Project Title	**Used Bike Prices - Feature Engineering and EDA**
Tools	Visual Studio code / jupyter notebook
Domain	Finance Analyst
Project Difficulties level	Advance

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

About this file

Dataset Information

model_name

The name of the bike's model. It contains some additional information like model year,engine etc.

model_year

The year in which the model was built.

kms_driven

Total kilometers the bike has been driven.

owner

The represents which type of owner the bike has like it is first owner which means the current owner had bought this bike as new, second owner means the bike has been sold to this owner from first owner and so on.

location

The location of the seller.

mileage

Average mileage the bike gives. Its is represented as kilometer per liter of petrol (kmpl).

power

Power is in terms of Bhp. BHP is the rate at which the torque generated by the engine in a bike is delivered to the wheels. Such that faster the deliverability, higher is the speed of the motorcycle and vice versa. For a bike that consists of a lower BHP can pull higher loads and for a bike that contains a greater BHP can propel the bike at faster speeds.

Thus, BHP depends on several factors such as deliverability, weight and power generation of the bike. It’s the most common way of rating a bike and determining which ones might be faster

price

This is the target variable of the dataset. t is in Indian rupee

Example: You can get the basic idea how you can create a project from here

Creating a major Machine Learning (ML) project on "Bike Prices - Feature Engineering and Exploratory Data Analysis (EDA)" requires an organized and systematic approach, especially to meet the level of detail expected for a 5-year experienced professional. Below is an outline and sample step-by-step guide that can help you build a comprehensive project.

Due to limitations here, I will provide a condensed version of the key steps involved. Let's break down the project into logical sections with major tasks:

Project Outline: Bike Prices Analysis

Dataset Columns:

- `model_name`: Name of the bike model
- `model_year`: Year the model was manufactured
- `kms_driven`: Kilometers driven by the bike
- `owner`: Owner category (first, second, etc.)
- `location`: City or region of the sale
- `mileage`: Fuel efficiency of the bike
- `power`: Power rating of the bike
- `price`: Selling price of the bike (target variable)
- `cc`: Engine capacity in cubic centimeters
- `brand`: Brand of the bike

Step-by-Step Project Guide

---

1. Data Collection and Preparation

1. **Loading and Inspecting the Data**
    - Load the dataset.
    - Inspect the dataset for basic structure, types, and any initial inconsistencies.

python
code

```
import pandas as pd


# Load the dataset
df = pd.read_csv("bike_prices.csv")
# Check first few rows
print(df.head())
print(df.info())
```

2.
3. **Data Cleaning**
    - **Missing Values**: Identify and handle missing values.
    - **Duplicates**: Check and remove any duplicate rows.
    - **Inconsistent Data**: Address inconsistencies like typos or irregular formats in `location` or `owner`.

python
code

```
# Check missing values
print(df.isnull().sum())
```

```
# Fill missing values (if any) or decide on removal based on analysis
df = df.dropna()  # For simplicity, remove rows with missing values
```

4.

2. Exploratory Data Analysis (EDA)

1. **Descriptive Statistics**
    - Get a statistical summary of each numerical column (`price`, `kms_driven`, `mileage`, `power`, etc.).

python
code

```
print(df.describe())
```

2.

3. **Univariate Analysis**
    - Plot histograms for continuous variables like `price`, `mileage`, `power`, `kms_driven`.
    - Analyze the frequency distribution of categorical features such as `model_year`, `owner`, and `brand`.

python
code

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


# Price Distribution
```

```
plt.figure(figsize=(8,6))
sns.histplot(df['price'], kde=True)
plt.title('Price Distribution')
plt.show()
```

4.

5. **Bivariate Analysis**
    - Analyze relationships, e.g., `price` vs. `model_year`, `price` vs. `kms_driven`.
    - Correlation heatmap for numerical variables to understand dependencies.

python
code

```
# Price vs Year
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x='model_year', y='price')
plt.title('Price vs Model Year')
plt.show()
```

6.

3. Feature Engineering

1. **Creating New Features**
    - **Age of Bike**: Calculate based on `model_year` and the current year.
    - **Power-to-Weight Ratio**: For bikes where both `power` and `cc` are available, create a ratio feature.
    - **Location Encoding**: Use encoding techniques to convert `location` to a numerical format.

python

code

```
# Example: Age of Bike
current_year = 2023
df['bike_age'] = current_year - df['model_year']
```

2.
3. **Handling Categorical Variables**
    - Use one-hot encoding for categorical features like `brand`, `location`, and `owner`.

python

code

```
# One-hot encode the brand and owner columns
df = pd.get_dummies(df, columns=['brand', 'owner'],
drop_first=True)
```

4.
5. **Feature Transformation**
    - **Normalization/Standardization**: Standardize numerical features such as `price`, `kms_driven`, `mileage`, and `power`.
    - **Log Transformation**: For skewed data, apply log transformation.

python

code

```
# Standardizing numerical features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['price', 'kms_driven', 'mileage', 'power']] =
```

```
scaler.fit_transform(df[['price', 'kms_driven', 'mileage',
'power']])
```

6.

4. Data Visualization

1. **Multivariate Plots**
    - Pairplot or scatter matrix for all numerical columns to visualize relationships.
    - Boxplots for categorical columns like `brand` and `owner` against `price`.

python
code

```
# Pairplot for selected features
sns.pairplot(df[['price', 'kms_driven', 'mileage', 'power',
'bike_age']])
plt.show()
```

2.
3. **Geographical Analysis**
    - Use maps or bar plots to show average bike prices by `location`.
4. **Time Series Analysis**
    - Analyze price trends over years if data has a temporal component.

5. Model Building

1. **Splitting Data**
    - Separate features (X) and target variable (y, which is `price`).
    - Split the data into training and testing sets.

python

code

```
from sklearn.model_selection import train_test_split


X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

2.
3. **Model Selection**
    - Start with simple regression models like Linear Regression, Ridge, or Lasso.
    - Experiment with tree-based models like Random Forest or Gradient Boosting.
4. **Model Training and Evaluation**
    - Train the models and use metrics like MAE, MSE, and R^2 to evaluate performance.

python
code

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


model = LinearRegression()
model.fit(X_train, y_train)


# Predict and evaluate
predictions = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
print(f'MSE: {mse}, R²: {r2}')
```

5.
6. **Hyperparameter Tuning**
    - Use techniques like GridSearchCV for hyperparameter tuning to improve model performance.

python
code

```
from sklearn.model_selection import GridSearchCV

# Example: RandomForest with GridSearchCV
from sklearn.ensemble import RandomForestRegressor
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}
grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

7.

6. Conclusion and Recommendations

- **Insights**: Summarize findings such as which factors most influence bike prices.
- **Model Evaluation**: Discuss model performance, strengths, and limitations.

- **Next Steps**: Suggest improvements like additional data collection, feature engineering, or advanced modeling.

Documentation

- Ensure that the report includes code explanations, visualizations, and key findings. For a lengthy document, include an index, and clearly divide sections.

