Car Price Prediction Project

Submitted by:

AKSHAY DINESH SHAH

ACKNOWLEDGMENT

I sincerely thank Zep Analytics for giving an opportunity to work on this project.

Thanks to cars24 for allowing their data to be used in this study.

INTRODUCTION

Business Problem Framing

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand , making them costly and some are not in demand, hence cheaper. The price of the car varies depending on many factors and even the car’s resale price. This study intends to build a machine learning model that predicts the resale prices of cars. Since, the sales of cars coming back to pace, this model helps the traders to valuate the resale price offered and make a decision. This model is built with therecently collected data from the popular cities of India.

Conceptual Background of the Domain Problem

- The title of the project comes under the domain of “Automobile Industry”.
- The main aim of this project is to analyse the data about the used cars based on different features and to give a detailed analysis by visualizing them.
- This analysis will be helpful for the customers who are planning to buy a used car based on different features.

Review of Literature

Before starting this project, I have enough knowledge of cars and different categories of possible features and it should be helpful to know the correlation Between different features. Studying different factors affecting the price of used cars. Going through various articles, we understand that the price of used cars are based various features available, age of car and many more features.

Motivation for the Problem Undertaken

Now a days used cars are very common it is important to know that what brand and features of car is giving weightage to the price of the car. It will be helpful for both the clients and traders to understand the whole scenario.

Analytical Problem Framing

❖ Mathematical/ Analytical Modelling of the Problem

- The website used in this project to scrape the data is Cars24
- Used cars data extracted from following 5 cities:
- 1) Hyderabad
- 2) Delhi
- 3) Mumbai
- 4) Bangalore
- 5) Chennai
  - Collecting the URL’s of used cars in different cities. (Hyderabad, Delhi, Mumbai, Bangalore, Chennai)
  - Since the data loads upon scrolling the website, Selenium web driver is used to scroll till bottom of the page and to store the html code after manipulating the page source.
  - After saving the html file, load the file and parse the file using Beautiful Soup.
  - After scraping the data, saving the data in a csv file.

Data Sources and their formats

I have collected data from cars24.The dataset consists of following features,

1) Brand name - To analyse the data based on brands.

2) Model name - To analyse based on different model.

3) Price of the car - To analyse based on price

4) Model Year - To analyse based on model year

5) Location - To analyse car details based on location

6) Fuel Type - Petrol/Diesel/LPG/CNG/Electric

7) Driven (Kms) - Analyse based on distance travelled by car

8) Gear Type - Automatic/Manual car

9) Ownership - Either the car is 1st hand or 2nd and soon

10) EMI (Monthly) - Describes the EMI price for different brands & models

Collecting the URL’s of used cars in different cities. (Hyderabad, Delhi, Mumbai, Bangalore, Chennai)

Since the data loads upon scrolling the website, Selenium web driver is used to scroll till bottom of the page and to store the html code after manipulating the page source.

After saving the html file, load the file and parse the file using Beautiful Soup. ϖ After scraping the data, saving the data in a csv file.

Data Preprocessing Done

- Cleaning poorly formatted columns from the extracted raw data.
- Skipping rows with Nan values.
- Removing duplicated rows. ϖ Changing the data types for necessary columns.
- Filtering the Data Frame and storing it to a csv file

Pre-processing Pipeline is an important step towards the data modelling, to make data ready for prediction. Following Steps, I followed for Pre-Processing:

1. Making data in the same format for each column before combining the data into single dataset.
2. Changing the datatype of the columns or correcting the data types of the column.
3. Finding null values in the features and treat them accordingly.
4. Encoding of the data.
5. Scaling the data.

Data Inputs- Logic- Output Relationships

In this case the target variable is continuous, and two Variables are numeric except that:

1.Year and driven(km). Where Year is negatively correlated, and Km is positively correlated with the price of the cars.

2. Increasing *model year (age of car),* the *price* decreases.

3.fuel type is positively correlated to price

4.Gear is also positively correlated to price

Hardware and Software Requirements and Tools Used

Tool:

- Jupyter Notebook 6.1.4
- Selenium – To manipulate the HTML code
- Beautiful Soup – HTML Parser
- Pandas, NumPy – To analyze the data
- Matplotlib, Seaborn – To visualize the data

Model/s Development and Evaluation

- **Identification of possible problem-solving approaches (methods)**
  The most time-consuming step, in this case, is to make the data in a similar format like EMI, Gear, price and features and remove unwanted values. Then also some *,'' symbols were still there I have replaced them with numeric values only. Also, in the transmissionand the brand name, I some unwanted was there that I have replaced with new categories. Label Encoding the feature to fit in the model.

Testing of Identified Approaches (Algorithms)

Following are the algorithms I have used in this problem:

i. Linear Regression.
ii. Decision Tree Regressor.
iii. Random Forest
iv. Gradient Descent
v. Adaboost Regressor
vi. XGBRegressor
vii. Support Vector Regressor

Run and evaluate selected models

Linear Regression

```
Lr = LinearRegression()
Lr.fit(X_train,y_train)
```

```
LinearRegression()
```

```
Lr.score(X_train,y_train)
```

```
1.0
```

```
pred_Lr = Lr.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_Lr))
print('MAE:',metrics.mean_absolute_error(y_test, pred_Lr))
print('MSE:',metrics.mean_squared_error(y_test, pred_Lr))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_Lr)))
```

```
R2_Score: 1.0
MAE: 2.4620267386372026e-10
MSE: 9.799637034810563e-20
RMSE: 3.1304371954745493e-10
```

DecisionTreeRegressor

```
dtc = DecisionTreeRegressor()
dtc.fit(X_train,y_train)
```

```
DecisionTreeRegressor()
```

```
dtc.score(X_train,y_train)
```

```
1.0
```

```
pred_dtc = dtc.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_dtc))
print('MAE:',metrics.mean_absolute_error(y_test, pred_dtc))
print('MSE:',metrics.mean_squared_error(y_test, pred_dtc))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_dtc)))
```

```
R2_Score: 0.9835683436467052
MAE: 2749.2105855855857
MSE: 1766944785.1362612
RMSE: 42035.04234726381
```

KNeighborsRegressor

```
knn = KNeighborsRegressor()
```

```
knn.fit(X_train,y_train)
```

```
KNeighborsRegressor()
```

```
pred_knn = knn.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_knn))
print('MAE:',metrics.mean_absolute_error(y_test, pred_knn))
print('MSE:',metrics.mean_squared_error(y_test, pred_knn))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_knn)))
```

```
R2_Score: 0.8429284982891683
MAE: 85010.69707207207
MSE: 16890364846.622747
RMSE: 129962.93643428787
```

RandomForestRegressor

```
rf = RandomForestRegressor()
```

```
rf.fit(X_train,y_train)
```

```
RandomForestRegressor()
```

```
rf.score(X_train,y_train)
```

```
0.9999714695025741
```

```
pred_rf = rf.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_rf))
print('MAE:',metrics.mean_absolute_error(y_test, pred_rf))
print('MSE:',metrics.mean_squared_error(y_test, pred_rf))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_rf)))
```

```
R2_Score: 0.9843984690010383
MAE: 2530.582207207074
MSE: 1677678941.552928
RMSE: 40959.47926369338
```

AdaBoostRegressor

```
ad = AdaBoostRegressor()
ad.fit(X_train,y_train)
```

```
AdaBoostRegressor()
```

```
ad.score(X_train,y_train)
```

```
0.9918299534724435
```

```
pred_ad = ad.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_ad))
print('MAE:',metrics.mean_absolute_error(y_test, pred_ad))
print('MSE:',metrics.mean_squared_error(y_test, pred_ad))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_ad)))
```

```
R2_Score: 0.978032756231073
MAE: 23486.08486689315
MSE: 2362202932.362304
RMSE: 48602.499239877616
```

GradientBoostingRegressor

```
grd =  GradientBoostingRegressor()
grd.fit(X_train,y_train)
```

```
GradientBoostingRegressor()
```

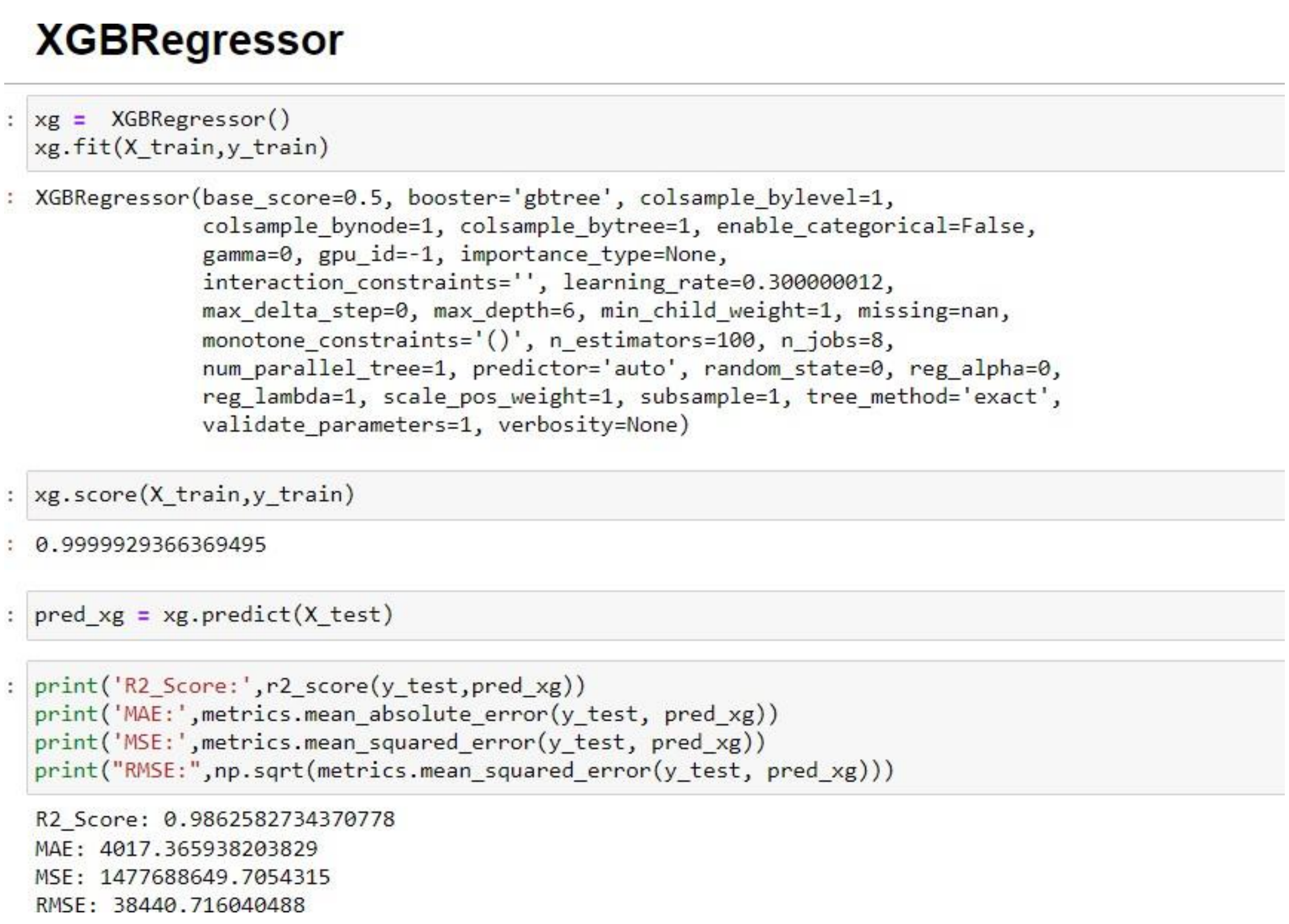
```
grd.score(X_train,y_train)
```

```
0.9999353988391735
```

```
pred_grd = grd.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_grd))
print('MAE:',metrics.mean_absolute_error(y_test, pred_grd))
print('MSE:',metrics.mean_squared_error(y_test, pred_grd))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_grd)))
```

```
R2_Score: 0.9873620759126852
MAE: 3685.7717838249187
MSE: 1358993492.8593624
RMSE: 36864.52892496203
```



