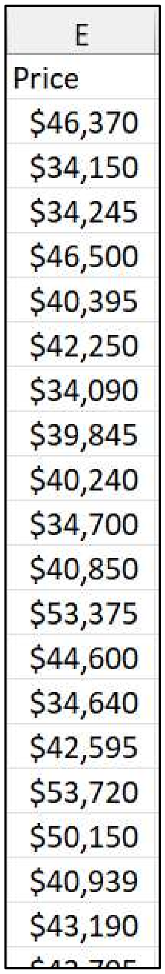
C. Data Cleaning

1. Handling Datatypes

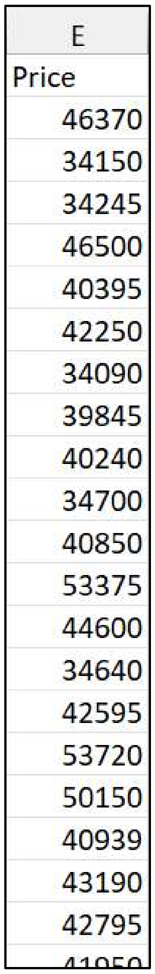
Column: Price

Before Cleaning:

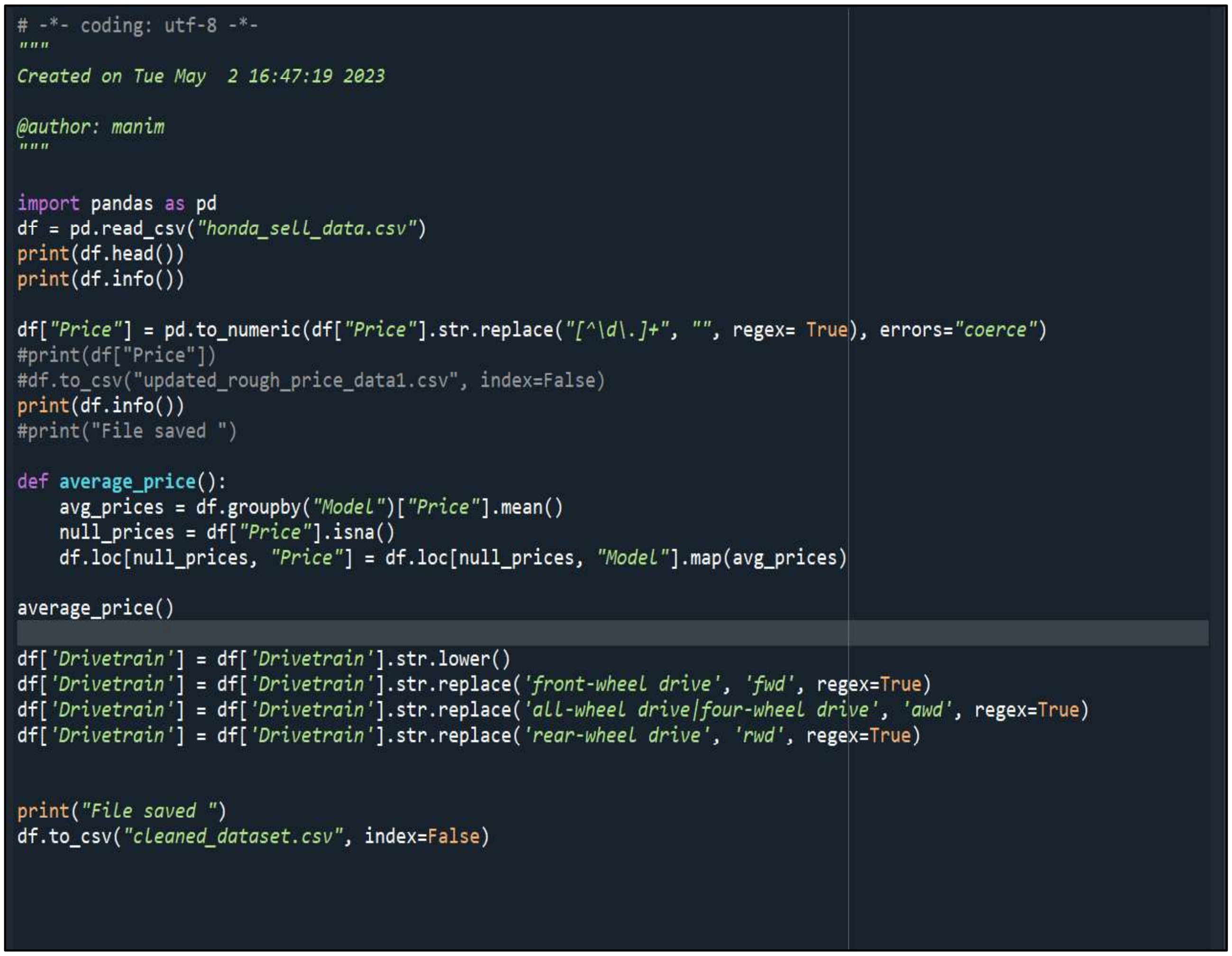


E
Price
$46,370
$34,150
$34,245
$46,500
$40,395
$42,250
$34,090
$39,845
$40,240
$34,700
$40,850
$53,375
$44,600
$34,640
$42,595
$53,720
$50,150
$40,939
$43,190

