# REPORT

- Sauatbek Zhanserik 220103332 15-P
- Aktureyev Orazali 220103173 17-P
- Zharylkassynov Galymbek 220103251 15-P
- Abzalbek Mukhamedali 220103209 18-P
- Kantore Arman 220103282 19-P

## 1. Collecting the data

We decided to parse the kolesa.kz site to find some information related to cars in Kazakhstan. So I started the parsing stage.

First, I imported all the necessary libraries

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

```
base_url = "https://kolesa.kz/cars/?auto-car-grbody="
```

**base_url**: This is the base URL of the website being scraped, which displays car listings.

```
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
}
```

**headers**: This simulates a browser visit by adding a User-Agent. It prevents the server from blocking requests that appear to come from scripts.

```
all_car_data = []
```

**all_car_data**: An empty list to store the scraped car data.

```
max_pages = 100
page = 1
```

**max_pages**: The maximum number of pages to scrape.

**page**: A counter to keep track of the current page being scraped.

```
while page <= max_pages:
```

This loop iterates over pages of car listings until the specified maximum (max_pages) is reached or there are no more listings.

```
url = f"{base_url}&page={page}"
```

**url**: Constructs the URL for the current page by appending the page number to the `base_url`.

```
response = requests.get(url, headers=headers)
response.raise_for_status()
print(f"Page {page} fetched successfully.")
```

**`requests.get`**: Sends a GET request to the URL.

**`raise_for_status`**: Throws an exception if the request fails.

```
soup = BeautifulSoup(response.text, "html.parser")
```

**`BeautifulSoup`**: Parses the HTML content of the webpage, making it easier to navigate and extract data.

```
car_listings = soup.find_all("div", class_="a-list__item")


if not car_listings:
    print("No more listings found. Ending scraping.")
    break
```

**`find_all`**: Searches for all `div` elements with the class `a-list__item`, which likely represents individual car listings on the page.

```
for car in car_listings:
    try:

        title = car.find("h5", class_="a-card__title").get_text(strip=True)


        price = car.find("span", class_="a-card__price").get_text(strip=True)


        details = car.find("p", class_="a-card__description").get_text(strip=True)


        location = car.find("span", class_="a-card__param").get_text(strip=True)
```

**`find`**: Locates specific elements within each car listing.

- **`title`**: The car's title or model, found in an <h5> tag with the class `a-card__title`.
- **`price`**: The car's price, found in a `<span>` tag with the class `a-card__price`.
- **`details`**: Additional details about the car, found in a <p> tag with the class `a-card__description`.
- **`location`**: The car's location, found in a `<span>` tag with the class `a-card__param`.

```
        all_car_data.append({
            "Title": title,
            "Price": price,
            "Details": details,
            "Location": location
        })
    except AttributeError:
        continue
```

The extracted details are stored as a dictionary and appended to the `all_car_data` list.

```
page += 1
```

Moves to the next page for scraping.

```
df = pd.DataFrame(all_car_data)
```

**`pd.DataFrame`**: Converts the list of dictionaries (`all_car_data`) into a structured DataFrame, making the data easier to analyze or export.

```
df.to_csv('test.csv')
```

```
df = pd.read_csv(r"C:\Users\Admin\test.csv")
df.head()
```

| | Unnamed: 0 | Title | Price | Details | Location |
|---|---|---|---|---|---|
| **0** | 0 | ВАЗ (Lada) Lada 2121 | 5 950 000₸ | 2020 г., Б/у внедорожник, 1.7 л, бензин, КПП м... | Семей |
| **1** | 1 | Subaru Legacy | 6 500 000₸ | 2004 г., Б/у седан, 2 л, бензин, Правый руль, ... | Алматы |
| **2** | 2 | Mercedes-Benz C 180 | 2 200 000₸ | 1996 г., Б/у седан, 1.8 л, бензин, КПП механик... | Жанаозен |
| **3** | 3 | Genesis G70 | 23 500 000₸ | 2021 г., Б/у седан, 3.3 л, бензин, КПП автомат... | Алматы |
| **4** | 4 | Hyundai Santa Fe | 14 790 152₸ | 2021 г., Б/у кроссовер, 2.5 л, бензин, КПП авт... | Павлодар |

Because my parsed dataset was not perfect I saved it like test.csv to do a future data preprocessing stage.

2. Data preprocessing

```
df[['Brand', 'Model']] = df['Title'].str.extract(r'^(?P<Brand>\S+)\s+(?P<Model>.+)$')
```

I divided the Title column to Brand and Model columns for future visualizations part,

```
details_split = df['Details'].str.extract(
    r'(?P<Year>\d{4}) г\., (?P<Condition>Б/у|Новые)? ?(?P<Type>[^,]+), (?P<Engine_Size>\d+\.\d+ л), (?P<Fuel_Type>[^,]+), (?P<Transmission>КПП [^,]+), с
)


df = pd.concat([df, details_split], axis=1)

df.head()
```

There I also divided the details column into several other columns with features and info about the cars and conceited it with my dataframe.

```
df.isna().sum()
```

```
Unnamed: 0         0
Title              0
Price              0
Details            0
Location           0
Brand              0
Model              0
Year            1048
Condition       1048
Type            1048
Engine_Size     1048
Fuel_Type       1048
Transmission    1048
Mileage         1048
Color           1048
Additional      1048
dtype: int64
```

```
df = df.dropna()
```

I saw a lot of missing values and I decided to drop them all because it would be very problematic to fill them with unknown.

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 952 entries, 0 to 1999
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Unnamed: 0    952 non-null    int64 
 1   Title         952 non-null    object
 2   Price         952 non-null    object
 3   Details       952 non-null    object
 4   Location      952 non-null    object
 5   Brand         952 non-null    object
 6   Model         952 non-null    object
 7   Year          952 non-null    object
 8   Condition     952 non-null    object
 9   Type          952 non-null    object
 10  Engine_Size   952 non-null    object
 11  Fuel_Type     952 non-null    object
 12  Transmission  952 non-null    object
 13  Mileage       952 non-null    object
 14  Color         952 non-null    object
 15  Additional    952 non-null    object
dtypes: int64(1), object(15)
memory usage: 126.4+ KB
```

There you see that all the types of columns are in the object type, which means I should change the types where it's necessary.

