Illinois Institute of Technology

Project Report

CSP554: Big Data Technologies

WORKING WITH APACHE SPARK MLLIB

Instructed by : Prof. Joseph Rosen

Submitted by :

Amogh Kori A20491465
Kajol Tanesh Shah A20496724
Vikas Pathak A20460927

INDEX

ABSTRACT

Transportation services have become core community necessities for both daily activities and travel. The aim of our project is to predict the selling price of a car based on the given features by using Big Data Technologies. For this, we have applied different machine learning algorithms to our dataset by using Apache Spark and MLlib. Apache Spark is an analytics engine for large-scale data processing with built-in modules for SQL, streaming, machine learning, and graph processing.

So, in this project, we have worked on a large car dataset containing many different features which are important and play an important role while buying/selling cars. We have visualized various parameters of this dataset to get a better understanding of the data and have built machine learning models using Apache Spark and MLLib to predict the outcomes.

OBJECTIVE

We aim to study the large car dataset of new and used car listings from dealers and get meaningful information using Apache Spark and ML libraries . Further, the model will be a good way for management to understand the pricing dynamics of a new market.


[4]

SPECIFIC QUESTIONS

Along with the problem statement, we have addressed the below questions to our best of ability

Question 1: How Long Before a Car is Sold?
Question 2: Which Cars are the Most Popular?
Question 3: What Color of the car do people like the most?
Question 4: How Much will the Final Price Differ from MSRP on Average?

INTRODUCTION

Different available big data machine learning systems are:-

H2O

H2O is a distributed, in-memory machine learning platform that is open-source and has linear scalability. H2O has AutoML functionality in addition to supporting the statistical and machine learning algorithms that are utilized the most frequently. Along with a built-in web interface, the platform has interfaces for R, Python, Scala, Java, JSON, and CoffeeScript/JavaScript.

SparkR

SparkR is a lightweight frontend for Apache Spark that can be used with R. SparkR also uses MLlib to support distributed machine learning.

AWS Sagemaker

Using Amazon SageMaker, which is a cloud-based machine learning platform, developers can develop, train, and deploy ML models in the cloud. SageMaker also makes it possible for developers to deploy ML models on edge devices and embedded systems.

Apache Spark

For large-scale data processing, Apache Spark is a fast and general-purpose cluster computing system. It is well-known for its speed, generality, ease of use, and platform compatibility. It executes programs in memory up to 100 times faster than Hadoop MapReduce and on disk up to ten times faster. An advanced DAG (Direct Acyclic Graph) execution engine in Apache Spark enables in-memory computing and acyclic data flow. It provides over 80 high-level operators, making parallel app development simple. It is a large-scale analytics engine with built-in SQL, streaming, machine learning, and graph processing modules.

Apache Spark MLlib

MLlib is Apache Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. Its features include great performance, ease of deployment and ease of usage. It is usable in Java, Scala, Python and R.

PySpark

PySpark is the Python API for Spark.

It includes the following Public classes:

- SparkContext: Main entry point for Spark functionality.
- RDD: A Resilient Distributed Dataset (RDD), the basic abstraction in Spark.
- Broadcast: A broadcast variable that gets reused across tasks.
- Accumulator: An "add-only" shared variable that tasks can only add values to.
- SparkConf: For configuring Spark.
- SparkFiles: Access files shipped with jobs.
- StorageLevel: Finer-grained cache persistence levels.
- TaskContext: Information about the current running task, available on the workers and experimental.

It contains the following sub-packages:

- pyspark.sql module

- pyspark.streaming module
- pyspark.ml package
- pyspark.mllib package

Jupyter

Jupyter notebook is accessed through our browser and can be launched locally on our machine as an area server or remotely on a server. The name notebook comes from the fact that it can contain live code, rich text elements such as equations, links, photos, tables, and so on. As a result, you may have a very good notebook to discuss your idea as well as the live code dead one document. For example, we have written major code parts entirely within the Jupyter notebook. The different platform and library tools like Pyspark, Scikit Learn, Numpy, Pandas, Seaborn are used for data analysis for more flexibility.

ML Algorithms:

Common machine learning algorithms such as classification, regression, clustering, and collaborative filtering.

Numpy:

Numpy is a math library to figure with n-dimensional arrays in Python. It enables you to try and do computation efficiently and effectively. It's better than regular python due to its amazing capabilities.. NumPy is installed by using the ‘pip install numpy’ command.

Matplotlib:

Matplotlib is a plotting package that provides 2D plotting as well as 3D plotting. Matplotlib is installed by using the ‘pip install matplotlib ‘command.

Pandas:

Pandas library could be a very high-level python library that has high-performance, easy to use data structures. it's many functions for data importing, manipulation and analysis. specifically, it offers

data structures and operations for manipulating numerical tables and statistics. Pandas is installed by using the ‘pip install pandas’ command.

Seaborn:
Seaborn could be a Python data visualization library supported by matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. For a quick introduction to the ideas behind the library, you'll be able to read the introductory notes. Seaborn is installed by using the ‘pip install seaborn’ command.

WHY APACHE SPARK AND MLLIB?

In our project, we have used Apache Spark and MLLib. For writing our code in Spark, we have used Jupyter Notebook. Working on large datasets on local systems comes with limitations like limited memory, lesser RAM, less processing speed, etc. and so, using Apache Spark is a good alternative. Also, the reason we have selected Apache Spark and MLLib over other big data machine learning systems like H2O, AWS Sagemaker, etc. is because writing code and building models in PySpark is similar to that in Python and as we have experience in Python, it was easier for us to understand PySpark syntaxes and write the code. Also, we have used Jupyter notebook over other coding platforms because it is great for showcasing our work as we can see both our code and the results. We can run our code cell by cell and get a good understanding of how our code is running.

DATA PROCESSING

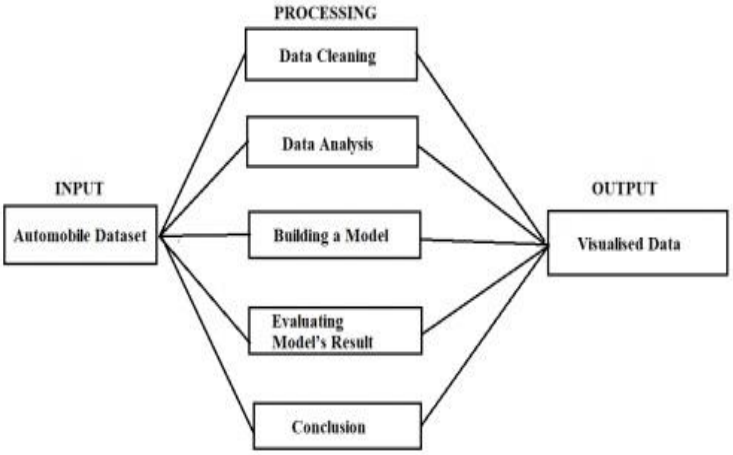


Figure: Workflow of our Machine Learning Model

Data Properties:

The data we utilized for this research came from Kaggle; the large car dataset is available at https://www.kaggle.com/datasets/cisautomotiveapi/large-car-dataset

The data is extractedWH from AutoDealerData.com, and it contains roughly 5.7 million entries pertaining to new and used vehicle listings.

The data set includes information about a lot of different types of columns. Here are listed some of key columns:

Field	DataType	Description
vin	chr	a hash of the vehicle identification number. The same vin will map to the same hash.
color	chr	exterior color
stockNum	chr	Dealer specific stock number
firstSeen	chr	Date we first saw the vehicle at the dealer. YYYY-MM-DD
lastSeen	chr	Date we last saw the vehicle at the dealer. YYYY-MM-DD

msrp	chr	Manufacturer's Suggested Retail Price
askPrice	chr	last ask price before the vehicle was sold
mileage	chr	mileage
isNew	chr	boolean. is a new car or used car.
interiorColor	chr	interiorColor
brandName	chr	brand of the vehicle, Chevrolet, Ford
modelName	chr	model of the vehicle, F-150, Silverado

In total, there are:

- 5,695,015 rows with 3,778,957 distinct count in each column
- 1475 distinct count of brandName and ModelName
- 32 interesting feature columns (mentioned below) which we have considered for our ML model

Vin - vehicle identification number
Firstseen
Lastseen
Msrp - target
Mileage
isNew
Color
Brandname
Modelname
vf_AirBagLocFront
vf_AirBagLocSide
vf_AirBagLocKnee
vf_Axles
vf_BasePrice

askPrice
vf_DisplacementCC
vf_DisplacementCI
vf_DisplacementL
vf_Doors
vf_EngineCylinders
vf_EngineHP
vf_EngineKW
vf_SeatRows
vf_Seats
vf_TopSpeedMPH
vf_TransmissionStyle
vf_TransmissionSpeeds
vf_WheelBaseShort
vf_WheelSizeFront
vf_WheelSizeRear
vf_Wheels
vf_Windows

Data Cleaning, Data Selection and Standardization

The above mentioned dataset is a big dataset and has many missing values. So, before we build a model for the dataset, we need to perform data cleaning to identify and fix errors and remove duplicates, inconsistencies and incorrect data from the dataset. There are various ways to deal with such data.

