INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT (IACSD), AKURDI, PUNE

Documentation on

"Used Cars Price Prediction"

PG-DBDA March 2023

Submitted By:

Group no. 17

Roll No.	Name:
233546	Shreya Choudhary
233539	Rinki Darade

Mr. Abhijit Nagargoje
Project Guide

Mr. Rohit Puranik
Centre Coordinator

Abstract

Determining whether the listed price of a used car is a challenging task, due to the many factors that drive a used vehicle's price on the market. The focus of this project is developing machine learning models that can accurately predict the price of a used car based on its features, in order to make informed purchases. We implement and evaluate various learning methods on a dataset consisting of the sale prices of different makes and models across cities in the United States. Our results show that catboost model and Neural Network yield the best results, but are compute heavy. Conventional linear regression also yielded satisfactory results, with the advantage of a significantly lower training time in comparison to the aforementioned methods.

Acknowledgement

I take this occasion to thank God, almighty for blessing us with his grace and taking our endeavor to a successful culmination. I extend my sincere and heartfelt thanks to our esteemed guide, **Mr. Abhijit Nagargoje** for providing me with the right guidance and advice at the crucial juncture sand for showing me the right way. I extend my sincere thanks to our respected Centre Co-Ordinator **Mr. Rohit Puranik**, for allowing us to use the facilities available. I would like to thank the other faculty members also, at this occasion. Last but not the least, I would like to thank my friends and family for the support and encouragement they have given me during the course of our work.

Shreya Choudhary (233546)

Rinki Darade (233539)

Table Of Contents

List of Figures

1. Introduction

1.1 Problem Statement

Used cars Price prediction using various Machine Learning and Deep learning Algorithms and comparing the evaluation metrics for all.

1.2 Abstract

Determining whether the listed price of a used car is a challenging task, due to the many factors that drive a used vehicle's price on the market. The focus of this project is developing machine learning models that can accurately predict the price of a used car based on its features, in order to make informed purchases. We implement and evaluate various learning methods on a dataset consisting of the sale prices of different makes and models across cities in the United States. Our results show that Random Forest model and K-Means clustering with linear regression yield the best results, but are compute heavy. Conventional linear regression also yielded satisfactory results, with the advantage of a significantly lower training time in comparison to the aforementioned methods.

1.3 Aim and objective

The objective is to model the price of used cars with the available independent variables. This model will then be used by the clients to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the clients to understand the pricing dynamics of a new market. Moreover, this might give the management team an insight of the real time world.

1.4 Dataset

For this project, we are using the dataset on used car sales from all over the United States, available on Kaggle. This dataset consists of over 8 Lakh rows and 8 columns in total. Out of all he columns, our target variable is Price of the car and The independent variables are: Year, Mileage, City, State, Vin, Make, Model.

[4]
```python
#loading the data
## true unprocessed data

df = pd.read_csv("/content/drive/MyDrive/pro - Shreya & Rinky/true_car_listings.csv")

df.shape
```

(852122, 8)

[5]
```python
df.sample(10)
```

	Price	Year	Mileage	City	State	Vin	Make	Model
669223	20000	2015	26525	Elk River	MN	KNMAT2MV7FP507751	Nissan	RogueSV
788581	24008	2015	18849	Baltimore	MD	5TDKK3DCXFS639422	Toyota	SiennaLE,
797739	16988	2008	95436	Dyersburg	TN	5TEJU62N28Z472202	Toyota	Tacoma2WD
167163	36900	2014	35941	Newport News	VA	3GCUKSEC5EG323408	Chevrolet	Silverado
189301	15383	2015	22750	Danvers	MA	1C3CCCBB4FN585301	Chrysler	200S
812356	32797	2016	32609	Hendersonville	TN	JTEBU5JR6G5306197	Toyota	4Runner4WD
50241	32998	2014	37677	Tempe	AZ	WBA1J7C59EVW84524	BMW	2
322371	32888	2015	27516	Milwaukie	OR	1FTEW1E80FFB03272	Ford	F-1504WD
748269	13995	2014	20995	West Allis	WI	2T1BURHE8EC175403	Toyota	CorollaS
100008	16995	2017	15546	Matteson	IL	1G1BF5SM2H7190898	Chevrolet	CruzeSedan

2. Overall Workflow

2.1 Big data ETL

We tried one approach where we implemented ETL on Hadoop services and made a Random forest model on Pyspark to check the accuracy it would provide.

Uploading our dataset into hdfs

```
talentum@talentum-virtual-machine:~$ hdfs dfs -mkdir -p /user/talentum
talentum@talentum-virtual-machine:~$ hdfs dfs -mkdir -p /user/talentum/cars_all
talentum@talentum-virtual-machine:~$ hdfs dfs -mkdir -p
-mkdir: Not enough arguments: expected 1 but got 0
Usage: hadoop fs [generic options] -mkdir [-p] <path> ...
talentum@talentum-virtual-machine:~$ hdfs dfs -ls
Found 1 items
drwxr-xr-x   - talentum supergroup          0 2023-08-18 21:27 cars_all
talentum@talentum-virtual-machine:~$ cd shared/
talentum@talentum-virtual-machine:~/shared$ ls
cars  cars_dataset
talentum@talentum-virtual-machine:~/shared$ cd cars_dataset/
talentum@talentum-virtual-machine:~/shared/cars_dataset$ ls
cars_clean_50k.csv  cars_final.ipynb  cars_raw_50k.csv  cars_spark.ipynb  true_car_listings.csv
talentum@talentum-virtual-machine:~/shared/cars_dataset$ hdfs dfs -put true_car_listings.csv /user/talentum/cars_all
talentum@talentum-virtual-machine:~/shared/cars_dataset$ hdfs dfs -ls /user/talentum/cars_all
Found 1 items
-rw-r--r--   1 talentum supergroup   56287873 2023-08-18 21:29 /user/talentum/cars_all/true_car_listings.csv
```

Creating a database on hive and using it to make a new table named 'cars'. We have to provide the same parameters and data types as our original data to accommodate values correctly.

```
talentum@talentum-virtual-machine:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/talentum/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/talentum/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLogger
Binder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/home/talentum/hive/lib/hive-common-2.3.6.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e.
 spark, tez) or using Hive 1.X releases.
hive> create database if not exists project;
OK
Time taken: 2.209 seconds
hive> show databases;
OK
default
project
Time taken: 0.225 seconds, Fetched: 2 row(s)
hive> use project;
OK
Time taken: 0.132 seconds
hive> drop table if exists cars purge;
OK
Time taken: 0.134 seconds
```

Created table and checked first 10 values

```
hive> create external table cars (Price INT, Year INT, Mileage INT ,City String, State String, Vin String,Make String, Model String) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LOCATION '/user/talentum/cars_all/' TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.2 seconds
hive> select * from cars limit 10;
OK
8995    2014    35725   El Paso  TX     19VDE2E53EE000083       Acura   ILX6-Speed
10888   2013    19606   Long Island City        NY      19VDE1F52DE012636       Acura   ILX5-Speed
8995    2013    48851   El Paso  TX     19VDE2E52DE000025       Acura   ILX6-Speed
10999   2014    39922   Windsor  CO     19VDE1F71EE003817       Acura   ILX5-Speed
14799   2016    22142   Lindon   UT     19UDE2F32GA001284       Acura   ILXAutomatic
7989    2012    105246  Miami    FL     JH4CU2F83CC019895       Acura   TSXAutomatic
14490   2014    34032   Greatneck       NY      JH4CU2F84EC002686       Acura   TSXSpecial
13995   2013    32384   West Jordan     UT      JH4CU2F64DC006203       Acura   TSX5-Speed
10495   2013    57596   Waterbury       CT      19VDE2E50DE000234       Acura   ILX6-Speed
9995    2013    63887   El Paso  TX     19VDE1F50DE010450       Acura   ILX5-Speed
Time taken: 0.38 seconds, Fetched: 10 row(s)
hive>
```

Activate Windows
Go to Settings to activate Windows.

Pyspark and Hive connectivity

In [1]:
```
# Intialization
import os
import sys

os.environ["SPARK_HOME"] = "/home/talentum/spark"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
# In below two lines, use /usr/bin/python2.7 if you want to use Python 2
os.environ["PYSPARK_PYTHON"] = "/usr/bin/python3.6"
os.environ["PYSPARK_DRIVER_PYTHON"] = "/usr/bin/python3"
sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.7-src.zip")
sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")

# NOTE: Whichever package you want mention here.
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-xml_2.11:0.6.0 pyspark-shell'
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-avro_2.11:2.4.0 pyspark-shell'
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-xml_2.11:0.6.0,org.apache.spark:spark-avro_2.11:
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-xml_2.11:0.6.0,org.apache.spark:spark-avro_2.1
```

In [2]:
```
#Entrypoint 2.x
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Spark SQL basic example").enableHiveSupport().getOrCreate()

# On yarn:
spark = SparkSession.builder.appName("Spark SQL basic example").enableHiveSupport().master("yarn").getOrCreate()
#specify.master("yarn")

sc = spark.sparkContext
```

