New Honda car price is $42,425. The Used Honda car has the highest price in the state of MO (Missouri). In the State PY(Pennsylvania), MD(Maryland), KY(Kentucky) has not much difference between the price of Honda Certified cars and new cars. Overall, there is not much significant difference between the Average price of NEW cars across the states.

Overall, this analysis provides useful insights for anyone interested in buying or selling Honda cars in the United States, as it highlights the importance of considering regional variations in price.

4. Display the correlation between the number of reviews and consumer ratings for the top 3 car models, which are determined based on their consumer ratings

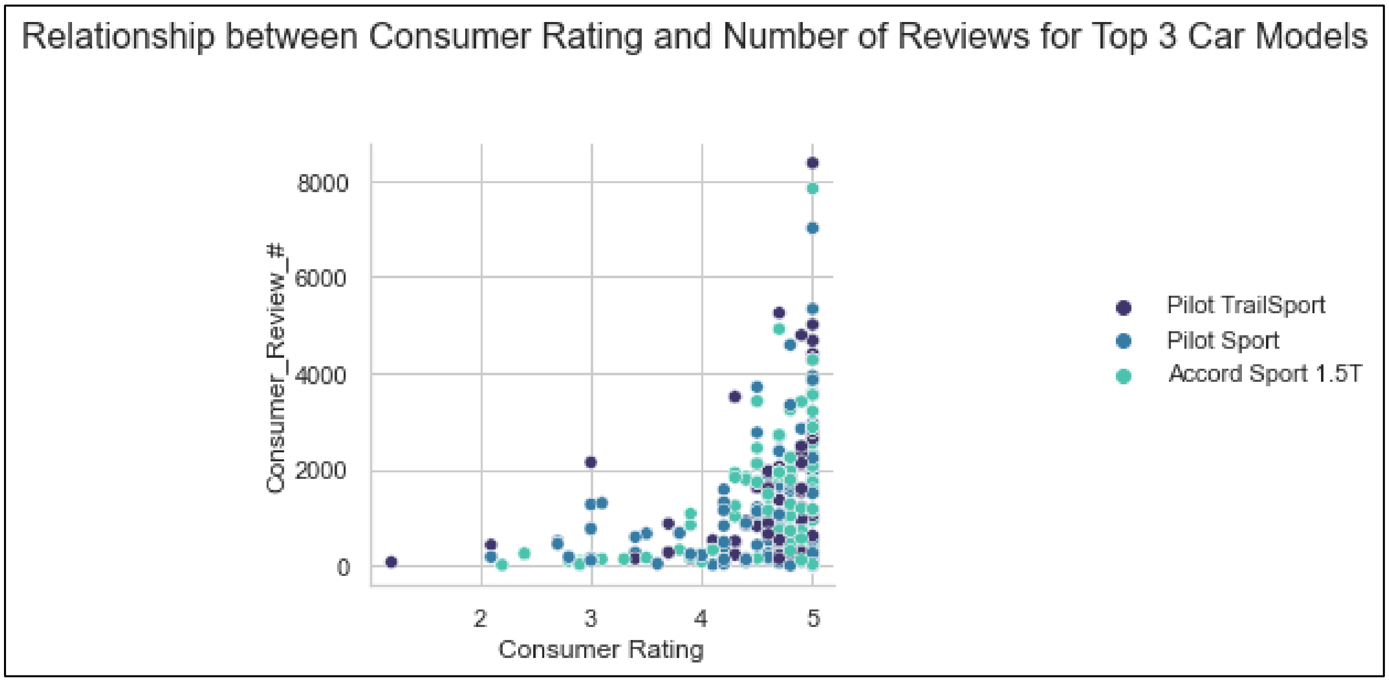


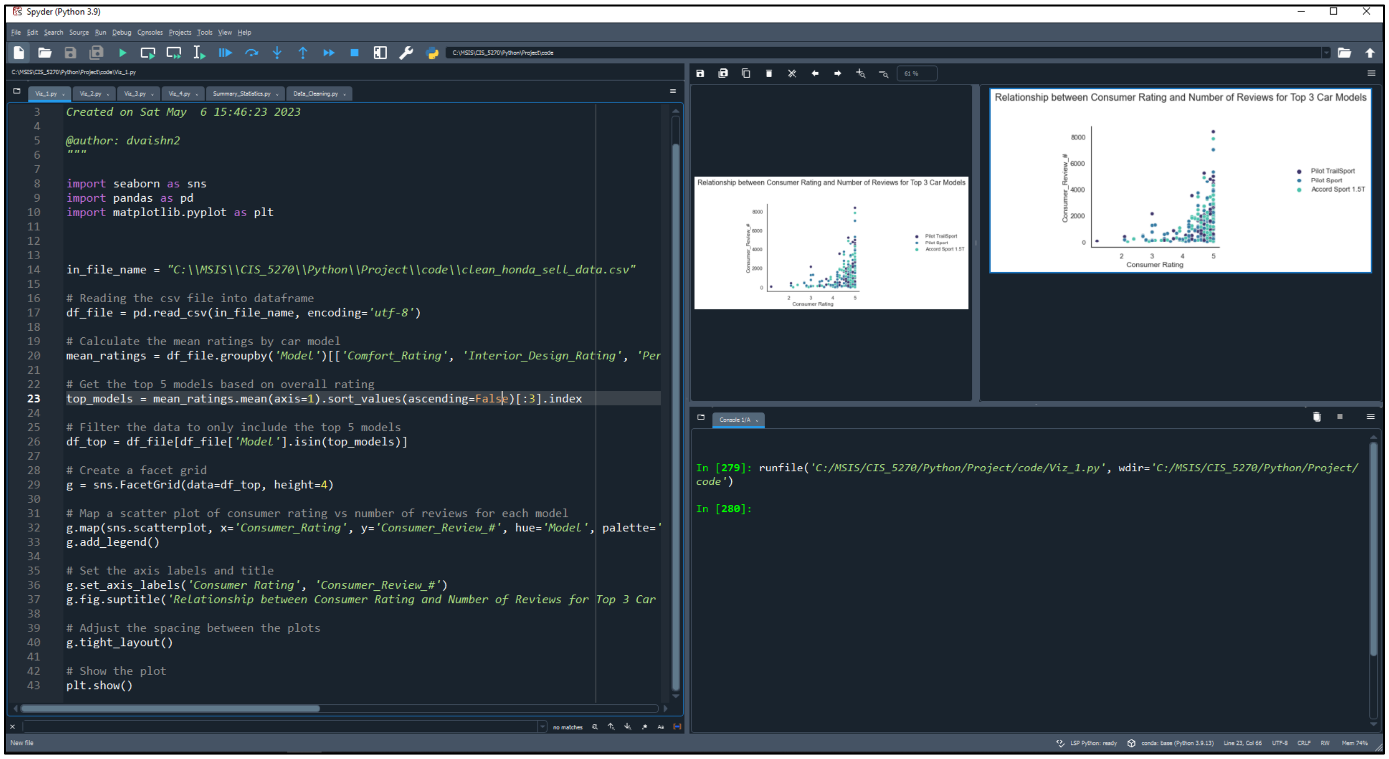
```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 15:46:23 2023

@author: dvaishn2
"""

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



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')

# Calculate the mean ratings by car model
mean_ratings = df_file.groupby('Model')[['Comfort_Rating', 'Interior_Design_Rating', 'Performance_Rating', 'Value_For_Money_Rating', 'Exterior_Styling_Rating', 'Reliability_Rating']].mean()

# Get the top 5 models based on overall rating
top_models = mean_ratings.mean(axis=1).sort_values(ascending=False)[:3].index

# Filter the data to only include the top 5 models
df_top = df_file[df_file['Model'].isin(top_models)]

# Create a facet grid
g = sns.FacetGrid(data=df_top, height=4)

# Map a scatter plot of consumer rating vs number of reviews for each model
g.map(sns.scatterplot, x='Consumer_Rating', y='Consumer_Review_#', hue='Model', palette='mako', data=df_top)
g.add_legend()

# Set the axis labels and title
g.set_axis_labels('Consumer Rating', 'Consumer_Review_#')
g.fig.suptitle('Relationship between Consumer Rating and Number of Reviews for Top 3 Car Models', fontsize=16, y=1.05)

# Adjust the spacing between the plots
g.tight_layout()

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

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 15:46:23 2023
@author: dvaishn2
"""
```