```python
df = df.drop('Details', axis=1)
```

```python
df = df.drop('Unnamed: 0', axis=1)
```

There I dropped unnecessary columns

```
df.head()
```

| | Title | Price | Location | Brand | Model | Year | Condition | Type | Engine_Size | Fuel_Type | Transmission | Mileage | Color | Additional |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ВАЗ (Lada) Lada 2121 | 5 950 000₸ | Семей | ВАЗ | (Lada) Lada 2121 | 2020 | Б/у | внедорожник | 1.7 л | бензин | КПП механика | 29 000 | белый | литые диски, тонировка, ветровики,... |
| **3** | Genesis G70 | 23 500 000₸ | Алматы | Genesis | G70 | 2021 | Б/у | седан | 3.3 л | бензин | КПП автомат | 16 500 | белый | металлик |
| **4** | Hyundai Santa Fe | 14 790 152₸ | Павлодар | Hyundai | Santa Fe | 2021 | Б/у | кроссовер | 2.5 л | бензин | КПП автомат | 107 000 | белый | литые диски, ксенон, кожа, аудиосист... |
| **7** | Ravon Nexia R3 | 5 200 000₸ | Кызылорда | Ravon | Nexia R3 | 2020 | Б/у | седан | 1.5 л | бензин | КПП автомат | 118 000 | серый | комбинированный, климат-контроль, Машина... |
| **11** | Toyota Camry | 18 000 000₸ | Астана | Toyota | Camry | 2020 | Б/у | седан | 2.5 л | бензин | КПП автомат | 39 000 | белый | металлик, кожа, кондиционер, Вся родная и... |

That's how the dataset looks so far.

```
desired_order = [
    'Title', 'Brand', 'Model', 'Year', 'Price', 'Condition', 'Type', 'Engine_Size', 'Fuel_Type',
    'Transmission', 'Mileage', 'Color', 'Additional'
]

df = df[desired_order]
```

I decided to change the order of the columns for better readability.

```
df['Price'] = df['Price'].str.replace(r'\D', '', regex=True).astype(int)
```

There I extracted the value of the price column and change the data type into an integer.

```
df['Engine_Size'] = df['Engine_Size'].str.extract(r'(\d+\.\d+|\d+)').astype(float)
```

I did the same to the engine_size column.

```
df['Mileage'] = df['Mileage'].str.replace(' ', '').astype(int)
```

and for the mileage column.
This is the final dataset

| | Title | Brand | Model | Year | Price | Condition | Type | Engine_Size | Fuel_Type | Transmission | Mileage | Color | Additional |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ВАЗ (Lada) Lada 2121 | ВАЗ | (Lada) Lada 2121 | 2020 | 5950000 | Б/у | внедорожник | 1.7 | бензин | КПП механика | 29000 | белый | литые диски, тонировка, ветровики,... |
| **3** | Genesis G70 | Genesis | G70 | 2021 | 23500000 | Б/у | седан | 3.3 | бензин | КПП автомат | 16500 | белый | металлик |
| **4** | Hyundai Santa Fe | Hyundai | Santa Fe | 2021 | 14790152 | Б/у | кроссовер | 2.5 | бензин | КПП автомат | 107000 | белый | литые диски, ксенон, кожа, аудиосист... |
| **7** | Ravon Nexia R3 | Ravon | Nexia R3 | 2020 | 5200000 | Б/у | седан | 1.5 | бензин | КПП автомат | 118000 | серый | комбинированный, климат-контроль, Машина... |
| **11** | Toyota Camry | Toyota | Camry | 2020 | 18000000 | Б/у | седан | 2.5 | бензин | КПП автомат | 39000 | белый | металлик, кожа, кондиционер, Вся родная и... |
| **12** | Volkswagen Golf | Volkswagen | Golf | 1998 | 2550000 | Б/у | хэтчбек | 1.8 | бензин | КПП автомат | 241524 | белый | металлик, литые диски, тонировка, спой... |
| **14** | ВАЗ (Lada) Priora 2170 | ВАЗ | (Lada) Priora 2170 | 2015 | 3950000 | Б/у | седан | 1.6 | бензин | КПП механика | 174547 | серебристый | металлик, литые диски, тонировка,... |
| **16** | Toyota Highlander | Toyota | Highlander | 2017 | 19500000 | Б/у | кроссовер | 3.5 | бензин | КПП автомат | 117000 | черный | металлик, литые диски, тонировка, л... |

```
df.to_csv('python_final2.csv')
```

Then I saved the final dataset to a .csv

```
df = df.drop('Unnamed: 0', axis=1)
```

```
df.head()
```

| | Title | Brand | Model | Year | Price | Condition | Type | Engine_Size | Fuel_Type | Transmission | Mileage | Color | Additional |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ВАЗ (Lada) Lada 2121 | ВАЗ | (Lada) Lada 2121 | 2020 | 5950000 | Б/у | внедорожник | 1.7 | бензин | КПП механика | 29000 | белый | литые диски, тонировка, ветровики,... |
| 1 | Genesis G70 | Genesis | G70 | 2021 | 23500000 | Б/у | седан | 3.3 | бензин | КПП автомат | 16500 | белый | металлик |
| 2 | Hyundai Santa Fe | Hyundai | Santa Fe | 2021 | 14790152 | Б/у | кроссовер | 2.5 | бензин | КПП автомат | 107000 | белый | литые диски, ксенон, кожа, аудиосист... |
| 3 | Ravon Nexia R3 | Ravon | Nexia R3 | 2020 | 5200000 | Б/у | седан | 1.5 | бензин | КПП автомат | 118000 | серый | комбинированный, климат-контроль, Машина... |
| 4 | Toyota Camry | Toyota | Camry | 2020 | 18000000 | Б/у | седан | 2.5 | бензин | КПП автомат | 39000 | белый | металлик, кожа, кондиционер, Вся родная и... |

Then we started to analyze our dataset.