In [3]:
```
# Here is the code for making connectivity to hive metastore from pyspark

spark = spark.builder.master('yarn').config('spark.sql.warehouse.dir', '/user/hive/warehouse').config('hive.metastore
spark = spark.builder.master('yarn').enableHiveSupport().getOrCreate()
spark.sql("show databases").show()
spark.sql("use project").show()
spark.sql("show tables").show()
print(" - ")
spark.table('cars').show()
print(spark.catalog.listTables())
print(spark.conf.get('spark.sql.warehouse.dir'))
```

```
+------------+
|databaseName|
+------------+
|     default|
|     project|
+------------+

++
||
++
++

+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
| project|     cars|      false|
+--------+---------+-----------+

 -
+-----+----+-------+----------------+-----+-----------------+-----+-----------+
|price|year|mileage|            city|state|              vin| make|      model|
+-----+----+-------+----------------+-----+-----------------+-----+-----------+
| null|null|   null|            City|State|              Vin| Make|      Model|
| 8995|2014|  35725|         El Paso|   TX|19VDE2E53EE000083|Acura|  ILX6-Speed|
|10888|2013|  19606|Long Island City|   NY|19VDE1F52DE012636|Acura|  ILX5-Speed|
| 8995|2013|  48851|         El Paso|   TX|19VDE2E52DE000025|Acura|  ILX6-Speed|
|10999|2014|  39922|         Windsor|   CO|19VDE1F71EE003817|Acura|  ILX5-Speed|
|14799|2016|  22142|          Lindon|   UT|19UDE2F32GA001284|Acura|ILXAutomatic|
| 7989|2012| 105246|           Miami|   FL|JH4CU2F83CC019895|Acura|TSXAutomatic|
|14490|2014|  34032|       Greatneck|   NY|JH4CU2F84EC002686|Acura|  TSXSpecial|
|13995|2013|  32384|     West Jordan|   UT|JH4CU2F64DC006203|Acura|  TSX5-Speed|
|10495|2013|  57596|       Waterbury|   CT|19VDE2E50DE000234|Acura|  ILX6-Speed|
| 9995|2013|  63887|         El Paso|   TX|19VDE1F50DE010450|Acura|  ILX5-Speed|
|12921|2012|  58550|           Boise|   ID|JH4CU2F44CC003220|Acura|TSXAutomatic|
|12000|2013|  40527|Long Island City|   NY|19VDE1F38DE020867|Acura|  ILX5-Speed|
| 7750|2009|  91980|     San Antonio|   TX|JH4CU26639C015787|Acura|      TSX4dr|
|17628|2015|  13797|           Fargo|   ND|19VDE1F38FE001240|Acura|  ILX5-Speed|
|13999|2013|  35035|       Santa Ana|   CA|JH4CU2F4XDC000369|Acura|  TSX5-Speed|
|14995|2014|  23454|     Hackettstown|   NJ|19VDE1F31EE009243|Acura|  ILX5-Speed|
|14990|2015|  23603|        Freeport|   NY|19VDE1F3XFE007606|Acura|  ILX5-Speed|
|14590|2010|  19250|      Clearwater|   FL|JH4CU2F6XAC041680|Acura|      TSX4dr|
| 9500|2011|  68289|         Arcadia|   FL|JH4CU2F62BC007928|Acura|      TSX4dr|
+-----+----+-------+----------------+-----+-----------------+-----+-----------+
only showing top 20 rows

[Table(name='cars', database='project', description=None, tableType='EXTERNAL', isTemporary=False)]
/user/hive/warehouse
```

```
In [4]: df=spark.read.table("cars")

In [5]: df.columns
Out[5]: ['price', 'year', 'mileage', 'city', 'state', 'vin', 'make', 'model']

In [6]: df.show()
+-----+----+-------+----------------+-----+-----------------+-----+------------+
|price|year|mileage|            city|state|              vin| make|       model|
+-----+----+-------+----------------+-----+-----------------+-----+------------+
| null|null|   null|            City|State|              Vin| Make|       Model|
| 8995|2014|  35725|         El Paso|   TX|19VDE2E53EE000083|Acura|  ILX6-Speed|
|10888|2013|  19606|Long Island City|   NY|19VDE1F52DE012636|Acura|  ILX5-Speed|
| 8995|2013|  48851|         El Paso|   TX|19VDE2E52DE000025|Acura|  ILX6-Speed|
|10999|2014|  39922|         Windsor|   CO|19VDE1F71EE003817|Acura|  ILX5-Speed|
|14799|2016|  22142|          Lindon|   UT|19UDE2F32GA001284|Acura|ILXAutomatic|
| 7989|2012| 105246|           Miami|   FL|JH4CU2F83CC019895|Acura|TSXAutomatic|
|14490|2014|  34032|       Greatneck|   NY|JH4CU2F84EC002686|Acura|  TSXSpecial|
|13995|2013|  32384|     West Jordan|   UT|JH4CU2F64DC006203|Acura|  TSX5-Speed|
|10495|2013|  57596|       Waterbury|   CT|19VDE2E50DE000234|Acura|  ILX6-Speed|
| 9995|2013|  63887|         El Paso|   TX|19VDE1F50DE010450|Acura|  ILX5-Speed|
|12921|2012|  58550|           Boise|   ID|JH4CU2F44CC003220|Acura|TSXAutomatic|
|12000|2013|  40527|Long Island City|   NY|19VDE1F38DE020867|Acura|  ILX5-Speed|
| 7750|2009|  91980|     San Antonio|   TX|JH4CU26639C015787|Acura|      TSX4dr|
|17628|2015|  13797|           Fargo|   ND|19VDE1F38FE001240|Acura|  ILX5-Speed|
|13999|2013|  35035|       Santa Ana|   CA|JH4CU2F4XDC000369|Acura|  TSX5-Speed|
|14995|2014|  23454|    Hackettstown|   NJ|19VDE1F31EE009243|Acura|  ILX5-Speed|
|14990|2015|  23603|        Freeport|   NY|19VDE1F3XFE007606|Acura|  ILX5-Speed|
|14590|2010|  19250|      Clearwater|   FL|JH4CU2F6XAC041680|Acura|      TSX4dr|
| 9500|2011|  68289|         Arcadia|   FL|JH4CU2F62BC007928|Acura|      TSX4dr|
```


Basic preprocessing

```
In [11]: df.dropDuplicates().show()

+-----+----+-------+-----+------------+
|price|year|mileage| make|       model|
+-----+----+-------+-----+------------+
|13998|2010|  41652|Acura|      TSX4dr|
|16750|2015|  57483|Acura|  ILX5-Speed|
|16517|2015|  48918|Acura|  ILX5-Speed|
|19968|2016|  31019|Acura|ILXAutomatic|
|17384|2013|  35257|Acura|      TLBase|
|14539|2012|  59046|Acura|       TL2WD|
| 7995|2008|  78692|Acura|      RDX4WD|
| 4988|2000| 109624|Acura| TLAutomatic|
|22500|2014|  35341|Acura|       TL2WD|
|15000|2011|  80442|Acura|      RDXFWD|
|20190|2015|  31235|Acura|      TLXFWD|
|23499|2014|  48194|Acura|     TLSH-AWD|
|21800|2015|  32649|Acura|      TLXFWD|
|27999|2016|  10398|Acura|      TLXFWD|
|27749|2016|  15429|Acura|      TLXFWD|
|25656|2015|  29778|Acura|      TLXFWD|
|16734|2012|  77424|Acura|       TL2WD|
|33990|2016|   7205|Acura|      RDXFWD|
|24420|2013|  56175|Acura|     MDXwith|
| 4499|2003| 184707|Acura| TLAutomatic|
+-----+----+-------+-----+------------+
only showing top 20 rows
```

2.2 Data cleaning and Preprocessing

Another approach is using Python on google collab

Out of all independent columns, the column Vin acts as an Id for the car models so we will be dropping it since there is no significance to it. Afterwards, duplicated values (if present) were checked, pandas has an inbuilt function which returns duplicate rows present in the dataframe.

```
[ ]  #Vin is unecessary column so we are dropping it
     df.drop('Vin',axis=1,inplace=True)
     df.shape
```

```
(852122, 7)
```

```
[ ]  #checking duplicate data that might be present in the dataset

     dfdup=df[df.duplicated()]
     dfdup
```

	Price	Year	Mileage	City	State	Make	Model
314	22000	2017	10	Chicago	IL	Acura	ILXAutomatic
1259	23566	2017	16	Larchmont	NY	Acura	ILXPremium
6258	36000	2018	5	Littleton	CO	Acura	RDXAWD
6356	33900	2017	4250	Salt Lake City	UT	Acura	RDXAWD
7180	38275	2018	5	Littleton	CO	Acura	RDXAWD
...	...	...	...	...	...	...	...
819060	36998	2014	35370	Houston	TX	Toyota	4Runner4x4
819874	32018	2017	6	Mechanicsville	VA	Toyota	Tacoma2WD
824387	16995	2017	15	San Antonio	TX	Volkswagen	Passat1.8T
825336	16996	2017	7787	San Antonio	TX	Volkswagen	Passat1.8T
851162	39000	2016	7	San Francisco	CA	Volvo	XC60FWD

554 rows × 7 columns

So, it could be seen that the dataset has 554 duplicated rows, it's typically a good idea to remove duplicate data points so the model can better generalize to the full dataset. The duplicated rows can be dropped as:

```
[ ]  1 #checking shape of dataset
     2 df.shape
```

```
(852122, 7)
```

```
[ ]  1 #removing duplicated rows from the dataset
     2 df.drop_duplicates(inplace=True)
     3
     4 #again checking for shape of the now data
     5 df.shape
```

```
(851568, 7)
```

Afterwards, missing values were checked. It came out that the dataset had no missing values present.

```
[ ]  1 #checking for missing values
     2
     3 df.isnull().sum()
```

```
Price      0
Year       0
Mileage    0
City       0
State      0
Make       0
Model      0
dtype: int64
```

```
[ ]  1 #checking all datatypes
     2
     3 df.dtypes
```

```
Price       int64
Year        int64
Mileage     int64
City       object
State      object
Make       object
Model      object
dtype: object
```

2.3 Exploratory Data Analysis

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns and to check summary statistics and graphical representations.

