Used Car Price Prediction

Data science and Artificial Intelligence Society

Jiho Lee, Sungjun Choi, Earl Chae, Haejeong Cho

Predicting the Price of Used Cars

1. **Data Extraction**
2. Feature Engineering
3. Model Performance Analysis

https://github.com/jayhoneylee527/Used-Car-Price-Estimator

Data Extraction

- Data scraped from ***Cargurus.com***
- Used *"uszipcode"* package to get zip codes all over Illinois
- Extracted used car data from *Cargus* at different region in Illinois by randomly selecting zip codes.

```
"""
We want to scrape used car listings in IL, so scrape zipcodes all over IL by population in descending order
"""

zipcodes = []
state_list = ['IL'] |

# Gather zipcodes from each state
search = SearchEngine()
for st in state_list:
    res = search.by_state(state=st, returns = 0) # Return all zipcodes in each state
    for r in res:
        zipcodes.append(r.zipcode)

# Shuffle zipcode list - For randomization
random.shuffle(zipcodes)
```

Data

- We successfully extracted 10,942 used car data

- There are 30 columns

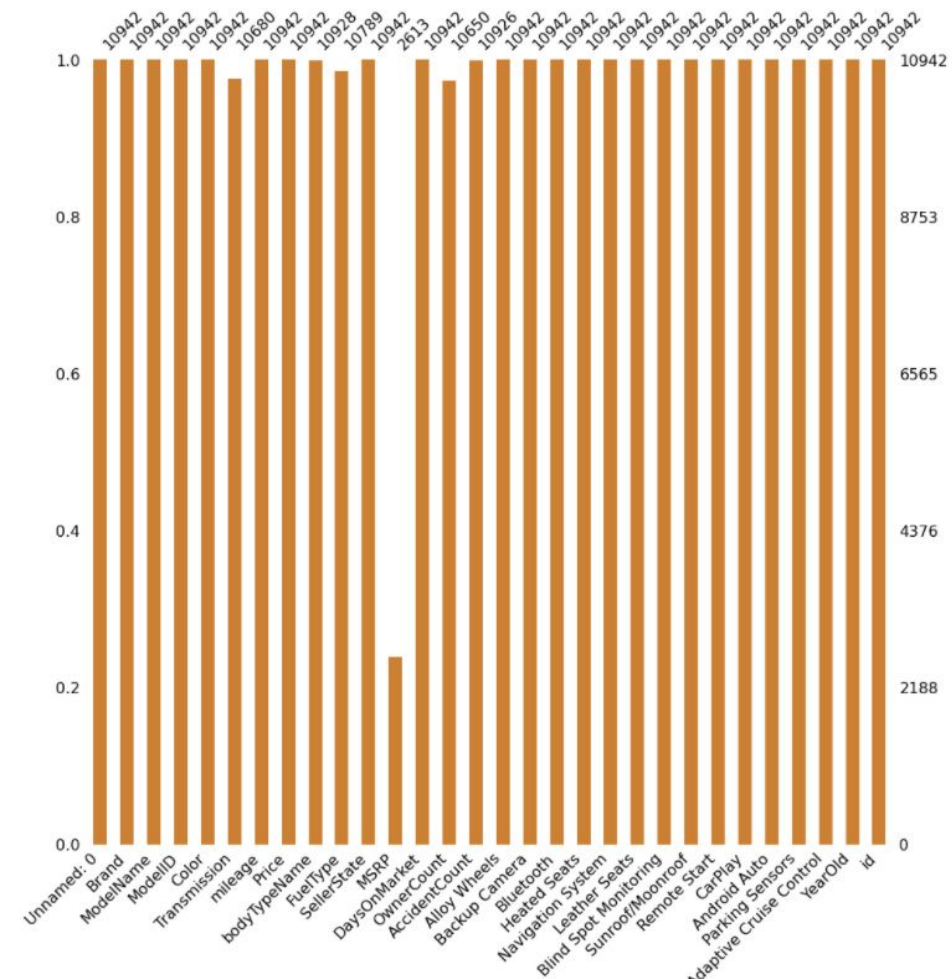
```
spec_tbl_df [10,942 x 30] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ ...1                    : num [1:10942] 0 1 2 3 4 5 6 7 8 9 ...
 $ Brand                   : chr [1:10942] "Ford" "Chrysler" "Toyota" "RAM" ...
 $ ModelName               : chr [1:10942] "F-150" "Pacifica" "4Runner" "2500" ...
 $ ModelID                 : chr [1:10942] "d337" "d177" "d290" "d2102" ...
 $ Color                   : chr [1:10942] "GRAY" "GRAY" "BLUE" "BLACK" ...
 $ Transmission            : chr [1:10942] "Automatic" "Automatic" "Automatic" "Automatic" ...
 $ mileage                 : num [1:10942] 25111 94774 179200 26438 93724 ...
 $ Price                   : num [1:10942] 36990 20990 21990 53580 25995 ...
 $ bodyTypeName            : chr [1:10942] "Pickup Truck" "Minivan" "SUV / Crossover" "Pickup Truck" ...
 $ FuelType                : chr [1:10942] "Flex Fuel Vehicle" "Gasoline" "Gasoline" "Gasoline" ...
 $ SellerState             : chr [1:10942] "IL" "IL" "IL" "IL" ...
 $ MSRP                    : num [1:10942] 37990 20990 NA NA NA ...
 $ DaysOnMarket            : num [1:10942] 211 11 203 37 9 232 68 23 69 23 ...
 $ OwnerCount              : num [1:10942] 1 1 2 2 1 1 1 1 1 1 ...
 $ AccidentCount           : num [1:10942] 0 1 0 0 0 0 0 0 0 0 ...
 $ Alloy Wheels            : num [1:10942] 1 1 0 0 1 1 1 1 0 1 ...
 $ Backup Camera           : num [1:10942] 1 1 1 1 0 1 1 1 1 1 ...
 $ Bluetooth               : num [1:10942] 1 1 1 1 1 1 1 0 1 1 ...
 $ Heated Seats            : num [1:10942] 1 0 1 1 0 1 1 0 0 1 ...
 $ Navigation System       : num [1:10942] 0 0 1 0 0 1 1 0 1 1 ...
 $ Leather Seats           : num [1:10942] 0 0 1 0 0 1 1 0 0 1 ...
 $ Blind Spot Monitoring   : num [1:10942] 0 0 0 1 0 1 0 0 1 1 ...
 $ Sunroof/Moonroof        : num [1:10942] 0 0 1 0 0 0 0 0 1 0 ...
 $ Remote Start            : num [1:10942] 1 1 0 1 0 1 1 1 1 0 ...
 $ CarPlay                 : num [1:10942] 1 0 0 0 0 0 0 0 1 1 ...
 $ Android Auto            : num [1:10942] 0 0 0 0 0 0 0 0 1 1 ...
 $ Parking Sensors         : num [1:10942] 0 0 0 1 0 1 1 0 0 0 ...
 $ Adaptive Cruise Control: num [1:10942] 0 0 0 0 0 1 0 0 0 0 ...
 $ YearOld                 : num [1:10942] 4 5 10 3 5 4 5 2 3 5 ...
 $ id                      : num [1:10942] 0 1 2 3 4 5 6 7 8 9 ...
```

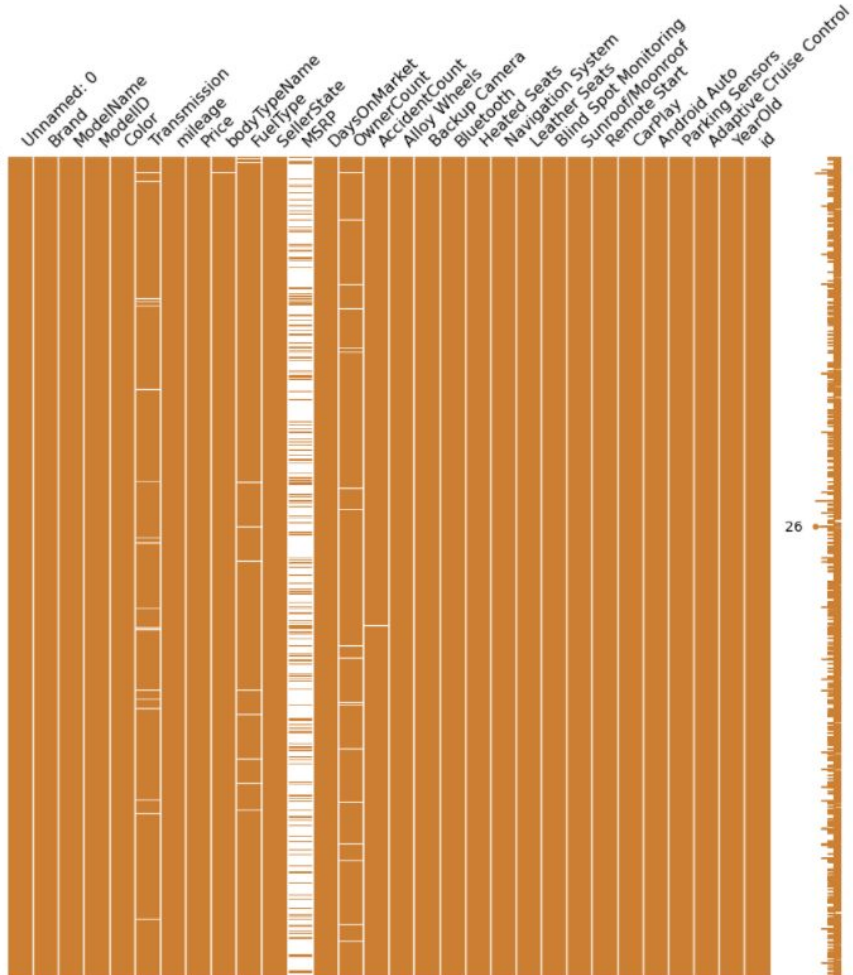
Checking Null Value

- There are 6 columns that has Null Value

- MSRP(manufacturer's suggested retail price) has 76.12% Null value

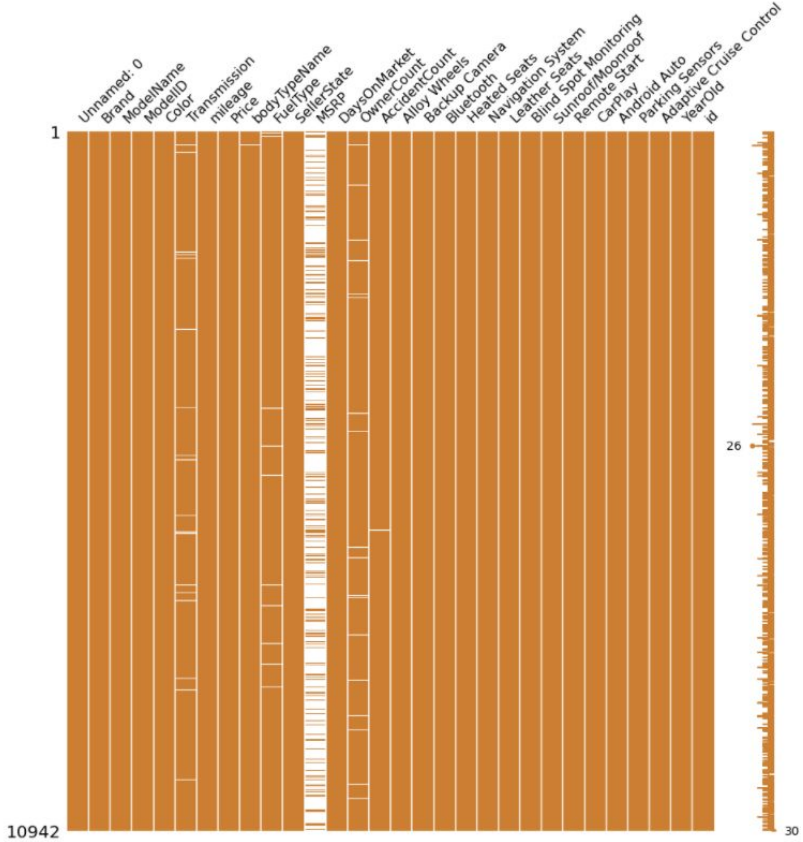
```
column: Unnamed: 0       Percent of NaN value: 0.00%
column:      Brand       Percent of NaN value: 0.00%
column:  ModelName       Percent of NaN value: 0.00%
column:    ModelID       Percent of NaN value: 0.00%
column:      Color       Percent of NaN value: 0.00%
column: Transmission     Percent of NaN value: 2.39%
column:    mileage       Percent of NaN value: 0.00%
column:      Price       Percent of NaN value: 0.00%
column: bodyTypeName     Percent of NaN value: 0.13%
column:   FuelType       Percent of NaN value: 1.40%
column: SellerState      Percent of NaN value: 0.00%
column:       MSRP       Percent of NaN value: 76.12%
column: DaysOnMarket     Percent of NaN value: 0.00%
column: OwnerCount       Percent of NaN value: 2.67%
column: AccidentCount    Percent of NaN value: 0.15%
column: Alloy Wheels     Percent of NaN value: 0.00%
column: Backup Camera    Percent of NaN value: 0.00%
column:  Bluetooth       Percent of NaN value: 0.00%
column: Heated Seats     Percent of NaN value: 0.00%
column: Navigation System        Percent of NaN value: 0.00%
column: Leather Seats    Percent of NaN value: 0.00%
column: Blind Spot Monitoring    Percent of NaN value: 0.00%
column: Sunroof/Moonroof         Percent of NaN value: 0.00%
column: Remote Start     Percent of NaN value: 0.00%
column:    CarPlay       Percent of NaN value: 0.00%
column: Android Auto     Percent of NaN value: 0.00%
column: Parking Sensors  Percent of NaN value: 0.00%
column: Adaptive Cruise Control  Percent of NaN value: 0.00%
column:    YearOld       Percent of NaN value: 0.00%
column:         id       Percent of NaN value: 0.00%
```


