

Data Analysis and Visualization Report



“Forza Horizon 5 Cars” Dataset Analysis

Programing II

(BS) Financial Mathematics

Group Members:

- Shaharyar Khan
- Hammad Ali

<u>[S.No]</u>	<u>Contents:</u>	<u>[Page]</u>
1.	<u>Introduction:</u>	3
2.	<u>Data science:</u>	3
2.1.	<u>Importance of Data science:</u>	3
2.2.	<u>How data science is transforming business:</u>	4
2.3.	<u>The benefits of a data science platform:</u>	4
3.	<u>Who is a Data Scientist?</u>	5
3.1.	<u>Role of data scientist:</u>	5
4.	<u>Python for Data Science:</u>	6
4.1.	<u>Useful features of Python language:</u>	6
5.	<u>Jupyter Notebook:</u>	7
6.	<u>About The Game:</u>	7
7.	<u>Libraries Used For The Program:</u>	8
7.1.	<u>Pandas:</u>	8
7.2.	<u>Plotly Express:</u>	8
7.3.	<u>Matplotlib:</u>	8
7.4.	<u>Seaborn:</u>	9
7.5.	<u>IPython.display.HTML:</u>	9
8.	<u>Keywords:</u>	10
9.	<u>Analysing Forza Horizon 5 Cars Dataset:</u>	11

■ **Introduction:**

Data analysis and visualization refers to insight examination of any piece of data and represent it visually through different plots and graphs, which would be helpful for making improvement, future decisions and predictions.

This thesis aims to discover and understand the methods of analysis of excel data set through Python. The proposal of this thesis to analyses “Forza Horizon 5 Cars” dataset through Python environment “Jupyter notebook” and visualize important points from the dataset. This report will investigate different specification of all the cars in the dataset of this game.

■ **Data science:**

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data. Data science practitioners apply machine learning algorithms to numbers, text, images, video, audio, and more to produce artificial intelligence (AI) systems to perform tasks that ordinarily require human intelligence. In turn, these systems generate insights , which analysts and business users can translate into tangible business value.

Importance of Data science:

At a time when 1.7 megabytes of data are generated every second for every person on Earth, it’s crucial to know how to wade through information, and structure, interpret, and present it in a meaningful way.

This enormous volume of data, known as big data, has prompted greater demand for skilled data science professionals.

Data science is high in demand and explains how digital data is transforming businesses and helping them make sharper and critical decisions. So data that is digital is ubiquitous for people who are looking to work as data scientists.

How data science is transforming business:

Organizations are using data science to turn data into a competitive advantage by refining products and services. Data science and machine learning use cases include:

- Determine customer churn by analyzing data collected from call centers, so marketing can take action to retain them
- Improve efficiency by analyzing traffic patterns, weather conditions, and other factors so logistics companies can improve delivery speeds and reduce costs
- Improve patient diagnoses by analyzing medical test data and reported symptoms so doctors can diagnose diseases earlier and treat them more effectively
- Optimize the supply chain by predicting when equipment will break down
- Detect fraud in financial services by recognizing suspicious behaviors and anomalous actions
- Improve sales by creating recommendations for customers based upon previous purchases

Many companies have made data science a priority and are investing in it heavily.

The benefits of a data science platform:

A data science platform reduces redundancy and drives innovation by enabling teams to share code, results, and reports. It removes bottlenecks in the flow of work by simplifying management and incorporating best practices.

In general, the best data science platforms aim to:

- Make data scientists more productive by helping them accelerate and deliver models faster, and with less error
- Make it easier for data scientists to work with large volumes and varieties of data
- Deliver trusted, enterprise-grade artificial intelligence that's bias-free, auditable, and reproducible

Data science platforms are built for collaboration by a range of users including expert data scientists, citizen data scientists, data engineers, and machine learning engineers or specialists. For example, a data science platform might allow data

scientists to deploy models as APIs, making it easy to integrate them into different applications. Data scientists can access tools, data, and infrastructure without having to wait for IT.

▪ **Who is a Data Scientist?**

Data scientists are a new growing breed of professionals, highly in demand today. This term was introduced some years back by data leads to companies in LinkedIn and Facebook. And today, we have a huge influx of data scientist nerds working across verticals. This demand happened due to the sudden need to find brains who could wrangle with data and help make discoveries and ultimately empower organizations to make data-driven decisions.

Role of data scientist:

Typically, a data scientist's role comprises handling humongous amounts of data and then analyzing it using data-driven methodologies. Once they can make sense of the data, they bridge the business gaps by communicating it to the information technology, leadership teams and understanding the patterns and trends through visualizations. Data scientists also leverage Machine Learning and AI, use their programming knowledge around Java, Python, SQL, Big data Hadoop, and data mining. They require to have great communication skills to translate to the business their data discovery insights effectively.

▪ **Python for Data Science:**

Python is open source, interpreted, high level language and provides great approach for object-oriented programming. It is one of the best language used by data scientist for various data science projects/application. Python provide great functionality to deal with mathematics, statistics and scientific function. It provides great libraries to deals with data science application.

One of the main reasons why Python is widely used in the scientific and research communities is because of its ease of use and simple syntax which makes it easy to adapt for people who do not have an engineering background. It is also more suited for quick prototyping.

Useful features of Python language:

Following are some useful features of Python language:

- It uses the elegant syntax, hence the programs are easier to read.
- It is a simple to access language, which makes it easy to achieve the program working.
- The large standard library and community support.
- The interactive mode of Python makes its simple to test codes.
- In Python, it is also simple to extend the code by appending new modules that are implemented in other compiled language like C++ or C.
- Python is an expressive language which is possible to embed into applications to offer a programmable interface.
- Allows developer to run the code anywhere, including Windows, Mac OS X, UNIX, and Linux.
- It is free software in a couple of categories. It does not cost anything to use or download Pythons or to add it to the application.

▪ **Jupyter Notebook:**

The Jupyter Notebook is an open-source web application that allows data scientists to create and share documents that integrate live code, equations, computational output, visualizations, and other multimedia resources, along with explanatory text in a single document. You can use Jupyter Notebooks for all sorts of data science tasks including data cleaning and transformation, numerical simulation, exploratory data analysis, data visualization, statistical modeling, machine learning, deep learning, and much more.

■ About The Game:

Forza Horizon 5 is a 2021 racing video game developed by Playground Games and published by Xbox Game Studios. The twelfth main instalment of the **Forza** series, the game is set in a fictionalized representation of Mexico. It was released on 9 November 2021 for Microsoft Windows, Xbox One, and Xbox Series X/S.

The game received universal acclaim and became a huge commercial success upon release; it launched to over ten million players in the first week, the biggest-ever launch for an Xbox Game Studios game. The game was nominated for three jury-voted awards at The Game Awards 2021, winning all three of its nominations and tying with **Hazelight's It Takes Two** for most wins, and was also nominated for the public-voted Players' Voice award, which went to fellow Xbox Game Studios title **Halo Infinite**.

For more information about the you can visit:

<https://forzamotorsport.net/en-US/games/fh5>

■ Libraries Used For The Program:

○ Pandas:

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language. Pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

○ Plotly Express:

Plotly Express is the easy-to-use, high-level interface to Plotly, which operates on a variety of types of data and produces easy-to-style figures. Plotly Express provides functions to visualize a variety of types of data. Most functions such as `px.bar` or `px.scatter` expect to operate on column-oriented data of the type you might store in a Pandas DataFrame (in either "long" or "wide" format, see below). `px.imshow` operates on matrix-like data you might store in a numpy or xarray array and functions

like `px.choropleth` and `px.choropleth_mapbox` can operate on geographic data of the kind you might store in a GeoPandas `GeoDataFrame`.

○ Matplotlib:

Matplotlib is a multi-platform data visualization library built on NumPy arrays, and designed to work with the broader SciPy stack. One of Matplotlib's most important features is its ability to play well with many operating systems and graphics backends. Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish. This cross-platform, everything-to-everyone approach has been one of the great strengths of Matplotlib.