Now we will be checking the unique number of values that are present in the categorical columns

```
[ ]
#checking number of unique values in the columns

df.nunique()
```

```
Price        47124
Year            22
Mileage     158836
City          2553
State           59
Make            58
Model         2736
dtype: int64
```

Unique values in the columns

```
[ ]
#checking out different classes or labels present in the variables
cat=df.select_dtypes(include=['object']).columns
for i in df.loc[:,cat]:
    print(df[i].value_counts(),'\n\n') ## value_count counts all unique values
    ##dropping all values who has count < 50
    counts = df[i].value_counts()
    df = df[~df[i].isin(counts[counts < 50].index)]
```

```
NH    5522
NE    4924
IA    4768
NM    4582
ID    3444
HI    2911
DE    2399
MT    1912
RI    1818
ME    1797
```

To check the correlation between all the columns present(categorical and numerical), we will be Encoding the categorical columns using Label encoder from sklearn

```
[ ] df_lbe=df.copy()
```

```
[ ] cat
```

```
Index(['City', 'State', 'Make', 'Model'], dtype='object')
```

```
[ ] #converting categorical variables into numerical ones using LabelEncoder so that we can check the coorelation between all variables

from sklearn.preprocessing import LabelEncoder

for i in cat:
    df_lbe[i] = LabelEncoder().fit_transform(df_lbe[i])
```

Checking the correlation of all variables with each other

```
[ ] #checking out correlation matrix

df_lbe.corr()
```

	Price	Year	Mileage	City	State	Make	Model
Price	1.000000	0.437095	-0.444979	-0.015443	0.027127	-0.079957	0.076785
Year	0.437095	1.000000	-0.765428	0.010485	-0.023711	0.023872	-0.014434
Mileage	-0.444979	-0.765428	1.000000	-0.013848	0.024843	-0.032335	0.045073
City	-0.015443	0.010485	-0.013848	1.000000	-0.048887	0.007954	-0.003859
State	0.027127	-0.023711	0.024843	-0.048887	1.000000	-0.003653	0.006630
Make	-0.079957	0.023872	-0.032335	0.007954	-0.003653	1.000000	0.030264
Model	0.076785	-0.014434	0.045073	-0.003859	0.006630	0.030264	1.000000

Checking the correlation of all variables only with respect to price

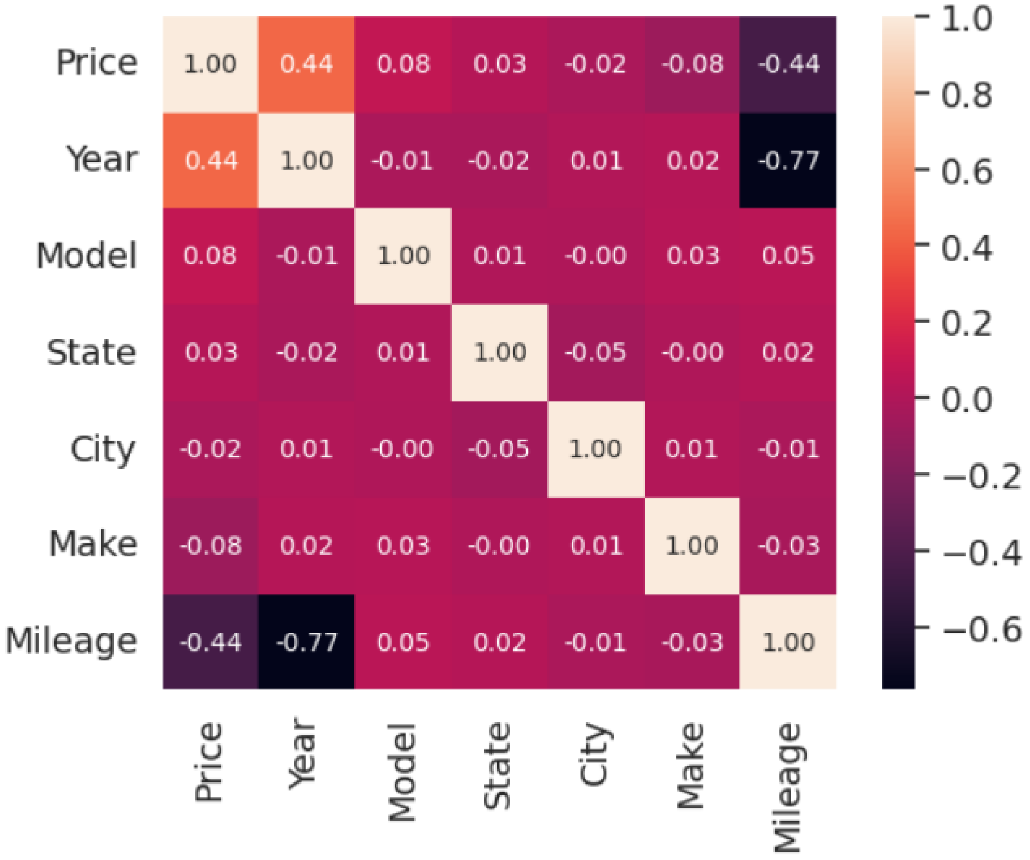
```
[ ] # Find most important features relative to target Price

print("Find most important features relative to target")
corr = df_lbe.corr()
corr.sort_values(["Price"], ascending = False, inplace = True)
print(corr.Price)
```

```
Find most important features relative to target
Price      1.000000
Year       0.437095
Model      0.076785
State      0.027127
City      -0.015443
Make      -0.079957
Mileage   -0.444979
Name: Price, dtype: float64
```

Checking the correlation graphically using heatmap

```python
# Price correlation matrix
k = 7   #number of variables for heatmap
corrmat = df_lbe.corr()
cols = corrmat.nlargest(k, 'Price')['Price'].index
cm = np.corrcoef(df_lbe[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



Conclusion from correlation matrix:

Column ‘Year’ has a mild positive correlation with target ‘Price’

Column ‘Mileage’ has a mild negative correlation with ‘Price’

Columns ‘Make’ and ‘Model’ have low positive correlation with target ‘Price’

Columns ‘City’ and ‘State’ have trivial correlation with the target column ‘Price’

2.4 Feature selection

First, we had dropped ‘City’ and ‘State’ from the dataset, because it must not contribute to the prediction of Prices also both these categorical variables have large number of unique classes present, so it would be a bit complex in converting these categorical variables into numerical ones.

```
[28]
#Dropping City and State as they have minimal correlation for prediction of price
df.drop(['City','State'],axis=1,inplace=True)
```

```
[29]
cat=df.select_dtypes(include=['object']).columns
```

```
[30]
df.head()
```

	Price	Year	Mileage	Make	Model
1	10888	2013	19606	Acura	ILX5-Speed
3	10999	2014	39922	Acura	ILX5-Speed
4	14799	2016	22142	Acura	ILXAutomatic
5	7989	2012	105246	Acura	TSXAutomatic

Now we will use Standard Scaler from sklearn to apply scaling on the columns Year and Mileage. This will help some models to perform better and give relevant predictions.

```
[30]
df.head()
```

	Price	Year	Mileage	Make	Model
1	10888	2013	19606	Acura	ILX5-Speed
3	10999	2014	39922	Acura	ILX5-Speed
4	14799	2016	22142	Acura	ILXAutomatic
5	7989	2012	105246	Acura	TSXAutomatic
6	14490	2014	34032	Acura	TSXSpecial

```
[31]
from sklearn.preprocessing import StandardScaler        #scaling the data
##Apply scaling on Independant variables only so price predicted in end by the model does not come scaled

sc= StandardScaler()
sc.fit(df.loc[:,['Year','Mileage']])
##fitting on only x(Independent variables)

df.loc[:,['Year','Mileage']] = sc.transform(df.loc[:,['Year','Mileage']])
```

Outliers

Z score is also called standard score. This score helps to understand if a data value is greater or smaller than mean and how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean. If the z score of a data point is more than 3 or -3, it indicates that the data point is quite different from the other data points. Such a data point can be an outlier. So we are only selecting the rows that have value in any of the columns greater than -3 and less than 3.

Outlier removal

[32]

```
#function to select only rows whose value is > -3 and value < 3

def outlier_removal_zscore(df , cont_columns):
    for col in cont_columns:
        print("Before removing outliers from col=",col)
        print("Shape =",df.shape)
        df = df.loc[(df[col] >= -3)&(df[col] <= 3),:]
        print("After removing outliers from col=",col)
        print("Shape =",df.shape)
    return df

df = outlier_removal_zscore(df.copy(),['Year','Mileage'])
```

```
Before removing outliers from col= Year
Shape = (818681, 5)
After removing outliers from col= Year
Shape = (804931, 5)
Before removing outliers from col= Mileage
Shape = (804931, 5)
After removing outliers from col= Mileage
Shape = (797354, 5)
```

We'll be using One hot encoded data while training the model

But we'll only be sampling 50000 rows from all of the cleaned and scaled data.

```
[33] #converting categorical variables into numerical ones using One Hot Encoder

df = pd.get_dummies(df)
```

```
[34] df.shape
```

```
(797354, 1277)
```

```
[35] df=df.sample(50000,random_state=7) #sampling only limited amount of data since there are too many OHE columns
```

```
[36] df.shape
```

```
(50000, 1277)
```

After One hot encoding we have 1277 columns.

Saving One hot encoded data into an empty csv file

```
[ ] # Saving the scaled and one hot encoded data with no outliers to an empty csv file
##DO NOT RUN AGAIN
#df.to_csv("/content/drive/MyDrive/pro - Shreya & Rinky/cars_ohe_sc.csv",index= False, header=True)
```

2.5 Splitting X and Y

```
[40] #seperating the independant and dependant variables
x = df.drop('Price',axis=1)
y = df['Price']

x.shape,y.shape
```

```
((50000, 1276), (50000,))
```

After splitting the data in x and y, and applying train test split, we are applying PCA to select only the significant values that will affect our target variable

2.6 Train test split and PCA

[41]
```
#using train test split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=777,test_size=.30)
```

PCA

[42]
```
from sklearn.decomposition import PCA

#applying pca to reduce dimensions and setting n_components to 0.98
pca = PCA(n_components = 0.98)

x_train = pca.fit_transform(x_train)
```

[43]
```
x_test = pca.transform(x_test)
```

[44]
```
explained_variance = pca.explained_variance_ratio_ #eigen values/ total eigen values
```

[45]
```
x_train.shape
```

```
(35000, 676)
```

After applying PCA we have 35000 rows and 676 columns which are the only relevant values for our Price prediction

3. Building Machine Learning Models

Now we will train several Machine Learning models and compare their results. Later on, we will use evaluation metrics.

3.1 Lasso

Laaso Regression

```
#Lasso

ls=Lasso()
ls.fit(x_train,y_train)
predtrain=ls.predict(x_train)
predls=ls.predict(x_test)

```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordina
  model = cd_fast.enet_coordinate_descent(
```

```
#checking error for Lasso

msels=mean_squared_error(y_test,predls)
maels=mean_absolute_error(y_test,predls)
r2ls=r2_score(y_test,predls)
print(msels)
print(maels)
print(r2ls)
```

```
28910370.352732252
3200.514511821493
0.7990412586745311
```