Example: You can get the basic idea how you can create a project from here

Sample code with output

Used Bike Prices - EDA and Modelling

Used Bike Price prediction using Machine Learning.

In [1]:

```
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
```

Data Loading

In [2]:

```
data =
pd.read_csv("../input/used-bike-price-in-india/bikes.csv")
data.head()
```

Out[2]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price
0	Bajaj Avenger Cruise 220 2017	2017	17000 Km	first owner	hyderabad	\n\n 35 kmpl	19 bhp	63500
1	Royal Enfield Classic 350cc 2016	2016	50000 Km	first owner	hyderabad	\n\n 35 kmpl	19.80 bhp	115000
2	Hyosung GT250R 2012	2012	14795 Km	first owner	hyderabad	\n\n 30 kmpl	28 bhp	300000
3	Bajaj Dominar 400 ABS 2017	2017	Mileage 28 Kms	first owner	pondicherry	\n\n 28 Kms	34.50 bhp	100000
4	Jawa Perak 330cc 2020	2020	2000 Km	first owner	bangalore	\n\n	30 bhp	197500

Data Engineering

In [3]:

```
data.isna().sum()
```

```
Out[3]:
model_name     0
model_year     0
kms_driven     0
owner          0
location      19
mileage       11
power         31
price          0

dtype: int64
```

CC from Model name

From the model name, following can be extracted :

- CC of the bike
- Brand and Model name

```
In [4]:
cc = []
for veh in data.model_name:
        models = veh.split(" ")
        models = " ".join(models[:len(models)-1]).lower()
        if re.search('[0-9]*cc',models,flags=re.I) != None:
```

```
                if
models[re.search('[0-9]*cc',models,flags=re.I).start():re.searc
h('[0-9]*cc',models,flags=re.I).end()] not in ['cc']:

cc.append(models[re.search('[0-9]*cc',models,flags=re.I).start(
):re.search('[0-9]*cc',models,flags=re.I).end()])
                else:
                    cc.append(models)
        elif re.search('[0-9]* (cc)',models,flags=re.I) !=
None:
                cc.append(models[re.search('[0-9]*
(cc)',models,flags=re.I).start():re.search('[0-9]*
(cc)',models,flags=re.I).end()] )
        elif  "hyosung" in models:
                cc.append(models)
        else:
                cc.append(models)
```

Few entries require manual processing because of their different naming convention and non availability of engine cc information in their naming

In [5]:

```
for i in range(len(cc)):
    if "1000" in cc[i]:
        cc[i] = "1000cc"
```

```
    elif "310" in cc[i]:
        cc[i] = "310cc"
    elif "apache rtr 200" in cc[i]:
        cc[i] = "200cc"
    elif "ns200" in cc[i]:
        cc[i] = "200cc"
    elif "rs200" in cc[i]:
        cc[i] = "200cc"
    elif "220" in cc[i]:
        cc[i] = "220cc"
    elif "400" in cc[i]:
        cc[i] = "400cc"
    elif "250" in cc[i]:
        cc[i] = "250cc"
    elif "125" in cc[i]:
        cc[i] = "125cc"
    elif "160" in cc[i]:
        cc[i] = "160cc"
    elif "150" in cc[i]:
        cc[i] = "150cc"
    elif "350" in cc[i]:
        cc[i] = "350cc"
    elif "200" in cc[i]:
        cc[i] = "200cc"
    elif "100" in cc[i]:
```

```
        cc[i] = "100cc"
    elif "180" in cc[i]:
        cc[i] = "180cc"
    elif "110" in cc[i]:
        cc[i] = "110cc"
    elif "390" in cc[i]:
        cc[i] = "390cc"
    elif "135" in cc[i]:
        cc[i] = "135cc"
    elif "r15" in cc[i]:
        cc[i] = "150cc"
    elif "650" in cc[i]:
        cc[i] = "650cc"
    elif "750" in cc[i]:
        cc[i] = "750cc"
    elif "800" in cc[i]:
        cc[i] = "800cc"
    elif "300" in cc[i]:
        cc[i] = "300cc"
    elif "765" in cc[i]:
        cc[i] = "765cc"
    elif "883" in cc[i]:
        cc[i] = "883cc"
    elif "797" in cc[i]:
        cc[i] = "797cc"
```

```
    elif "810" in cc[i]:
        cc[i] = "810cc"
    elif "321" in cc[i]:
        cc[i] = "321cc"
    elif "821" in cc[i]:
        cc[i] = "821cc"
    elif "120" in cc[i]:
        cc[i] = "120cc"
    elif "1745" in cc[i]:
        cc[i] = "1745cc"
    elif "899" in cc[i]:
        cc[i] = "899cc"
    elif "900" in cc[i]:
        cc[i] = "900cc"
    elif "302" in cc[i]:
        cc[i] = "302cc"
    elif "959" in cc[i]:
        cc[i] = "959cc"
    elif "600" in cc[i]:
        cc[i] = "600cc"
    elif "502" in cc[i]:
        cc[i] = "502cc"
    elif "um renegade" in cc[i]:
        cc[i] = "279cc"
    elif "hero splendor" in cc[i]:
```

```
        cc[i] = "97cc"
    elif "hero passion plus" in cc[i]:
        cc[i] = "97cc"
    elif "yamaha fz" in cc[i]:
        cc[i] = "150cc"
    elif "honda hornet" in cc[i]:
        cc[i] = "184cc"
    elif "royal enfield interceptor" in cc[i]:
        cc[i] = "650cc"
    elif "hero passion pro" in cc[i]:
        cc[i] = "113cc"
    elif "hero passion xpro" in cc[i]:
        cc[i] = "109cc"
    elif "harley-davidson street bob" in cc[i]:
        cc[i] = "1868cc"
    elif "harley-davidson fat bob" in cc[i]:
        cc[i] = "1868cc"
    elif "harley-davidson fat boy" in cc[i]:
        cc[i] = "1868cc"
    elif "harley-davidson street rod" in cc[i]:
        cc[i] = "749cc"
    elif "zx-10r" in cc[i]:
        cc[i] = "1000cc"
    elif "rsv4" in cc[i]:
        cc[i] = "1099cc"
```

```
    elif "tvs sport" in cc[i]:
        cc[i] = "109cc"
    elif "tvs star city" in cc[i]:
        cc[i] = "109cc"
    elif "harley-davidson superlow" in cc[i]:
        cc[i] = "883cc"
    elif "harley-davidson roadster" in cc[i]:
        cc[i] = "1202cc"
    elif "harley-davidson forty eight" in cc[i]:
        cc[i] = "1202cc"
    elif "harley-davidson night rod special" in cc[i]:
        cc[i] = "1247cc"
    elif "triumph rocket iii roadster" in cc[i]:
        cc[i] = "2458cc"
    elif "triumph thunderbird lt" in cc[i]:
        cc[i] = "1699cc"
    elif "kawasaki vulcan s black" in cc[i]:
        cc[i] = "649cc"
    elif "mahindra mojo black pearl" in cc[i]:
        cc[i] = "300cc"
    elif "ducati diavel carbon" in cc[i]:
        cc[i] = "1198cc"
    elif "triumph tiger explorer" in cc[i]:
        cc[i] = "1215cc"
    elif "royal enfield continental" in cc[i]:
```

```
        cc[i] = "650cc"
    elif "triumph street triple s" in cc[i]:
        cc[i] = "765cc"
    elif "harley-davidson breakout" in cc[i]:
        cc[i] = "1868cc"
    elif "harley-davidson cvo" in cc[i]:
        cc[i] = "1923cc"
```

