EAST WEST UNIVERSITY

Assignment 02 (REPORT)

Course Code: CSE303
Course Title: Statistics for Data Science
Section: 07

Submitted by:

Mohammad Ashiquzzaman Hadi
ID: 2021-3-60-143
Dept. of CSE
Date of submission: 30 December 2024

Submitted to:

Puja Chakraborty
Department of Computer Science and Engineering
East West University

Project Name: Cars for Sale

What are the types of data? (e.g. Qualitative vs Quantitative; Nominal vs Ordinal; Discrete vs Continuous)

Data can be classified in various ways based on its nature and how it is measured or represented. Below are the key types of data, with explanations and examples:

1. Qualitative vs Quantitative Data

- **Qualitative (Categorical) Data**: Data that describes qualities or characteristics and is not numerical.
  - **Example**:
    - Colors (e.g., red, blue, green)
    - Types of animals (e.g., dog, cat, bird)
    - Gender (e.g., male, female)
- **Quantitative (Numerical) Data**: Data that represents numerical values and can be measured or counted.
  - **Example**:
    - Age (e.g., 25 years)
    - Height (e.g., 170 cm)
    - Temperature (e.g., 22°C)

2. Nominal vs Ordinal Data

- **Nominal Data**: A type of qualitative data that has no intrinsic order or ranking.
  - **Example**:
    - Eye color (e.g., blue, brown, green)
    - Nationality (e.g., American, French, Indian)
    - Type of cuisine (e.g., Italian, Chinese, Indian)
- **Ordinal Data**: A type of qualitative data where categories have a meaningful order, but the intervals between them are not necessarily consistent.
  - **Example**:
    - Education level (e.g., High School, Bachelor’s, Master’s, PhD)
    - Satisfaction rating (e.g., Poor, Fair, Good, Excellent)
    - Rank (e.g., 1st place, 2nd place, 3rd place)

3. Discrete vs Continuous Data

- **Discrete Data**: A type of quantitative data that can only take specific, distinct values (often integers), and there are no intermediate values.
  - **Example**:
    - Number of children in a family (e.g., 2, 3, 4)
    - Number of cars in a parking lot (e.g., 10, 15, 20)
    - Number of students in a classroom
- **Continuous Data**: A type of quantitative data that can take any value within a given range, including fractions or decimals.
  - **Example**:

        - Height (e.g., 170.5 cm, 180.3 cm)
        - Weight (e.g., 65.4 kg)
        - Time (e.g., 3.7 seconds, 12.5 hours)

4. Interval vs Ratio Data

- **Interval Data**: A type of quantitative data where the difference between values is meaningful, but there is no true zero point (i.e., the zero doesn't mean 'none').
    - **Example**:
        - Temperature in Celsius or Fahrenheit (e.g., 20°C, 30°C; the difference is meaningful, but 0°C doesn't mean "no temperature")
        - Dates (e.g., 1990, 2000, 2020)
- **Ratio Data**: A type of quantitative data with a true zero point, meaning zero signifies the absence of the quantity.
    - **Example**:
        - Weight (e.g., 0 kg means no weight)
        - Height (e.g., 0 cm means no height)
        - Income (e.g., $0 means no income)

5. Binary Data

- **Binary Data**: Data that can only take one of two possible values, typically represented as "0" or "1."
    - **Example**:
        - Yes/No responses (e.g., Is the sky blue? Yes/No)
        - Gender (in some contexts, e.g., Male/Female)
        - Presence of a feature (e.g., 1 for presence, 0 for absence)

6. Time-Series Data vs Cross-Sectional Data

- **Time-Series Data**: Data collected at different time points, often used for tracking trends or patterns over time.
    - **Example**:
        - Stock market prices over time (e.g., daily closing prices)
        - Annual rainfall data for a city
- **Cross-Sectional Data**: Data collected at a single point in time or over a short period, used to analyze different variables at that moment.
    - **Example**:
        - Survey data collected from people at one time (e.g., opinion polls)
        - Population demographics in a city at a specific time

Summary of Key Points:

- **Qualitative Data**: Descriptive and categorical (e.g., colors, names).
- **Quantitative Data**: Numerical, measurable (e.g., height, age).
- **Nominal Data**: Categories with no inherent order (e.g., eye color).

- **Ordinal Data**: Categories with a meaningful order (e.g., rankings).
- **Discrete Data**: Countable, distinct values (e.g., number of children).
- **Continuous Data**: Measurable with infinite values within a range (e.g., weight).
- **Interval Data**: No true zero, meaningful differences (e.g., temperature in Celsius).
- **Ratio Data**: True zero, meaningful differences and ratios (e.g., income).
- **Binary Data**: Two possible outcomes (e.g., yes/no).
- **Time-Series Data**: Data collected over time (e.g., stock prices).
- **Cross-Sectional Data**: Data collected at a single point in time (e.g., survey results).

These classifications help in selecting the right analytical tools and interpreting the data correctly based on its characteristics.

What is balanced or imbalanced dataset? How does it might affect the performance of a model?

- **Balanced Dataset**: A dataset where the classes or categories are approximately equally represented. For example, in a binary classification problem (e.g., predicting whether a customer will buy a product: yes or no), a balanced dataset would have nearly the same number of "yes" and "no" instances.

  Example:

  - 500 "Yes" instances and 500 "No" instances.
- **Imbalanced Dataset**: A dataset where one class or category significantly outnumbers the other(s). In an imbalanced dataset, one class is overrepresented, and the other class is underrepresented. For example, in a fraud detection model, there are usually many more non-fraudulent transactions than fraudulent ones.

  Example:

  - 950 "No fraud" instances and 50 "Fraud" instances (95% vs. 5%).

Effect on Model Performance

An imbalanced dataset can negatively impact the performance of machine learning models, especially if the model is not properly designed to handle it. Here's how:

1. **Bias Toward Majority Class**:
   - In an imbalanced dataset, the model tends to be biased toward predicting the majority class because it has more data to learn from. It may predict the majority class correctly most of the time, but perform poorly on the minority class.

Example:

- In fraud detection, the model might classify most transactions as "No fraud" just because this is the majority class. This leads to high accuracy but poor performance for detecting fraud.

2. **Accuracy is Misleading**:
    - Accuracy is often not a good metric in imbalanced datasets because a model can achieve high accuracy by simply predicting the majority class for most instances. However, this doesn't reflect the model's ability to correctly identify the minority class.

Example:

- If 95% of the transactions are non-fraudulent and the model always predicts "No fraud," it would have 95% accuracy, but it would fail to detect the fraudulent transactions (which is the main task).

3. **Poor Generalization**:
    - Models trained on imbalanced data may fail to generalize well to real-world scenarios, where the classes may be more balanced. They might not be able to effectively learn the patterns of the minority class, leading to poor performance on new, unseen data.

How to Handle Imbalanced Datasets:

1. **Resampling**:
    - **Oversampling the Minority Class**: Increasing the number of instances in the minority class, often by duplicating data points or generating synthetic examples (e.g., using techniques like SMOTE).
    - **Under sampling the Majority Class**: Reducing the number of instances in the majority class to balance the dataset.
2. **Adjusting Class Weights**:
    - Many machine learning algorithms (e.g., logistic regression, decision trees) allow you to assign higher weights to the minority class, making the model pay more attention to it during training.
3. **Use of Different Metrics**:
    - Instead of accuracy, use metrics like **Precision**, **Recall**, **F1-Score**, or **AUC-ROC**, which provide a more balanced view of model performance, especially for the minority class.
4. **Anomaly Detection Models**:
    - For highly imbalanced datasets (e.g., fraud detection), anomaly detection techniques may be more suitable as they are designed to identify rare events.

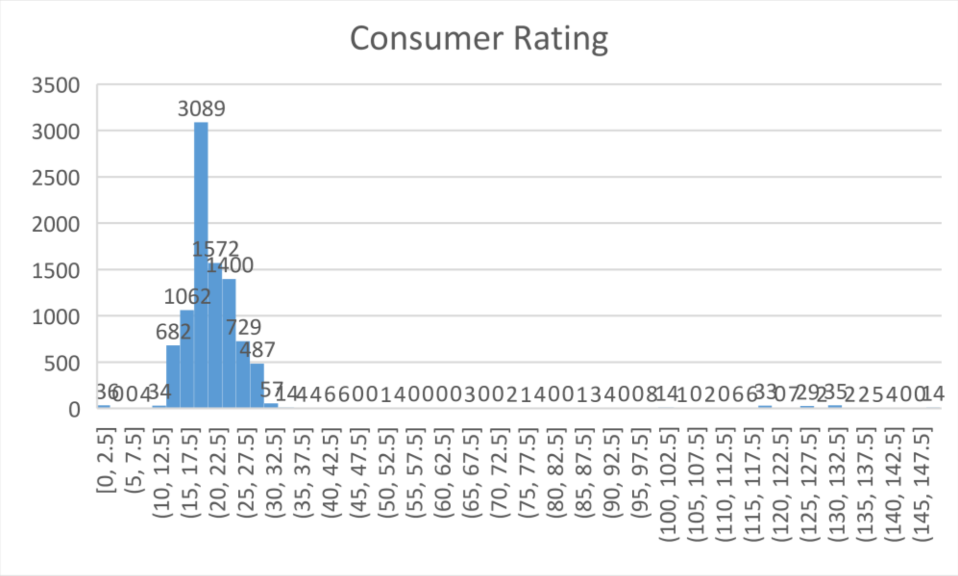
In summary, an **imbalanced dataset** can skew model performance, causing the model to be biased toward the majority class. Proper techniques, such as resampling or using specialized metrics, are essential to build a more effective and fair model in such cases.