First we read the dataset into a PySpark dataframe

```
In [70]: import pyspark
         from pyspark.sql import SparkSession

In [72]: import findspark
         findspark.init('/usr/local/Cellar/apache-spark/2.2.0/libexec')

In [75]: spark = SparkSession.builder.appName('BigdataProject').getOrCreate()

In [76]: spark

Out[76]: SparkSession - in-memory
         SparkContext

         Spark UI
         Version
         v3.3.1
         Master
         local[*]
         AppName
         BigdataProject

In [77]: df = spark.read.option('header','true').csv('CIS_Automotive_Kaggle_Sample.csv',inferSchema=True)

In [80]: df.columns

Out[80]: ['vin',
          'stockNum',
          'firstSeen',
          'lastSeen',
          'msrp',
          'askPrice',
          'mileage',
          'isNew',
          'color',
          'interiorColor',
          'brandName',
          'modelName',
          'dealerID',
          'vf_ABS',
          'vf_ActiveSafetySysNote',
          'vf_AdaptiveCruiseControl',
          'vf_AdaptiveDrivingBeam',
          'vf_AdaptiveHeadlights',
          'vf_AdditionalErrorText',
```

As there are too many columns in the dataset, we have performed data selection and only selected the features that are important and relevant to our problem statement and dropped the others.

```
In [87]: df_pyspark = df.select(['vin','firstseen','lastseen','askPrice','mileage','isNew','brandName','modelName',
           'vf_AirBagLocFront','vf_AirBagLocKnee', 'vf_AirBagLocSide','vf_Axles','vf_BasePrice','vf_DisplacementCC','vf_Displace
           'vf_EngineCylinders','vf_EngineHP','vf_EngineKW','vf_SeatRows','vf_Seats','vf_TopSpeedMPH','vf_TransmissionSpeeds','v

In [88]: df_pyspark

Out[88]: DataFrame[vin: string, firstseen: timestamp, lastseen: timestamp, askPrice: int, mileage: int, isNew: boolean, brandN
         ame: string, modelName: string, vf_AirBagLocFront: string, vf_AirBagLocKnee: string, vf_AirBagLocSide: string, vf_Axl
         es: int, vf_BasePrice: double, vf_DisplacementCC: double, vf_DisplacementCI: double, vf_DisplacementL: double, vf_Doo
         rs: int, vf_EngineCylinders: int, vf_EngineHP: double, vf_EngineKW: double, vf_SeatRows: int, vf_Seats: int, vf_TopSp
         eedMPH: int, vf_TransmissionSpeeds: int, vf_WheelBaseShort: double, vf_WheelSizeFront: int, vf_WheelSizeRear: int, vf
         _Wheels: int, vf_Windows: int, msrp: int]
```

We have combined some features and created new columns and then dropped their original columns.

```
In [89]: #Q1
from pyspark.sql.functions import datediff,col,lit
df1 = df_pyspark.withColumn('Time in lot',datediff(df['lastSeen'],df['FirstSeen']))

In [95]: #dropping first seen and last seen columns as we have already added a new column 'Time in Lot' for it
df2 = df1.drop(col("firstseen"))
df2 = df2.drop(col("lastseen"))
```

In order to improve the quality of our data so that we get better accuracy with our ML model, we have dropped the rows which have missing values in some columns.

```
In [132]: from pyspark.sql.functions import mean

df2 = df2.na.drop(subset=['msrp','vf_BasePrice','askPrice'])

In [133]: df2.select('msrp','vf_BasePrice','askPrice').show()
+-----+------------+--------+
| msrp|vf_BasePrice|askPrice|
+-----+------------+--------+
|12387|     23475.0|   12387|
|12970|     26500.0|   12970|
|15218|     23940.0|   15218|
|18755|     38495.0|   18755|
|36999|     34175.0|   36999|
|18276|     24105.0|   18276|
|22140|     28000.0|   22140|
|    0|     34175.0|       0|
|20494|     30530.0|   20494|
|19026|     26500.0|   19026|
|18951|     26500.0|   18951|
|16649|     27995.0|   16649|
|16671|     30420.0|   16671|
|16998|     27995.0|   16998|
|20294|     30530.0|   20294|
|45996|     63000.0|   45996|
```

As our problem is a regression problem, we have converted some important columns of type string/boolean to integer.

```
In [99]: import pyspark.sql.functions as F
df3 = df2.withColumn('isNew', F.when(df2['isNew'] == 'FALSE', 0).otherwise(1))
df3.select('isNew').show()
+-----+
|isNew|
+-----+
|    0|
|    0|
|    0|
|    0|
|    0|
|    1|
|    1|
```

```
In [100]: #changing vf_AirBagLocFront from string to integer
          from pyspark.sql.functions import when
          df4 = df3.withColumn('vf_AirBagLocFront', (when(df3['vf_AirBagLocFront'] == '1st Row (Driver & Passenger)', 2).when(df3
          df4.select('vf_AirBagLocFront').show(45)

          +-----------------+
          |vf_AirBagLocFront|
          +-----------------+
          |                2|
          |                2|
          |                2|
          |                2|
          |                2|
          |                2|
```

Also, for handling other missing values, we have taken the average of that column and replaced missing values in that column with its average.

```
In [104]: #vf_Axles - filling null values with average
          avg_axles = df6.agg({'vf_Axles': 'mean' })
          df7 = df6.na.fill(value=avg_axles.first()[0],subset=["vf_Axles"])
          df7.select('vf_Axles').show()

In [106]: #vf_BasePrice - filling null values with average
          avg_bprice = df7.agg({'vf_BasePrice': 'mean' })
          df8 = df7.na.fill(value=int(avg_bprice.first()[0]),subset=["vf_BasePrice"])
          df8.select('vf_BasePrice').show()

In [107]: #vf_DisplacementCI, vf_DisplacementCC, vf_DisplacementL, vf_Doors, vf_EngineCylinders, vf_EngineHP, vf_EngineKW,
          #vf_SeatRows, vf_Seats, vf_TopSpeedMPH, vf_TransmissionSpeeds, vf_WheelBaseShort, vf_WheelSizeFront,
          #vf_WheelSizeRear, vf_Wheels, vf_Windows - filling null values with average
          avg_dcc = df8.agg({'vf_DisplacementCC': 'mean' })
          df9 = df8.na.fill(value=int(avg_dcc.first()[0]),subset=["vf_DisplacementCC"])
          avg_dci = df9.agg({'vf_DisplacementCI': 'mean' })
          df9 = df9.na.fill(value=int(avg_dci.first()[0]),subset=["vf_DisplacementCI"])
          avg_dl = df9.agg({'vf_DisplacementL': 'mean' })
          df9 = df9.na.fill(value=int(avg_dl.first()[0]),subset=["vf_DisplacementL"])
          df9 = df9.na.fill(value=4,subset=['vf_Doors'])
          avg_cyl = df9.agg({'vf_EngineCylinders': 'mean' })
          df9 = df9.na.fill(value=int(avg_cyl.first()[0]),subset=["vf_EngineCylinders"])
          avg_ehp = df9.agg({'vf_EngineHP': 'mean' })
          df9 = df9.na.fill(value=avg_ehp.first()[0],subset=["vf_EngineHP"])
          avg_ekw = df9.agg({'vf_EngineKW': 'mean' })
          df9 = df9.na.fill(value=avg_ekw.first()[0],subset=["vf_EngineKW"])
          avg_sr = df9.agg({'vf_SeatRows': 'mean' })
          df9 = df9.na.fill(value=int(avg_sr.first()[0]),subset=["vf_SeatRows"])
          avg_seats = df9.agg({'vf_Seats': 'mean' })
          df9 = df9.na.fill(value=int(avg_seats.first()[0]),subset=["vf_Seats"])
          avg_speed = df9.agg({'vf_TopSpeedMPH': 'mean' })
          df9 = df9.na.fill(value=int(avg_speed.first()[0]),subset=["vf_TopSpeedMPH"])
          avg_tspeed = df9.agg({'vf_TransmissionSpeeds': 'mean' })
          df9 = df9.na.fill(value=int(avg_tspeed.first()[0]),subset=["vf_TransmissionSpeeds"])
          avg_wbs = df9.agg({'vf_WheelBaseShort': 'mean' })
          df9 = df9.na.fill(value=avg_wbs.first()[0],subset=["vf_WheelBaseShort"])
          avg_wsf = df9.agg({'vf_WheelSizeFront': 'mean' })
          df9 = df9.na.fill(value=int(avg_wsf.first()[0]),subset=["vf_WheelSizeFront"])
          avg_wsr = df9.agg({'vf_WheelSizeRear': 'mean' })
          df9 = df9.na.fill(value=int(avg_wsr.first()[0]),subset=["vf_WheelSizeRear"])
          avg_wheels = df9.agg({'vf_Wheels': 'mean' })
          df9 = df9.na.fill(value=int(avg_wheels.first()[0]),subset=["vf_Wheels"])
          df9 = df9.na.fill(value=4,subset=['vf_Windows'])
          df9.select('vf_DisplacementCC','vf_DisplacementCI','vf_DisplacementL','vf_Doors','vf_EngineCylinders','vf_EngineHP','v
```

The final columns after performing the above steps are:-

```
In [108]: df9.dtypes

Out[108]: [('vin', 'string'),
           ('askPrice', 'int'),
           ('mileage', 'int'),
           ('isNew', 'int'),
           ('brandName', 'string'),
           ('modelName', 'string'),
           ('vf_AirBagLocFront', 'int'),
           ('vf_AirBagLocKnee', 'int'),
           ('vf_AirBagLocSide', 'int'),
           ('vf_Axles', 'int'),
           ('vf_BasePrice', 'double'),
           ('vf_DisplacementCC', 'double'),
           ('vf_DisplacementCI', 'double'),
           ('vf_DisplacementL', 'double'),
           ('vf_Doors', 'int'),
           ('vf_EngineCylinders', 'int'),
           ('vf_EngineHP', 'double'),
           ('vf_EngineKW', 'double'),
           ('vf_SeatRows', 'int'),
           ('vf_Seats', 'int'),
           ('vf_TopSpeedMPH', 'int'),
           ('vf_TransmissionSpeeds', 'int'),
           ('vf_WheelBaseShort', 'double'),
           ('vf_WheelSizeFront', 'int'),
           ('vf_WheelSizeRear', 'int'),
           ('vf_Wheels', 'int'),
           ('vf_Windows', 'int'),
           ('msrp', 'int'),
           ('Time in lot', 'int')]
```

DATA ANALYSIS

The dataset is extracted from AutoDealerData.com, and it contains roughly 5.7 million entries pertaining to new and used vehicle listings from dealers in Illinois. The car dataset has detailed information for each vehicle which was sold as it relates to vin, firstseen, lastseen, askPrice, mileage, isNew, brandName, modelName, etc. The dataset also has other parameters like basePrice, number of airbags, top speed, wheel size, engine cylinders, etc. which are important for predicting the selling price of a car.

Exploratory Data Analysis

Exploratory Data analysis on Automobile analytics allows companies to form decisions supporting performance of their manufacturing products. After analysis, automobile manufacturers and individuals should be induced to produce pertinent information on the vehicles. The accuracy in this situation is seriously in doubt. Thus, we are able to ascertain the precise information about the automobiles with the use of machine learning algorithms. Automakers are already leveraging Big

Data to build better cars and provide better customer service. Vehicles linked to Big Data will undoubtedly change our lives. The Big Data Team's function is critical. To begin, it examines data pertaining to planning, sales, or marketing.