3.2 Ridge

Ridge Regression

```
[ ]
#Ridge

lrr=Ridge()
lrr.fit(x_train,y_train)
predlrr=lrr.predict(x_test)
print(r2_score(y_test,predlrr)*100)

```

```
81.70848343584713
```

```
[ ]
#checking error for Ridge

mselrr=mean_squared_error(y_test,predlrr)
maelrr=mean_absolute_error(y_test,predlrr)
r2lrr=r2_score(y_test,predlrr)
print(mselrr)
print(maelrr)
print(r2lrr)
```

```
26314581.52528624
2947.8487873351924
0.8170848343584713
```

3.3 Random Forest

Random Forest Regressor

```
#RandomForest

rf=RandomForestRegressor()
rf.fit(x_train,y_train)
predrf=rf.predict(x_test)
print(r2_score(y_test,predrf)*100)
```

```
80.41570919614958
```

[55]

```
#checking error for RandomForest

mserf=mean_squared_error(y_test,predrf)
maerf=mean_absolute_error(y_test,predrf)
r2rf=r2_score(y_test,predrf)
print(mserf)
print(maerf)
print(r2rf)
```

```
28174395.22662692
2880.621816409524
0.8041570919614958
```

3.4 Catboost

Catboost Regression

[56]

```
X = df.drop('Price',axis=1)
Y = df['Price']
```

[57]

```
# Catboost requires validation dataset as well so we are implementing train_valid_test_split method from a fast_ml library

from fast_ml.model_development import train_valid_test_split
X_train,Y_train,X_valid,Y_valid,X_test,Y_test=train_valid_test_split(df,target='Price',train_size=0.6,valid_size=0.2,test_size=0.2,random_state=777)

X_train.shape,Y_train.shape, X_valid.shape,Y_valid.shape,X_test.shape,  Y_test.shape
```

```
((30000, 1276), (30000,), (10000, 1276), (10000,), (10000, 1276), (10000,))
```

[58]

```
#catboost needs to be given pool of data hence Pool method
train_dataset = cb.Pool(X_train, Y_train)
eval_dataset = cb.Pool(X_valid, Y_valid)
```

[59]

```
#Creating model
cbt = cb.CatBoostRegressor(loss_function='RMSE')
```

[60]

```
#Running possibilities for the best model
grid = {'iterations': [ 250, 300],
        'learning_rate': [ 0.2,0.5,0.7],
        'depth': [8, 10],
        'l2_leaf_reg': [0.2, 0.4]}
cbt.grid_search(grid, train_dataset)
```

```
Streaming output truncated to the last 5000 lines.
41:     learn: 6955.9700441     test: 7255.1079661      best: 7255.1079661 (41) total: 834ms    remaining: 5.13s
42:     learn: 6921.9623581     test: 7220.5414385      best: 7220.5414385 (42) total: 854ms    remaining: 5.1s
43:     learn: 6881.1427863     test: 7184.1292651      best: 7184.1292651 (43) total: 874ms    remaining: 5.08s
44:     learn: 6846.7009618     test: 7137.4235909      best: 7137.4235909 (44) total: 892ms    remaining: 5.06s
45:     learn: 6816.2292652     test: 7119.5837000      best: 7119.5837000 (45) total: 922ms    remaining: 5.09s
46:     learn: 6787.5088556     test: 7099.0851610      best: 7099.0851610 (46) total: 940ms    remaining: 5.06s
47:     learn: 6760.7678046     test: 7080.0754851      best: 7080.0754851 (47) total: 964ms    remaining: 5.06s
48:     learn: 6729.8440281     test: 7054.9357216      best: 7054.9357216 (48) total: 987ms    remaining: 5.05s
49:     learn: 6704.0447037     test: 7030.7931330      best: 7030.7931330 (49) total: 1s       remaining: 5.02s
50:     learn: 6673.2559944     test: 7005.1140692      best: 7005.1140692 (50) total: 1.02s    remaining: 4.97s
```

[61]

```
#fit on best model
cbt.fit(train_dataset,eval_set=eval_dataset,early_stopping_rounds=50,plot=True,use_best_model=True, silent=False)
```

```
201:    learn: 3723.0781970     test: 4320.5763834      best: 4320.5763834 (201)        total: 2.7s     remaining: 641ms
202:    learn: 3719.4464177     test: 4319.0156706      best: 4319.0156706 (202)        total: 2.71s    remaining: 628ms
203:    learn: 3716.8812838     test: 4318.3066445      best: 4318.3066445 (203)        total: 2.72s    remaining: 614ms
204:    learn: 3712.6483496     test: 4317.8308741      best: 4317.8308741 (204)        total: 2.74s    remaining: 601ms
205:    learn: 3708.2120752     test: 4303.7900889      best: 4303.7900889 (205)        total: 2.75s    remaining: 588ms
206:    learn: 3705.5522389     test: 4301.3395999      best: 4301.3395999 (206)        total: 2.76s    remaining: 574ms
207:    learn: 3698.3753700     test: 4299.1454469      best: 4299.1454469 (207)        total: 2.78s    remaining: 561ms
208:    learn: 3693.6419538     test: 4292.2102901      best: 4292.2102901 (208)        total: 2.79s    remaining: 548ms
209:    learn: 3691.0667249     test: 4291.1599904      best: 4291.1599904 (209)        total: 2.8s     remaining: 534ms
210:    learn: 3688.5140968     test: 4288.4916059      best: 4288.4916059 (210)        total: 2.82s    remaining: 521ms
211:    learn: 3686.2703221     test: 4285.9722831      best: 4285.9722831 (211)        total: 2.83s    remaining: 507ms
212:    learn: 3679.7828635     test: 4284.3174402      best: 4284.3174402 (212)        total: 2.84s    remaining: 494ms
213:    learn: 3677.4227806     test: 4282.3576595      best: 4282.3576595 (213)        total: 2.86s    remaining: 481ms
214:    learn: 3672.6907165     test: 4278.7113929      best: 4278.7113929 (214)        total: 2.87s    remaining: 467ms
215:    learn: 3667.2899862     test: 4277.8750415      best: 4277.8750415 (215)        total: 2.88s    remaining: 454ms
216:    learn: 3661.0243113     test: 4274.2241038      best: 4274.2241038 (216)        total: 2.9s     remaining: 441ms
217:    learn: 3658.7163561     test: 4272.0864194      best: 4272.0864194 (217)        total: 2.91s    remaining: 428ms
```

[62]

```
#evaluation metrics
predcbt = cbt.predict(X_test)
msecbt = (mean_squared_error(Y_test, predcbt))
maecbt = mean_absolute_error(Y_test, predcbt)
r2cbt = r2_score(Y_test, predcbt)

print(msecbt)
print(maecbt)
print(r2cbt)
```

```
20457770.70380485
2596.579351178401
0.851610217534246
```

3.5 XGBoost

XGBoost

[63]

```
#XGBoost

xgb=XGBRegressor()
xgb.fit(x_train,y_train)
predxgb=xgb.predict(x_test)
print(r2_score(y_test,predxgb)*100)
```

```
77.84917476352999
```

[64]

```
#checking error for GradientBoosting

msexgb=mean_squared_error(y_test,predxgb)
maexgb=mean_absolute_error(y_test,predxgb)
r2xgb=r2_score(y_test,predxgb)
print(msexgb)
print(maexgb)
print(r2xgb)
```

```
31866668.59979161
3692.6774275065104
0.7784917476352998
```

3.6 Neural Network

NN

[65]

```python
import math
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError

```

[66]

```python
# define model
model1 = Sequential()
model1.add(Dense(128, activation='relu'))
model1.add(Dense(64, activation='relu'))
model1.add(Dense(32, activation='relu'))
# model1.add(Dense(8, activation='relu'))
model1.add(Dense(1, activation='linear'))

```

[67]

```python
# loss function
# msle = MeanSquaredLogarithmicError()

model1.compile(
    loss='mse',
    optimizer=Adam(learning_rate=0.005),
    metrics=['mae']
)
```

[68]

```python
# train the model
history = model1.fit(
    x_train,
    y_train,
    epochs=25,
    batch_size=64,
    validation_split=0.2
)
```

```
Epoch 1/25
438/438 [==============================] - 8s 5ms/step - loss: 137140752.0000 - mae: 6966.0171 - val_loss: 30577726.0000 - val_mae: 3406.1130
Epoch 2/25
438/438 [==============================] - 3s 6ms/step - loss: 30799670.0000 - mae: 2990.4158 - val_loss: 22988992.0000 - val_mae: 2995.0996
Epoch 3/25
438/438 [==============================] - 2s 4ms/step - loss: 24534156.0000 - mae: 2720.6313 - val_loss: 19042686.0000 - val_mae: 2615.3176
Epoch 4/25
438/438 [==============================] - 2s 4ms/step - loss: 21824194.0000 - mae: 2598.6443 - val_loss: 17809506.0000 - val_mae: 2532.5300
```

[69]
```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               163456

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 173,825
Trainable params: 173,825
Non-trainable params: 0
_________________________________________________________________
```

[70]
```
prednn= model1.predict(x_test)
prednn

```

```
469/469 [==============================] - 1s 2ms/step
array([[25024.29 ],
       [16655.719],
       [30460.64 ],
       ...,
       [ 7312.421],
       [20297.074],
       [29923.604]], dtype=float32)
```

[71]
```
msenn, maenn = model1.evaluate(x_test,y_test)
msenn, maenn
r2nn=r2_score(y_test,prednn)
```

```
469/469 [==============================] - 1s 2ms/step - loss: 20280962.0000 - mae: 2464.4556
```

[72]
```
print(msenn, maenn, r2nn)
```

```
20280962.0 2464.45556640625 0.8590250582285328
```

