Regression Task

Data preparation

```
# import libraries
import csv
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from google.colab import drive
```

Importing the Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Path to our CSV file
csv_file_path = '/content/drive/Shareddrives/Machine Learning/vehicles.csv'

data = []
with open(csv_file_path, 'r', errors='replace') as file:
    reader = csv.reader(file)
    for row in reader:
        data.append(row)

df = pd.DataFrame(data[1:], columns=data[0])
```

Let's visualize the Dataset, its characteristic and understand our variables. Also we are gonna clean and prepare our dataset to further analysis

```
df
```

	id	url	region	region_url	price	year	manufacturer	model	c
0	7222695916	https://prescott.craigslist.org/cto/d/prescott...	prescott	https://prescott.craigslist.org	6000				
1	7218891961	https://fayar.craigslist.org/ctd/d/bentonville...	fayetteville	https://fayar.craigslist.org	11900				
2	7221797935	https://keys.craigslist.org/cto/d/summerland-k	florida keys	https://keys.craigslist.org	21000				

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202711 entries, 0 to 202710
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            202711 non-null  object
 1   url           202711 non-null  object
 2   region        202711 non-null  object
 3   region_url    202711 non-null  object
 4   price         202711 non-null  object
 5   year          202711 non-null  object
 6   manufacturer  202711 non-null  object
 7   model         202711 non-null  object
 8   condition     202711 non-null  object
 9   cylinders     202711 non-null  object
 10  fuel          202711 non-null  object
 11  odometer      202711 non-null  object
 12  title_status  202711 non-null  object
 13  transmission  202711 non-null  object
 14  VIN           202711 non-null  object
 15  drive         202711 non-null  object
 16  size          202711 non-null  object
 17  type          202711 non-null  object
 18  paint_color   202711 non-null  object
 19  image_url     202711 non-null  object
 20  description   202711 non-null  object
 21  county        202710 non-null  object
 22  state         202710 non-null  object
 23  lat           202710 non-null  object
 24  long          202710 non-null  object
 25  posting_date  202710 non-null  object
dtypes: object(26)
memory usage: 40.2+ MB
```

We can see that the dataset has around 200k rows and 26 features.

```
# check if there are some null data
print("Number of null data\n", df.isnull().sum())
```

```
Number of null data
 id              0
url             0
region          0
region_url      0
price           0
year            0
manufacturer    0
model           0
condition       0
cylinders       0
fuel            0
odometer        0
title_status    0
transmission    0
VIN             0
drive           0
size            0
type            0
paint_color     0
image_url       0
description     0
county          1
state           1
lat             1
long            1
posting_date    1
dtype: int64
```

```
# check for duplicated rows
duplicate_rows = df[df.duplicated()]
print(duplicate_rows)
```

```
Empty DataFrame
Columns: [id, url, region, region_url, price, year, manufacturer, model, condition, cylinders, fuel, odometer, title_status, transmissio
Index: []

[0 rows x 26 columns]
```

```
# there are no duplicates value, however there are some null values
# we remove the only 4 missing value found, since they are a few and it is not worth to estimate them
df = df.dropna()
```

With the *describe()* method, we can see some interesting facts about the dataset, for example count, unique value, top values or frequencies.

```
summary_stats = df.describe(include='all')
summary_stats
```

	id	url	region	region_url	price	year	manufacturer	model	conditio
count	202710	202710	202710	202710	202710	202710	202710	202710	20271
unique	202710	202710	192	192	10691	107	43	19038	
top	7222695916	https://prescott.craigslist.org/cto/d/prescott...	orlando	https://orlando.craigslist.org	0	2017	ford	f-150	
freq	1	1	2983	2983	13845	17303	32612	3510	7986