Remove the CC from names and convert all to integer

In [6]:

```
cc[cc.index('hero karizma 223 cc')] = '223cc'
cc = [int(i.replace('cc','')) for i in cc]
```

In [7]:

```
data['cc'] = cc
data
```

Out[7]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc

0	Bajaj Avenger Cruise 220 2017	2017	17000 Km	first owner	hyderabad	\n\n35 kmpl	19 bhp	63500	220
1	Royal Enfield Classic 350cc 2016	2016	50000 Km	first owner	hyderabad	\n\n35 kmpl	19.80 bhp	115000	350
2	Hyosung GT250R 2012	2012	14795 Km	first owner	hyderabad	\n\n30 kmpl	28 bhp	300000	250
3	Bajaj Dominar 400 ABS 2017	2017	Mileage 28 Kms	first owner	pondicherry	\n\n28 Kms	34.50 bhp	100000	400
4	Jawa Perak 330cc 2020	2020	2000 Km	first owner	bangalore	\n\n	30 bhp	197500	330
...	...	...	...	...	...	...	...	...	...

7852	Yamaha YZF-R15 150cc 2011	2011	7000 Km	first owner	agra	\n\n42 kmpl	16 bhp	55000	150
7853	Bajaj Discover 100cc 2015	2015	Mileage 80 Kmpl	first owner	delhi	\n\n80 Kmpl	7.7	28000	100
7854	Bajaj Pulsar 180cc 2016	2016	6407 Km	first owner	bangalore	\n\n65 kmpl	17 bhp	61740	180
7855	Bajaj V15 150cc 2016	2016	7524 Km	first owner	bangalore	\n\n57 kmpl	11.80 bhp	49000	150
7856	Bajaj Pulsar 220cc 2016	2016	15000 Km	first owner	chennai	\n\n38 kmpl	21 bhp	65000	220

7857 rows × 9 columns

Split the model name by spaces and take first split as the brand name

In [8]:

```
np.unique([i.split()[0] for i in data.model_name])
```

Out[8]:

```
array(['Aprilia', 'BMW', 'Bajaj', 'Benelli',
'BenelliImperiale', 'Ducati',
       'Fb', 'Harley-Davidson', 'Hero', 'Honda', 'Husqvarna',
'Hyosung',
       'Ideal', 'Indian', 'Jawa', 'KTM', 'Kawasaki', 'MV',
'Mahindra',
       'Moto', 'Royal', 'Suzuki', 'TVS', 'Triumph', 'UM',
'Yamaha',

       'Yazdi', 'yamaha'], dtype='<U16')
```

Benelli has inconsistent naming convetion shows repeating values. Preprocess them to generate spacing between brand and model name

In [9]:

```
data[data.model_name.str.contains("BenelliImperiale") > 0]
```

Out[9]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc
2809	BenelliImperiale 400 BS6 2020	2020	2800 Km	first owner	mumbai	\n\n35 kmpl	20.7 bhp	207500	400
2978	BenelliImperiale 400 2020	2020	1500 Km	first owner	mumbai	\n\n	20.7 bhp	207500	400
2979	BenelliImperiale 400 2020	2020	3500 Km	first owner	mumbai	\n\n	20.7 bhp	206462	400
4177	BenelliImperiale 400 2020	2020	3500 Km	first owner	mumbai	\n\n	20.7 bhp	220000	400
4399	BenelliImperiale 400 2020	2020	7700 Km	first owne	bangalore	\n\n	20.7 bhp	220000	40

				r					0
5177	BenelliImperiale 400 2020	2020	1900 Km	first owner	secunderabad	\n\n	20.7 bhp	240000	400

In [10]:

```
data.model_name.replace('BenelliImperiale 400 2020','Benelli Imperiale 400 2020',inplace=True)
data.model_name.replace('BenelliImperiale 400 BS6 2020','Benelli Imperiale 400 BS6 2020',inplace=True)
```

In [11]:

```
data[data.model_name.str.contains("Benelli") > 0]
```

Out[11]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc
16	Benelli 302R	2017	11000	first	bangalo	\n\n	38.26	230	30

51	300CC 2017		Km	owner	re		bhp	000	0
1668	Benelli 302R 300CC 2018	2018	23933 Km	first owner	bangalore	\n\n	38.26 bhp	270000	3000
1926	Benelli 302R 300CC 2018	2018	15025 Km	second owner	mumbai	\n\n	38.26 bhp	240000	3000
1939	Benelli Leoncino 500cc 2019	2019	3300 Km	first owner	delhi	\n\n	46.8 bhp	435262	5000
2140	Benelli TNT 300 ABS 2020	2020	4500 Km	first owner	delhi	\n\n 25 kmpl	37.73 bhp	295000	3000
2307	Benelli TNT 300 2017	2017	10000 Km	first owner	mumbai	\n\n 25 kmpl	37.73 bhp	259375	3000

2420	Benelli TNT 300 ABS 2020	2020	13000 Km	first owner	belgaum	\n\n25 kmpl	37.73 bhp	315000	300
2435	Benelli TNT 899 2015	2015	3174 Km	first owner	jajpur	\n\n17 kmpl	119.37 bhp	569400	899
2554	Benelli 302R 300CC 2019	2019	12000 Km	first owner	ranchi	\n\n	38.26 bhp	269000	300
2652	Benelli TNT 600 GT 2016	2016	11000 Km	first owner	chandigarh	\n\n19 kmpl	82.70 bhp	497000	600
2809	Benelli Imperiale 400 BS6 2020	2020	2800 Km	first owner	mumbai	\n\n35 kmpl	20.7 bhp	207500	400
29	Benelli TNT 300	2017	11000	first	mumbai	\n\n25	37.73	240	30

32	2017		Km	owner		kmpl	bhp	000	0
2978	Benelli Imperiale 400 2020	2020	1500 Km	first owner	mumbai	\n\n	20.7 bhp	207500	400
2979	Benelli Imperiale 400 2020	2020	3500 Km	first owner	mumbai	\n\n	20.7 bhp	206462	400
3020	Benelli TNT 300 2016	2016	6000 Km	first owner	delhi	\n\n 25 kmpl	37.73 bhp	255000	300
3281	Benelli TNT 899 2017	2017	5000 Km	second owner	bangalore	\n\n 17 kmpl	119.37 bhp	600000	899
3366	Benelli 302R 300CC 2017	2017	13236 Km	first owner	chennai	\n\n	38.26 bhp	218000	300