After Cleaning:



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



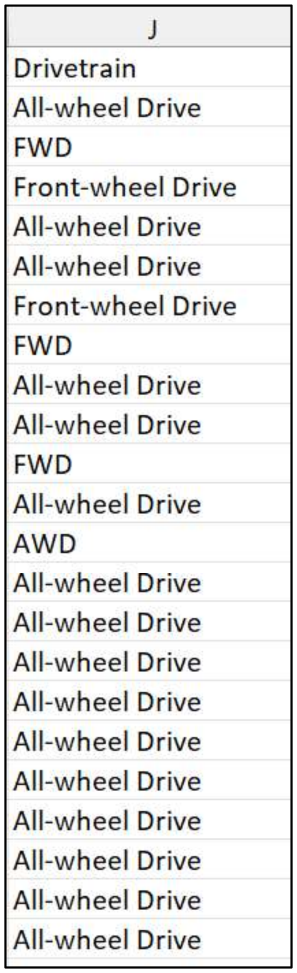
```python
# -*- coding: utf-8 -*-
"""
Created on Tue May  2 16:47:19 2023

@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.

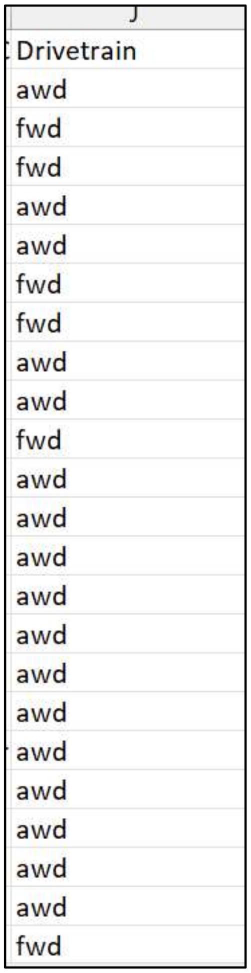
The code initially uses the **pd.to_numeric()** function to change the "Price" column in the dataframe from **a string to a numeric data type**. All non-numeric characters **(“,” “$”)** are eliminated from each text in the column using a regular expression. The errors option is configured to "coerce"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



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

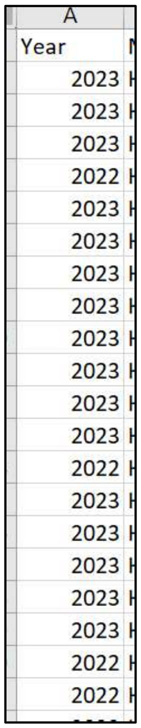
ii) **Function Used: Group by, Replace, to lower.**

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.

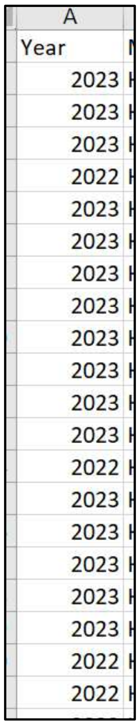
Column: Year

Before Cleaning:

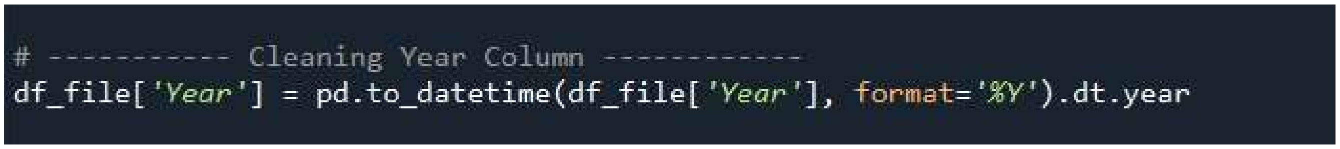


A
Year
2023
2023
2023
2022
2023
2023
2023
2023
2023
2023
2023
2023
2022
2023
2023