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

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')

# Calculate the mean ratings by car model
mean_ratings = df_file.groupby('Model')[['Comfort_Rating', 'Interior_Design_Rating',
'Performance_Rating', 'Value_For_Money_Rating', 'Exterior_Styling_Rating',
'Reliability_Rating']].mean()

# Get the top 5 models based on overall rating
top_models = mean_ratings.mean(axis=1).sort_values(ascending=False)[:3].index

# Filter the data to only include the top 5 models
df_top = df_file[df_file['Model'].isin(top_models)]

# Create a facet grid
g = sns.FacetGrid(data=df_top, height=4)

# Map a scatter plot of consumer rating vs number of reviews for each model
g.map(sns.scatterplot, x='Consumer_Rating', y='Consumer_Review_#', hue='Model',
palette='mako', data=df_top)
g.add_legend()

# Set the axis labels and title
```

```
g.set_axis_labels('Consumer Rating', 'Consumer_Review_#')

g.fig.suptitle('Relationship between Consumer Rating and Number of Reviews for Top 3 Car
Models', fontsize=16, y=1.05)


# Adjust the spacing between the plots

g.tight_layout()


# Show the plot

plt.show()
```





```
Created on Sat May  6 15:46:23 2023

@author: dvaishn2
"""


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



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')

# Calculate the mean ratings by car model
mean_ratings = df_file.groupby('Model')[['Comfort_Rating', 'Interior_Design_Rating', 'Per

# Get the top 5 models based on overall rating
top_models = mean_ratings.mean(axis=1).sort_values(ascending=False)[:3].index

# Filter the data to only include the top 5 models
df_top = df_file[df_file['Model'].isin(top_models)]

# Create a facet grid
g = sns.FacetGrid(data=df_top, height=4)

# Map a scatter plot of consumer rating vs number of reviews for each model
g.map(sns.scatterplot, x='Consumer_Rating', y='Consumer_Review_#', hue='Model', palette='
g.add_legend()

# Set the axis labels and title
g.set_axis_labels('Consumer Rating', 'Consumer_Review_#')
g.fig.suptitle('Relationship between Consumer Rating and Number of Reviews for Top 3 Car

# Adjust the spacing between the plots
g.tight_layout()

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

```
In [279]: runfile('C:/MSIS/CIS_5270/Python/Project/code/Viz_1.py', wdir='C:/MSIS/CIS_5270/Python/Project/
code')

In [280]:
```

Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\MSIS\CIS_5270\Python\Project\code
C:\MSIS\CIS_5270\Python\Project\code\Viz_1.py
Viz_1.py
Viz_2.py
Viz_3.py
Viz_4.py
Summary_Statistics.py
Data_Cleaning.py
Relationship between Consumer Rating and Number of Reviews for Top 3 Car Models
Consumer_Review_#
Consumer Rating
Pilot TrailSport
Pilot Sport
Accord Sport 1.5T
Console 1/A
no matches
New file

Insights:

This visualization shows the relationship between consumer ratings and the number of reviews for the top three car models based on their overall rating.

A Facet Grid is created, which is a grid of plots showing the same relationship conditioned on different levels of a variable. In this case, Facet Grid contains scatterplots of consumer ratings vs. the number of reviews for each of the top 3 car models.

The scatterplots show the relationship between consumer rating and the number of reviews, with each dot representing a different review.