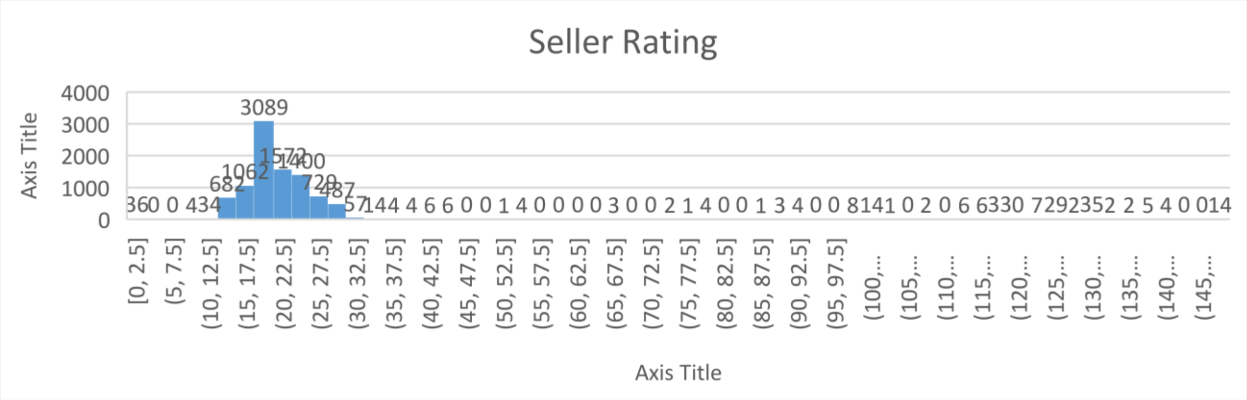
Write a short description of your task and dataset, such as what are the column in your dataset, their type, what are the range or categories of the values in each column etc.

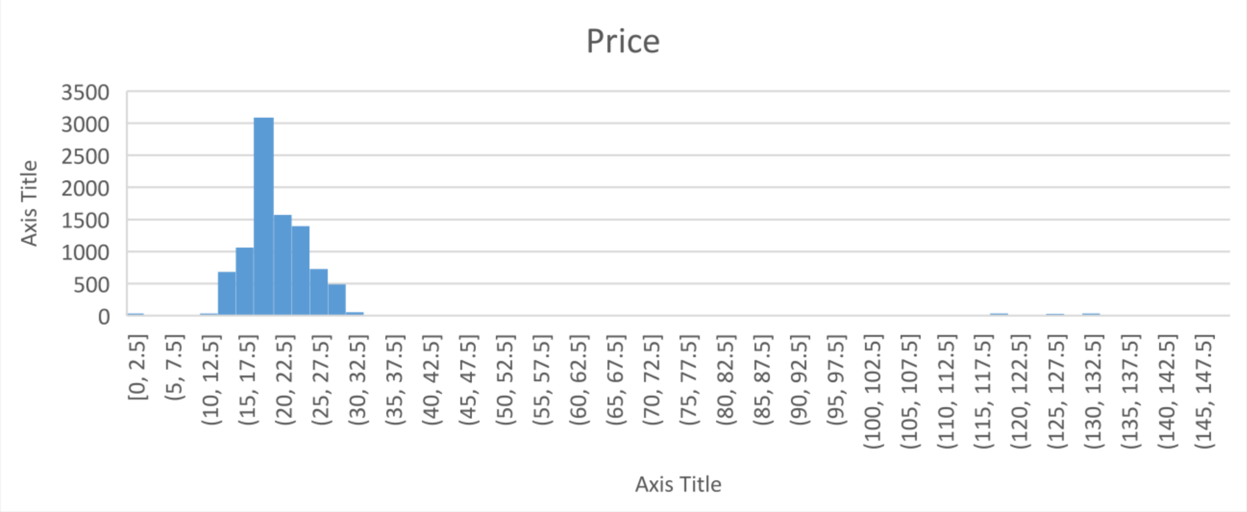
Column Info:

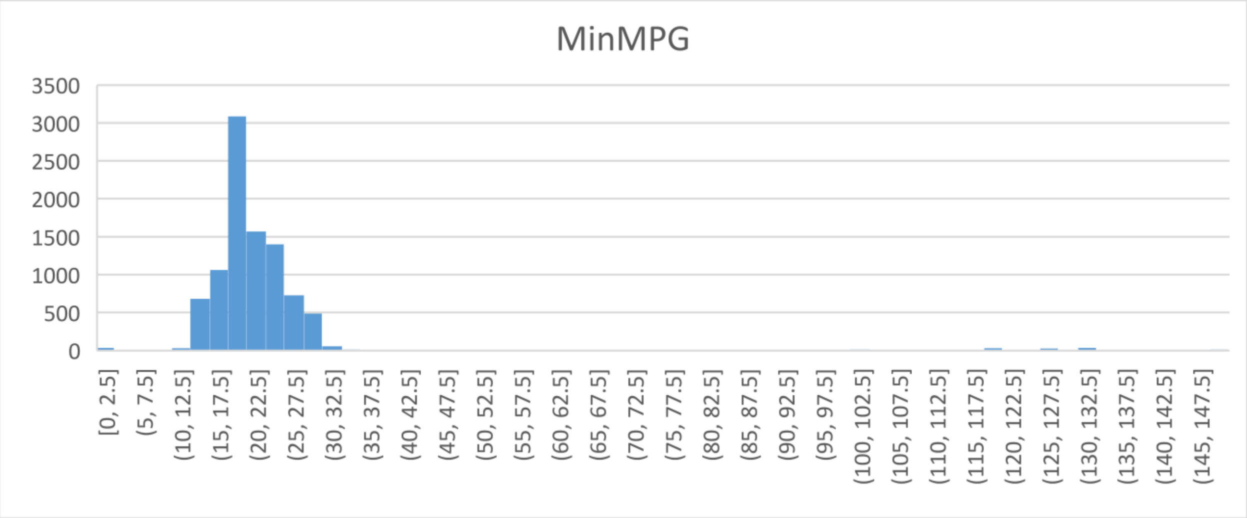
- Year: Model year
- Make: Brand of the car
- Model: Car model
- Used/New: Whether the car being sold is used, new, or certified.
- Price: Listing price
- Consumer Rating: Average consumer rating based on submitted consumer reviews
- Consumer Reviews: The number of reviews submitted for the car in the listing
- SellerType: Whether the seller is a dealer or private
- SellerName: Name of the seller
- StreetName: Name of the street where the seller is located
- State: State of the seller's location
- Zipcode: Zipcode of the seller's location
- DealType: How good is the deal based on the average market price for the car in the listing? (Great, Good, Fair, NA)
- ComfortRating: How consumers rated the car's comfort
- InteriorDesignRating: How consumers rated the car's interior design
- PerformanceRating: How consumers rated the car's performance
- ValueForMoneyRating: How consumers rated the car's value for the price
- ExteriorStylingRating: How consumers rated the car's exterior styling
- ReliabilityRating: How consumers rated the car's reliability
- ExteriorColor: The car's exterior color
- InteriorColor: The car's interior color
- Drivetrain: The drivetrain type of the car
- MinMPG: Bottom end miles per gallon
- MaxMPG: Top end miles per gallon
- FuelType: Type of fuel that the car uses. (Gas, Electric, Hybrid, etc.)
- Transmission: Type of transmission
- Engine: Name of the engine
- VIN: VIN Number
- Stock# : The listing's stock number
- Mileage: Number of miles on the car

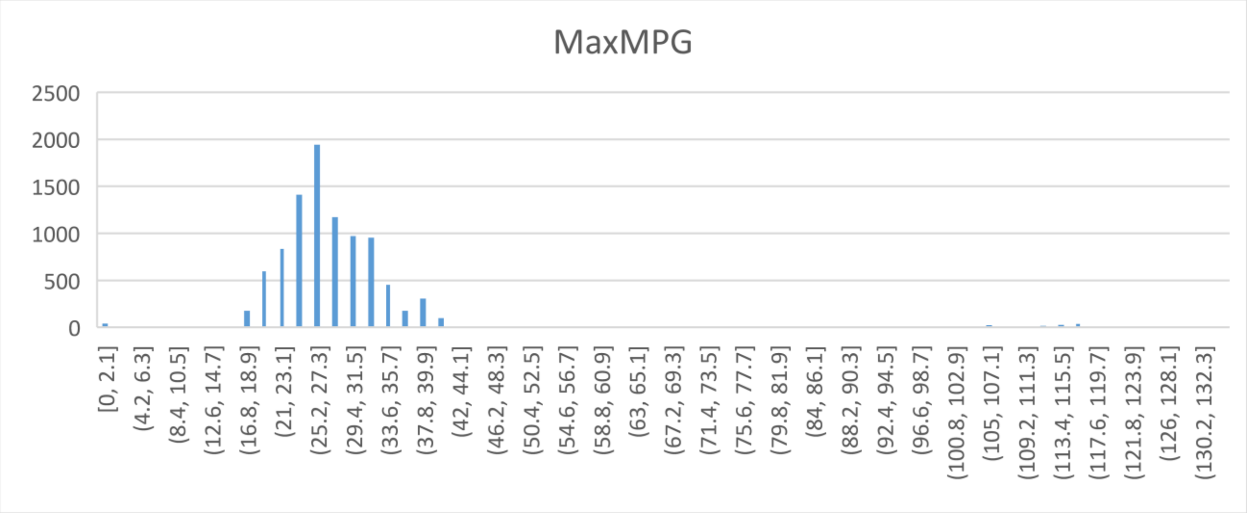
Mention if your dataset is balanced or not with quantitative result?











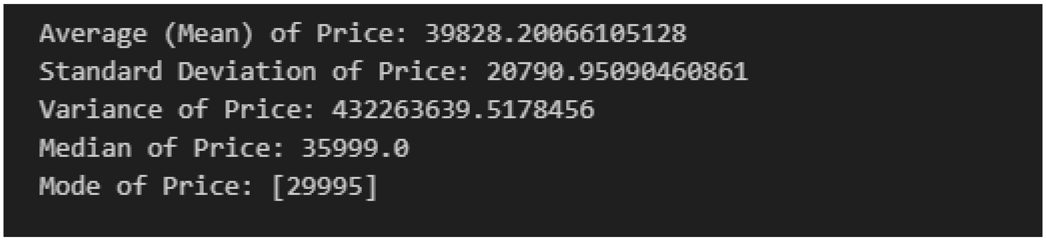
Showing the graphical representation of some data from the dataset. We cannot say that the data is perfectly balanced. There are so such equal or almost equal value data. There are deviations and mixed interpretation of data.