2023
2023
2023
2022
2022

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()

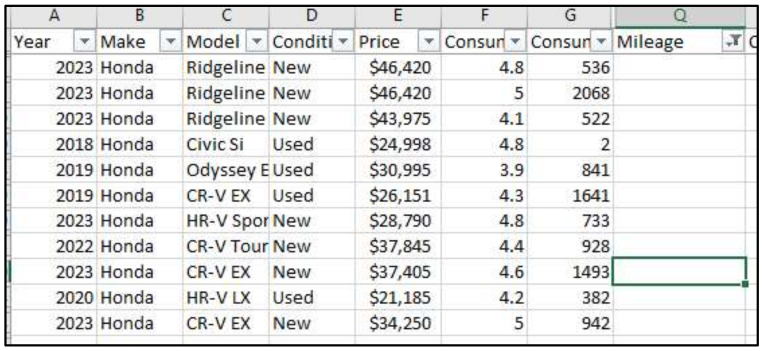
The data in column ‘Year’ is in String format. To use the ‘Year’ to plot the graphs, it needs to be converted to the ‘datetime’ format.

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.

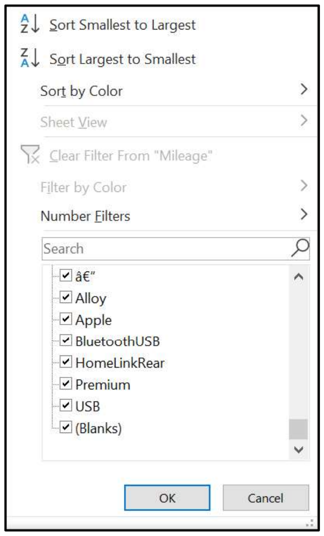
2. Handling Missing & Invalid Values

Column: Mileage

Before Cleaning:

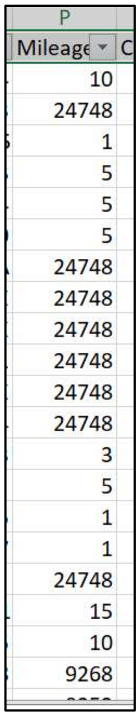


Year	Make	Model	Conditi	Price	Consur	Consur	Mileage
2023	Honda	Ridgeline	New	$46,420	4.8	536	
2023	Honda	Ridgeline	New	$46,420	5	2068	
2023	Honda	Ridgeline	New	$43,975	4.1	522	
2018	Honda	Civic Si	Used	$24,998	4.8	2	
2019	Honda	Odyssey E	Used	$30,995	3.9	841	
2019	Honda	CR-V EX	Used	$26,151	4.3	1641	
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	

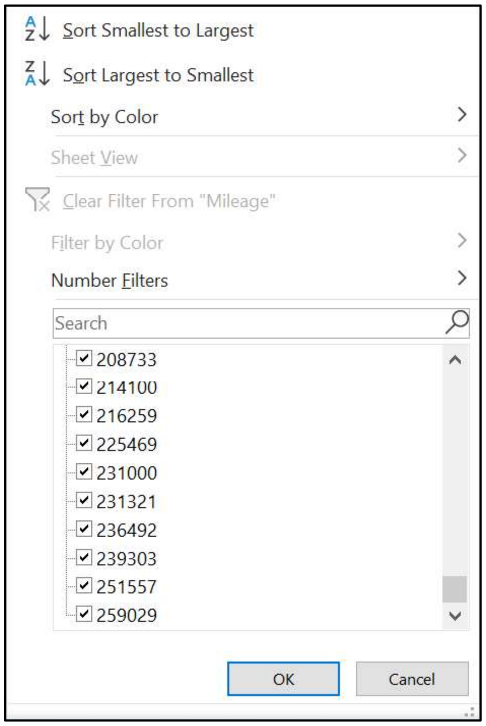


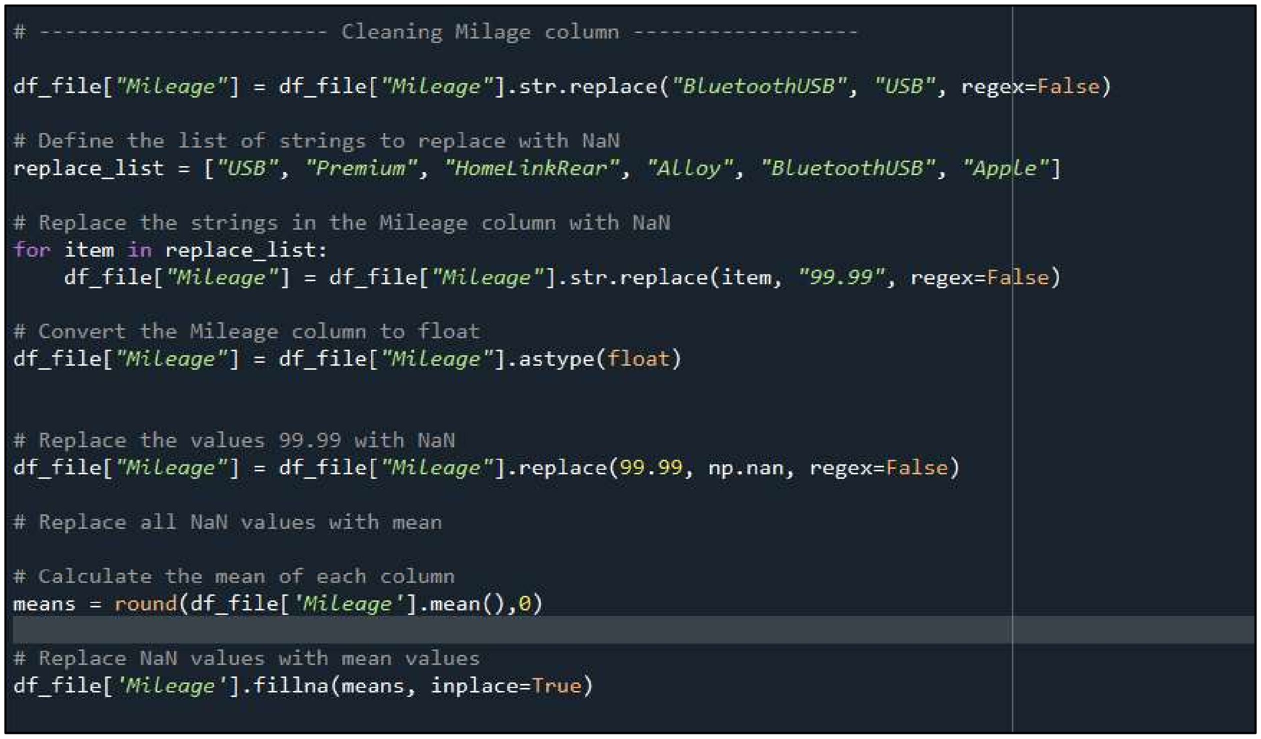
☑ â€"
☑ Alloy
☑ Apple
☑ BluetoothUSB
☑ HomeLinkRear
☑ Premium
☑ USB
☑ (Blanks)

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)
```