XGBRegressor

```
: xg =  XGBRegressor()
  xg.fit(X_train,y_train)
```

```
: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
               colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
               gamma=0, gpu_id=-1, importance_type=None,
               interaction_constraints='', learning_rate=0.300000012,
               max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
               monotone_constraints='()', n_estimators=100, n_jobs=8,
               num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
               reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
               validate_parameters=1, verbosity=None)
```

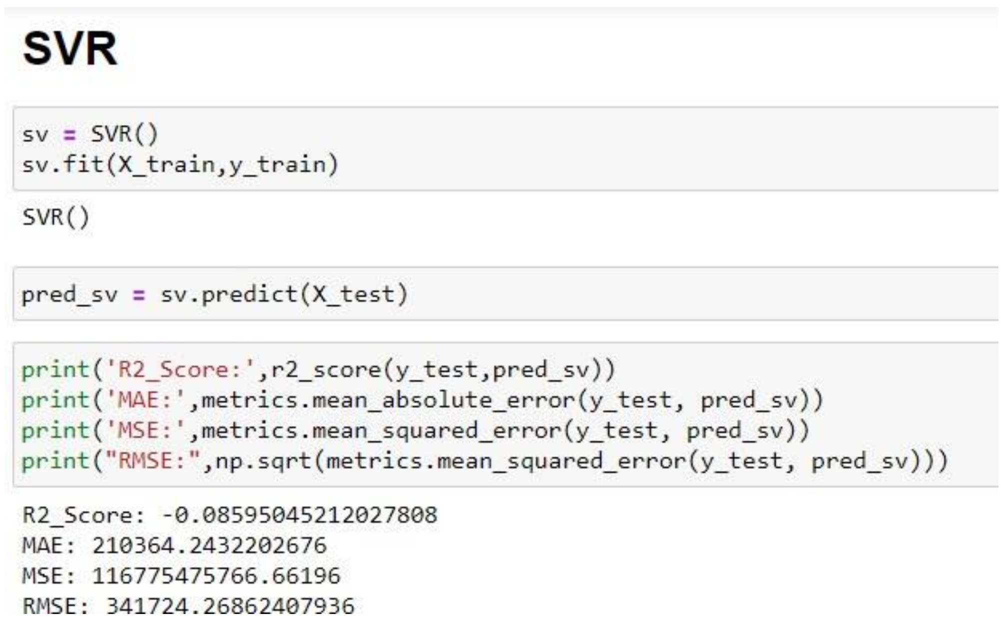
```
: xg.score(X_train,y_train)
```

```
: 0.9999929366369495
```

```
: pred_xg = xg.predict(X_test)
```

```
: print('R2_Score:',r2_score(y_test,pred_xg))
  print('MAE:',metrics.mean_absolute_error(y_test, pred_xg))
  print('MSE:',metrics.mean_squared_error(y_test, pred_xg))
  print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_xg)))
```

```
R2_Score: 0.9862582734370778
MAE: 4017.365938203829
MSE: 1477688649.7054315
RMSE: 38440.716040488
```



SVR

```
sv = SVR()
sv.fit(X_train,y_train)
```

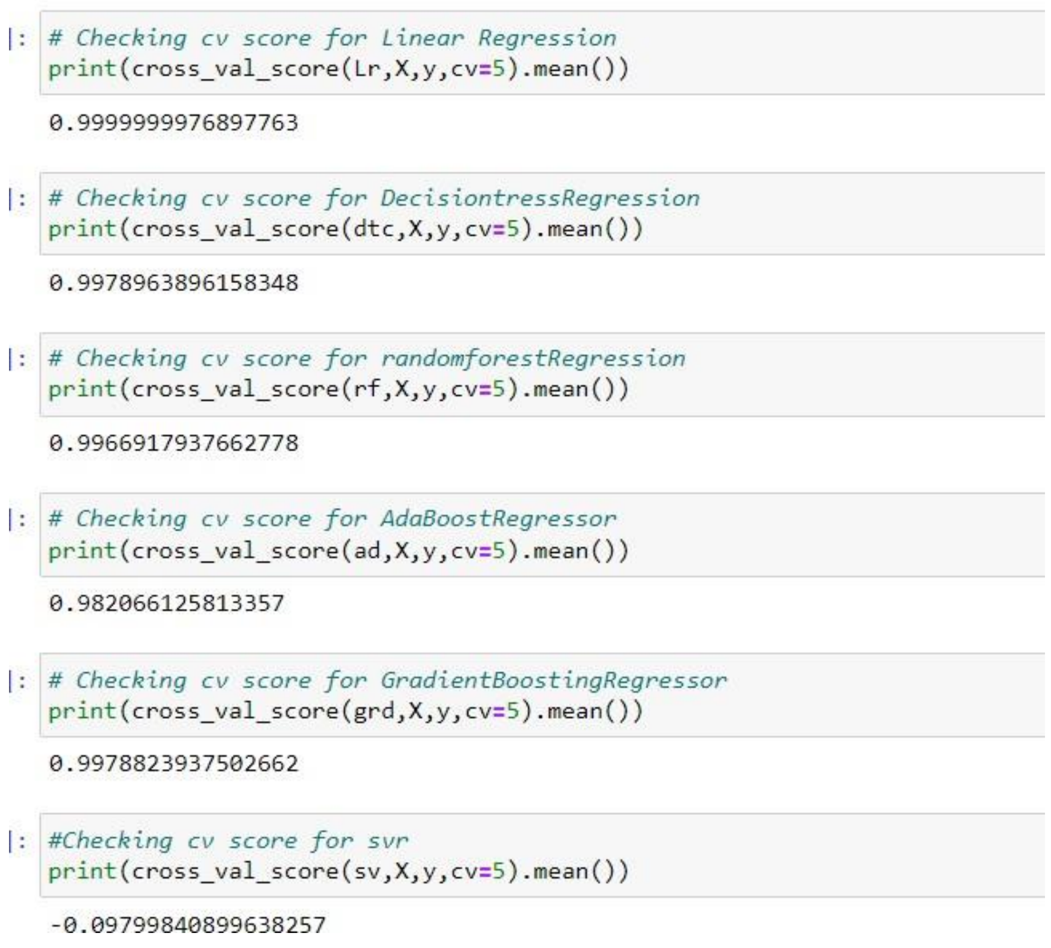
```
SVR()
```

```
pred_sv = sv.predict(X_test)
```

```
print('R2_Score:',r2_score(y_test,pred_sv))
print('MAE:',metrics.mean_absolute_error(y_test, pred_sv))
print('MSE:',metrics.mean_squared_error(y_test, pred_sv))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred_sv)))
```

```
R2_Score: -0.08595045212027808
MAE: 210364.2432202676
MSE: 116775475766.66196
RMSE: 341724.26862407936
```

Cross Validation Score



```
[ ]: # Checking cv score for Linear Regression
     print(cross_val_score(Lr,X,y,cv=5).mean())
```

```
0.9999999976897763
```

```
[ ]: # Checking cv score for DecisiontressRegression
     print(cross_val_score(dtc,X,y,cv=5).mean())
```

```
0.9978963896158348
```

```
[ ]: # Checking cv score for randomforestRegression
     print(cross_val_score(rf,X,y,cv=5).mean())
```

```
0.9966917937662778
```

```
[ ]: # Checking cv score for AdaBoostRegressor
     print(cross_val_score(ad,X,y,cv=5).mean())
```