○ Seaborn:

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

○ IPython.display.HTML:

Create a display object given raw data.

When this object is returned by an expression or passed to the `display` function, it will result in the data being displayed in the frontend. The MIME type of the data should match the subclasses used, so the `Png` subclass should be used for 'image/png' data. If the data is a URL, the data will first be downloaded and then displayed. If

Parameters

- **data** (*unicode, str or bytes*) – The raw data or a URL or file to load the data from
- **url** (*unicode*) – A URL to download the data from.
- **filename** (*unicode*) – Path to a local file to load the data from.
- **metadata** (*dict*) – Dict of metadata associated to be the object when displayed

▪ **Keywords:**

- 1) **Csv:** A CSV (comma-separated values) file is a text file that has a specific format, which allows data to be saved in a table structured format.
- 2) **def:** used to define a function.
- 3) **src:** src attribute specifies the URL of the image.
- 4) **:** The HTML tag is used to embed an image in a web page.
- 5) **escape:** bool, default True Convert the characters <, >, and & to HTML-safe sequences.
- 6) **formatters:** list tuple or dict of one-param. functions, optional Formatter functions to apply to columns' elements by position or name.
- 7) **dict:** Python's efficient key/value hash table structure is called a "dict".
- 8) **Counter:** Counter will hold the count of each of the elements present in the container.
- 9) **.keys():** returns a view object that displays a list of all the keys in the dictionary in order of insertion.
- 10) **.values():** returns a list of all the values available in a given dictionary.
- 11) **.sort_values:** Pandas sort_values() function sorts a data frame in Ascending or Descending order of passed Column.
- 12) **hover_data:** The hover_data argument accepts a list of column names to be added to the hover tooltip.
- 13) **.drop:** The drop() function is used to drop specified labels from rows or columns.
- 14) **.loc:** loc is label-based, which means that we have to specify the name of the rows and columns that we need to filter out.

■ Analysing Forza Horizon 5 Cars Dataset:

This dataset is openly available on the website ‘[www.kaggle.com](https://www.kaggle.com/deepcontractor/froza-horizon-5-cars-dataset)’ (<https://www.kaggle.com/deepcontractor/froza-horizon-5-cars-dataset>). In this dataset, we will analyze about different specifications of all the cars available in this game and we will compare them and see it through different plots. For This analyzation, I have used Jupyter notebook.

The steps of analyzation are as follow:

- Step 1 (Importing Libraries):

```
In [1]: import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns
from IPython.display import Image, HTML
```

First, we will import all Libraries that are required in this program. These libraries will be used later in the program for the sake of analysis and visualization.

- Step 2 (Reading csv file):

```
In [2]: df = pd.read_csv(".\Documents\Python\Forza_Horizon_Cars.csv")
df.head()
```

Out[2]:

	Car_Image	Name_and_model	Model_type	In_Game_Price	car_source	stock_specs	Stock_Rating	Drive_Type	speed	ha
0	https://www.kudosprime.com/fh5/images/cars/sid...	2001 Acura Integra Type R	RETRO HOT HATCH	25,000	Autoshow	C	596.0	FWD	5.6	
1	https://www.kudosprime.com/fh5/images/cars/sid...	2002 Acura RSX Type S	RETRO HOT HATCH	25,000	Autoshow	C	585.0	FWD	5.6	
2	https://www.kudosprime.com/fh5/images/cars/sid...	2017 Acura NSX	MODERN SUPERCARS	170,000	Autoshow	S1	831.0	AWD	7.0	
3	https://www.kudosprime.com/fh5/images/cars/sid...	1973 Alpine A110 1600s	CLASSIC RALLY	98,000	Autoshow	C	550.0	RWD	5.0	
4	https://www.kudosprime.com/fh5/images/cars/sid...	2017 Alpine A110	MODERN SPORTS CARS	67,500	Wheelspin	B	694.0	RWD	6.0	

After importing libraries, we will read the csv file of our dataset and for this we have write the directory where the file is saved and the name of file in the inverted commas. Now to see the head of the dataset write in the next line, `df.head ()`.

- Step 3 (Converting Html to Image):

```
In [4]: #converting HTML to image
def path_to_image_html(path):
    return ''
HTML(df.to_html(escape=False ,formatters=dict(Car_Image=path_to_image_html)))
```

For converting Html into Images, we have already imported Image and HTML from Ipython.display. The return type of the function is the traditional **img** tag, that we use to render images on a webpage.

```
return ''
```






The reason for doing this is that if all the paths are converted into image tags, then at the end with the help of a built-in HTML method all this will be rendered as images. We will use the **to_html** method to convert the dataframe into an HTML table. There are some parameters that we need to play with first the **escape = False**, escapes the HTML tags, we need to call the **path_to_image_html** method, convert it into a dictionary, and assign it to the formatters built-in variable.

```
df.to_html(escape=False ,formatters=dict(Car_Image=path_to_image_html))
```

And the last thing to do is to call the built-in **HTML** method and pass the whole dataframe as the parameter.

```
HTML(df.to_html(escape=False ,formatters=dict(Car_Image=path_to_image_html)))
```

Out[4]:

	Car_Image	Name_and_model	Model_type	In_Game_Price	car_source	stock_specs	Stock_Rating	Drive_Type
0		2001 Acura Integra Type R	RETRO HOT HATCH	25,000	Autoshow	C	596.0	FWD
1		2002 Acura RSX Type S	RETRO HOT HATCH	25,000	Autoshow	C	585.0	FWD
2		2017 Acura NSX	MODERN SUPERCARS	170,000	Autoshow	S1	831.0	AWD
3		1973 Alpine A110 1600s	CLASSIC RALLY	98,000	Autoshow	C	550.0	RWD
4		2017 Alpine A110	MODERN SPORTS CARS	67,500	Wheelspin	B	694.0	RWD

Now the Html links in Car_image column are converted in to images.

- Step 4 (Analyzing types of car models used in the game):

```
In [6]: # count elements of Model_type
Model_count= dict(Counter(df['Model_type']))
print(Model_count)
```

For analyzing the types of models used in the game, we will store Model_type column from the dataframe along with 'dict' and 'Counter' in a separate variable. Counter will count each model type in the Model type column.

```
{'RETRO HOT HATCH': 14, 'MODERN SUPERCARS': 26, 'CLASSIC RALLY': 9, 'MODERN SPORTS CARS': 21, 'UNLIMITED BUGGIES': 6, 'CULT CAR S': 11, 'UNLIMITED OFFROAD': 12, 'EXTREME TRACK TOYS': 23, 'RARE CLASSICS': 13, 'SUPER GT': 12, 'TRACK TOYS': 32, 'HYPERCARS': 27, 'GT CARS': 6, 'RETRO RALLY': 10, 'RETRO SALOONS': 14, 'SUPER SALOONS': 26, 'SUPER HOT HATCH': 11, 'HOT HATCH': 9, 'CLASSIC SPORTS CARS': 11, 'VINTAGE RACERS': 6, 'SPORTS UTILITY HEROES': 16, 'RETRO SUPERCARS': 30, 'RETRO SPORTS CARS': 36, 'CLASSIC MUSCLE': 22, 'RETRO MUSCLE': 10, 'MODERN MUSCLE': 12, "UTV'S": 3, 'CLASSIC RACERS': 14, 'RODS AND CUSTOMS': 9, 'VANS AND UTILITY': 4, "PICK-UP & 4X4'S": 22, 'DRIFT CARS': 22, 'RALLY MONSTERS': 18, 'OFFROAD': 5, 'info_not_found': 1, 'TRUCKS': 2, 'MODERN RALLY': 10, 'CULT CLASSICS': 2, 'BUGGIES': 2}
```