An **imbalanced dataset** can skew model performance, causing the model to be biased toward the majority class. Proper techniques, such as resampling or using specialized metrics, are essential to build a more effective and fair model in such cases.

So from my point of view, the following dataset is not perfectly balanced in quantitative data.

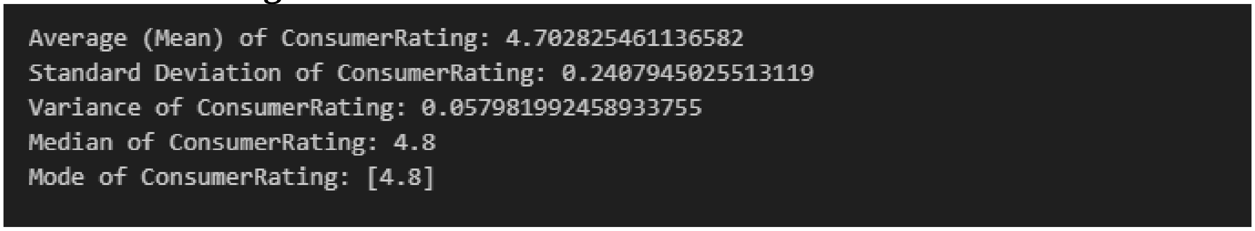
If any column contains continuous numerical values then calculate the mean, median, variance and standard deviation of that column, otherwise if the values are categorical or discrete then count the frequency and percentage of each type of values.

Price:



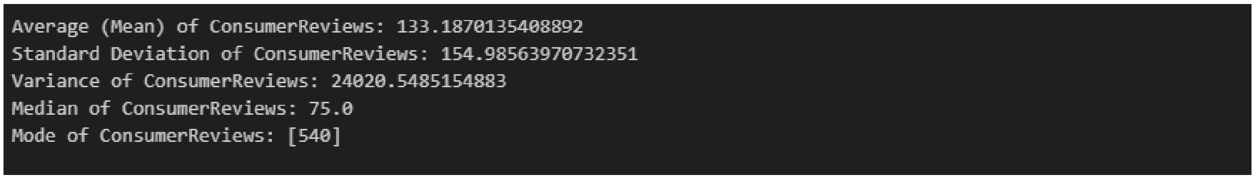
```
Average (Mean) of Price: 39828.20066105128
Standard Deviation of Price: 20790.95090460861
Variance of Price: 432263639.5178456
Median of Price: 35999.0
Mode of Price: [29995]
```

ConsumerRating:



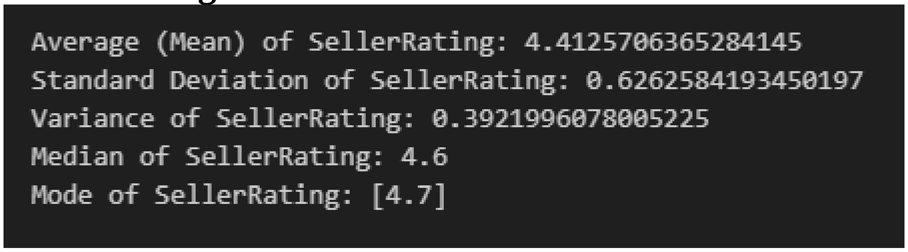
```
Average (Mean) of ConsumerRating: 4.702825461136582
Standard Deviation of ConsumerRating: 0.2407945025513119
Variance of ConsumerRating: 0.057981992458933755
Median of ConsumerRating: 4.8
Mode of ConsumerRating: [4.8]
```

Consumer Reviews:



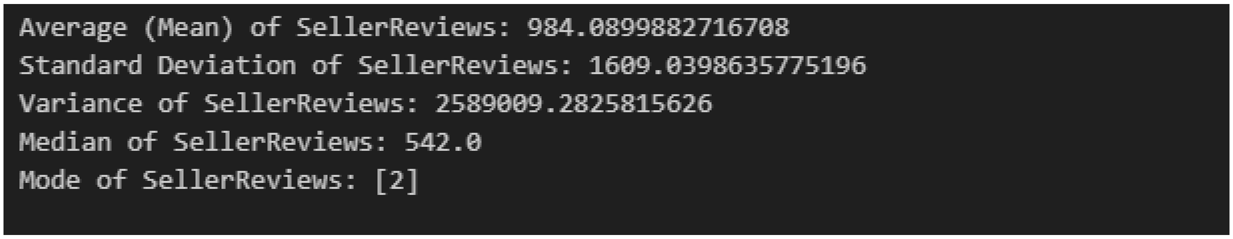
```
Average (Mean) of ConsumerReviews: 133.1870135408892
Standard Deviation of ConsumerReviews: 154.98563970732351
Variance of ConsumerReviews: 24020.5485154883
Median of ConsumerReviews: 75.0
Mode of ConsumerReviews: [540]
```

Seller Rating:



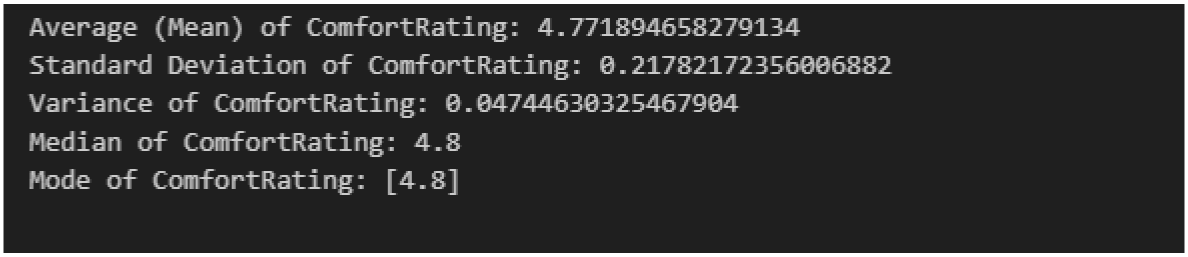
```
Average (Mean) of SellerRating: 4.4125706365284145
Standard Deviation of SellerRating: 0.6262584193450197
Variance of SellerRating: 0.3921996078005225
Median of SellerRating: 4.6
Mode of SellerRating: [4.7]
```

Seller Reviews



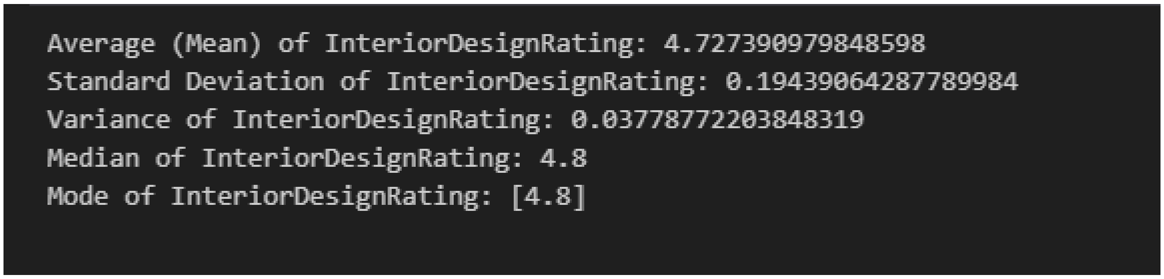
```
Average (Mean) of SellerReviews: 984.0899882716708
Standard Deviation of SellerReviews: 1609.0398635775196
Variance of SellerReviews: 2589009.2825815626
Median of SellerReviews: 542.0
Mode of SellerReviews: [2]
```

Comfort Rating:



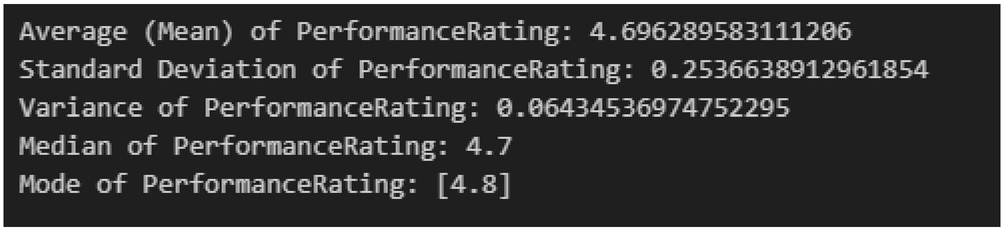
```
Average (Mean) of ComfortRating: 4.771894658279134
Standard Deviation of ComfortRating: 0.21782172356006882
Variance of ComfortRating: 0.04744630325467904
Median of ComfortRating: 4.8
Mode of ComfortRating: [4.8]
```

Interior Design Rating:



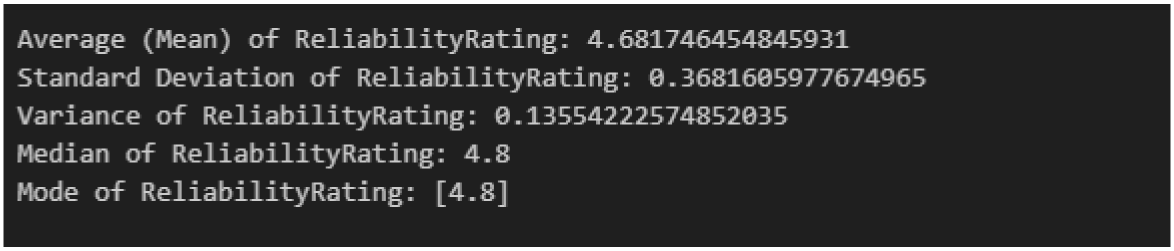
```
Average (Mean) of InteriorDesignRating: 4.727390979848598
Standard Deviation of InteriorDesignRating: 0.19439064287789984
Variance of InteriorDesignRating: 0.03778772203848319
Median of InteriorDesignRating: 4.8
Mode of InteriorDesignRating: [4.8]
```

Performance Rating:



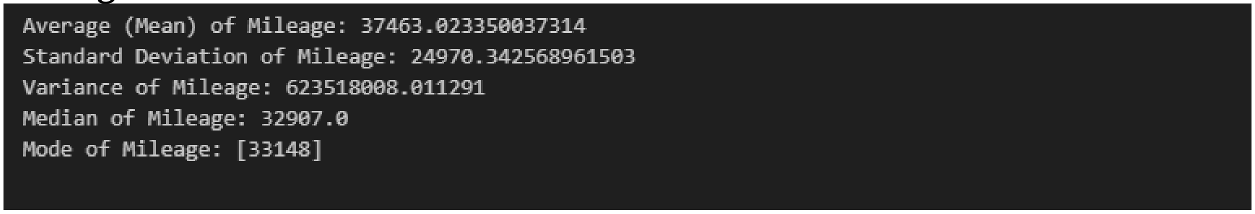
```
Average (Mean) of PerformanceRating: 4.696289583111206
Standard Deviation of PerformanceRating: 0.2536638912961854
Variance of PerformanceRating: 0.06434536974752295
Median of PerformanceRating: 4.7
Mode of PerformanceRating: [4.8]
```

Reliability Rating:



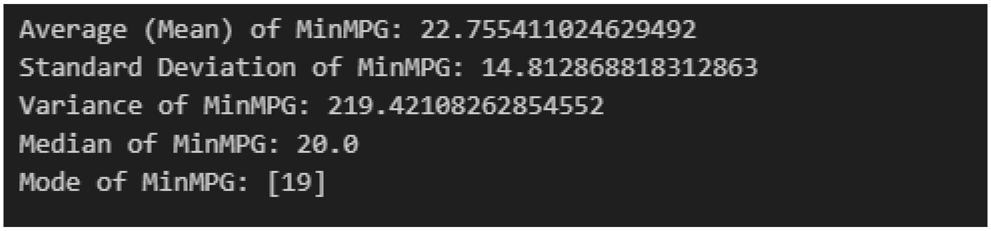
```
Average (Mean) of ReliabilityRating: 4.681746454845931
Standard Deviation of ReliabilityRating: 0.3681605977674965
Variance of ReliabilityRating: 0.13554222574852035
Median of ReliabilityRating: 4.8
Mode of ReliabilityRating: [4.8]
```

Mileage:



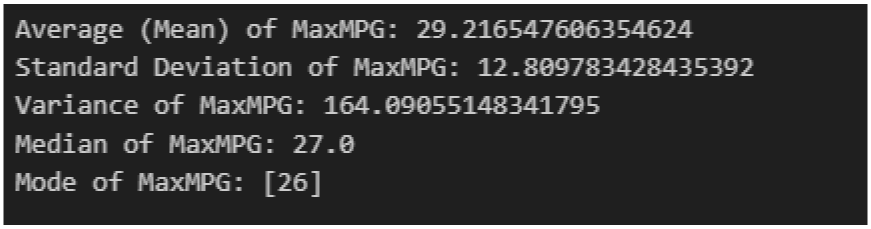
```
Average (Mean) of Mileage: 37463.023350037314
Standard Deviation of Mileage: 24970.342568961503
Variance of Mileage: 623518008.011291
Median of Mileage: 32907.0
Mode of Mileage: [33148]
```

Min MPG:



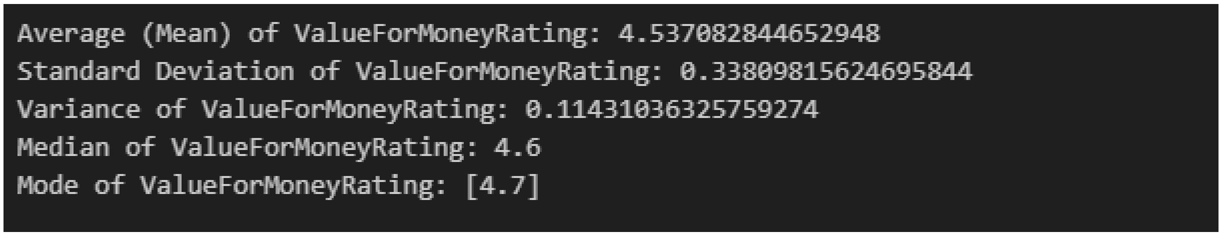
```
Average (Mean) of MinMPG: 22.755411024629492
Standard Deviation of MinMPG: 14.812868818312863
Variance of MinMPG: 219.42108262854552
Median of MinMPG: 20.0
Mode of MinMPG: [19]
```

Max MPG:



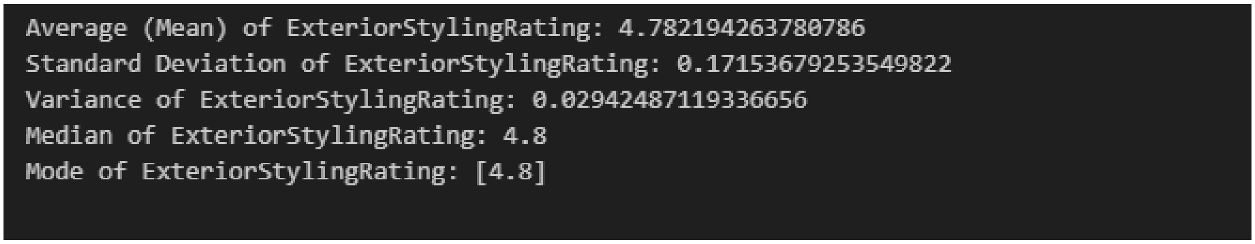
```
Average (Mean) of MaxMPG: 29.216547606354624
Standard Deviation of MaxMPG: 12.809783428435392
Variance of MaxMPG: 164.09055148341795
Median of MaxMPG: 27.0
Mode of MaxMPG: [26]
```

ValueForMoneyRating:



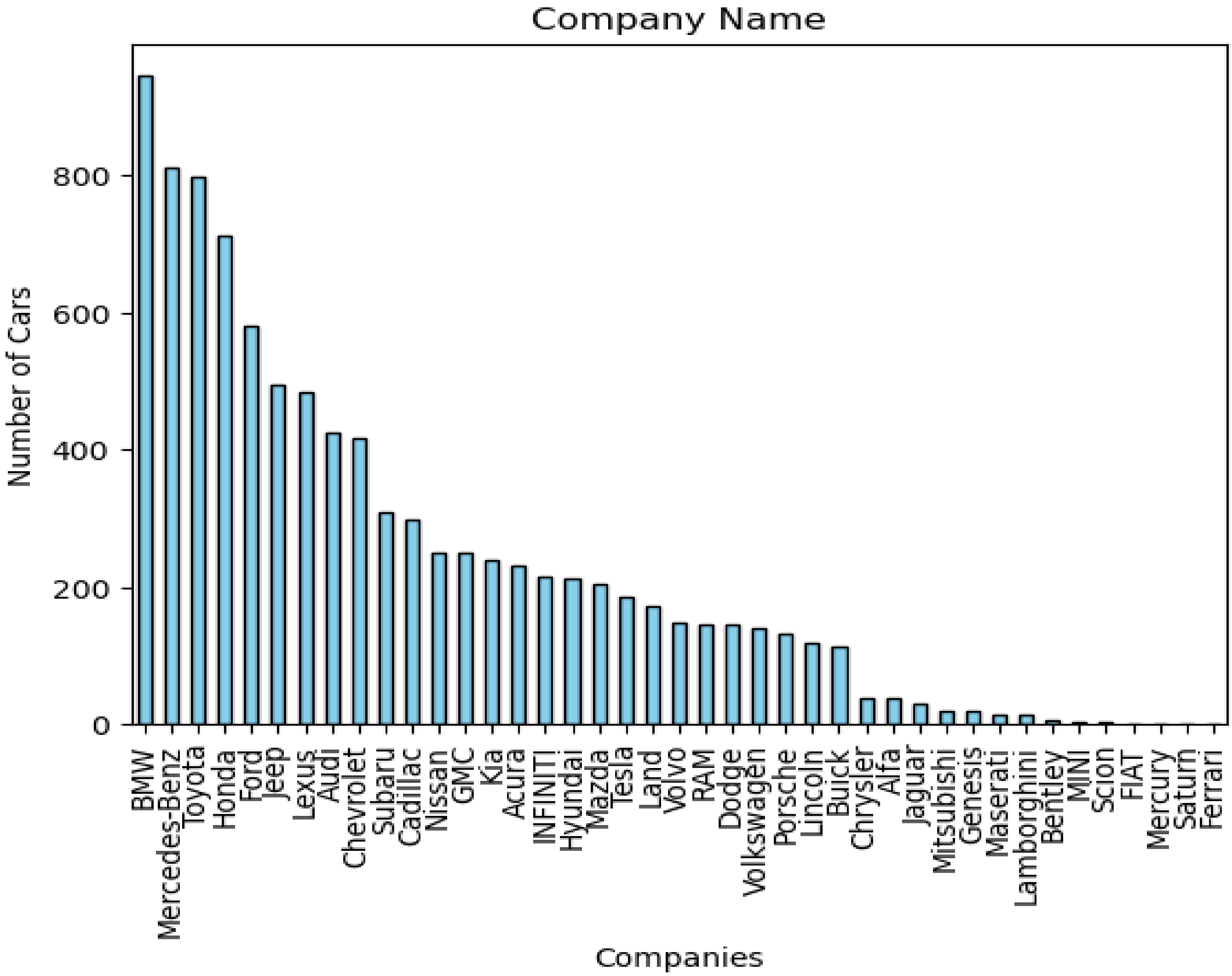
```
Average (Mean) of ValueForMoneyRating: 4.537082844652948
Standard Deviation of ValueForMoneyRating: 0.33809815624695844
Variance of ValueForMoneyRating: 0.11431036325759274
Median of ValueForMoneyRating: 4.6
Mode of ValueForMoneyRating: [4.7]
```

ExteriorStylingRating:



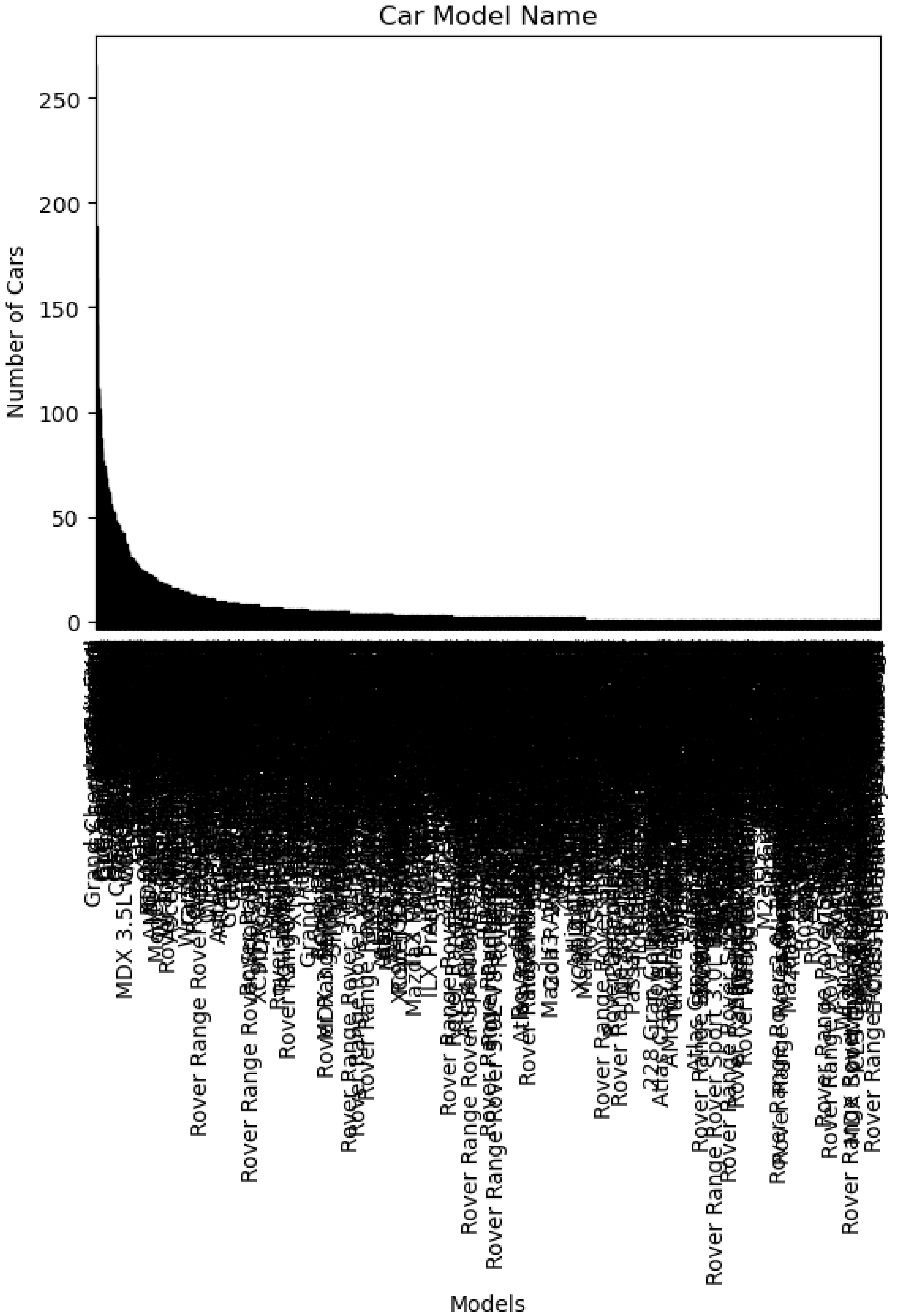
```
Average (Mean) of ExteriorStylingRating: 4.782194263780786
Standard Deviation of ExteriorStylingRating: 0.17153679253549822
Variance of ExteriorStylingRating: 0.02942487119336656
Median of ExteriorStylingRating: 4.8
Mode of ExteriorStylingRating: [4.8]
```

Companies:

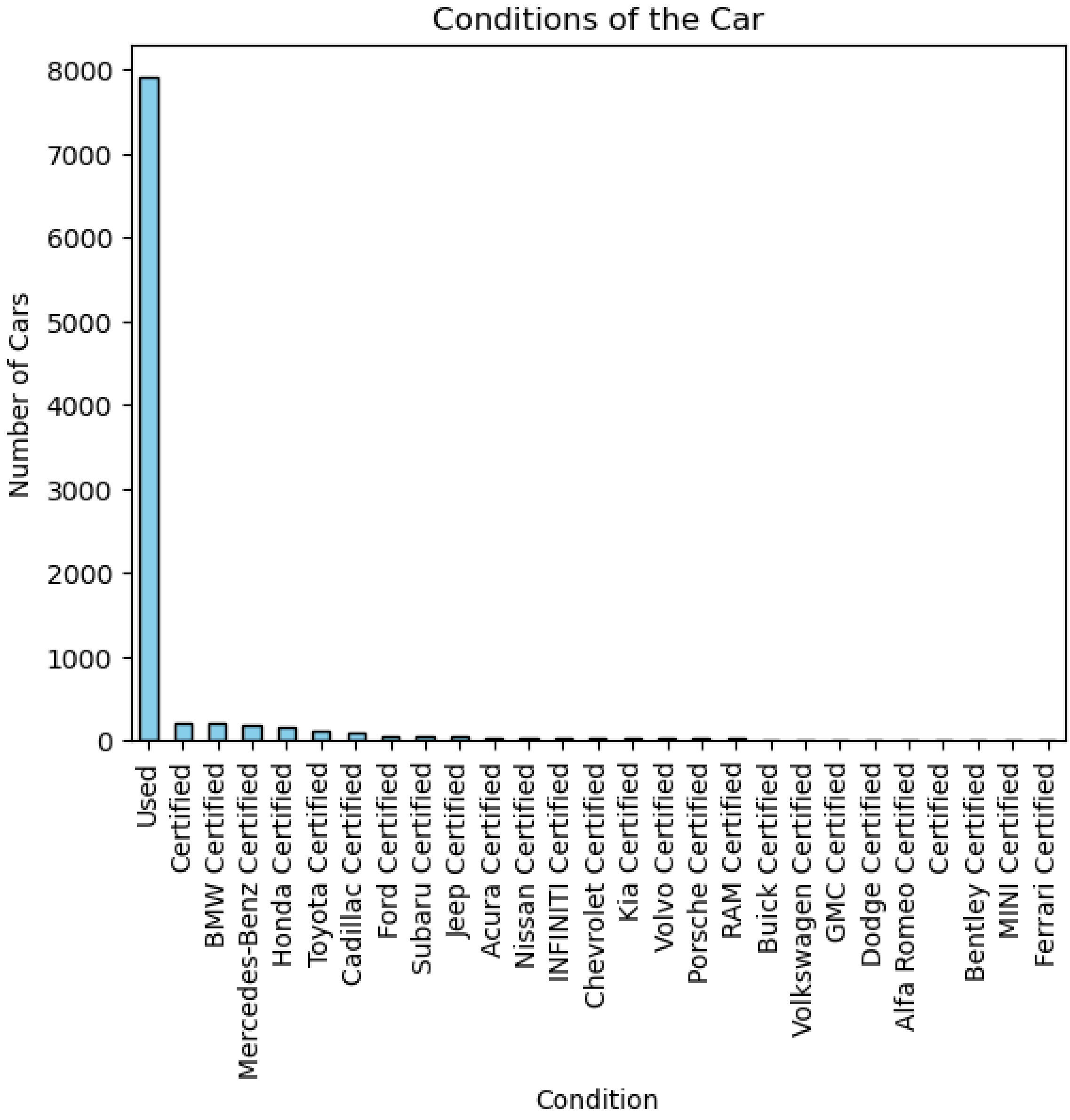


Company Name
Number of Cars
0
200
400
600
800
BMW
Mercedes-Benz
Toyota
Honda
Ford
Jeep
Lexus
Audi
Chevrolet
Subaru
Cadillac
Nissan
GMC
Kia
Acura
INFINITI
Hyundai
Mazda
Tesla
Land
Volvo
RAM
Dodge
Volkswagen
Porsche
Lincoln
Buick
Chrysler
Alfa
Jaguar
Mitsubishi
Genesis
Maserati
Lamborghini
Bentley
MINI
Scion
FIAT
Mercury
Saturn
Ferrari
Companies

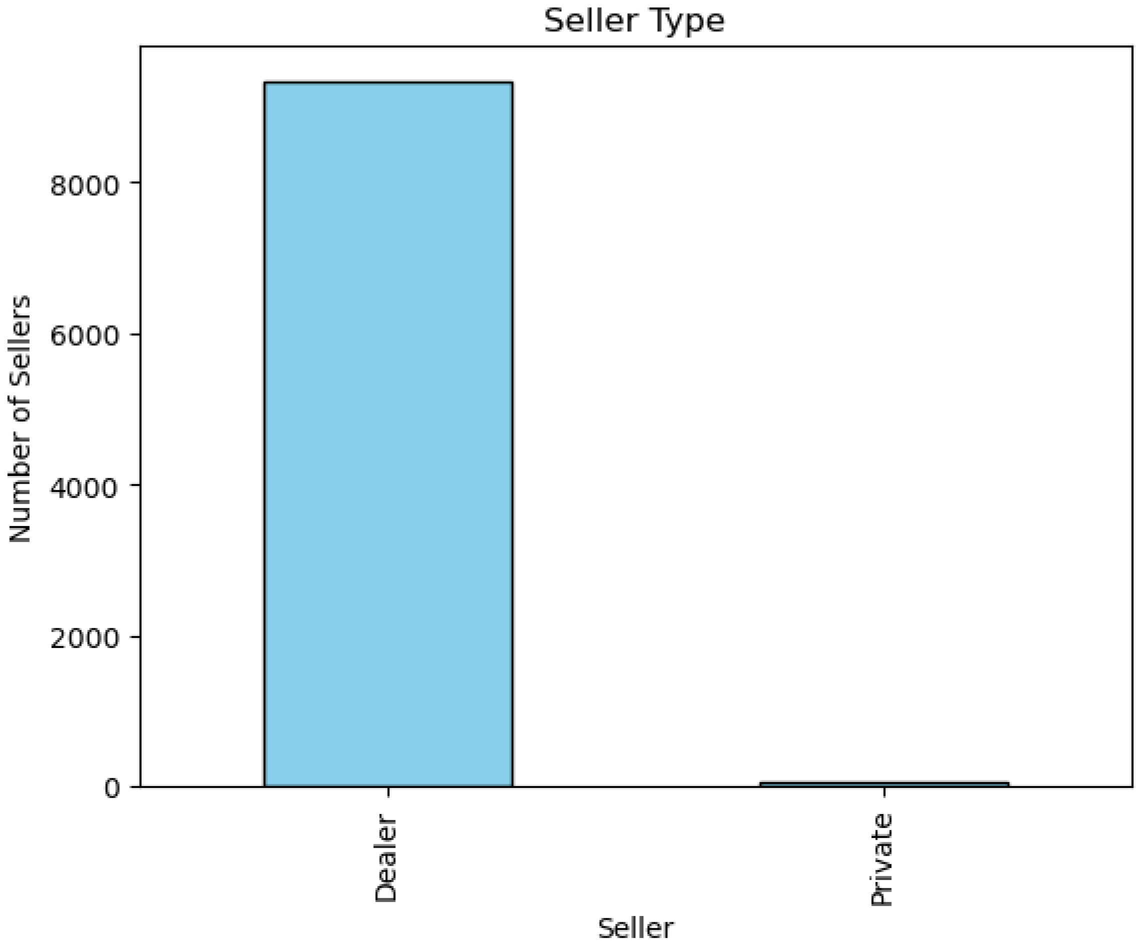
Model Names:



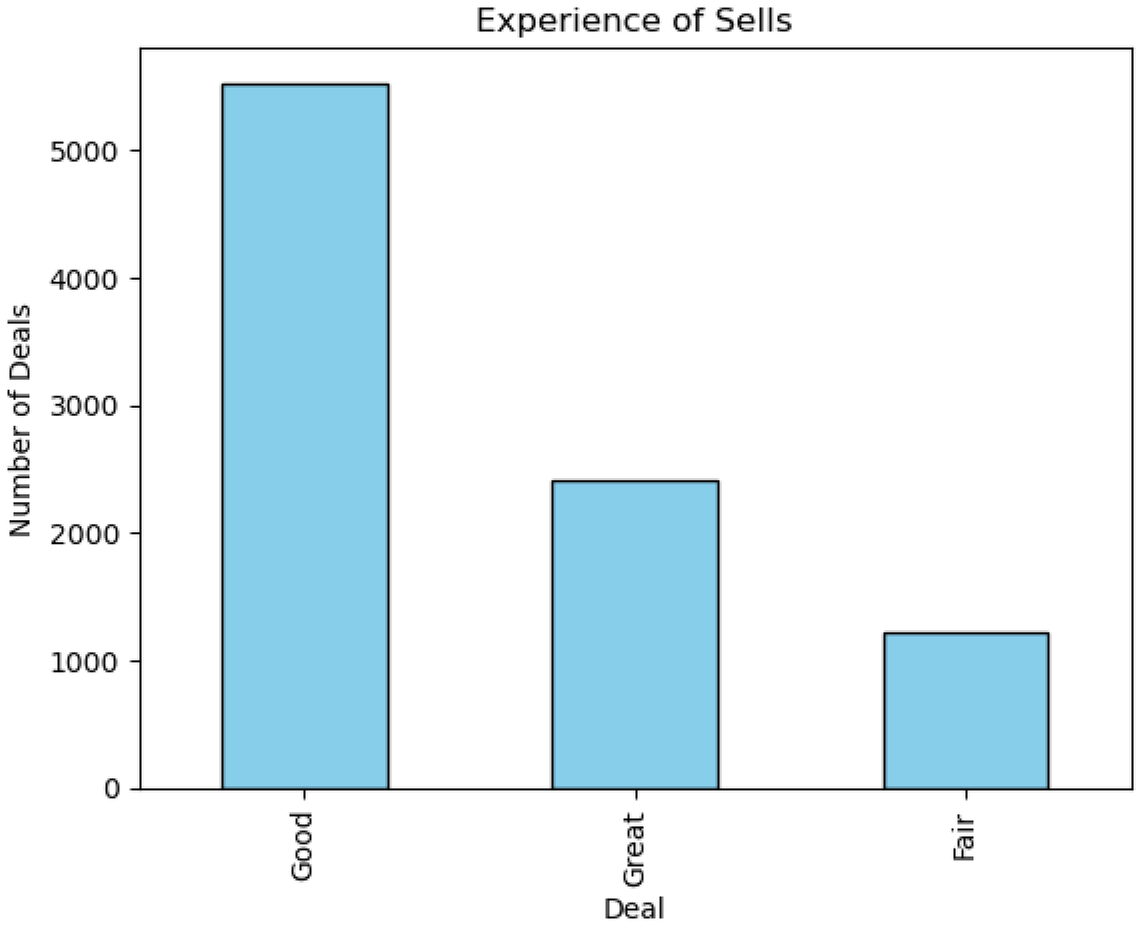
Conditions of the cars:



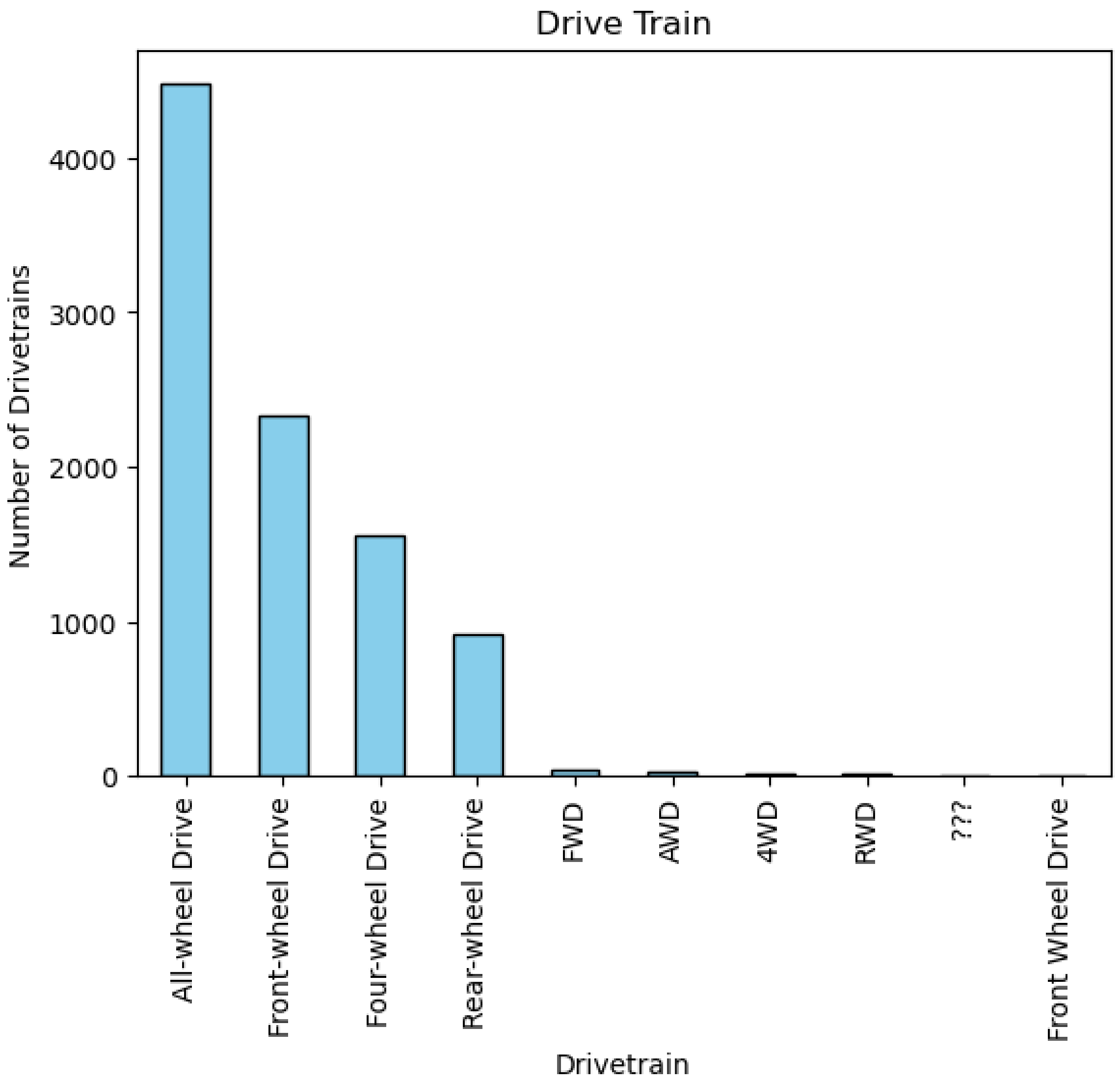
Seller type:


Seller Type
Number of Sellers
0
2000
4000
6000
8000
Dealer
Private
Seller

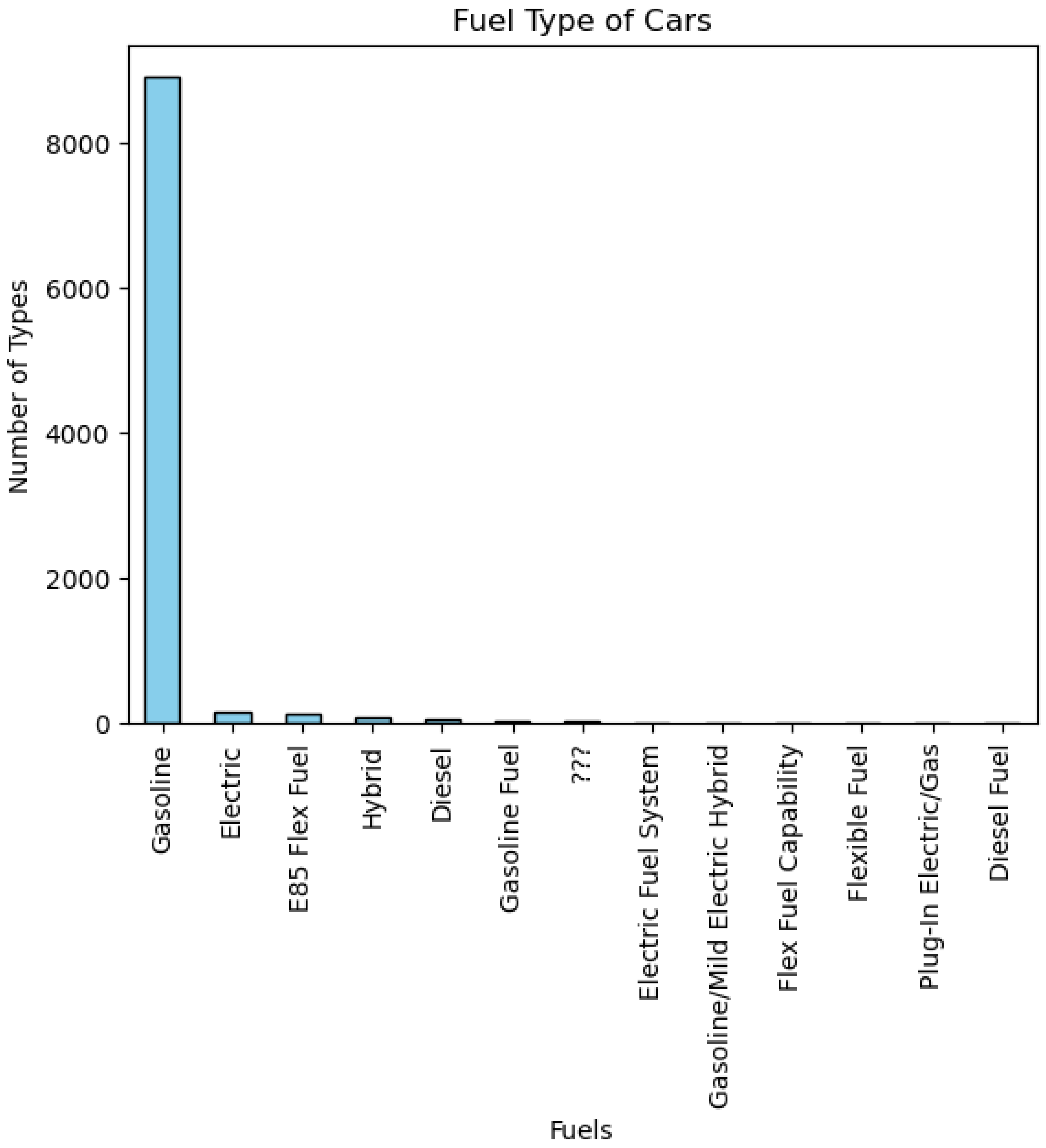
Deal Type:


Experience of Sells
Number of Deals
5000
4000
3000
2000
1000
0
Good
Great
Fair
Deal

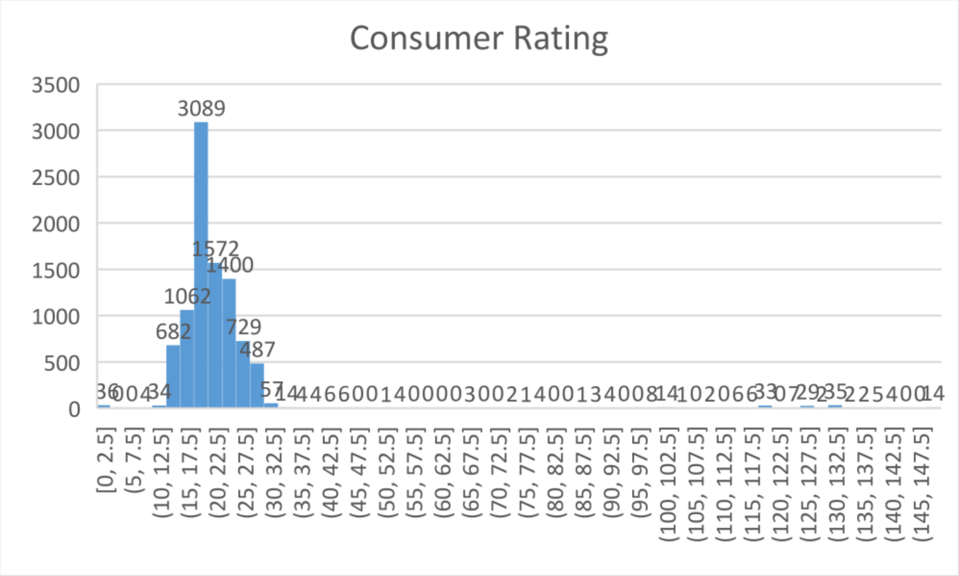
Drive Train:

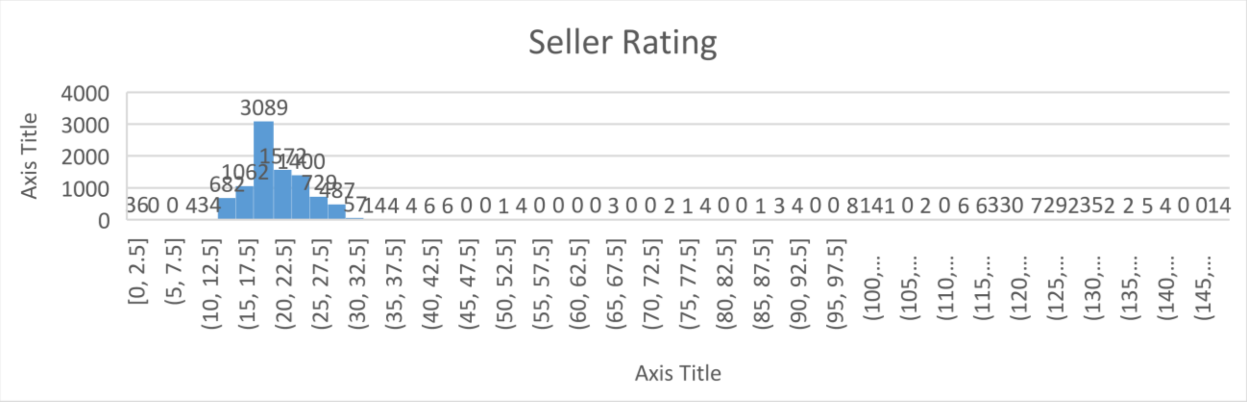

Drive Train
Number of Drivetrains
0
1000
2000
3000
4000
All-wheel Drive
Front-wheel Drive
Four-wheel Drive
Rear-wheel Drive
FWD
AWD
4WD
RWD
???
Front Wheel Drive
Drivetrain

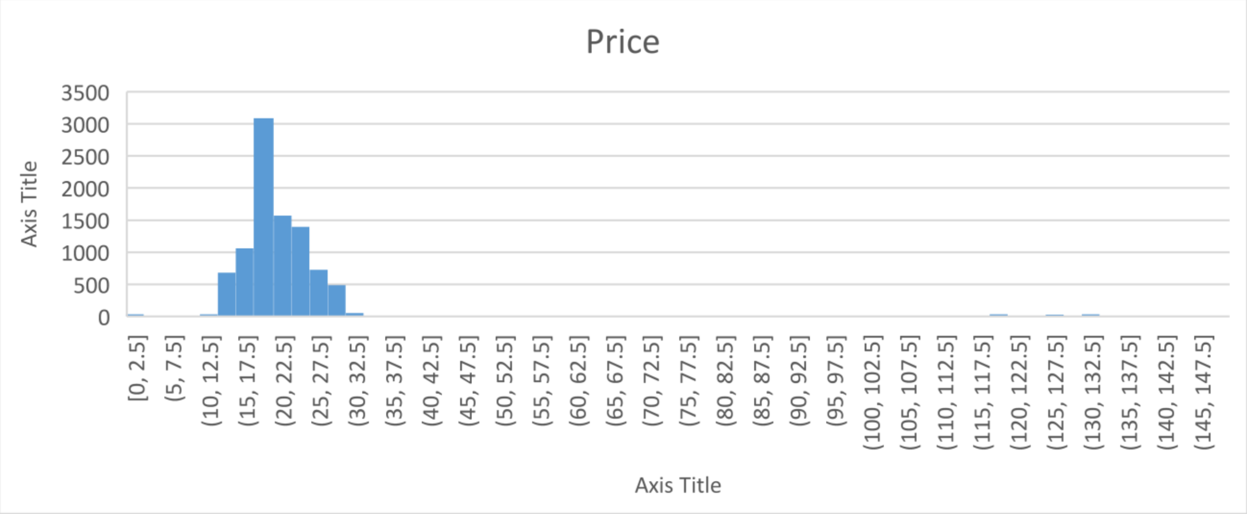
Fuel Type:



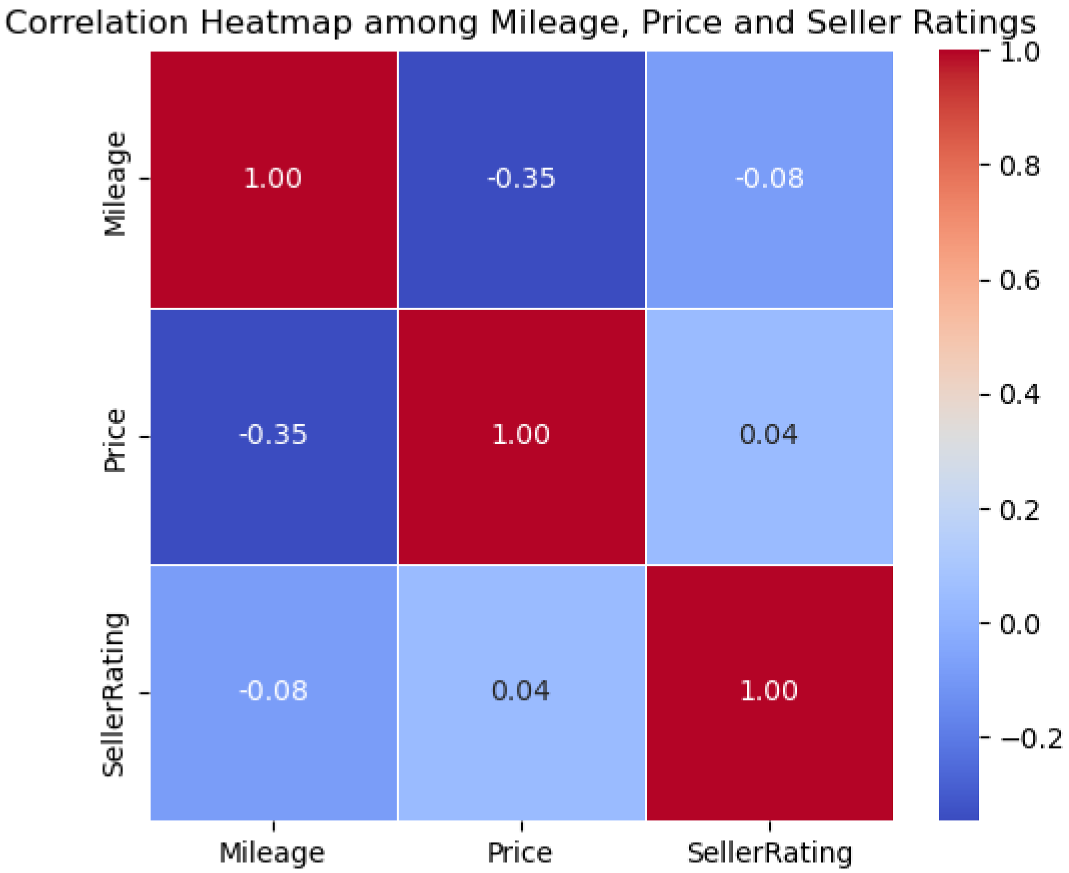
Plot bar chart, line graph or pie chart where necessary.


Consumer Rating
3500
3000
2500
2000
1500
1000
500
0
3089
1572
1400
1062
682
729
487
36004 34
5714 4466001400003002140013400814102066 33 07 29 2 35 225400 14
[0, 2.5]
(5, 7.5]
(10, 12.5]
(15, 17.5]
(20, 22.5]
(25, 27.5]
(30, 32.5]
(35, 37.5]
(40, 42.5]
(45, 47.5]
(50, 52.5]
(55, 57.5]
(60, 62.5]
(65, 67.5]
(70, 72.5]
(75, 77.5]
(80, 82.5]
(85, 87.5]
(90, 92.5]
(95, 97.5]
(100, 102.5]
(105, 107.5]
(110, 112.5]
(115, 117.5]
(120, 122.5]
(125, 127.5]
(130, 132.5]
(135, 137.5]
(140, 142.5]
(145, 147.5]


Seller Rating
Axis Title
4000
3000
2000
1000
0
3089
1572
1400
1062
682
729
487
360 0 434
57144 4 6 6 0 0 1 4 0 0 0 0 3 0 0 2 1 4 0 0 1 3 4 0 0 8141 0 2 0 6 6330 7292352 2 5 4 0 014
[0, 2.5]
(5, 7.5]
(10, 12.5]
(15, 17.5]
(20, 22.5]
(25, 27.5]
(30, 32.5]
(35, 37.5]
(40, 42.5]
(45, 47.5]
(50, 52.5]
(55, 57.5]
(60, 62.5]
(65, 67.5]
(70, 72.5]
(75, 77.5]
(80, 82.5]
(85, 87.5]
(90, 92.5]
(95, 97.5]
(100,...
(105,...
(110,...
(115,...
(120,...
(125,...
(130,...
(135,...
(140,...
(145,...
Axis Title


Price
Axis Title
3500
3000
2500
2000
1500
1000
500
0
[0, 2.5]
(5, 7.5]
(10, 12.5]
(15, 17.5]
(20, 22.5]
(25, 27.5]
(30, 32.5]
(35, 37.5]
(40, 42.5]
(45, 47.5]
(50, 52.5]
(55, 57.5]
(60, 62.5]
(65, 67.5]
(70, 72.5]
(75, 77.5]
(80, 82.5]
(85, 87.5]
(90, 92.5]
(95, 97.5]
(100, 102.5]
(105, 107.5]
(110, 112.5]
(115, 117.5]
(120, 122.5]
(125, 127.5]
(130, 132.5]
(135, 137.5]
(140, 142.5]
(145, 147.5]
Axis Title

Create a correlation heat-map with each column.



How to convert a categorical data to numerical values to get features for machine learning projects?

Converting **categorical data** into **numerical values** is an essential step in preparing data for machine learning models, as most models can only work with numerical inputs. There are several techniques for converting categorical data to numerical values, depending on the nature of the categorical variable (e.g., whether it's ordinal or nominal). Below are common methods to achieve this:

1. Label Encoding

- **What it is**: Label encoding assigns each unique category in a categorical feature a distinct integer value. It is best suited for **ordinal** data (where the categories have an inherent order).

Example: If you have a feature like Education Level with categories ["High School", "Bachelor's", "Master's", "PhD"], label encoding would assign values like:

- High School → 0
- Bachelor's → 1
- Master's → 2
- PhD → 3

How to use:

- In Python (using scikit-learn):

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['education_level'] = le.fit_transform(data['education_level'])
```

2. One-Hot Encoding

- **What it is**: One-hot encoding transforms each category into a new binary (0 or 1) column, where a 1 indicates the presence of that category, and a 0 indicates the absence. This method is best suited for **nominal** data (where categories don't have a specific order).

Example: If you have a feature like Color with categories ["Red", "Green", "Blue"], one-hot encoding would create 3 new columns:

- Red: 1 if the color is red, 0 otherwise.
- Green: 1 if the color is green, 0 otherwise.
- Blue: 1 if the color is blue, 0 otherwise.

How to use:

- In Python (using pandas):

  ```
  data = pd.get_dummies(data, columns=['color'])
  ```

  Or using scikit-learn:

  ```
  from sklearn.preprocessing import OneHotEncoder

  encoder = OneHotEncoder(sparse=False)

  encoded_data = encoder.fit_transform(data[['color']])
  ```

3. Ordinal Encoding

- **What it is**: Similar to label encoding, but explicitly for **ordinal** data where the categories have an ordered relationship. It assigns integers to categories based on their order.

Example: For a feature like Size with categories ["Small", "Medium", "Large"], ordinal encoding would assign:

- Small → 0
- Medium → 1
- Large → 2

How to use:

- This is often done manually (since it's based on the inherent order of the categories). You can use LabelEncoder for this as well, ensuring the correct ordering of categories:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['size'] = le.fit_transform(data['size'])
```

4. Binary Encoding

- **What it is**: Binary encoding is a compromise between label encoding and one-hot encoding. It encodes the categories into binary numbers, using fewer columns than one-hot encoding. It is most useful when dealing with **high cardinality** categorical variables (i.e., when there are many unique categories).

Example: If you have a feature Category with categories ["A", "B", "C", "D"], the binary encoding might look like:

- A → 00
- B → 01
- C → 10
- D → 11

How to use:

- You can use libraries like CategoryEncoders to apply binary encoding:

```
import category_encoders as ce
encoder = ce.BinaryEncoder(cols=['category'])
data = encoder.fit_transform(data)
```

5. Frequency or Count Encoding

- **What it is**: Frequency encoding replaces each category with the count (or frequency) of how often that category appears in the data. It’s useful when you have categorical variables with many unique categories but don’t want to use one-hot encoding due to dimensionality issues.

Example: For a feature like City with categories ["New York", "Paris", "Berlin", "New York"], frequency encoding would replace:

- New York → 2 (because it appears 2 times)
- Paris → 1
- Berlin → 1

How to use:

- This can be done manually in pandas:

```
data['city'] = data['city'].map(data['city'].value_counts())
```

6. Target Encoding (Mean Encoding)

- **What it is**: Target encoding replaces each category with the **mean** of the target variable for that category. This is typically used for **categorical features** that are related to a target variable.

Example: If you have a feature City and target Price, target encoding might replace:

- New York → mean(Price for New York)
- Paris → mean(Price for Paris)
- Berlin → mean(Price for Berlin)

How to use:

- This can be implemented using libraries like CategoryEncoders:

```
import category_encoders as ce
encoder = ce.TargetEncoder(cols=['city'])
data = encoder.fit_transform(data, data['price'])
```

7. Hashing Encoding (Feature Hashing)

- **What it is**: Hashing encoding applies a hash function to categories and reduces them to a fixed number of features, regardless of the original number of categories. It is useful when you have a large number of unique categories.

How to use:

- You can use the HashingEncoder from the category_encoders library:

```
import category_encoders as ce

encoder = ce.HashingEncoder(cols=['category'])

data = encoder.fit_transform(data)
```

Choosing the Right Encoding Method

- **Label Encoding**: Use for ordinal data (when categories have a meaningful order).
- **One-Hot Encoding**: Use for nominal data (when categories have no order).
- **Binary Encoding**: Useful for high cardinality categorical features (many unique categories).
- **Frequency/Count Encoding**: Useful for features with many categories and when you want to preserve frequency information.
- **Target Encoding**: Useful when there is a relationship between the categorical feature and the target variable.
- **Hashing Encoding**: Useful for high-cardinality features where other encoding methods may result in too many features.

By applying the appropriate encoding techniques, you can convert categorical features into numerical form, allowing machine learning models to process the data effectively.

Also mention if there are any missing values in any column of not? How do you want to handle those missing values and why?

Studying the whole csv file, there were few data that were needed to be checked. There were few missing values, and the values were inserted by studying the past values of that objects. In some cases, the values were put as per trend. If these values were not present, our model would have not performed according to our expectation. That’s why, to avoid such occurrences, the dataset was taken care of.

Write a conclusion paragraph stating your overall observation.

We have carried out range of calculations to draw conclusion. The sale of a car depends of lot of categories. It seeks the price, mileage, comfort rating, engine type and many others. Overall the dataset is not perfectly balanced. The dataset holds some major data numbers that would make it baised.