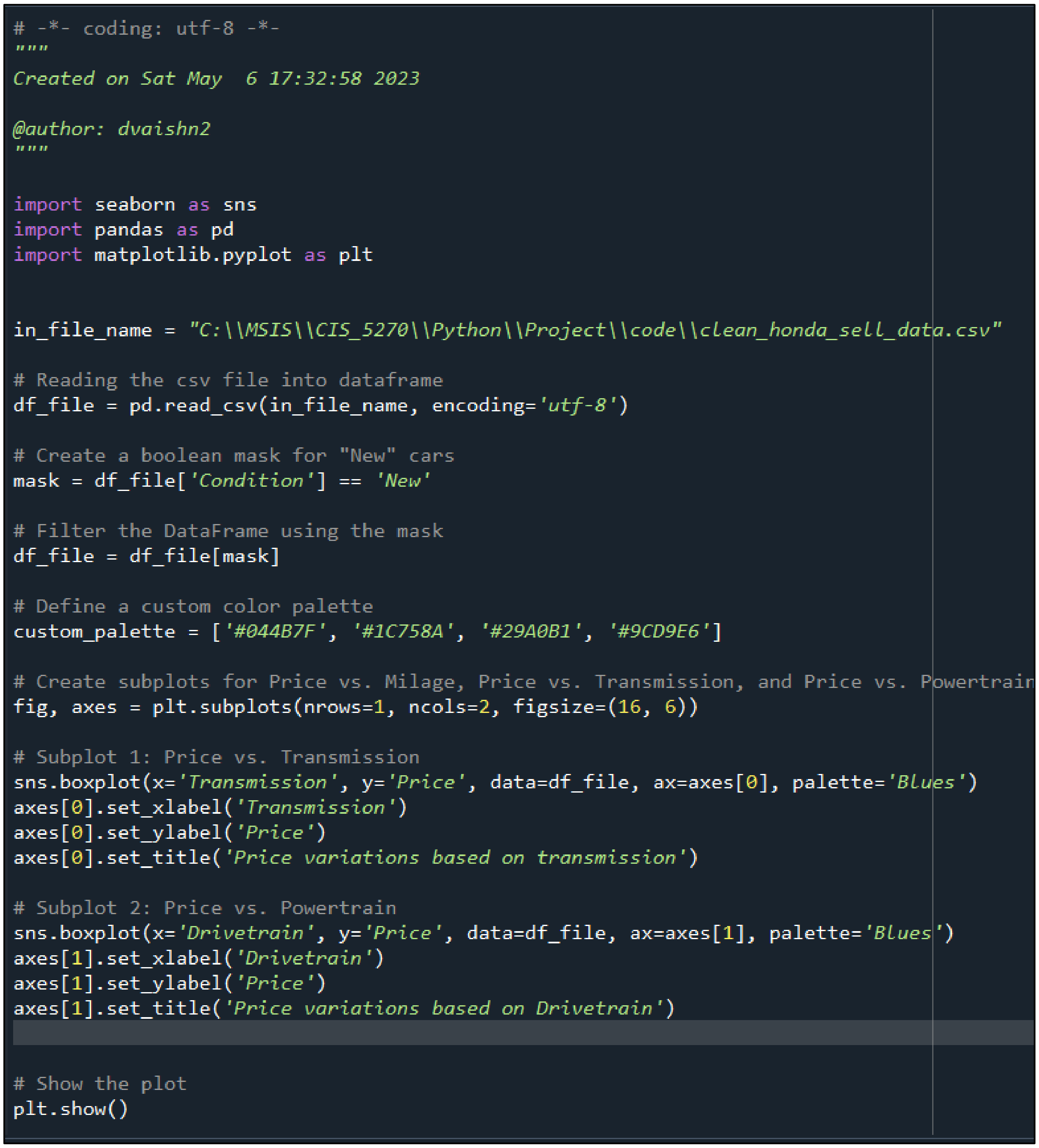
The x-axis of the scatter plot represents the consumer rating while the y-axis represents the number of reviews. The color of each data point represents the car model, and a legend is added to the plot for clarity.

The plot shows that there is a positive relationship between consumer ratings and the number of reviews for each of the top three car models. As the number of reviews increases, so does the consumer rating. However, the magnitude of the relationship differs across the three car models.

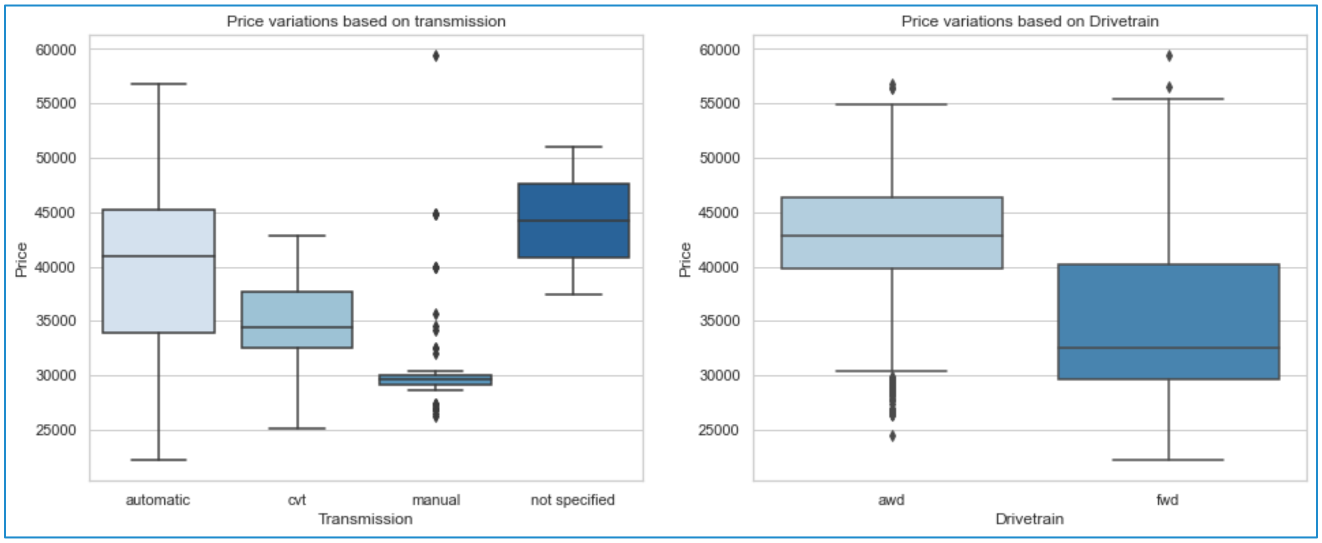
The graph helps in understanding the popularity of the top three car models and how consumer ratings and reviews affect their overall rating.