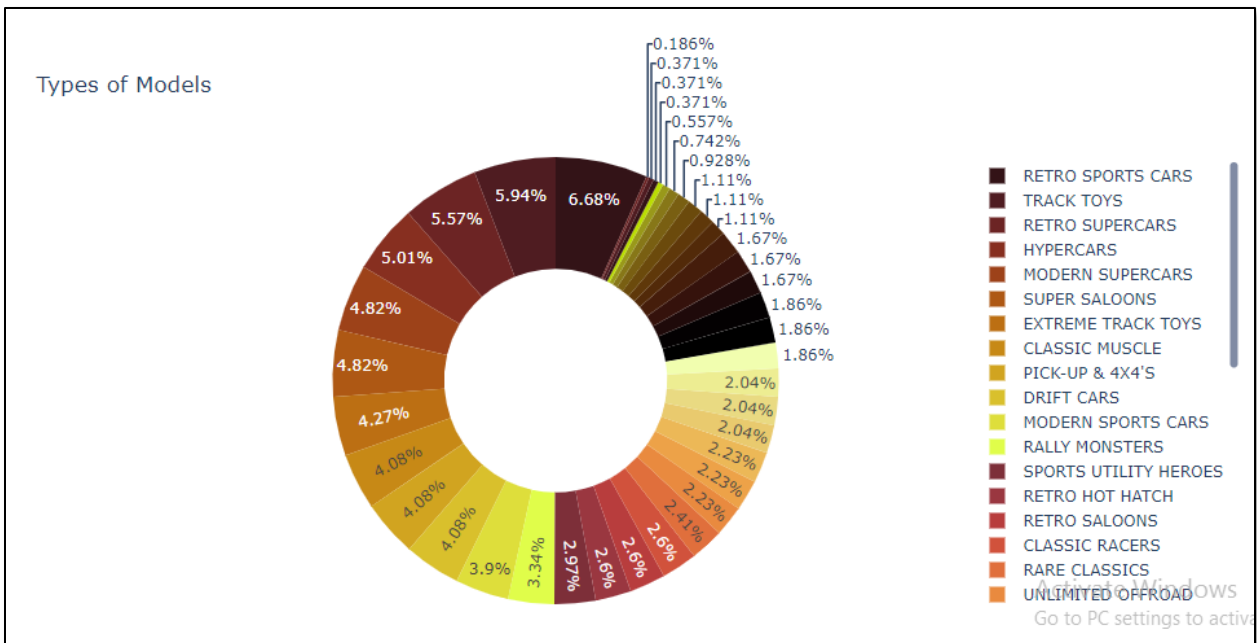
```
In [8]: # splitting Model_count into two lists
Model_count= {'Model_type': list(Model_count.keys()), 'count': list(Model_count.values())}
print(Model_count)
```

Now we will split the Model_count variable into two lists. In the first list 'Model_type' all the names of the car models are stored by using .keys() and all the counts of each model are stored in 'count' list by using .values().

```
{'Model_type': ['RETRO HOT HATCH', 'MODERN SUPERCARS', 'CLASSIC RALLY', 'MODERN SPORTS CARS', 'UNLIMITED BUGGIES', 'CULT CARS', 'UNLIMITED OFFROAD', 'EXTREME TRACK TOYS', 'RARE CLASSICS', 'SUPER GT', 'TRACK TOYS', 'HYPERCARS', 'GT CARS', 'RETRO RALLY', 'RETRO SALOONS', 'SUPER SALOONS', 'SUPER HOT HATCH', 'HOT HATCH', 'CLASSIC SPORTS CARS', 'VINTAGE RACERS', 'SPORTS UTILITY HEROES', 'RETRO SUPERCARS', 'RETRO SPORTS CARS', 'CLASSIC MUSCLE', 'RETRO MUSCLE', 'MODERN MUSCLE', "UTV'S", 'CLASSIC RACERS', 'RODS AND CUSTOMS', 'VANS AND UTILITY', "PICK-UP & 4X4'S", 'DRIFT CARS', 'RALLY MONSTERS', 'OFFROAD', 'info_not_found', 'TRUCKS', 'MODERN RALLY', 'CULT CLASSICS', 'BUGGIES'], 'count': [14, 26, 9, 21, 6, 11, 12, 23, 13, 12, 32, 27, 6, 10, 14, 26, 11, 9, 11, 6, 16, 30, 36, 22, 10, 12, 3, 14, 9, 4, 22, 22, 18, 5, 1, 2, 10, 2, 2]}
```

```
In [9]: #Visualize by pie chart
fig = px.pie(Model_count, values = 'count', names = 'Model_type', title = 'Types of Models',
             hole = 0.5, color_discrete_sequence = px.colors.sequential.solar)
fig.show()
```

Now we will visualize the result through pie chart by using the library Plotly.express that we have called with px.



- Step 5 (Analyzing Stock_specs):

```
In [10]: #count Stocks_specs column
Spec_count= dict(Counter(df['stock_specs']))
print(Spec_count)

{'C': 70, 'S1': 122, 'B': 109, 'D': 80, 'A': 113, 'S2': 44, 'info_not_found': 1}
```

Now to analyze the Stock specs of cars, we will again create a variable and store stock_specs column with Counter to count the number of each stock of car.

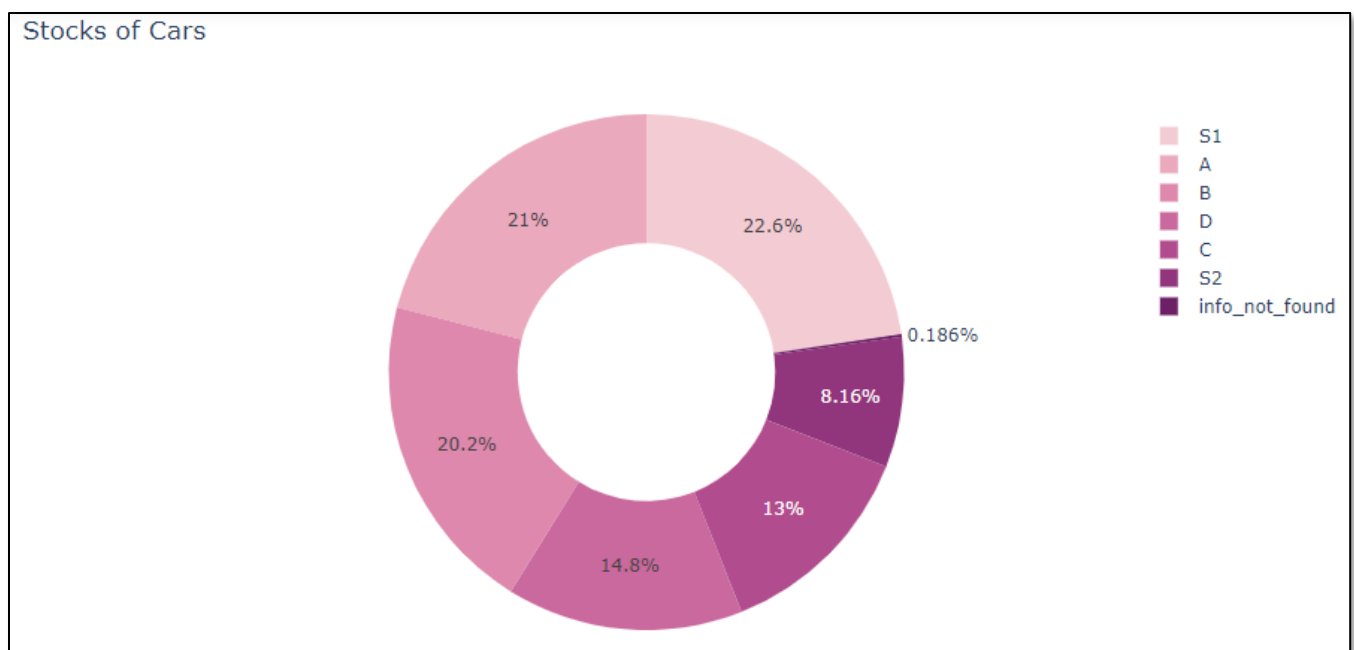
```
In [11]: # splitting into two list
Spec_count= dict(Counter(df['stock_specs']))
Spec_count= {'stock_specs': list(Spec_count.keys()), 'Cars': list(Spec_count.values())}
print(Spec_count)

{'stock_specs': ['C', 'S1', 'B', 'D', 'A', 'S2', 'info_not_found'], 'Cars': [70, 122, 109, 80, 113, 44, 1]}
```

we will split the Spec_count variable into two lists 'stock_specs' and 'Cars' for separating the names and the count of the specs of cars. After splitting the variable we will visualize it through pie chart.

```
In [12]: #Visualize by pie chart
fig = px.pie(Spec_count, values = 'Cars', names = 'stock_specs', title = 'Stocks of Cars',
             hole = 0.5, color_discrete_sequence = px.colors.sequential.Magenta)
fig.show()
```

To make a pie chart we will use the library plotly.express as px and the chart will be created from the Spec_count variable in which names and count of each stock of cars are stored.