10942
10942
10942
10942
10942
10942
10680
10942
10942
10928
10789
10942
2613
10942
10650
10926
10942
10942
10942
10942
10942
10942
10942
10942
10942
10942
10942
10942
10942
10942
1.0
0.8
0.6
0.4
0.2
0.0
10942
8753
6565
4376
2188
0
Unnamed: 0
Brand
ModelName
ModelID
Color
Transmission
mileage
Price
bodyTypeName
FuelType
SellerState
MSRP
DaysOnMarket
OwnerCount
AccidentCount
Alloy Wheels
Backup Camera
Bluetooth
Heated Seats
Navigation System
Leather Seats
Blind Spot Monitoring
Sunroof/Moonroof
Remote Start
CarPlay
Android Auto
Parking Sensors
Adaptive Cruise Control
YearOld
id

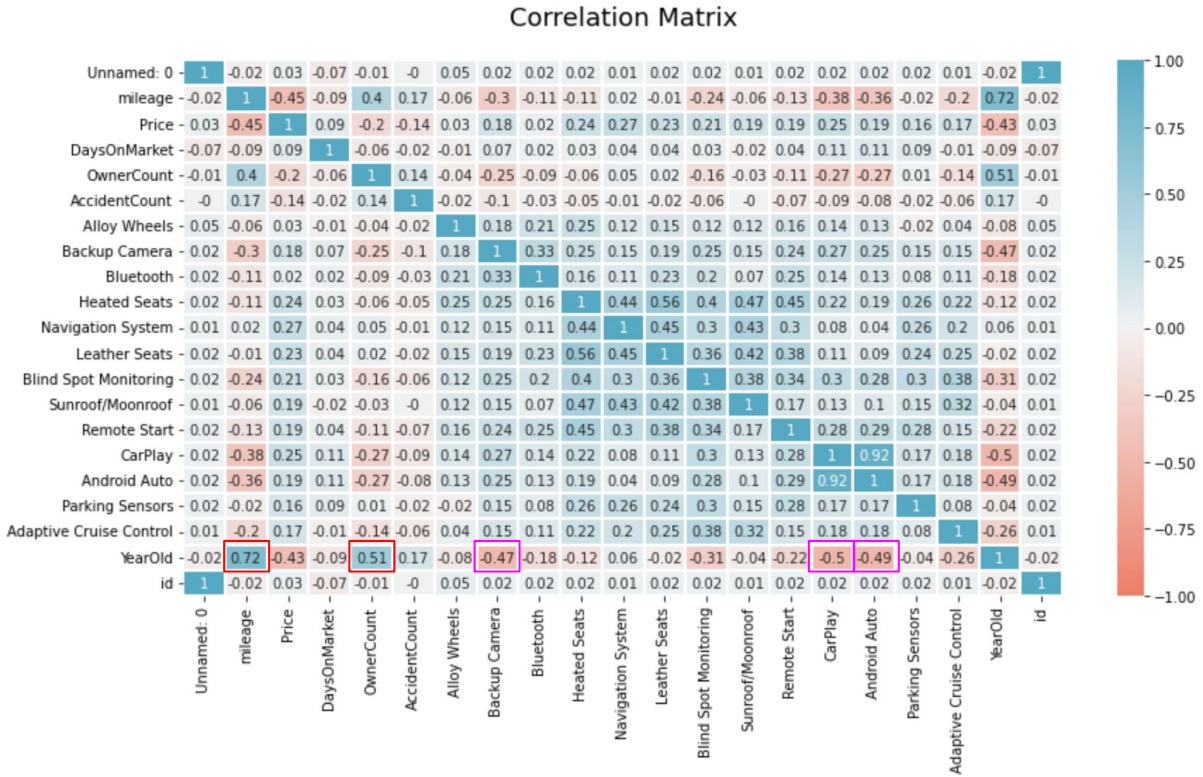

Unnamed: 0
Brand
ModelName
ModelID
Color
Transmission
mileage
Price
bodyTypeName
FuelType
SellerState
MSRP
DaysOnMarket
OwnerCount
AccidentCount
Alloy Wheels
Backup Camera
Bluetooth
Heated Seats
Navigation System
Leather Seats
Blind Spot Monitoring
Sunroof/Moonroof
Remote Start
CarPlay
Android Auto
Parking Sensors
Adaptive Cruise Control
YearOld
id
1
26

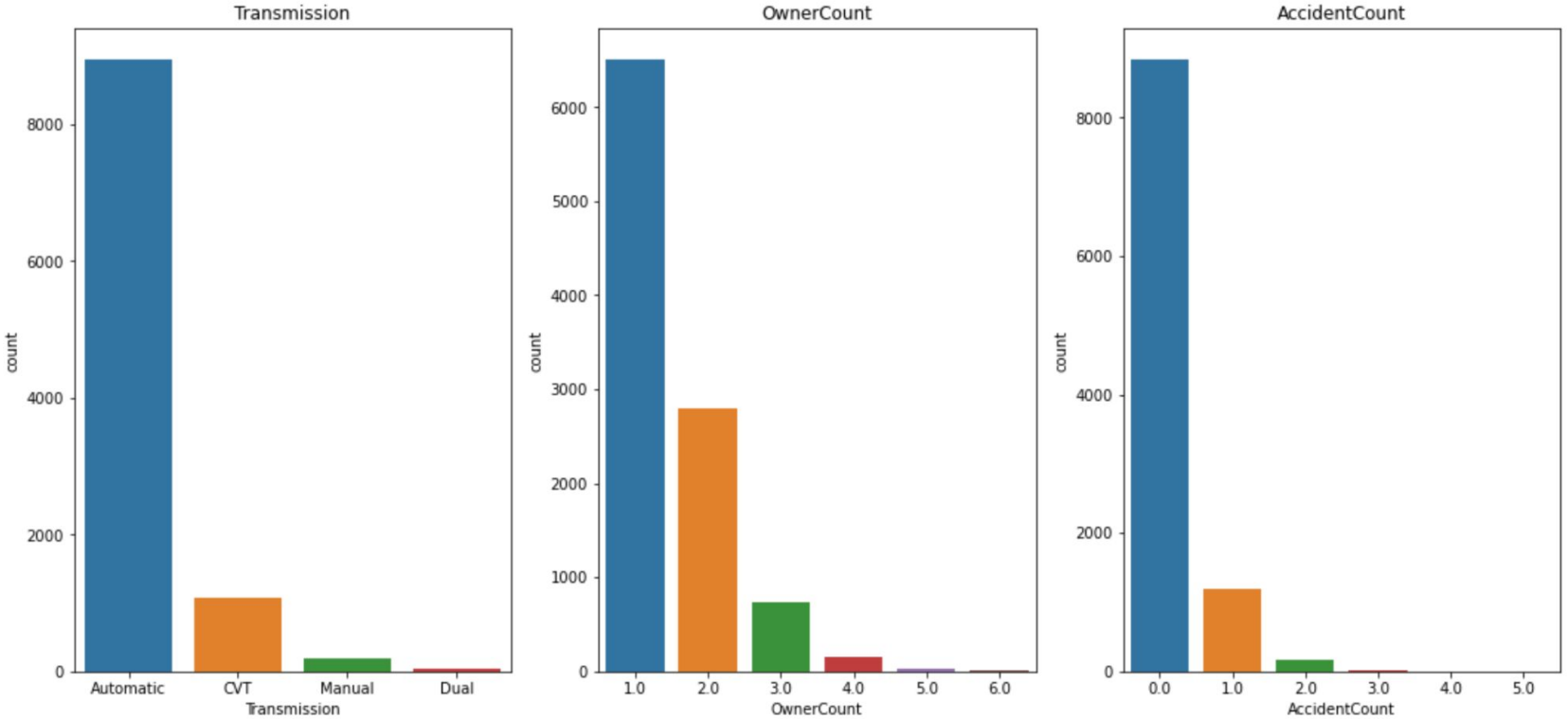
Recap from Week 1

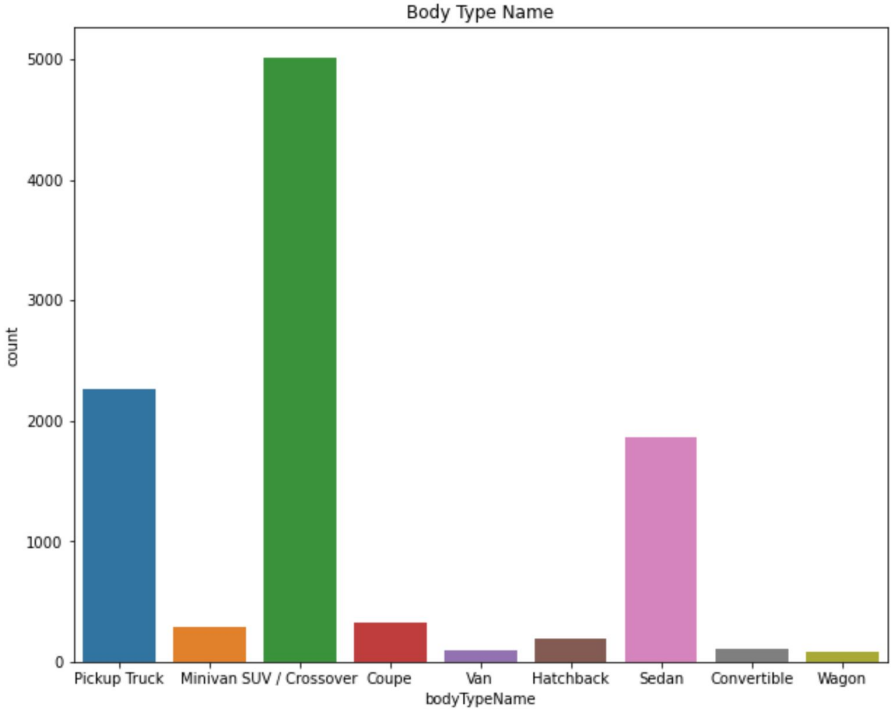
1. **Data extracted** from ***Cargurus.com*** using web scraping method
2. **Checked** Null Data
    a. **76.12%** of null data for MSRP (manufacturer's suggested retail price)

