CSCI - 6515 - Machine Learning for Big Data - Fall 2022

Project : Predicting Used Car Prices using Machine Learning and Deep Neural Networks

Guryash Singh Dhall
(B00910690)

Vinay Vilas Patil
(B00911203)

Chanpreet Singh
(B00896766)

Smriti Mishra
(B00904799)

Importing and Downloading Data

In [1]:
```
# Installing Kaggle Dependencies to Pull Data Directly From Kaggle
! pip install kaggle
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

In [2]:
```
! kaggle datasets download austinreese/craigslist-carstrucks-data
! unzip craigslist-carstrucks-data.zip
```

Data understanding and feature engineering

In [119]:
```
# Importing Required Libraries for the Project

import pandas as pd
# import warnings filter
from warnings import simplefilter
import warnings
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from sklearn.model_selection import train_test_split
import folium
from folium.plugins import HeatMap
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor
import math
from sklearn.metrics import mean_absolute_percentage_error, mean_squared_error, mean_absolute_error, r2_score
from keras.callbacks import ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
import seaborn as sb
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.model_selection import RandomizedSearchCV
```

In [4]:
```
sb.set(rc={'figure.figsize':(10,6)})
```

In [5]:
```
main_df = pd.read_csv('../vehicles.csv')
```

In [6]:
```
data_df = main_df.copy()
```

In [7]:
```
data_df.head()
```

Out[7]:

	id	url	region	region_url	price	year	manufacturer	model	condition	cylinders	...	siz
0	7222695916	https://prescott.craigslist.org/cto/d/prescott...	prescott	https://prescott.craigslist.org	6000	NaN	NaN	NaN	NaN	NaN	...	Na
1	7218891961	https://fayar.craigslist.org/ctd/d/bentonville...	fayetteville	https://fayar.craigslist.org	11900	NaN	NaN	NaN	NaN	NaN	...	Na
2	7221797935	https://keys.craigslist.org/cto/d/summerland-k...	florida keys	https://keys.craigslist.org	21000	NaN	NaN	NaN	NaN	NaN	...	Na
3	7222270760	https://worcester.craigslist.org/cto/d/west-br...	worcester / central MA	https://worcester.craigslist.org	1500	NaN	NaN	NaN	NaN	NaN	...	Na
4	7210384030	https://greensboro.craigslist.org/cto/d/trinit...	greensboro	https://greensboro.craigslist.org	4900	NaN	NaN	NaN	NaN	NaN	...	Na

5 rows × 26 columns

In [8]:
```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            426880 non-null  int64  
 1   url           426880 non-null  object 
 2   region        426880 non-null  object 
 3   region_url    426880 non-null  object 
 4   price         426880 non-null  int64  
 5   year          425675 non-null  float64
 6   manufacturer  409234 non-null  object 
 7   model         421603 non-null  object 
 8   condition     252776 non-null  object 
 9   cylinders     249202 non-null  object 
 10  fuel          423867 non-null  object 
 11  odometer      422480 non-null  float64
 12  title_status  418638 non-null  object 
 13  transmission  424324 non-null  object 
 14  VIN           265838 non-null  object 
 15  drive         296313 non-null  object 
 16  size          120519 non-null  object 
 17  type          334022 non-null  object 
 18  paint_color   296677 non-null  object 
 19  image_url     426812 non-null  object 
 20  description   426810 non-null  object 
 21  county        0 non-null       float64
 22  state         426880 non-null  object 
 23  lat           420331 non-null  float64
 24  long          420331 non-null  float64
 25  posting_date  426812 non-null  object 
dtypes: float64(5), int64(2), object(19)
memory usage: 84.7+ MB
```

In [9]:
```
data_df.describe()
```

Out[9]:

	id	price	year	odometer	county	lat	long
count	4.268800e+05	4.268800e+05	425675.000000	4.224800e+05	0.0	420331.000000	420331.000000
mean	7.311487e+09	7.519903e+04	2011.235191	9.804333e+04	NaN	38.493940	-94.748599
std	4.473170e+06	1.218228e+07	9.452120	2.138815e+05	NaN	5.841533	18.365462
min	7.207408e+09	0.000000e+00	1900.000000	0.000000e+00	NaN	-84.122245	-159.827728
25%	7.308143e+09	5.900000e+03	2008.000000	3.770400e+04	NaN	34.601900	-111.939847
50%	7.312621e+09	1.395000e+04	2013.000000	8.554800e+04	NaN	39.150100	-88.432600
75%	7.315254e+09	2.648575e+04	2017.000000	1.335425e+05	NaN	42.398900	-80.832039
max	7.317101e+09	3.736929e+09	2022.000000	1.000000e+07	NaN	82.390818	173.885502

In [10]:
```
data_df.iloc[0]
```

Out[10]:
```
id                                                                  7222695916
url            https://prescott.craigslist.org/cto/d/prescott... (https://prescott.craigslist.org/cto/d/prescott...)
region                                                                prescott
region_url                                https://prescott.craigslist.org (https://prescott.craigslist.org)
price                                                                     6000
year                                                                       NaN
manufacturer                                                               NaN
model                                                                      NaN
condition                                                                  NaN
cylinders                                                                  NaN
fuel                                                                       NaN
odometer                                                                   NaN
title_status                                                               NaN
transmission                                                               NaN
VIN                                                                        NaN
drive                                                                      NaN
size                                                                       NaN
type                                                                       NaN
paint_color                                                                NaN
image_url                                                                  NaN
description                                                                NaN
county                                                                     NaN
state                                                                       az
lat                                                                        NaN
long                                                                       NaN
posting_date                                                               NaN
Name: 0, dtype: object
```

In [11]:
```
data_df.dtypes
```

Out[11]:
```
id                int64
url              object
region           object
region_url       object
price             int64
year            float64
manufacturer     object
model            object
condition        object
cylinders        object
fuel             object
odometer        float64
title_status     object
transmission     object
VIN              object
drive            object
size             object
type             object
paint_color      object
image_url        object
description      object
county          float64
state            object
lat             float64
long            float64
posting_date     object
dtype: object
```

Continuous Feature Report

Continuous features report includes:

1. Min
2. 1st quartile
3. Mean
4. 2nd quartile - Median
5. 3rd quartile
6. Max
7. Standard deviation
8. Total num of instances
9. % missing values
10. Cardinality - num of distinct values for a feature

Using Pandas provides a function for generating data quality reports however it doesn't include all the statistics.a

In [12]:
```
data_df.describe(include=['number'])
```

Out[12]:

	id	price	year	odometer	county	lat	long
count	4.268800e+05	4.268800e+05	425675.000000	4.224800e+05	0.0	420331.000000	420331.000000
mean	7.311487e+09	7.519903e+04	2011.235191	9.804333e+04	NaN	38.493940	-94.748599
std	4.473170e+06	1.218228e+07	9.452120	2.138815e+05	NaN	5.841533	18.365462
min	7.207408e+09	0.000000e+00	1900.000000	0.000000e+00	NaN	-84.122245	-159.827728
25%	7.308143e+09	5.900000e+03	2008.000000	3.770400e+04	NaN	34.601900	-111.939847
50%	7.312621e+09	1.395000e+04	2013.000000	8.554800e+04	NaN	39.150100	-88.432600
75%	7.315254e+09	2.648575e+04	2017.000000	1.335425e+05	NaN	42.398900	-80.832039
max	7.317101e+09	3.736929e+09	2022.000000	1.000000e+07	NaN	82.390818	173.885502

In [13]:
```
import warnings
def build_continuous_features_report(telecom):

    """Build tabular report for continuous features"""

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Min": lambda df: df.min(),
        "1st Qrt.": lambda df: df.quantile(0.25),
        "Mean": lambda df: df.mean(),
        "Median": lambda df: df.median(),
        "3rd Qrt": lambda df: df.quantile(0.75),
        "Max": lambda df: df.max(),
        "Std. Dev.": lambda df: df.std(),
    }

    contin_feat_names = telecom.select_dtypes("number").columns
    continuous_data_df = telecom[contin_feat_names]

    report_df = pd.DataFrame(index=contin_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [14]:
```
build_continuous_features_report(data_df)
```

Out[14]:

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
id	426880	0.000000	426880	7.207408e+09	7.308143e+09	7.311487e+09	7.312621e+09	7.315254e+09	7.317101e+09	4.473170e+06
price	426880	0.000000	15655	0.000000e+00	5.900000e+03	7.519903e+04	1.395000e+04	2.648575e+04	3.736929e+09	1.218228e+07
year	426880	0.282281	114	1.900000e+03	2.008000e+03	2.011235e+03	2.013000e+03	2.017000e+03	2.022000e+03	9.452120e+00
odometer	426880	1.030735	104870	0.000000e+00	3.770400e+04	9.804333e+04	8.554800e+04	1.335425e+05	1.000000e+07	2.138815e+05
county	426880	100.000000	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
lat	426880	1.534155	53181	-8.412225e+01	3.460190e+01	3.849394e+01	3.915010e+01	4.239890e+01	8.239082e+01	5.841533e+00
long	426880	1.534155	53772	-1.598277e+02	-1.119398e+02	-9.474860e+01	-8.843260e+01	-8.083204e+01	1.738855e+02	1.836546e+01

Categorical Feature Report

Categorical features report includes:

1. Mode - the most frequent value
2. 2nd mode - the second most frequent value
3. Frequency of mode
4. Proportion of mode in the dataset
5. Frequency of 2nd mode
6. Proportion of 2nd mode in the dataset
7. % missing values
8. Cardinality

Pandas provides a function for generating data quality reports however it doesn't include all the statistics.

In [15]:
```python
data_df.describe(exclude=['number'])
```

Out[15]:

	url	region	region_url	manufacturer	model	condition	cylinders	fuel	title_status	transmissi
count	426880	426880	426880	409234	421603	252776	249202	423867	418638	4243
unique	426880	404	413	42	29667	6	8	5	6	
top	https://prescott.craigslist.org/cto/d/prescott...	columbus	https://spokane.craigslist.org	ford	f-150	good	6 cylinders	gas	clean	automa
freq	1	3608	2988	70985	8009	121456	94169	356209	405117	3365

In [16]:
```python
def build_categorical_features_report(telecom):

    """Build tabular report for categorical features"""

    def _mode(df):
        return df.apply(lambda ft: ft.mode().to_list()).T

    def _mode_freq(df):
        return df.apply(lambda ft: ft.value_counts()[ft.mode()].sum())

    def _second_mode(df):
        return df.apply(lambda ft: ft[~ft.isin(ft.mode())].mode().to_list())

    def _second_mode_freq(df):
        return df.apply(
            lambda ft: ft[~ft.isin(ft.mode())]
            .value_counts()[ft[~ft.isin(ft.mode())].mode()]
            .sum()
        )

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Mode": _mode,
        "Mode Freq": _mode_freq,
        "Mode %": lambda df: _mode_freq(df) / len(df) * 100,
        "2nd Mode": _second_mode,
        "2nd Mode Freq": _second_mode_freq,
        "2nd Mode %": lambda df: _second_mode_freq(df) / len(df) * 100,
    }

    cat_feat_names = telecom.select_dtypes(exclude="number").columns
    continuous_data_df = telecom[cat_feat_names]

    report_df = pd.DataFrame(index=cat_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [17]:
```
build_categorical_features_report(data_df)
```

Out[17]:

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2 Mo Fr
url	426880	0.000000	426880	[https://abilene.craigslist.org/ctd/d/abilene-...	426880	100.000000	[]	
region	426880	0.000000	404	[columbus]	3608	0.845202	[jacksonville]	35
region_url	426880	0.000000	413	[https://spokane.craigslist.org]	2988	0.699963	[https://eugene.craigslist.org]	29
manufacturer	426880	4.133714	42	[ford]	70985	16.628795	[chevrolet]	550
model	426880	1.236179	29667	[f-150]	8009	1.876171	[silverado 1500]	51
condition	426880	40.785232	6	[good]	121456	28.452024	[excellent]	1014
cylinders	426880	41.622470	8	[6 cylinders]	94169	22.059829	[4 cylinders]	776
fuel	426880	0.705819	5	[gas]	356209	83.444762	[other]	307
title_status	426880	1.930753	6	[clean]	405117	94.901846	[rebuilt]	72
transmission	426880	0.598763	3	[automatic]	336524	78.833396	[other]	626
VIN	426880	37.725356	118264	[1FMJU1JT1HEA52352]	261	0.061141	[3C6JR6DT3KG560649]	2
drive	426880	30.586347	3	[4wd]	131904	30.899550	[fwd]	1055
size	426880	71.767476	4	[full-size]	63465	14.867176	[mid-size]	344
type	426880	21.752717	13	[sedan]	87056	20.393553	[SUV]	772
paint_color	426880	30.501078	12	[white]	79285	18.573135	[black]	628
image_url	426880	0.015930	241899	[https://images.craigslist.org/00N0N_1xMPvfxRA...	7357	1.723435	[https://images.craigslist.org/00R0R_lwWjXSEWN...	21
description	426880	0.016398	360911	[35 VEHICLES PRICED UNDER $3000!!! BIG TIME! ...	231	0.054114	[Call or text today to find out more. (602) 62...	1
state	426880	0.000000	51	[ca]	50614	11.856728	[fl]	285
posting_date	426880	0.015930	381536	[2021-04-23T22:13:05-0400]	12	0.002811	[2021-04-13T13:19:15-0500, 2021-04-22T20:32:05...	

Checking NULL Values

In [18]:
```
data_df.isnull().sum()
```

Out[18]:
```
id                   0
url                  0
region               0
region_url           0
price                0
year              1205
manufacturer     17646
model             5277
condition       174104
cylinders       177678
fuel              3013
odometer          4400
title_status      8242
transmission      2556
VIN             161042
drive           130567
size            306361
type             92858
paint_color     130203
image_url           68
description         70
county          426880
state                0
lat               6549
long              6549
posting_date        68
dtype: int64
```

20 features have null values, but many features have more than 40% null values, so we will not be using them as they can distort the analysis and the model

Remove columns with more than 40% missing values

In [19]:
```
columns_null = data_df.isna().sum()