4 rows × 26 columns

```
df.shape
```

```
(202710, 26)
```

We can see that we "lost" just one row.

```
df.columns
```

```
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')
```

```
df.describe(include="object")
```

	id	url	region	region_url	price	year	manufacturer	model	conditio
count	202710	202710	202710	202710	202710	202710	202710	202710	20271
unique	202710	202710	192	192	10691	107	43	19038	
top	7222695916	https://prescott.craigslist.org/cto/d/prescott...	orlando	https://orlando.craigslist.org	0	2017	ford	f-150	
freq	1	1	2983	2983	13845	17303	32612	3510	7986

4 rows × 26 columns

```
for column in df.columns:
    dtype = df[column].dtype
    print(f"{column}: {dtype}")
```

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

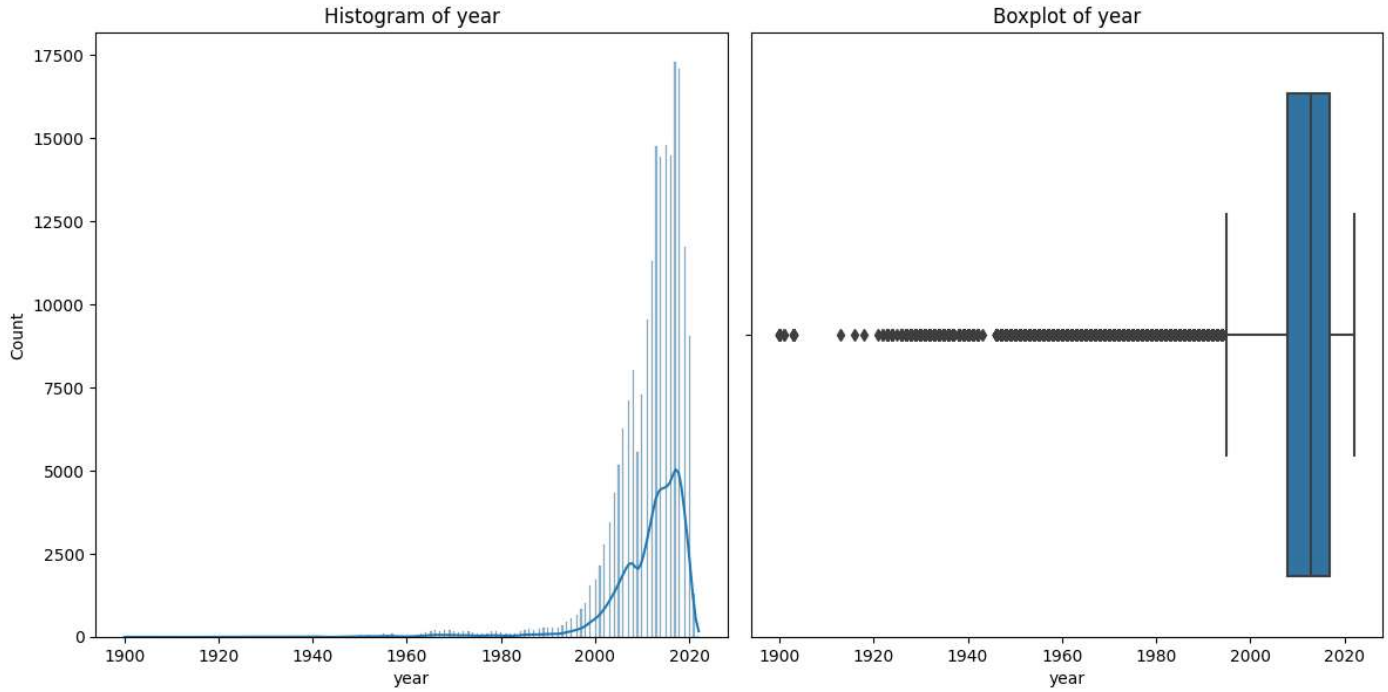
We need to convert some features from "object" type to numerical values.