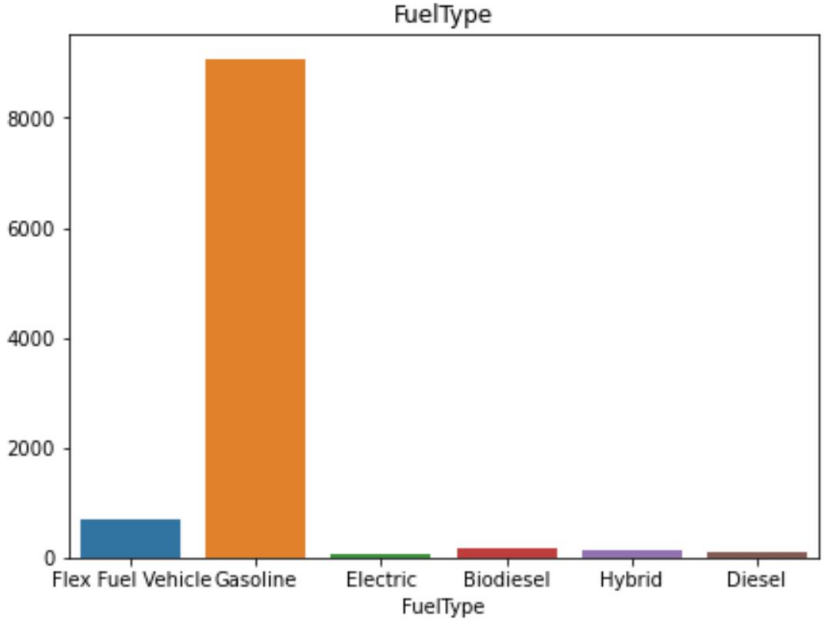


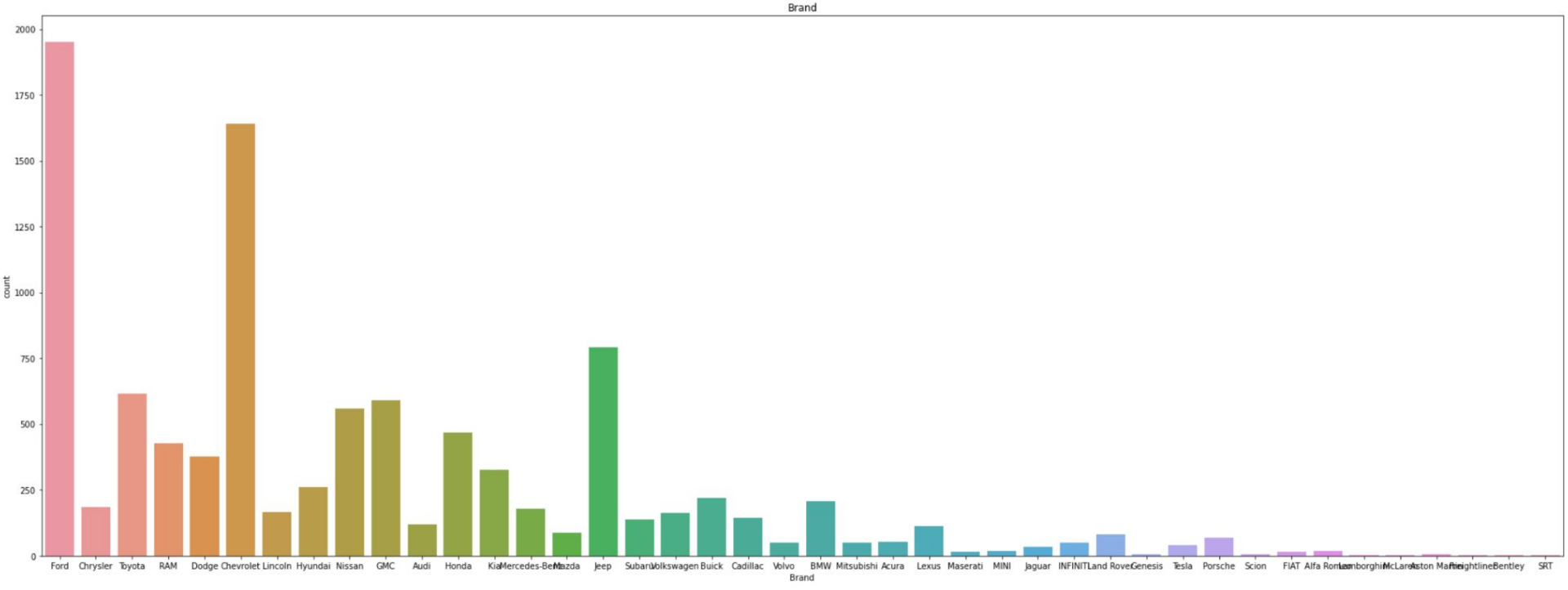
EDA


Correlation Matrix
Unnamed: 0 1 -0.02 0.03 -0.07 -0.01 -0 0.05 0.02 0.02 0.02 0.01 0.02 0.02 0.01 0.02 0.02 0.02 0.02 0.01 -0.02 1
mileage -0.02 1 -0.45 -0.09 0.4 0.17 -0.06 -0.3 -0.11 -0.11 0.02 -0.01 -0.24 -0.06 -0.13 -0.38 -0.36 -0.02 -0.2 0.72 -0.02
Price 0.03 -0.45 1 0.09 -0.2 -0.14 0.03 0.18 0.02 0.24 0.27 0.23 0.21 0.19 0.19 0.25 0.19 0.16 0.17 -0.43 0.03
DaysOnMarket -0.07 -0.09 0.09 1 -0.06 -0.02 -0.01 0.07 0.02 0.03 0.04 0.04 0.03 -0.02 0.04 0.11 0.11 0.09 -0.01 -0.09 -0.07
OwnerCount -0.01 0.4 -0.2 -0.06 1 0.14 -0.04 -0.25 -0.09 -0.06 0.05 0.02 -0.16 -0.03 -0.11 -0.27 -0.27 0.01 -0.14 0.51 -0.01
AccidentCount -0 0.17 -0.14 -0.02 0.14 1 -0.02 -0.1 -0.03 -0.05 -0.01 -0.02 -0.06 -0 -0.07 -0.09 -0.08 -0.02 -0.06 0.17 -0
Alloy Wheels 0.05 -0.06 0.03 -0.01 -0.04 -0.02 1 0.18 0.21 0.25 0.12 0.15 0.12 0.12 0.16 0.14 0.13 -0.02 0.04 -0.08 0.05
Backup Camera 0.02 -0.3 0.18 0.07 -0.25 -0.1 0.18 1 0.33 0.25 0.15 0.19 0.25 0.15 0.24 0.27 0.25 0.15 0.15 -0.47 0.02
Bluetooth 0.02 -0.11 0.02 0.02 -0.09 -0.03 0.21 0.33 1 0.16 0.11 0.23 0.2 0.07 0.25 0.14 0.13 0.08 0.11 -0.18 0.02
Heated Seats 0.02 -0.11 0.24 0.03 -0.06 -0.05 0.25 0.25 0.16 1 0.44 0.56 0.4 0.47 0.45 0.22 0.19 0.26 0.22 -0.12 0.02
Navigation System 0.01 0.02 0.27 0.04 0.05 -0.01 0.12 0.15 0.11 0.44 1 0.45 0.3 0.43 0.3 0.08 0.04 0.26 0.2 0.06 0.01
Leather Seats 0.02 -0.01 0.23 0.04 0.02 -0.02 0.15 0.19 0.23 0.56 0.45 1 0.36 0.42 0.38 0.11 0.09 0.24 0.25 -0.02 0.02
Blind Spot Monitoring 0.02 -0.24 0.21 0.03 -0.16 -0.06 0.12 0.25 0.2 0.4 0.3 0.36 1 0.38 0.34 0.3 0.28 0.3 0.38 -0.31 0.02
Sunroof/Moonroof 0.01 -0.06 0.19 -0.02 -0.03 -0 0.12 0.15 0.07 0.47 0.43 0.42 0.38 1 0.17 0.13 0.1 0.15 0.32 -0.04 0.01
Remote Start 0.02 -0.13 0.19 0.04 -0.11 -0.07 0.16 0.24 0.25 0.45 0.3 0.38 0.34 0.17 1 0.28 0.29 0.28 0.15 -0.22 0.02
CarPlay 0.02 -0.38 0.25 0.11 -0.27 -0.09 0.14 0.27 0.14 0.22 0.08 0.11 0.3 0.13 0.28 1 0.92 0.17 0.18 -0.5 0.02
Android Auto 0.02 -0.36 0.19 0.11 -0.27 -0.08 0.13 0.25 0.13 0.19 0.04 0.09 0.28 0.1 0.29 0.92 1 0.17 0.18 -0.49 0.02
Parking Sensors 0.02 -0.02 0.16 0.09 0.01 -0.02 -0.02 0.15 0.08 0.26 0.26 0.24 0.3 0.15 0.28 0.17 0.17 1 0.08 -0.04 0.02
Adaptive Cruise Control 0.01 -0.2 0.17 -0.01 -0.14 -0.06 0.04 0.15 0.11 0.22 0.2 0.25 0.38 0.32 0.15 0.18 0.18 0.08 1 -0.26 0.01
YearOld -0.02 0.72 -0.43 -0.09 0.51 0.17 -0.08 -0.47 -0.18 -0.12 0.06 -0.02 -0.31 -0.04 -0.22 -0.5 -0.49 -0.04 -0.26 1 -0.02
id 1 -0.02 0.03 -0.07 -0.01 -0 0.05 0.02 0.02 0.02 0.01 0.02 0.02 0.01 0.02 0.02 0.02 0.02 0.01 -0.02 1
Unnamed: 0
mileage
Price
DaysOnMarket
OwnerCount
AccidentCount
Alloy Wheels
Backup Camera
Bluetooth
Heated Seats
Navigation System
Leather Seats
Blind Spot Monitoring
Sunroof/Moonroof
Remote Start
CarPlay
Android Auto
Parking Sensors
Adaptive Cruise Control
YearOld
id
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Transmission
count
8000
6000
4000
2000
0
Automatic
CVT
Manual
Dual
Transmission
OwnerCount
count
6000
5000
4000
3000
2000
1000
0
1.0
2.0
3.0
4.0
5.0
6.0
OwnerCount
AccidentCount
count
8000
6000
4000
2000
0
0.0
1.0
2.0
3.0
4.0
5.0
AccidentCount


Body Type Name
5000
4000
3000
2000
1000
0
count
Pickup Truck
Minivan
SUV / Crossover
Coupe
Van
Hatchback
Sedan
Convertible
Wagon
bodyTypeName

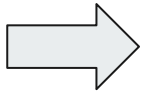

FuelType
8000
6000
4000
2000
0
Flex Fuel Vehicle
Gasoline
Electric
Biodiesel
Hybrid
Diesel
FuelType


Brand
count
2000
1750
1500
1250
1000
750
500
250
0
Ford
Chrysler
Toyota
RAM
Dodge
Chevrolet
Lincoln
Hyundai
Nissan
GMC
Audi
Honda
Kia
Mercedes-Benz
Mazda
Jeep
Subaru
Volkswagen
Buick
Cadillac
Volvo
BMW
Mitsubishi
Acura
Lexus
Maserati
MINI
Jaguar
INFINITI
Land Rover
Genesis
Tesla
Porsche
Scion
FIAT
Alfa Romeo
Lamborghini
McLaren
Aston Martin
Freightliner
Bentley
SRT
Brand

Data Cleaning

- **Dropped** MSRP and all the rows that have null value.
  - Dropped 703 entry from 10,942 u car data
- 0 % null value for all the columns after data cleaning

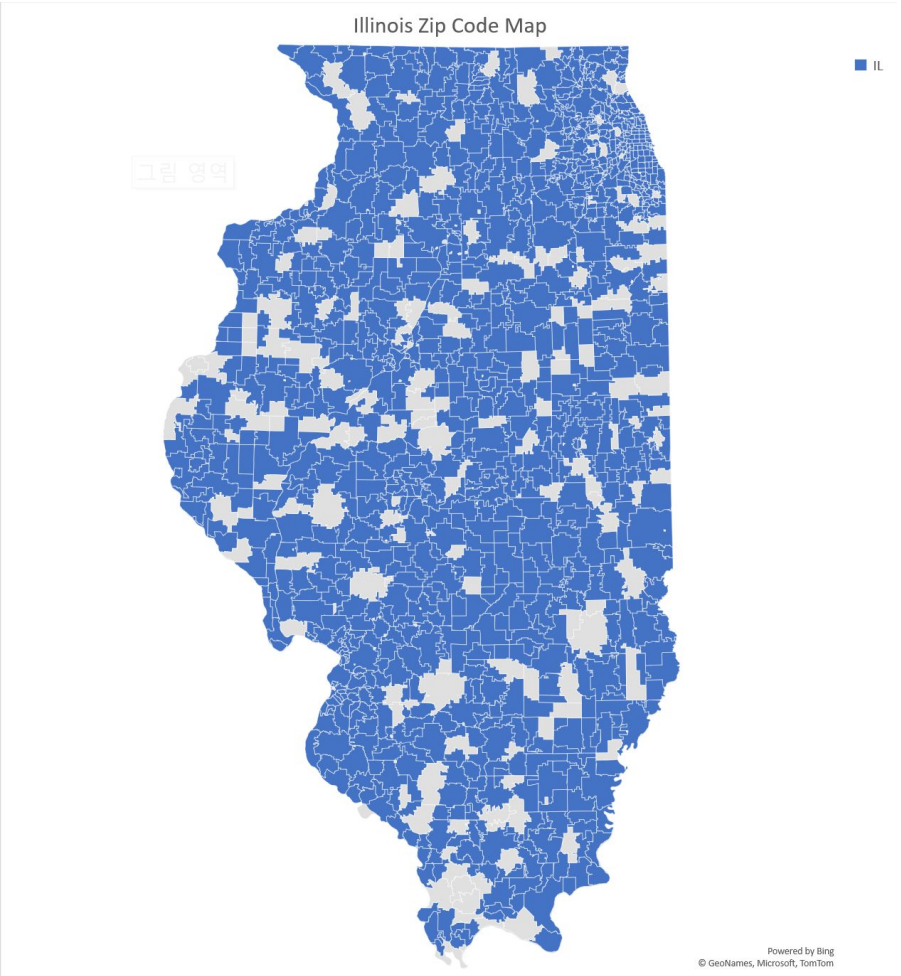
```
column: Unnamed: 0       Percent of NaN value: 0.00%
column:      Brand       Percent of NaN value: 0.00%
column:  ModelName       Percent of NaN value: 0.00%
column:    ModelID       Percent of NaN value: 0.00%
column:      Color       Percent of NaN value: 0.00%
column: Transmission     Percent of NaN value: 2.39%
column:    mileage       Percent of NaN value: 0.00%
column:      Price       Percent of NaN value: 0.00%
column: bodyTypeName     Percent of NaN value: 0.13%
column:   FuelType       Percent of NaN value: 1.40%
column: SellerState      Percent of NaN value: 0.00%
column:       MSRP       Percent of NaN value: 76.12%
column: DaysOnMarket     Percent of NaN value: 0.00%
column: OwnerCount       Percent of NaN value: 2.67%
column: AccidentCount    Percent of NaN value: 0.15%
column: Alloy Wheels     Percent of NaN value: 0.00%
column: Backup Camera    Percent of NaN value: 0.00%
column:  Bluetooth       Percent of NaN value: 0.00%
column: Heated Seats     Percent of NaN value: 0.00%
column: Navigation System        Percent of NaN value: 0.00%
column: Leather Seats    Percent of NaN value: 0.00%
column: Blind Spot Monitoring    Percent of NaN value: 0.00%
column: Sunroof/Moonroof         Percent of NaN value: 0.00%
column: Remote Start     Percent of NaN value: 0.00%
column:    CarPlay       Percent of NaN value: 0.00%
column: Android Auto     Percent of NaN value: 0.00%
column: Parking Sensors  Percent of NaN value: 0.00%
column: Adaptive Cruise Control  Percent of NaN value: 0.00%
column:    YearOld       Percent of NaN value: 0.00%
column:         id       Percent of NaN value: 0.00%
```