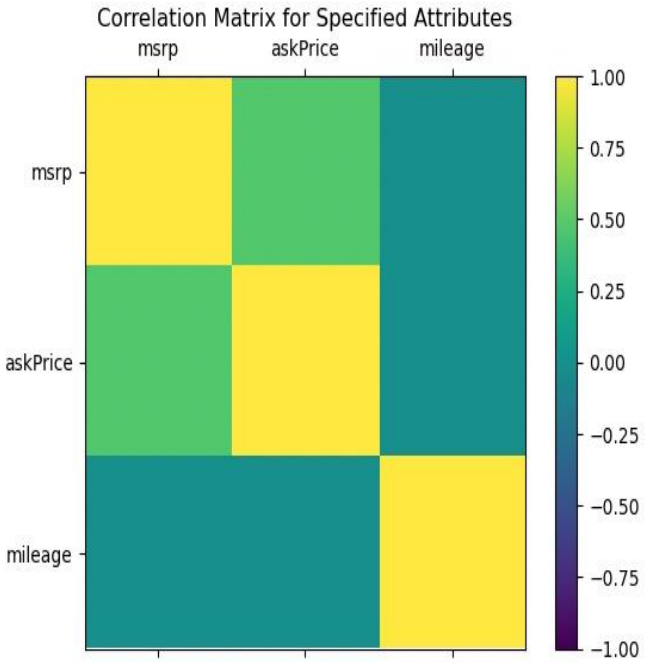
Let’s see how msrp, askPrice, mileage are correlated as per our dataset speaks

[104]:
```
import matplotlib.pyplot as plt
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation

columns = ['msrp','askPrice','mileage']
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=[
 'msrp',
 'askPrice',
 'mileage'],outputCol=vector_col)
myGraph_vector = assembler.transform(df_car_pyspark).select(vector_col)
matrix = Correlation.corr(myGraph_vector, vector_col)
```

[105]:
```
matrix = Correlation.corr(myGraph_vector, vector_col).collect()[0][0]
corrmatrix = matrix.toArray().tolist()
```

```
In [106]: def plot_corr_matrix(correlations,attr,fig_no):
              fig=plt.figure(fig_no)
              ax=fig.add_subplot(111)
              ax.set_title("Correlation Matrix for Specified Attributes")
              ax.set_xticklabels(['']+attr)
              ax.set_yticklabels(['']+attr)
              cax=ax.matshow(correlations,vmax=1,vmin=-1)
              fig.colorbar(cax)
              plt.show()

          plot_corr_matrix(corrmatrix, columns, 234)
```

```
/var/folders/wg/trcddx8531q4js5nks1rqkrm0000gn/T/ipykernel_902/403585419.py:5: UserWarning: FixedFormatter should onl
y be used together with FixedLocator
  ax.set_xticklabels(['']+attr)
/var/folders/wg/trcddx8531q4js5nks1rqkrm0000gn/T/ipykernel_902/403585419.py:6: UserWarning: FixedFormatter should onl
y be used together with FixedLocator
  ax.set_yticklabels(['']+attr)
```



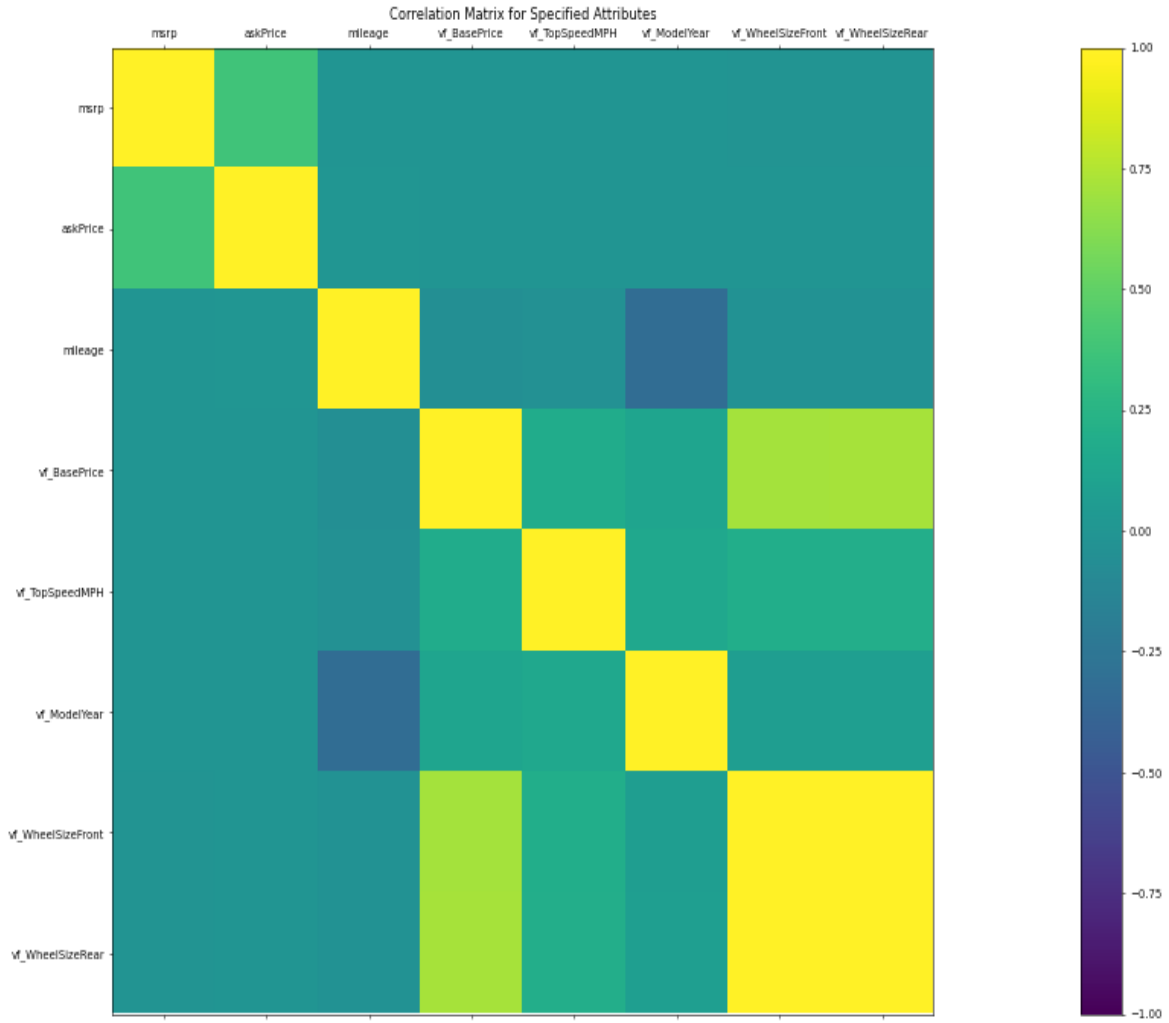
As we observe from correlation matrix and correlation plot. The attributes **msrp** and **askPrice** are positively correlated. And there is no significant correlation or rather slightly negative correlation between **msrp** and **mileage** and similarly is the case with **askPrice** and **mileage**, slightly insignificant negative correlation.

```
: cordf = spark.createDataFrame(corrmatrix,columns)
  cordf.show()
```

```
+--------------------+--------------------+--------------------+
|                msrp|            askPrice|             mileage|
+--------------------+--------------------+--------------------+
|                 1.0| 0.47134298012114856|-0.00767595923878...|
| 0.47134298012114856|                 1.0|-0.00217154095209...|
|-0.00767595923878...|-0.00217154095209...|                 1.0|
+--------------------+--------------------+--------------------+
```

Similarly, let's see correlation between other variables

'vf_BasePrice', 'vf_TopSpeedMPH' ,'vf_ModelYear' ,'vf_WheelSizeFront', 'vf_WheelSizeRear'



As we can see from the above matrix , there is not much correlation between above attributes or features except 'vf_TopSpeedMPH' with ,'vf_WheelSizeFront', and 'vf_WheelSizeRear'. Hence there is very little relationship between independent variables themselves.

For numerical Records, What is Min, Max, Mean, Std dev etc

```
For numerical Records , what is Min,MAX, Mean, STD etc
```

In [ ]:
```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('BigdataProject').getOrCreate()
spark
```

In [ ]:
```
df = spark.read.option('header','true').csv('CIS_Automotive_Kaggle_Sample.csv',inferSchema=True)
```

In [196]:
```
from pyspark.sql.types import IntegerType,BooleanType,DateType
df_all_categories_numeric = df.select('msrp',
                                      'askPrice',
                                      'mileage',
                                      (df.isNew.cast(IntegerType())),
                                      )
```

In [197]:
```
df_all_categories_numeric.describe().show()
```

```
[Stage 90:=============================================>          (32 + 8) / 40]
```

```
+-------+-----------------+------------------+-----------------+-------------------+
|summary|             msrp|          askPrice|          mileage|              isNew|
+-------+-----------------+------------------+-----------------+-------------------+
|  count|          5695015|           5695015|          5695015|            5695015|
|   mean| 744569.672316403|  186920.6197764185|22414.654346652293|0.34360102651178265|
| stddev|3.92644268470064E7|1.876751952261008E7| 901052.7846954644| 0.4749098869204381|
|    min|                0|                 0|                0|                  0|
|    max|       2147483647|        2147483647|       2147483647|                  1|
+-------+-----------------+------------------+-----------------+-------------------+
```

```
In [198]: df_all_categories_numeric_others = df.select('vf_BasePrice',
                                                       'vf_TopSpeedMPH',
                                                       'vf_ModelYear',
                                                       'vf_WheelSizeFront',
                                                       'vf_WheelSizeRear',
                                                       'vf_TransmissionSpeeds')
```

```
In [199]: df_all_categories_numeric_others.describe().show()
```