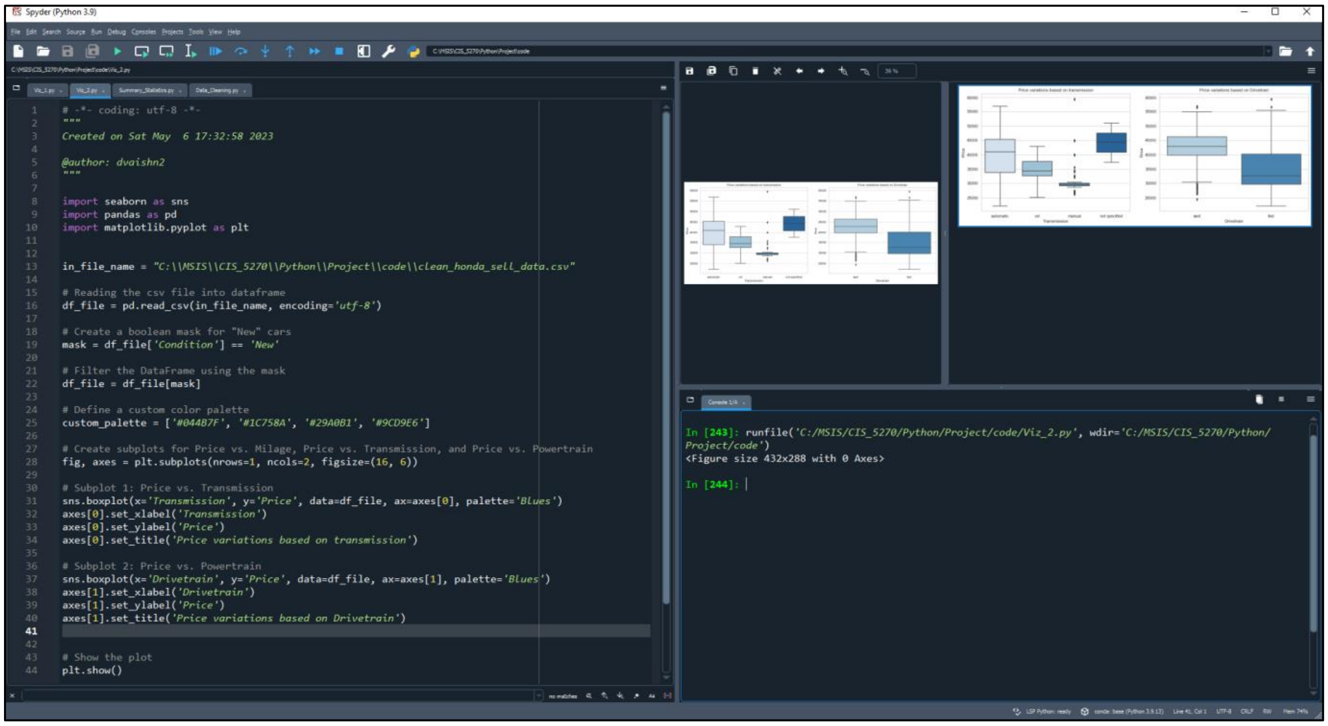
It also provides insights into which car models have a stronger relationship between consumer ratings and the number of reviews.

5. How do the price variations based on transmission compare to the price variations based on powertrain?



```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 17:32:58 2023

@author: dvaishn2
"""

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


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')

# Create a boolean mask for "New" cars
mask = df_file['Condition'] == 'New'

# Filter the DataFrame using the mask
df_file = df_file[mask]

# Define a custom color palette
custom_palette = ['#044B7F', '#1C758A', '#29A0B1', '#9CD9E6']

# Create subplots for Price vs. Milage, Price vs. Transmission, and Price vs. Powertrain
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

# Subplot 1: Price vs. Transmission
sns.boxplot(x='Transmission', y='Price', data=df_file, ax=axes[0], palette='Blues')
axes[0].set_xlabel('Transmission')
axes[0].set_ylabel('Price')
axes[0].set_title('Price variations based on transmission')

# Subplot 2: Price vs. Powertrain
sns.boxplot(x='Drivetrain', y='Price', data=df_file, ax=axes[1], palette='Blues')
axes[1].set_xlabel('Drivetrain')
axes[1].set_ylabel('Price')
axes[1].set_title('Price variations based on Drivetrain')


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




Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\MSIS\CIS_5270\Python\Project\code
Viz_1.py
Viz_2.py
Summary_Statistics.py
Data_Cleaning.py
Console 1/A

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 17:32:58 2023

@author: dvaishn2
"""

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


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')

# Create a boolean mask for "New" cars
mask = df_file['Condition'] == 'New'

# Filter the DataFrame using the mask
df_file = df_file[mask]

# Define a custom color palette
custom_palette = ['#044B7F', '#1C758A', '#29A0B1', '#9CD9E6']

# Create subplots for Price vs. Milage, Price vs. Transmission, and Price vs. Powertrain
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

# Subplot 1: Price vs. Transmission
sns.boxplot(x='Transmission', y='Price', data=df_file, ax=axes[0], palette='Blues')
axes[0].set_xlabel('Transmission')
axes[0].set_ylabel('Price')
axes[0].set_title('Price variations based on transmission')

# Subplot 2: Price vs. Powertrain
sns.boxplot(x='Drivetrain', y='Price', data=df_file, ax=axes[1], palette='Blues')
axes[1].set_xlabel('Drivetrain')
axes[1].set_ylabel('Price')
axes[1].set_title('Price variations based on Drivetrain')


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

```
In [243]: runfile('C:/MSIS/CIS_5270/Python/Project/code/Viz_2.py', wdir='C:/MSIS/CIS_5270/Python/
Project/code')
<Figure size 432x288 with 0 Axes>