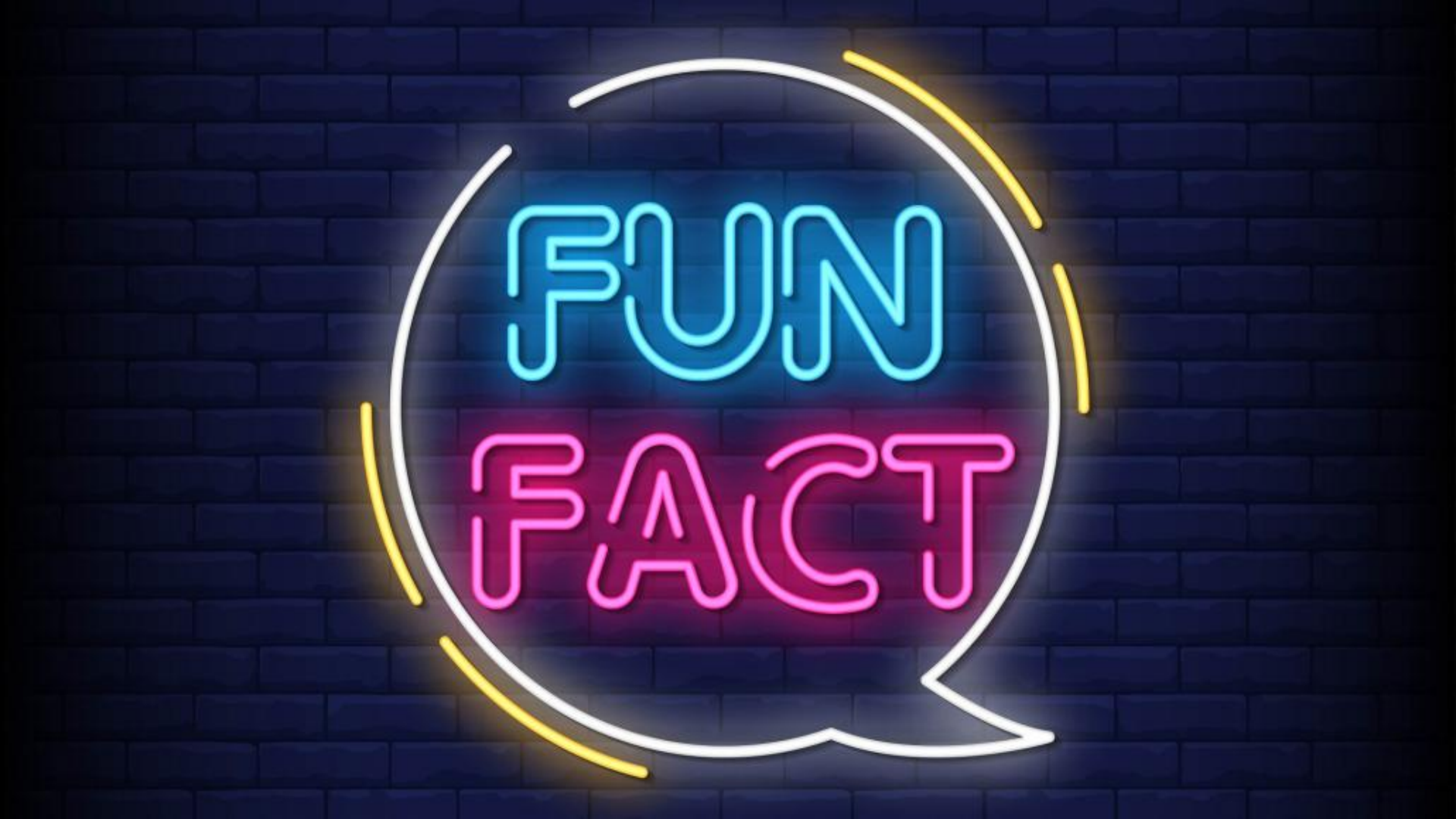


```
column: Unnamed: 0       Percent of NaN value: 0.00%
column:      Brand       Percent of NaN value: 0.00%
column:  ModelName       Percent of NaN value: 0.00%
column:    ModelID       Percent of NaN value: 0.00%
column:      Color       Percent of NaN value: 0.00%
column: Transmission     Percent of NaN value: 0.00%
column:    mileage       Percent of NaN value: 0.00%
column:      Price       Percent of NaN value: 0.00%
column: bodyTypeName     Percent of NaN value: 0.00%
column:   FuelType       Percent of NaN value: 0.00%
column: SellerState      Percent of NaN value: 0.00%
column: DaysOnMarket     Percent of NaN value: 0.00%
column: OwnerCount       Percent of NaN value: 0.00%
column: AccidentCount    Percent of NaN value: 0.00%
column: Alloy Wheels     Percent of NaN value: 0.00%
column: Backup Camera    Percent of NaN value: 0.00%
column:  Bluetooth       Percent of NaN value: 0.00%
column: Heated Seats     Percent of NaN value: 0.00%
column: Navigation System        Percent of NaN value: 0.00%
column: Leather Seats    Percent of NaN value: 0.00%
column: Blind Spot Monitoring    Percent of NaN value: 0.00%
column: Sunroof/Moonroof         Percent of NaN value: 0.00%
column: Remote Start     Percent of NaN value: 0.00%
column:    CarPlay       Percent of NaN value: 0.00%
column: Android Auto     Percent of NaN value: 0.00%
column: Parking Sensors  Percent of NaN value: 0.00%
column: Adaptive Cruise Control  Percent of NaN value: 0.00%
column:    YearOld       Percent of NaN value: 0.00%
column:         id       Percent of NaN value: 0.00%
```

Zip Codes Mapping

- This is the map of Illinois zip codes that we randomly selected
- Used car datas are from these zip codes




FUN
FACT

Brand	ModelName	ModelID	Color	Transmission	mileage	Price	bodyTypeName	FuelType	SellerState	MSRP	DaysOnMarket	OwnerCount
Ford	Focus	d346	GRAY	NaN	168252.0	3990.0	Sedan	Gasoline	IN	NaN	17	2.0

AccidentCount	Alloy Wheels	Backup Camera	Bluetooth	Heated Seats	Navigation System	Leather Seats	Blind Spot Monitoring	Sunroof/Moonroof	Remote Start	CarPlay	Android Auto	Parking Sensors	Adaptive Cruise Control
0.0	1	0	1	0	1	0	0	0	0	0	0	0	0

YearOld
11

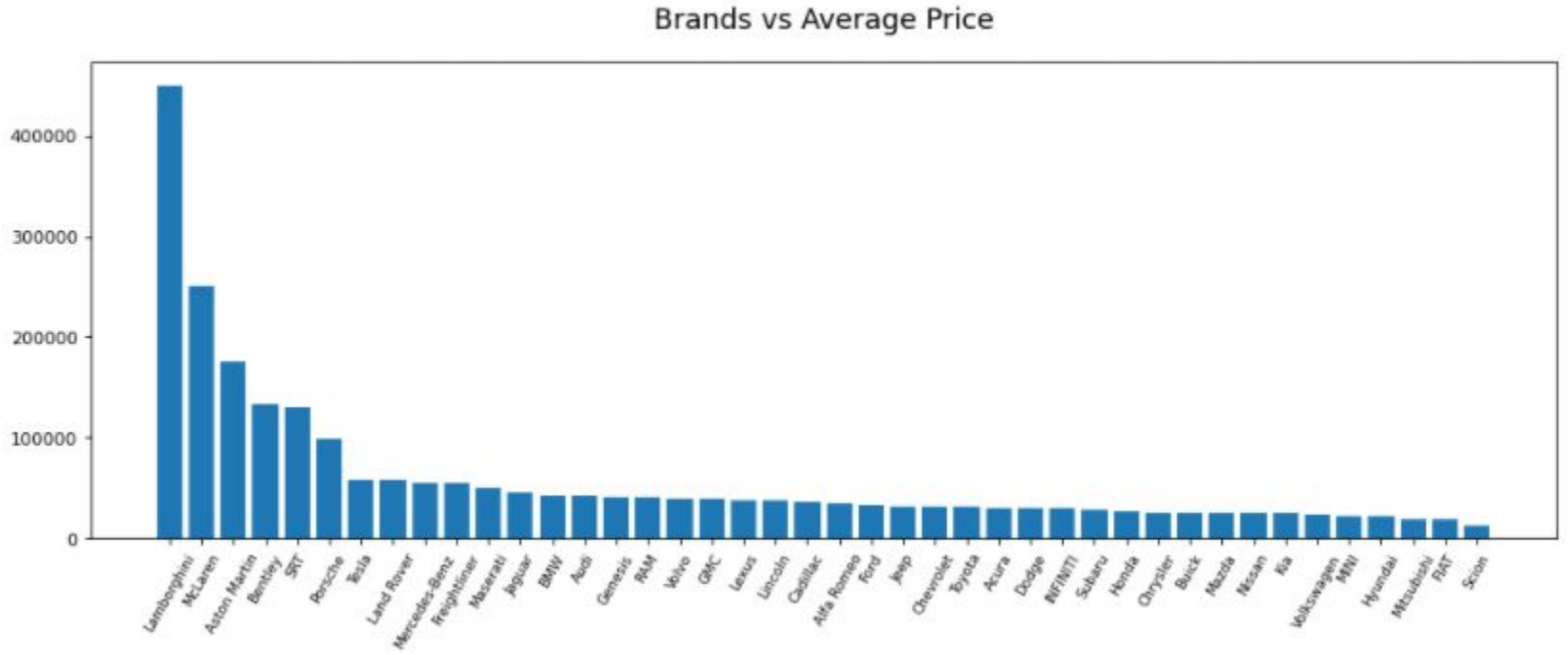
Brand	ModelName	ModelID	Color	Transmission	mileage	Price	bodyTypeName	FuelType	SellerState	MSRP	DaysOnMarket	OwnerCount	YearOld
Lamborghini	Aventador	d2125	ORANGE	Automatic	13077.0	450000.0	Coupe	Gasoline	IL	NaN	121	2.0	6