3.7 Best Model?

[73]

```
##ALL models used
##                                   Lasso, Ridge, Random Forest, Catboost, XGBoost, NN
results = pd.DataFrame({'Mean Squared Error':[msels,mselrr,mserf,msecbt,msexgb,msenn],
                        'Mean Absolute Error':[maels,maelrr,maerf,maecbt,maexgb,maenn],
                        'Accuracy Score':[r2ls*100,r2lrr*100,r2rf*100,r2cbt*100,r2xgb*100,r2nn*100]},
                       index=['Lasso','Ridge','RandomForestRegressor','CatBoostRegressor','XGBRegressor','NeauralNetworkRegressor'])
result_df = results.sort_values(by='Accuracy Score', ascending=False)
result_df
```

	Mean Squared Error	Mean Absolute Error	Accuracy Score
NeauralNetworkRegressor	2.028096e+07	2464.455566	85.902506
CatBoostRegressor	2.045777e+07	2596.579351	85.161022
Ridge	2.631458e+07	2947.848787	81.708483
RandomForestRegressor	2.817440e+07	2880.621816	80.415709
Lasso	2.891037e+07	3200.514512	79.904126
XGBRegressor	3.186667e+07	3692.677428	77.849175

3.8 Random forest in Pyspark

In [12]:
```
from pyspark.ml.linalg import Vector
from pyspark.ml.feature import VectorAssembler
```

In [13]:
```
df.columns
```

Out[13]: ['price', 'year', 'mileage', 'make', 'model']

In [16]:
```
# Preprocessing: StringIndexer for categorical labels

from pyspark.ml.feature import StringIndexer


stringIndexer1  = StringIndexer(inputCol="model", outputCol="model_label")


model=stringIndexer1.fit(df)
indexed1=model.transform(df)

stringIndexer2  = StringIndexer(inputCol="make", outputCol="make_label")
model2=stringIndexer2.fit(indexed1)
indexed2=model2.transform(indexed1)
```

In [17]:
```
from pyspark.ml.linalg import Vector
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=['year', 'mileage', 'make_label', 'model_label'], outputCol="features")
```

In [18]:
```
indexed2.show()
```

```
+-----+----+-------+-----+------------+-----------+----------+
|price|year|mileage| make|       model|model_label|make_label|
+-----+----+-------+-----+------------+-----------+----------+
| 8995|2013|  48851|Acura|  ILX6-Speed|     1611.0|      22.0|
|13995|2013|  32384|Acura|  TSX5-Speed|      726.0|      22.0|
|10495|2013|  57596|Acura|  ILX6-Speed|     1611.0|      22.0|
|12921|2012|  58550|Acura|TSXAutomatic|      646.0|      22.0|
|17628|2015|  13797|Acura|  ILX5-Speed|      300.0|      22.0|
|13999|2013|  35035|Acura|  TSX5-Speed|      726.0|      22.0|
|14995|2014|  23454|Acura|  ILX5-Speed|      300.0|      22.0|
|14990|2015|  23603|Acura|  ILX5-Speed|      300.0|      22.0|
|14590|2010|  19250|Acura|      TSX4dr|      431.0|      22.0|
| 9500|2011|  68289|Acura|      TSX4dr|      431.0|      22.0|
|16994|2015|  23946|Acura|  ILX5-Speed|      300.0|      22.0|
|15499|2014|  27171|Acura|  TSX5-Speed|      726.0|      22.0|
|13499|2014|  35037|Acura|  ILX5-Speed|      300.0|      22.0|
|14999|2014|  17669|Acura|  ILX5-Speed|      300.0|      22.0|
|14500|2010|  25926|Acura|      TSX4dr|      431.0|      22.0|
|16000|2015|  30881|Acura|  ILX5-Speed|      300.0|      22.0|
|17419|2015|  15390|Acura|  ILX5-Speed|      300.0|      22.0|
|14999|2015|  27333|Acura|  ILX5-Speed|      300.0|      22.0|
|14999|2015|  28326|Acura|  ILX5-Speed|      300.0|      22.0|
|17000|2015|  24671|Acura|  ILX5-Speed|      300.0|      22.0|
+-----+----+-------+-----+------------+-----------+----------+
only showing top 20 rows
```

In [19]:
```
output=assembler.transform(indexed2)
```

In [20]:
```
output.show(truncate=False)
```

```
+-----+----+-------+-----+------------+-----------+----------+---------------------------+
|price|year|mileage|make |model       |model_label|make_label|features                   |
+-----+----+-------+-----+------------+-----------+----------+---------------------------+
|8995 |2013|48851  |Acura|ILX6-Speed  |1611.0     |22.0      |[2013.0,48851.0,22.0,1611.0]|
|13995|2013|32384  |Acura|TSX5-Speed  |726.0      |22.0      |[2013.0,32384.0,22.0,726.0] |
|10495|2013|57596  |Acura|ILX6-Speed  |1611.0     |22.0      |[2013.0,57596.0,22.0,1611.0]|
|12921|2012|58550  |Acura|TSXAutomatic|646.0      |22.0      |[2012.0,58550.0,22.0,646.0] |
|17628|2015|13797  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,13797.0,22.0,300.0] |
|13999|2013|35035  |Acura|TSX5-Speed  |726.0      |22.0      |[2013.0,35035.0,22.0,726.0] |
|14995|2014|23454  |Acura|ILX5-Speed  |300.0      |22.0      |[2014.0,23454.0,22.0,300.0] |
|14990|2015|23603  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,23603.0,22.0,300.0] |
|14590|2010|19250  |Acura|TSX4dr      |431.0      |22.0      |[2010.0,19250.0,22.0,431.0] |
|9500 |2011|68289  |Acura|TSX4dr      |431.0      |22.0      |[2011.0,68289.0,22.0,431.0] |
|16994|2015|23946  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,23946.0,22.0,300.0] |
|15499|2014|27171  |Acura|TSX5-Speed  |726.0      |22.0      |[2014.0,27171.0,22.0,726.0] |
|13499|2014|35037  |Acura|ILX5-Speed  |300.0      |22.0      |[2014.0,35037.0,22.0,300.0] |
|14999|2014|17669  |Acura|ILX5-Speed  |300.0      |22.0      |[2014.0,17669.0,22.0,300.0] |
|14500|2010|25926  |Acura|TSX4dr      |431.0      |22.0      |[2010.0,25926.0,22.0,431.0] |
|16000|2015|30881  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,30881.0,22.0,300.0] |
|17419|2015|15390  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,15390.0,22.0,300.0] |
|14999|2015|27333  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,27333.0,22.0,300.0] |
|14999|2015|28326  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,28326.0,22.0,300.0] |
|17000|2015|24671  |Acura|ILX5-Speed  |300.0      |22.0      |[2015.0,24671.0,22.0,300.0] |
+-----+----+-------+-----+------------+-----------+----------+---------------------------+
only showing top 20 rows
```


In [21]:
```
train_data, test_data = output.randomSplit([0.8, 0.2], seed=123)
```

In [22]:
```
# Create a RandomForestRegressor
from pyspark.ml.regression import RandomForestRegressor
rf_regressor = RandomForestRegressor(featuresCol="features", labelCol="price", numTrees=150, maxBins=3000)
```

In [23]:
```
rf_model = rf_regressor.fit(train_data)
```

In [24]:
```
predictions = rf_model.transform(test_data)
```

In [25]:
```
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Root Mean Squared Error (RMSE): 6353.793822417411
```

In [26]:
```
# R-squared (R2)
r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})
print("R-squared (R2):", r2)
```

```
R-squared (R2): 0.7792687452839657
```

4. Transformation on Pyspark

We have applied narrow and wide transformations on Pyspark and saved those respective dataframes into hdfs in csv file format