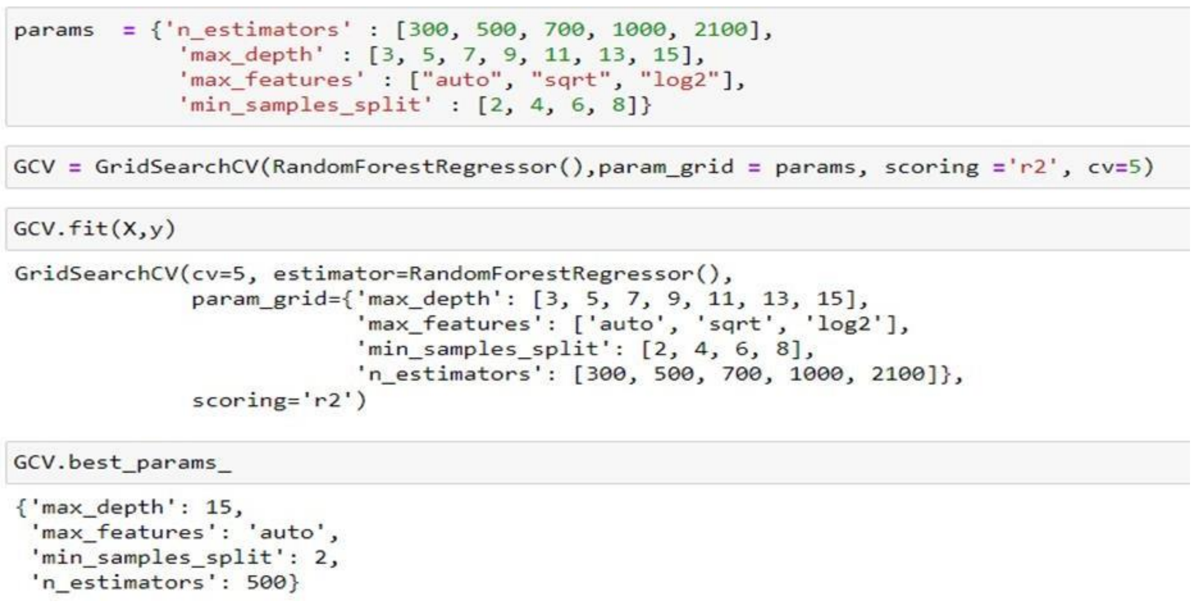
```
0.982066125813357
```

```
[ ]: # Checking cv score for GradientBoostingRegressor
     print(cross_val_score(grd,X,y,cv=5).mean())
```

```
0.9978823937502662
```

```
[ ]: #Checking cv score for svr
     print(cross_val_score(sv,X,y,cv=5).mean())
```

```
-0.09799840899638257
```



```
params  = {'n_estimators' : [300, 500, 700, 1000, 2100],
           'max_depth' : [3, 5, 7, 9, 11, 13, 15],
           'max_features' : ["auto", "sqrt", "log2"],
           'min_samples_split' : [2, 4, 6, 8]}
```

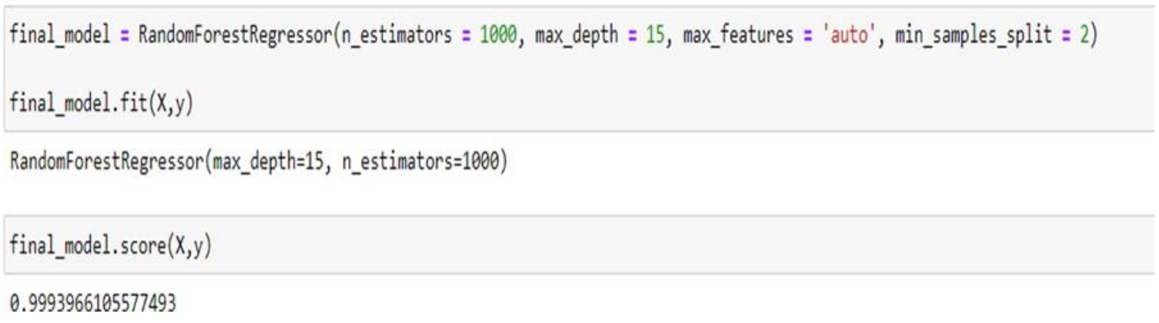
```
GCV = GridSearchCV(RandomForestRegressor(),param_grid = params, scoring ='r2', cv=5)
```

```
GCV.fit(X,y)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [3, 5, 7, 9, 11, 13, 15],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [2, 4, 6, 8],
                         'n_estimators': [300, 500, 700, 1000, 2100]},
             scoring='r2')
```

```
GCV.best_params_
```

```
{'max_depth': 15,
 'max_features': 'auto',
 'min_samples_split': 2,
 'n_estimators': 500}
```



```
final_model = RandomForestRegressor(n_estimators = 1000, max_depth = 15, max_features = 'auto', min_samples_split = 2)

final_model.fit(X,y)
```

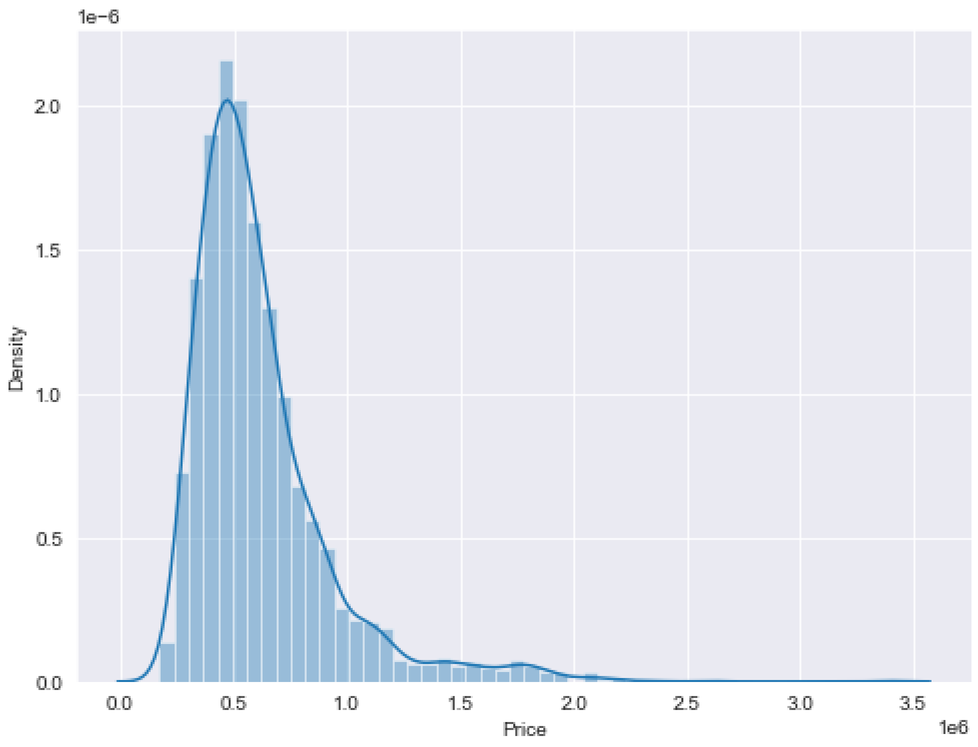
```
RandomForestRegressor(max_depth=15, n_estimators=1000)
```

```
final_model.score(X,y)
```

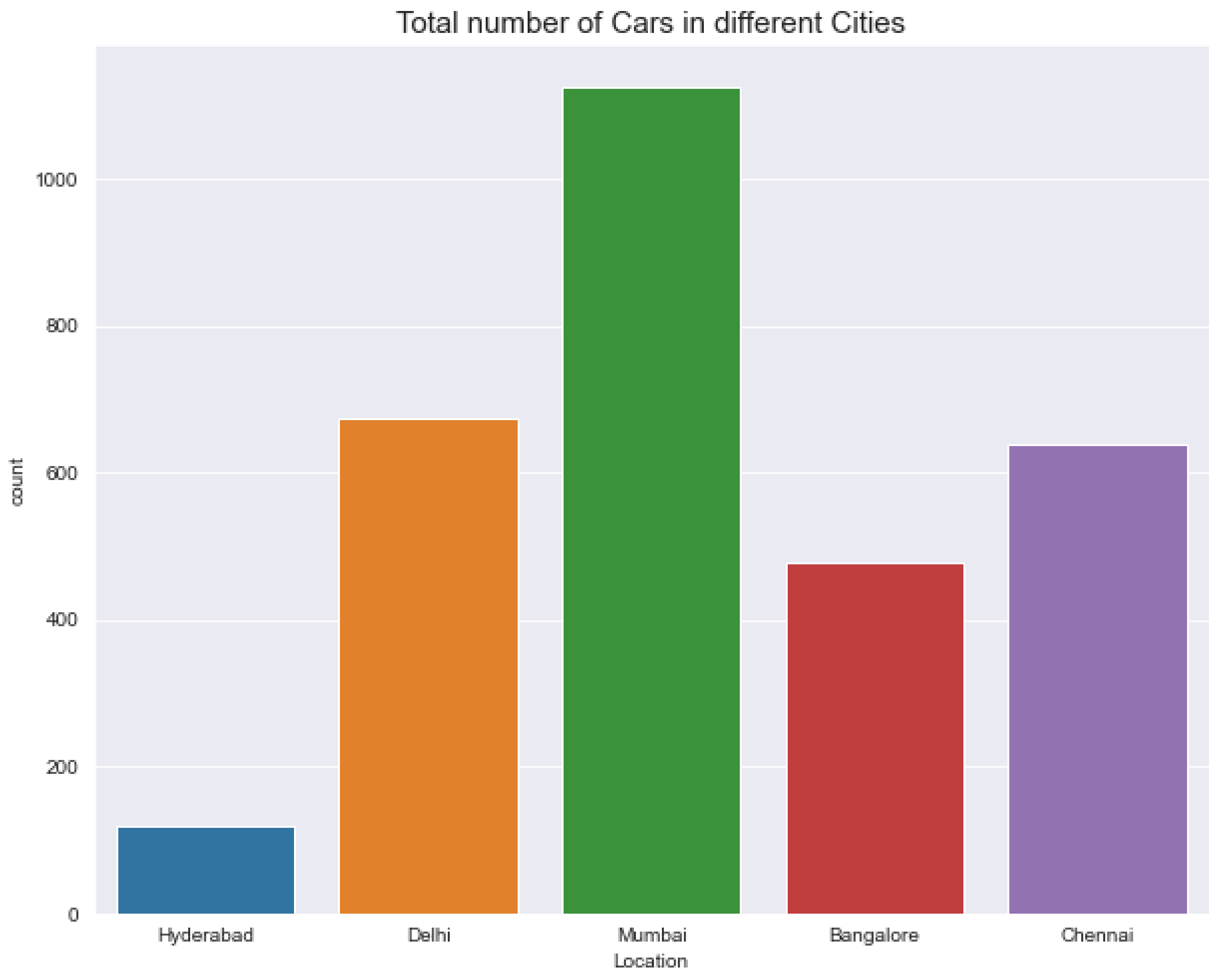
```
0.9993966105577493
```

Visualization:

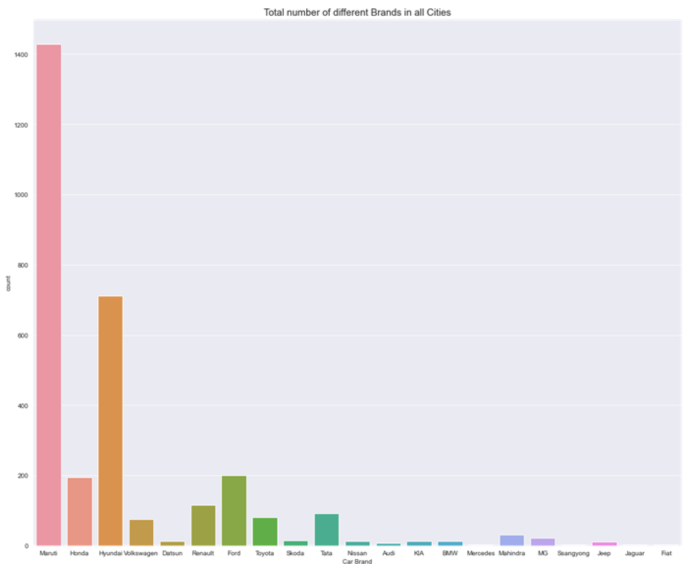
1.’Price’



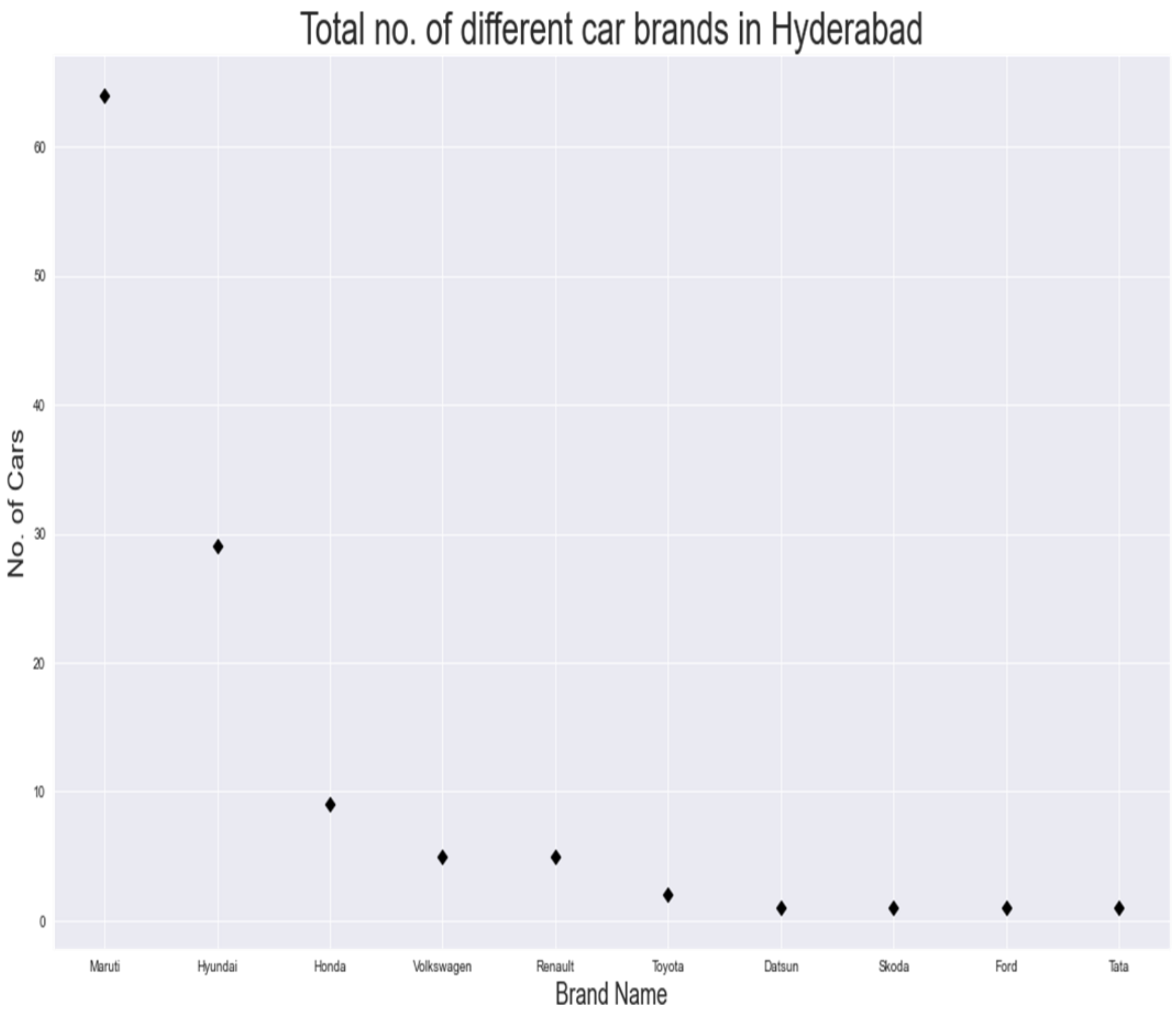
2. Total number of cars in different Cities



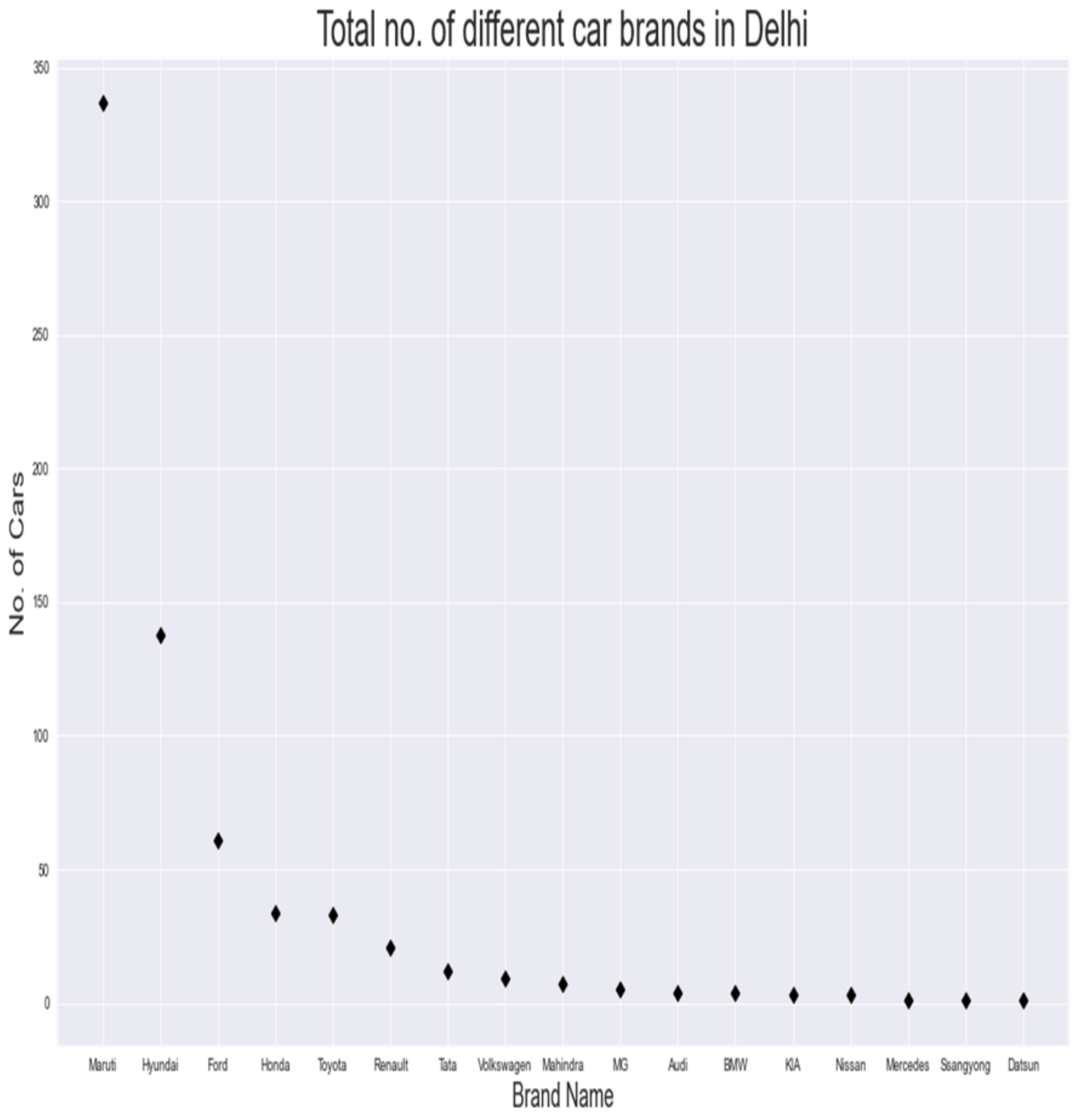
3. No. of different Brands in all Cities