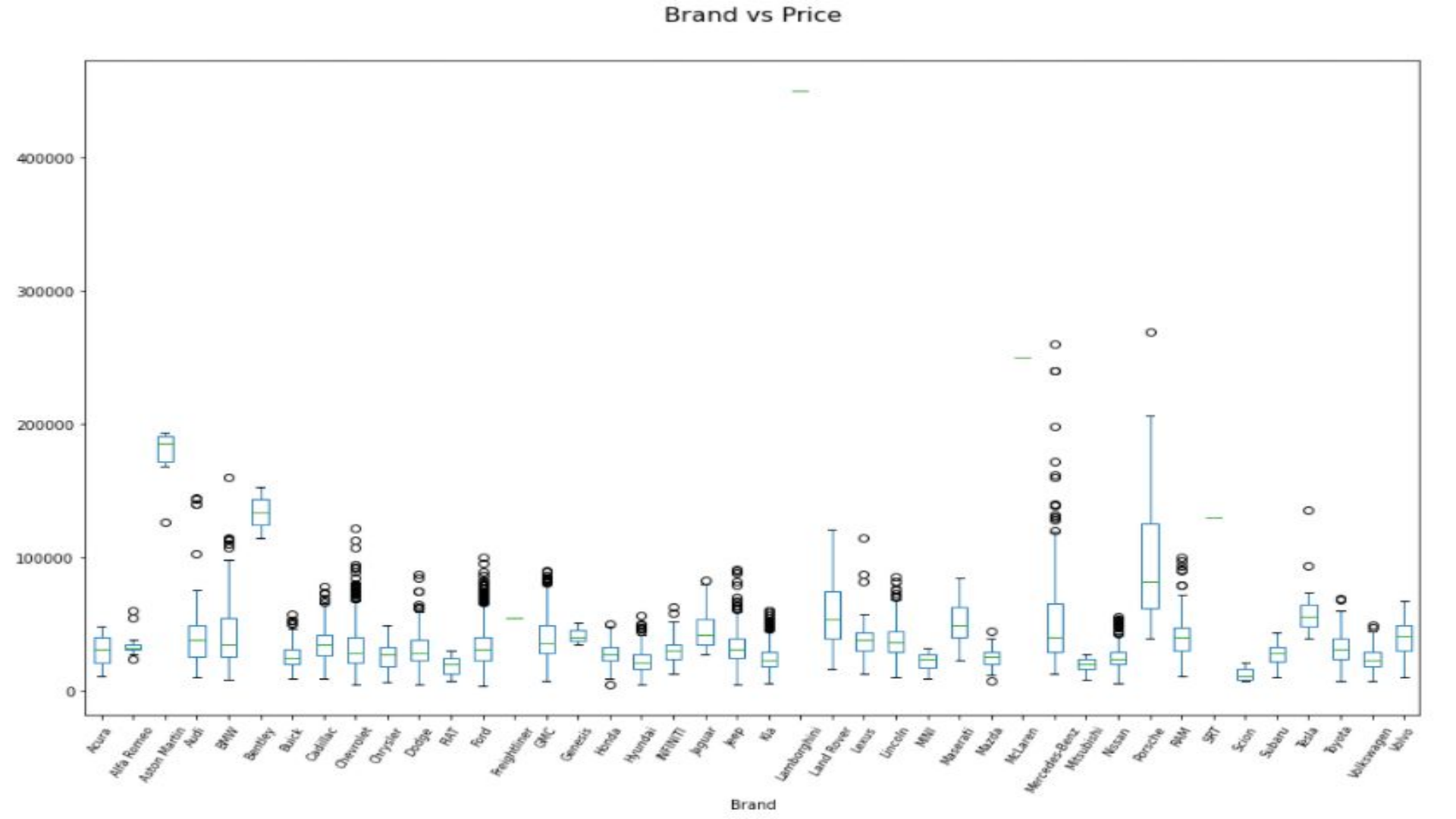
AccidentCount	Alloy Wheels	Backup Camera	Bluetooth	Heated Seats	Navigation System	Leather Seats	Blind Spot Monitoring	Sunroof/Moonroof	Remote Start	CarPlay	Android Auto	Parking Sensors	Adaptive Cruise Control
0.0	1	0	0	0	1	0	0	0	0	0	0	0	0

JIDDMOTORS.COM
1313 RAND RD, DES PLAINES, IL 60016

LEATHER INTERIOR AWD NAVIGATION POWER WINDOWS POWER DOOR LOCKS A/C

Check Target(Price) Based Distribution




Brand vs Price
400000
300000
200000
100000
0
Acura
Alfa Romeo
Aston Martin
Audi
BMW
Bentley
Buick
Cadillac
Chevrolet
Chrysler
Dodge
FIAT
Ford
Freightliner
GMC
Genesis
Honda
Hyundai
INFINITI
Jaguar
Jeep
Kia
Lamborghini
Land Rover
Lexus
Lincoln
MINI
Maserati
Mazda
McLaren
Mercedes-Benz
Mitsubishi
Nissan
Porsche
RAM
SRT
Scion
Subaru
Tesla
Toyota
Volkswagen
Volvo
Brand

Dropping Unnecessary Columns and Split the Data

- Drop Price column
- Drop Seller State because there are other States besides IL because zipcodes we used could be used at another state or another possibility is that region near the borderline of the States dealer posted their vehicle at Illinois.
- Split the dataset with the ratio of 70:30
- Check if model ID and model name match to see if we can drop model name
- Result came out true, so drop model ID

```
# Train test split - 30% of whole dataset into test data
X = df.drop(['Price', 'SellerState'], axis= 1)
y = df.Price
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123456789)
```

```
#Check if model id and model name match
df.loc[df.ModelID == 'd622', 'ModelName']

# Drop modelname because modelname can be found using modelid
X_train.drop(['id','ModelName'], axis=1, inplace=True)
X_test.drop(['id','ModelName'], axis=1, inplace=True)
```

Target Encoding and One hot encoding

- Replace columns that has categorical value with the mean of target value : Brand, ModelID, Color, bodyTypeName
- Apply one hot encoding with Transmission and FuelType

```python
# Target Encoding
target_enc_cols = ['Brand', 'ModelID', 'Color', 'bodyTypeName']
enc = ce.target_encoder.TargetEncoder(return_df=True, handle_unknown='value')
X_train[target_enc_cols] = enc.fit_transform(X_train[target_enc_cols], y_train)
X_test[target_enc_cols] = enc.transform(X_test[target_enc_cols])
```

```python
# One Hot Encoding
one_hot_cols = ['Transmission','FuelType']
enc = OneHotEncoder(handle_unknown='ignore')
encoded = enc.fit_transform(X_train[one_hot_cols]).toarray()
X_train[enc.get_feature_names_out()] = encoded
X_train.drop(one_hot_cols, axis=1, inplace=True)

encoded = enc.transform(X_test[one_hot_cols]).toarray()
X_test[enc.get_feature_names_out()] = encoded
X_test.drop(one_hot_cols, axis=1, inplace=True)
```

Brand	ModelID	Color	mileage	bodyTypeName	DaysOnMarket	OwnerCount	AccidentCount	Alloy Wheels	Backup Camera	...	Transmission_Automatic	Transmission_CVT	Transmission_Dual	Transmission_Manual	FuelType_Biodiesel	FuelType_Diesel	FuelType_Electric	FuelType_Flex Fuel Vehicle	Fuel
25092.975104	19523.001522	35068.213720	37987.0	25714.477992	49	1.0	1.0	1	1	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
31453.737557	38858.600000	35068.213720	30211.0	32503.244747	68	1.0	0.0	1	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
31453.737557	36155.017241	34354.042976	26871.0	32503.244747	16	1.0	0.0	1	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
24625.203704	18340.347830	34354.042976	34158.0	25714.477992	131	2.0	0.0	1	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
33004.947761	32388.273973	34354.042976	71760.0	32503.244747	42	3.0	1.0	0	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
33004.947761	23162.576923	34354.042976	21852.0	32503.244747	90	1.0	0.0	0	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
31498.822026	19691.069767	35068.213720	48427.0	25714.477992	27	2.0	0.0	1	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
40024.953177	37219.842342	34354.042976	34477.0	39998.255251	207	1.0	0.0	1	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
27855.642229	24039.288136	31400.783534	85383.0	25714.477992	132	1.0	0.0	1	1	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
40024.953177	37219.842342	33218.560827	88206.0	39998.255251	89	2.0	0.0	0	0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

Model Fitting

1. Linear Regression

```
#modeling Linear Regression

model = LinearRegression(fit_intercept=True, n_jobs=4)
reg = model.fit(X_train, y_train)
pred = model.predict(X_test)

#R^2 score
print("Linear Regression R^2 Score for test set: ", reg.score(X_test, y_test))
print("Linear Regression MSE: ", mean_squared_error(pred, y_test))
print("Linear Regression MAE: ", mean_absolute_error(pred, y_test))
```

```
Linear Regression R^2 Score for test set:  0.7936733576176747
Linear Regression MSE:  72671238.66087796
Linear Regression MAE:  4553.894298360479
```

```
model.intercept_

model.coef_.shape

#unable to visualize because it is 32 dimension
```

```
(32,)
```

- Unable to visualize because we have 32 parameters.

2. Polynomial Regression Degree 2

```
# Polynomial Regression - degree 2
poly = PolynomialFeatures(degree= 2)
X_train_ = poly.fit_transform(X_train)
X_test_ = poly.fit_transform(X_test)

model = LinearRegression(fit_intercept=True, n_jobs=4)
reg = model.fit(X_train_, y_train)
pred = model.predict(X_test_)

print("Polynomial Regression R^2 Score for test set: ", reg.score(X_test_, y_test))
print("Polynomial Regression MSE: ", mean_squared_error(pred, y_test))
print("Polynomial Regression MAE: ", mean_absolute_error(pred, y_test))
```

```
Polynomial Regression R^2 Score for test set:  0.8232675511760047
Polynomial Regression MSE:  62247734.07503561
Polynomial Regression MAE:  4477.578155412296
```

Polynomial Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

3. Random Forest

```python
# With the best parameter
rf = RandomForestRegressor(n_estimators= 150, min_samples_split=2, max_depth=None, random_state=42)
rf.fit(X_train, y_train)
pred = rf.predict(X_test)
print("Test MSE: ", mean_squared_error(pred, y_test))
print("Test MAE: ", mean_absolute_error(pred, y_test))

pred = rf.predict(X_train)
print("Train MSE: ", mean_squared_error(pred, y_train))
print("Train MAE: ", mean_absolute_error(pred, y_train))

# Show feature importances
feature_dict = dict(zip(rf.feature_names_in_, rf.feature_importances_.round(4)))
display(sorted(feature_dict.items(), key=lambda x:x[1], reverse=True))
```

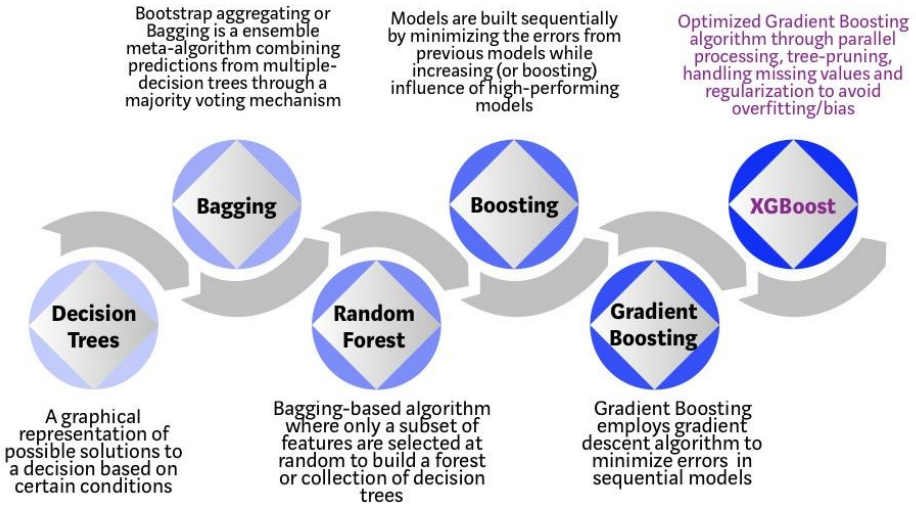
```
Test MSE:  47375925.13666701
Test MAE:  3504.609691049727
Train MSE:  7823178.279621231
Train MAE:  1254.8186176855559
[('ModelID', 0.5816),
 ('mileage', 0.1488),
 ('YearOld', 0.0876),
 ('Color', 0.04),
 ('DaysOnMarket', 0.0361),
 ('Brand', 0.0176),
 ('OwnerCount', 0.0128),
 ('bodyTypeName', 0.0104),
 ('Leather Seats', 0.0098),
 ('Backup Camera', 0.0068),
 ('Bluetooth', 0.0068),
 ('Heated Seats', 0.0052),
 ('Adaptive Cruise Control', 0.0051),
 ('Navigation System', 0.0043),
 ('Sunroof/Moonroof', 0.0039),
 ('Remote Start', 0.0031),
 ('Blind Spot Monitoring', 0.0028),
 ('CarPlay', 0.0028),
 ('Parking Sensors', 0.0028),
 ('Alloy Wheels', 0.0025),
 ('Android Auto', 0.0019),
 ('FuelType_Biodiesel', 0.0017),
 ('AccidentCount', 0.0012),
 ('FuelType_Gasoline', 0.0012),
 ('FuelType_Diesel', 0.001),
 ('FuelType_Flex Fuel Vehicle', 0.0007),
 ('FuelType_Electric', 0.0005),
 ('Transmission_Automatic', 0.0004),
 ('Transmission_Manual', 0.0003),
 ('Transmission_CVT', 0.0002),
 ('FuelType_Hybrid', 0.0001),
 ('Transmission_Dual', 0.0)]
```

```python
#accuracy
from sklearn.metrics import accuracy_score
print('정확도 : {:.4f}'.format(model.score(X_test, y_test)))
```

```
정확도 : 0.7937
```

Future Plans

Xgboost - decision-tree-based ensemble Machine Learning algorithm



XGBoost(Extreme Gradient Boost)

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

XGBoost(Extreme Gradient Boost)

Pros :