def filter_null(na, threshold = .4):
    #only selecting the features that are having less than 40% null values
    col_pass = []
    for i in na.keys():
        if na[i]/data_df.shape[0]<threshold:
            col_pass.append(i)
    return col_pass

cleaned_data_df = data_df[filter_null(columns_null)]
cleaned_data_df.columns
```

Out[19]:
```
Index(['id', 'url', 'region', 'region_url', 'price', 'year', 'manufacturer',
       'model', 'fuel', 'odometer', 'title_status', 'transmission', 'VIN',
       'drive', 'type', 'paint_color', 'image_url', 'description', 'state',
       'lat', 'long', 'posting_date'],
      dtype='object')
```

In [20]:
```
cleaned_data_df.isnull().sum()
```

Out[20]:
```
id                   0
url                  0
region               0
region_url           0
price                0
year              1205
manufacturer     17646
model             5277
fuel              3013
odometer          4400
title_status      8242
transmission      2556
VIN             161042
drive           130567
type             92858
paint_color     130203
image_url           68
description         70
state                0
lat               6549
long              6549
posting_date        68
dtype: int64
```

Checking Duplicate Records in Database

In [21]:
```
duplicate_records = cleaned_data_df[cleaned_data_df.duplicated()]

duplicate_records
```

Out[21]:

id	url	region	region_url	price	year	manufacturer	model	fuel	odometer	...	VIN	drive	type	paint_color	image_url	description	state	lat	long	postir

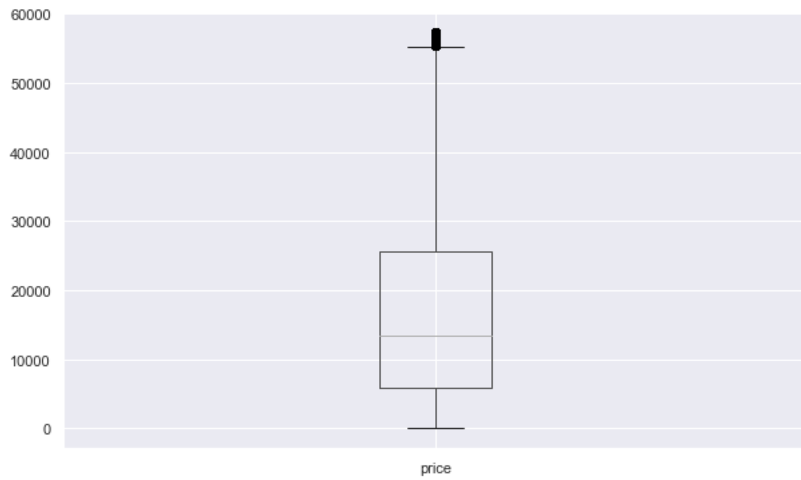
0 rows × 22 columns

Checking the outlier in target feature "Price"

In [22]:
```
# Computing IQR
Q1 = cleaned_data_df['price'].quantile(0.25)
Q3 = cleaned_data_df['price'].quantile(0.75)
IQR = Q3 - Q1