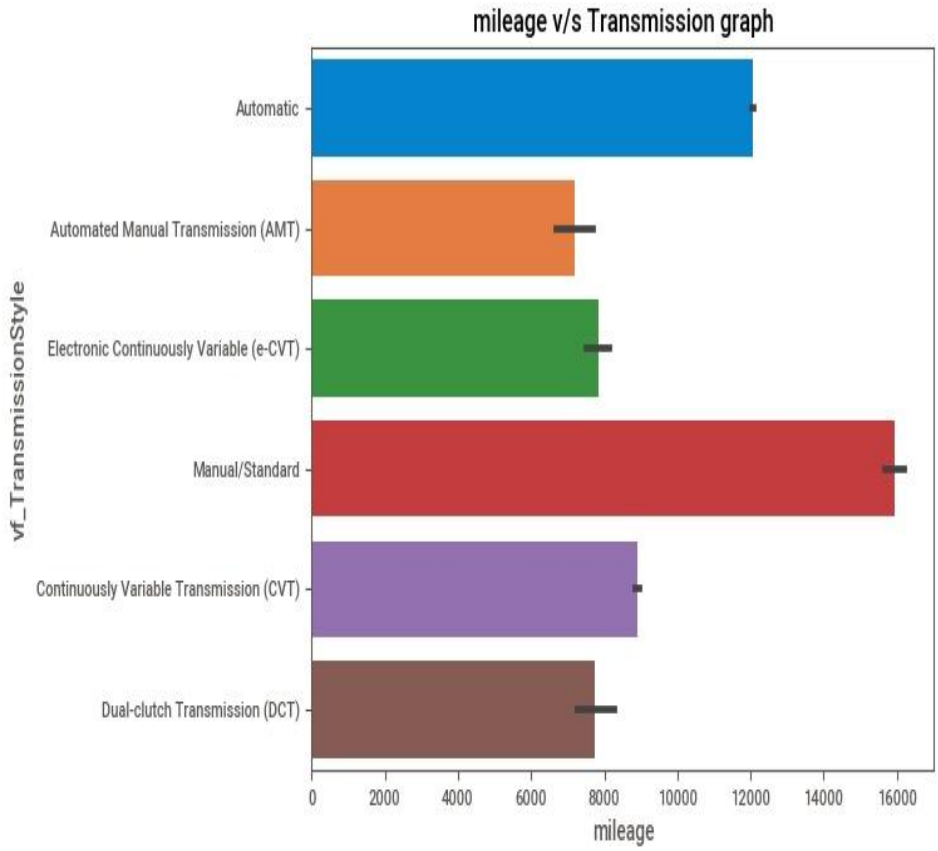
```
[Stage 93:=========================================>          (32 + 8) / 40]

+-------+-----------------+-----------------+----------------+-----------------+-----------------+-----------
---------+
|summary|      vf_BasePrice|    vf_TopSpeedMPH|     vf_ModelYear| vf_WheelSizeFront|  vf_WheelSizeRear|vf_Transmiss
ionSpeeds|
+-------+-----------------+-----------------+----------------+-----------------+-----------------+-----------
---------+
|  count|          1856743|           897427|         5693733|          1730210|          1729777|
1293522|
|   mean|33861.567958462765|126.63256732859608| 2015.30454835167|17.672653608521507| 17.67579231311319|    6.894028
087655254|
| stddev| 14525.66599862042|  14.7212905262075|4.613181706024689| 1.298755667555675|1.2932947852509338|    1.390091
978364042|
|    min|             13.9|               81|            1980|                2|               14|
1|
|    max|         420325.0|              205|            2021|               22|               22|
10|
+-------+-----------------+-----------------+----------------+-----------------+-----------------+-----------
---------+
```

No let's start the **Deduction Show**:

This is an important step in Exploratory data analysis, as it can retrieve the most innovative and interesting deductions to make data driven decisions.

Mileage vs Transmission Style

```
In [59]: model_transmission = sns.barplot(x='mileage', y='vf_TransmissionStyle', data=pandas_df_MT)

         #Graph title
         plt.title('mileage v/s Transmission graph')

Out[59]: Text(0.5, 1.0, 'mileage v/s Transmission graph')
```

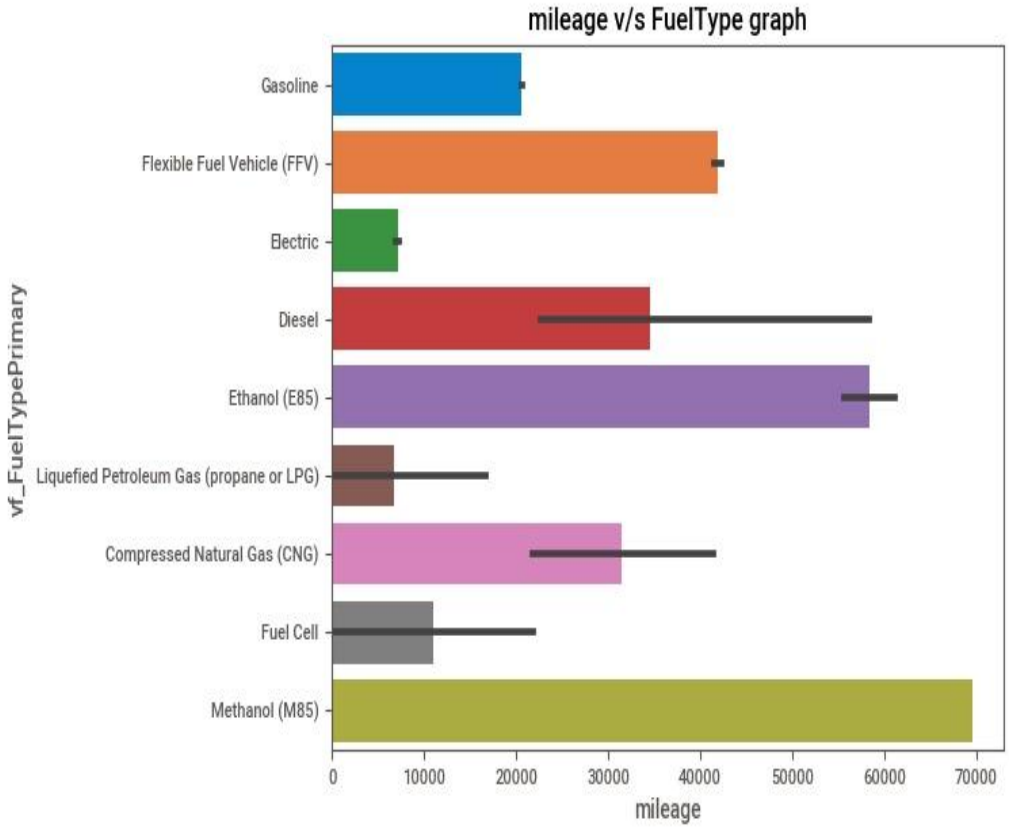


When compared to the other transmission systems, manual transmission has the most mileage among the three most modern transmission systems.

Manual transmissions include more gears and a simpler design, making the transmission system lighter. A simpler method minimizes the annual fuel consumption of the vehicle and, as a consequence, the cost of maintenance.

Fuel-type vs Mileage

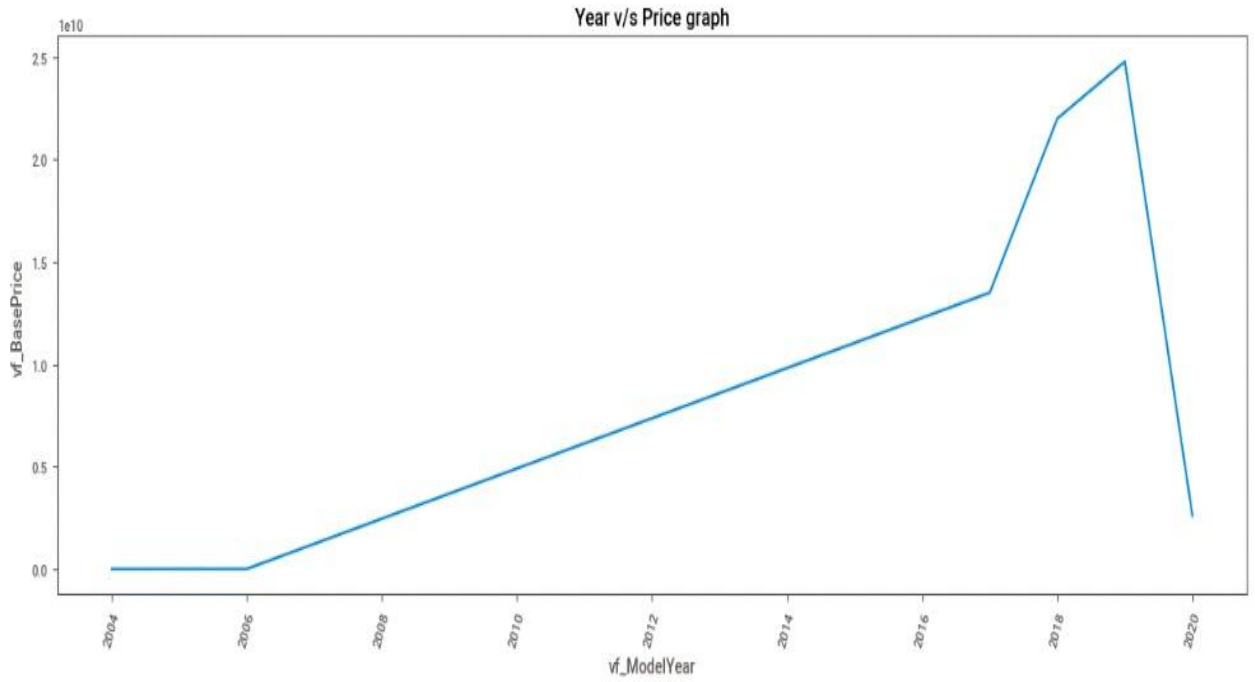
```
In [74]: model_fuel = sns.barplot(x='mileage', y='vf_FuelTypePrimary', data=pandas_df_MFT)

         #Graph title
         plt.title('mileage v/s FuelType graph')

Out[74]: Text(0.5, 1.0, 'mileage v/s FuelType graph')
```



Diesel engines are suitable for long-distance travel. Despite having higher efficiency and cheaper prices than gasoline, diesel engines are limited for vehicles that regularly travel, such as trucks, buses, and off-road vehicles. Diesel engines are limited for vehicles that regularly travel, such as trucks, buses, and off-road vehicles, due to higher greenhouse gas emissions.

Year vs Price

```
In [77]: #load the data in group for visual representation of the data
         #Take the sum of price attribute of the data
         #sort the values from higher to lower
         model_price = pandas_df_YP.groupby('vf_ModelYear')['vf_BasePrice'].sum().sort_values()

         #load matplotlib and seaborn library to generate visual graphs
         plt.figure(figsize=(15,5))
         pal = sns.color_palette("Blues", len(model_price))
         sns.lineplot(x=model_price.index , y=model_price , palette=pal)

         plt.xticks(rotation=70)

         #Graph title
         plt.title('Year v/s Price graph')
```



The price graph is drawn in accordance with the trend line y = 1.3 + 2.5x (x 10^8).