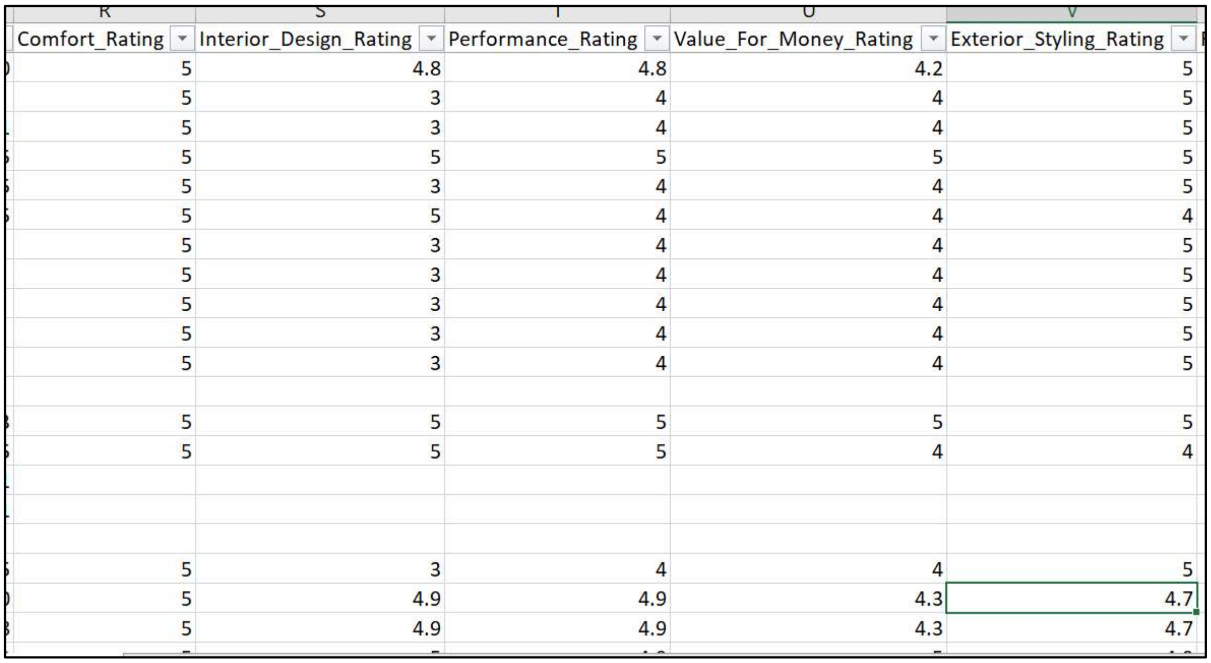
Functions Used: mean(), replace(), fillna()

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.