# Filtering Values between Q1-1.5IQR and Q3+1.5IQR
df_filtered = cleaned_data_df.query('(@Q1 - 1.5 * @IQR) <= price <= (@Q3 + 1.5 * @IQR)')
df_filtered.boxplot('price')
```

Out[22]: <AxesSubplot:>

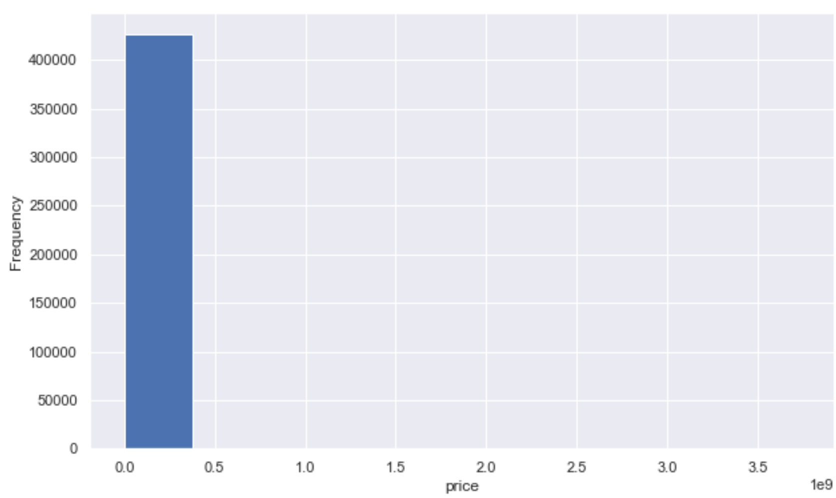


Data Visualization

In [23]:
```
cars_data = main_df.copy()
```

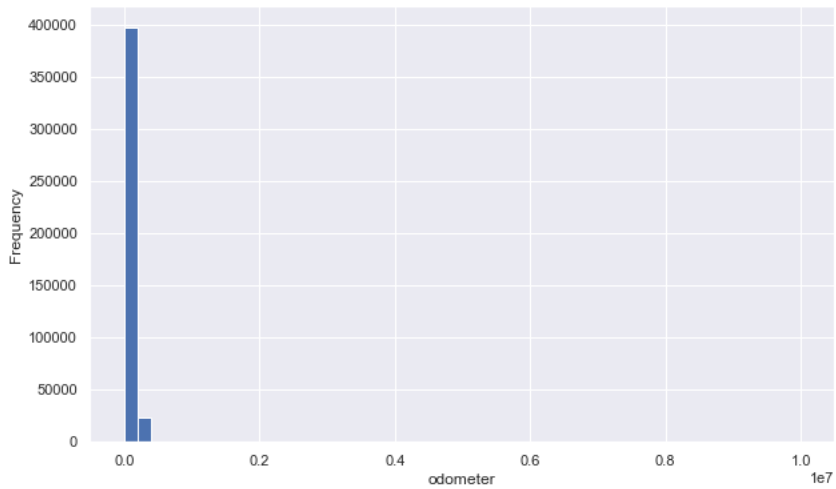
Price Column

In [24]:
```
axes = cars_data['price'].plot.hist(bins=10)
_ = axes.set_xlabel("price")
```



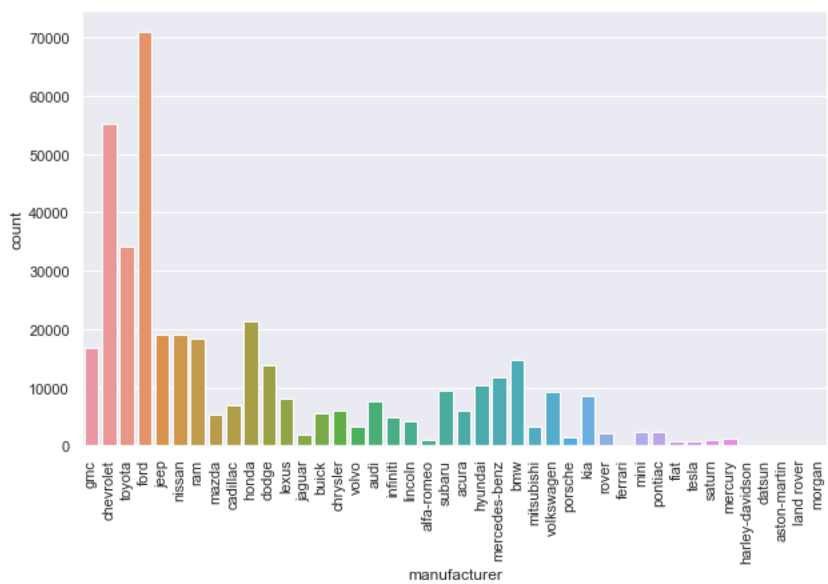
Odometer Column

In [25]:
```
axes = cars_data['odometer'].plot.hist(bins=50)
_ = axes.set_xlabel("odometer")
```



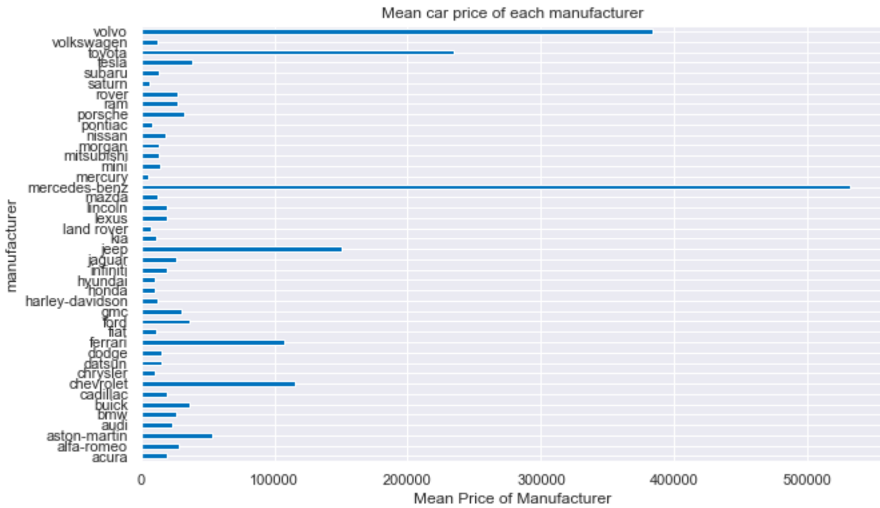
Manufacturer

In [27]:
```
plt.figure()
sb.countplot(data = cars_data, x="manufacturer")
plt.xticks(rotation = 90)
plt.show()
```



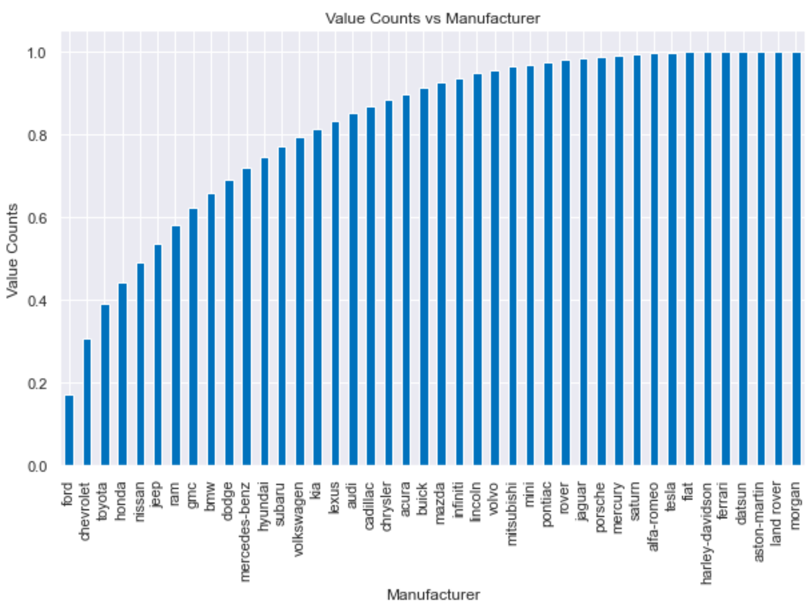
In [28]:
```
fig, ax = plt.subplots()
ax.set_title('Mean car price of each manufacturer')
cars_data.groupby(['manufacturer']).mean()['price'].plot.barh(ax=ax, color='#0072BD')
plt.xlabel('Mean Price of Manufacturer')
```

Out[28]: Text(0.5, 0, 'Mean Price of Manufacturer')



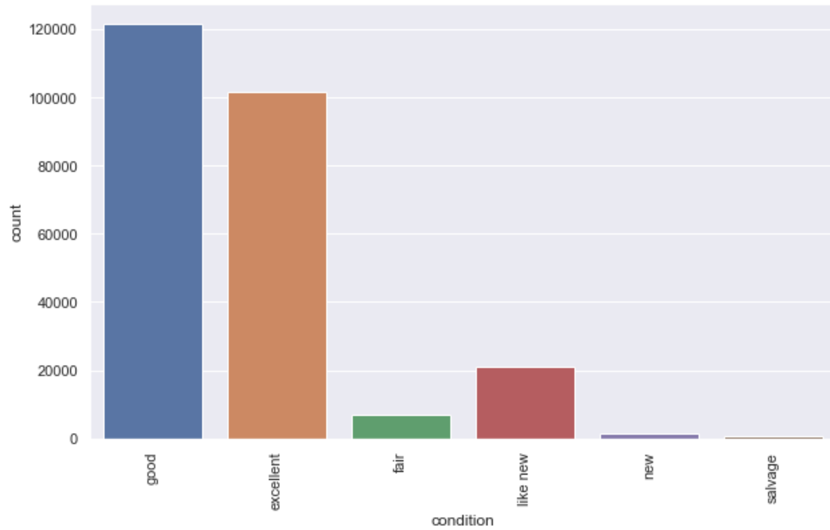
In [29]:
```
cars_data['manufacturer'].value_counts(normalize=True).cumsum().plot(kind='bar', color='#0072BD')
plt.xlabel('Manufacturer')
plt.ylabel('Value Counts')
plt.title('Value Counts vs Manufacturer')
```

Out[29]: Text(0.5, 1.0, 'Value Counts vs Manufacturer')



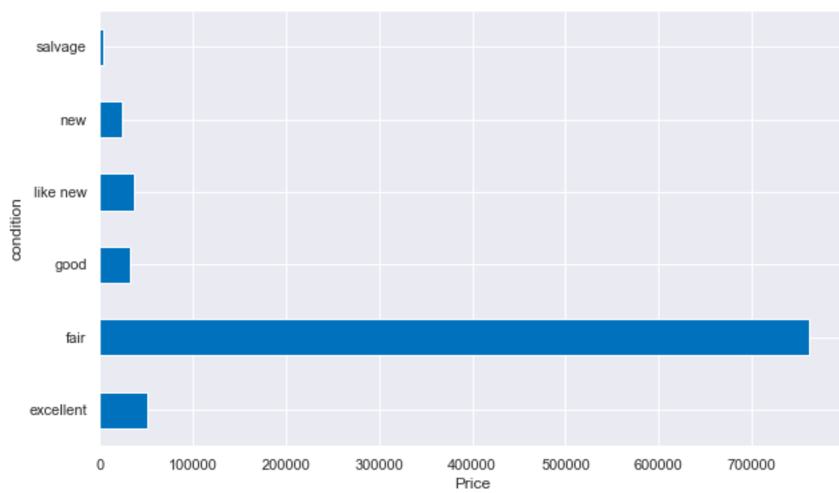
Condition Columns

In [30]:
```
plt.figure()
sb.countplot(data = cars_data, x="condition")
plt.xticks(rotation = 90)
plt.show()
```



In [31]:
```
fig, ax = plt.subplots()
cars_data.groupby(['condition']).mean()['price'].plot.barh(ax=ax, color='#0072BD')
plt.xlabel('Price')
```

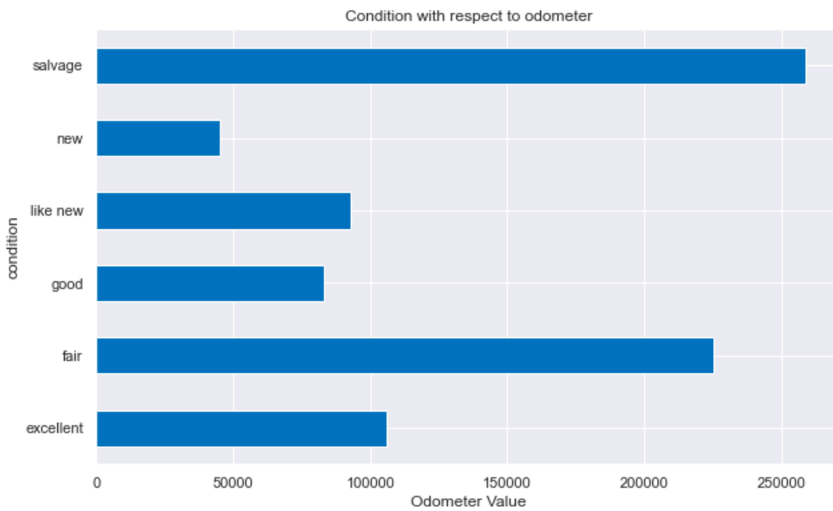
Out[31]: Text(0.5, 0, 'Price')



Condition of the Car with Respect to Odometer

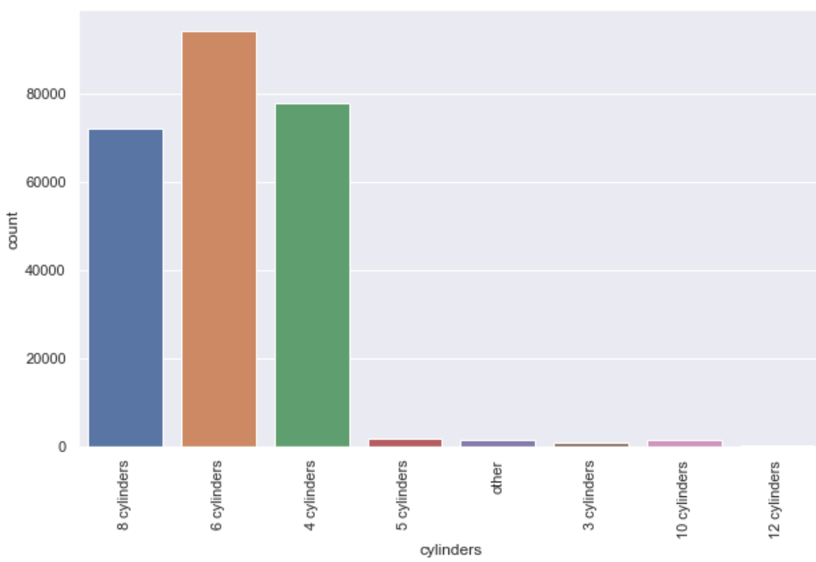
In [32]:
```
fig, ax = plt.subplots()
ax.set_title('Condition with respect to odometer')
cars_data.groupby(['condition']).mean()['odometer'].plot.barh(ax=ax, color='#0072BD')
plt.xlabel('Odometer Value')
```

Out[32]: `Text(0.5, 0, 'Odometer Value')`



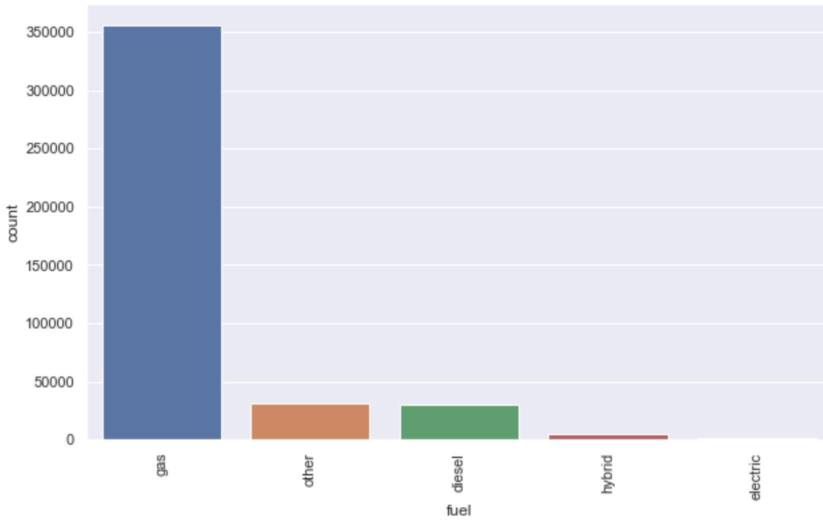
Cylinder Column

In [33]:
```
plt.figure()
sb.countplot(data = cars_data, x="cylinders")
plt.xticks(rotation = 90)
plt.show()
```



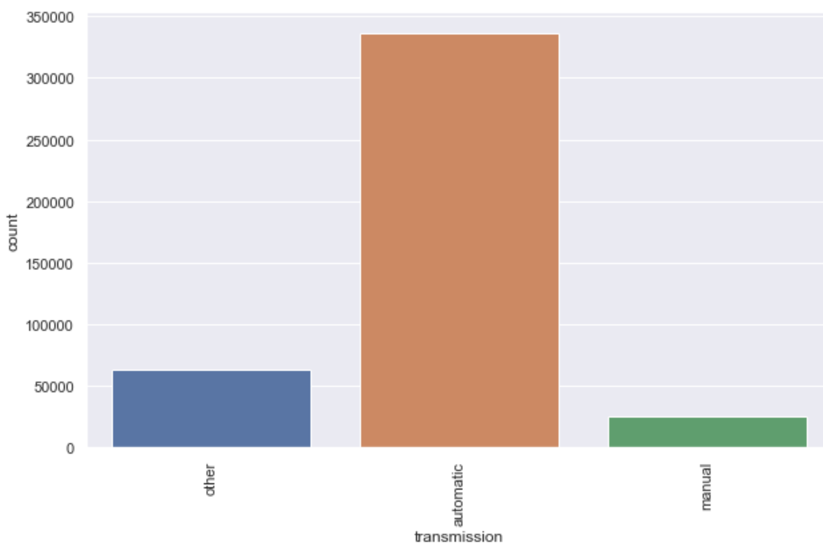
Fuel Type

In [34]:
```
plt.figure()
sb.countplot(data = cars_data, x="fuel")
plt.xticks(rotation = 90)
plt.show()
```



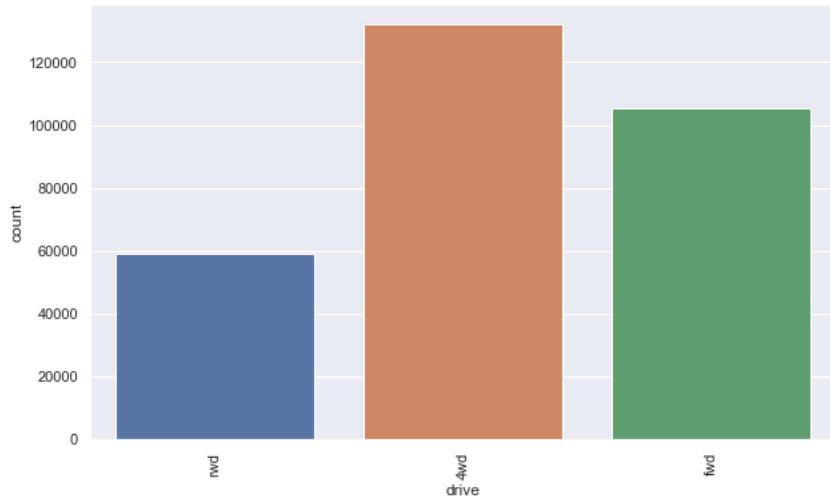
Transmission Column

In [35]:
```
plt.figure()
sb.countplot(data = cars_data, x="transmission")
plt.xticks(rotation = 90)
plt.show()
```



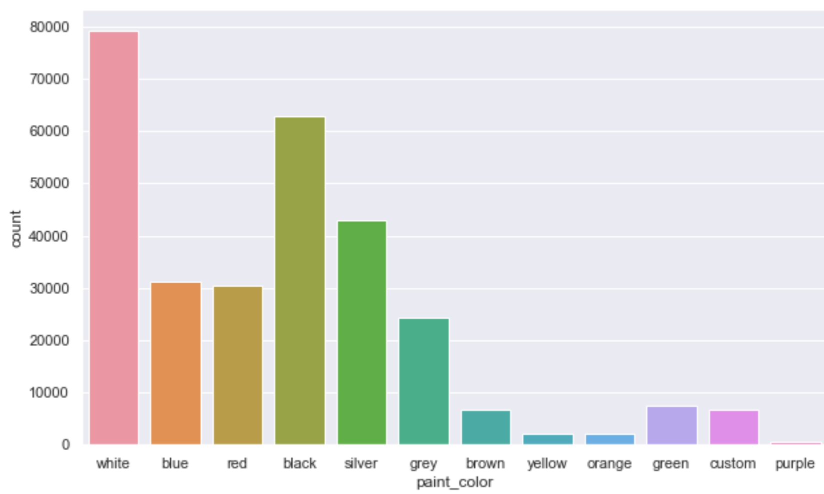
Drive

In [36]:
```
plt.figure()
sb.countplot(data = cars_data, x="drive")
plt.xticks(rotation = 90)
plt.show()
```



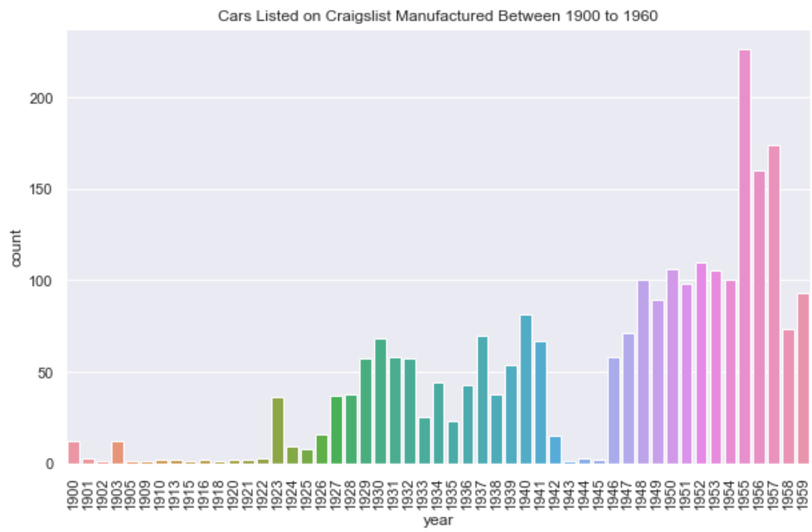
Paint Color

In [37]:
```
plt.figure()
sb.countplot(data = cars_data, x="paint_color")
plt.show()
```



Cars Listed on Craigslist Manufactured Between 1900 to 1960

In [38]:
```
new_df1 = cars_data[(cars_data['year'] >= 1900) & (cars_data['year'] <= 1959)].copy()
new_df1['year'] = new_df1['year'].astype(int)
sb.countplot(data = new_df1, x = 'year')
plt.xticks(rotation = 90)
plt.title("Cars Listed on Craigslist Manufactured Between 1900 to 1960")
plt.show()
```

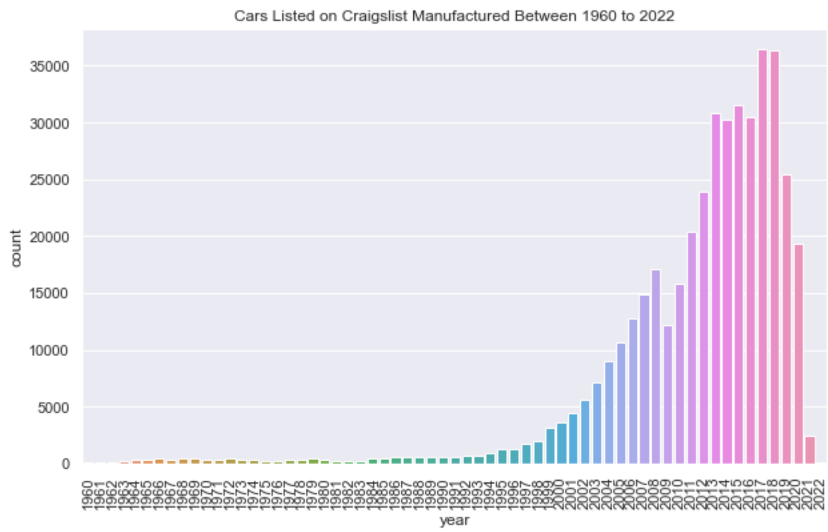


In [39]:
```
map = folium.Map(location=[42.5,-71], zoom_start=5)
cars=cars_data[cars_data["year"]<1960].iloc[:,23:25]
cars.lat.fillna(0, inplace = True)
cars.long.fillna(0, inplace = True)
HeatMap(data=cars, radius=16).add_to(map)
map
```

Out[39]: Make this Notebook Trusted to load map: File -> Trust Notebook

Cars Listed on Craigslist Manufactured Between 1960 to 2022

In [40]:
```
new_df = cars_data[(cars_data['year'] >= 1960) & (cars_data['year'] <= 2022)].copy()
new_df['year'] = new_df['year'].astype(int)
sb.countplot(data = new_df, x = 'year')
plt.title("Cars Listed on Craigslist Manufactured Between 1960 to 2022")
plt.xticks(rotation = 90)
plt.show()
```



In [41]:
```
map = folium.Map(location=[42.5,-71], zoom_start=5)
cars=cars_data[cars_data["year"]>1960].iloc[:,23:25]
cars.lat.fillna(0, inplace = True)
cars.long.fillna(0, inplace = True)
HeatMap(data=cars, radius=16).add_to(map)
map
```

Out[41]: Make this Notebook Trusted to load map: File -> Trust Notebook

Data Cleaning

In [63]:
```
df = main_df.copy()
```

In [64]:
```
null_values=df.isnull().sum()
null_values=pd.DataFrame(null_values,columns=['null'])
j=1
sum_tot=len(df)
null_values['percent']=null_values['null']/sum_tot
round(null_values,3).sort_values('percent',ascending=False)
```

Out[64]:

	null	percent
county	426880	1.000
size	306361	0.718
cylinders	177678	0.416
condition	174104	0.408
VIN	161042	0.377
drive	130567	0.306
paint_color	130203	0.305
type	92858	0.218
manufacturer	17646	0.041
title_status	8242	0.019
lat	6549	0.015
long	6549	0.015
model	5277	0.012
odometer	4400	0.010
fuel	3013	0.007
transmission	2556	0.006
year	1205	0.003
description	70	0.000
state	0	0.000
id	0	0.000
image_url	68	0.000
url	0	0.000
price	0	0.000
region_url	0	0.000
region	0	0.000
posting_date	68	0.000

In [65]:
```
df.columns
```

Out[65]:
```
Index(['id', 'url', 'region', 'region_url', 'price', 'year', 'manufacturer',
       'model', 'condition', 'cylinders', 'fuel', 'odometer', 'title_status',
       'transmission', 'VIN', 'drive', 'size', 'type', 'paint_color',
       'image_url', 'description', 'county', 'state', 'lat', 'long',
       'posting_date'],
      dtype='object')
```

In [66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            426880 non-null  int64  
 1   url           426880 non-null  object 
 2   region        426880 non-null  object 
 3   region_url    426880 non-null  object 
 4   price         426880 non-null  int64  
 5   year          425675 non-null  float64
 6   manufacturer  409234 non-null  object 
 7   model         421603 non-null  object 
 8   condition     252776 non-null  object 
 9   cylinders     249202 non-null  object 
 10  fuel          423867 non-null  object 
 11  odometer      422480 non-null  float64
 12  title_status  418638 non-null  object 
 13  transmission  424324 non-null  object 
 14  VIN           265838 non-null  object 
 15  drive         296313 non-null  object 
 16  size          120519 non-null  object 
 17  type          334022 non-null  object 
 18  paint_color   296677 non-null  object 
 19  image_url     426812 non-null  object 
 20  description   426810 non-null  object 
 21  county        0 non-null       float64
 22  state         426880 non-null  object 
 23  lat           420331 non-null  float64
 24  long          420331 non-null  float64
 25  posting_date  426812 non-null  object 
dtypes: float64(5), int64(2), object(19)
memory usage: 84.7+ MB
```