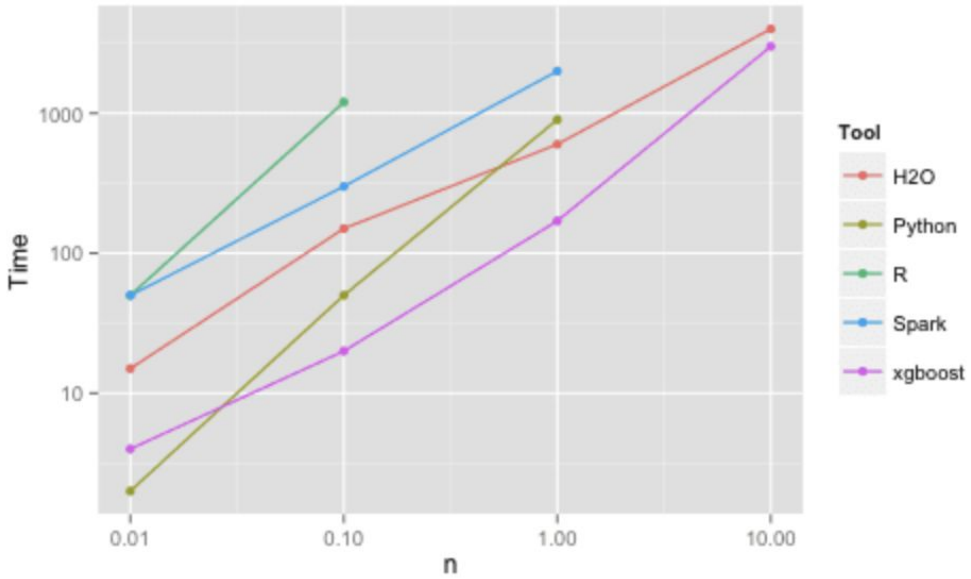
1. GBM Improved speed compare to GBM
   a. use parallel tree boosting
2. Efficient, Flexible, and Portable
3. Prevent overfitting
4. Optimal model can be made by tuning hyperparameter

Cons:

1. XGBoost doesn't work well on dispersed data
2. Sensitive to outliers
   a. XGBoost classifier is forced to fix every error
3. Complicated hyperparameter

Execution Speed

- benchmarks comparison Random Forest Implementation

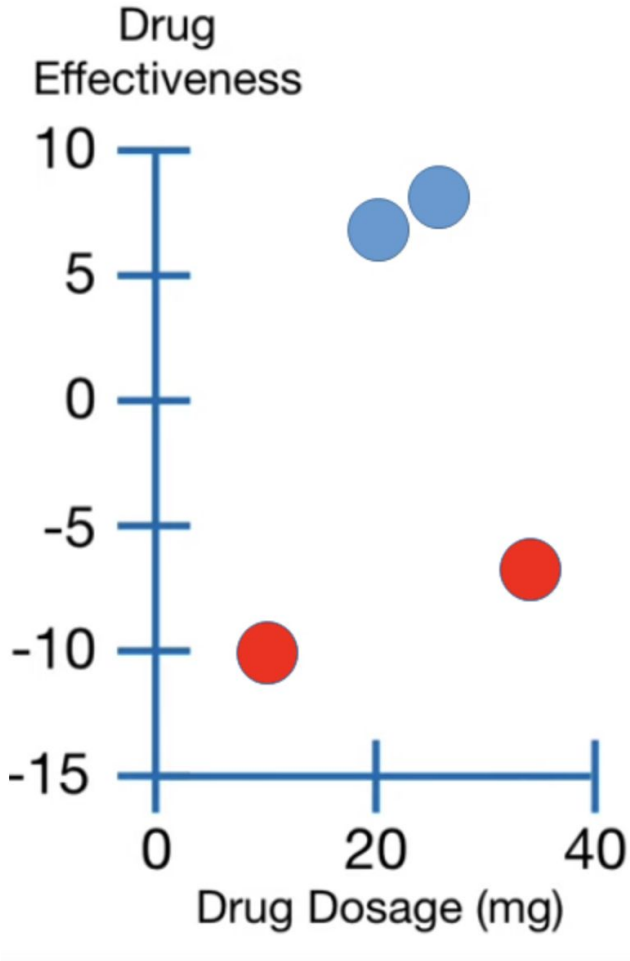


Example

Simple data with 4 observations

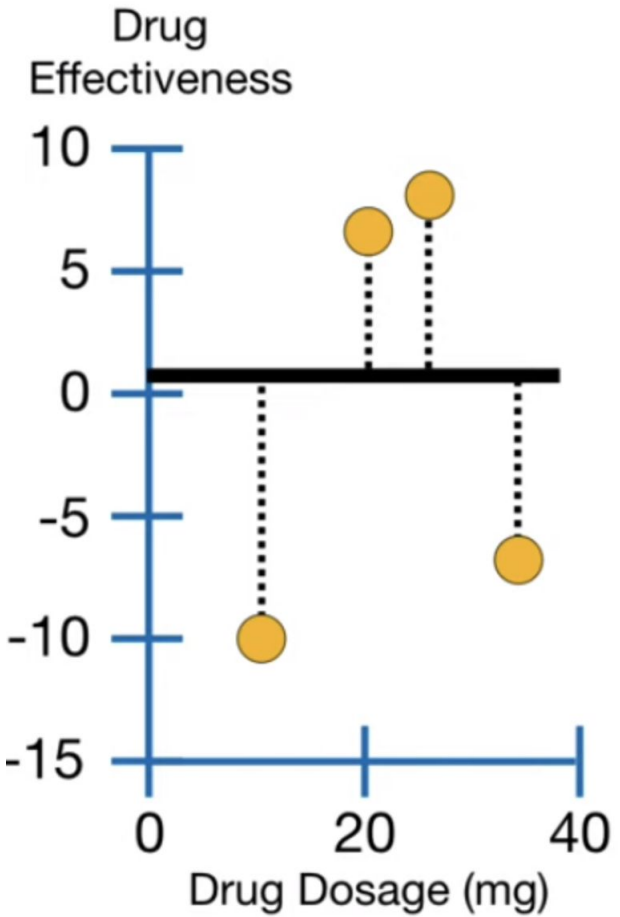
X : Drug Dose
Y = Drug Effectiveness

Using this data, we are going to make drug dosage effectiveness model



Initial Prediction Value (Black line): 0.5

We think this is because computer predicts either 0 or 1 so they set it to middle point which is 0.5 as base score



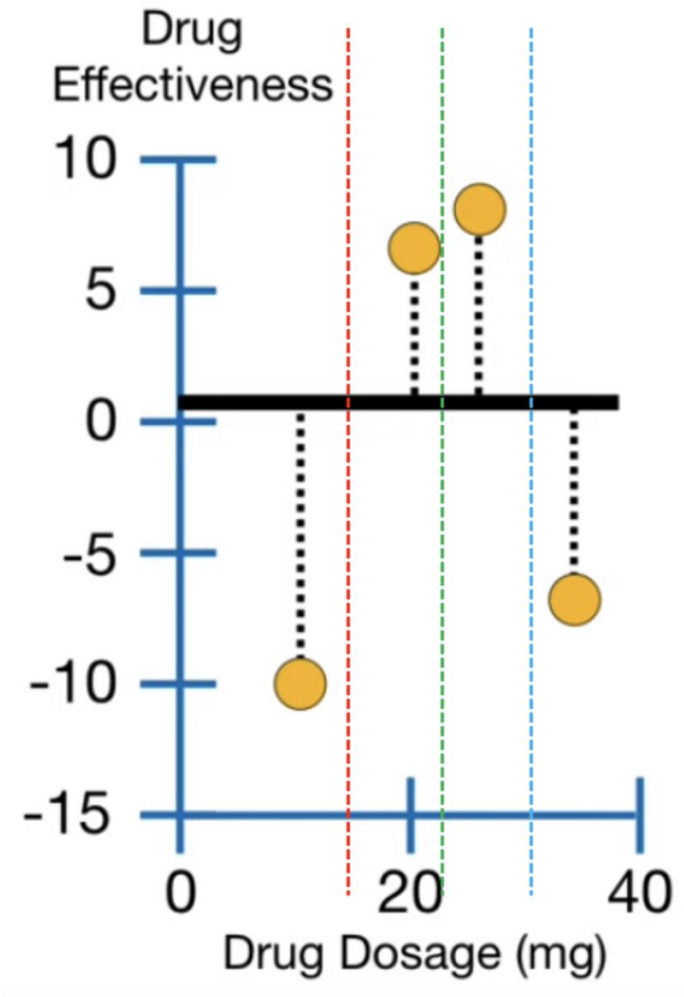
We want to maximize our **Gain**

- Gain = (New Similarity Score) - (Old Similarity Score)

Similarity Score = (Sum of Residuals)^2 / (Number of Residuals + lambda)

We can consider lambda as 0 here. Lambda is just a parameter to prevent overfitting.

So in this case, our old similarity score is
(-10.5 + 6.5 + 7.5 - 7.5)^2 / (4 + 0) = 4



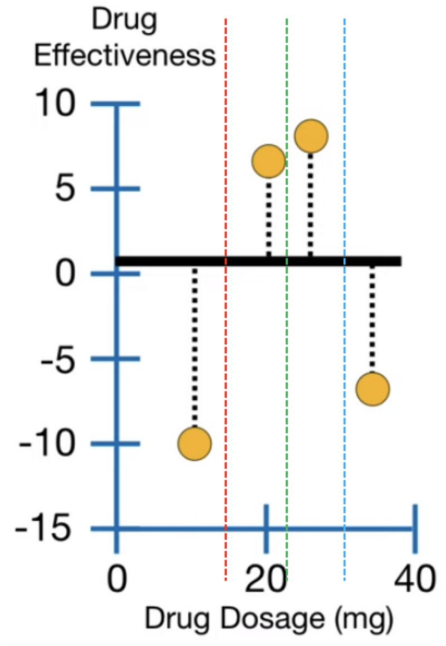
- Gain = (New Similarity Score) - (Old Similarity Score)

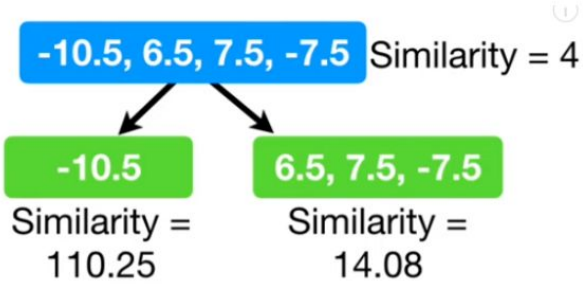
Dividing it by the red line

- (-10.5)^2 / 1 = 110.25 (New similarity Score)
- (6.5 + 7.5 - 7.5) ^ 2 / 3 = 14.08 (New similarity score)

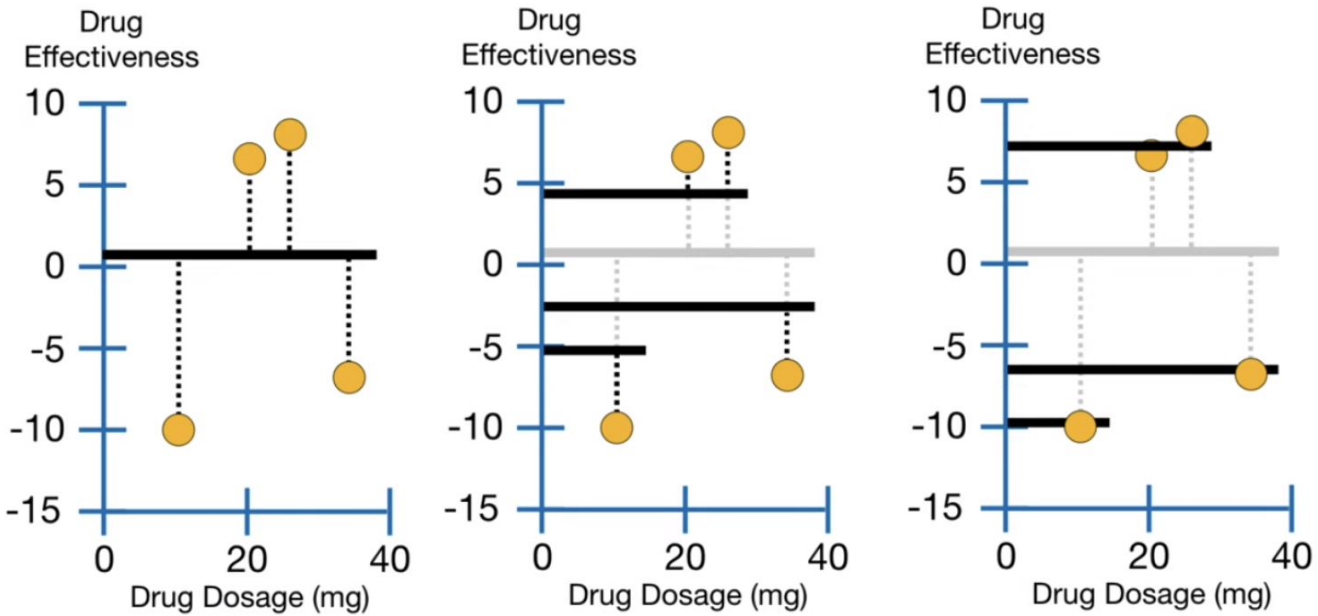
Using this information, we calculate the Gain