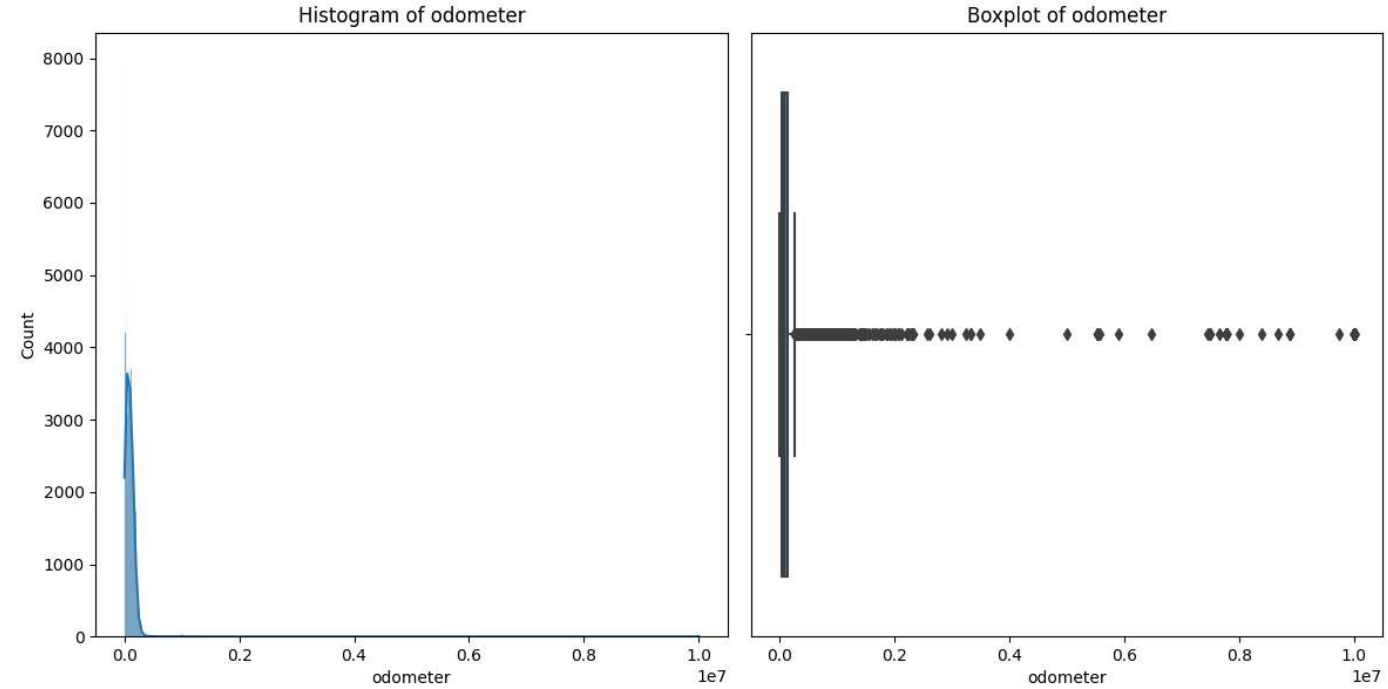
```
# Convert 'year' to numeric
df['year'] = pd.to_numeric(df['year'], errors='coerce')

# Convert 'odometer' to numeric
df['odometer'] = pd.to_numeric(df['odometer'].str.replace('([^\d])', '', regex=True), errors='coerce')

# Convert 'lat' and 'long' to numeric
df['lat'] = pd.to_numeric(df['lat'], errors='coerce')
df['long'] = pd.to_numeric(df['long'], errors='coerce')

#errors='coerce' it's use to handle eventual errors during the conversion

# Check the data types after conversion
print(df.dtypes)
```