In [244]:
```

Insights:

This visualization has two boxplots to show the variations in price of new cars based on their transmission type and drivetrain.

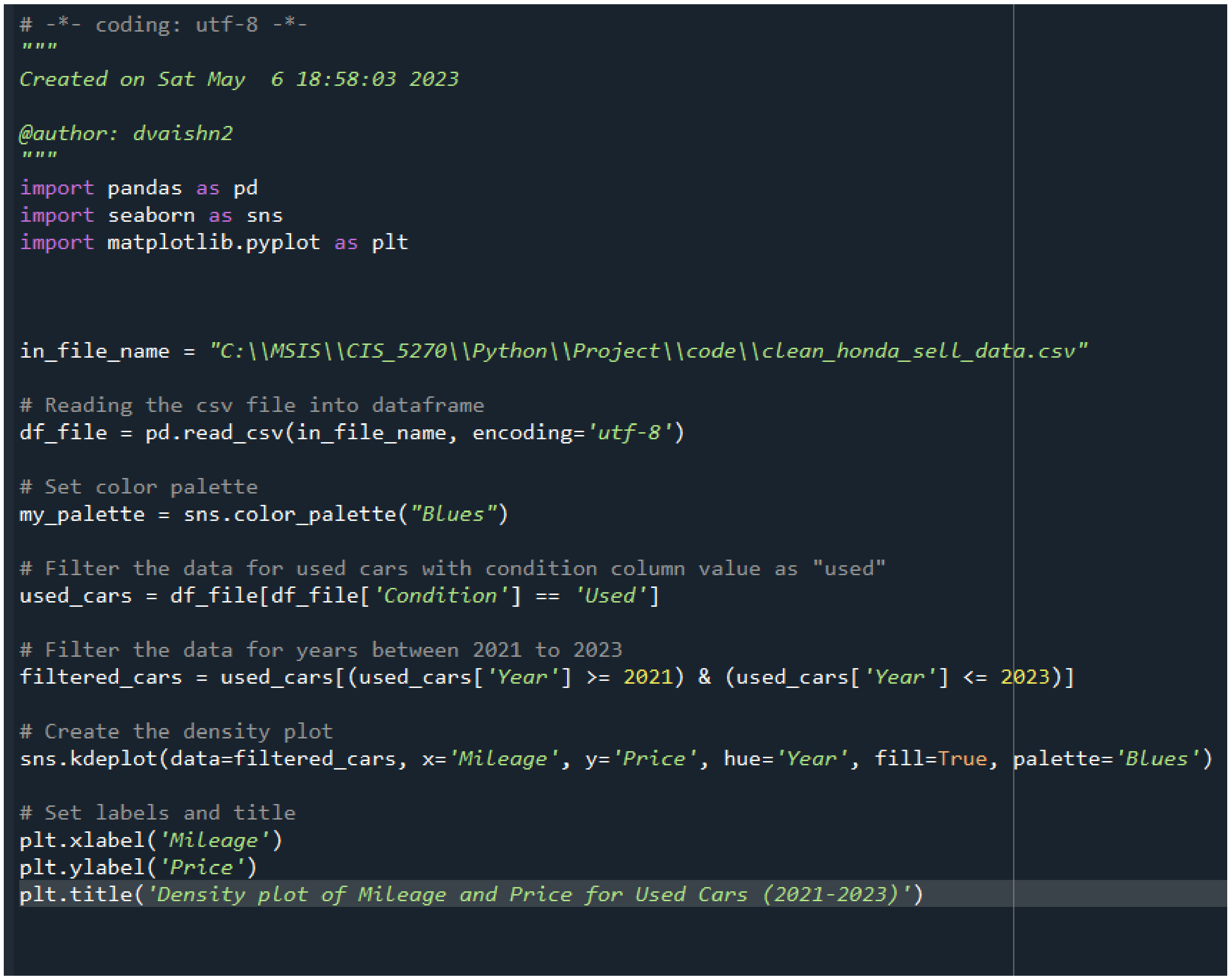
The left plot shows the relationship between price and transmission, while the right plot shows the relationship between price and drivetrain.

The boxplots display the distribution of prices for each category and highlight the median (the line in the box), the interquartile range (the box itself), and the range of values (the whiskers). The plot helps to identify whether there are significant differences in prices between different categories of transmission and drivetrain.

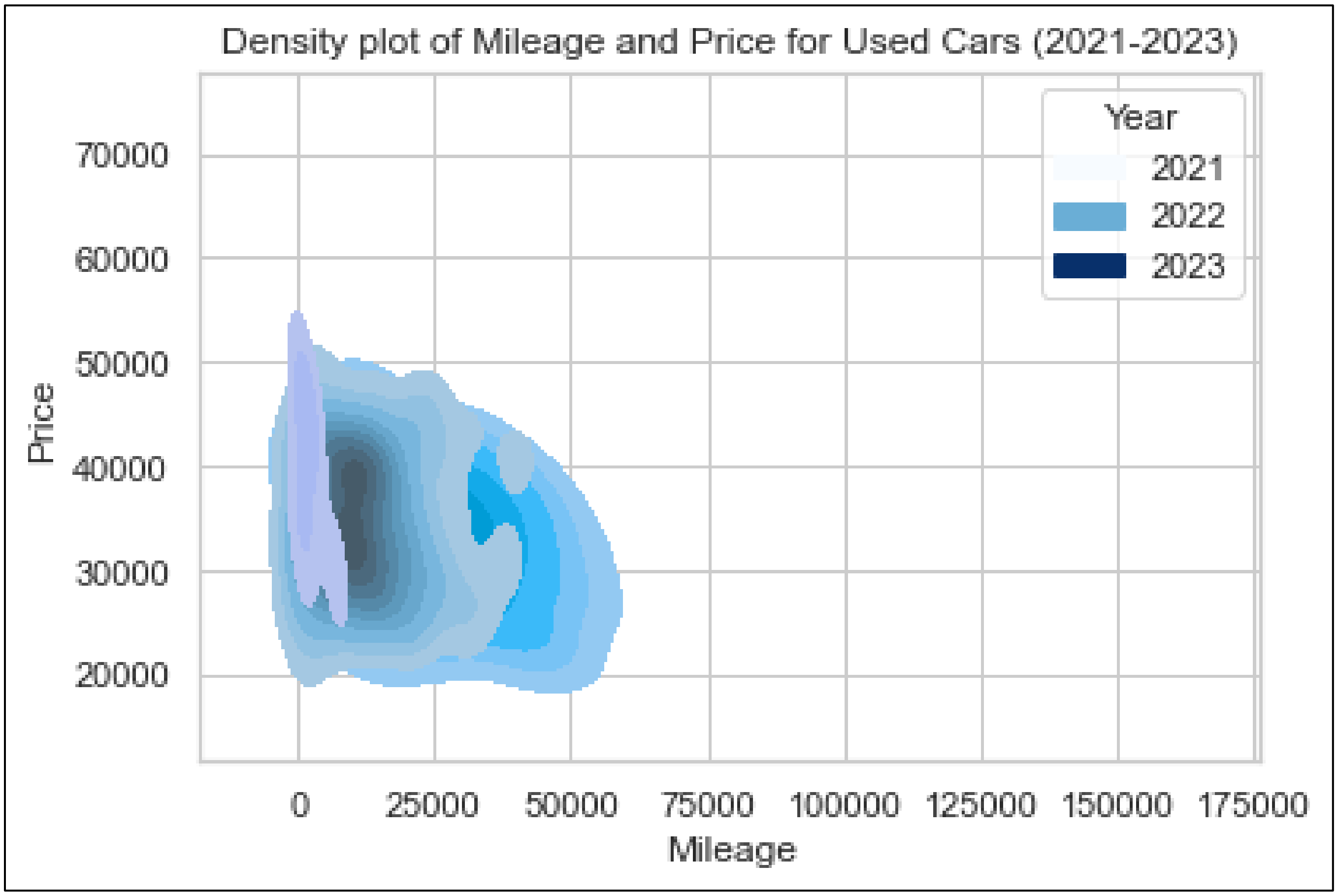
From the visualization, we can see that automatic transmission and all-wheel drivetrain cars tend to have higher prices compared to other categories (except for the not specified ones).

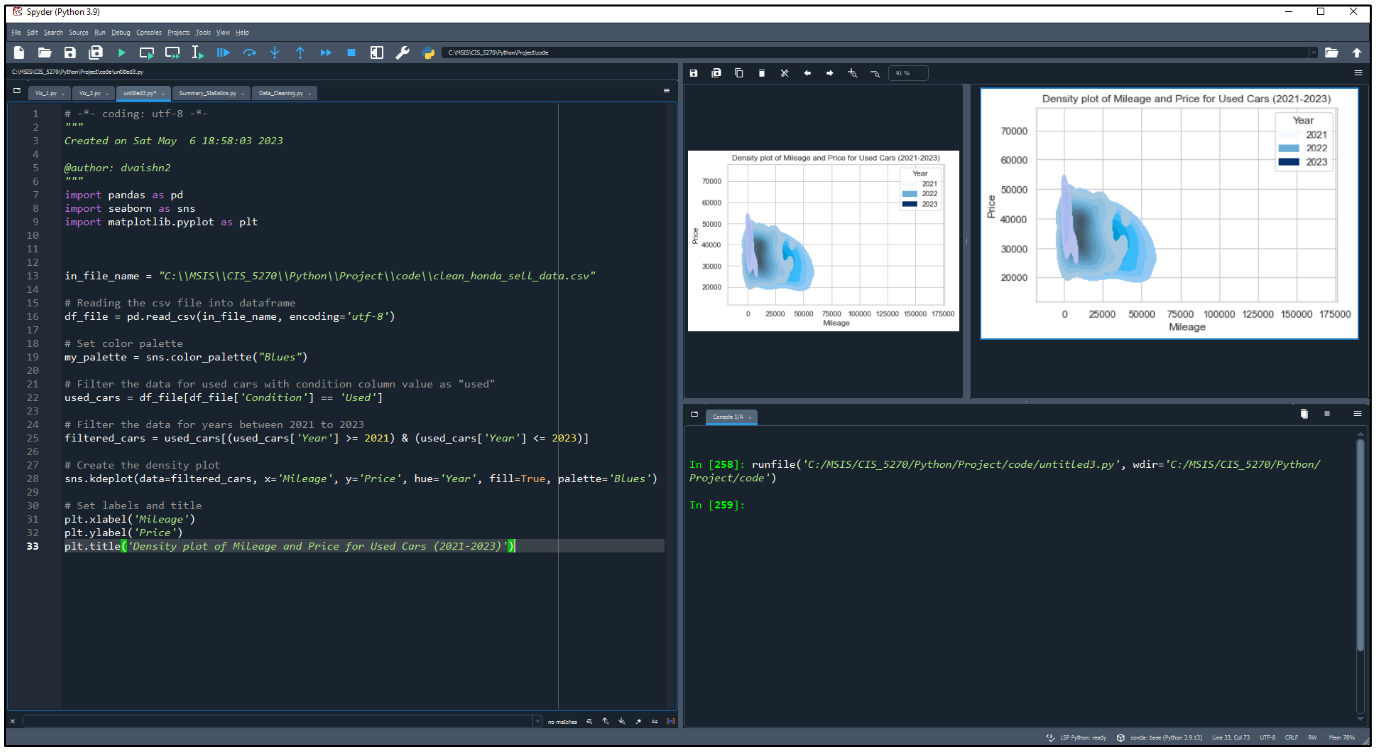
Additionally, the boxplots also show the presence of outliers, which are the individual data points outside the whiskers of the boxplots. These outliers are cars with prices that are significantly higher or lower than the typical prices for a given category.

6. Show the relationship between the MILEAGE values and PRICES for USED cars over a span of three years.