Based on above observation the features which are too common or of no use like url can be dropped

In [67]:
```
df= df.drop(columns=['id','url', 'region_url', 'VIN', 'image_url', 'description', 'lat', 'long','county','region', 'size', 'posti
```

In [68]:
```
df.isnull().sum()
```

Out[68]:
```
price                0
year              1205
manufacturer     17646
model             5277
condition       174104
cylinders       177678
fuel              3013
odometer          4400
title_status      8242
transmission      2556
drive           130567
type             92858
paint_color     130203
state                0
dtype: int64
```

Since Price is our target variable we have to handle outliers in the 'Price'

In [69]:
```
q1, q3 = np.percentile(sorted(df['price']),[10,90])
```

In [70]:
```
q1,q3
```

Out[70]:
```
(500.0, 37590.0)
```

In [71]:
```
df=df[(df.price < 37590) & (df.price >= 500 )]
df.shape
```

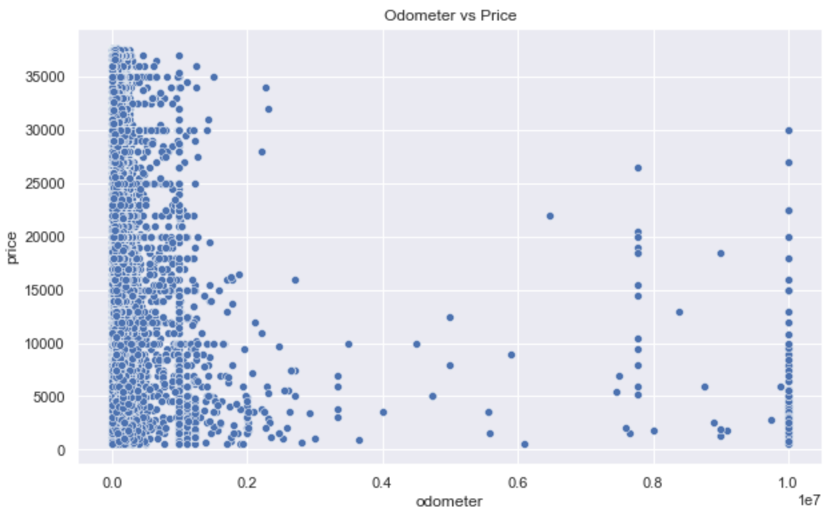
Out[71]:
```
(341976, 14)
```

In [72]:
```
df['odometer'].describe()
```

Out[72]:
```
count    3.400870e+05
mean     1.048883e+05
std      2.005880e+05
min      0.000000e+00
25%      4.600000e+04
50%      9.530800e+04
75%      1.410000e+05
max      1.000000e+07
Name: odometer, dtype: float64
```

In [74]:
```
ax = sns.scatterplot(x="odometer", y="price", data=df)
plt.title('Odometer vs Price')
```

Out[74]:
```
Text(0.5, 1.0, 'Odometer vs Price')
```



In [75]:
```
df["odometer"].max()
```

Out[75]:
```
10000000.0
```

In [76]:
```
df["odometer"].min()
```

Out[76]:
```
0.0
```

From the above scatter plot analysis we can see that there are many outliers that lie towards the 10000000.0 mark. The highest value in the odometer column is 10000000.0 and the lowest value is 0.0. Both of these values are redundant so we will be considering only those values which are relevant.