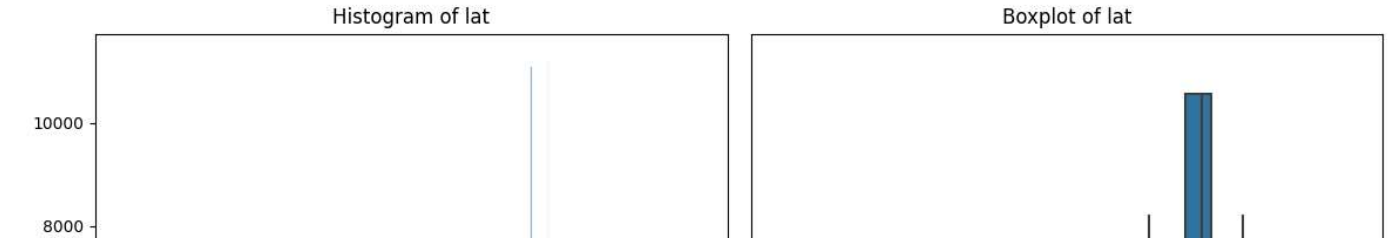
```
id               object
url              object
region           object
region_url       object
price            object
year            float64
manufacturer     object
model            object
condition        object
cylinders        object
fuel             object
odometer        float64
title_status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

EDA Phase: let's start by adding some visualization of our variables and the relations between them. We are using histograms and boxplots.

```
num_col = list(df.select_dtypes(include=["int64", "float64"]).columns)
cat_col = list(df.select_dtypes(include="object").columns)


fig, axes = plt.subplots(nrows=len(num_col), ncols=2, figsize=(12, 6 * len(num_col)))

# Loop through each numeric column and plot histogram and boxplot
for i, col in enumerate(num_col):
    # Plot histogram
    sns.histplot(df[col], kde=True, ax=axes[i, 0])
    axes[i, 0].set_title(f'Histogram of {col}')