```
df.describe
```

```
<bound method NDFrame.describe of                    Title    Brand               Title   Brand
0      ВАЗ (Lada) Lada 2121      ВАЗ   (Lada) Lada 2121  2020   5950000
1               Genesis G70  Genesis                G70  2021  23500000
2          Hyundai Santa Fe  Hyundai           Santa Fe  2021  14790152
3            Ravon Nexia R3    Ravon           Nexia R3  2020   5200000
4              Toyota Camry   Toyota              Camry  2020  18000000
..                      ...      ...                ...   ...       ...
947  ВАЗ (Lada) Granta 2190      ВАЗ  (Lada) Granta 2190  2013   2870000
948            Kia Sportage      Kia           Sportage  2023  18000000
949        Hyundai Santa Fe  Hyundai           Santa Fe  2023  19700000
950             Kia Sorento      Kia            Sorento  2021  19000000
951        ВАЗ (Lada) 2112       ВАЗ        (Lada) 2112  2000    650000

    Condition         Type  Engine_Size Fuel_Type Transmission  Mileage
0         Б/у  внедорожник          1.7    бензин  КПП механика    29000
1         Б/у        седан          3.3    бензин   КПП автомат    16500
2         Б/у    кроссовер          2.5    бензин   КПП автомат   107000
3         Б/у        седан          1.5    бензин   КПП автомат   118000
4         Б/у        седан          2.5    бензин   КПП автомат    39000
..        ...          ...          ...       ...           ...      ...
947       Б/у        седан          1.6    бензин  КПП механика   157000
948       Б/у    кроссовер          2.5    бензин   КПП автомат     8000
949       Б/у    кроссовер          2.5    бензин   КПП автомат    20000
950       Б/у    кроссовер          2.5    бензин   КПП автомат    40000
951       Б/у      хэтчбек          1.6    бензин  КПП механика   222222
```

```
df.dtypes
```

```
Title            object
Brand            object
Model            object
Year              int64
Price             int64
Condition        object
Type             object
Engine_Size     float64
Fuel_Type        object
Transmission     object
Mileage           int64
Color            object
Additional       object
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 952 entries, 0 to 951
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Title         952 non-null    object
 1   Brand         952 non-null    object
 2   Model         952 non-null    object
 3   Year          952 non-null    int64
 4   Price         952 non-null    int64
 5   Condition     952 non-null    object
 6   Type          952 non-null    object
 7   Engine_Size   952 non-null    float64
 8   Fuel_Type     952 non-null    object
 9   Transmission  952 non-null    object
 10  Mileage       952 non-null    int64
 11  Color         952 non-null    object
 12  Additional    952 non-null    object
dtypes: float64(1), int64(3), object(9)
memory usage: 96.8+ KB
```

```
df.isna().sum()
```

```
Title           0
Brand           0
Model           0
Year            0
Price           0
Condition       0
Type            0
Engine_Size     0
Fuel_Type       0
Transmission    0
Mileage         0
Color           0
Additional      0
dtype: int64
```

As you can see here, we used some functions that will give us the descriptive analysis of our dataset.

```
df_price_mean = df['Price'].mean()
df_engine_mean = df['Engine_Size'].mean()
df_mileage_mean = df['Mileage'].mean()

print('Price mean: ', df_price_mean)
print('Engine size mean: ', df_engine_mean)
print('Mileage mean: ', df_mileage_mean)
```

```
Price mean:  10518212.87184874
Engine size mean:  2.521323529411765
Mileage mean:  156474.58088235295
```

I used the mean() function to identify average values for integer-type columns.

```
df_price_max = df['Price'].max()
df_engine_max = df['Engine_Size'].max()
df_mileage_max = df['Mileage'].max()

df_price_min = df['Price'].min()
df_engine_min = df['Engine_Size'].min()
df_mileage_min = df['Mileage'].min()

print('Price minimum and maximum: ', df_price_min, ' ' , df_price_max)
print('Engine size minimum and maximum: ', df_engine_min, ' ' , df_engine_max)
print('Mileage minimum and maximum: ', df_mileage_min, ' ' , df_mileage_max)
```

```
Price minimum and maximum:  300000   80000000
Engine size minimum and maximum:  0.1   6.6
Mileage minimum and maximum:  1   888888
```

Also, I used max() and min() to identify minimum and maximum values on each column.

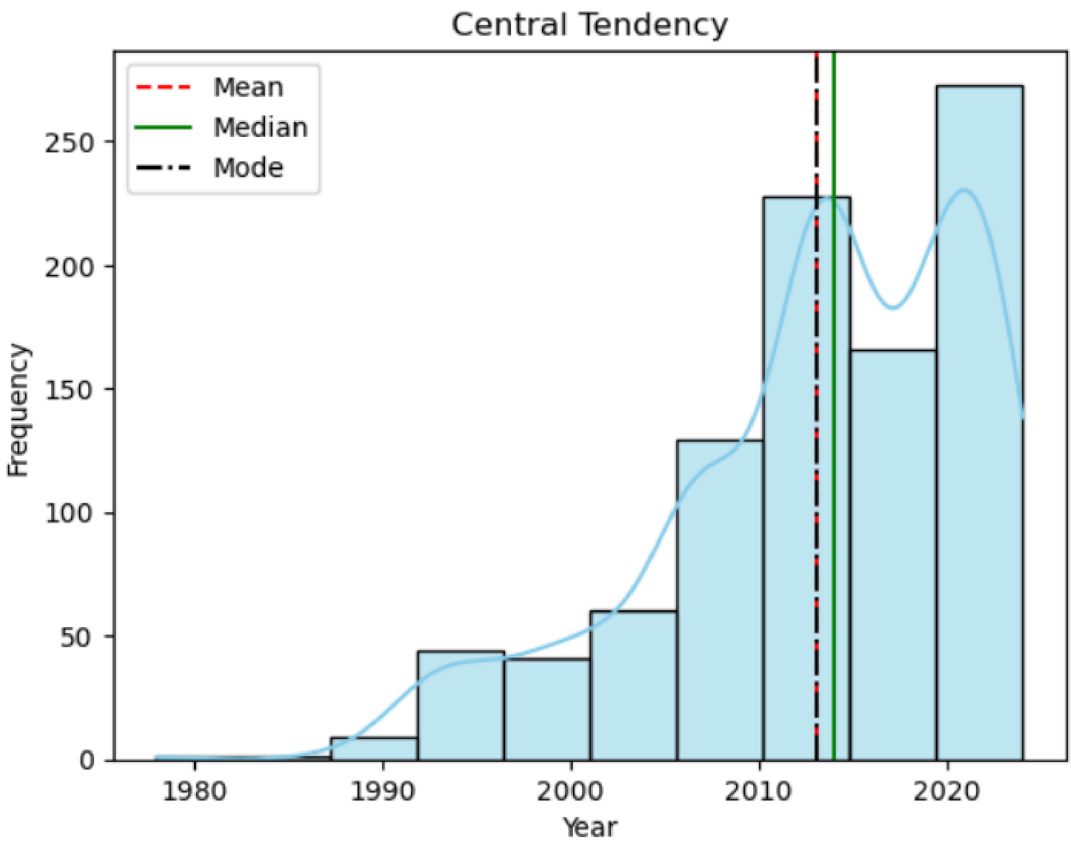
```
skewness = df['Year'].skew()
print(f"Skewness of Year: {skewness}")

kurtosis = df['Year'].kurt()
print(f"Kurtosis of Year: {kurtosis}")
```

```
Skewness of Year: -0.8853670970001123
Kurtosis of Year: 0.29032769010037374
```

Then I tried to find skewness and kurtosis for better analysis in the future. Our skewness is the negative. It means that most of our data in the right side of our normal distribution.