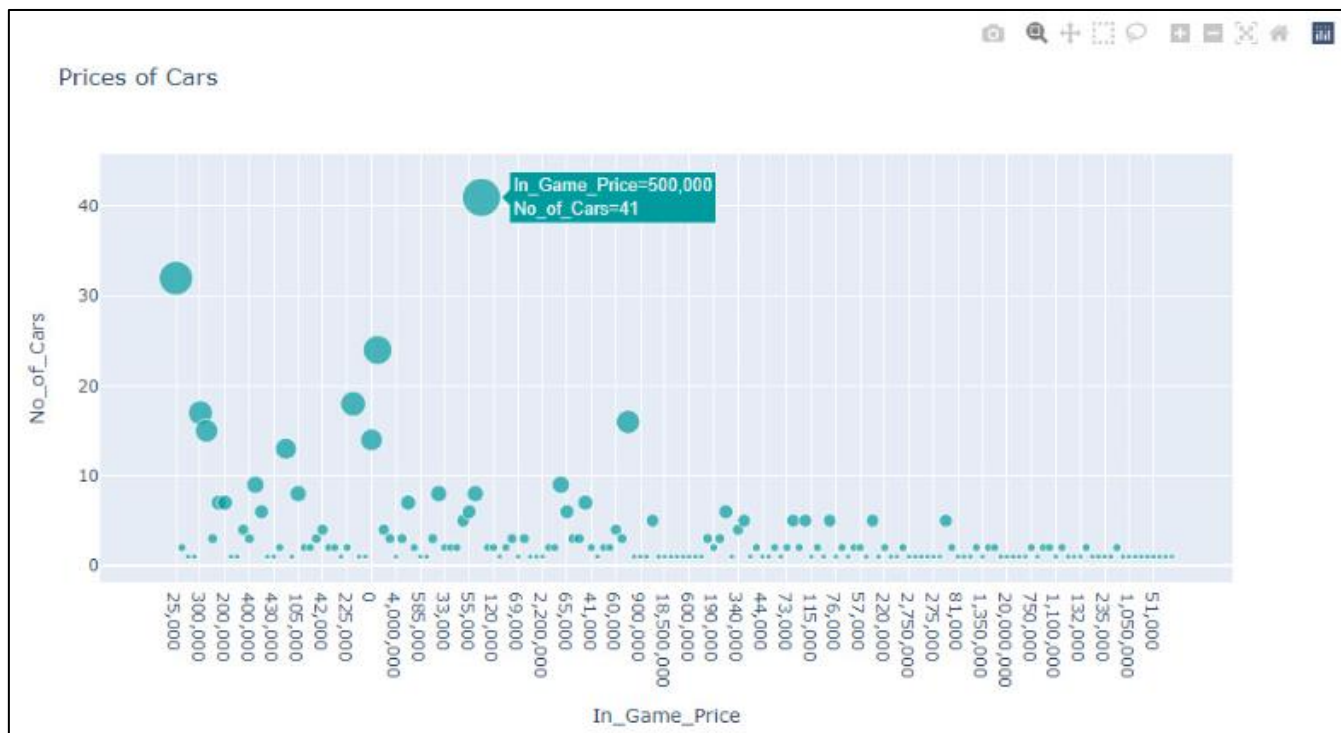
- Step 6 (Analyzing Cars available in specific price):

```
In [13]: # count of In_Game_Price column
Price_count= dict(Counter(df['In_Game_Price']))
Price_count= {'In_Game_Price': list(Price_count.keys()), 'No_of_Cars': list(Price_count.values())}
print(Price count)
```

To analyze that how many cars are available in a specific price we will take the counts of In Game Price column and split the Price count variable into two lists

```
In [14]: #Visualize by scatter plot
fig = px.scatter(Price_count, x = 'In_Game_Price', y = 'No_of_Cars' , title = 'Prices of Cars',
                 size= 'No_of_Cars', color_discrete_sequence = px.colors.diverging.Tropic)
fig.show()
```

After splitting into two lists we will visualize this through scatter plot using `plotly.express` library, 'In_Game_Price' is set on the x-axis and 'No_of_Cars' on the y-axis and we have used the size parameter for the size of scatter point according to the list 'No of Cars'.



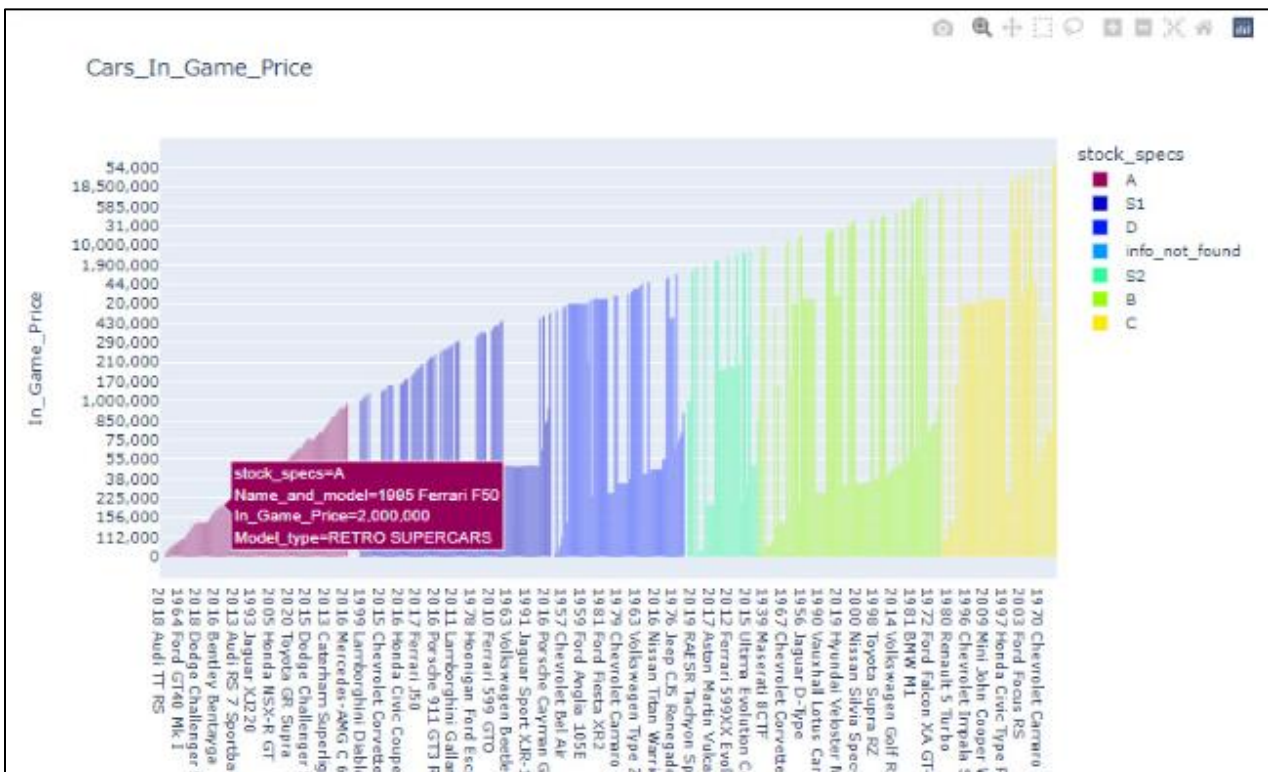
- Step 7 (Analyzing Prices of all the cars in the Game):

```
In [15]: # arranging values in ascending according with In_Game_Price column
Price=df.sort_values("In_Game_Price",ascending=True)
print(Price)
```