    # Plot boxplot
    sns.boxplot(x=df[col], ax=axes[i, 1])
    axes[i, 1].set_title(f'Boxplot of {col}')

# Adjust layout
plt.tight_layout()
plt.show()
```



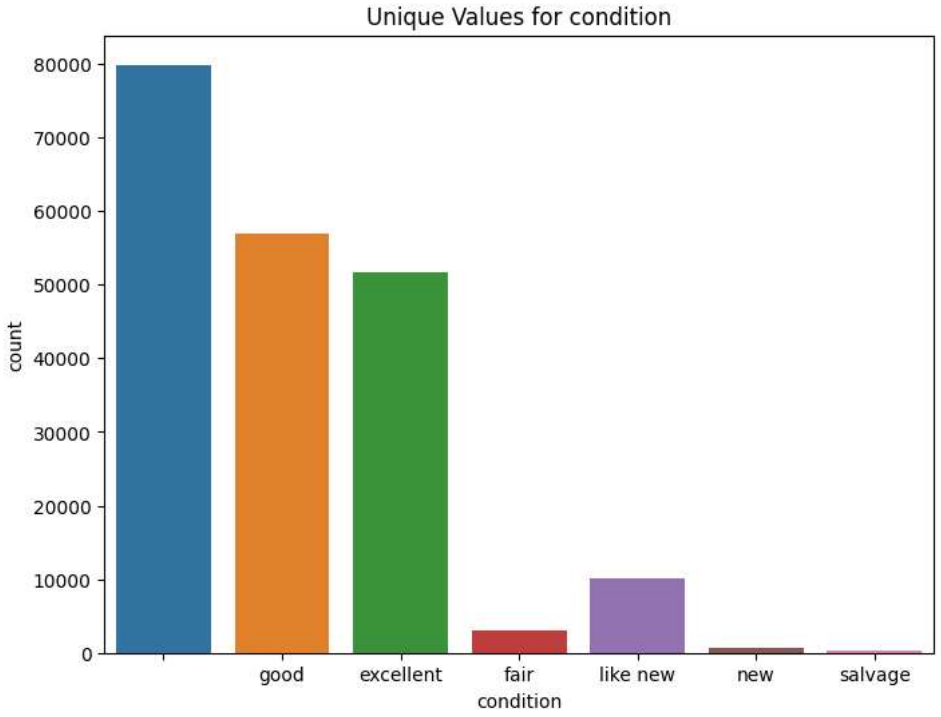


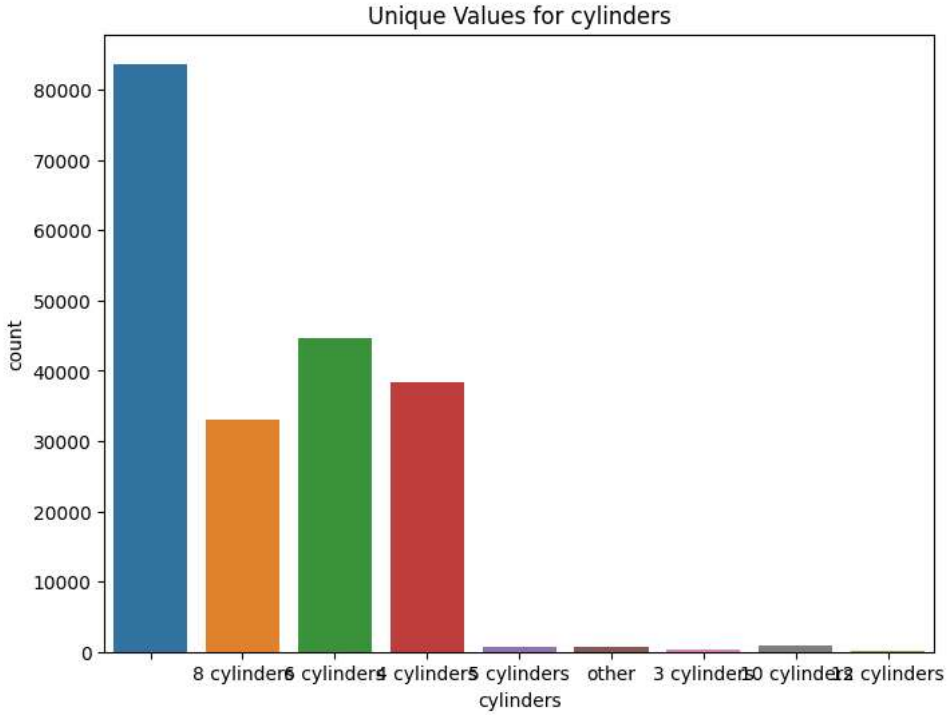


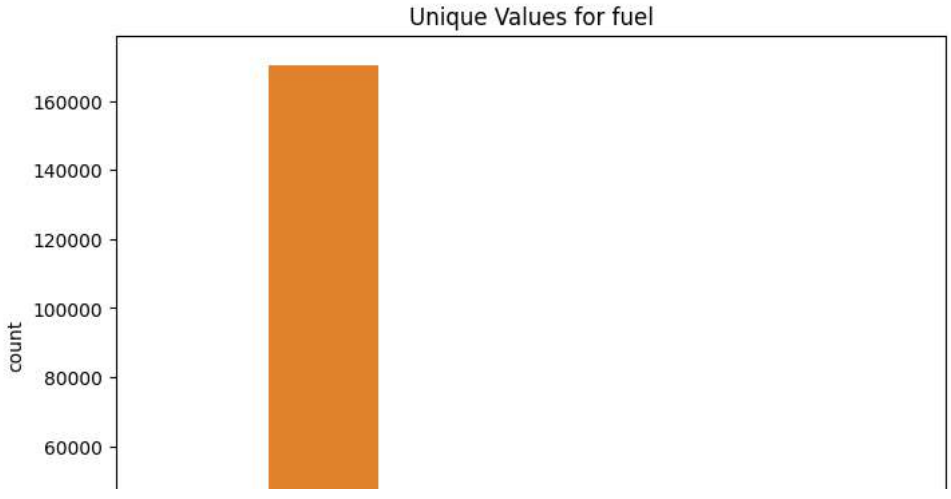
We can see that we work with cars mostly between the years 