transformations

```
In [29]: df=spark.read.table("cars")
```

```
In [41]: df=df.dropna()
```

```
In [42]: df1=df.orderBy(df.city.asc(),df.price.asc())
```

```
In [44]: df1.show() #cities and price sorted in ascending order
```

```
+-----+----+-------+----------+-----+-----------------+-------------+---------------+
|price|year|mileage|      city|state|              vin|         make|          model|
+-----+----+-------+----------+-----+-----------------+-------------+---------------+
|10339|2011| 102333|     AKRON|   OH|WDDGF8BB2BR150111|Mercedes-Benz|     C-Class4dr|
|15889|2011|  92812|     AKRON|   OH|4JGBB8GB0BA681651|Mercedes-Benz|  M-Class4MATIC|
|16999|2008|  39493|     AKRON|   OH|4JGBB22EX8A421443|Mercedes-Benz|     M-Class4WD|
|17889|2011|  72832|     AKRON|   OH|WDDHF8HB4BA501126|Mercedes-Benz|     E-Class4dr|
|23876|2013|  39961|     AKRON|   OH|WDCGG8JB1DG016238|Mercedes-Benz|GLK-ClassGLK350|
|25229|2014|  66378|     AKRON|   OH|WDCGG8JB4EG326238|Mercedes-Benz|GLK-ClassGLK350|
|26449|2004|  25403|     AKRON|   OH|WDBSK75F24F071548|Mercedes-Benz|  SL-ClassSL500|
|33959|2014|  35461|     AKRON|   OH|4JGDA5HB1EA353653|Mercedes-Benz|   M-ClassML350|
|35889|2015|  29610|     AKRON|   OH|WDCGG8JB8FG378795|Mercedes-Benz|GLK-ClassGLK350|
| 4995|2004| 187317|ALEXANDRIA|   VA|2HKYF18534H553810|        Honda|        PilotEX|
| 5795|2005| 180159|ALEXANDRIA|   VA|1HGCM66565A061282|        Honda|         Accord|
| 5995|2005| 106412|ALEXANDRIA|   VA|JM1BK343151250325|        Mazda|      Mazda35dr|
| 6795|2006|  54181|ALEXANDRIA|   VA|YV1MS390862216864|        Volvo|        S402.4L|
| 8995|2012|  53308|ALEXANDRIA|   VA|1FAHP3N28CL408507|         Ford|       Focus5dr|
| 8995|2010|  26345|ALEXANDRIA|   VA|KNAFW6A30A5256595|          Kia|          Forte|
| 8995|2012|  83654|ALEXANDRIA|   VA|1YVHZ8DH7C5M35879|        Mazda|      Mazda64dr|
| 9995|2013|  20871|ALEXANDRIA|   VA|1FADP3K21DL231433|         Ford| FocusHatchback|
|11795|2014| 104369|ALEXANDRIA|   VA|JM3KE2CY6E0350691|        Mazda|    CX-5Touring|
|11995|2015|  27124|ALEXANDRIA|   VA|3FADP4CJXFM216202|         Ford|    FiestaSedan|
|12990|2008|  75210|ALEXANDRIA|   LA|WBAWB33548P135043|          BMW|              3|
+-----+----+-------+----------+-----+-----------------+-------------+---------------+
only showing top 20 rows
```

```
In [56]: df1.coalesce(1).write.format("csv").mode('overwrite').save("/user/talentum/tfm1")
```

```
In [43]: df2=df.groupBy("year").count()

In [47]: df2.sort(df.year.desc()).show() #yearly count of cars sold in descending order

+----+------+
|year| count|
+----+------+
|2018|   922|
|2017| 91608|
|2016|132136|
|2015|157516|
|2014|162432|
|2013| 74701|
|2012| 49764|
|2011| 39768|
|2010| 27539|
|2009| 19061|
|2008| 24713|
|2007| 21171|
|2006| 15079|
|2005| 11005|
|2004|  8117|
|2003|  5649|
|2002|  3800|
|2001|  2584|
|2000|  1933|
|1999|  1254|
+----+------+
only showing top 20 rows

In [57]: df2.coalesce(1).write.format("csv").mode('overwrite').save("/user/talentum/tfm2")

In [50]: df3=df.groupBy("make").count()
         df3.show()

+-----------+------+
|       make| count|
+-----------+------+
| Volkswagen| 23249|
| Oldsmobile|   122|
|         AM|    19|
|      Lexus| 20641|
|     Jaguar|  2200|
|     Saturn|   963|
|       FIAT|  1782|
|   Maserati|  1047|
|Rolls-Royce|    92|
|      Scion|  3043|
|       Jeep| 40373|
| Mitsubishi|  4080|
|        Kia| 28636|
|  Chevrolet|102268|
|      Volvo|  5106|
|    Hyundai| 35837|
|       Saab|   260|
|      Honda| 50193|
|   INFINITI| 12258|
|       MINI|  4375|
+-----------+------+
only showing top 20 rows

In [58]: df3.coalesce(1).write.format("csv").mode('overwrite').save("/user/talentum/tfm3")
```

```
In [62]: df4=df.filter(df.city=='El Paso').groupBy("make").count()
```

```
In [63]: df4.show()
```

```
+----------+-----+
|      make|count|
+----------+-----+
|Volkswagen|  100|
|     Lexus|   34|
|    Jaguar|    5|
|    Saturn|    5|
|      FIAT|   12|
|  Maserati|    1|
|     Scion|   12|
|      Jeep|  193|
|Mitsubishi|   15|
|       Kia|   77|
| Chevrolet|  475|
|     Volvo|    2|
|   Hyundai|  138|
|     Honda|  184|
|  INFINITI|   31|
|      MINI|   19|
|      Audi|   18|
|       Ram|   66|
|  Cadillac|   56|
|   Pontiac|    6|
+----------+-----+
only showing top 20 rows
```

```
In [64]: df4.coalesce(1).write.format("csv").mode('overwrite').save("/user/talentum/tfm4")
```

```
talentum@talentum-virtual-machine:~$ hdfs dfs -ls
Found 5 items
drwxr-xr-x   - talentum supergroup          0 2023-08-18 21:29 cars_all
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:17 tfm1
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:17 tfm2
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:17 tfm3
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:20 tfm4
```

```
talentum@talentum-virtual-machine:~$ hdfs dfs -ls -R
drwxr-xr-x   - talentum supergroup          0 2023-08-18 21:29 cars_all
-rw-r--r--   1 talentum supergroup   56287873 2023-08-18 21:29 cars_all/true_car_listings.csv
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:17 tfm1
-rw-r--r--   1 talentum supergroup          0 2023-08-30 08:17 tfm1/_SUCCESS
-rw-r--r--   1 talentum supergroup   54597868 2023-08-30 08:17 tfm1/part-00000-8140de3a-1141-4d8b-bb6c-2ccc7d2624eb-c000.csv
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:17 tfm2
-rw-r--r--   1 talentum supergroup          0 2023-08-30 08:17 tfm2/_SUCCESS
-rw-r--r--   1 talentum supergroup        696 2023-08-30 08:17 tfm2/part-00000-5d8f28b0-91cc-4acc-aa21-9dc6f576dc24-c000.csv
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:17 tfm3
-rw-r--r--   1 talentum supergroup          0 2023-08-30 08:17 tfm3/_SUCCESS
-rw-r--r--   1 talentum supergroup        696 2023-08-30 08:17 tfm3/part-00000-9be698f4-de65-4988-bea9-3930909e8d7c-c000.csv
drwxr-xr-x   - talentum supergroup          0 2023-08-30 08:20 tfm4
-rw-r--r--   1 talentum supergroup          0 2023-08-30 08:20 tfm4/_SUCCESS
-rw-r--r--   1 talentum supergroup        380 2023-08-30 08:20 tfm4/part-00000-d7f80cd5-fb65-4e6c-b291-113d0bf9fcb2-c000.csv
```

```
talentum@talentum-virtual-machine:~$ hdfs dfs -ls tfm3
Found 2 items
-rw-r--r--   1 talentum supergroup          0 2023-08-30 08:17 tfm3/_SUCCESS
-rw-r--r--   1 talentum supergroup        696 2023-08-30 08:17 tfm3/part-00000-9be698f4-de65-4988-bea9-3930909e8d7c-c000.csv
talentum@talentum-virtual-machine:~$ hdfs dfs -cat tfm3/part-00000-9be698f4-de65-4988-bea9-3930909e8d7c-c000.csv
Volkswagen,23249
Oldsmobile,122
AM,19
Lexus,20641
Jaguar,2200
Saturn,963
FIAT,1782
Maserati,1047
Rolls-Royce,92
Scion,3043
Jeep,40373
Mitsubishi,4080
Kia,28636
Chevrolet,102268
Volvo,5106
Hyundai,35837
Saab,260
Honda,50193
INFINITI,12258
MINI,4375
Audi,12618
Lamborghini,121
Ram,19808
Cadillac,15047
Isuzu,76
Plymouth,51
```