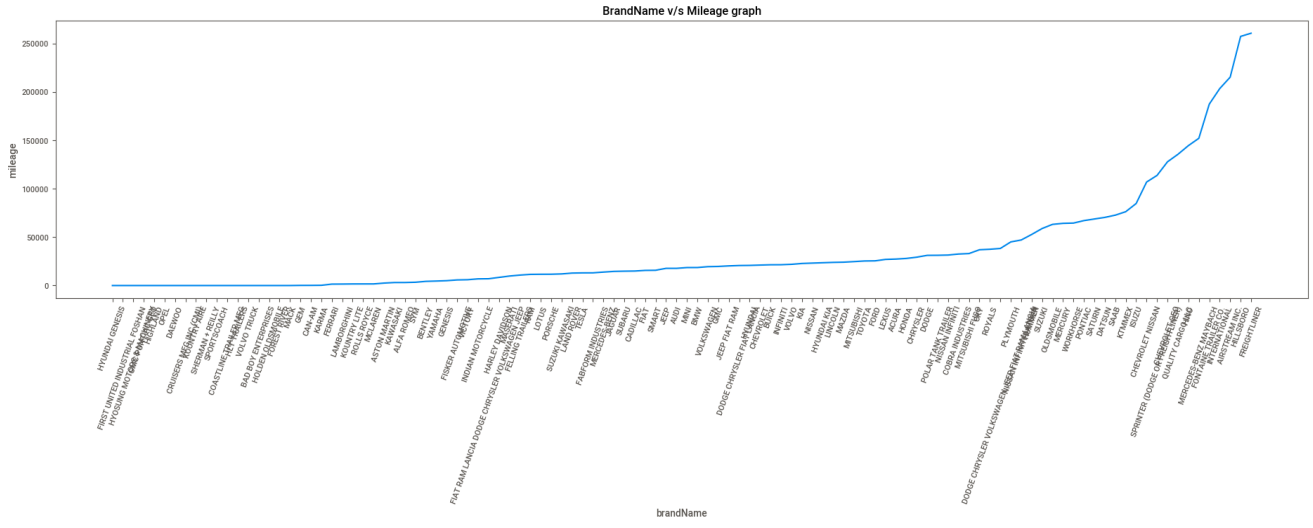
The trend line was calculated using the Least Squares Method. This method can be used to predict future prices, give a brief summary of previous sales, and identify growth prospects.

The sales are growing from left to right, and the slope of the trend line is (m) = 2.5, which is positive.

Sales are increasing right to left. Although sales of motor manufacturers were halted due to the pandemic, resulting in a price drop, a price reduction from 2019 to 2020 has a higher impact on corporate profit margins.

Model vs Mileage graph

```
In [141]: #load the data in group for visual representation of the data
          #Take the mean of Mileage (Distance travelled) attribute of the data
          #sort the values from higher to lower
          model_mileage = pandas_df_MM.groupby('brandName')['mileage'].mean().sort_values()

          #load matplotlib and seaborn library to generate visual graphs
          plt.figure(figsize=(23,5))
          pal = sns.color_palette("Blues", len(model_mileage))
          sns.lineplot(x=model_mileage.index , y=model_mileage , palette=pal)

          plt.xticks(rotation=70)

          #Graph title
          plt.title('BrandName v/s Mileage graph')
```



The graph is a little blurry but according to the median of the statistics, Chevrolet Nissan, Sprinter, ISUZU, Hillsboro, Freightliner model automobiles had better mileage performance than other brands like Ford, Daewoo,Hyundai Genesis, Yamaha,Porsche, Lotus.

Now, let's look at the brands of white cars :

```
In [168]: df_d1 = panda_df1[panda_df1['color'] =='White']
          print(set(df_d1['brandName']))

          {'MITSUBISHI', 'FIAT', 'SUBARU', 'GMC', 'LEXUS', 'CHEVROLET', 'HONDA', 'CADILLAC', 'FORD', 'TOYOTA', 'SMART', 'VOLKSW
          AGEN', 'NISSAN', 'VOLVO', 'LINCOLN', 'LAND ROVER', 'ALFA ROMEO', 'ACURA', 'DODGE', 'JAGUAR', 'ROLLS ROYCE', 'BMW', 'J
          EEP', 'TESLA', 'RAM', 'HYUNDAI', 'BUICK', 'MINI', 'MASERATI', 'AUDI', 'INFINITI', 'MERCEDES-BENZ', 'KIA', 'BENTLEY',
          'CHRYSLER', 'MAZDA', 'PORSCHE'}
```

As per above data results 'MITSUBISHI', 'FIAT', 'SUBARU', 'GMC', 'LEXUS', 'CHEVROLET', 'HONDA', 'CADILLAC', 'FORD', 'TOYOTA', 'SMART', 'VOLKSWAGEN', 'NISSAN', 'VOLVO', 'LINCOLN', 'LAND ROVER', 'ALFA ROMEO', 'ACURA', 'DODGE', 'JAGUAR', 'ROLLS ROYCE', 'BMW', 'JEEP', 'TESLA', 'RAM', 'HYUNDAI', 'BUICK', 'MINI', 'MASERATI', 'AUDI', 'INFINITI', 'MERCEDES-BENZ', 'KIA', 'BENTLEY', 'CHRYSLER', 'MAZDA', 'PORSCHE' are the brands that have white color cars in their models.

```
In [172]: print(dict(Counter(df_d1['brandName']).most_common(5)))

          {'FORD': 5650, 'JEEP': 3379, 'VOLKSWAGEN': 2458, 'HONDA': 2397, 'CHEVROLET': 1878}

          We see that most of the white cars are FORD,JEEP,VOLKSWAGEN,HONDA,CHEVROLET
```

We see that most of the white cars are in **FORD, JEEP, VOLKSWAGEN, HONDA, CHEVROLET** brands.

Following that, summary statistics from numerical columns such as 'price' would be useful. Let's construct a function that takes three arguments: a data frame, a categorical column, and a numerical column. The mean and standard deviation of each numerical column are stored in a data frame, which is then sorted descendingly by mean.

We are able to quickly see if different categories have higher or lower mean and/or standard deviation values for a certain numerical column.

In [173]:
```
def return_statistics(data_frame, categorical_column, numerical_column):
    mean = []
    std = []
    field = []
    for i in set(list(data_frame[categorical_column].values)):
        new_data = data_frame[data_frame[categorical_column] == i]
        field.append(i)
        mean.append(new_data[numerical_column].mean())
        std.append(new_data[numerical_column].std())
    df = pd.DataFrame({'{}'.format(categorical_column): field, 'mean {}'.format(numerical_column): mean, 'std in {}'.fo
    df.sort_values('mean {}'.format(numerical_column), inplace = True, ascending = False)
    df.dropna(inplace = True)
    return df
```

In [176]:
```
stats = return_statistics(panda_df1, 'brandName', 'vf_BasePrice')
print(stats.head(15))
```

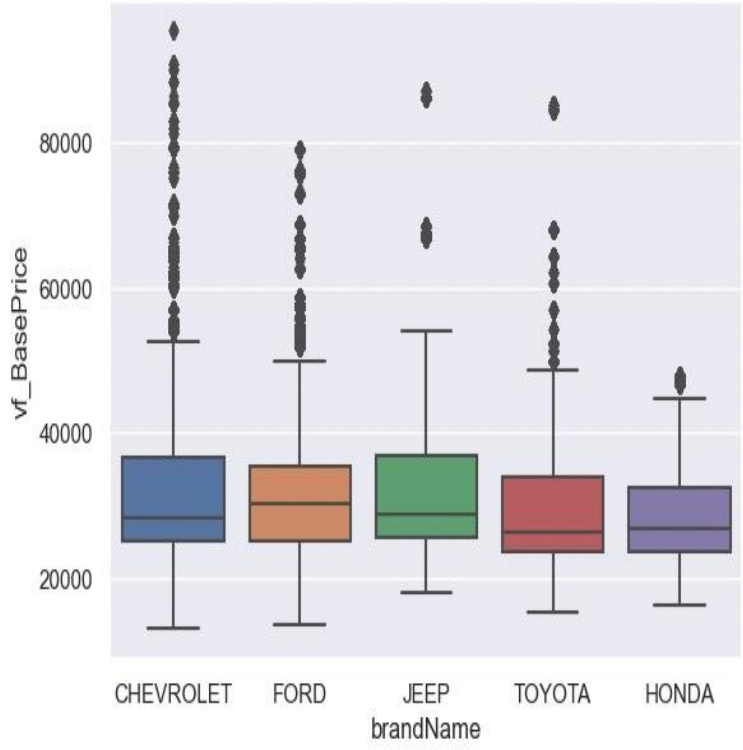
	brandName	mean vf_BasePrice	std in vf_BasePrice
20	ROLLS ROYCE	320561.290323	22740.218445
33	BENTLEY	274861.463768	88478.494420
28	MASERATI	79869.322917	12341.036686
17	PORSCHE	75610.085152	35687.328507
23	TESLA	74138.364780	14847.314886
14	LAND ROVER	65661.068281	23136.114821
7	CADILLAC	57198.227118	16790.909157
19	JAGUAR	55216.399321	15482.207517
31	MERCEDES-BENZ	54500.230937	22404.925526
29	AUDI	52109.713198	11524.425060
21	BMW	52098.200283	15417.641589
30	INFINITI	50932.604167	9670.421461
15	ALFA ROMEO	50382.388708	12712.186358
12	VOLVO	47911.333031	6742.631966
4	LEXUS	47350.192019	12901.736966

Boxplots for 'price' in the 5 most commonly occurring 'brand' categories:

In [190]:
```
import matplotlib.pyplot as plt
def get_boxplot_of_categories(data_frame, categorical_column, numerical_column, limit):
    import seaborn as sns
    from collections import Counter
    keys = []
    for i in dict(Counter(panda_df1[categorical_column].values).most_common(limit)):
        keys.append(i)
    print(keys)
    df_new = panda_df1[panda_df1[categorical_column].isin(keys)]
    sns.set()
    sns.boxplot(x = df_new[categorical_column], y = df_new[numerical_column])
    plt.show()
```

In [191]:
```
get_boxplot_of_categories(panda_df1, 'brandName', 'vf_BasePrice', 5)
```

```
['FORD', 'CHEVROLET', 'JEEP', 'TOYOTA', 'HONDA']
```



Data Analysis for specific questions as proposed

How Long Before a Car is Sold?

For exploring this question we have shown interest in two columns from our dataFrame : i.e, lastSeen , FirstSeen

```
|-- firstSeen: timestamp (nullable = true)
|-- lastSeen: timestamp (nullable = true)
```

We have calculated the difference between two dates to get the number of days the car has stayed in Lot. We called it "Time_in_Lot" and created a new column in our DataFrame for this. We have observed that the minimum number of days that a car was in a dealer's lot was zero whereas the max was 1432 days which is around ~ 3.9 years.which we created during the data preprocessing stage.

```
+-------+------------------+
|summary|       Time_in_lot|
+-------+------------------+
|  count|            568890|
|   mean| 67.62708432210093|
| stddev|105.51040395578812|
|    min|             -1115|
|    max|              1409|
+-------+------------------+
```

As seen above, a summary of the timeInLot variable (measured in days) shows that the minimum number of days that a car was in a dealer's lot was zero, whereas the max was 1,409 days (~3.9 years). The mean and median were more reasonable showing that a car will typically be in a dealer's lot between one to three months. Now, we want to be able to predict how long a car will sit in a dealer's lot based on the other variables in the dataset.

Which Cars are the Most Popular?

We are using the brandName and their ModelName. Fortunately, there were no missing values present within the relevant columns for this question.