```python
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 18:58:03 2023

@author: dvaishn2
"""
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt



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')

# Set color palette
my_palette = sns.color_palette("Blues")

# Filter the data for used cars with condition column value as "used"
used_cars = df_file[df_file['Condition'] == 'Used']

# Filter the data for years between 2021 to 2023
filtered_cars = used_cars[(used_cars['Year'] >= 2021) & (used_cars['Year'] <= 2023)]

# Create the density plot
sns.kdeplot(data=filtered_cars, x='Mileage', y='Price', hue='Year', fill=True, palette='Blues')

# Set labels and title
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.title('Density plot of Mileage and Price for Used Cars (2021-2023)')
```




Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\MSIS\CIS_5270\Python\Project\code
C:\MSIS\CIS_5270\Python\Project\code\untitled3.py
Viz_1.py
Viz_2.py
untitled3.py*
Summary_Statistics.py
Data_Cleaning.py
Console 1/A

```python
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 18:58:03 2023

@author: dvaishn2
"""
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt



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')

# Set color palette
my_palette = sns.color_palette("Blues")

# Filter the data for used cars with condition column value as "used"
used_cars = df_file[df_file['Condition'] == 'Used']

# Filter the data for years between 2021 to 2023
filtered_cars = used_cars[(used_cars['Year'] >= 2021) & (used_cars['Year'] <= 2023)]

# Create the density plot
sns.kdeplot(data=filtered_cars, x='Mileage', y='Price', hue='Year', fill=True, palette='Blues')

# Set labels and title
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.title('Density plot of Mileage and Price for Used Cars (2021-2023)')
```

```
In [258]: runfile('C:/MSIS/CIS_5270/Python/Project/code/untitled3.py', wdir='C:/MSIS/CIS_5270/Python/
Project/code')