- (110.25 + 14.08) - 4 = 120.33





Decision Tree


Drug Effectiveness
10
5
0
-5
-10
-15
0
20
40
Drug Dosage (mg)
Drug Effectiveness
10
5
0
-5
-10
-15
0
20
40
Drug Dosage (mg)
Drug Effectiveness
10
5
0
-5
-10
-15
0
20
40
Drug Dosage (mg)

Used Car Price Prediction Using XGBoost Regressor

In [119…

```
"""
XGB Regressor - with Randomized Grid Search
"""

xgb = XGBRegressor(objective='reg:squarederror', eval_metric='rmse')
params = {'n_estimators':[700,800,900], 'max_depth':[5,6], 'learning_rate':[0.1]}
model = RandomizedSearchCV(xgb, params, n_iter=5, cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=0)
model.fit(X_train, y_train)

print(model.best_params_)
pred = model.predict(X_test)
print("MSE: ", mean_squared_error(pred, y_test))
print("MAE: ", mean_absolute_error(pred, y_test))
```

```
{'n_estimators': 800, 'max_depth': 5, 'learning_rate': 0.1}
MSE:  39510885.08189203
MAE:  3184.4337905248008
```

정확도 : 86.8579

In [139…

```
# With the best parameters searched
xgb = XGBRegressor(objective='reg:squarederror', eval_metric='rmse', n_estimators=1000
                   , max_depth=6, learning_rate=0.1)
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
print("XGB MSE: ", mean_squared_error(xgb_pred, y_test))
print("XGB MAE: ", mean_absolute_error(xgb_pred, y_test))
```

```
XGB MSE:  39493181.22963319
XGB MAE:  3191.6426135698953
```

1. Linear Regression

```
#modeling Linear Regression

model = LinearRegression(fit_intercept=True, n_jobs=4)
reg = model.fit(X_train, y_train)
pred = model.predict(X_test)

#R^2 score
print("Linear Regression R^2 Score for test set: ", reg.score(X_test, y_test))
print("Linear Regression MSE: ", mean_squared_error(pred, y_test))
print("Linear Regression MAE: ", mean_absolute_error(pred, y_test))
```

```
Linear Regression R^2 Score for test set:  0.7936733576176747
Linear Regression MSE:  72671238.66087796
Linear Regression MAE:  4553.894298360479
```

2. Polynomial Regression Degree 2

```
# Polynomial Regression - degree 2
poly = PolynomialFeatures(degree= 2)
X_train_ = poly.fit_transform(X_train)
X_test_ = poly.fit_transform(X_test)

model = LinearRegression(fit_intercept=True, n_jobs=4)
reg = model.fit(X_train_, y_train)
pred = model.predict(X_test_)

print("Polynomial Regression R^2 Score for test set: ", reg.score(X_test_, y_test))
print("Polynomial Regression MSE: ", mean_squared_error(pred, y_test))
print("Polynomial Regression MAE: ", mean_absolute_error(pred, y_test))
```

```
Polynomial Regression R^2 Score for test set:  0.8232675511760047
Polynomial Regression MSE:  62247734.07503561
Polynomial Regression MAE:  4477.578155412296
```

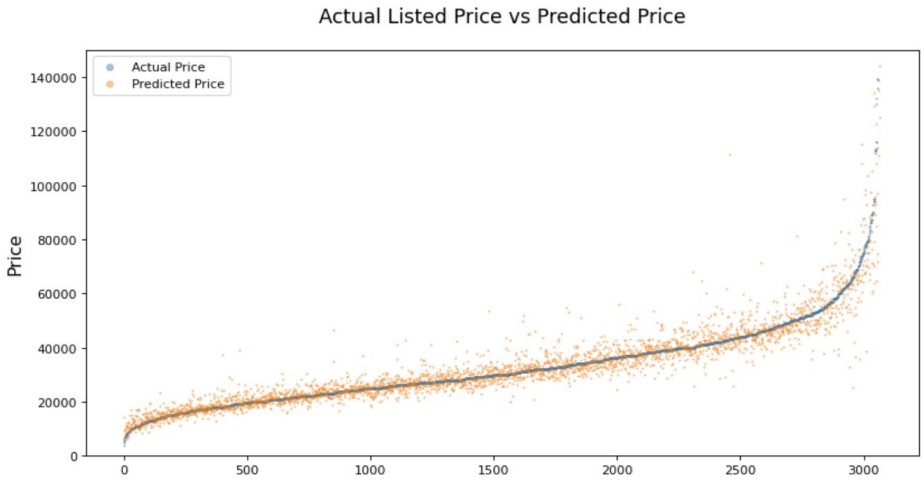
Polynomial Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

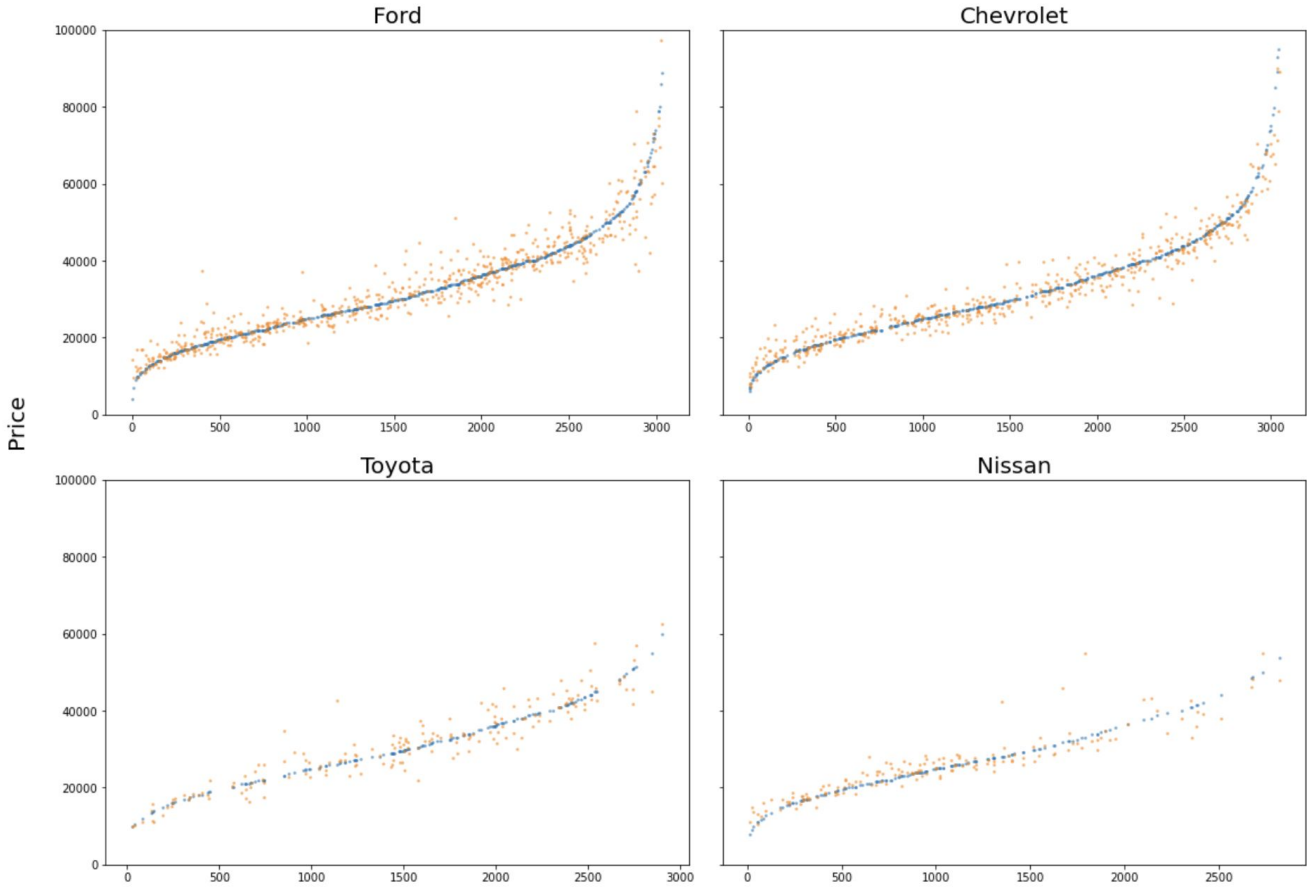
3. Random Forest

```
# With the best parameter
rf = RandomForestRegressor(n_estimators= 150, min_samples_split=2, max_depth=None, random_state=42)
rf.fit(X_train, y_train)
pred = rf.predict(X_test)
print("Test MSE: ", mean_squared_error(pred, y_test))
print("Test MAE: ", mean_absolute_error(pred, y_test))

pred = rf.predict(X_train)
print("Train MSE: ", mean_squared_error(pred, y_train))
print("Train MAE: ", mean_absolute_error(pred, y_train))

# Show feature importances
feature_dict = dict(zip(rf.feature_names_in_, rf.feature_importances_.round(4)))
display(sorted(feature_dict.items(), key=lambda x:x[1], reverse=True))
```

정확도 : 82.9708

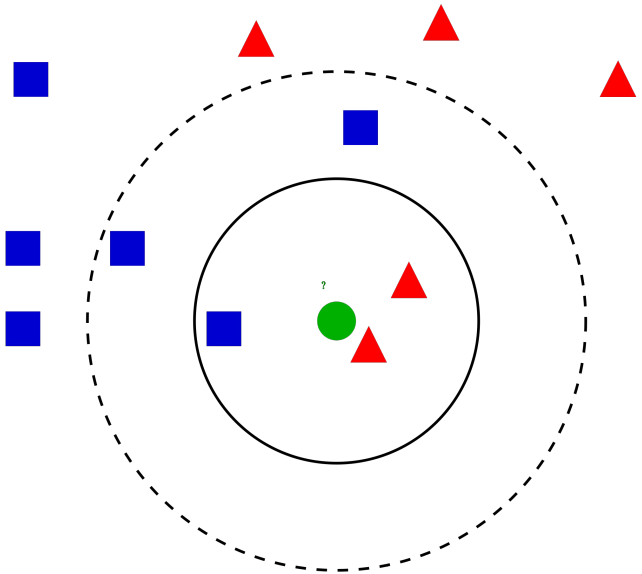



```
-------------------- Mean Absolute Percentage Error on Prediction by Brands --------------------

Ford : 9.27%
Chevrolet : 8.55%
Toyota : 8.40%
Nissan : 8.70%


-------------------- Mean Absolute Percentage Error on Prediction by Body Types --------------------

Pickup Truck : 9.02%
Sedan : 10.38%
SUV / Crossover : 8.25%
Wagon : 22.56%
Hatchback : 13.46%
Van : 14.26%
Minivan : 6.82%
Coupe : 16.39%
Convertible : 15.44%
```


Major Brands Actual Price vs Predicted Price
Ford
Chevrolet
Toyota
Nissan
Price
0
20000
40000
60000
80000
100000
0
500
1000
1500
2000
2500
3000

Improving the model

1. Enlarged the data set!
    a. From 1,000 zip codes from IL
    -> 3,000 zip codes from 5 states (IL, IA, WI, MO, IN)
2. Tried KNN with different scaling methods
3. Dropped Android Auto
    a. Due to multicollinearity determined by VIF (variance inflation factor)
    b. High market share of Apple CarPlay
        i. Either merge the two or drop Android Auto



	variables	VIF
0	CarPlay	12.465760
1	Android Auto	12.215435
2	YearOld	6.697340
3	mileage	6.702101

	variables	VIF
0	CarPlay	1.184677
1	YearOld	6.690461
2	mileage	6.698865

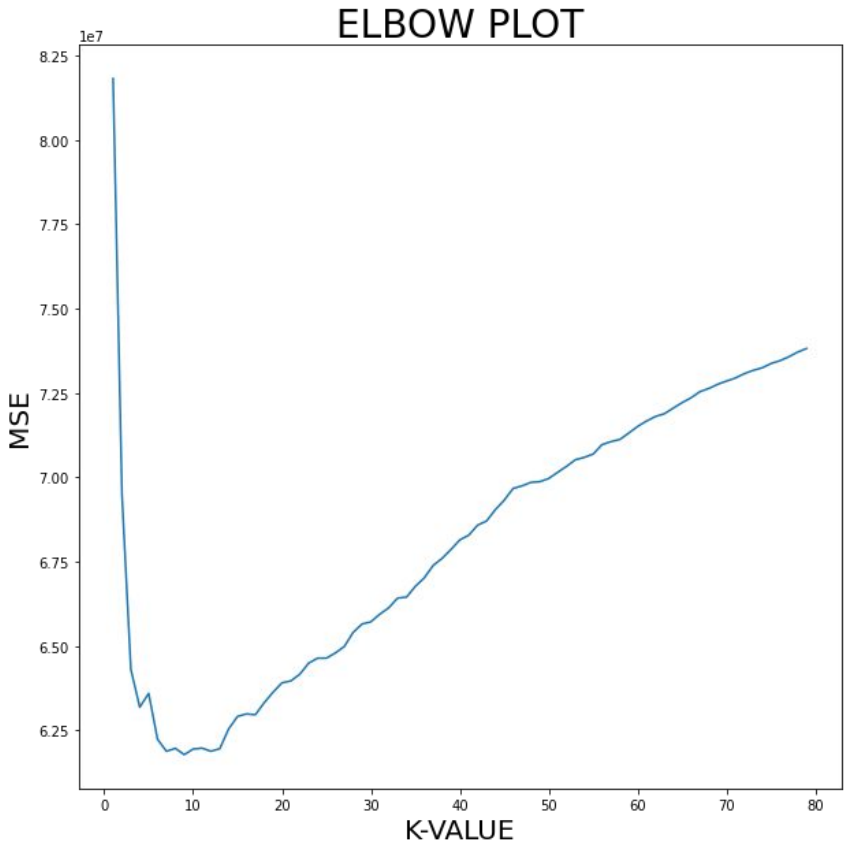
K Nearest Neighbors

MinMax Scaling : choose minimum and maximum values of a feature to rescale values

Standard Scaling : scaling features to be approximately standard normally distributed.