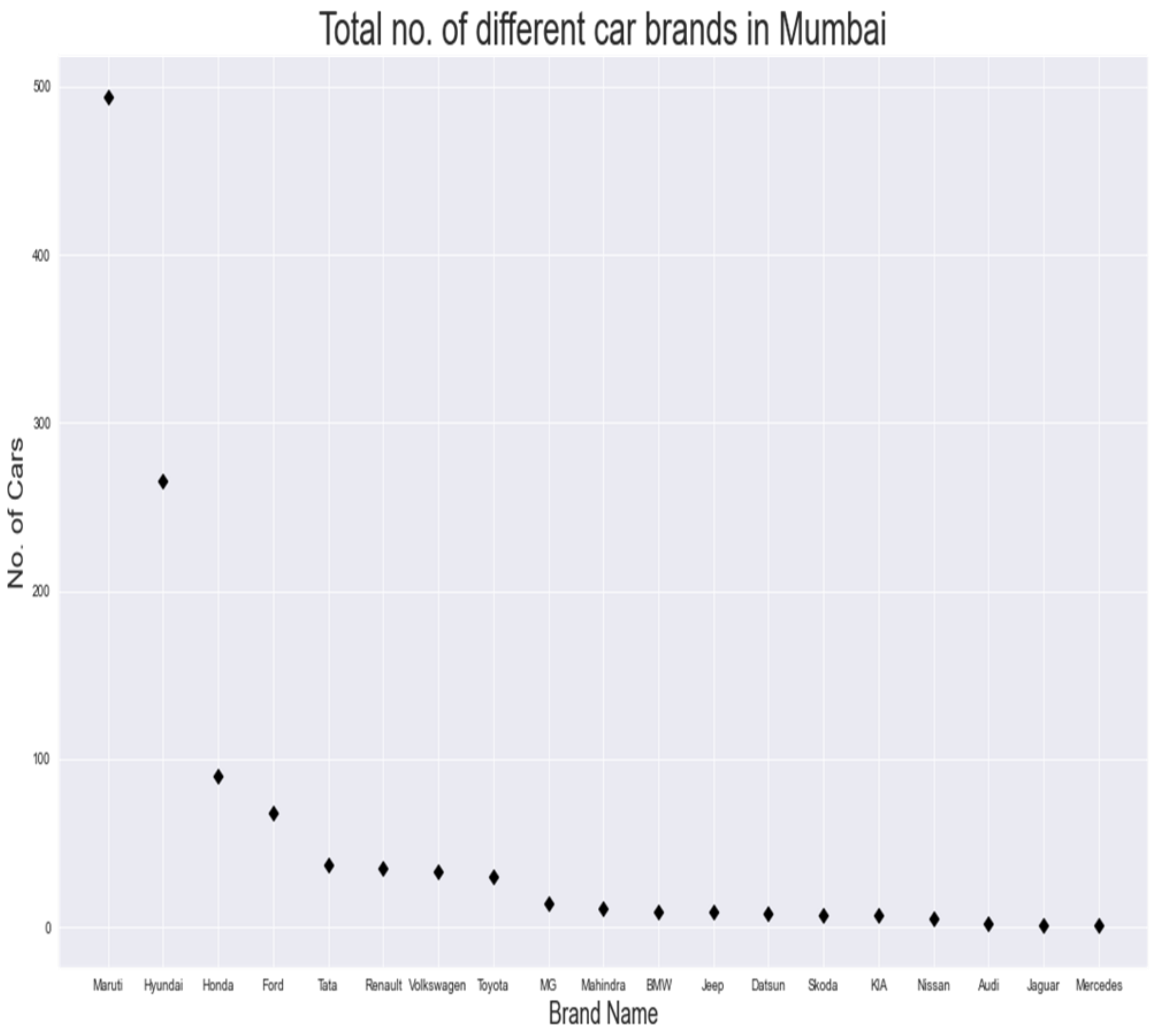
4. Total no. of different Car Brands in Hyderabad



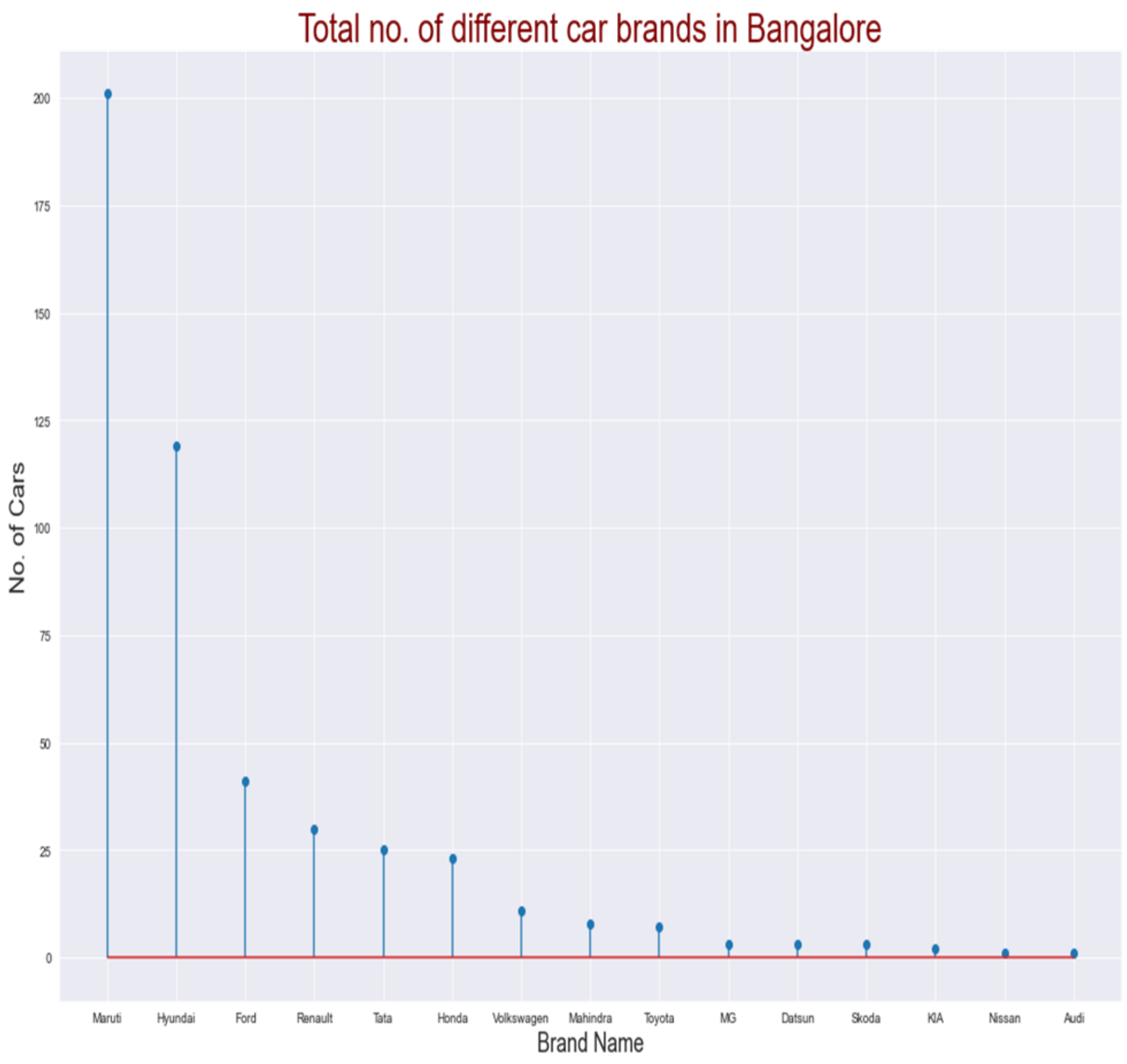
5. Total no. of different Car Brands in Delhi



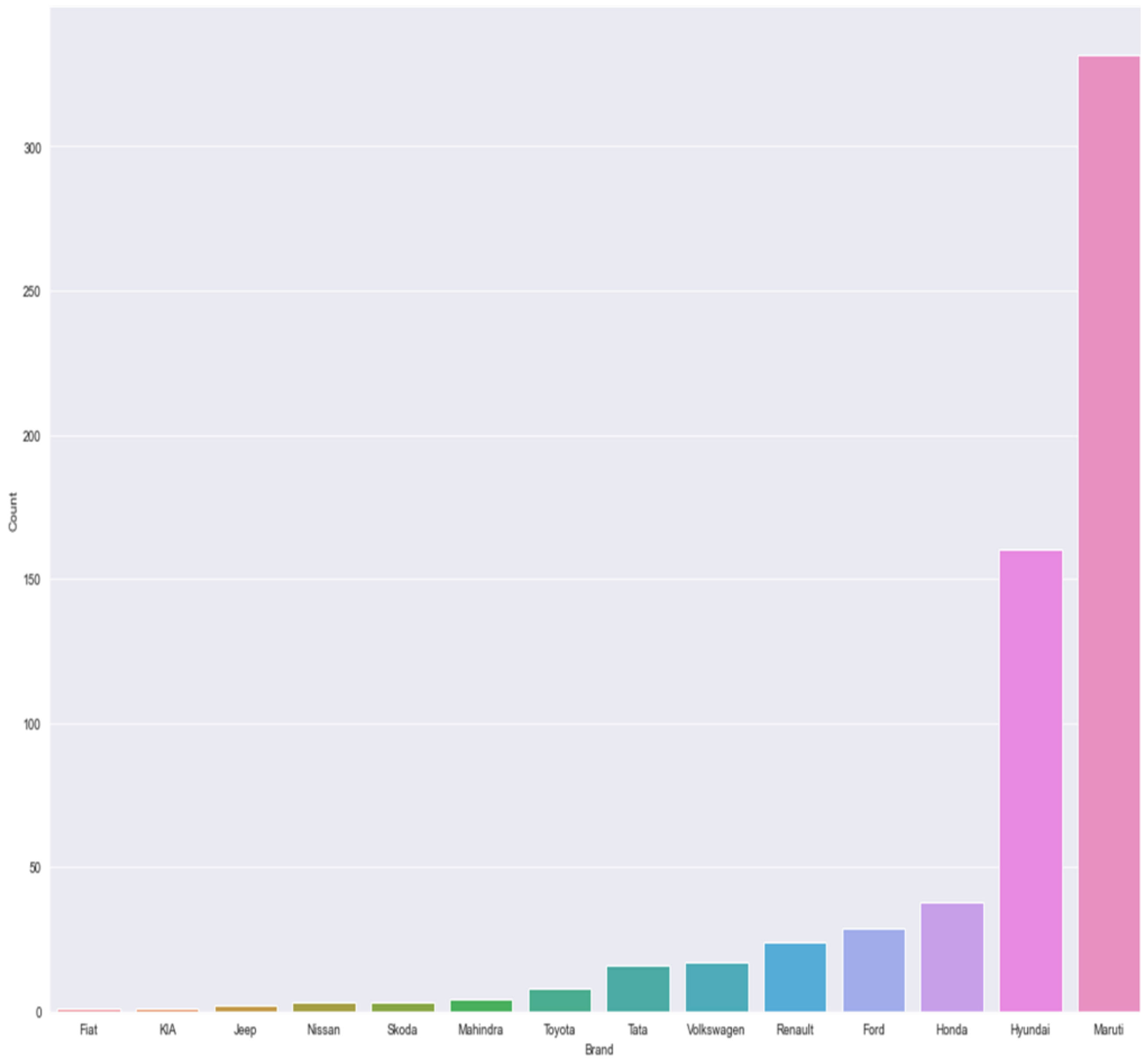
6. Total no. of different Car Brands in Mumbai