In [259]:
```

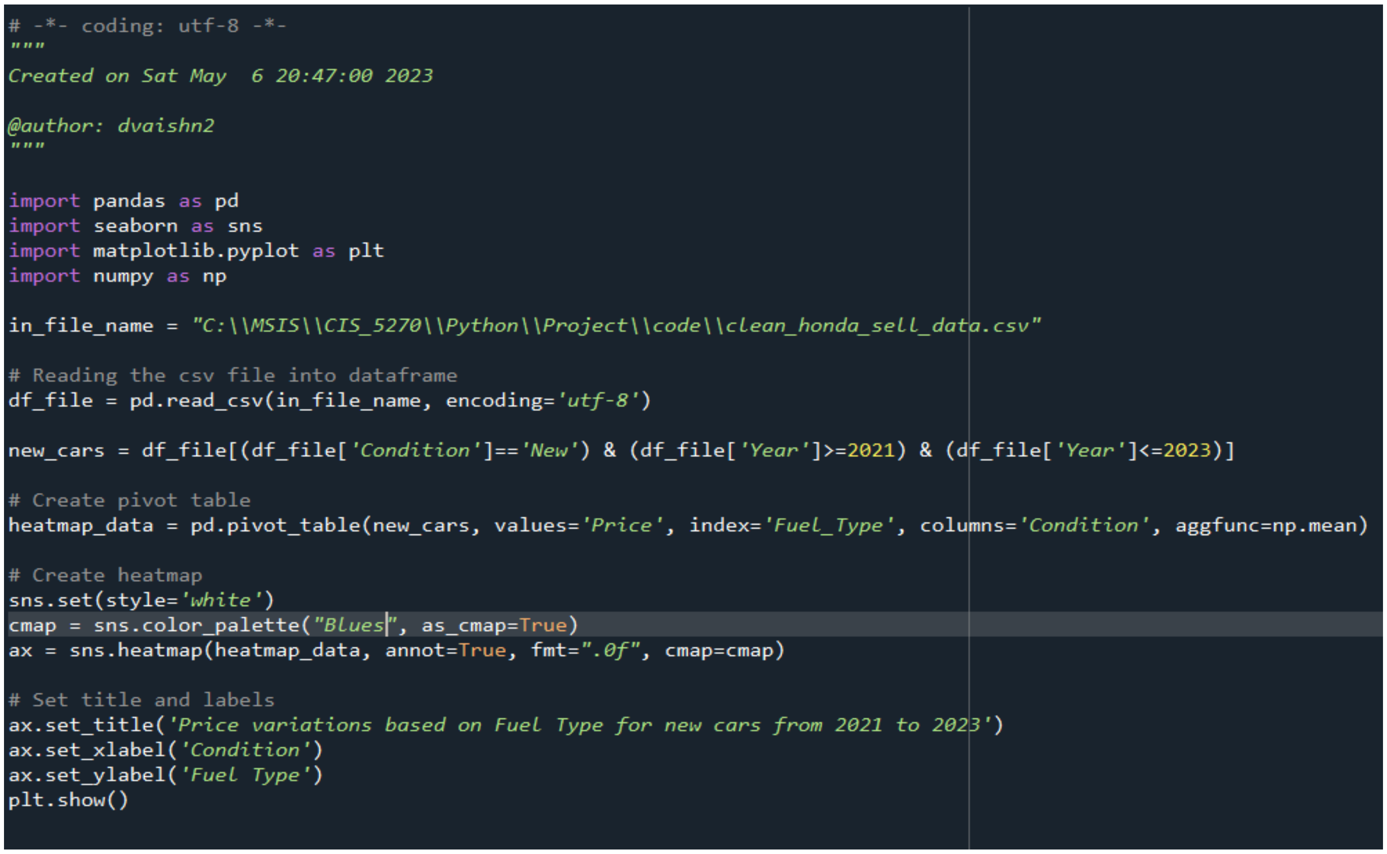
Insights:

The density plot visualizes the spread of used car prices based on their mileage and year of manufacture. The x-axis displays the mileage, the y-axis represents the year, and the color of the plot denotes the density of the price distribution.

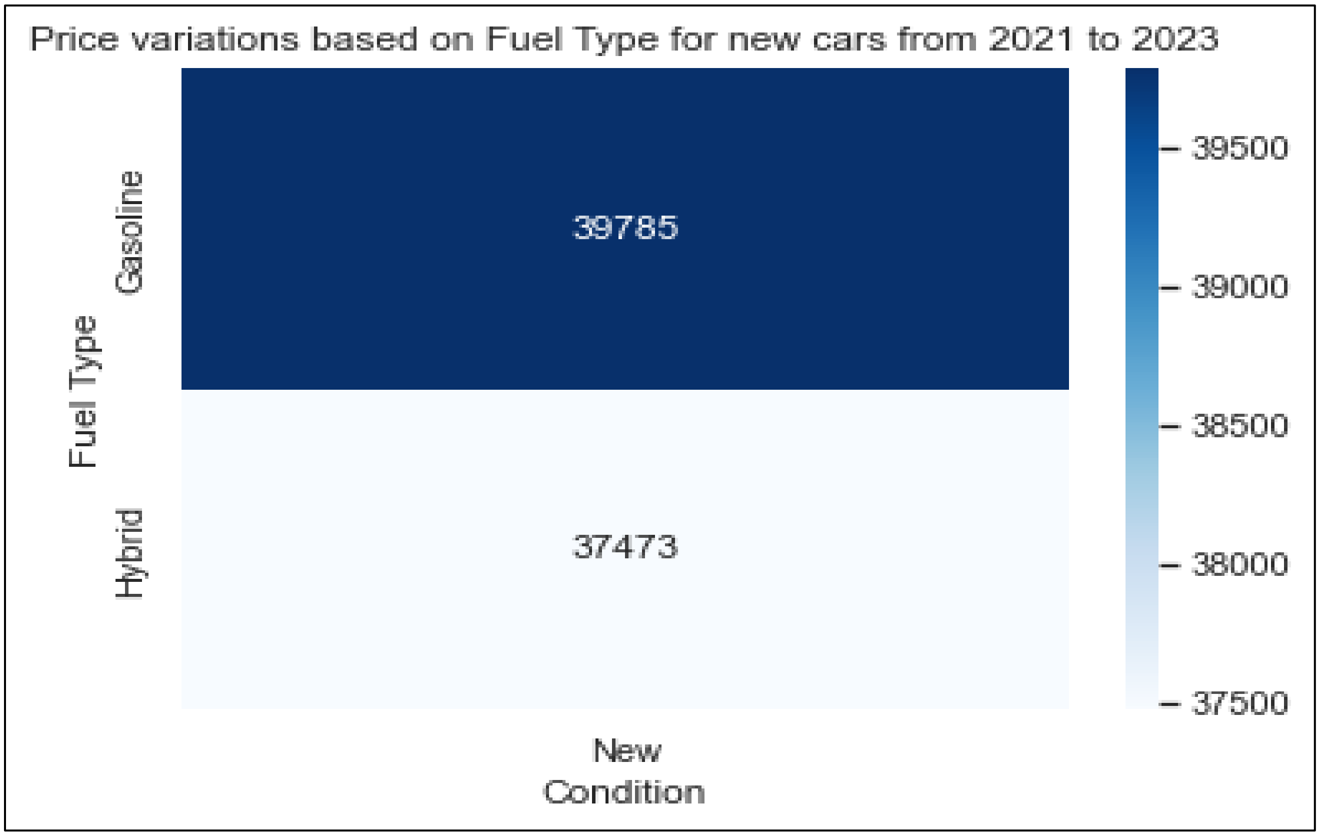
Darker regions on the plot indicate that there are more used cars with similar prices, while lighter regions show a lower density of prices. We can observe that most used car prices are clustered in the range of 0 to 60,000 miles (about 96560.64 km) and for cars manufactured in 2021. As the mileage and year of the car increase, the density of prices decreases.

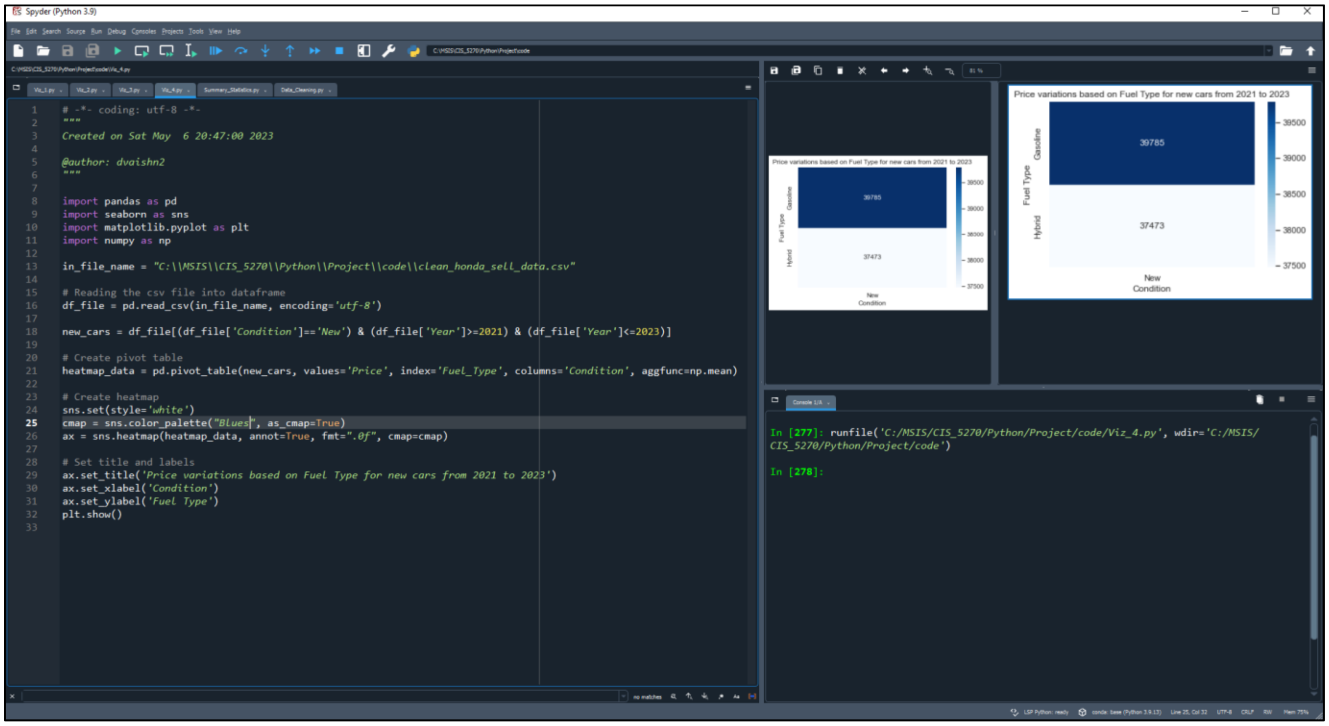
This plot can be useful in spotting patterns in the used car market, like the relationship between the year and mileage of a car and its price. It can also aid in identifying potential outliers, i.e., cars that are priced significantly higher or lower than the typical price range for a particular mileage and year combination.

7. Display how the prices are spread across various FUEL TYPES in the market for NEW cars for the last three years.