1990 and 2020 with odometer values usually from 0 to around 200 000 miles.

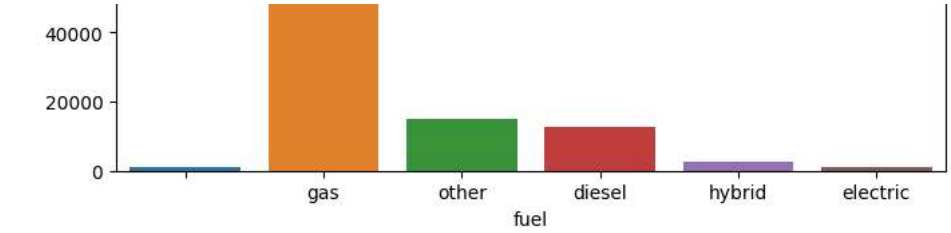
From the Boxplots we can see how the outliers in our data are distributed

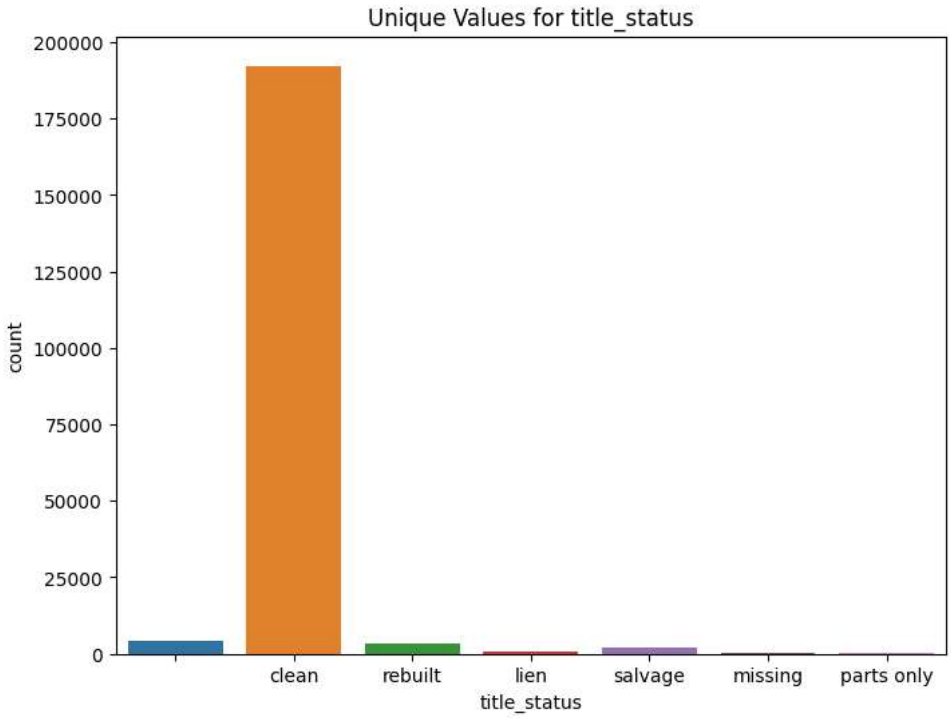
```
    for col in cat_col:
        unique_values = df[col].unique()
        num_unique_values = len(unique_values)