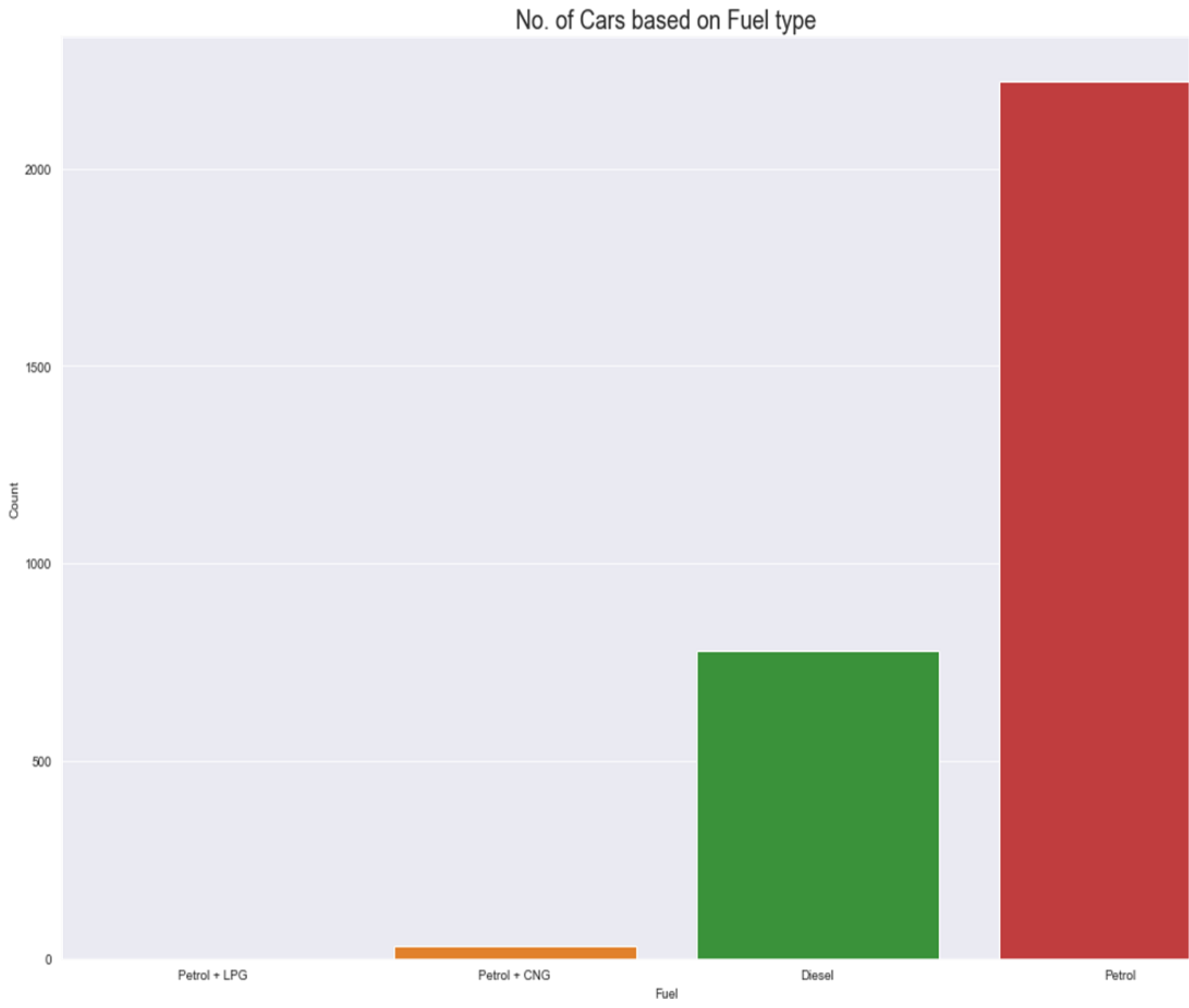
7. Total no. of different Car Brands in Bangalore



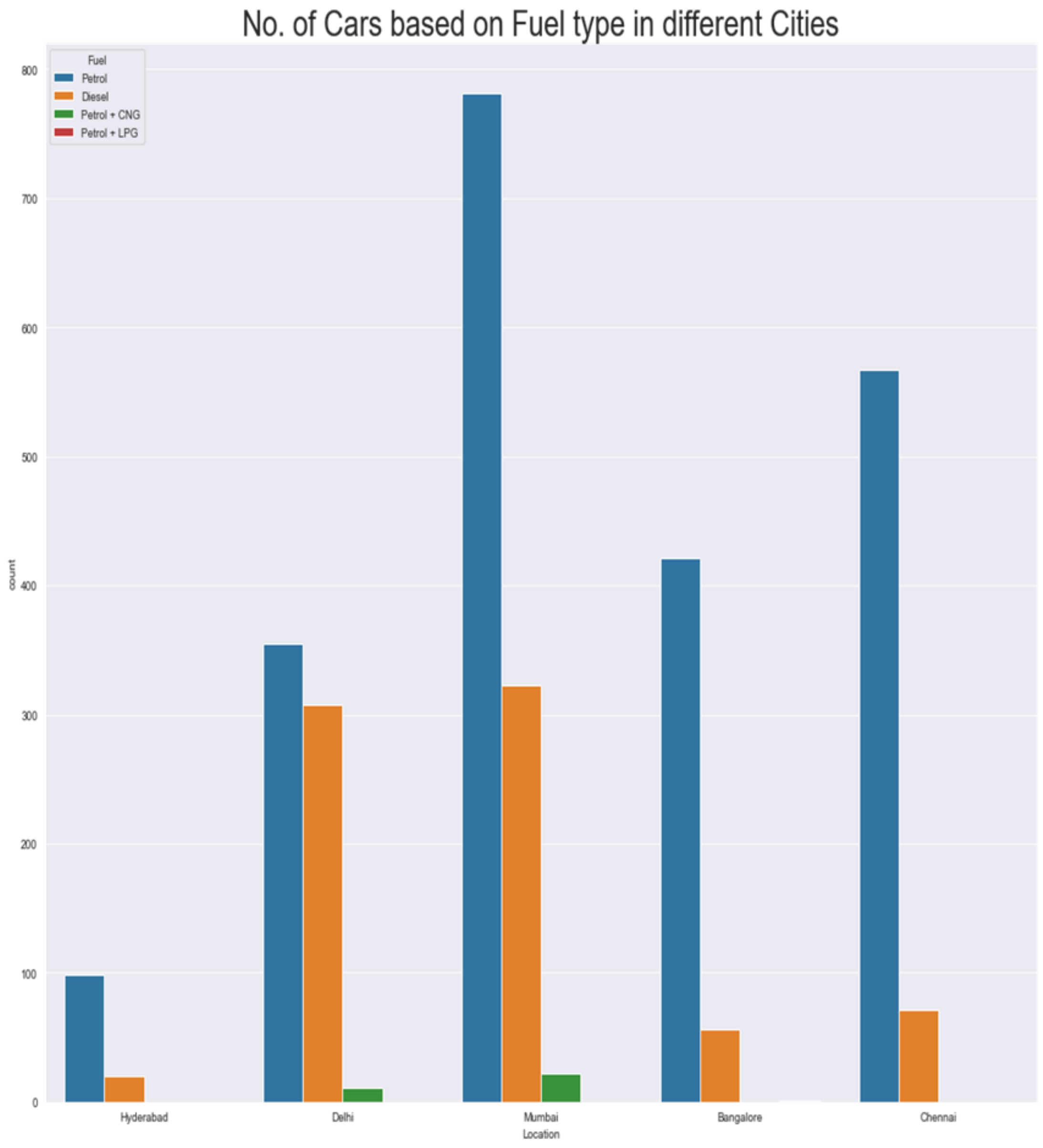
8.Total no. of different Car Brands in Chennai