We made a new column in our data table called “car” by combining the variables brandName and modelName. It contains car names with the count of how many cars are listed from the total cars in the dataset. Let’s explore cars popularity with their brand and their model

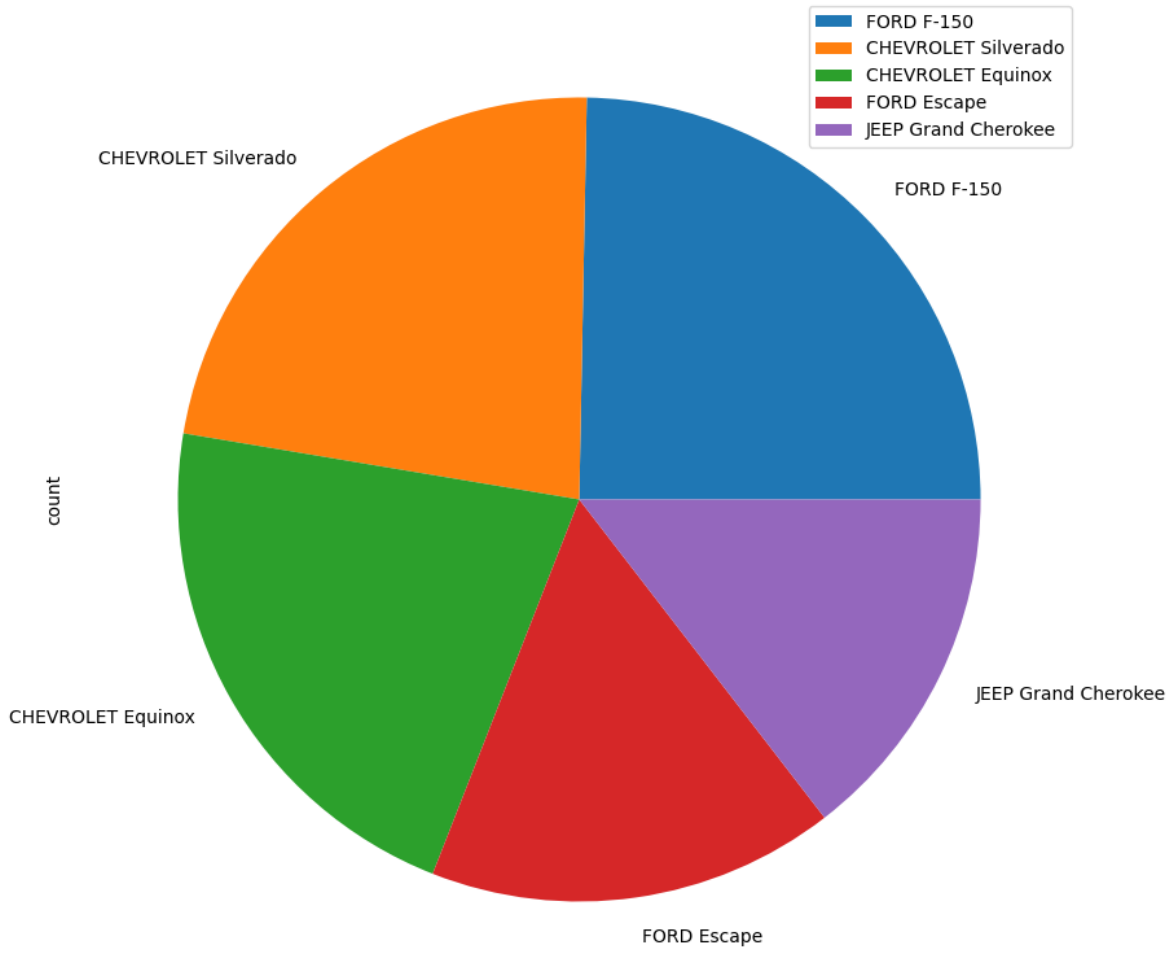
First we concatenate both columns into a new column Cars, and then we group by count of most popular cars.

```
df2 = df_car_pyspark.select(concat(col("brandName"),lit(" "),col("modelName")).alias("Cars"))
```

```
+-------------+
|         Cars|
+-------------+
|FORD Explorer|
|   FORD F-150|
|JAGUAR F-Type|
|    FORD Edge|
|  FORD Fusion|
|FORD Explorer|
|   FORD Focus|
|FORD Explorer|
|  FORD Escape|
|  FORD Fusion|
+-------------+
only showing top 10 rows
```

```
+--------------------+-----+
|                Cars|count|
+--------------------+-----+
|           FORD F-150|17802|
|CHEVROLET Silverado|16337|
|  CHEVROLET Equinox|15626|
|          FORD Escape|11833|
|JEEP Grand Cherokee|10474|
+--------------------+-----+
only showing top 5 rows
```

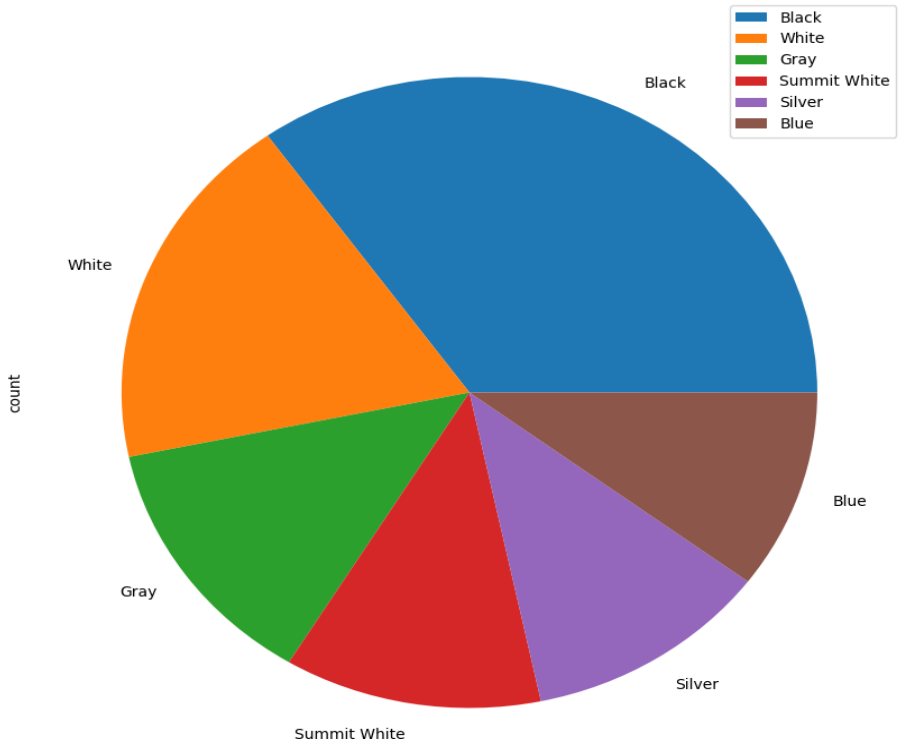
Above, we have the top 5 most popular cars from our dataset. You can see that Ford F-150 is the most popular car listed by dealers followed by Chevrolet Silverado and Chevrolet Equinox.
Overall, we can say that Ford and Chevrolet are the two most popular brands among car buyers



What Color of the car do people like the most?

In order to address this question we are going to be interested in color column color .We have grouped by count of color values in color column and found Black is the most popular colors which people like the most. Then white and then gray and so on.

```
+------------+------+
|       color| count|
+------------+------+
|         N/A|210388|
|       Black| 21048|
|       White| 11125|
|        Gray|  7875|
|Summit White|  7235|
|      Silver|  6929|
+------------+------+
only showing top 6 rows
```



How much will the final price differ from MSRP on Average?

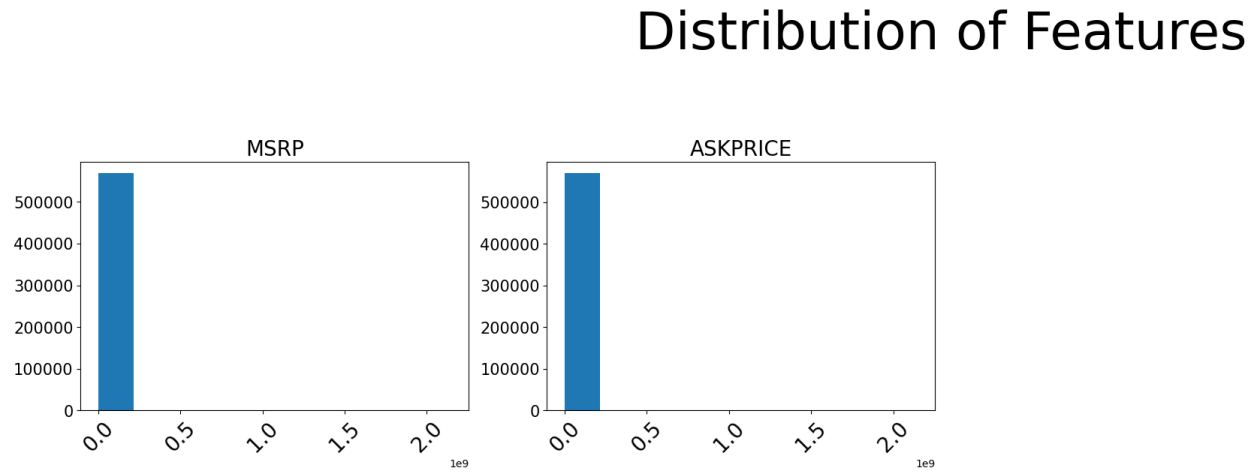
```
root
 |-- msrp: integer (nullable = true)
 |-- askPrice: integer (nullable = true)
```

The manufacturer's suggested retail price (MSRP) is the price that a product's manufacturer recommends it be sold for at the point of sale. Any retail product can have an MSRP, but the term is frequently used with automobiles. An MSRP is sometimes informally known as the "sticker price." Whereas the final price is the last ask price before the vehicle was sold. So how much will the ask price differ from the MSRP on the average car listed?

```
df5.corr("msrp","askPrice")
```

```
0.48286155837331335
```

We will use msrp & askPrice variables to find the average price difference between MSRP and ask price.The correlation between the two fields is given below. i.e (48%)



As seen in the boxplot, there is little difference between msrp and ask price, so they appear to be fairly similar to one another. Now, we make a new variable called price_diff which is simply the difference between msrp and askPrice. A summary of the price_diff variable is given below.

msrp	askPrice	Price_diff
0	0	0
47174	47174	0
45996	45996	0
23537	23537	0
12930	12930	0
24457	24457	0
14529	14529	0
45644	45644	0
24286	24286	0
12394	12394	0
13819	13769	50
23550	23550	0
0	0	0
0	0	0
17989	17989	0
999	999	0
11998	11998	0
21208	18958	2250
12875	12875	0
15196	15196	0

only showing top 20 rows

summary	price_diff
count	568890
mean	524971.7153808293
stddev	3.34535627086548 47E7
min	0
max	2147481314

As you can see above, the mean of the price_diff variable is $524971.7. Therefore, we can conclude that the average price difference between the final price (ask price) and MSRP is around $524971.7

MODEL TRAINING

In the final and most important stage of the data science lifecycle, we must decide which model to use to produce predictions. We can determine which model to utilize by implementing different machine learning models and checking their accuracy. We will determine the best model to use based on whichever model has the best accuracy and minimum RMSE score.