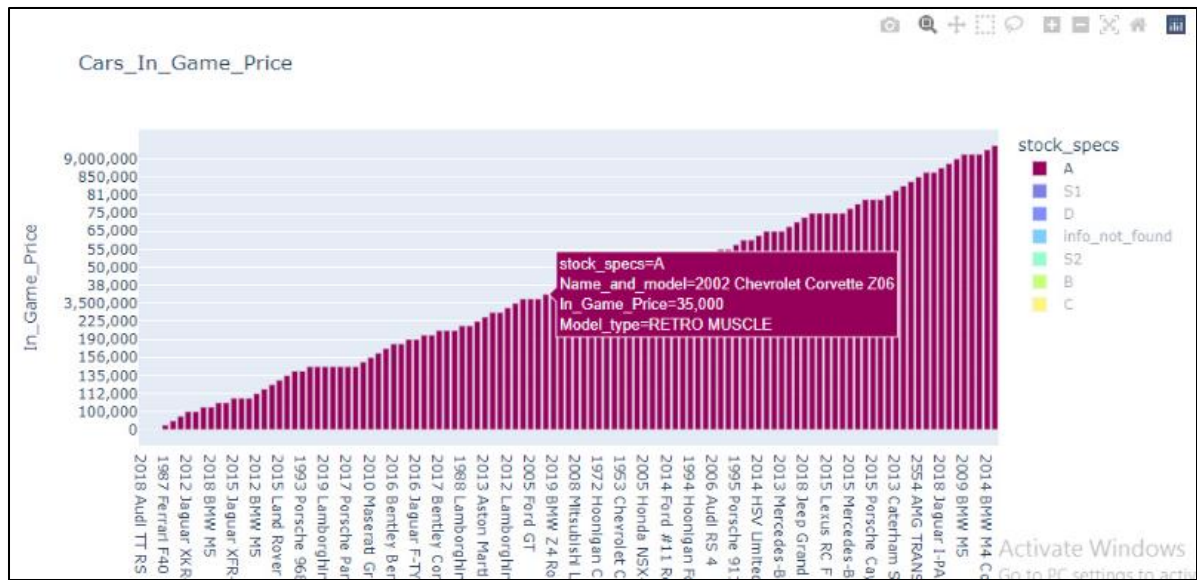
To analyze the prices of all the cars in the game we will store complete dataset in 'Price' variable and sort all values in ascending order according to 'In_Game_Price' column

```
In [16]: #Visualize Prices of all cars by bar chart
fig = px.bar(Price, x='Name_and_model', y='In_Game_Price',
             hover_data=['In_Game_Price', 'Name_and_model', 'Model_type'], color='stock_specs',
             labels={'price':'Car price'},title = 'Cars_In_Game_Price',
             height=800, color_discrete_sequence = px.colors.sequential.Rainbow)
fig.show()
```

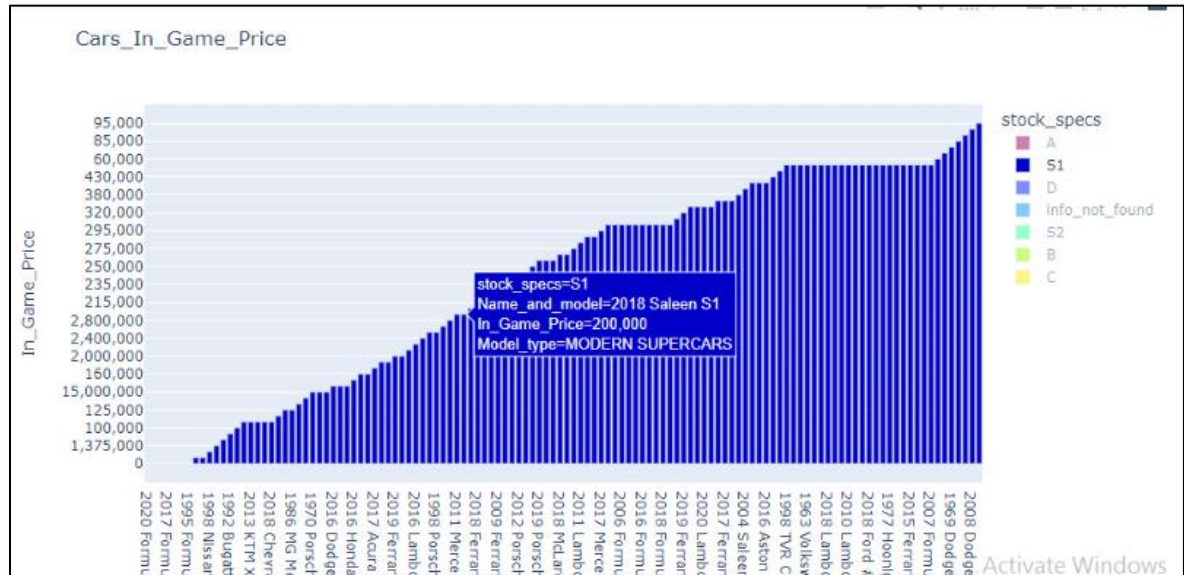
To visualize this through bar plot we will use 'px.bar'. On the x-axis, we will plot the 'Name_and_model' and on the y-axis we will plot 'In_Game_Price'. We will use 'hover_data' that includes the additional information of the columns mentioned in the hover_data and we will use color attribute to differentiate the plots according to 'stoc_specs' column.



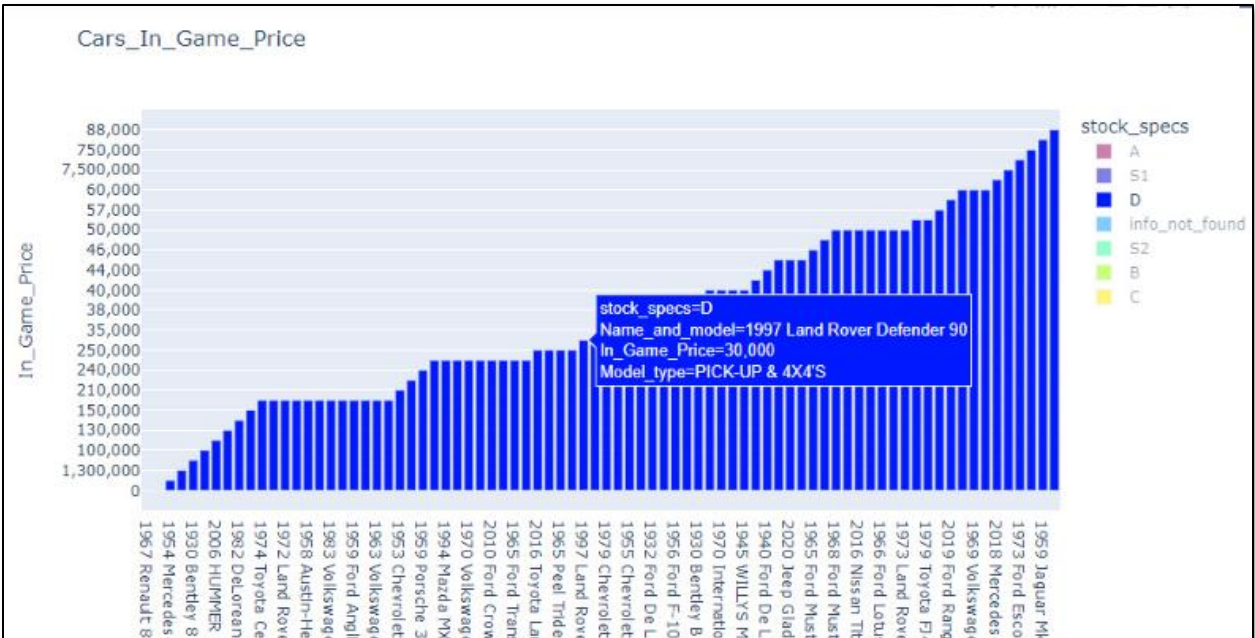
We can also see the separate plot of each stock of car by simply selecting stocks name on the left of the graph.