3390	Benelli 302R 300CC 2017	2017	17000 Km	first owner	chennai	\n\n	38.26 bhp	250000	300
3588	Benelli 302R 2017	2017	12000 Km	first owner	jalandhar	\n\n	38.26 bhp	250000	302
3647	Benelli TNT 300 ABS 2019	2019	5000 Km	first owner	chennai	\n\n 25 kmpl	37.73 bhp	295000	300
3675	Benelli TNT 300 2016	2016	17500 Km	second owner	pune	\n\n 25 kmpl	37.73 bhp	151045	300
3759	Benelli TNT 300 2016	2016	30000 Km	second owner	mhasla	\n\n 25 kmpl	37.73 bhp	185000	300
37	Benelli TNT 25	2017	33500	first	chennai	\n\n 25	28.16	125	25

86	250cc 2017		Km	owner		kmpl	bhp	000	0
3899	Benelli TNT 300 2016	2016	9000 Km	first owner	chennai	\n\n 25 kmpl	37.73 bhp	203921	300
3908	Benelli Leoncino 250 2019	2019	11000 Km	first owner	secund erabad	\n\n	25.1 bhp	208800	250
3920	Benelli 302R 300CC 2019	2019	29300 Km	first owner	ahmeda bad	\n\n	38.26 bhp	280000	300
3972	Benelli TNT 300 2015	2015	21500 Km	first owner	mumbai	\n\n 25 kmpl	37.73 bhp	180000	300
3977	Benelli 302S 2021	2021	16000 Km	first owner	ahmeda bad	\n\n	NaN	315000	302

4118	Benelli Leoncino 500cc 2020	2020	1000 Km	first owner	mumbai	\n\n	46.8 bhp	466875	5000
4133	Benelli TNT 300 2016	2016	15000 Km	first owner	mumbai	\n\n 25 kmpl	37.73 bhp	228250	3000
4135	Benelli TNT 300 2018	2018	10000 Km	first owner	mumbai	\n\n 25 kmpl	37.73 bhp	259375	3000
4137	Benelli TNT 300 2016	2016	3500 Km	first owner	mumbai	\n\n 25 kmpl	37.73 bhp	233437	3000
4142	Benelli TNT 600i 2016	2016	14000 Km	first owner	mumbai	\n\n 19 kmpl	82.70 bhp	415000	6000
41	Benelli Imperiale 400	2020	3500	first	mumbai	\n\n	20.7	220	40

77	2020		Km	owner			bhp	000	0
4399	Benelli Imperiale 400 2020	2020	7700 Km	first owner	bangalore	\n\n	20.7 bhp	220000	400
4437	Benelli TNT 600 GT 2018	2018	26500 Km	first owner	hyderabad	\n\n 19 kmpl	82.70 bhp	515000	600
4546	Benelli TNT 25 250cc 2016	2016	25000 Km	second owner	pune	\n\n 25 kmpl	28.16 bhp	106000	250
4630	Benelli 302R 300CC 2019	2019	2500 Km	first owner	bangalore	\n\n	38.26 bhp	310000	300
4700	Benelli TRK 502 2019	2019	3250 Km	first owner	bangalore	\n\n	47 bhp	500000	502

4748	Benelli TNT 300 2017	2017	10000 Km	first owner	mumbai	\n\n25 kmpl	37.73 bhp	230000	300
4997	Benelli TNT 300 2016	2016	6400 Km	second owner	pune	\n\n25 kmpl	37.73 bhp	200000	300
4998	Benelli TNT 300 2016	2016	32000 Km	first owner	mumbai	\n\n25 kmpl	37.73 bhp	218100	300
5059	Benelli TNT 300 2017	2017	9500 Km	first owner	chennai	\n\n25 kmpl	37.73 bhp	215000	300
5177	Benelli Imperiale 400 2020	2020	1900 Km	first owner	secunderabad	\n\n	20.7 bhp	240000	400
53	Benelli TNT 25	2018	14000	first	chennai	\n\n25	28.16	170	25

05	250cc 2018		Km	owner		kmpl	bhp	000	0
5383	Benelli TNT 899 2018	2018	4500 Km	first owner	delhi	\n\n 17 kmpl	119.37 bhp	700000	899
5509	Benelli TNT 300 2015	2015	21000 Km	first owner	kolkata	\n\n 25 kmpl	37.73 bhp	210000	300
5683	Benelli TNT 300 2017	2017	11000 Km	first owner	bhubaneshwar	\n\n 25 kmpl	37.73 bhp	260000	300
5718	Benelli TNT 300 2017	2017	411 Km	first owner	kolkata	\n\n 25 kmpl	37.73 bhp	295000	300
5722	Benelli TNT 300 2017	2017	5000 Km	first owner	delhi	\n\n 25 kmpl	37.73 bhp	275000	300

5728	Benelli TNT 300 ABS 2019	2019	10700 Km	first owner	hyderabad	\n\n25 kmpl	37.73 bhp	280000	300
6594	Benelli TNT 600i ABS 2018	2018	11500 Km	second owner	mumbai	\n\n19 kmpl	82.70 bhp	475000	600
6636	Benelli TNT 600 GT 2018	2018	10800 Km	first owner	ahmedabad	\n\n19 kmpl	82.70 bhp	425000	600
6777	Benelli TNT 300 i 300cc 2016	2016	7500 Km	first owner	mumbai	\n\n25 kmpl	37.73 bhp	270000	300
6781	Benelli TNT 25 250 cc 2016	2016	6300 Km	first owner	lonavala	\n\n	NaN	140000	250
67	Benelli TNT 300	2017	11700	second	mumbai	\n\n25	37.73	280	30

83	2017		Km	owner		kmpl	bhp	000	0
6788	Benelli TNT 300 2016	2016	13000 Km	second owner	bardhaman	\n\n25 kmpl	37.73 bhp	220000	300
6791	Benelli TNT 600i ABS 2019	2019	1000 Km	first owner	kolkata	\n\n19 kmpl	82.70 bhp	650000	600
7023	Benelli 302R 2017	2017	4130 Km	first owner	mumbai	\n\n	38.26 bhp	300000	302

Extracing Brand from model name

In [12]:

```
brands = [i.split()[0] for i in data.model_name]
data['brand'] = brands
```

In [13]:

```
data
```

Out[13]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc	brand
0	Bajaj Avenger Cruise 220 2017	2017	17000 Km	first owner	hyderabad	\n\n 35 kmpl	19 bhp	63500	220	Bajaj
1	Royal Enfield Classic 350cc 2016	2016	50000 Km	first owner	hyderabad	\n\n 35 kmpl	19.80 bhp	115000	350	Royal
2	Hyosung GT250R 2012	2012	14795 Km	first owner	hyderabad	\n\n 30 kmpl	28 bhp	300000	250	Hyosung
3	Bajaj Dominar 400 ABS 2017	2017	Mileage 28 Kms	first owner	pondicherry	\n\n 28 Kms	34.50 bhp	100000	400	Bajaj