In [77]:
```
df.drop(df[df["odometer"]==10000000.0].index,inplace=True)
df.drop(df[df["odometer"]==0.0].index,inplace=True)
```

In [78]:
```
ax = sns.scatterplot(x="odometer", y="price", data=df)
ax.get_xaxis().get_major_formatter().set_scientific(False)
ax.get_yaxis().get_major_formatter().set_scientific(False)
plt.title('Scatter of Price Vs Odometer Before Removing Outliers')
```

Out[78]: Text(0.5, 1.0, 'Scatter of Price Vs Odometer Before Removing Outliers')

In [79]:
```
df["odometer"].isna().sum()
```

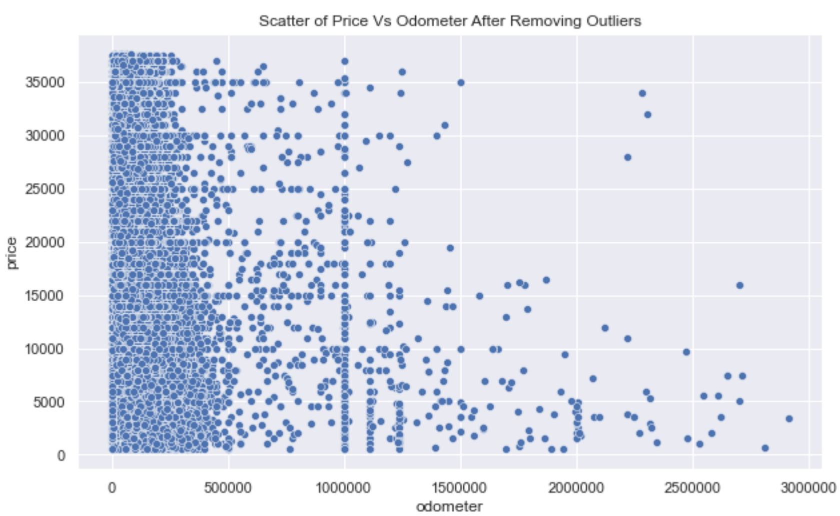
Out[79]: 1889

From the above scatter we can see that most of the values are below the 3000000 mark so we will be only considering those values below the mark

In [80]:
```
df=df[(df.odometer < 3000000)]
```

In [81]:
```
ax = sns.scatterplot(x="odometer", y="price", data=df)
ax.get_xaxis().get_major_formatter().set_scientific(False)
plt.title('Scatter of Price Vs Odometer After Removing Outliers')
```

Out[81]: Text(0.5, 1.0, 'Scatter of Price Vs Odometer After Removing Outliers')

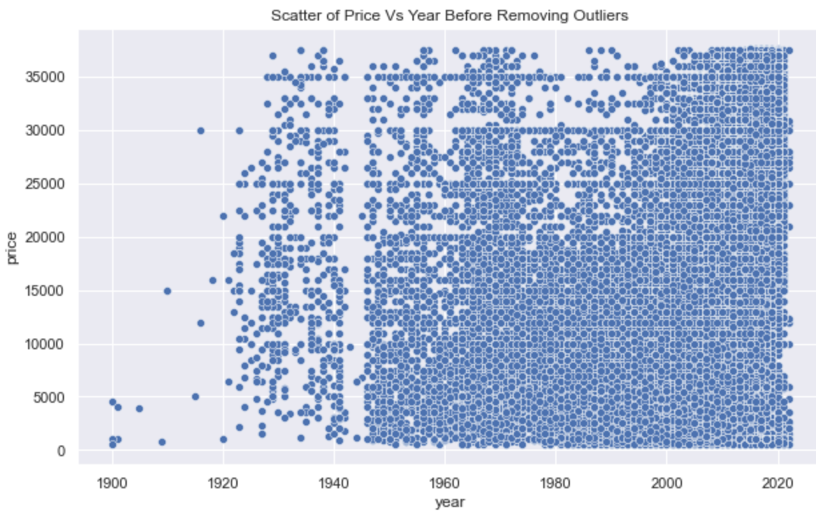


In [82]:
```
df["odometer"].isna().sum()
```

Out[82]: 0

In [84]:
```
bx = sns.scatterplot(x="year", y="price", data=df)
plt.title('Scatter of Price Vs Year Before Removing Outliers')
```

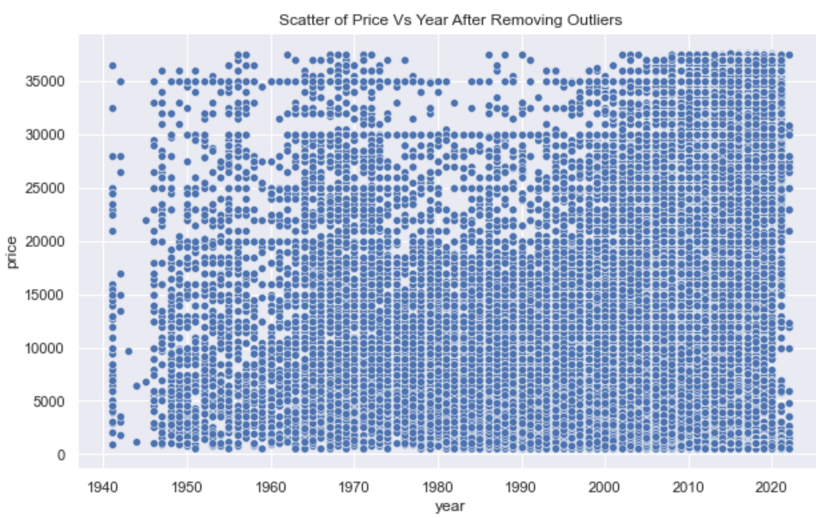
Out[84]: Text(0.5, 1.0, 'Scatter of Price Vs Year Before Removing Outliers')



In [85]:
```
df.drop(df[df["year"]==0.0].index,inplace=True)
df=df.dropna(subset=['year'])
df=df[(df.year > 1940)]
```

In [87]:
```
bx = sns.scatterplot(x="year", y="price", data=df)
plt.title('Scatter of Price Vs Year After Removing Outliers')
```

Out[87]: Text(0.5, 1.0, 'Scatter of Price Vs Year After Removing Outliers')



In [86]:
```
null_values=df.isnull().sum()
null_values=pd.DataFrame(null_values,columns=['null'])
j=1
sum_tot=len(df)
null_values['percent']=null_values['null']/sum_tot
round(null_values*100,3).sort_values('percent',ascending=False)
```

Out[86]:

	null	percent
cylinders	13587200	40.220
condition	12300600	36.411
drive	10397800	30.779
paint_color	9788300	28.974
type	7443300	22.033
manufacturer	1214900	3.596
title_status	569800	1.687
model	364100	1.078
fuel	193200	0.572
transmission	128800	0.381
price	0	0.000
year	0	0.000
odometer	0	0.000
state	0	0.000

In [88]:
```
df.condition.value_counts()
```

Out[88]:
```
good         104845
excellent     85091
like new      17147
fair           6511
new             688
salvage         537
Name: condition, dtype: int64
```

Since the column condition has null values, so we will be using odometer as our base column to fill values into these null values. Based on the value of the odometer we will be setting the condition of the car

In [89]:
```
df.loc[df.year>=2019, 'condition'] = df.loc[df.year>=2019, 'condition'].fillna('new')
```

In [90]:
```
excellent_odo_mean = df[df['condition'] == 'excellent']['odometer'].mean()
good_odo_mean = df[df['condition'] == 'good']['odometer'].mean()
like_new_odo_mean = df[df['condition'] == 'like new']['odometer'].mean()
salvage_odo_mean = df[df['condition'] == 'salvage']['odometer'].mean()
fair_odo_mean = df[df['condition'] == 'fair']['odometer'].mean()
```

In [91]:
```
df.loc[df['odometer'] <= like_new_odo_mean, 'condition'] = df.loc[df['odometer'] <= like_new_odo_mean, 'condition'].fillna('like

df.loc[df['odometer'] >= fair_odo_mean, 'condition'] = df.loc[df['odometer'] >= fair_odo_mean, 'condition'].fillna('fair')

df.loc[((df['odometer'] > good_odo_mean) &
       (df['odometer'] <= excellent_odo_mean)), 'condition'] = df.loc[((df['odometer'] > good_odo_mean) &
       (df['odometer'] <= excellent_odo_mean)), 'condition'].fillna('excellent')

df.loc[((df['odometer'] > like_new_odo_mean) &
       (df['odometer'] <= good_odo_mean)), 'condition'] = df.loc[((df['odometer'] > like_new_odo_mean) &
       (df['odometer'] <= good_odo_mean)), 'condition'].fillna('good')

df.loc[((df['odometer'] > good_odo_mean) &
       (df['odometer'] <= fair_odo_mean)), 'condition'] = df.loc[((df['odometer'] > good_odo_mean) &
       (df['odometer'] <= fair_odo_mean)), 'condition'].fillna('salvage')
```

In [92]:
```
df=df.dropna(subset=['title_status','fuel','transmission','model','manufacturer'])
```

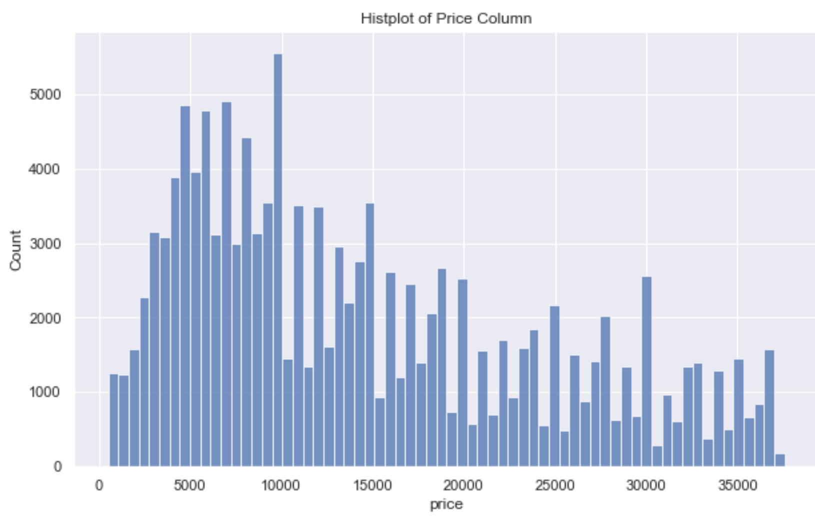
In [93]:
```
df.dropna(inplace=True)
```

In [94]:
```
df.isnull().sum()
```

Out[94]:
```
price           0
year            0
manufacturer    0
model           0
condition       0
cylinders       0
fuel            0
odometer        0
title_status    0
transmission    0
drive           0
type            0
paint_color     0
state           0
dtype: int64
```

After Cleaning the entire data we can see that there are no Null values present in the data as well as from the figure below we can say that our data is fairly balanced based on the target variable.

In [98]:
```
sns.histplot(df['price'])
plt.title('Histplot of Price Column')
plt.show()
```



Data Preprocessing

We will be using LabelEncoder to convert categorical data to numerical data.

In [100]:
```
manufacturer_le = LabelEncoder()
model_le = LabelEncoder()
condition_le = LabelEncoder()
cylinders_le = LabelEncoder()
fuel_le = LabelEncoder()
title_status_le = LabelEncoder()
transmission_le = LabelEncoder()
drive_le = LabelEncoder()
type_le = LabelEncoder()
paint_color_le = LabelEncoder()
state_le = LabelEncoder()
```

In [101]:
```
df['manufacturer'] = manufacturer_le.fit_transform(df['manufacturer'])
df['model'] = model_le.fit_transform(df['model'])
df['condition'] = condition_le.fit_transform(df['condition'])
df['cylinders'] = cylinders_le.fit_transform(df['cylinders'])
df['fuel'] = fuel_le.fit_transform(df['fuel'])
df['title_status'] = title_status_le.fit_transform(df['title_status'])
df['transmission'] = transmission_le.fit_transform(df['transmission'])
df['drive'] = drive_le.fit_transform(df['drive'])
df['type'] = type_le.fit_transform(df['type'])
df['paint_color'] = paint_color_le.fit_transform(df['paint_color'])
df['state'] = state_le.fit_transform(df['state'])
```

In [102]:
```
df.head()
```

Out[102]:

	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	drive	type	paint_color	state
31	15000	2013.0	13	4763	0	5	2	128000.0	0	0	2	10	0	1
32	27990	2012.0	14	9301	2	6	2	68696.0	0	2	0	8	0	1
33	34590	2016.0	7	9424	2	5	2	29499.0	0	2	0	8	9	1
34	35000	2019.0	38	10120	0	5	2	43000.0	0	0	0	10	5	1
35	29990	2016.0	7	3036	2	5	2	17302.0	0	2	0	8	8	1

Feature Selection

Using Random Forest Regressor

In [105]:
```
X = df.drop(['price'], axis=1)
y = df['price']
```

In [107]:
```
rfe_selector = RFE(estimator=RandomForestRegressor(),n_features_to_select = 12, step = 1)
rfe_selector.fit(X, y.astype(int))
cols = rfe_selector.get_support(indices=True)
X_new = X.iloc[:,cols]
X_new.head()
```