```
from sklearn.preprocessing import MinMaxScaler

normalizer = MinMaxScaler()

normalized_data = normalizer.fit_transform(df[['Year', 'Price', 'Engine_Size', 'Mileage']])
normalized_df = pd.DataFrame(normalized_data, columns=['Year', 'Price', 'Engine_Size', 'Mileage'])

print("Normalized Data:")
print(normalized_df)

Normalized Data:
         Year     Price  Engine_Size   Mileage
0    0.913043  0.070891     0.246154  0.032624
1    0.934783  0.291092     0.492308  0.018561
2    0.934783  0.181809     0.369231  0.120374
3    0.913043  0.061481     0.215385  0.132749
4    0.913043  0.222083     0.369231  0.043874
..        ...       ...          ...       ...
947  0.760870  0.032246     0.230769  0.176624
948  0.978261  0.222083     0.369231  0.008999
949  0.978261  0.243413     0.369231  0.022499
950  0.934783  0.234630     0.369231  0.044999
951  0.478261  0.004391     0.230769  0.249999
```

Then we used MinMaxScaler() to normalize our data for use in the future.



This is our central tendency. From this figure, we can observe that our mean and mode are in one position.

```
grouped_stats = df.groupby('Year').agg({'Price': ['mean', 'sum', 'std']})
grouped_stats
```

We used this aggregation function to find the most popular year of cars, which are on the website.

```
grouped_stats_name = df.groupby('Title').agg({'Price': ['mean', 'sum']})
sorted_df = grouped_stats_name.sort_values(by=('Price', 'sum'), ascending=False)
sorted_df.head()
```

Same thing but with brands.

```
unsorted = df.groupby('Title')['Mileage'].max().reset_index()
sorted = unsorted.sort_values(by='Mileage', ascending=True)
sorted
```

With this function, we can find cars that can pass a lot of kilometers. It will help If you are looking for a car.

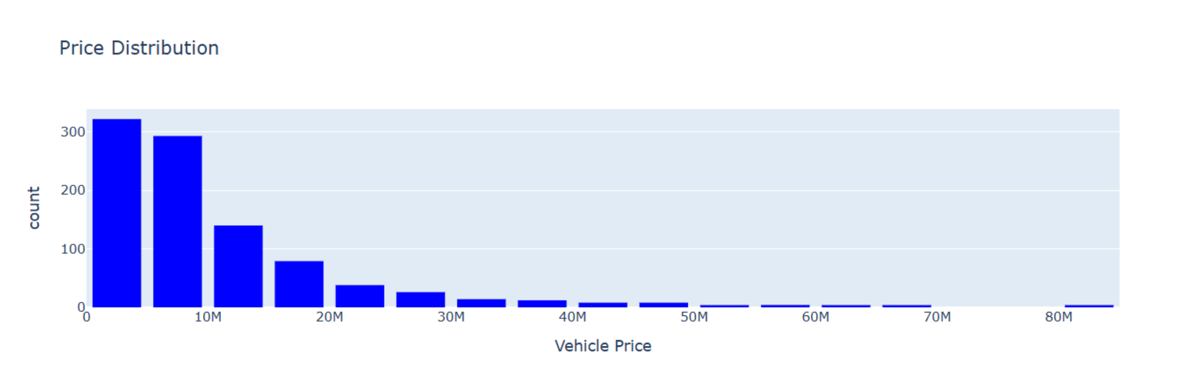
```
mileage_range_type = df.groupby('Type')['Mileage'].agg(['min', 'max']).reset_index()
mileage_range_type
```

Here we can identify the best type of vehicle, for passing a lot of kilometers. It is a necessary part of cars.

```
average_price_by_brand = df.groupby('Brand')['Price'].mean()
sorted = average_price_by_brand.sort_values( ascending=False)
sorted
```

This is the distribution that shows the price differences between the brands.

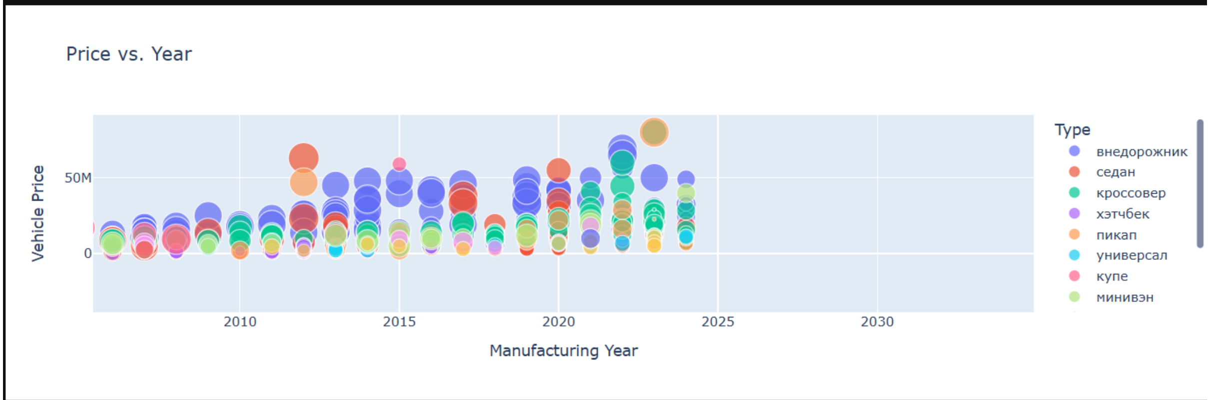
```python
import plotly.express as px