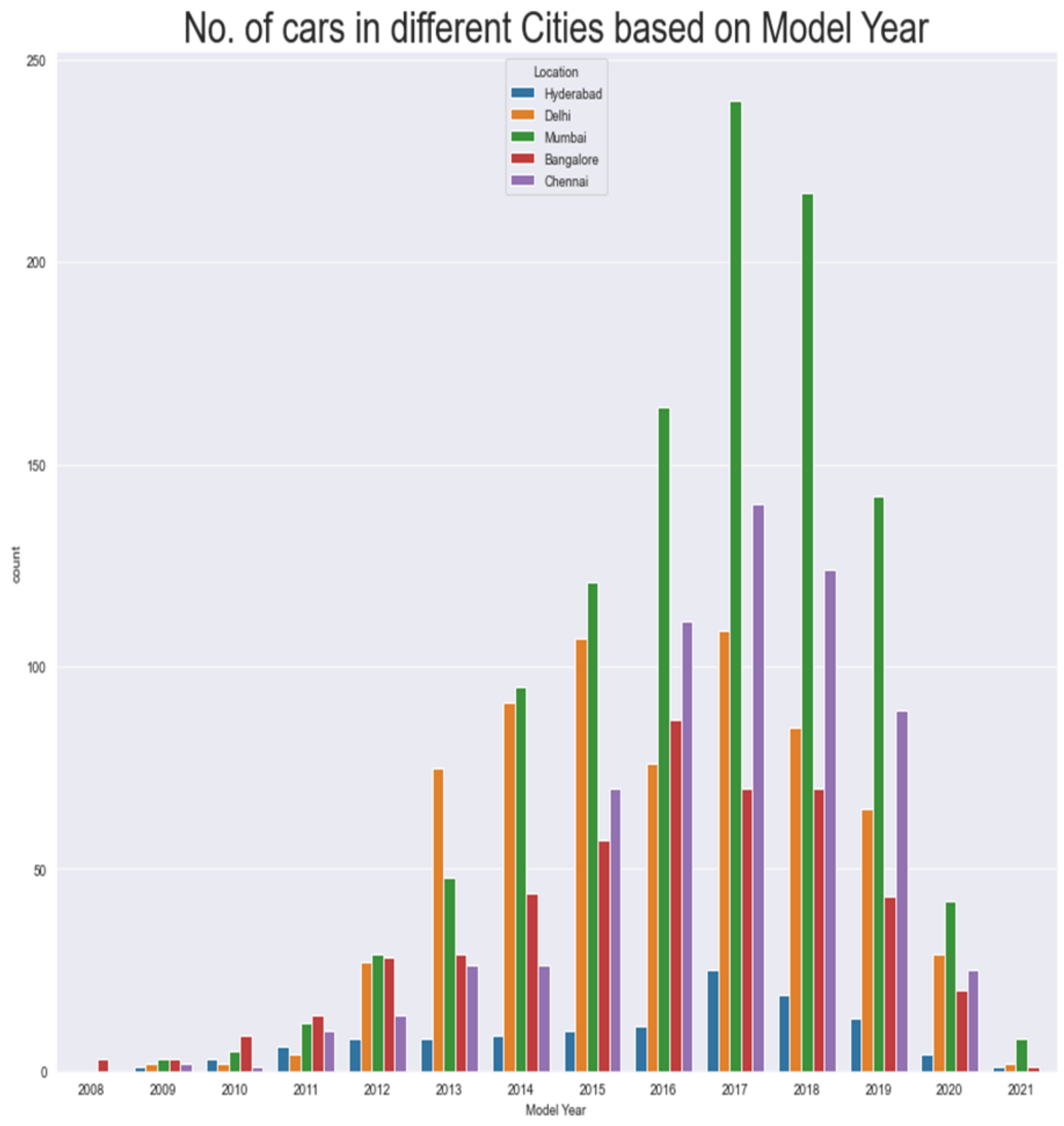
Number of Cars based on Fuel type in all cities



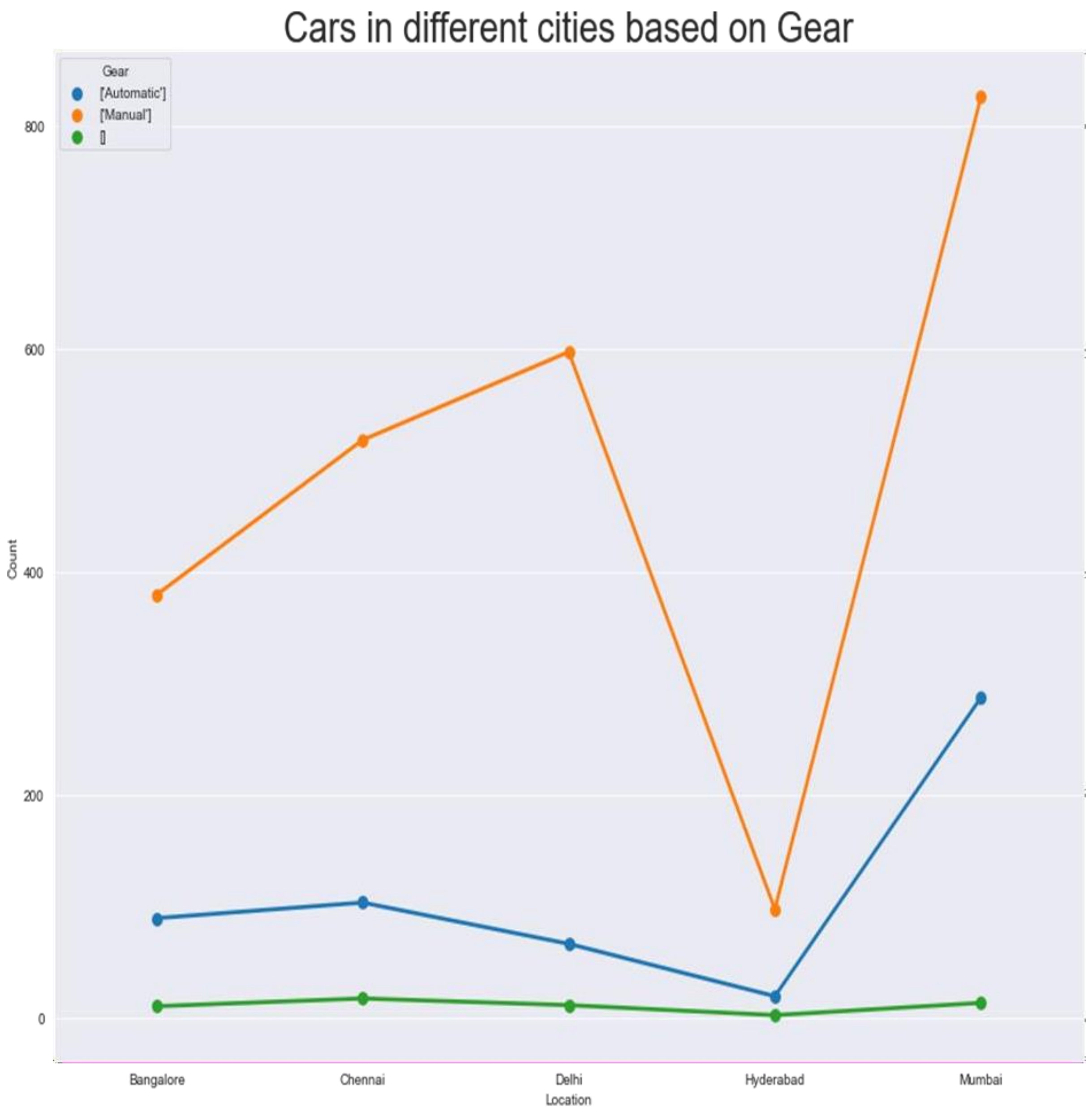
No. of Cars based on Fuel type in different Cities



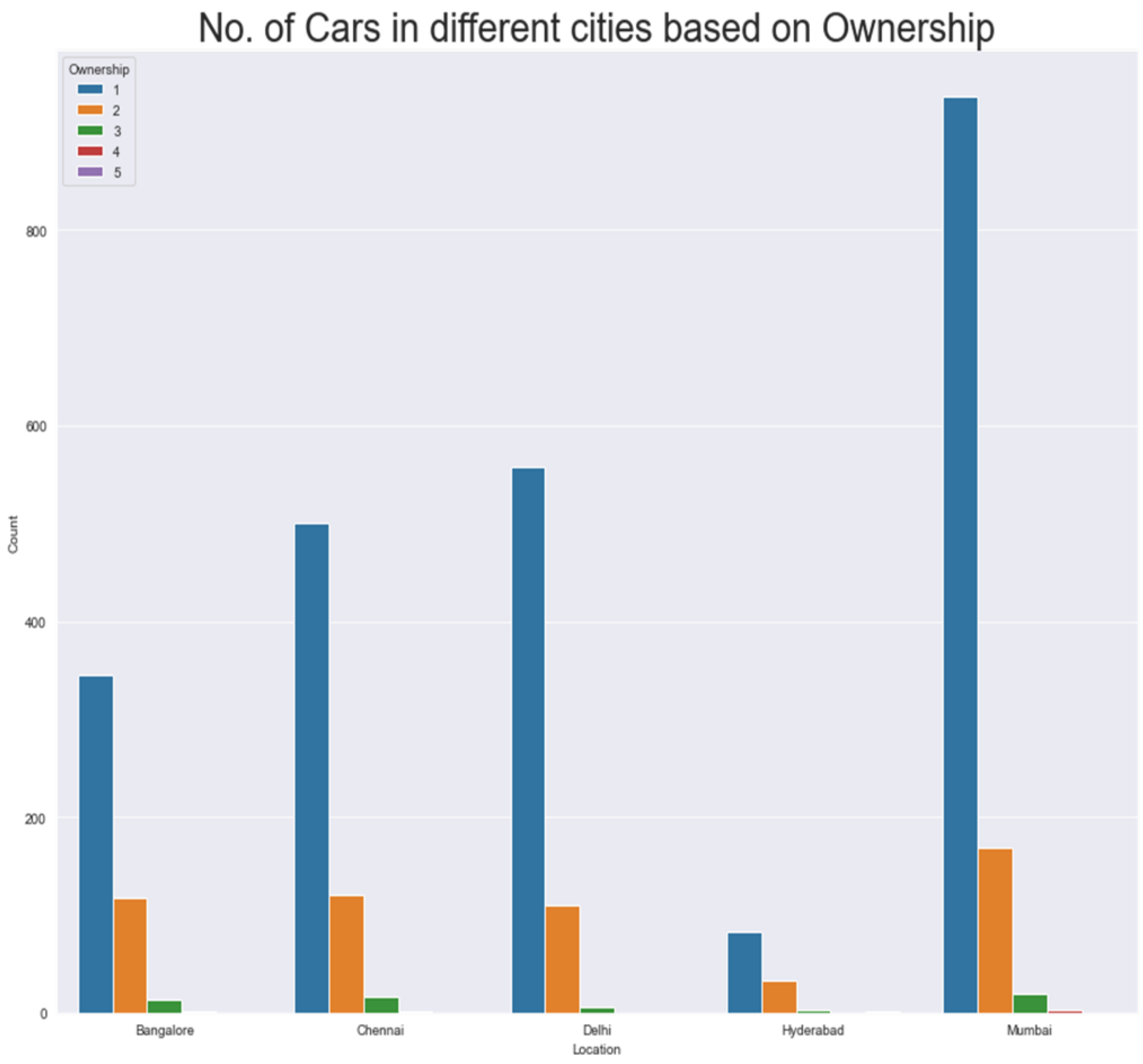
No. of cars in different Cities based on Model Year