5. Analysis on Tableau

The analysis of data was performed on

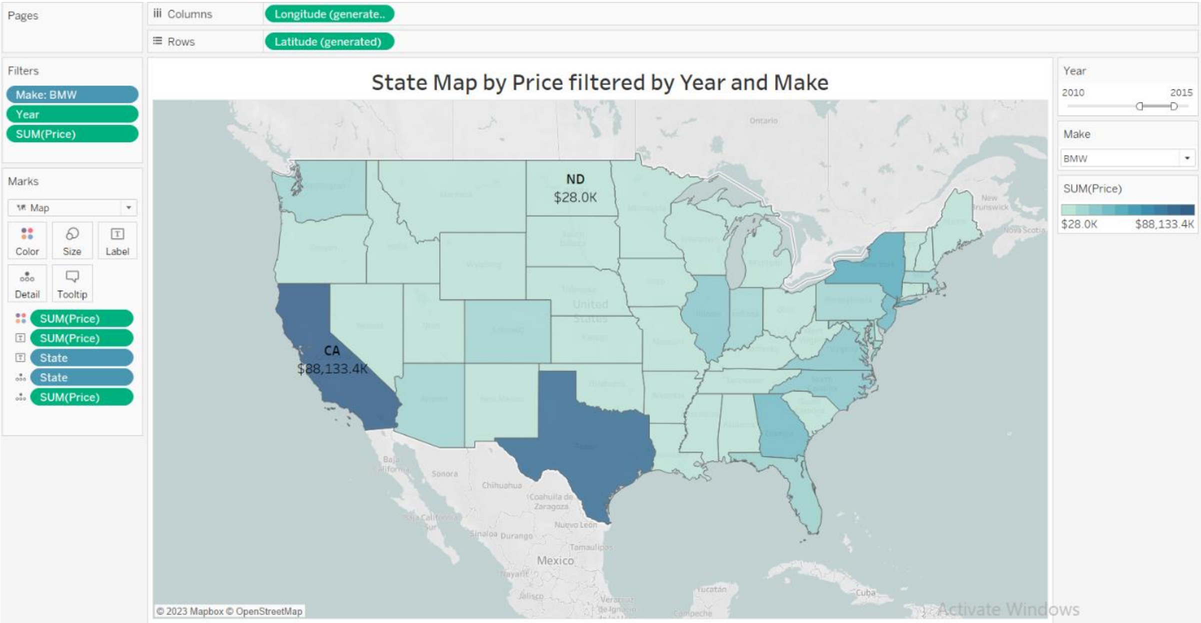


Fig 5.1 State map by price filtered by Year and Make

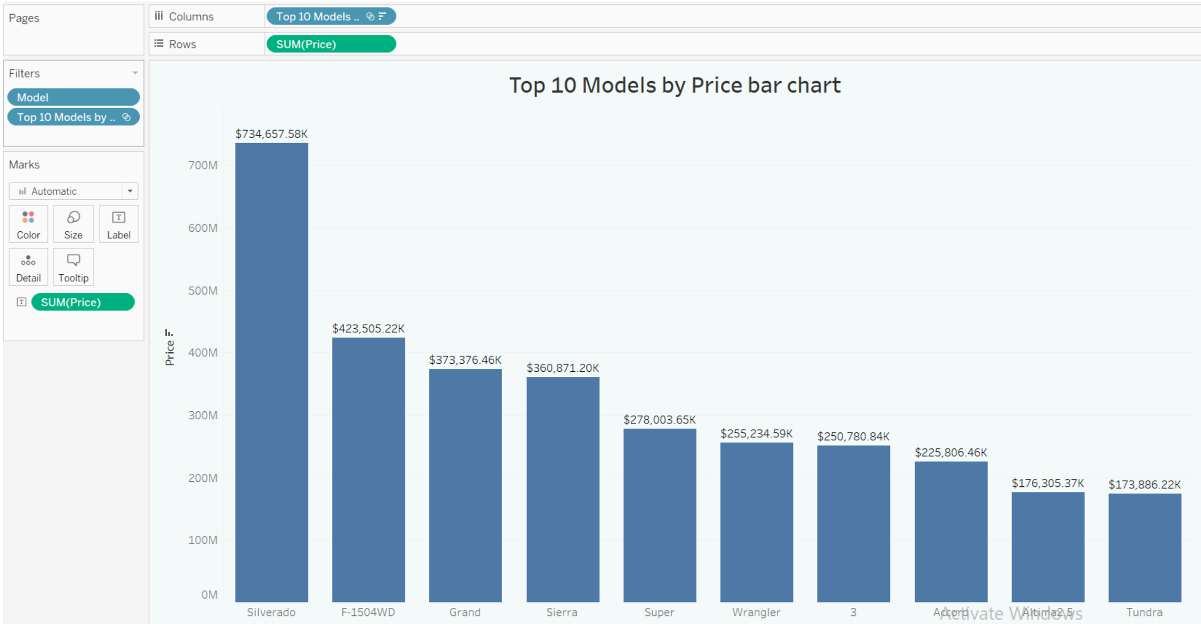


Fig 5.2 Top 10 Models by Price bar chart

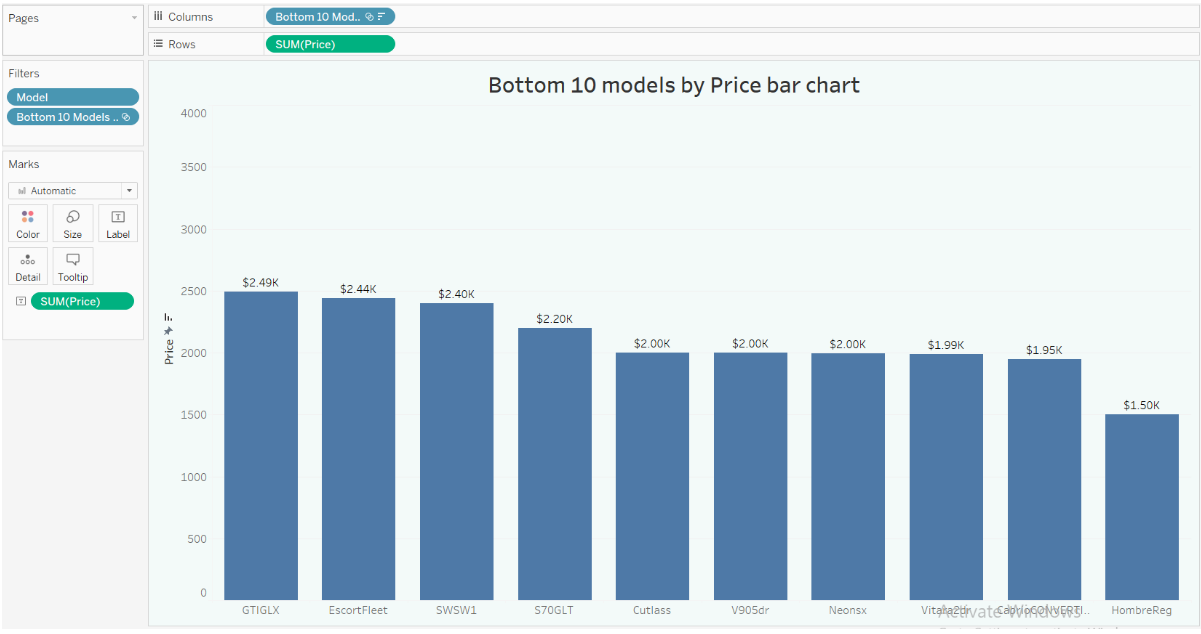


Fig 5.3 Bottom 10 Models by Price bar chart

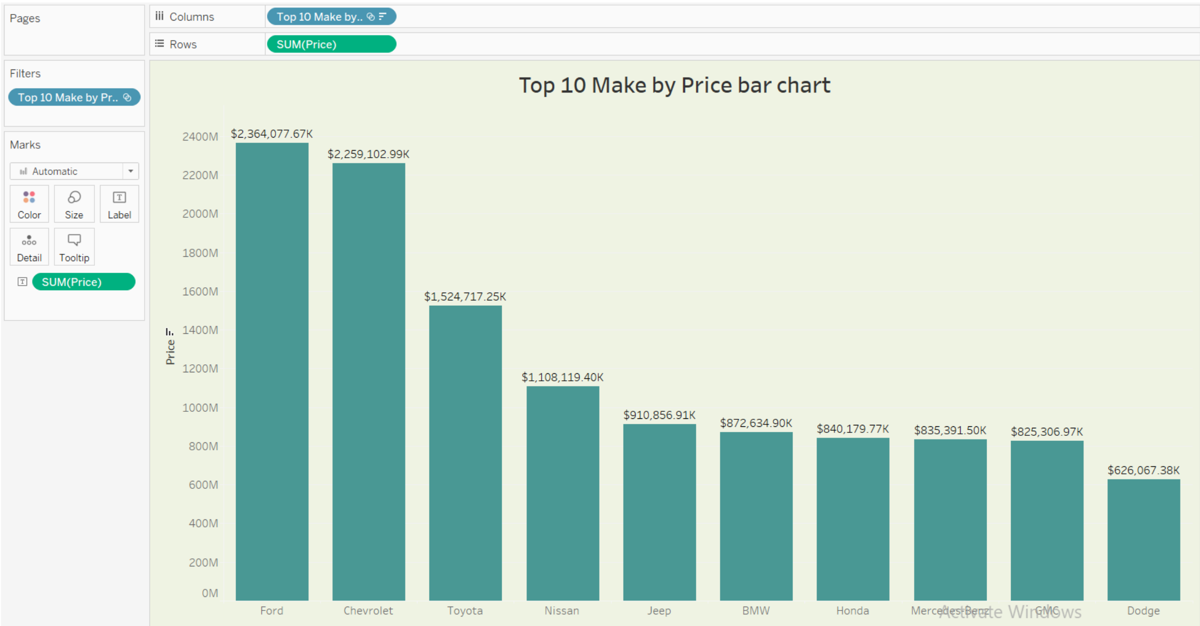


Fig 5.4 Top 10 Make by Price bar chart

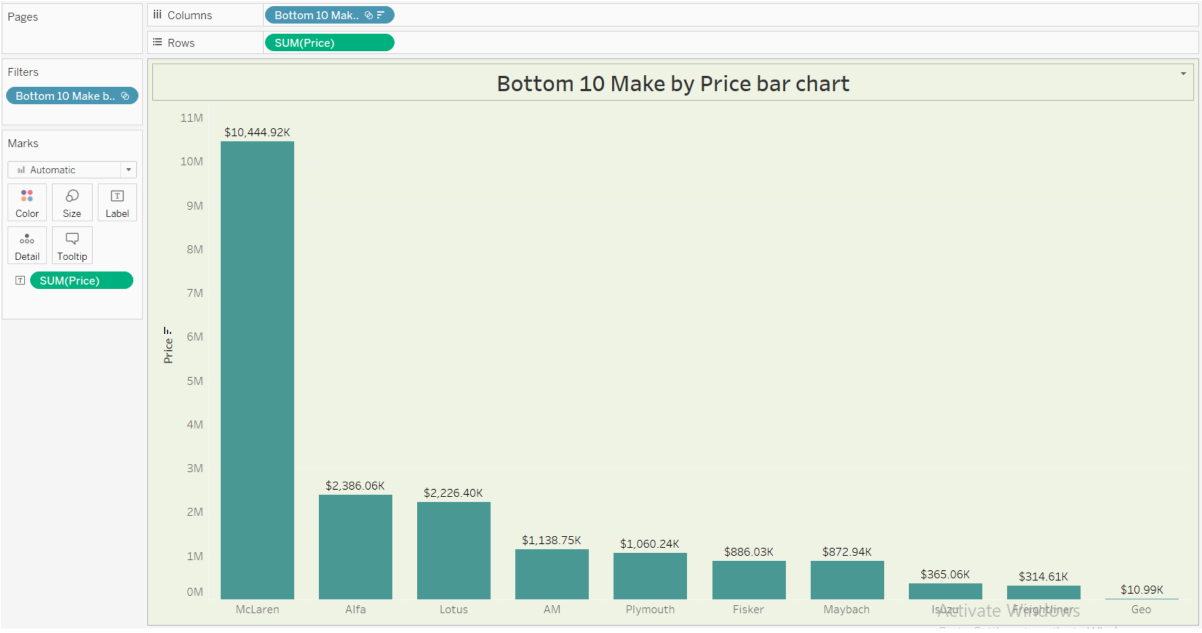


Fig 5.5 Bottom 10 Make by Price bar chart

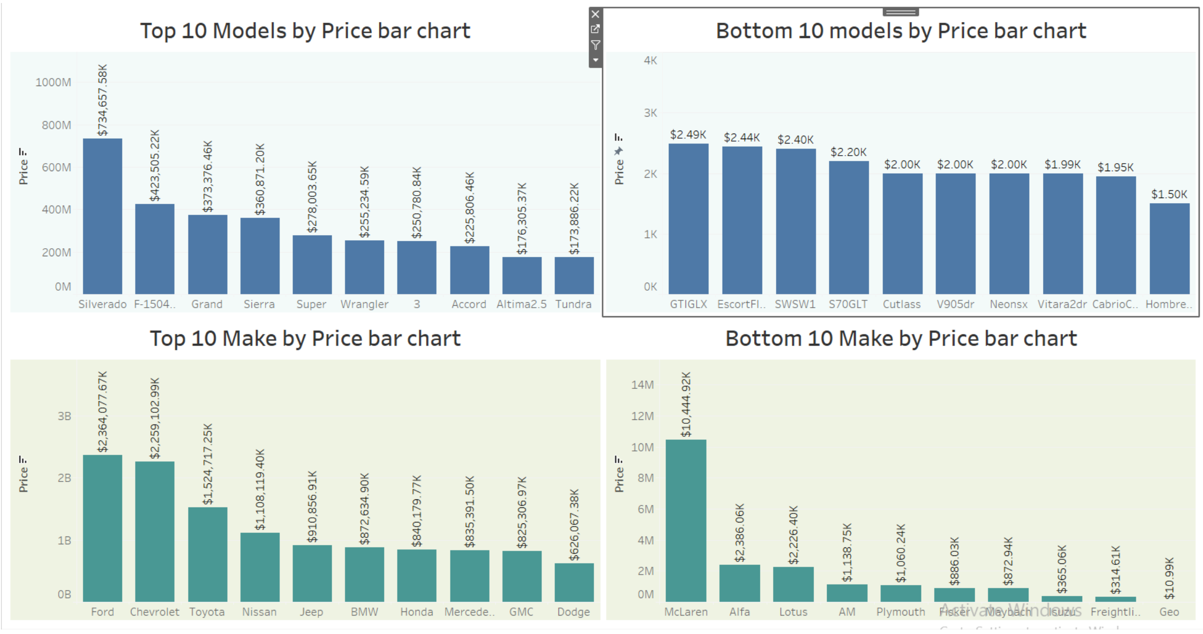


Fig 5.6 Dashboard of Top and Bottom features

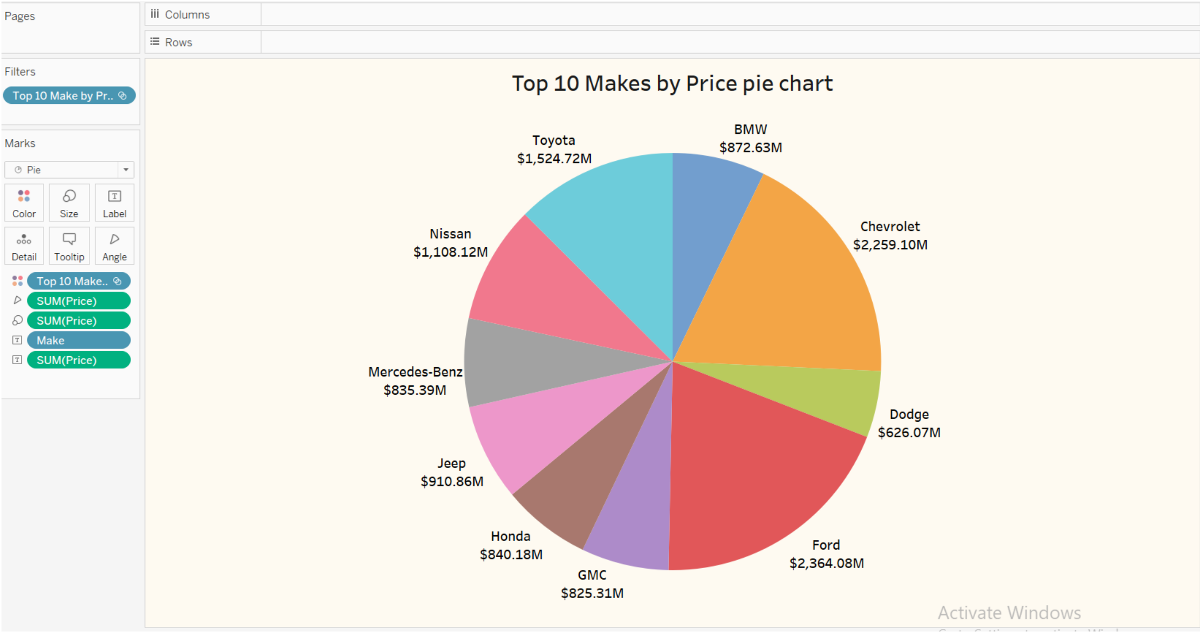


Fig 5.7 Top 10 Makes by Price pie chart

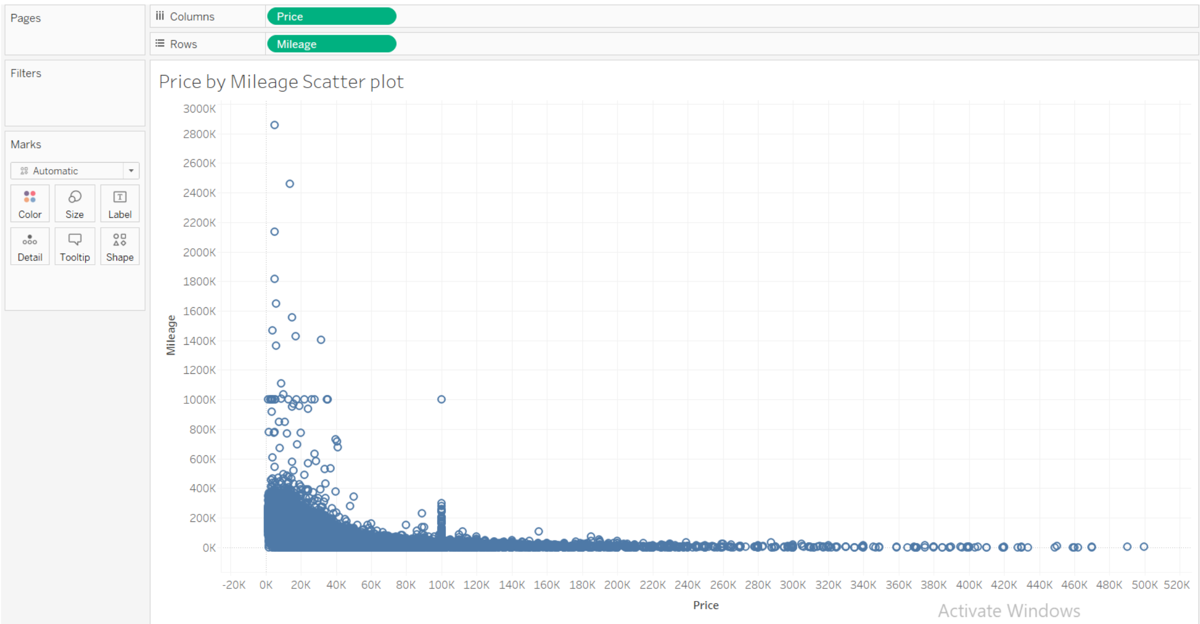


Fig 5.8 Price by Mileage scatter plot

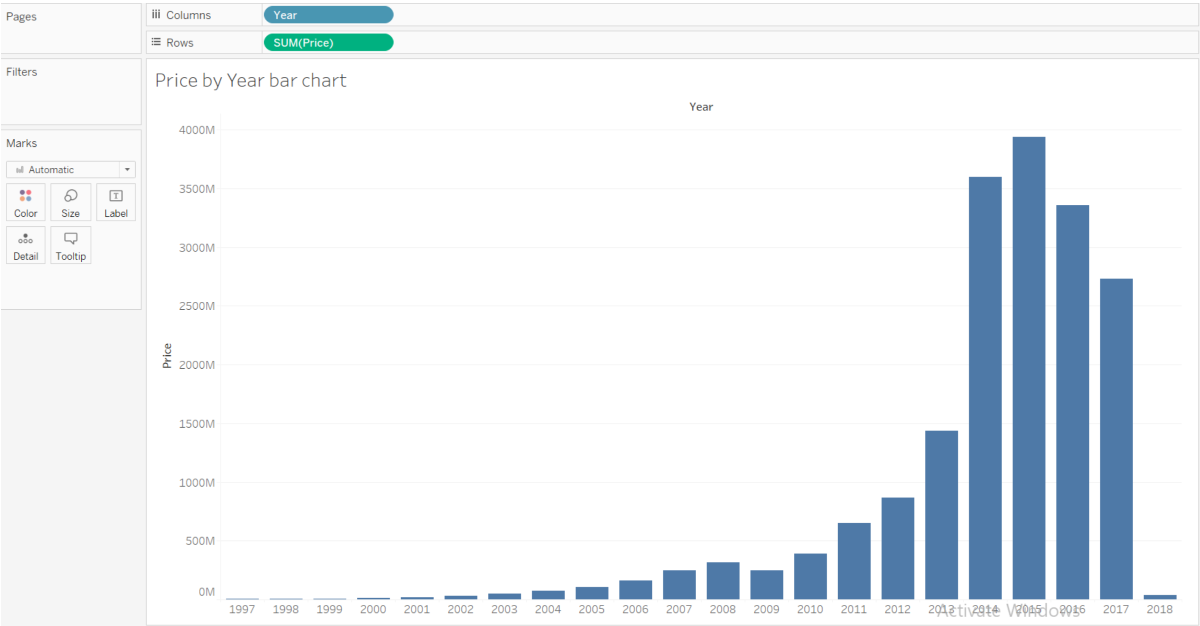


Fig 5.9 Price by Year bar chart

6. Conclusion

The objective of this project is to fit models to predict the used cars price and find some important aspects of the used cars. In order to achieve this goal, we have cleaned and preprocessed the data, applied EDA and Feature selection, PCA for dimensionality reduction and then fit five different regression models to the dataset: lasso regression, ridge regression, random forest, catboost, xgboost and neural network.

The best two models were:

Neural network performed the best amongst all models with 3 Dense layers and activation function as RELu. Catboost had dataset split into train, validation and split parts . With further fine tuning there is a chance to find the best possible parameters for regression and improve the models further.

We have also performed ETL using big data technologies and performed transformation on Pyspark and stored them to HDFS.

In addition to Prediction of Prices we have also performed Analysis on our data on Tableau.

7. References

- https://www.geeksforgeeks.org/principal-component-analysis-with-python/
- https://towardsdatascience.com/catboost-regression-in-6-minutes-3487f3e5b329
- https://www.analyticsvidhya.com/blog/2021/08/a-walk-through-of-regression-analysis-using-artificial-neural-networks-in-tensorflow/
- https://medium.com/vedity/python-data-preprocessing-using-pyspark-cc3f709c3c23
- https://www.machinelearningplus.com/pyspark/pyspark-random-forest/