fig = px.histogram(df, x='Price', nbins=30, title='Price Distribution',
                   labels={'Price': 'Vehicle Price'},
                   color_discrete_sequence=['blue'])
fig.update_layout(bargap=0.2)
fig.show()
```



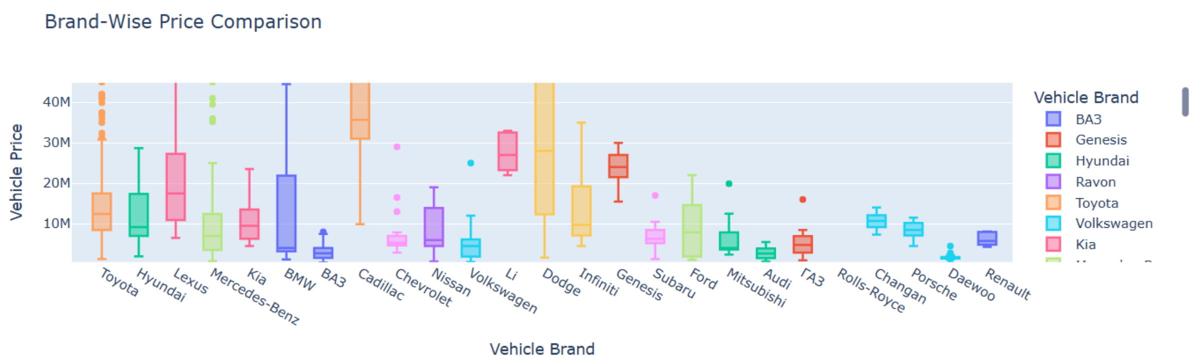

We used the “Plotly” library for our visualizations. Because It is the interactive library of Python.

Here you can see the price distribution for all cars.

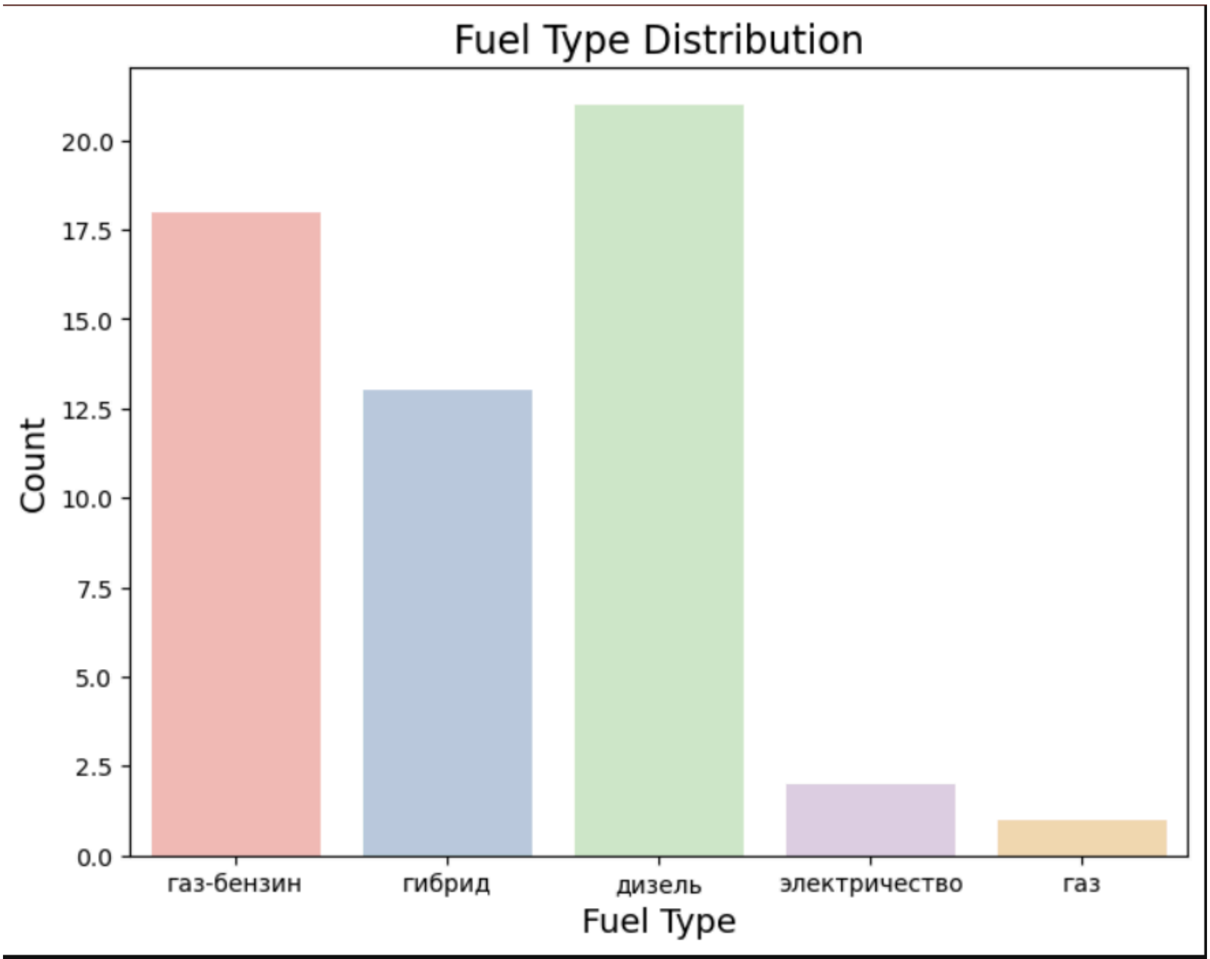
```python
fig = px.scatter(df, x='Year', y='Price', title='Price vs. Year',
                 labels={'Year': 'Manufacturing Year', 'Price': 'Vehicle Price'},
                 color='Type',
                 size='Engine_Size',
                 hover_data=['Brand', 'Model'])
fig.update_traces(marker=dict(opacity=0.7))
fig.show()
```



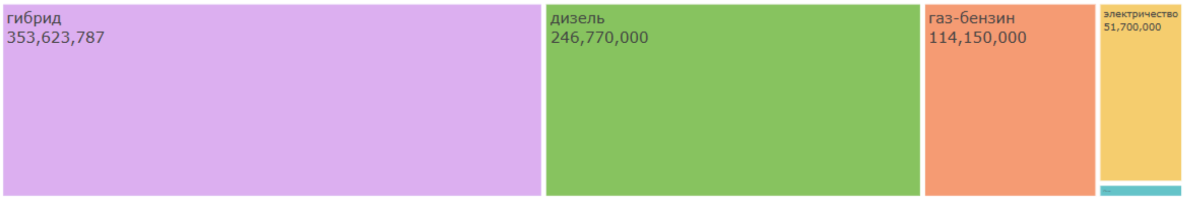

This is the price, year, and type of the car on the scatter plot.




Here you can see price distribution by brands. But you also can see the price differences between some models of the types.




Here you can find the most popular fuel types. (without oil)




This is the price for the any type of the fuel.

# Data Aggregation and Grouping Operations

My part (Daniyar) here is started with Data Aggregation and Grouping, so I will use such functions as: **groupby(), pivot(), unstacked()**.

**groupby()**: This is used to group data by one or more columns, then we can apply aggregation functions like **sum, mean, count,** etc., to these groups.

**pivot()**: it is to create a pivot table by reshaping the data.

**unstacked()**: This is used to "unstack" (reshape) the data, turning the hierarchical index levels into columns.