Stock A



Stock S1



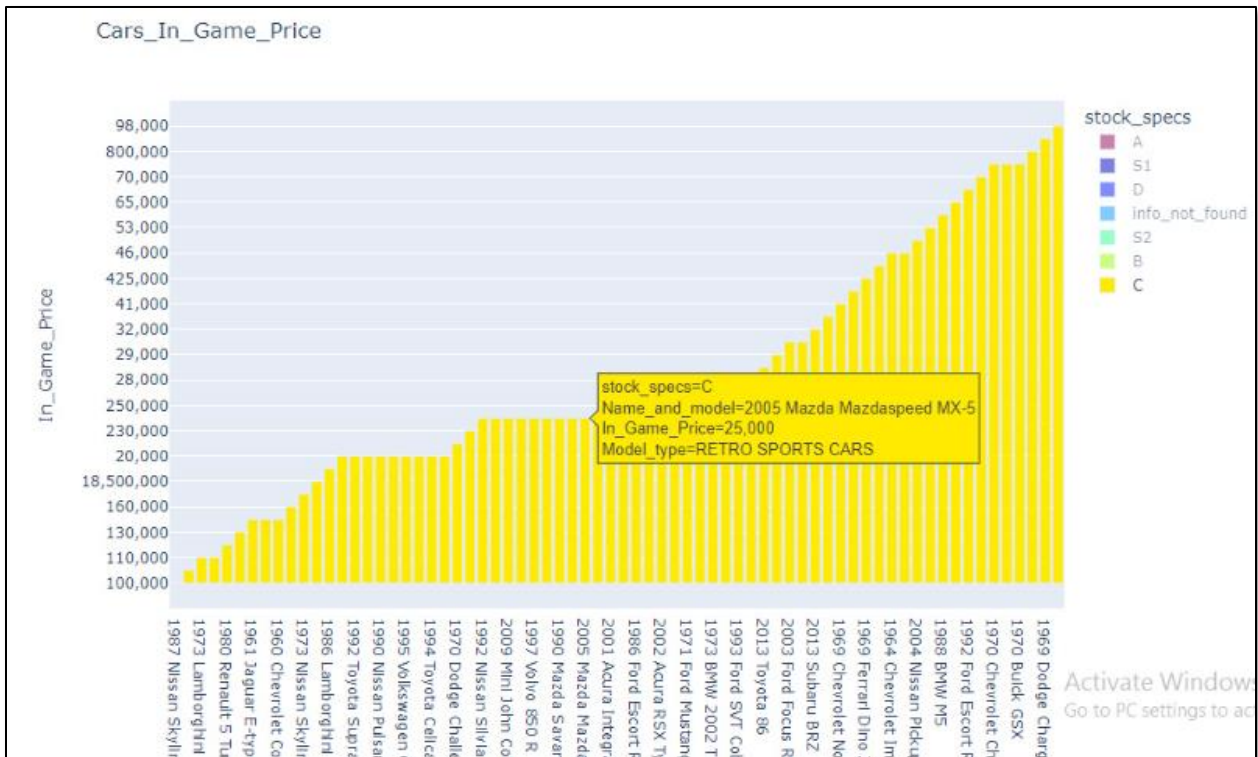
Stock D



Stock S2



Stock B



Stock C

- Step 8 (Analyzing cars by drive type):

```
In [57]: #count Drive_type column
dict(Counter(df['Drive_Type']))

Out[57]: {'FWD': 32, 'AWD': 157, 'RWD': 348, 'info_not_found': 2}
```

To analyze cars by their drive type we will use Counter to see the count of each drive type.

```
In [58]: #Visualize by Scatter plot
fig = px.scatter(df, x = 'Name_and_model', y = 'Drive_Type' , title = 'Categorize by Drive_Type',
                height=800 ,color='Model_type',hover_data=['Model_type', 'Name_and_model', 'stock_specs'],
                color_discrete_sequence = px.colors.sequential.Agsunset)
fig.show()
```

Now we will visualize it through scatter plot, by plotting Name_and_model on the x-axis and Drive_type on y-axis.



- Step 9 (Analyzing car_source, car_source_1, car_source_2):

In order to analyze 'car_source', 'car_source_1' and 'car_source_2' columns we will see the counts of each value in these columns by using '.value_counts()'.

```
In [19]: df['car_source'].value_counts()
Out[19]: Autoshow                431
         Wheelspin              48
         Season Event           21
         Barn                  13
         Car Collection          9
         Accolade               5
         Mastery Tree           5
         Other                  4
         This info will be available soon 3
         Name: car_source, dtype: int64
```

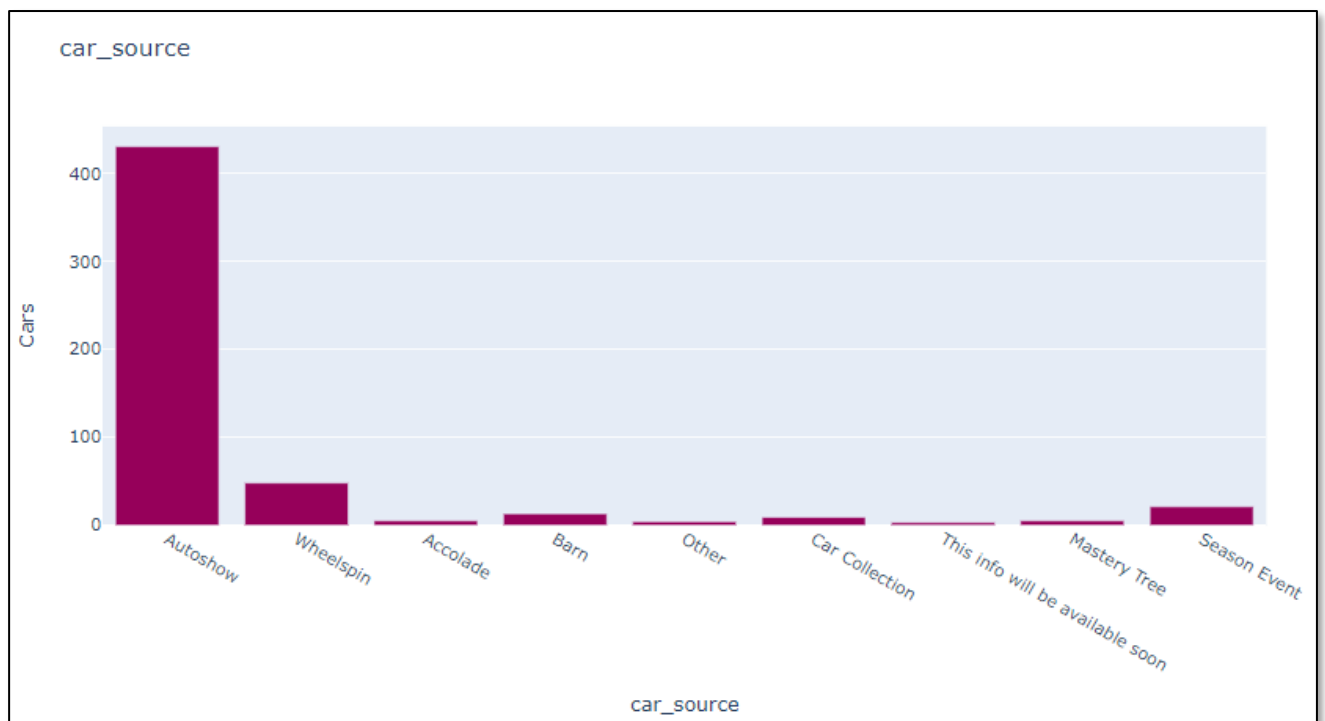
```
In [20]: df["car_source_1"].value_counts()
Out[20]: Wheelspin              390
         info_not_found         62
         Season Event           42
         Accolade               21
         Festival Playlist       21
         Other                   3
         Name: car_source_1, dtype: int64
```

```
In [21]: df["car_source_2"].value_counts()
Out[21]: info_not_found         416
         Accolade               51
         Festival Playlist       45
         Forza Shop             26
         Other                   1
         Name: car_source_2, dtype: int64
```