First, we will use VectorAssembler to combine all our features into a single feature column.

```
In [109]: from pyspark.ml.feature import VectorAssembler
          ip_cols = ['askPrice','mileage','isNew','vf_AirBagLocFront','vf_AirBagLocKnee', 'vf_AirBagLocSide','vf_Axles','vf_Basel
            'vf_EngineCylinders','vf_EngineHP','vf_EngineKW','vf_SeatRows','vf_Seats','vf_TopSpeedMPH','vf_TransmissionSpeeds','
          op_col = "Features"
          vec_df = VectorAssembler(inputCols = ip_cols, outputCol = op_col)

          df_ml = vec_df.transform(df9)
          df_ml.select(['Features']).toPandas().head(5)
```

Out[109]:

	Features
0	[12387.0, 0.0, 0.0, 2.0, 2.0, 4.0, 2.0, 23475....
1	[12970.0, 0.0, 0.0, 2.0, 2.0, 4.0, 2.0, 26500....
2	[15218.0, 0.0, 0.0, 2.0, 0.0, 4.0, 2.0, 23940....
3	[18755.0, 0.0, 0.0, 2.0, 0.0, 6.0, 2.0, 38495....
4	[36999.0, 0.0, 0.0, 2.0, 0.0, 6.0, 2.0, 34175....

```
In [111]: final_df = df_ml.select(['Features','msrp'])
          final_df.show(1)
```

```
+--------------------+-----+
|            Features| msrp|
+--------------------+-----+
|[12387.0,0.0,0.0,...|12387|
+--------------------+-----+
only showing top 1 row
```

After that, we will split the dataframe into train and test sets.

```
In [84]: len_df = final_df.count()

         train_len = int(0.7*len_df)

In [86]: test_df = final_df
         train_df = final_df.limit(train_len)
         train_df.show()
         train_df.count()
```

```
+--------------------+-----+
|            Features| msrp|
+--------------------+-----+
|[12387.0,0.0,0.0,...|12387|
|[12970.0,0.0,0.0,...|12970|
|[15218.0,0.0,0.0,...|15218|
|[18755.0,0.0,0.0,...|18755|
|[36999.0,0.0,0.0,...|36999|
|[18276.0,0.0,1.0,...|18276|
|[22140.0,0.0,1.0,...|22140|
|[0.0,0.0,1.0,2.0,...|    0|
|[20494.0,0.0,0.0,...|20494|
|[19026.0,0.0,1.0,...|19026|
```

```
In [87]: test_df = test_df.subtract(train_df)
         test_df.show()
         test_df.count()
```

```
+--------------------+-----+
|            Features| msrp|
+--------------------+-----+
|[27962.0,0.0,0.0,...|27962|
|[32962.0,0.0,0.0,...|32962|
|[28430.0,0.0,1.0,...|28430|
|[37562.0,0.0,1.0,...|44135|
|[33962.0,0.0,0.0,...|33962|
|[56962.0,0.0,0.0,...|56962|
|[33962.0,0.0,0.0,...|33962|
|[23962.0,0.0,0.0,...|23962|
```

Linear Regression

Linear regression, by definition, investigates linear relationships between the dependent and independent variables. It assumes that the variables have a straight-line relationship and attempts to determine the line that best fits the data. It is not always the case that variables have a linear relationship. Below is the screenshot of linear regression model implemented on our dataset:-

```
In [91]: from pyspark.ml.regression import LinearRegression

In [92]: lr = LinearRegression(featuresCol = 'Features', labelCol='msrp', maxIter=1000)
```

```
In [95]: lr_model = lr.fit(train_df)

22/12/07 09:38:43 WARN Instrumentation: [1459a871] regParam is zero, which might cause numerical instability and over
fitting.
22/12/07 09:38:43 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
22/12/07 09:38:43 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.Foreign
LinkerBLAS
22/12/07 09:38:43 WARN InstanceBuilder$NativeLAPACK: Failed to load implementation from:dev.ludovic.netlib.lapack.JNI
LAPACK
22/12/07 09:38:43 WARN Instrumentation: [1459a871] Cholesky solver failed due to singular covariance matrix. Retrying
with Quasi-Newton solver.

In [96]: y_pred = lr_model.transform(test_df)

In [97]: y_pred.show()

+--------------------+-----+------------------+
|            Features| msrp|        prediction|
+--------------------+-----+------------------+
|[27962.0,0.0,0.0,...|27962| 28613.18119566859|
|[32962.0,0.0,0.0,...|32962| 33263.44111300582|
|[28430.0,0.0,1.0,...|28430|30446.303625853638|
|[37562.0,0.0,1.0,...|44135|39906.832302568124|
|[33962.0,0.0,0.0,...|33962| 33989.25386221418|
|[56962.0,0.0,0.0,...|56962|57356.144496655084|
```

Below are the results of the Linear regression model. We can see the mean, min, max and std deviation of the predicted and actual values and compare them.

```
In [98]: y_pred.describe().show()

+-------+-----------------+------------------+
|summary|             msrp|        prediction|
+-------+-----------------+------------------+
|  count|             1092|              1092|
|   mean|35446.53754578755| 34560.03562395302|
| stddev|20163.11506465241| 14004.00870247903|
|    min|                0|-255.99827909185456|
|    max|           412623| 95495.75298610076|
+-------+-----------------+------------------+


In [99]: print("Coefficients: " + str(lr_model.coefficients))
         print("Intercept: " + str(lr_model.intercept))

Coefficients: [1.0043975112183214,-0.014797486413576973,2091.2713028936464,-12.486804960087552,58.61398418279996,34.9
11650476930205,0.0,-0.002479340368617685,0.0017661263484190018,0.028921764524519385,1.7391306909587945,380.6831580857
333,-95.057096739133,-2.6779545590939557,5.38493994757674,230.026279073777,-45.847650175004325,0.558284231486903,33.6
5684647631677,10.074809848199479,4.0404912314828705,4.0404912314828705,-916.1899163254149,-204.5554514001452,5.656405
2860576535]
Intercept: 1111.3590915967047
```

Below, we can see that the Root Mean Squared Error of the training model is 1485 which is a large value. But, if we increase the number of iterations, the model will converge and the value of RMSE will decrease. But, as our dataset size is too big, we have used a small number of iterations for training our model. The value of R2 is 0.98 which is a very good score. The test value of R2 is 0.49. You can also see that the predicted and actual values are not having too much difference.

```
In [100]: trainingSummary = lr_model.summary
          print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
          print("r2: %f" % trainingSummary.r2)

          RMSE: 1485.681797
          r2: 0.989587

In [101]: y_pred.select("prediction","msrp","Features").show()
          from pyspark.ml.evaluation import RegressionEvaluator
          lr_evaluator = RegressionEvaluator(predictionCol="prediction", labelCol="msrp",metricName="r2")
          print("R Squared (R2) on test data = %g" % lr_evaluator.evaluate(y_pred))

          +------------------+-----+--------------------+
          |        prediction| msrp|            Features|
          +------------------+-----+--------------------+
          | 28613.18119566859|27962|[27962.0,0.0,0.0,...|
          | 33263.44111300582|32962|[32962.0,0.0,0.0,...|
          |30446.303625853638|28430|[28430.0,0.0,1.0,...|
          |39906.832302568124|44135|[37562.0,0.0,1.0,...|
          | 33989.25386221418|33962|[33962.0,0.0,0.0,...|
          |57356.144496655084|56962|[56962.0,0.0,0.0,...|
          | 34034.50510450264|33962|[33962.0,0.0,0.0,...|
          |24012.808437279633|23962|[23962.0,0.0,0.0,...|
          |26978.042044894195|26962|[26462.0,0.0,0.0,...|
          |33972.309695202595|33962|[33962.0,0.0,0.0,...|
          | 61904.03188024244|61462|[61462.0,0.0,0.0,...|
          | 35140.83301554947|37040|[31762.0,0.0,1.0,...|
          |32348.598976167024|32462|[31962.0,0.0,0.0,...|
          | 42594.78199001183|47380|[39962.0,0.0,1.0,...|
          |  56011.1259364615|55962|[55962.0,0.0,0.0,...|
          | 33104.11465479882|32962|[32962.0,0.0,0.0,...|
          | 25408.80916867444|25962|[25962.0,0.0,0.0,...|
          | 52547.22657504253|52962|[52962.0,0.0,0.0,...|
          | 37198.15989667976|36962|[36962.0,0.0,0.0,...|
          | 32942.85244690456|32962|[32462.0,0.0,0.0,...|
          +------------------+-----+--------------------+
          only showing top 20 rows

          R Squared (R2) on test data = 0.49683
```

RMSE of test data is 14296.

```
In [113]: y_pred.select("prediction","msrp","Features").show()
          from pyspark.ml.evaluation import RegressionEvaluator
          lr_evaluator = RegressionEvaluator(predictionCol="prediction", labelCol="msrp",metricName="rmse")
          print("RMSE on test data = %g" % lr_evaluator.evaluate(y_pred))

          +------------------+-----+--------------------+
          |        prediction| msrp|            Features|
          +------------------+-----+--------------------+
          | 28613.18119566859|27962|[27962.0,0.0,0.0,...|
          | 33263.44111300582|32962|[32962.0,0.0,0.0,...|
          |30446.303625853638|28430|[28430.0,0.0,1.0,...|
          |39906.832302568124|44135|[37562.0,0.0,1.0,...|
          | 33989.25386221418|33962|[33962.0,0.0,0.0,...|
          |57356.144496655084|56962|[56962.0,0.0,0.0,...|
          | 34034.50510450264|33962|[33962.0,0.0,0.0,...|
          |24012.808437279633|23962|[23962.0,0.0,0.0,...|
          |26978.042044894195|26962|[26462.0,0.0,0.0,...|
          |33972.309695202595|33962|[33962.0,0.0,0.0,...|
          | 61904.03188024244|61462|[61462.0,0.0,0.0,...|
          | 35140.83301554947|37040|[31762.0,0.0,1.0,...|
          |32348.598976167024|32462|[31962.0,0.0,0.0,...|
          | 42594.78199001183|47380|[39962.0,0.0,1.0,...|
          |  56011.1259364615|55962|[55962.0,0.0,0.0,...|
          | 33104.11465479882|32962|[32962.0,0.0,0.0,...|
          | 25408.80916867444|25962|[25962.0,0.0,0.0,...|
          | 52547.22657504253|52962|[52962.0,0.0,0.0,...|
          | 37198.15989667976|36962|[36962.0,0.0,0.0,...|
          | 32942.85244690456|32962|[32462.0,0.0,0.0,...|
          +------------------+-----+--------------------+
          only showing top 20 rows

          RMSE on test data = 14296
```