```
Ввод [30]: group_type = df1.groupby('Type')['Price'].sum().sort_values(ascending=False)

Ввод [31]: group_type

Out[31]: Type
         седан           3488840339
         внедорожник     3005122999
         кроссовер       2294975317
         пикап            302600000
         минивэн          246800000
         универсал        210540000
         хэтчбек          163100000
         купе             125500000
         лифтбек          104490000
         фургон            50869999
         микроавтобус      10000000
         родстер            7900000
         кабриолет          2600000
         Name: Price, dtype: int64

Ввод [32]: group_car = df1.groupby('Brand')['Price'].sum().sort_values(ascending=False)

Ввод [33]: group_car

Out[33]: Brand
         Toyota           2901515072
         Hyundai          1353540703
         Lexus             909950000
         Mercedes-Benz     673020000
         Kia               658060000
         BMW               422257000
         ВАЗ               348429000
         Cadillac          315300000
         Chevrolet         302389999
         Nissan            296210555
         Volkswagen        214825000
         Li                193400000
         Dodge             137650000
         Infiniti          129147328
         Genesis           119000000
         Subaru             84200000
         Ford               78000000
         Mitsubishi         77000000
         Audi               70830000
         ГАЗ                69995000
         Rolls-Royce        63000000
         Changan            53349999
```

For the first, I used the **groupby()** function to find prices by the type and brand, so I can easily find the needed price.

Ввод [34]:
```
grouped = df1.groupby('Color').agg(Total_Price=('Price','sum'), Avg_Price=('Price','mean')).sort_values(by='Total_Price',ascendir
```

Ввод [35]:
```
grouped
```

Out[35]:

| Color | Total_Price | Avg_Price |
|---|---|---|
| черный | 3455206884 | 1.577720e+07 |
| белый | 3117709490 | 1.101664e+07 |
| серый | 917021282 | 8.817512e+06 |
| серебристый | 728639999 | 6.684771e+06 |
| синий | 330969999 | 6.364808e+06 |
| металлик | 276150000 | 1.022778e+07 |
| коричневый | 158100000 | 7.528571e+06 |
| бордовый | 113100000 | 7.540000e+06 |
| жёлтый | 103730000 | 1.728833e+07 |
| зеленый | 100462000 | 3.863923e+06 |
| литые диски | 97000000 | 1.616667e+07 |
| красный | 82680000 | 6.360000e+06 |
| хамелеон | 64500000 | 1.290000e+07 |
| велюр | 51100000 | 6.387500e+06 |
| оранжевый | 48100000 | 1.603333e+07 |
| бежевый | 39450000 | 5.635714e+06 |
| вишня | 39150000 | 5.592857e+06 |
| голубой | 33400000 | 5.566667e+06 |
| Весь в родном окрасе | 23499000 | 2.349900e+07 |

same thing for here, I used the **groupby()** function to group the data by car colors and applied the **agg()** function to calculate the total and average price for each color.

```
Ввод [36]: pivot = df1.pivot_table(values='Price',index='Brand',columns='Color',aggfunc='sum', fill_value=0)
```

```
Ввод [37]: pivot
```

Out[37]:

| Color | 001261 Автоцентр — это простота | 080440 Автоцентр — это простота | 129015 Автоцентр — это простота | 184405 Автоцентр — это простота | 3 | Богатая комплектация Flagship | В наличии! Новая машина | Весь в родном окрасе | Двигатель 406 | Дилерский центр «Chery Astana» | ... | металлик | оранжевый |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Brand** | | | | | | | | | | | | | |
| **Aston** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Audi** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 6800000 | ( |
| **BMW** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 65400000 | ( |
| **BYD** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Cadillac** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Changan** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Chery** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7900000 | ... | 0 | ( |
| **Chevrolet** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2890000 | ... | 0 | ( |
| **Chrysler** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Daewoo** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1550000 | ( |
| **Dodge** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **DongFeng** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **EXEED** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **FAW** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Ford** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Geely** | 0 | 0 | 0 | 0 | 0 | 9700000 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Genesis** | 0 | 0 | 0 | 0 | 15500000 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Haval** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Honda** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Hyundai** | 0 | 7900000 | 11990000 | 7990000 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 70000000 | ( |

The resulting **pivot table** will display the total prices (sum of the "Price" column) for each combination of car brand (rows) and car color (columns). For example, for the brand "**Audi**" and color "**металлик**" the total price will be the sum of all prices for **Audi** cars with that color. If no cars of a certain brand and color combination exist, the value will be 0.

```
pivot_mean = df1.pivot_table(values='Price',index='Brand',columns='Color',aggfunc=['mean','count'],fill_value=0)
```

```
pivot_mean
```

| | mean | | | | | | | | | | ... | count | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Color** | 001261 Автоцентр — это простота | 080440 Автоцентр — это простота | 129015 Автоцентр — это простота | 184405 Автоцентр — это простота | 3 | Богатая комплектация Flagship | В наличии! Новая машина | Весь в родном окрасе | Двигатель 406 | Дилерский центр «Chery Astana» | ... | металлик | оранжевый |
| **Brand** | | | | | | | | | | | | | |
| **Aston** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Audi** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 2 | ( |
| **BMW** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 3 | ( |
| **BYD** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Cadillac** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Changan** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Chery** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7900000 | ... | 0 | ( |
| **Chevrolet** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2890000 | ... | 0 | ( |
| **Chrysler** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Daewoo** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1 | ( |
| **Dodge** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **DongFeng** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **EXEED** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **FAW** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |

Same, but I used **aggfunc** to calculate the total, average price, and count for each color in the pivot table