Robust Scaling : overcome outliers within the data. This is acquired by using the first and third quartile values in the scaling process.

Best K selected : 8



K Nearest Neighbors - MinMax Scaling

With our best K selected (8), MinMax scale range of (0,1)

```
KNN Score for test set after scailing:  0.6356390551835813
KNN MSE:  100904088.59185754
KNN MAE:  5711.1753761418595
```

K Nearest Neighbors - Standard Scaling

With our best K selected (8)

```
KNN Score for test set after scailing:  0.7614678844856192
KNN MSE:  66057754.15362708
KNN MAE:  4676.748361096184
```

K Nearest Neighbors - Robust Scaling

With our best K selected (8)

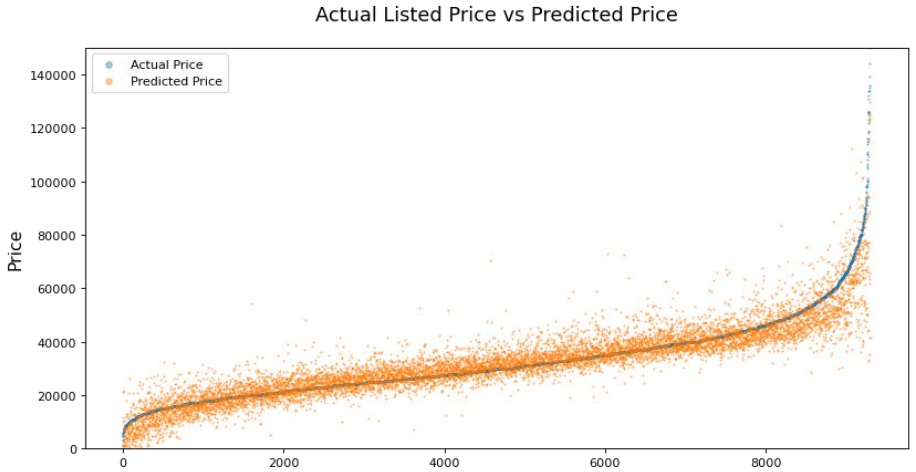
```
KNN Score for test set after scailing:  0.8138833082892858
KNN MSE:  51542118.92348536
KNN MAE:  3843.2880171950565
```

Results are lower when using Standard and MinMax scaling. We think this is because of weight problem.

Model Performance

Linear Regression

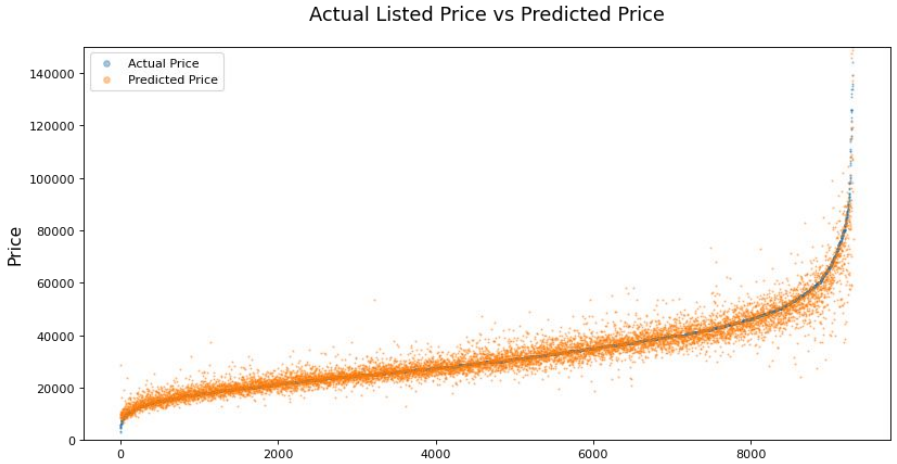
	Before	After
R^2	0.7937	0.7851
MSE	72,671,238	59,288,734
MAE	4554	4154.19



Model Performance

KNN Regression

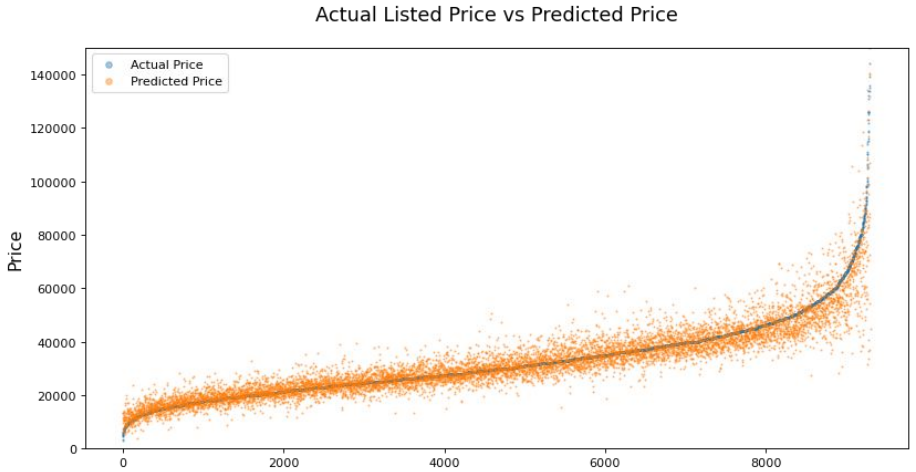
	Before Scaling	After Scaling
R^2	0.7762	0.8138
MSE	61,969,064	51,542,118
MAE	4643.04	3843.28



Model Performance

Polynomial Regression

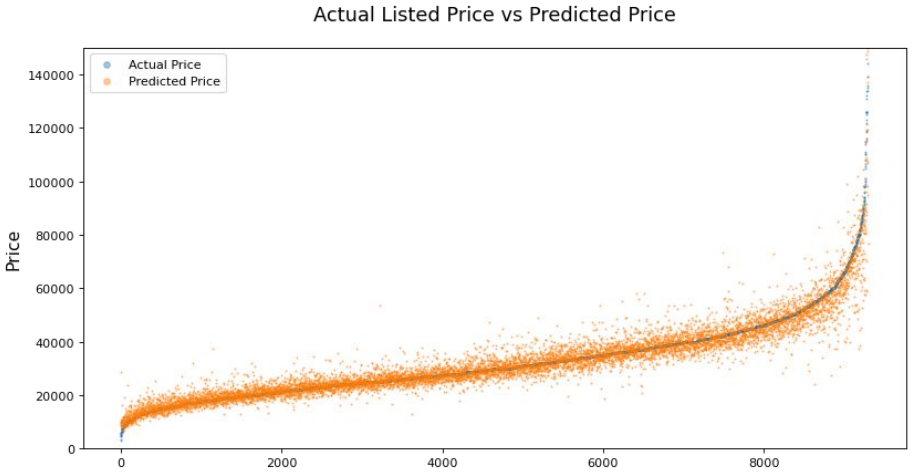
	Before	After
R^2	0.8232	0.8331
MSE	62,247,734	46,202,202
MAE	4477.58	3443.15



Model Performance

Random Forest

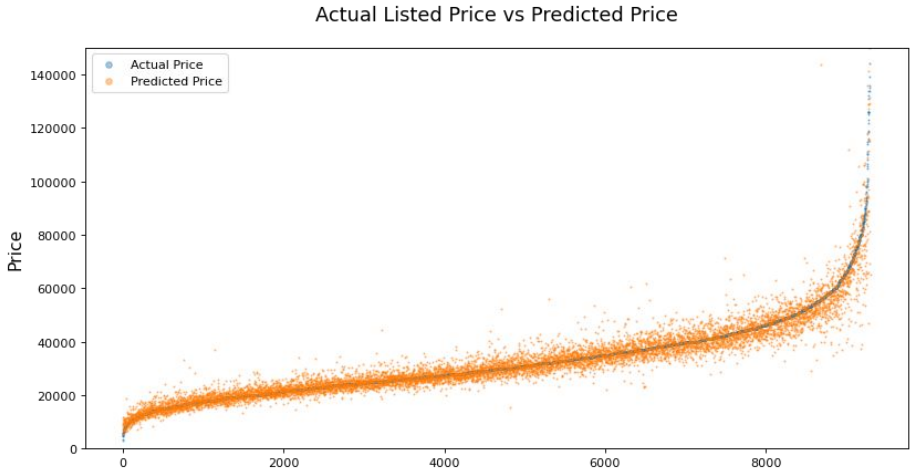
	Before	After
R^2	0.83	0.8924
MSE	39,510,885	29,780,137
MAE	3184	2724.70



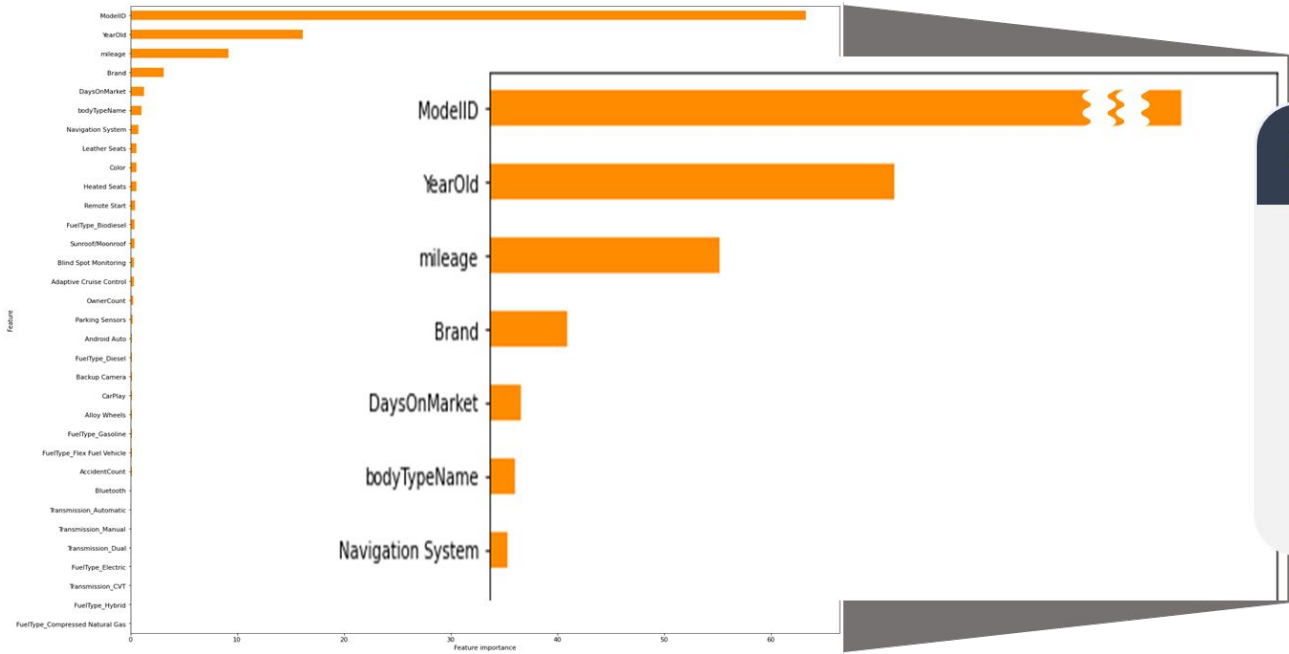
Model Performance

XGBoost Regression

	Before	After
R^2	0.8686	0.8930
MSE	39,493,181	29,623,373
MAE	3191.64	2584.23



Feature Importance



Key Feature Importance

Model ID	63.27
Year Old	16.14
Mileage	9.18
Brand	3.13