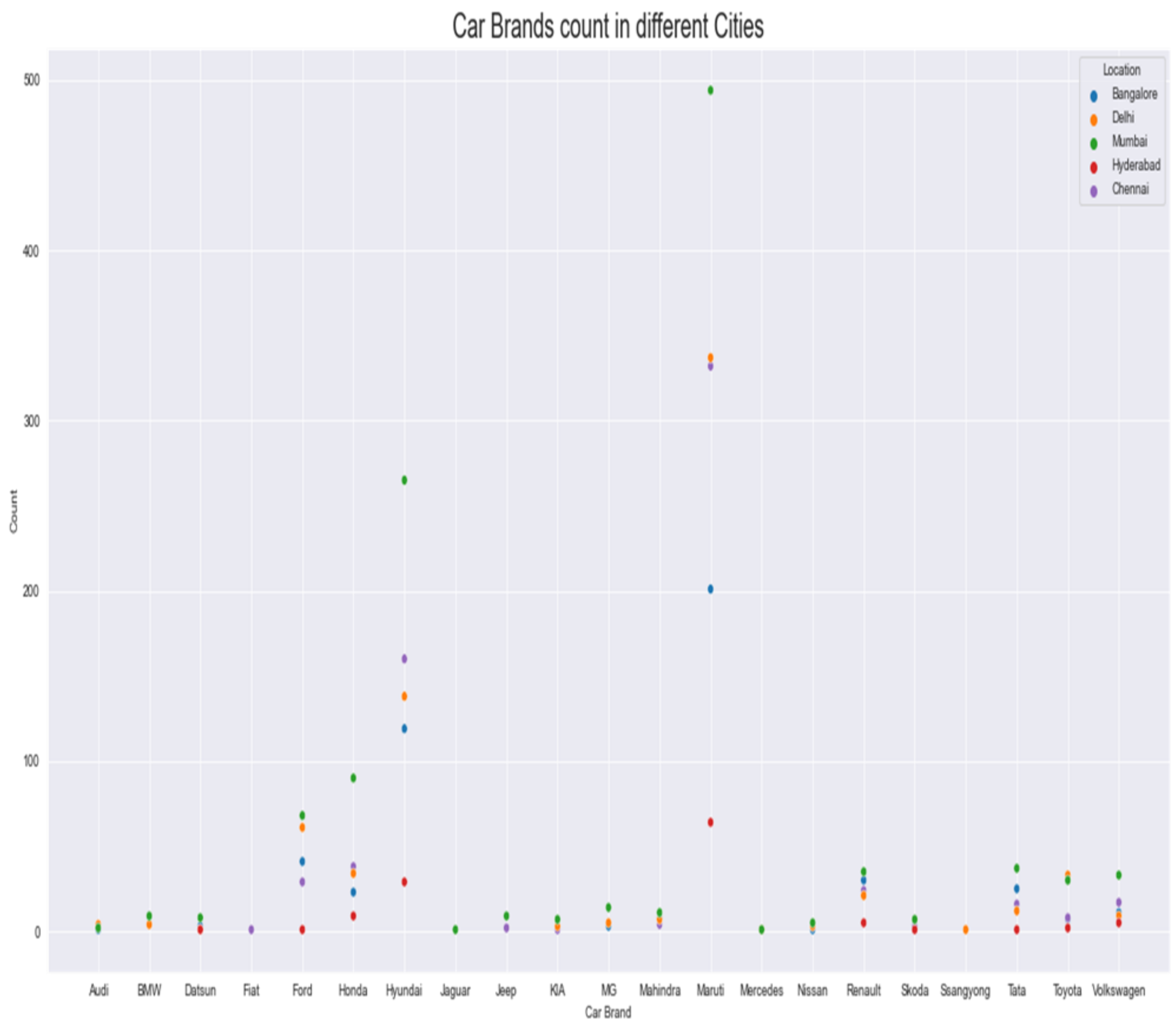
No. of Cars in different cities based on Gear



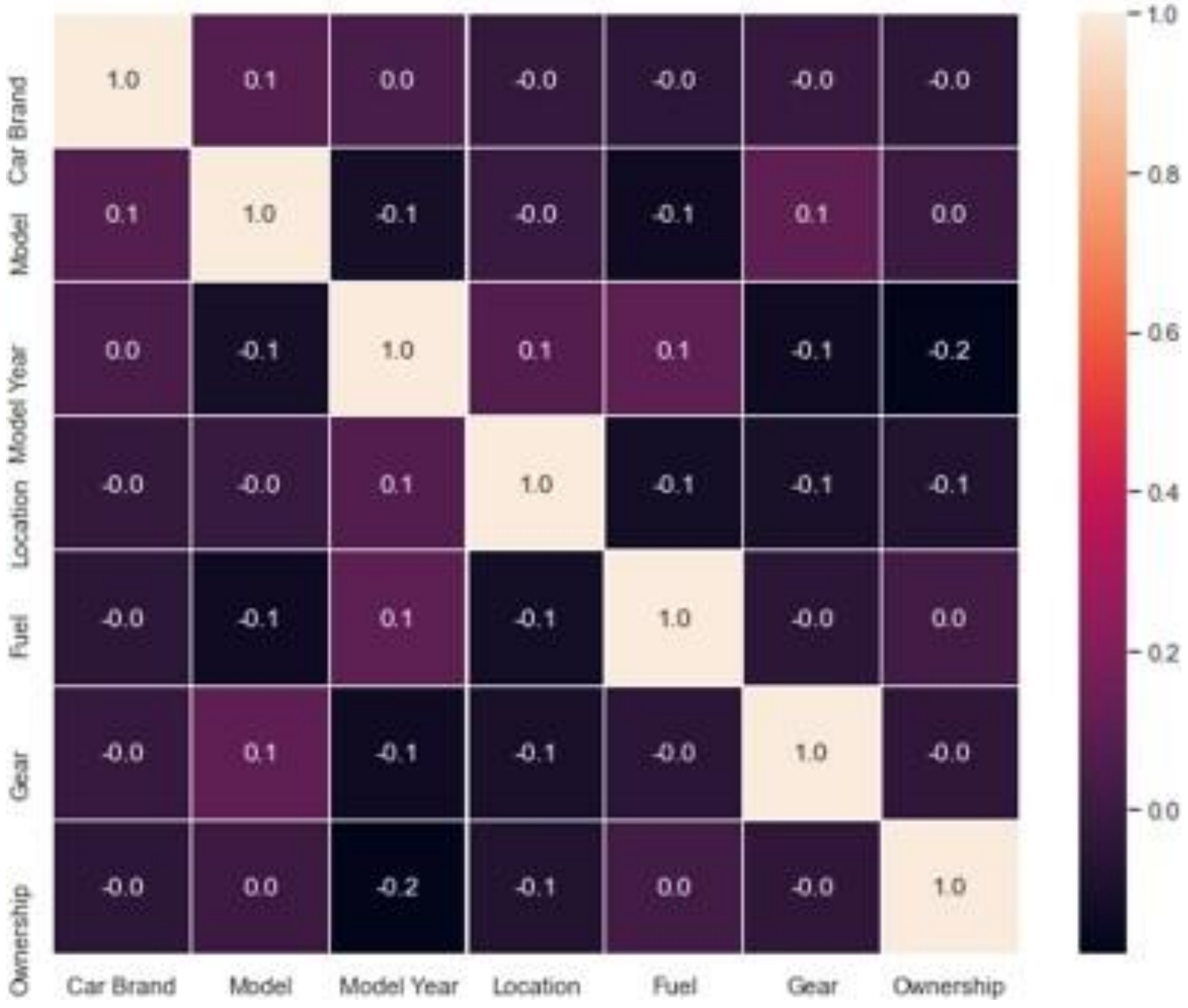
No. of Cars in different cities based on Ownership



Comparision of car brand count in different Cities



Heat Map


Car Brand
Model
Model Year
Location
Fuel
Gear
Ownership
1.0 0.1 0.0 -0.0 -0.0 -0.0 -0.0
0.1 1.0 -0.1 -0.0 -0.1 0.1 0.0
0.0 -0.1 1.0 0.1 0.1 -0.1 -0.2
-0.0 -0.0 0.1 1.0 -0.1 -0.1 -0.1
-0.0 -0.1 0.1 -0.1 1.0 -0.0 0.0
-0.0 0.1 -0.1 -0.1 -0.0 1.0 -0.0
-0.0 0.0 -0.2 -0.1 0.0 -0.0 1.0
1.0
0.8
0.6
0.4
0.2
0.0

Interpretation of the Results

The results are obvious the fuel type is most frequent is diesel and petrol.

And the transmission is manual.

The year is negatively correlated with price and Km ispositively correlated With prices.

Petrol vehicles are large in number followed by diesel.

CONCLUSION

- **Key Findings and Conclusions of the Study**

1. The key step of this problem is to collect the data and make that in the same format and the unwanted values.

2. Key finding is that the most time-consuming thing is data cleaning and data collection.

3. Data cleaning includes formatting data in the same and correctformat, find null values and treat them.

4. As the common scenario the price of the car depends on the year and km and brand.

Learning Outcomes of the Study in respect of Data Science.

As I have scraped the data by my own it will be a good learning to know how to fetch the data and what data will be useful for which output. To understand which steps to take while work on the data from different sources. As unnecessary symbols in the data and removing that is time taking task. To separate the unwanted data like km and symbol of currency etc.

• Limitations of this work and Scope for Future Work.

The data has been collected by scraping the website of CARS24. The data has been cleaned, analysed, encoded and models based on different techniques have been built. The final model is predicting with an r2 score of more than 90%. This can be further improved by expanding the data collection from many other sources and cities.

Since ‘Cars24’ is a dynamic website, the data in this website changes every minute. To analyse the data, every time the code has to be updated.

This analysis will be helpful for the customers / dealers while buying or selling a car.