```
filtered_engine = df1[df1['Engine_Size'] > 2.5]
```

```
filtered_engine
```

| | Title | Brand | Model | Year | Price | Condition | Type | Engine_Size | Fuel_Type | Transmission | Mileage | Color | Additional |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Genesis G70 | Genesis | G70 | 2021 | 23500000 | Б/у | седан | 3.3 | бензин | КПП автомат | 16500 | белый | металлик |
| 7 | Toyota Highlander | Toyota | Highlander | 2017 | 19500000 | Б/у | кроссовер | 3.5 | бензин | КПП автомат | 117000 | черный | металлик, литые диски, тонировка, л... |
| 13 | Mercedes-Benz E 350 | Mercedes-Benz | E 350 | 2013 | 12500000 | Б/у | седан | 3.5 | бензин | КПП автомат | 110000 | литые диски | тонировка, панорамная крыша, хруст... |
| 16 | Toyota Camry | Toyota | Camry | 2007 | 4000000 | Б/у | седан | 3.5 | бензин | КПП автомат | 413035 | золотистый | литые диски, тонировка, кожа, аудио... |
| 20 | Infiniti QX80 | Infiniti | QX80 | 2014 | 19000000 | Б/у | внедорожник | 5.6 | бензин | КПП автомат | 191000 | бордовый | металлик, тонировка, люк, ветро... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 933 | Lexus LX 570 | Lexus | LX 570 | 2017 | 46000000 | Б/у | внедорожник | 5.7 | бензин | КПП автомат | 70000 | черный | металлик, Машина в идеальном состо... |
| 934 | Porsche Cayenne | Porsche | Cayenne | 2006 | 8000000 | Б/у | кроссовер | 4.5 | бензин | КПП автомат | 155000 | красный | металлик, литые диски, тонировка,... |
| 938 | Lexus LX 570 | Lexus | LX 570 | 2016 | 40000000 | Б/у | внедорожник | 5.7 | бензин | КПП автомат | 270000 | серебристый | металлик, тонировка, люк, сп... |
| 940 | Jeep Grand Cherokee | Jeep | Grand Cherokee | 2014 | 14250000 | Б/у | внедорожник | 3.6 | бензин | КПП автомат | 164000 | черный | металлик, литые диски, тонировка,... |
| 942 | Mercedes-Benz C 240 | Mercedes-Benz | C 240 | 2000 | 3300000 | Б/у | седан | 2.6 | бензин | КПП автомат | 277000 | металлик | тонировка, ксенон, велюр, аудиосистем... |

Here I simply just filtered the data by creating a boolean condition where only rows with an "**Engine_Size**" greater than 2.5 are selected.

```
]: grouped_c = df.groupby(['Brand', 'Color'])['Price'].sum()
```

```
]: grouped_c
```

```
]: Brand  Color
   Aston  голубой       17000000
   Audi   вишня          3000000
          голубой        3600000
          зеленый        5850000
          коричневый     3900000
                          ...
   УАЗ    белый          9200000
          жёлтый         4750000
          зеленый        3000000
          серебристый    2300000
          серый          6599000
   Name: Price, Length: 269, dtype: int64
```

```
]: unstacked = grouped_c.unstack(fill_value=0)
```

Here I want to calculate the total price for each combination of car brand and color, also using **groupby()** function.

```
Ввод [49]: unstacked = grouped_c.unstack(fill_value=0)
```

```
Ввод [50]: unstacked
```

Out[50]:

| Color<br>Brand | 001261 Автоцентр — это простота | 080440 Автоцентр — это простота | 129015 Автоцентр — это простота | 184405 Автоцентр — это простота | 3 | Богатая комплектация Flagship | В наличии! Новая машина | Весь в родном окрасе | Двигатель 406 | Дилерский центр «Chery Astana» | ... | металлик | оранжевый |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Aston** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Audi** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 6800000 | ( |
| **BMW** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 65400000 | ( |
| **BYD** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Cadillac** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Changan** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Chery** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7900000 | ... | 0 | ( |
| **Chevrolet** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2890000 | ... | 0 | ( |
| **Chrysler** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Daewoo** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1550000 | ( |
| **Dodge** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **DongFeng** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **EXEED** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **FAW** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Ford** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Geely** | 0 | 0 | 0 | 0 | 0 | 9700000 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Genesis** | 0 | 0 | 0 | 0 | 15500000 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Haval** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Honda** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Hyundai** | 0 | 7900000 | 11990000 | 7990000 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 70000000 | ( |
| **Infiniti** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **JAC** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Jaguar** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |
| **Jeep** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | ( |

Then, we are coming to the **unstacked()** function, to view the aggregated data in a tabular format with car brands and colors. For example, if no "**Audi**" cars are available in the color "red," the corresponding value will be 0.

The result, as you can see, is a **DataFrame** with car brands as rows, car colors as columns, and the total price for each brand-color combination.

## Data Analysis using other libraries such Patsy, Statsmodel and Scikit-learn

```
Ввод [60]: from patsy import dmatrices
```

Now, we are going to deal with the libraries that are also important for Data Analytics in Python.

1. **Patsy**: Create a formula or design matrices.
2. **Statsmodels**: Fit a statistical model to the data (e.g., linear regression, ANOVA).
3. **Scikit-learn**: Fit a machine learning model, evaluate it, and fine-tune hyperparameters.

```
Ввод [61]: y, X = dmatrices('Price ~ Year + Engine_Size + C(Type)', data=df1)

Ввод [62]: y

Out[62]: DesignMatrix with shape (952, 1)
              Price
            5950000
           23500000
           14790152
            5200000
           18000000
            2550000
            3950000
           19500000
            9100000
            8700000
            2500000
            5350000
            7000000
           12500000
            4800000
            5200000
            4000000

Ввод [63]: X

Out[63]: DesignMatrix with shape (952, 15)
           Columns:
             ['Intercept',
              'C(Type)[T.кабриолет]',
              'C(Type)[T.кроссовер]',
              'C(Type)[T.купе]',
              'C(Type)[T.лифтбек]',
              'C(Type)[T.микроавтобус]',
              'C(Type)[T.минивэн]',
              'C(Type)[T.пикап]',
              'C(Type)[T.родстер]',
              'C(Type)[T.седан]',
              'C(Type)[T.универсал]',
```

It shows us the coefficients for **Year, Engine_Size**, and the dummy variables for **Type**. And we also get statistical details like p-values, R-squared, and other model diagnostics.This process helps to prepare data for statistical modeling. It automatically handles categorical variables, creates the necessary design matrices

Here we are attempting to perform a least-squares regression using the **Patsy** formula and **NumPy** for solving the linear system

**Coefficients (coef)**: Tell us how much each factor (Year, Engine_Size, Type e.g.,) influences the price.

**Residuals (resid)**: Tell us how far off the model's predictions are from the actual prices.