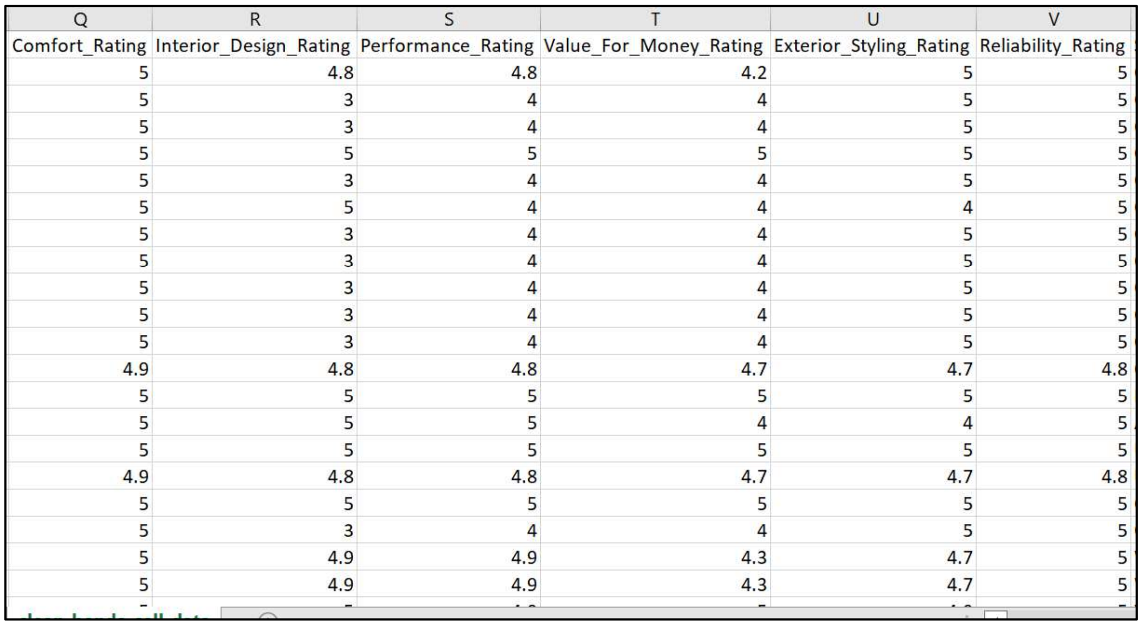
Columns: Comfort_Rating, Interior_Design_Rating, Performance_Rating, Value_For_Money_Rating, Exterior_Styling_Rating, Reliability_Rating

Before Cleaning:

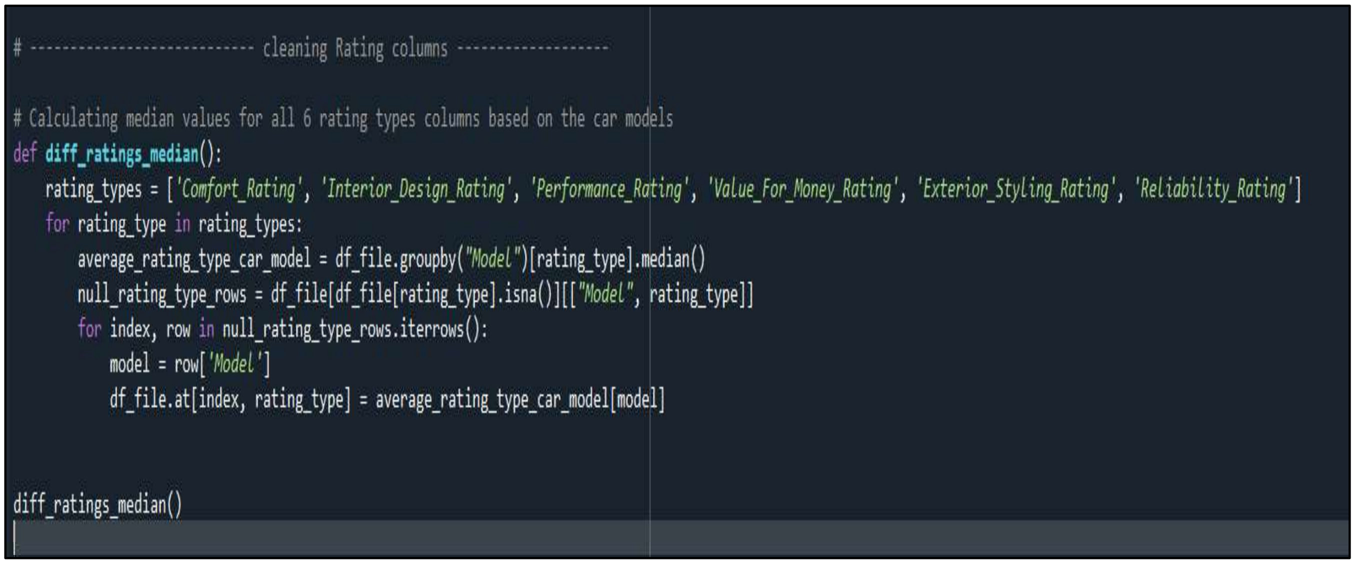


Comfort_Rating	Interior_Design_Rating	Performance_Rating	Value_For_Money_Rating	Exterior_Styling_Rating
5	4.8	4.8	4.2	5
5	3	4	4	5
5	3	4	4	5
5	5	5	5	5
5	3	4	4	5
5	5	4	4	4
5	3	4	4	5
5	3	4	4	5
5	3	4	4	5
5	3	4	4	5
5	3	4	4	5
5	5	5	5	5
5	5	5	4	4
5	3	4	4	5
5	4.9	4.9	4.3	4.7
5	4.9	4.9	4.3	4.7