Decision Trees

Data is represented by decision trees as a tree with hierarchical branches. It is a flowchart-like structure in which each internal node represents a test on an attribute (for example, whether a coin flip comes up heads or tails), each branch represents the test result, and each leaf node represents a class label (decision taken after computing all attributes). Classification rules are represented by the pathways from the root to the leaf. Decision Trees are capable of performing both regression and classification problems. Below, we can see the result of the Decision tree model. The RMSE is 14567.7 which is a high value but may reduce if we train our model on more iterations.

```
In [109]: #Decision tree
          from pyspark.ml.regression import DecisionTreeRegressor
          dt = DecisionTreeRegressor(featuresCol ='Features', labelCol = 'msrp')
          dt_model = dt.fit(train_df)
          dt_pred = dt_model.transform(test_df)
          dt_evaluator = RegressionEvaluator(
              labelCol="msrp", predictionCol="prediction", metricName="rmse")
          rmse = dt_evaluator.evaluate(dt_pred)
          dt_pred.select("prediction","msrp","Features").show(5)
          print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

          +------------------+-----+--------------------+
          |        prediction| msrp|            Features|
          +------------------+-----+--------------------+
          |28851.537313432837|27962|[27962.0,0.0,0.0,...|
          |34362.063953488374|32962|[32962.0,0.0,0.0,...|
          |31645.285714285714|28430|[28430.0,0.0,1.0,...|
          | 37567.87333333334|44135|[37562.0,0.0,1.0,...|
          |34362.063953488374|33962|[33962.0,0.0,0.0,...|
          +------------------+-----+--------------------+
          only showing top 5 rows

          Root Mean Squared Error (RMSE) on test data = 14567.7
```

Random Forests

Random forests, also known as random decision forests, are a type of ensemble learning for classification, regression, and other tasks. It works by building a lot of decision trees during training. The individual trees' mean or average prediction is returned for regression tasks. Random decision forests eliminate the tendency of decision trees to overfit their training set. Though their accuracy is lower than that of gradient-boosted trees, random forests generally perform better than decision trees. Below is the result of our Random Forest Regression model.

The RMSE is 14974.9 which is a high value but may reduce if we train our model on more iterations.

```
In [105]: #Random Forest regression

          from pyspark.ml.regression import RandomForestRegressor
          rf = RandomForestRegressor(featuresCol = 'Features', labelCol = 'msrp')
          rf_model = rf.fit(train_df)
          rf_predictions = rf_model.transform(test_df)
          rf_predictions.select('prediction', 'msrp', 'Features').show(5)

          +------------------+-----+--------------------+
          |        prediction| msrp|            Features|
          +------------------+-----+--------------------+
          |32410.118371011682|27962|[27962.0,0.0,0.0,...|
          |35798.568380479366|32962|[32962.0,0.0,0.0,...|
          |27913.756314866023|28430|[28430.0,0.0,1.0,...|
          |38809.698206700974|44135|[37562.0,0.0,1.0,...|
          | 35406.18711739892|33962|[33962.0,0.0,0.0,...|
          +------------------+-----+--------------------+
          only showing top 5 rows

In [106]: rf_evaluator = RegressionEvaluator(
              labelCol="msrp", predictionCol="prediction", metricName="rmse")
          rmse = rf_evaluator.evaluate(rf_predictions)
          print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

          Root Mean Squared Error (RMSE) on test data = 14974.9
```

Gradient Boosted Trees Regression

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. Below is the result of our Gradient Boosted Trees Regression model. The RMSE is 14414.6 which is a high value but may reduce if we train our model on more iterations.

```
In [103]: #Gradient boosted tree regression

from pyspark.ml.regression import GBTRegressor
gbt = GBTRegressor(featuresCol = 'Features', labelCol = 'msrp', maxIter=10)
gbt_model = gbt.fit(train_df)
gbt_predictions = gbt_model.transform(test_df)
gbt_predictions.select('prediction', 'msrp', 'Features').show(5)
```

```
+------------------+-----+--------------------+
|        prediction| msrp|            Features|
+------------------+-----+--------------------+
| 28547.29476285316|27962|[27962.0,0.0,0.0,...|
|33632.054347140955|32962|[32962.0,0.0,0.0,...|
|30313.922722000254|28430|[28430.0,0.0,1.0,...|
|40843.328451429115|44135|[37562.0,0.0,1.0,...|
| 33891.63099132587|33962|[33962.0,0.0,0.0,...|
+------------------+-----+--------------------+
only showing top 5 rows
```

```
In [104]: gbt_evaluator = RegressionEvaluator(labelCol="msrp", predictionCol="prediction", metricName="rmse")
rmse = gbt_evaluator.evaluate(gbt_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 14414.6
```

Isotonic Regression

Isotonic regression is the technique of fitting a free-form line to a sequence of observations such that the fitted line is non-decreasing (or non-increasing) everywhere, and lies as close to the observations as possible. Below is the result of our Isotonic Regression model. The RMSE is 14436.3 which is a high value but may reduce if we train our model on more iterations.

```
In [107]: #isotonic regressor

from pyspark.ml.regression import IsotonicRegression
ir = IsotonicRegression(featuresCol = 'Features', labelCol = 'msrp')
ir_model = ir.fit(train_df)
ir_predictions = ir_model.transform(test_df)
ir_predictions.select('prediction', 'msrp', 'Features').show(5)
```

```
+------------------+-----+--------------------+
|        prediction| msrp|            Features|
+------------------+-----+--------------------+
|           28962.0|27962|[27962.0,0.0,0.0,...|
| 33346.55714285714|32962|[32962.0,0.0,0.0,...|
|           29242.0|28430|[28430.0,0.0,1.0,...|
|38781.793103448275|44135|[37562.0,0.0,1.0,...|
| 34947.76388888889|33962|[33962.0,0.0,0.0,...|
+------------------+-----+--------------------+
only showing top 5 rows
```

```
In [108]: ir_evaluator = RegressionEvaluator(
    labelCol="msrp", predictionCol="prediction", metricName="rmse")
rmse = ir_evaluator.evaluate(ir_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 14436.3
```

After comparing all the above models, we can say that Linear Regression performed the best out of all the models.

CONCLUSION

We were able to successfully visualize our dataset and create Machine Learning models for solving our problem statement using Apache Spark and MLlib. Also, we were successfully able to answer all the questions we posed at the beginning of this project. While we were able to answer a few through data analysis and visualization, we quickly learned that we just did not have the resources needed to handle the models we desired to build. However, this caused us to have to step out of our comfort zone and find ways to work around our big data problem. Our daily lives now practically revolve around cars. Demand for and use of cars are rising constantly. Ten or twenty years ago, a company might produce two or three models annually. However, everything has abruptly changed. Businesses are engaged in a strong competition to please customers while boosting sales and profits. They now release new models on a monthly basis as a result. As a result, a review of their models can help the business strengthen its following models. Working on large datasets on local systems comes with limitations like limited memory, lesser RAM, less processing speed, etc. But Apache Spark helped us to process big datasets much faster as compared to local machines.

FUTURE WORK

- In the future, we hope to see our steps repeated but with more powerful machines that will be able to handle the complexity of the models we initially set out to build. With these more complex models, the models will likely have better predictive power to answer our questions.
- Also, in the future, if the size of the datasets is too large, we can use cloud platforms like AWS, GCP, etc. and use Spark on it. Also, we would like to implement our problem statement on different big data machine learning platforms like H2O, AWS Sagemaker, etc. and find out which performs better and is more flexible.

- Another major cause for concern during this project was the dataset. There were just too many missing values across the dataset. There was not a single row that had no missing values. A more complete dataset with fewer inconsistencies would be possibly the best boon to the strength of our models. A potential solution to this problem is to source data directly from the website our dataset was sampled from.
- Sourcing data directly from the website would also allow us to access millions of more robust data points that not only contain information about the state of Illinois but also nationwide data. This data would also be current, and we would be able to see the impact of seasonality within future models.
- We would also like to make a recommendation system and a user interface part. Also, exploring some advanced models like Artificial Neural Networks (ANN). Following this, exploring other big data technologies and doing more comparisons.

DATA SOURCES

The data we utilized for this research came from Kaggle; the large car dataset is available at https://www.kaggle.com/datasets/cisautomotiveapi/large-car-dataset

SOURCE CODE

Here is the link for the code, saved in the repo:
GitHub Link: https://github.com/pathak-vikas/LargeCarDataSetBigDataProject
The src folder consists of three .ipynb files.

1. Data_visualization_for_Large_car_datset.pynb is Data Exploration file
2. ML_Algorithm_on_Partial_dataset.ipynb is the file where ML models are executed on a subset of the dataset.
3. ML_Algorithm_on_Large_car_dataset.ipyb is the file where ML models are executed on the entire dataset.

BIBLIOGRAPHY

[1] Classification and regression - Spark 2.1.0 Documentation. (n.d.). Retrieved December 2, 2022, from https://spark.apache.org/docs/2.1.0/ml-classification-regression.html

[2] *Pyspark.ml package*¶. pyspark.ml package - PySpark master documentation. (n.d.). Retrieved December 2, 2022, from https://spark.apache.org/docs/2.3.0/api/python/pyspark.ml.html

[3] "Pyspark.Ml Package — PySpark Master Documentation," n.d. https://spark.apache.org/docs/2.3.0/api/python/pyspark.ml.html.

[4] HD Desktop Wallpapers. "Volkswagen T2, Volkswagen, van, Motorhome, Car, 4k," n.d. https://wallpapercrafter.com/10822-volkswagen-t2-volkswagen-van-motorhome-car-4k.html.

[5] Kaggle. "Large Car Dataset," August 3, 2021. https://www.kaggle.com/datasets/cisautomotiveapi/large-car-dataset.

[6] Nutan. "Feature Transformer Vectorassembler in PySpark ML Feature - Part 3." *Medium*, Medium, 10 Nov. 2020, https://medium.com/@nutanbhogendrasharma/feature-transformer-vectorassembler-in-pyspark-ml-feature-part-3-b3c2c3c93ee9.

[7] Li, Susan. "Building a Linear Regression with PySpark and MLlib." *Medium*, Towards Data Science, 7 May 2018, https://towardsdatascience.com/building-a-linear-regression-with-pyspark-and-mllib-d065c3ba246a.