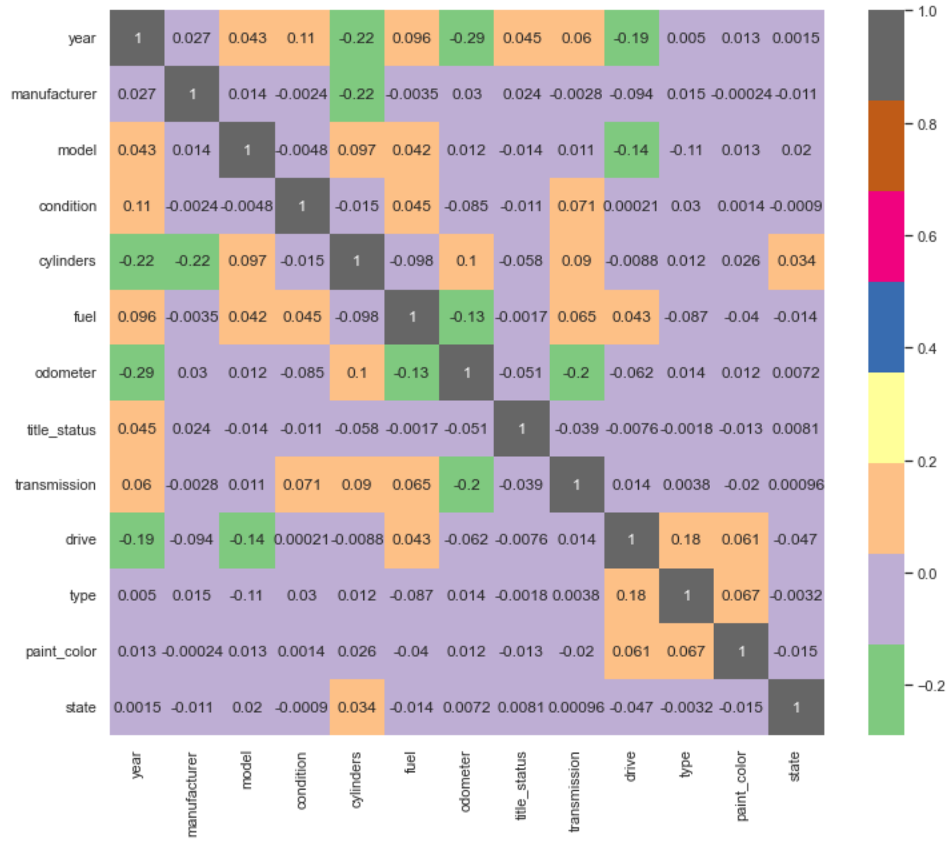
Out[107]:

	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	drive	type	paint_color	state
31	2013.0	13	4763	0	5	2	128000.0	0	2	10	0	1
32	2012.0	14	9301	2	6	2	68696.0	0	0	8	0	1
33	2016.0	7	9424	2	5	2	29499.0	0	0	8	9	1
34	2019.0	38	10120	0	5	2	43000.0	0	0	10	5	1
35	2016.0	7	3036	2	5	2	17302.0	0	0	8	8	1

Using Pearson Co-Relation

In [109]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

plt.figure(figsize=(12,10))
corelation  = X_train.corr()
sns.heatmap(corelation, annot=True, cmap=plt.cm.Accent)
plt.show()
```



In [111]:
```
def find_corelation(data,threshold):
  col_corr = set()
  corr_matrix = data.corr()
  for i in range(len(corr_matrix.columns)):
    for j in range(i):
      if corr_matrix.iloc[i, j] > threshold: # In this function we are considering only positive co-relation
        colname = corr_matrix.columns[i]
        col_corr.add(colname)
  return col_corr
```

In [112]:
```
find_corelation(X_train, 0.7)
```

Out[112]:
```
set()
```

We used pearson co relation to find highly co-related values and from the given output we can see that there no highly co-related values. The threshold for finidng co-relation was set to 0.70.

Model Building & Testing

In [133]:
```
mean_squared_error_list = []
root_mean_squared_error_list = []
mean_absolute_error_list = []
mean_absolute_percentage_error_list = []
r2_score_list = []
```

In [134]:
```
def add_data_to_list(pred):
    mean_squared_error_list.append(mean_squared_error(pred, y_test))
    root_mean_squared_error_list.append(math.sqrt(mean_squared_error(pred, y_test)))
    mean_absolute_error_list.append(mean_absolute_error(pred, y_test))
    mean_absolute_percentage_error_list.append(mean_absolute_percentage_error(pred, y_test))
    r2_score_list.append(r2_score(pred,y_test))
```

In [135]:
```
def print_individual_metrics(pred, algo):
    print("Metrics For: ", '\033[1m' + algo + '\033[0m')
    print("Mean Absolute Percentage Error: ", mean_absolute_percentage_error(pred, y_test))
    print("Mean Absolute Error: ", mean_absolute_error(pred, y_test))
    print("Mean Squared Error: ", mean_squared_error(pred, y_test))
    print("Root Mean Squared Error : ", math.sqrt(mean_squared_error(pred, y_test)))
    print("R2 Score: ", r2_score(pred, y_test))
```

In [169]:
```
X = df.drop(['price'], axis=1)
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Model Building Using Sampling

In [192]:
```
sample_df = df.copy()

sample_df = sample_df.sample(35000)

sample_df = sample_df.sort_values(by='year')
```

In [193]:
```
train_sample = sample_df.head(30000)
test_sample = sample_df.tail(5000)
```

In [194]:
```
X_train_sample = train_sample.drop(['price'], axis=1)
y_train_sample = train_sample['price']

X_test_sample = test_sample.drop(['price'], axis=1)
y_test_sample = test_sample['price']
```

In [196]:
```
sample_rf = RandomForestRegressor()
sample_rf.fit(X_train_sample,y_train_sample)

sample_pred = sample_rf.predict(X_test_sample)
```

In [198]:
```
    print("Metrics For: ", '\033[1m' + 'Sampled Random Forest' + '\033[0m')
    print("Mean Absolute Percentage Error: ", mean_absolute_percentage_error(sample_pred, y_test_sample))
    print("Mean Absolute Error: ", mean_absolute_error(sample_pred, y_test_sample))
    print("Mean Squared Error: ", mean_squared_error(sample_pred, y_test_sample))
    print("Root Mean Squared Error : ", math.sqrt(mean_squared_error(sample_pred, y_test_sample)))
    print("R2 Score: ", r2_score(sample_pred, y_test_sample))
```

```
Metrics For:  Sampled Random Forest
Mean Absolute Percentage Error:  0.17232569659473007
Mean Absolute Error:  3693.4893836978363
Mean Squared Error:  26259583.157638967
Root Mean Squared Error :  5124.410518063416
R2 Score:  0.37525936828513917
```

Random Forest

In [137]:
```
rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)

rfr_pred = rfr.predict(X_test)

rf_error = mean_absolute_percentage_error(rfr_pred, y_test)
add_data_to_list(rfr_pred)

print_individual_metrics(rfr_pred, 'Random Forest')
```

```
Metrics For:  Random Forest
Mean Absolute Percentage Error:  0.15568268784600395
Mean Absolute Error:  1676.8953591615525
Mean Squared Error:  8538928.209779803
Root Mean Squared Error :  2922.144453954972
R2 Score:  0.8979600700372223
```

DNN

In [139]:
```
NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = X_train.shape[1], activation='relu'))

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_error'])
NN_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1792

 dense_1 (Dense)             (None, 256)               33024

 dense_2 (Dense)             (None, 256)               65792

 dense_3 (Dense)             (None, 256)               65792

 dense_4 (Dense)             (None, 1)                 257

=================================================================
Total params: 166,657
Trainable params: 166,657
Non-trainable params: 0
_________________________________________________________________
```

In [140]:
```
checkpoint_name = 'Weights-{epoch:03d}--{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', verbose = 1, save_best_only = True, mode ='auto')
callbacks_list = [checkpoint]
```

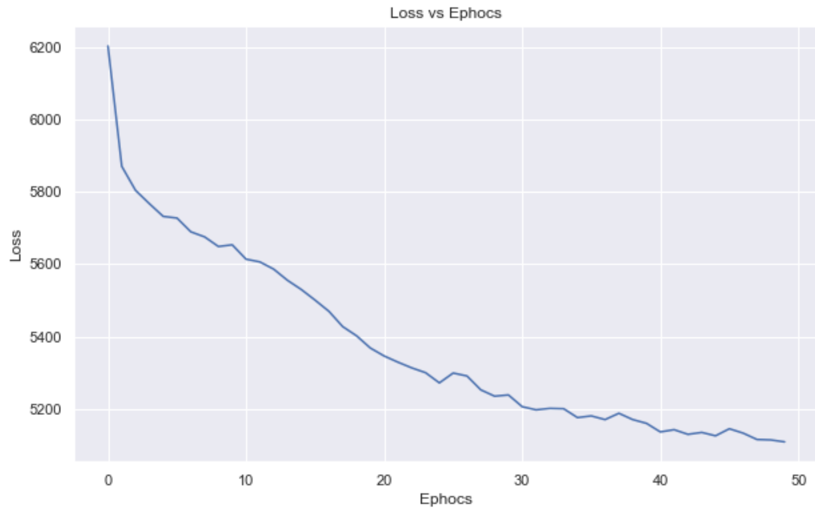
In [141]:
```
history = NN_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split = 0.2, callbacks=callbacks_list)
```

```
Epoch 21/50
2290/2310 [============================>.] - ETA: 0s - loss: 5348.6968 - mean_absolute_error: 5348.6968
Epoch 21: val_loss improved from 5265.18115 to 5130.68408, saving model to Weights-021--5130.68408.hdf5
2310/2310 [==============================] - 6s 2ms/step - loss: 5346.7866 - mean_absolute_error: 5346.7866 - val_loss: 5130.
6841 - val_mean_absolute_error: 5130.6841
Epoch 22/50
2308/2310 [============================>.] - ETA: 0s - loss: 5329.5972 - mean_absolute_error: 5329.5972
Epoch 22: val_loss did not improve from 5130.68408
2310/2310 [==============================] - 6s 2ms/step - loss: 5329.6860 - mean_absolute_error: 5329.6860 - val_loss: 5212.
4937 - val_mean_absolute_error: 5212.4937
Epoch 23/50
2297/2310 [============================>.] - ETA: 0s - loss: 5311.7681 - mean_absolute_error: 5311.7681
Epoch 23: val_loss did not improve from 5130.68408
2310/2310 [==============================] - 6s 3ms/step - loss: 5313.9604 - mean_absolute_error: 5313.9604 - val_loss: 5364.
1162 - val_mean_absolute_error: 5364.1162
Epoch 24/50

2296/2310 [============================>.] - ETA: 0s - loss: 5300.4707 - mean_absolute_error: 5300.4707
Epoch 24: val_loss improved from 5130.68408 to 5121.81201, saving model to Weights-024--5121.81201.hdf5
2310/2310 [==============================] - 6s 2ms/step - loss: 5300.6885 - mean_absolute_error: 5300.6885 - val_loss: 5121.
```

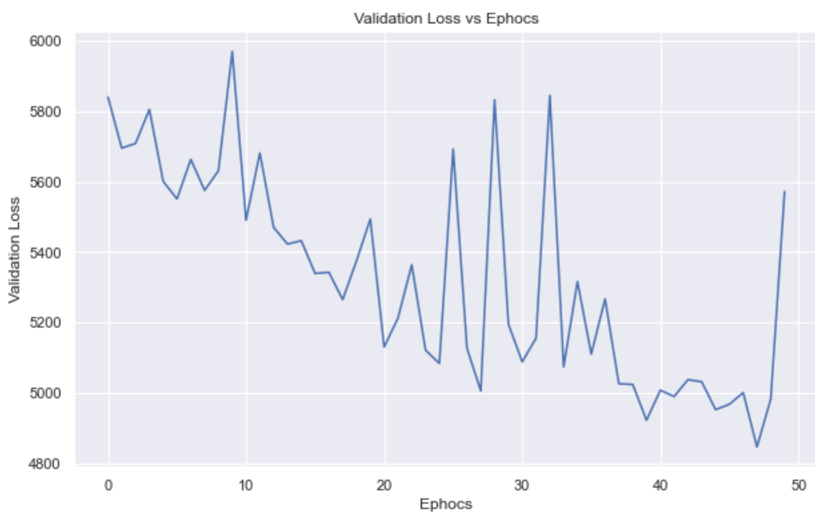
In [142]:
```
plt.plot(history.history['loss'])
plt.xlabel('Ephocs')
plt.ylabel('Loss')
plt.title('Loss vs Ephocs')
```