4	Jawa Perak 330cc 2020	2020	2000 Km	first owner	bangalore	\n\n	30 bhp	197500	330	Jawa
...	...	...	...	...	...	...	...	...	...	...
7852	Yamaha YZF-R15 150cc 2011	2011	7000 Km	first owner	agra	\n\n42 kmpl	16 bhp	55000	150	Yamaha
7853	Bajaj Discover 100cc 2015	2015	Mileage 80 Kmpl	first owner	delhi	\n\n80 Kmpl	7.7	28000	100	Bajaj
7854	Bajaj Pulsar 180cc 2016	2016	6407 Km	first owner	bangalore	\n\n65 kmpl	17 bhp	61740	180	Bajaj
7855	Bajaj V15 150cc 2016	2016	7524 Km	first owner	bangalore	\n\n57 kmpl	11.80 bhp	49000	150	Bajaj

7856	Bajaj Pulsar 220cc 2016	2016	15000 Km	first owner	chennai	\n\n38 kmpl	21 bhp	65000	220	Bajaj

7857 rows × 10 columns

In [14]:

```
data.replace('Ideal',"Jawa",inplace=True)
data.replace("yamaha","Yamaha",inplace=True)
```

Preprocessing Mileage

In [15]:

```
mil = data.mileage.to_list()
mil = [str(i).lower().replace('kmpl','').split("-")[0] for i in
mil ]
mil = [str(i).lower().replace('kms','') for i in mil ]
mil = [str(i).replace('\n','') for i in mil ]
mil = [str(i).strip() for i in mil]
data.mileage = mil
```

In [16]:

```
data[data.mileage == 'liquid cooled']
```

Out[16]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc	brand
157	Jawa Forty Two 295CC 2019	2019	4930 Km	first owner	moradabad	liquid cooled	27 bhp	140000	295	Jawa
211	Jawa Forty Two 295CC Dual ABS BS6 2020	2020	5000 Km	first owner	pune	liquid cooled	27 bhp	190000	295	Jawa
300	Jawa Forty Two 295CC 2019	2019	600 Km	first owner	ludhiana	liquid cooled	27 bhp	162000	295	Jawa
347	Jawa Forty Two 295CC 2019	2019	3200 Km	first owner	ghaziabad	liquid cooled	27 bhp	160000	295	Jawa

549	Jawa Forty Two Dual ABS 295CC 2020	2020	419 Km	first owner	chennai	liquid cooled	27 bhp	165000	295	Jawa
656	Jawa Forty Two 295CC 2019	2019	1800 Km	first owner	nagpur	liquid cooled	27 bhp	142000	295	Jawa
668	Jawa Standard 295CC ABS BS6 2020	2020	3000 Km	first owner	rajkot	liquid cooled	27 bhp	167000	295	Jawa
699	Jawa Standard 295CC Dual ABS BS6 2020	2020	2700 Km	first owner	mumbai	liquid cooled	27 bhp	160000	295	Jawa
775	Jawa Forty Two Dual ABS 295CC 2020	2020	1700 Km	first owner	allahabad	liquid cooled	27 bhp	220000	295	Jawa
81	Jawa Forty Two	2019	11500	first owne	madur	liquid coole	27 bh	170	29	Ja

5	295CC 2019		Km	r	ai	d	p	000	5	wa
1096	Jawa Forty Two 295CC 2019	2019	10500 Km	first owner	mumbai	liquid cooled	27 bhp	150000	295	Jawa
1219	Jawa Standard 295CC 2020	2020	1003 Km	first owner	hyderabad	liquid cooled	27 bhp	175000	295	Jawa
1307	Jawa Forty Two 295CC 2019	2019	2450 Km	first owner	pune	liquid cooled	27 bhp	150000	295	Jawa
1319	Jawa Forty Two 295CC 2019	2019	7541 Km	first owner	delhi	liquid cooled	27 bhp	144000	295	Jawa
1545	Jawa Forty Two 295CC 2019	2019	1600 Km	first owner	mumbai	liquid cooled	27 bhp	155625	295	Jawa

1884	Jawa Forty Two 295CC 2019	2019	3800 Km	first owner	panvel	liquid cooled	27 bhp	138750	295	Jawa
2021	Jawa Standard 295CC 2019	2019	2400 Km	first owner	delhi	liquid cooled	27 bhp	150000	295	Jawa
2565	Jawa Forty Two 295CC 2019	2019	47000 Km	second owner	hyderabad	liquid cooled	27 bhp	160000	295	Jawa
2567	Jawa Forty Two 295CC 2019	2019	4600 Km	first owner	hyderabad	liquid cooled	27 bhp	155000	295	Jawa
3003	Jawa Forty Two 295CC Dual ABS BS6 2020	2020	1100 Km	first owner	delhi	liquid cooled	27 bhp	165000	295	Jawa

3004	Jawa Forty Two 295CC Dual ABS BS6 2021	2021	700 Km	first owner	delhi	liquid cooled	27 bhp	180000	295	Jawa
3022	Jawa Standard 295CC 2020	2020	800 Km	first owner	delhi	liquid cooled	27 bhp	190000	295	Jawa
3204	Jawa Forty Two 295CC 2019	2019	7500 Km	first owner	jaipur	liquid cooled	27 bhp	138300	295	Jawa
3725	Jawa Standard 295CC 2020	2020	8280 Km	first owner	ajmer	liquid cooled	27 bhp	164400	295	Jawa
3811	Jawa Forty Two 295CC 2019	2019	10000 Km	second owner	bangalore	liquid cooled	27 bhp	150000	295	Jawa

3834	Jawa Forty Two 295CC Dual ABS BS6 2020	2020	5500 Km	first owner	mohali	liquid cooled	27 bhp	180000	295	Jawa
4282	Jawa Forty Two 295CC 2020	2020	8000 Km	first owner	bangalore	liquid cooled	27 bhp	165000	295	Jawa
4347	Jawa Forty Two 295CC 2019	2019	15451 Km	first owner	chennai	liquid cooled	27 bhp	139000	295	Jawa
4383	Jawa Forty Two 295CC 2020	2020	1100 Km	first owner	ahmedabad	liquid cooled	27 bhp	123750	295	Jawa
4483	Jawa Standard 295CC Dual ABS BS6 2021	2021	1000 Km	first owner	indore	liquid cooled	27 bhp	190000	295	Jawa
45	Jawa Forty Two	2020	9600	first owne	mumb	liquid coole	27 bh	150	29	Ja

68	295CC 2020		Km	r	ai	d	p	437	5	wa
4911	Jawa Forty Two 295CC 2019	2019	2100 Km	first owner	hosur	liquid cooled	27 bhp	210000	295	Jawa
5057	Jawa Forty Two 295CC 2019	2019	2388 Km	first owner	chennai	liquid cooled	27 bhp	175000	295	Jawa
5406	Jawa Forty Two 295CC 2019	2019	5400 Km	first owner	bangalore	liquid cooled	27 bhp	168000	295	Jawa
5823	Jawa Forty Two 295CC 2019	2019	2000 Km	first owner	ghaziabad	liquid cooled	27 bhp	155000	295	Jawa
5994	Jawa Forty Two 295CC 2019	2019	245 Km	first owner	faridabad	liquid cooled	27 bhp	137800	295	Jawa

Replacing wrong values with their correct ones.