```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 20:47:00 2023

@author: dvaishn2
"""

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

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')

new_cars = df_file[(df_file['Condition']=='New') & (df_file['Year']>=2021) & (df_file['Year']<=2023)]

# Create pivot table
heatmap_data = pd.pivot_table(new_cars, values='Price', index='Fuel_Type', columns='Condition', aggfunc=np.mean)

# Create heatmap
sns.set(style='white')
cmap = sns.color_palette("Blues", as_cmap=True)
ax = sns.heatmap(heatmap_data, annot=True, fmt=".0f", cmap=cmap)

# Set title and labels
ax.set_title('Price variations based on Fuel Type for new cars from 2021 to 2023')
ax.set_xlabel('Condition')
ax.set_ylabel('Fuel Type')
plt.show()
```




Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\MSIS\CIS_5270\Python\Project\code
C:\MSIS\CIS_5270\Python\Project\code\Viz_4.py
Viz_1.py
Viz_2.py
Viz_3.py
Viz_4.py
Summary_Statistics.py
Data_Cleaning.py
Console 1/A

```python
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 20:47:00 2023

@author: dvaishn2
"""

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

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')

new_cars = df_file[(df_file['Condition']=='New') & (df_file['Year']>=2021) & (df_file['Year']<=2023)]

# Create pivot table
heatmap_data = pd.pivot_table(new_cars, values='Price', index='Fuel_Type', columns='Condition', aggfunc=np.mean)

# Create heatmap
sns.set(style='white')
cmap = sns.color_palette("Blues", as_cmap=True)
ax = sns.heatmap(heatmap_data, annot=True, fmt=".0f", cmap=cmap)

# Set title and labels
ax.set_title('Price variations based on Fuel Type for new cars from 2021 to 2023')
ax.set_xlabel('Condition')
ax.set_ylabel('Fuel Type')
plt.show()
```

```
In [277]: runfile('C:/MSIS/CIS_5270/Python/Project/code/Viz_4.py', wdir='C:/MSIS/
CIS_5270/Python/Project/code')

In [278]:
```

Insights:

The heatmap plot in the graph showcases the average price of new cars for different fuel types and conditions, thereby providing a quick and easy way to compare the prices across various categories.

The heat map is color-coded, with the lighter shades indicating lower values and the darker shades indicating higher values.

By analyzing the heatmap, we can get insights into which fuel type is preferred for new cars and its influence on price.

For instance, the graph could reveal that hybrid cars tend to have a higher price point than gasoline cars in the new car market.

Additionally, we can observe the price variations among different conditions for a particular fuel type.

Overall, this graph provides an effective way to understand the relationship between the average price of new cars and fuel types under different conditions, giving us valuable insights into market trends and consumer preferences.

F. References

[1] Used Cars Analysis: From Scrapping the Data to Visualization https://medium.datadriveninvestor.com/used-cars-analysis-from-scrapping-the-data-to-visualisation-962c863c89e1, January 08, 2023

[2] Pandas Cleansing and Visualization https://medium.com/@wekrklnd/cleaning-data-using-pandas-feead071aa7b , December 09, 2019