Now we will separate the count and the names of these columns by storing in each in separate variable and splitting into two lists. After separating counts and name for each of these columns, we will visualize them through bar plots.

```
In [62]: # separating car_source and its count in two lists
Source= dict(Counter(df['car_source']))
Source= {'car_source': list(Source.keys()), 'Cars': list(Source.values())}
Source
```

```
In [63]: # plotting car_source
fig1 = px.bar(Source, x='car_source', y='Cars', title = 'car_source',
              color_discrete_sequence = px.colors.sequential.Rainbow)
fig1.show()
```

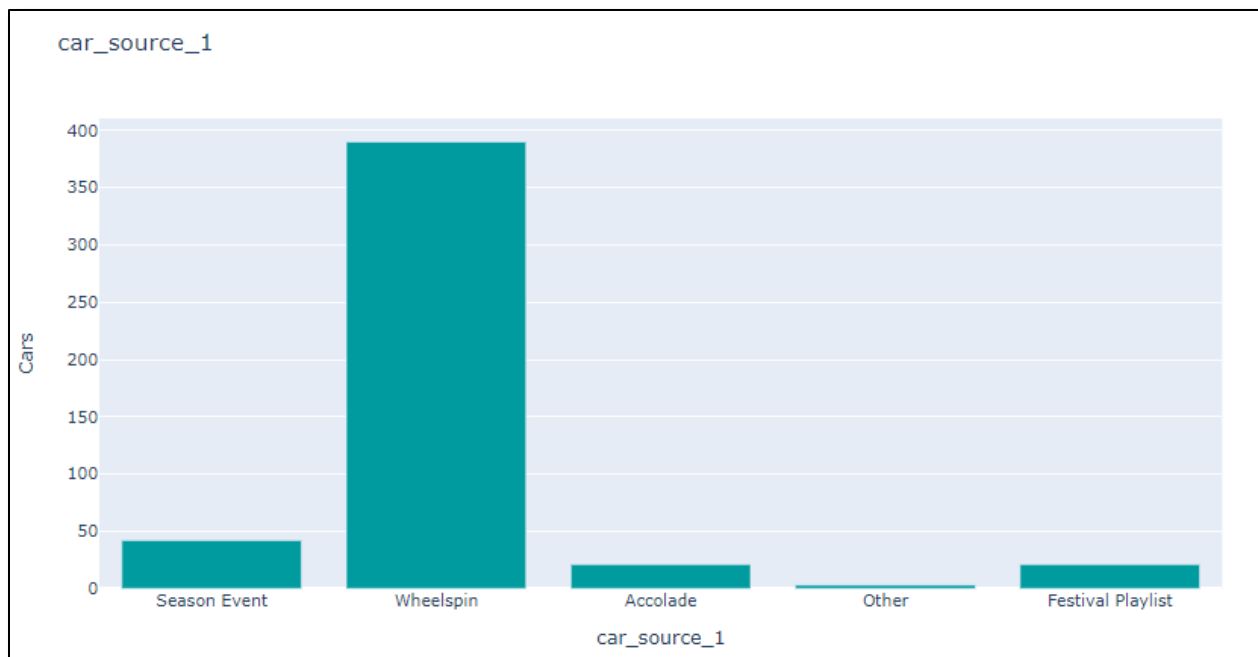



```
In [64]: # removing info_not_found from car_source_1
df.drop(df.loc[df['car_source_1'] == 'info_not_found'].index, inplace = True)
```

To remove the info_not_found from car_source_1 column we will use '.drop' and 'inplace = True' to fill the spaces.

```
In [65]: # separating car_source_1 and its count in two lists
Source1= dict(Counter(df['car_source_1']))
Source1= {'car_source_1': list(Source1.keys()), 'Cars': list(Source1.values())}
Source1
```

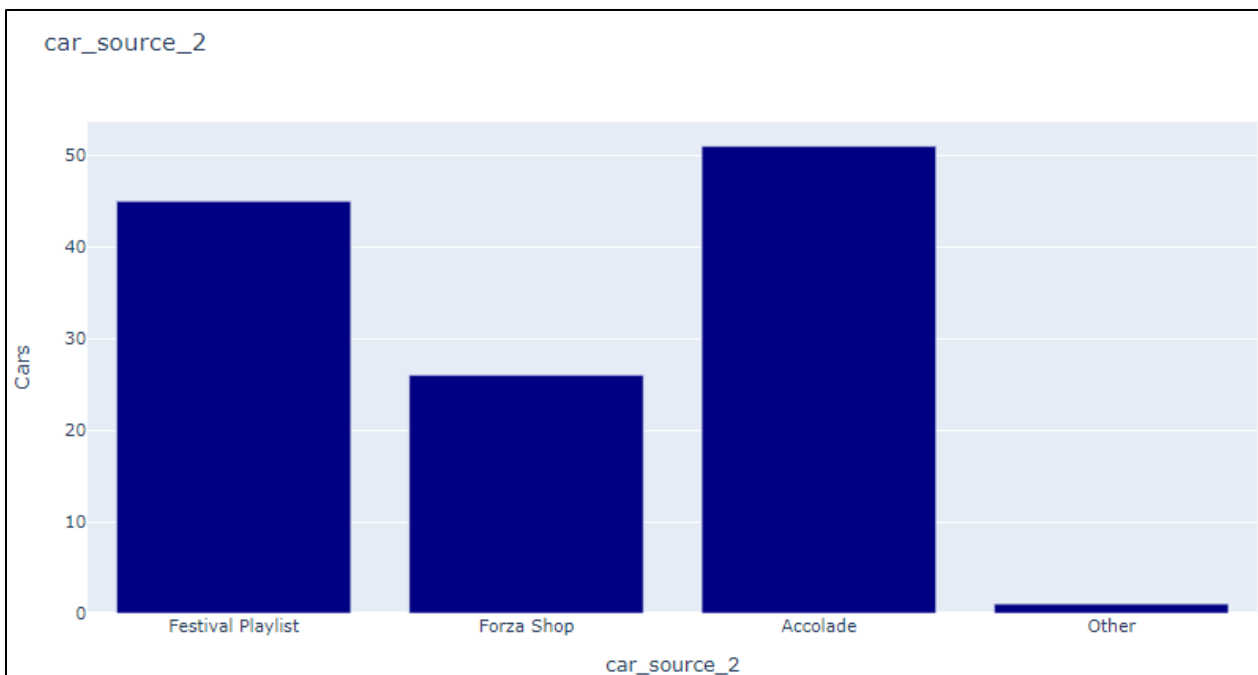
```
In [66]: # plotting car_source_1
fig2 = px.bar(Source1, x='car_source_1', y='Cars', title = 'car_source_1',
              color_discrete_sequence = px.colors.diverging.Tropic)
fig2.show()
```



```
In [67]: # removing info_not_found from car_source_2
df.drop(df.loc[df['car_source_2'] == 'info_not_found'].index, inplace = True)
```

```
In [68]: # separating car_source_2 and its count in two lists
Source2=dict(Counter(df['car_source_2']))
Source2= {'car_source_2': list(Source2.keys()), 'Cars': list(Source2.values())}
Source2
```

```
In [69]: # plotting car_source_2
fig3 = px.bar(Source2, x='car_source_2', y='Cars', title = 'car_source_2',
              color_discrete_sequence = px.colors.sequential.Jet)
fig3.show()
```