Out[142]: Text(0.5, 1.0, 'Loss vs Ephocs')



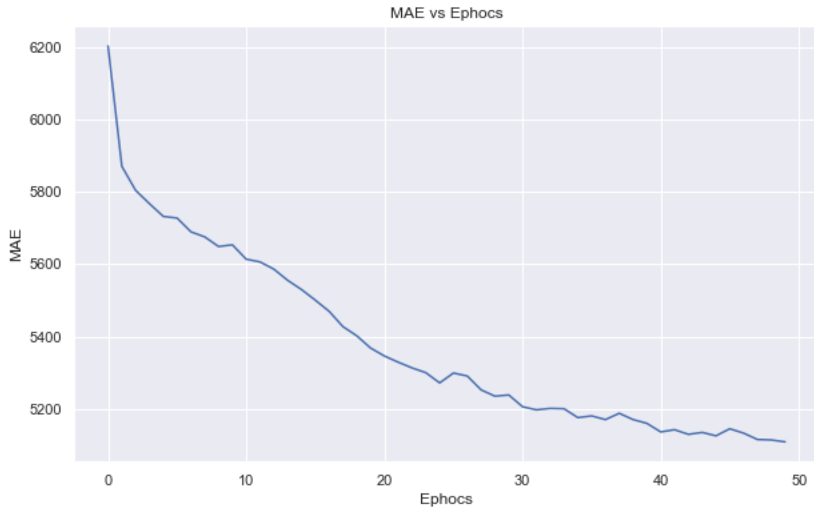
In [143]:
```
plt.plot(history.history['val_loss'])
plt.xlabel('Ephocs')
plt.ylabel('Validation Loss')
plt.title('Validation Loss vs Ephocs')
```

Out[143]: Text(0.5, 1.0, 'Validation Loss vs Ephocs')



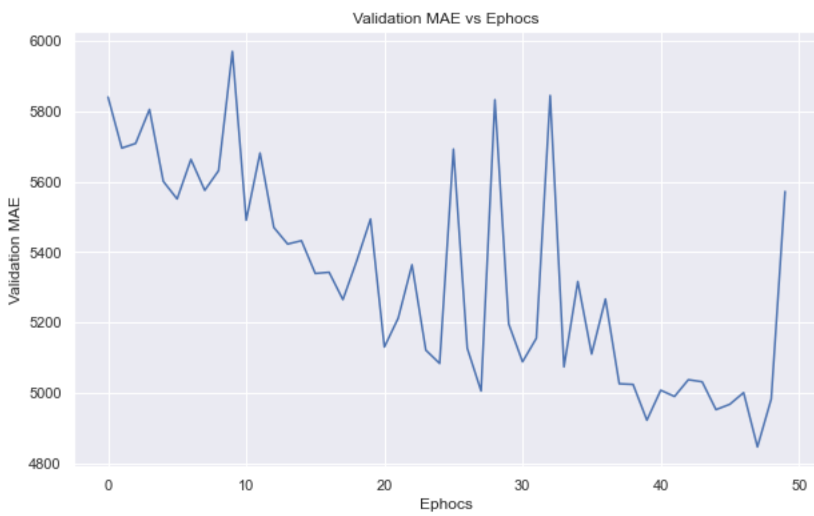
In [144]:
```
plt.plot(history.history['mean_absolute_error'])
plt.xlabel('Ephocs')
plt.ylabel('MAE')
plt.title('MAE vs Ephocs')
```

Out[144]: Text(0.5, 1.0, 'MAE vs Ephocs')



In [145]:
```
plt.plot(history.history['val_mean_absolute_error'])
plt.xlabel('Ephocs')
plt.ylabel('Validation MAE')
plt.title('Validation MAE vs Ephocs')
```

Out[145]: Text(0.5, 1.0, 'Validation MAE vs Ephocs')



In [146]:
```
dnn_pred = NN_model.predict(X_test)
```

```
1238/1238 [==============================] - 1s 1ms/step
```

In [147]:
```
add_data_to_list(dnn_pred)
print_individual_metrics(dnn_pred, 'DNN')
```

```
Metrics For:  DNN
Mean Absolute Percentage Error:  0.6415725790840552
Mean Absolute Error:  5607.323595585602
Mean Squared Error:  62883892.442196995
Root Mean Squared Error :  7929.9364715107895
R2 Score:  -0.369584889392079
```

Linear Regression

In [148]:
```
reg = LinearRegression().fit(X_train, y_train)

reg_pred = reg.predict(X_test)

add_data_to_list(reg_pred)
```

In [151]:
```
print_individual_metrics(reg_pred, 'Linear Regression')
```

```
Metrics For:  Linear Regression
Mean Absolute Percentage Error:  2.162309496211331
Mean Absolute Error:  5110.434505411953
Mean Squared Error:  46949927.86361402
Root Mean Squared Error :  6852.0017413609885
R2 Score:  -0.04002914075761721
```

XGBoost

In [150]:
```
xgb = XGBRegressor()
xgb.fit(X_train,y_train)

xgb_pred = xgb.predict(X_test)

add_data_to_list(xgb_pred)
```

In [152]:
```
print_individual_metrics(xgb_pred, 'XGBoost')
```

```
Metrics For:  XGBoost
Mean Absolute Percentage Error:  0.2039359377125654
Mean Absolute Error:  2125.0758294815696
Mean Squared Error:  10029012.676804695
Root Mean Squared Error :  3166.8616447209524
R2 Score:  0.8787716425664734
```

Light GBM

In [154]:
```
lgbm = LinearRegression()

lgbm.fit(X_train,y_train)

lgbm_pred = lgbm.predict(X_test)

add_data_to_list(lgbm_pred)
```

In [155]:
```
print_individual_metrics(lgbm_pred, 'Light GBM')
```

```
Metrics For:  Light GBM
Mean Absolute Percentage Error:  2.162309496211331
Mean Absolute Error:  5110.434505411953
Mean Squared Error:  46949927.86361402
Root Mean Squared Error :  6852.0017413609885
R2 Score:  -0.04002914075761721
```

Model Comparison

In [201]:
```
models = pd.DataFrame({
    'Model': ['Random Forest', 'Deep Neural Network', 'Linear Regression', 'XGBoost', 'Light GBM'],
    'mean_squared_error' : mean_squared_error_list,
    'root_mean_squared_error' : root_mean_squared_error_list,
    'mean_absolute_error' : mean_absolute_error_list,
    'mean_absolute_percentage_error':mean_absolute_percentage_error_list,
    'r2_score':r2_score_list
                      })
```

In [202]:
```
pd.options.display.float_format = '{:,.2f}'.format
```

In [158]:
```
models.sort_values(by=['mean_absolute_percentage_error'], ascending=True)
```

Out[158]:

	Model	mean_squared_error	root_mean_squared_error	mean_absolute_error	mean_absolute_percentage_error	r2_score
0	Random Forest	8,538,928.21	2,922.14	1,676.90	0.16	0.90
3	XGBoost	10,029,012.68	3,166.86	2,125.08	0.20	0.88
1	Deep Neural Network	62,883,892.44	7,929.94	5,607.32	0.64	-0.37
2	Linear Regression	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
4	Light GBM	46,949,927.86	6,852.00	5,110.43	2.16	-0.04

In [159]:
```
models.sort_values(by=['r2_score'], ascending=False)
```

Out[159]:

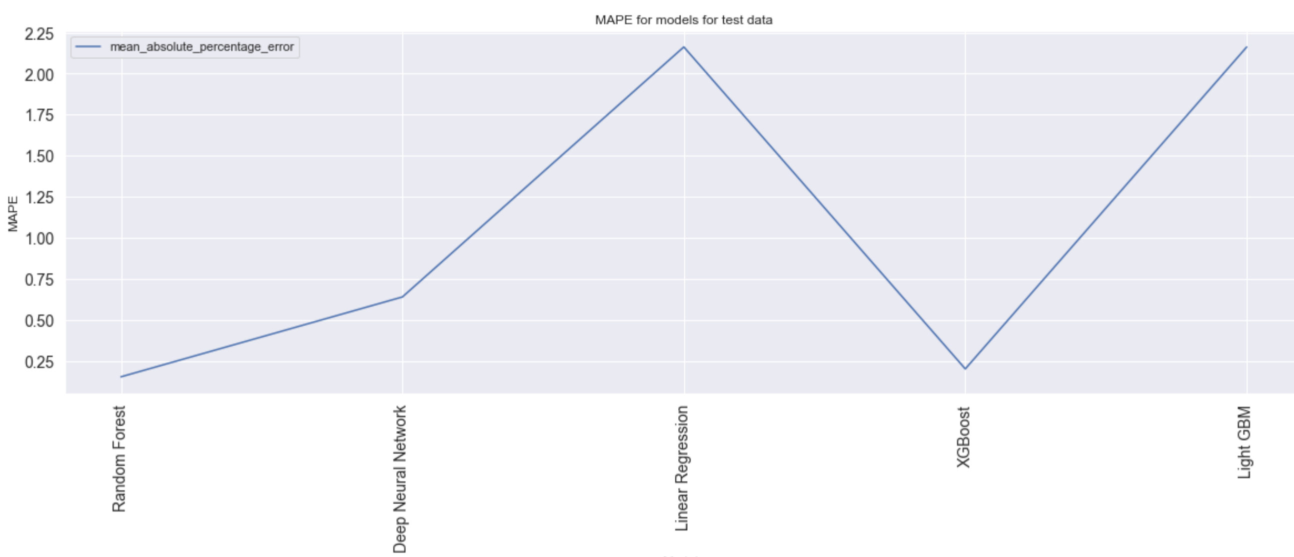
	Model	mean_squared_error	root_mean_squared_error	mean_absolute_error	mean_absolute_percentage_error	r2_score
0	Random Forest	8,538,928.21	2,922.14	1,676.90	0.16	0.90
3	XGBoost	10,029,012.68	3,166.86	2,125.08	0.20	0.88
2	Linear Regression	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
4	Light GBM	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
1	Deep Neural Network	62,883,892.44	7,929.94	5,607.32	0.64	-0.37

In [160]:
```
models.sort_values(by=['mean_squared_error'], ascending=True)
```

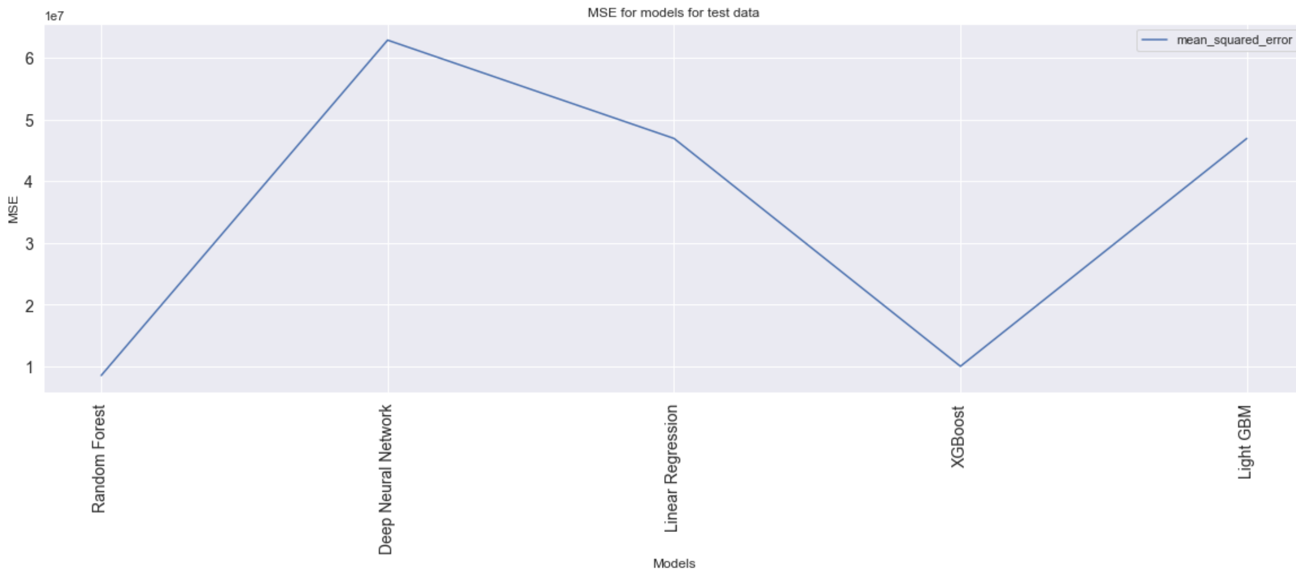
Out[160]:

	Model	mean_squared_error	root_mean_squared_error	mean_absolute_error	mean_absolute_percentage_error	r2_score
0	Random Forest	8,538,928.21	2,922.14	1,676.90	0.16	0.90
3	XGBoost	10,029,012.68	3,166.86	2,125.08	0.20	0.88
2	Linear Regression	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
4	Light GBM	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
1	Deep Neural Network	62,883,892.44	7,929.94	5,607.32	0.64	-0.37

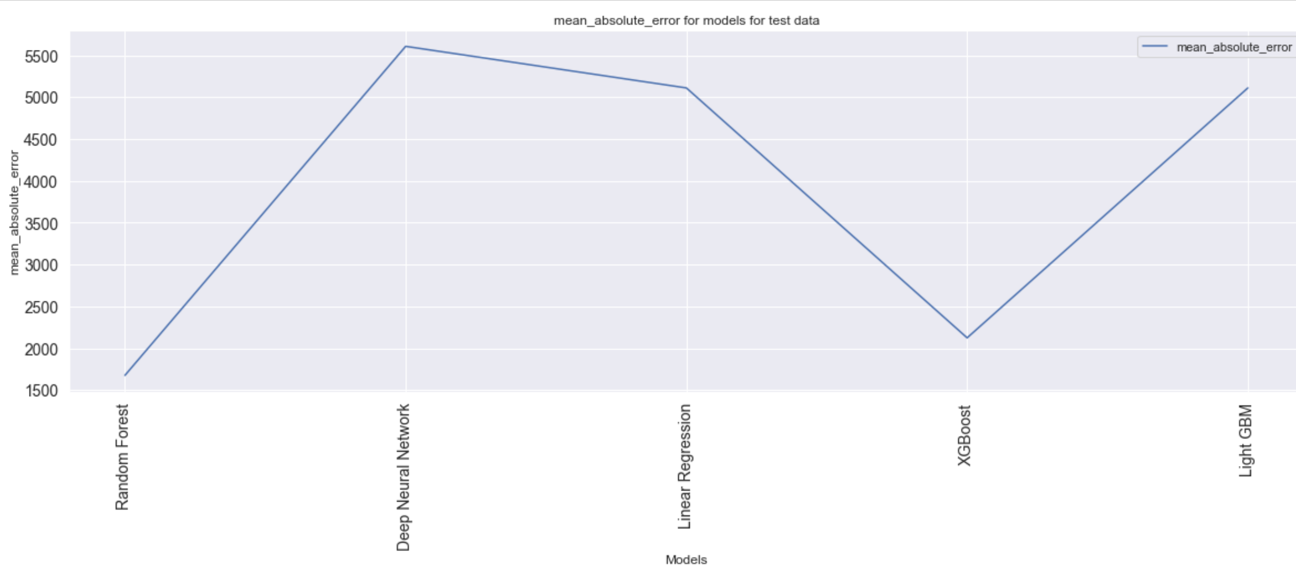
In [163]:
```
plt.figure(figsize=[20,6])
xx = models['Model']
plt.tick_params(labelsize=14)
plt.plot(xx, models['mean_absolute_percentage_error'], label = 'mean_absolute_percentage_error')
plt.legend()
plt.title('MAPE for models for test data')
plt.xlabel('Models')
plt.ylabel('MAPE')
plt.xticks(xx, rotation='vertical')
plt.show()
```



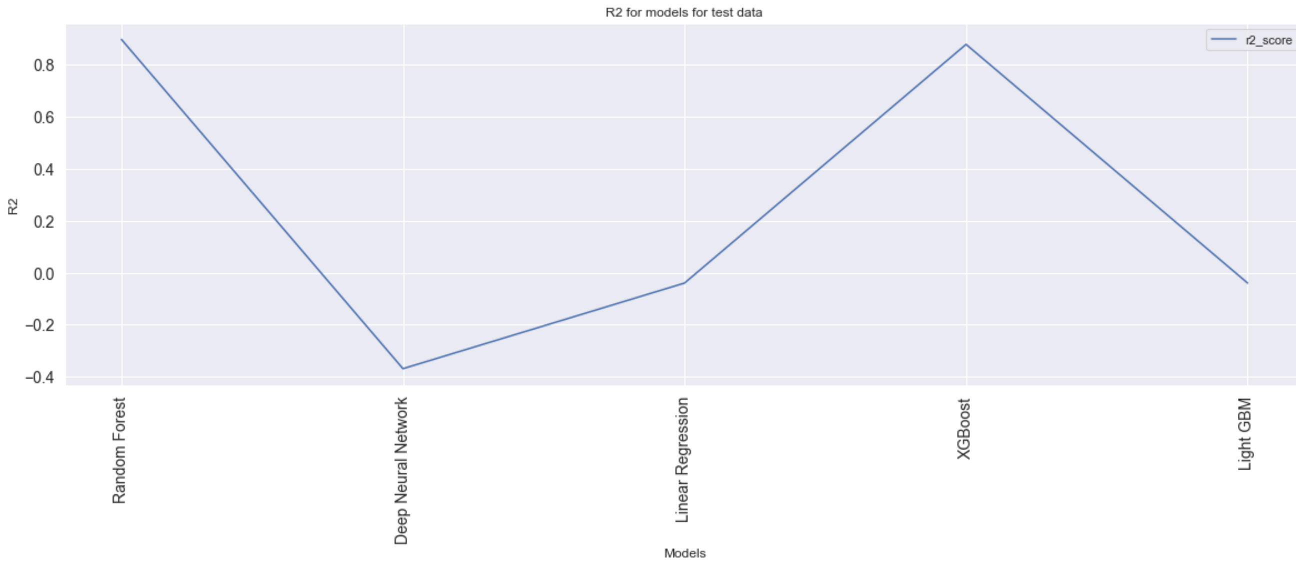
In [164]:
```python
plt.figure(figsize=[20,6])
xx = models['Model']
plt.tick_params(labelsize=14)
plt.plot(xx, models['mean_squared_error'], label = 'mean_squared_error')
plt.legend()
plt.title('MSE for models for test data')
plt.xlabel('Models')
plt.ylabel('MSE')
plt.xticks(xx, rotation='vertical')
plt.show()
```



In [165]:
```python
plt.figure(figsize=[20,6])
xx = models['Model']
plt.tick_params(labelsize=14)
plt.plot(xx, models['mean_absolute_error'], label = 'mean_absolute_error')
plt.legend()
plt.title('mean_absolute_error for models for test data')
plt.xlabel('Models')
plt.ylabel('mean_absolute_error')
plt.xticks(xx, rotation='vertical')
plt.show()
```



In [166]:
```
plt.figure(figsize=[20,6])
xx = models['Model']
plt.tick_params(labelsize=14)
plt.plot(xx, models['r2_score'], label = 'r2_score')
plt.legend()
plt.title('R2 for models for test data')
plt.xlabel('Models')
plt.ylabel('R2')
plt.xticks(xx, rotation='vertical')
plt.show()
```



Random Forest with Randomized Search CV & Hyper Parameter Tuning

In [168]:
```
n_estimators = [5,20,50,100] # number of trees in the random forest
max_features = ['auto', 'sqrt'] # number of features in consideration at every split
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)] # maximum number of levels allowed in each decision tree
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False] # method used to sample data points

random_grid = {'n_estimators': n_estimators,

'max_features': max_features,

'max_depth': max_depth,

'min_samples_split': min_samples_split,

'min_samples_leaf': min_samples_leaf,

'bootstrap': bootstrap}

rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf,param_distributions = random_grid,
               n_iter = 100, cv = 5, verbose=2, random_state=35, n_jobs = -1)

rf_random.fit(X_train, y_train)

print ('Best Parameters: ', rf_random.best_params_, ' \n')
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters:  {'n_estimators': 50, 'min_samples_split': 6, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 100,
'bootstrap': False}
```

In [170]:
```
rfr_gc = RandomForestRegressor(n_estimators = 50, min_samples_split = 6, min_samples_leaf= 1, max_features = 'sqrt', max_depth= 1
rfr_gc.fit(X_train, y_train)
```

Out[170]:
```
RandomForestRegressor(bootstrap=False, max_depth=100, max_features='sqrt',
                      min_samples_split=6, n_estimators=50)
```

In [171]:
```
rf_pred = rfr_gc.predict(X_test)
```

In [172]:
```
print("Mean Absolute Percentage Error: ", mean_absolute_percentage_error(rf_pred, y_test))
print("Mean Absolute Error: ", mean_absolute_error(rf_pred, y_test))
print("Mean Squared Error: ", mean_squared_error(rf_pred, y_test))
print("Root Mean Squared Error : ", math.sqrt(mean_squared_error(rf_pred, y_test)))
print("R2 Score: ", r2_score(rf_pred, y_test))
```

```
Mean Absolute Percentage Error:  0.15278040297931464
Mean Absolute Error:  1668.8892517372626
Mean Squared Error:  8379736.861651168
Root Mean Squared Error :  2894.7775150520924
R2 Score:  0.8958249637889165
```

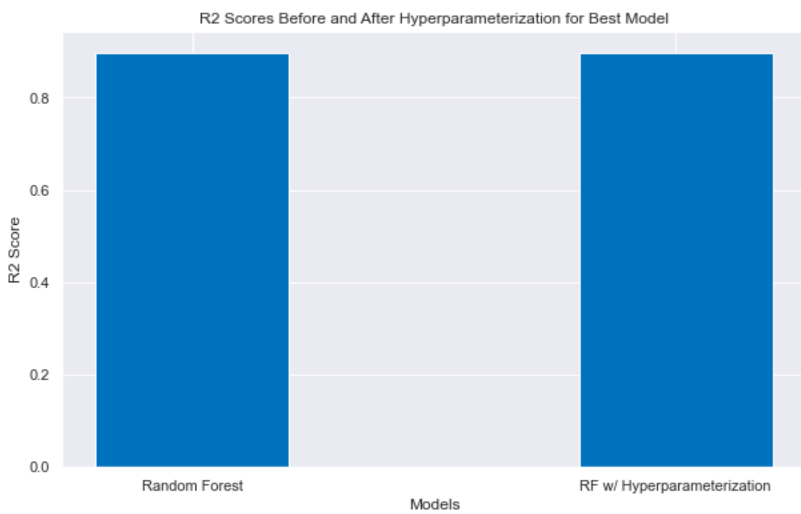
In [173]:
```
print("Percentage of Values Accurately Predicted: ", 1 - mean_absolute_percentage_error(rf_pred, y_test))
```

```
Percentage of Values Accurately Predicted:  0.8472195970206854
```

In [175]:
```
data = {'Random Forest':0.8979, 'RF w/ Hyperparameterization':0.8958}
models = list(data.keys())
r2_values = list(data.values())

plt.bar(models, r2_values, color ='#0072BD',
        width = 0.4)

plt.xlabel("Models")
plt.ylabel("R2 Score")
plt.title("R2 Scores Before and After Hyperparameterization for Best Model")
plt.show()
```



Statistical Significance Test

In [204]:
```
models.sort_values(by=['r2_score'], ascending=False)
```

Out[204]:

	Model	mean_squared_error	root_mean_squared_error	mean_absolute_error	mean_absolute_percentage_error	r2_score
0	Random Forest	8,538,928.21	2,922.14	1,676.90	0.16	0.90
3	XGBoost	10,029,012.68	3,166.86	2,125.08	0.20	0.88
2	Linear Regression	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
4	Light GBM	46,949,927.86	6,852.00	5,110.43	2.16	-0.04
1	Deep Neural Network	62,883,892.44	7,929.94	5,607.32	0.64	-0.37

In [222]:
```
sns.catplot(x="Model", y="r2_score", data=models, kind='bar')
plt.xticks(rotation=45)
plt.title('R Squared Error for Statistical Significance Test')
plt.ylabel('R Squared Error')
```

Out[222]:
```
Text(2.585000000000001, 0.5, 'R Squared Error')
```