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()
```

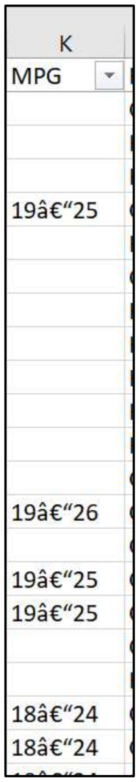
Functions Used: groupby(), median()

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.

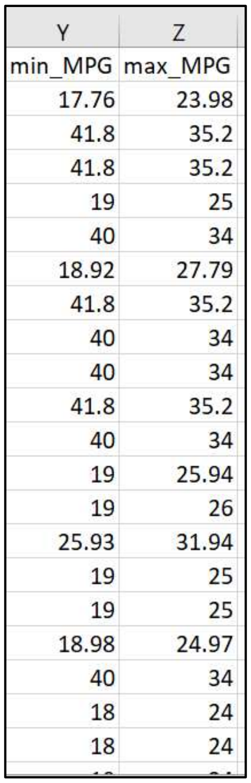
Column: MPG

Before Cleaning:



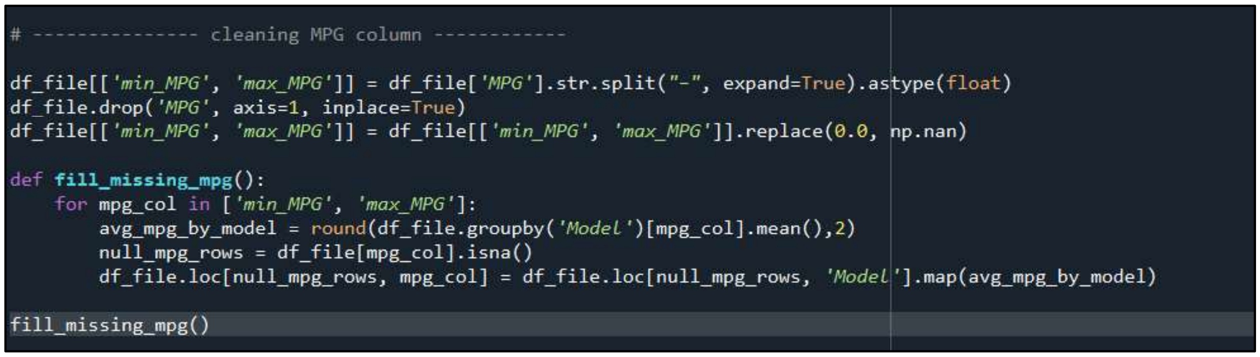
K
MPG
19â€“25
19â€“26
19â€“25
19â€“25
18â€“24
18â€“24

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.

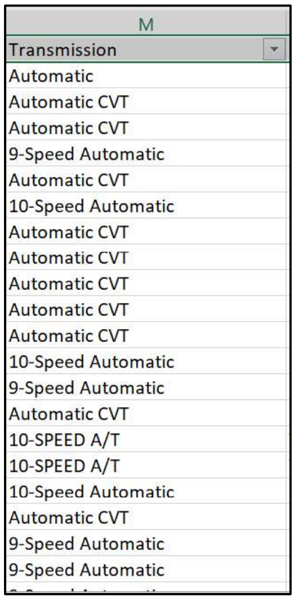
Next, any '0.0' values in the new columns are replaced with NaN to signify missing values.

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.

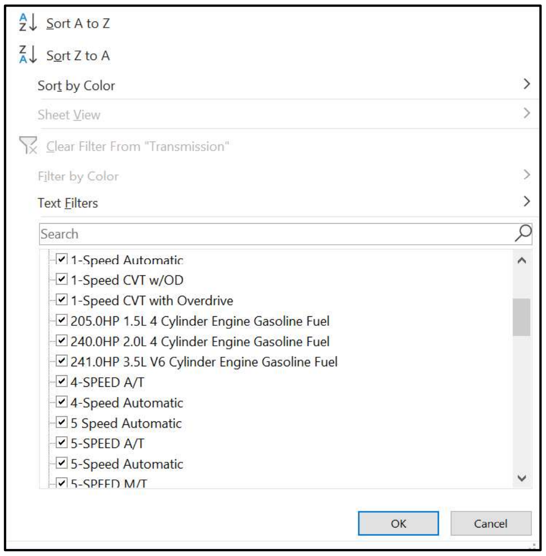
3. Data Standardization

Column: Transmission

Before Cleaning:

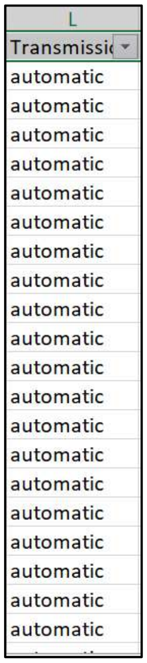


M
Transmission
Automatic
Automatic CVT
Automatic CVT
9-Speed Automatic
Automatic CVT
10-Speed Automatic
Automatic CVT
Automatic CVT
Automatic CVT
Automatic CVT
Automatic CVT
10-Speed Automatic
9-Speed Automatic
Automatic CVT
10-SPEED A/T
10-SPEED A/T