In [17]:

```
data.replace('liquid cooled','37.6',inplace=True)
```

In [18]:

```
np.unique(data.mileage)
```

Out[18]:

```
array(['', '104', '12', '12.5', '13', '14', '15', '16', '17',
'17.85',
       '18', '18.86', '19', '20', '20.3', '20.40', '21', '22',
'23', '25',
       '26', '27', '28', '29', '30', '31.85', '32', '35', '37',
'37.6',
       '38', '38.5', '39.1', '39.4', '40', '42', '43', '45',
'45.8',
       '46.40', '48', '5', '50', '52', '53', '53.72', '54',
'55', '55.47',
       '56', '57', '58', '59', '60', '62', '63', '63.97', '64',
'65',
       '67', '68', '69', '70', '71', '72', '74', '75', '77',
'80', '81',
```

```
       '82', '82.4', '83', '84', '85', '89', '90', '95',
'96.9', 'nan'],

      dtype=object)
```

In [19]:

```
data[data.mileage == 'nan']
```

Out[19]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc	brand
144	TVS Apache RTR 160cc 2014	2014	20372 Km	first owner	delhi	nan	15.2 bhp @ 8500 rpm	42000	160	TVS
193	Hero Karizma R 223cc 2010	2010	70000 Km	first owner	noida	nan	17 bhp @ 7,000 rpm	19215	223	Hero

422	TVS Apache 150cc 2007	2007	36486 Km	second owner	bathinda	nan	13.7 Bhp @ 8500 rpm	19500	150	TVS
1376	KTM RC 390cc 2015	2015	28000 Km	second owner	hyderabad	nan	43 bhp @ 9,000 rpm	130000	390	KTM
1488	Royal Enfield Thunderbird 350cc 2015	2015	24000 Km	first owner	bangalore	nan	19.80 bhp @ 5,250 rpm	115000	350	Royal
1640	Kawasaki Ninja 650cc 2020	2020	8000 Km	second owner	delhi	nan	66.4 bhp @ 8,000 rpm	520000	650	Kawasaki
2160	Harley-Davidson Street	2019	600 Km	first own	bangalore	nan	70 bhp	700700	74	Harley-David

	Rod 2019			er					9	son
2277	KTM Duke 200cc 2012	2012	30000 Km	first owner	bangalore	nan	25 bhp @ 10,000 rpm	85000	200	KTM
4646	Hero Passion Pro 100cc 2014	2014	16834 Km	first owner	chandigarh	nan	8.2 Bhp @ 8000 rpm	40000	100	Hero
4655	Bajaj Pulsar 180cc 2013	2013	82000 Km	first owner	chandigarh	nan	15 bhp @ 9,000 rpm	40000	180	Bajaj
7394	Yamaha YZF-R15 2.0 150cc 2015	2015	18000 Km	second owner	delhi	nan	16.70 bhp @ 8,500 rpm	62000	150	Yamaha

Mileage values taken from internet

In [20]:

```
data.iloc[144,data.columns.get_loc('mileage')] = "54"
data.iloc[193,data.columns.get_loc('mileage')] = "45"
data.iloc[422,data.columns.get_loc('mileage')] = "62"
data.iloc[1376,data.columns.get_loc('mileage')] = "25"
data.iloc[1488,data.columns.get_loc('mileage')] = "40"
data.iloc[1640,data.columns.get_loc('mileage')] = "25"
data.iloc[2160,data.columns.get_loc('mileage')] = "18"
data.iloc[2277,data.columns.get_loc('mileage')] = "35"
data.iloc[4646,data.columns.get_loc('mileage')] = "65"
data.iloc[4655,data.columns.get_loc('mileage')] = "45"
data.iloc[7394,data.columns.get_loc('mileage')] = "40"
```

Few entires have just an empty string as the input. Replace them to 0 and further modify it as teh average mileage of that brand bikes.

In [21]:

```
data.replace('','0',inplace=True)
```

In [22]:

```
null_brands = np.unique(data[data.mileage == '0'].brand)
for brand in null_brands:
    values = data[data.brand == brand].mileage.to_numpy()
    values = np.delete(values,np.where(values == '0'))
```

```
    if(len(values) == 0):
        print(brand)
        mean=0
    else:
        mean = values.astype("float32").mean()

    for index,rows in data[data.brand == brand].iterrows():
        if data.iloc[index,data.columns.get_loc("mileage")] ==
'0':
            data.iloc[index,data.columns.get_loc("mileage")] =
mean
```

```
BMW
Fb
Yazdi
```

Few values have no mileage given to taken mean, so input them manually - values taken from internet.

In [23]:

```
data.iloc[2023,data.columns.get_loc("mileage")] = "30"
data.iloc[2832,data.columns.get_loc("mileage")] = "30"
data.iloc[3400,data.columns.get_loc("mileage")] = "30"
data.iloc[3827,data.columns.get_loc("mileage")] = "30"
```

```
data.iloc[4154,data.columns.get_loc("mileage")] = "30"
data.iloc[4392,data.columns.get_loc("mileage")] = "30"
data.iloc[5435,data.columns.get_loc("mileage")] = "30"
data.iloc[5586,data.columns.get_loc("mileage")] = "17"
data.iloc[5727,data.columns.get_loc("mileage")] = "30"
data.iloc[6561,data.columns.get_loc("mileage")] = "30"
```

In [24]:

```
data.iloc[3773,data.columns.get_loc("mileage")] = "26"
data.iloc[5031,data.columns.get_loc("mileage")] = "26"
```

In [25]:

```
data.iloc[5584,data.columns.get_loc("mileage")] = "35"
```

Preprocessing Location

Replace missing location entries with mode values

In [26]:

```
for index,rows in data[data.location.isna()].iterrows():
    data.iloc[index,data.columns.get_loc("location")] =
data[data.brand == rows.brand].dropna().location.mode()
```

Preprocessing Vehicle Power

In [27]:

```
val=  data.power.isna()
np.where(val == True)
```

Out[27]:

```
(array([ 135, 1627, 1691, 1716, 2442, 2487, 2638, 2659, 2714,
2819, 3088,
        3977, 4393, 4425, 4591, 4760, 5626, 5673, 5694, 5788,
5950, 6155,

        6321, 6781, 6853, 7306, 7442, 7513, 7783, 7796,
7801]),)
```

Few power entries does not have any value - enter them manually using values from internet.