```
import statsmodels.api as sm
```

Here we are coming to statsmodel library

```
X2 = df1[['Year', 'Engine_Size']]
y2 = df1['Price']
```

```
X2 = sm.add_constant(X2)
```

```
model = sm.OLS(y2, X2)
```

```
results = model.fit()
```

```
results.summary()
```

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | Price | R-squared: | 0.595 |
| Model: | OLS | Adj. R-squared: | 0.594 |
| Method: | Least Squares | F-statistic: | 696.8 |
| Date: | Tue, 24 Dec 2024 | Prob (F-statistic): | 6.27e-187 |
| Time: | 19:21:31 | Log-Likelihood: | -16322. |
| No. Observations: | 952 | AIC: | 3.265e+04 |
| Df Residuals: | 949 | BIC: | 3.266e+04 |
| Df Model: | 2 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | -1.254e+09 | 5.32e+07 | -23.581 | 0.000 | -1.36e+09 | -1.15e+09 |
| Year | 6.207e+05 | 2.64e+04 | 23.525 | 0.000 | 5.69e+05 | 6.72e+05 |
| Engine_Size | 5.762e+06 | 1.86e+05 | 30.938 | 0.000 | 5.4e+06 | 6.13e+06 |

| | | | |
|---|---|---|---|
| Omnibus: | 595.849 | Durbin-Watson: | 2.054 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 8703.824 |
| Skew: | 2.616 | Prob(JB): | 0.00 |
| Kurtosis: | 16.858 | Cond. No. | 4.88e+05 |

We are using **Ordinary Least Squares (OLS) regression** using **statsmodels** to model the relationship between car prices (Price) and two independent variables:(Year and Engine_Size)

**Here are the explanation of results of OLS Regression Results Summary:**

- **R-squared**: 0.595 (explains 59.5% of price variation).
- **Coefficients**:

    **Intercept**: -1.254e+09 (not meaningful).

    **Year**: 620,700 increase per year.

    **Engine_Size**: 5.76 million increase per unit.

- **Significance**: Both Year and Engine_Size are statistically significant with very low p-values.
- **Diagnostics**: Residuals may not be normally distributed (Jarque-Bera and Omnibus tests), but no autocorrelation (Durbin-Watson = 2.054).

The model explains a good portion of price variation.

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 4.88e+05. This might indicate that there are

strong multicollinearity or other numerical problems.

Ввод [79]:
```python
df1['Price_above_10M'] = (df1['Price'] > 10000000).astype(int)
```

Ввод [80]:
```python
X3 = df1[['Year', 'Engine_Size']]
y3 = df1['Price_above_10M']
```

Ввод [81]:
```python
X3 = sm.add_constant(X3)
```

Ввод [82]:
```python
model = sm.Logit(y3, X3)
results = model.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.305531
         Iterations 9
```

Ввод [83]:
```python
results.summary()
```

Out[83]:

Logit Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Price_above_10M | **No. Observations:** | 952 |
| **Model:** | Logit | **Df Residuals:** | 949 |
| **Method:** | MLE | **Df Model:** | 2 |
| **Date:** | Tue, 24 Dec 2024 | **Pseudo R-squ.:** | 0.5261 |
| **Time:** | 19:21:31 | **Log-Likelihood:** | -290.87 |
| **converged:** | True | **LL-Null:** | -613.73 |
| **Covariance Type:** | nonrobust | **LLR p-value:** | 6.058e-141 |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -727.6010 | 56.947 | -12.777 | 0.000 | -839.214 | -615.988 |
| **Year** | 0.3579 | 0.028 | 12.745 | 0.000 | 0.303 | 0.413 |
| **Engine_Size** | 2.1872 | 0.163 | 13.381 | 0.000 | 1.867 | 2.508 |

The model shows that both Year and Engine_Size significantly impact the likelihood of the car price being above 10 million. The **Pseudo R-squared** value of 0.5261 suggests a moderate fit. Both variables are highly significant predictors.

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
```

**Scikit-learn is one of the most popular libraries for machine learning in Python. It provides simple and efficient tools for data mining and data analysis.**

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```python
X4 = df1[['Year', 'Engine_Size']]
y4 = df1['Price']
```

```python
X_train, X_test, y_train, y_test = train_test_split(X4, y4, test_size=0.2, random_state=42)


model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Mean Squared Error: 38464954738395.836
R-squared: 0.6375018976123719
```

**Linear Regression Results Summary:**

**Mean Squared Error (MSE)**: 38.46 trillion
Measures the average squared difference between actual and predicted prices. The high value suggests large prediction errors, likely due to the wide range of prices in the dataset.

**R-squared ($R^2$)**: 0.638
Indicates that **63.8%** of the variance in car prices is explained by Year and Engine_Size.

**Conclusion:** The model provides a moderate fit, but the high **MSE** suggests further improvements are needed. Possible enhancements include scaling the data, adding more features, or using non-linear regression models.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

```python
df1['Price_above_10M'] = (df1['Price'] > 10000000).astype(int)

X5 = df1[['Year', 'Engine_Size']]
y5 = df1['Price_above_10M']

X_train, X_test, y_train, y_test = train_test_split(X5, y5, test_size=0.2, random_state=42)

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

```
Accuracy: 0.7958115183246073
Confusion Matrix:
[[119   8]
 [ 31  33]]
```

1. **Accuracy (79.6%)** suggests reasonably good classification performance.
2. **Confusion Matrix** shows:

119 and 33 are correct predictions for each class.

8 and 31 are the misclassifications, where the model predicted incorrectly.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
```

```
df1 = load_iris()
X6 = df1.data
y6 = df1.target

X_train, X_test, y_train, y_test = train_test_split(X6, y6, test_size=0.3, random_state=42)

rf = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

grid_search.fit(X_train, y_train)

print("Best hyperparameters found: ", grid_search.best_params_)

best_rf = grid_search.best_estimator_
test_accuracy = best_rf.score(X_test, y_test)
print(f"Test set accuracy: {test_accuracy:.4f}")
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best hyperparameters found:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Test set accuracy: 1.0000
```

This code uses **GridSearchCV** to find the best ***hyperparameters*** for a Random Forest classifier on the Iris dataset. It splits the data into training and test sets (70/30), tests 108 parameter combinations with 5-fold cross-validation, and identifies the optimal settings: 100 estimators, no maximum depth, minimum split of 2, and minimum leaf of 1. The final model achieves 100% accuracy on the test set, indicating a perfect fit, which may suggest the dataset is simple or the model is overfitting.