- Step 10 (Analyzing Correlation between some columns):

For analyzing Correlation between some columns, we will store some specific columns in a separate variable 'df1'.

```
In [71]: # separating columns in a variable
df1= df[['speed','handling', 'acceleration', 'launch', 'braking', 'Offroad','Horse_Power', 'Weight_lbs']]
df1
```

```
Out[71]:
```

	speed	handling	acceleration	launch	braking	Offroad	Horse_Power	Weight_lbs
4	6.0	6.4	5.4	5.7	4.2	4.2	248	2,432
8	7.4	10.0	7.7	8.4	10.0	3.9	780	2,756
16	7.7	9.6	6.5	7.0	10.0	3.2	820	2,998
19	7.0	6.7	6.0	6.4	6.1	4.6	503	3,497
29	7.3	7.0	7.1	6.6	6.9	5.0	542	3,682
...
509	4.1	3.8	2.5	2.2	2.7	9.5	161	4,993
511	8.7	6.7	5.4	5.7	6.6	3.4	800	2,150
512	6.7	6.2	5.3	5.5	5.0	4.3	406	2,377
515	6.6	4.7	4.1	2.4	3.2	4.7	379	3,642
535	6.1	5.9	5.5	3.6	3.9	5.1	346	3,985

123 rows × 8 columns

Now we will remove the string data from 'Horse_Power' and 'Weight_lbs' columns.

```
In [72]: # coverting Str to Int
df1['Weight_lbs'] = [int(float(str(i).replace(',',''))) for i in df['Weight_lbs']]
df1['Horse_Power'] = [int(float(str(i).replace(',',''))) for i in df['Horse_Power']]
df1
```

Out[72]:

	speed	handling	acceleration	launch	braking	Offroad	Horse_Power	Weight_lbs
4	6.0	6.4	5.4	5.7	4.2	4.2	248	2432
8	7.4	10.0	7.7	8.4	10.0	3.9	780	2756
16	7.7	9.6	6.5	7.0	10.0	3.2	820	2998
19	7.0	6.7	6.0	6.4	6.1	4.6	503	3497
29	7.3	7.0	7.1	6.6	6.9	5.0	542	3682
...
509	4.1	3.8	2.5	2.2	2.7	9.5	161	4993
511	8.7	6.7	5.4	5.7	6.6	3.4	800	2150
512	6.7	6.2	5.3	5.5	5.0	4.3	406	2377
515	6.6	4.7	4.1	2.4	3.2	4.7	379	3642
535	6.1	5.9	5.5	3.6	3.9	5.1	346	3985

123 rows × 8 columns

Now we will convert all the columns in the df1 variable into float form.

```
In [77]: #Converting Int to float
df1 = df1.astype({'speed':float, 'handling':float, 'acceleration':float, 'launch':float,
                  'braking':float, 'Offroad':float, 'Horse_Power':float, 'Weight_lbs':float})
df1.head()
```

Out[77]:

	speed	handling	acceleration	launch	braking	Offroad	Horse_Power	Weight_lbs
4	6.0	6.4	5.4	5.7	4.2	4.2	248.0	2432.0
8	7.4	10.0	7.7	8.4	10.0	3.9	780.0	2756.0
16	7.7	9.6	6.5	7.0	10.0	3.2	820.0	2998.0
19	7.0	6.7	6.0	6.4	6.1	4.6	503.0	3497.0
29	7.3	7.0	7.1	6.6	6.9	5.0	542.0	3682.0

For analyzing Correlation we will use '.corr()'

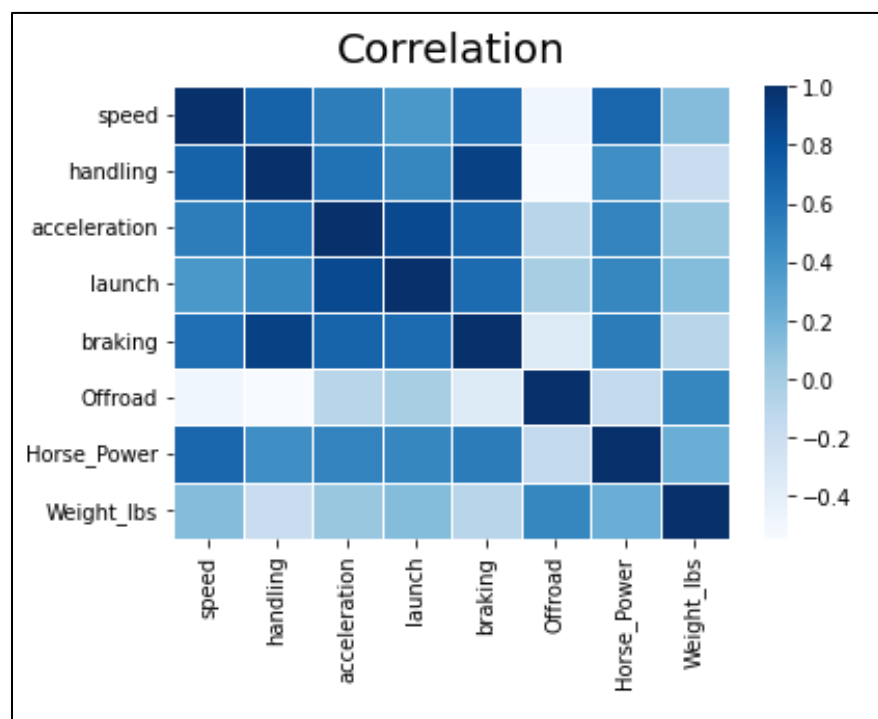
```
In [78]: # Analyzing Correlation between columns
df1.corr()
```

Out[78]:

	speed	handling	acceleration	launch	braking	Offroad	Horse_Power	Weight_lbs
speed	1.000000	0.692716	0.545070	0.381954	0.619464	-0.489164	0.676622	0.123589
handling	0.692716	1.000000	0.611534	0.491220	0.896290	-0.548287	0.436938	-0.186177
acceleration	0.545070	0.611534	1.000000	0.854390	0.687017	-0.091363	0.505764	0.053215
launch	0.381954	0.491220	0.854390	1.000000	0.648135	-0.011044	0.487905	0.132310
braking	0.619464	0.896290	0.687017	0.648135	1.000000	-0.338846	0.551424	-0.089443
Offroad	-0.489164	-0.548287	-0.091363	-0.011044	-0.338846	1.000000	-0.146245	0.488792
Horse_Power	0.676622	0.436938	0.505764	0.487905	0.551424	-0.146245	1.000000	0.228616
Weight_lbs	0.123589	-0.186177	0.053215	0.132310	-0.089443	0.488792	0.228616	1.000000

We have analyze the correlation of columns in df1 variable, now we will visualize it through heatmap using the library seaborn as sns.

```
In [79]: # Visualize Correlation by heatmap
sns.heatmap(df1.corr(), vmax = 1, linewidths = 1, cmap = 'Blues' )
plt.title('Correlation', fontsize = 20, pad = 12)
plt.show()
```



- Step 11 (Creating histogram of separated columns):

Now in the last, we will create a histogram of the separated columns in the df1 variable.

```
In [80]: #Creating histograms of separated columns  
df1.hist(figsize = (20,21))
```

```
Out[80]: array([[<AxesSubplot:title={'center':'speed'}>,  
                <AxesSubplot:title={'center':'handling'}>,  
                <AxesSubplot:title={'center':'acceleration'}>],  
               [<AxesSubplot:title={'center':'launch'}>,  
                <AxesSubplot:title={'center':'braking'}>,  
                <AxesSubplot:title={'center':'Offroad'}>],  
               [<AxesSubplot:title={'center':'Horse_Power'}>,  
                <AxesSubplot:title={'center':'Weight_lbs'}>],  
               <AxesSubplot:~>],  
            dtype=object)
```