In [28]:

```
data.iloc[135,data.columns.get_loc("power")] = "28"
data.iloc[1627,data.columns.get_loc("power")] = "14"
data.iloc[1691,data.columns.get_loc("power")] = "18"
data.iloc[1716,data.columns.get_loc("power")] = "18"
data.iloc[2442,data.columns.get_loc("power")] = "87"
data.iloc[2487,data.columns.get_loc("power")] = "18"
data.iloc[2638,data.columns.get_loc("power")] = "16"
```

```
data.iloc[2659,data.columns.get_loc("power")] = "64"
data.iloc[2714,data.columns.get_loc("power")] = "18"
data.iloc[2819,data.columns.get_loc("power")] = "87"
data.iloc[3088,data.columns.get_loc("power")] = "18"
data.iloc[3977,data.columns.get_loc("power")] = "37.5"
data.iloc[4393,data.columns.get_loc("power")] = "18"
data.iloc[4425,data.columns.get_loc("power")] = "111"
data.iloc[4591,data.columns.get_loc("power")] = "75.1"
data.iloc[4760,data.columns.get_loc("power")] = "18"
data.iloc[5626,data.columns.get_loc("power")] = "65"
data.iloc[5673,data.columns.get_loc("power")] = "89.2"
data.iloc[5694,data.columns.get_loc("power")] = "64"
data.iloc[5788,data.columns.get_loc("power")] = "11"
data.iloc[5950,data.columns.get_loc("power")] = "26.21"
data.iloc[6155,data.columns.get_loc("power")] = "15.2"
data.iloc[6321,data.columns.get_loc("power")] = "26.21"
data.iloc[6781,data.columns.get_loc("power")] = "28.15"
data.iloc[6853,data.columns.get_loc("power")] = "15.2"
data.iloc[7306,data.columns.get_loc("power")] = "10.70"
data.iloc[7442,data.columns.get_loc("power")] = "7.4"
data.iloc[7513,data.columns.get_loc("power")] = "23.17"
data.iloc[7783,data.columns.get_loc("power")] = "24.13"
data.iloc[7796,data.columns.get_loc("power")] = "7.4"
data.iloc[7801,data.columns.get_loc("power")] = "8.24"
```

Resolve inconsistent naming conventions in power column by removing character values like units -bhp, kw, ps etc. Similarly, few entries are in different units - Kw and PS, convert them to bhp for consistency

In [29]:

```
for index,rows in data.iterrows():
    if "bhp" in
str(data.iloc[index,data.columns.get_loc("power")]).lower():
        data.iloc[index,data.columns.get_loc("power")] =
data.iloc[index,data.columns.get_loc("power")].replace("bhp",''
)
    if "hp" in
str(data.iloc[index,data.columns.get_loc("power")]).lower():
        data.iloc[index,data.columns.get_loc("power")] =
float(data.iloc[index,data.columns.get_loc("power")].split("
")[0].replace("hp","").split("-")[0]) * 0.986
    if "kw" in
str(data.iloc[index,data.columns.get_loc("power")]).lower():
        data.iloc[index,data.columns.get_loc("power")] =
float(data.iloc[index,data.columns.get_loc("power")].split("
")[0].lower().replace("kw","")) * 1.341
    if "ps" in
str(data.iloc[index,data.columns.get_loc("power")]).lower():
        data.iloc[index,data.columns.get_loc("power")] =
```

```
float(data.iloc[index,data.columns.get_loc("power")].split("
")[0].lower().replace("ps",""))) * 0.99
    if "@" in
str(data.iloc[index,data.columns.get_loc("power")]).lower():
        data.iloc[index,data.columns.get_loc("power")] =
data.iloc[index,data.columns.get_loc("power")].split("@")[0].st
rip()
```

In [30]:

```
data.isna().sum()
```

Out[30]:

```
model_name    0
model_year    0
kms_driven    0
owner         0
location      0
mileage       0
power         0
price         0
cc            0
brand         0
```

```
dtype: int64
```

Preprocessing KMS driven

```
In [31]:
data
```

Out[31]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price	cc	brand
0	Bajaj Avenger Cruise 220 2017	2017	17000 Km	first owner	hyderabad	35	19	63500	220	Bajaj
1	Royal Enfield Classic 350cc 2016	2016	50000 Km	first owner	hyderabad	35	19.80	115000	350	Royal
2	Hyosung GT250R 2012	2012	14795 Km	first own	hyderabad	30	28	300000	25	Hyosung

				er					0	
3	Bajaj Dominar 400 ABS 2017	2017	Mileage 28 Kms	first owner	pondicherry	28	34.50	100000	400	Bajaj
4	Jawa Perak 330cc 2020	2020	2000 Km	first owner	bangalore	37.599995	30	197500	330	Jawa
...	...	...	...	...	...	...	...	...	...	...
7852	Yamaha YZF-R15 150cc 2011	2011	7000 Km	first owner	agra	42	16	55000	150	Yamaha
7853	Bajaj Discover 100cc 2015	2015	Mileage 80 Kmpl	first owner	delhi	80	7.7	28000	100	Bajaj
78	Bajaj Pulsar	2016	6407	first own	banga	65	17	617	18	Baja

54	180cc 2016		Km	er	lore			40	0	j
7855	Bajaj V15 150cc 2016	2016	7524 Km	first owner	bangalore	57	11.80	49000	150	Bajaj
7856	Bajaj Pulsar 220cc 2016	2016	15000 Km	first owner	chennai	38	21	65000	220	Bajaj

7857 rows × 10 columns

Distance driven is represented as a string with units. Remove the units and keep only the distance value.

In [32]:

```
kms = data.kms_driven.to_numpy()
for i in range(len(kms)):
    kms[i] = kms[i].split(" ")[0]
```

There are inconsistent entries within the kms_drive column - few are given as mileage value and otheres are just "yes" strinngs. Replace them with average distance driven for a standard used bike.

In [33]:

```
for i in range(len(kms)):
    if str(kms[i]).lower() in ["mileage","yes"]:
        kms[i] = np.nan
data['kms_driven'] = kms
mean =
kms[~np.isnan(kms.astype('float'))].astype('float').mean()
data.kms_driven.fillna(value = mean,inplace=True)
```

Fixing Data Types

Depending on column, cast each one to respective data type like float, int or string.

In [34]:

```
data.isna().sum()
```

Out[34]:

```
model_name    0
model_year    0
kms_driven    0
owner         0
location      0
mileage       0
power         0
price         0
```

```
cc             0
brand          0

dtype: int64
```

In [35]:

```
data = data.astype({"model_name" : str,
                    "model_year" : int,
                    "kms_driven" : int,
                    "owner"      : str,
                    "location"   : str,
                    "mileage"    : float,
                    "power"      : float,
                    "price"      : int,
                    "cc"         : int,
                    "brand"      : str})
```

In [36]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7857 entries, 0 to 7856
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
```

```
---  ------      --------------  -----
 0   model_name  7857 non-null   object
 1   model_year  7857 non-null   int64
 2   kms_driven  7857 non-null   int64
 3   owner       7857 non-null   object
 4   location    7857 non-null   object
 5   mileage     7857 non-null   float64
 6   power       7857 non-null   float64
 7   price       7857 non-null   int64
 8   cc          7857 non-null   int64
 9   brand       7857 non-null   object
dtypes: float64(2), int64(4), object(4)
memory usage: 614.0+ KB
```

Exploratory Data Analysis

Statistics

In [37]:

```
pd.set_option("display.float",str)
data.describe().drop(["model_year"],axis=1).round(2)
```

Out[37]:

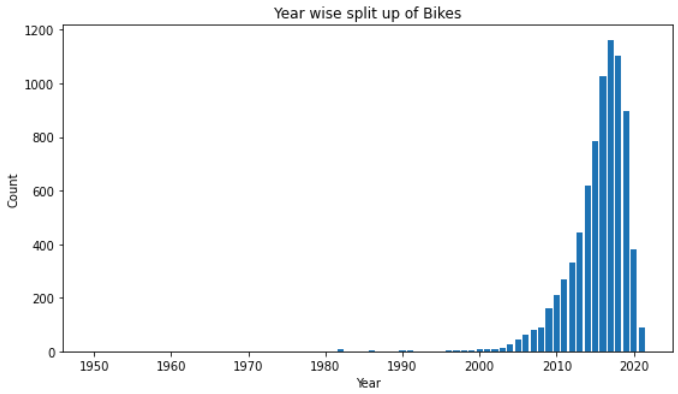
	kms_driven	mileage	power	price	cc
count	7857.0	7857.0	7857.0	7857.0	7857.0
mean	23090.09	44.68	20.83	106791.34	254.08
std	24813.44	16.42	15.09	138926.12	182.84
min	0.0	5.0	7.0	0.0	97.0
25%	11204.0	35.0	14.0	42000.0	150.0
50%	23090.0	40.0	19.0	75000.0	200.0

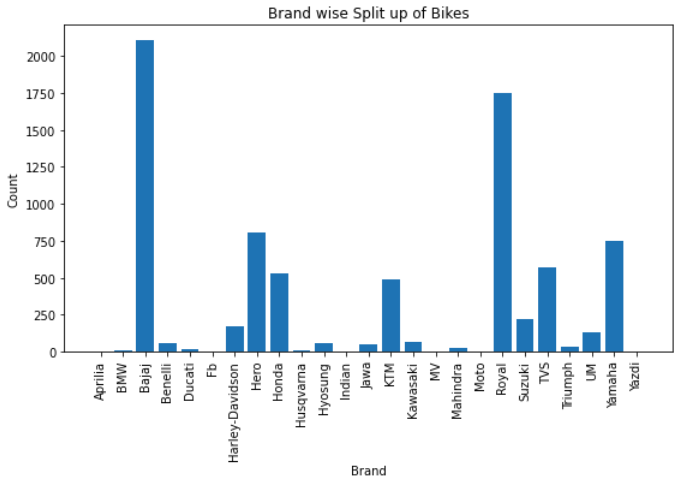
75%	25000.0	55.0	23.86	125000.0	350.0
max	1000000.0	104.0	197.3	3000000.0	2458.0

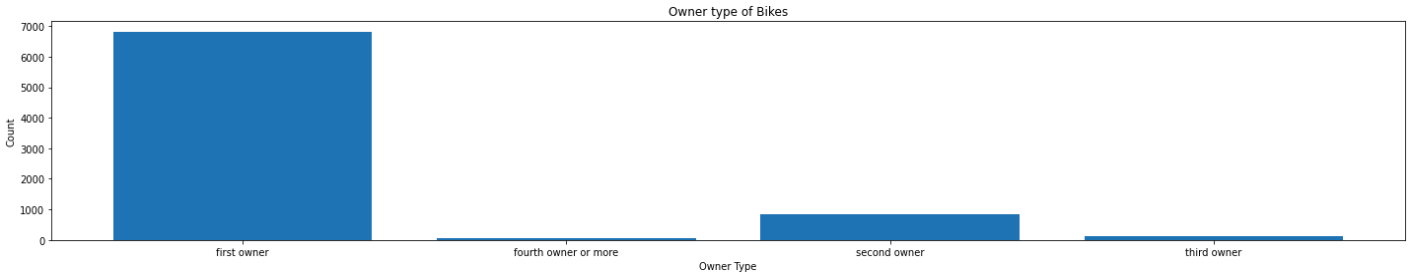
In [38]:

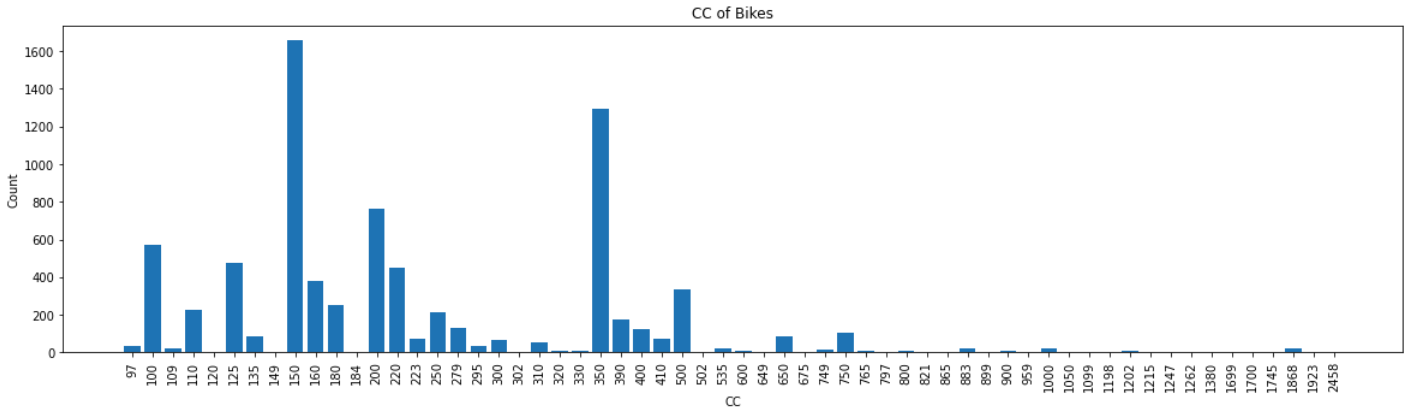
```
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
x,count = np.unique(data.model_year,return_counts=True)
plt.bar(x,count)
plt.title("Year wise split up of Bikes")
plt.xlabel("Year")
plt.ylabel("Count")

plt.subplot(1,2,2)
x,count = np.unique(data.brand,return_counts=True)
plt.bar(x,count)
plt.title("Brand wise Split up of Bikes")
plt.xlabel("Brand")
plt.xticks(rotation=90)
plt.ylabel("Count")
```

```
plt.figure(figsize=(20,4))
x,count = np.unique(data.owner,return_counts=True)
plt.bar(x,count)
plt.title("Owner type of Bikes")
plt.xlabel("Owner Type")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

plt.figure(figsize=(20,5))
x,count = np.unique(data.cc,return_counts=True)
plt.bar([str(i) for i in x],count)
plt.title("CC of Bikes")
plt.xlabel("CC")
plt.xticks(rotation=90)
plt.ylabel("Count")
plt.show()
```









linkcode

Inference : The year wise split up shows a skewed graph with the number of models rising from the year 2000 with 2015 being the highest and the number getting floored from there onwards until 2020. Brands belonging to Bajaj and Royal Enfield are the highest in stock followed by Yamaha and Hero. The majority of bikes belong to First Owner which means it has been owned only by one customer. 150 cc and 350 cc

bikes are the highest in number followed by other common capacities like 200 cc and below. 500 cc stands out in the mid range power. Superbikes (1000 cc and above) are very less in count comparatively.

Reference link