10-Speed Automatic
Automatic CVT
9-Speed Automatic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

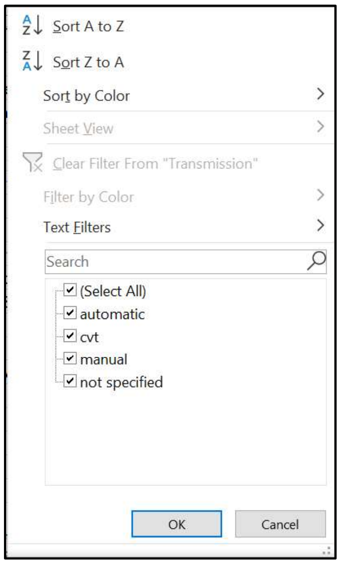


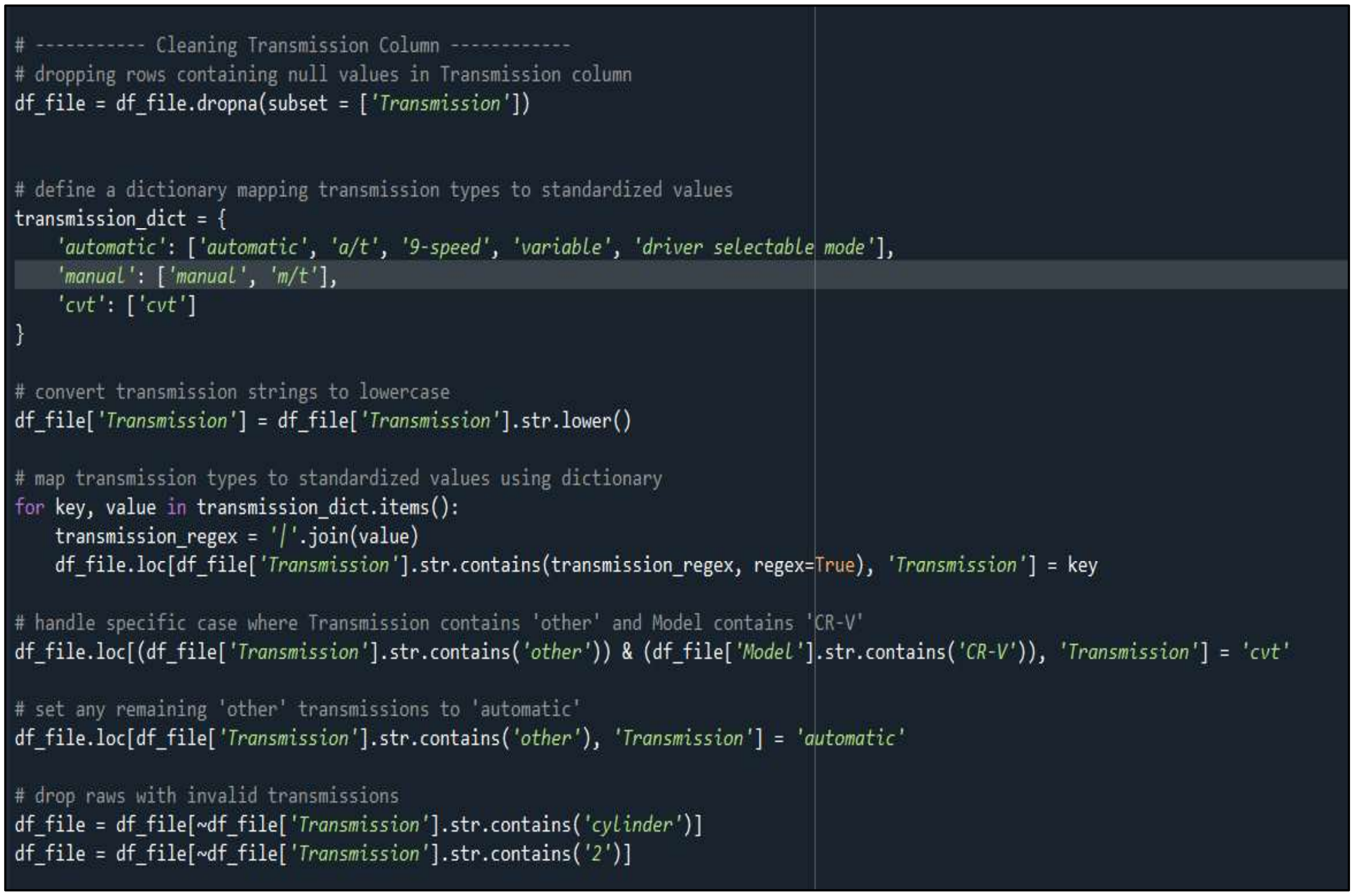
- ☑ 1-Speed Automatic
- ☑ 1-Speed CVT w/OD
- ☑ 1-Speed CVT with Overdrive
- ☑ 205.0HP 1.5L 4 Cylinder Engine Gasoline Fuel
- ☑ 240.0HP 2.0L 4 Cylinder Engine Gasoline Fuel
- ☑ 241.0HP 3.5L V6 Cylinder Engine Gasoline Fuel
- ☑ 4-SPEED A/T
- ☑ 4-Speed Automatic
- ☑ 5 Speed Automatic
- ☑ 5-SPEED A/T
- ☑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

After Cleaning:



L
Transmissio
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.

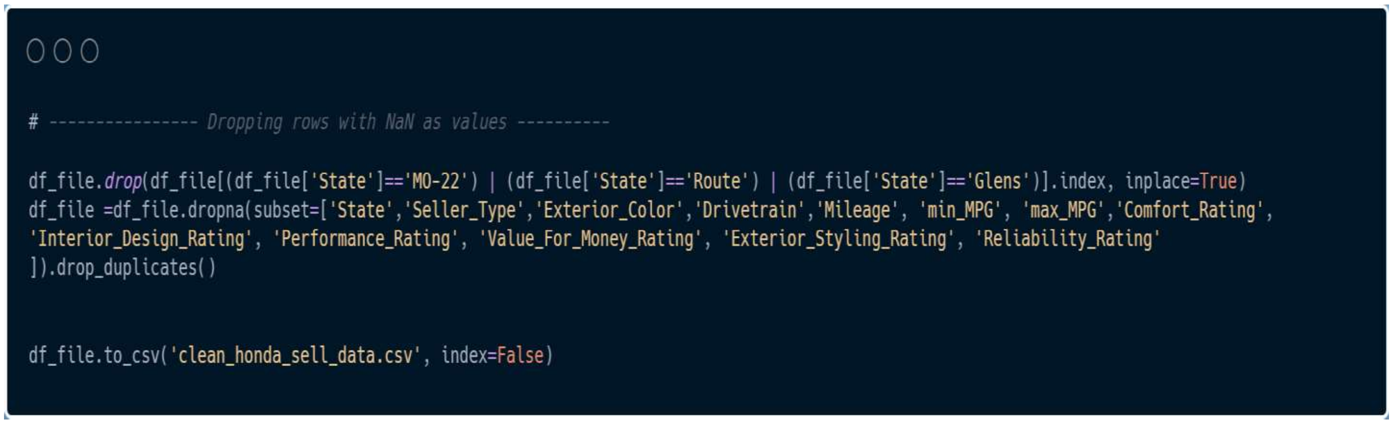
The values that contain 'automatic', 'a/t', '9-speed', 'variable', or 'driver selectable mode' are replaced by 'automatic', since they all belong to automatic transmission. Similarly, values that contain 'manual' and 'm/t' are replaced by 'manual', since they all belong to manual transmission.

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.