        if num_unique_values < 15:
            plt.figure(figsize=(8, 6))
            sns.countplot(x=col, data=df, order=unique_values)
            plt.title(f'Unique Values for {col}')
            plt.show()
```

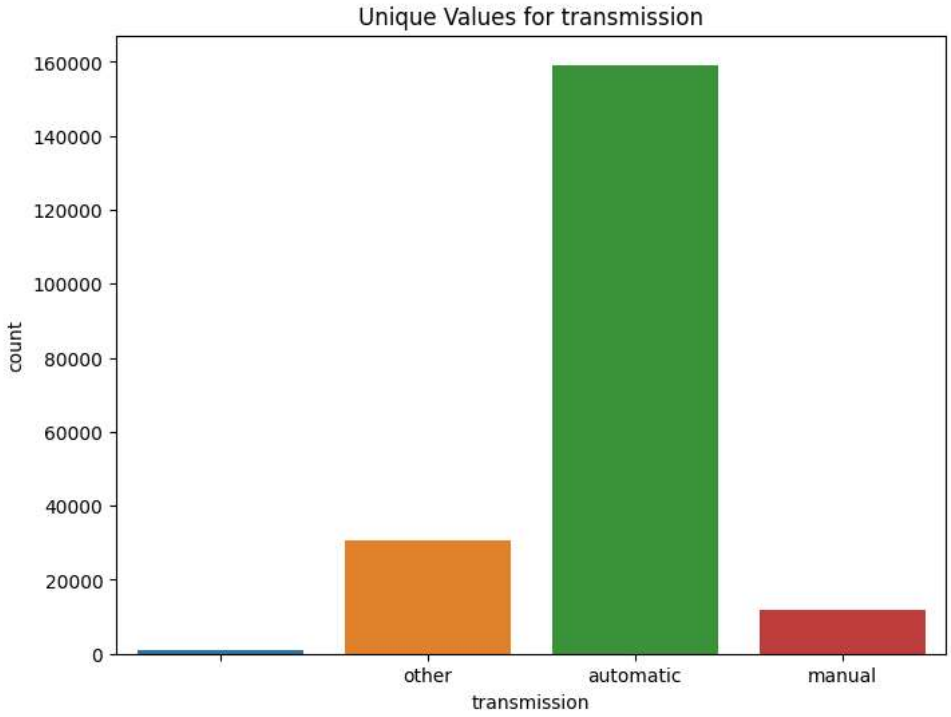

Unique Values for condition
count
condition
good
excellent
fair
like new
new
salvage

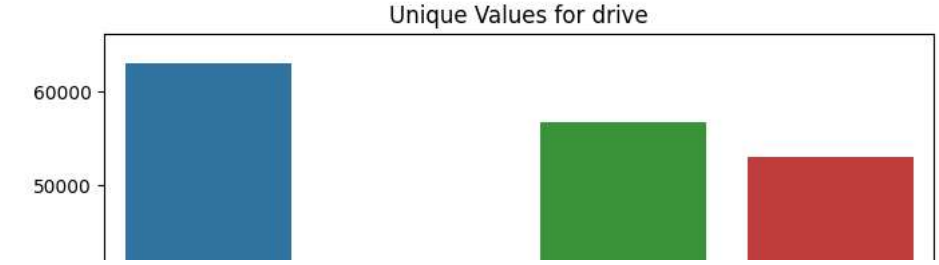

Unique Values for cylinders
count
cylinders
8 cylinders
6 cylinders
4 cylinders
5 cylinders
other
3 cylinders
10 cylinders
12 cylinders

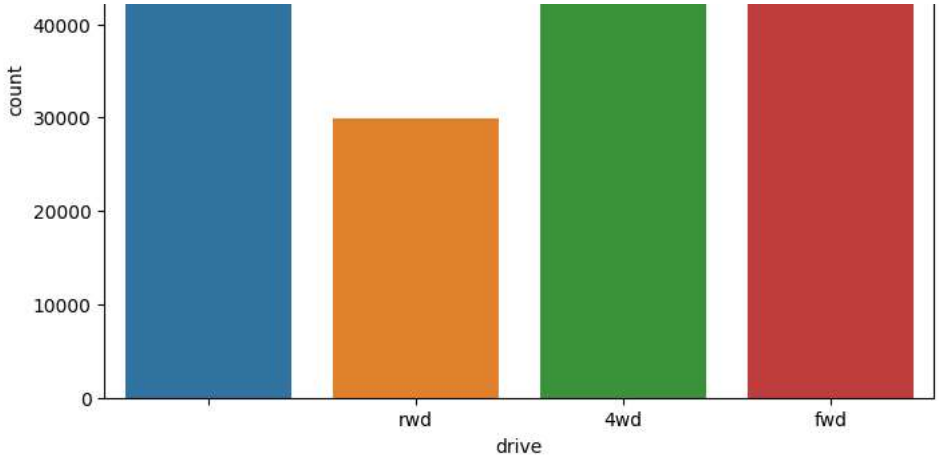

Unique Values for fuel
count

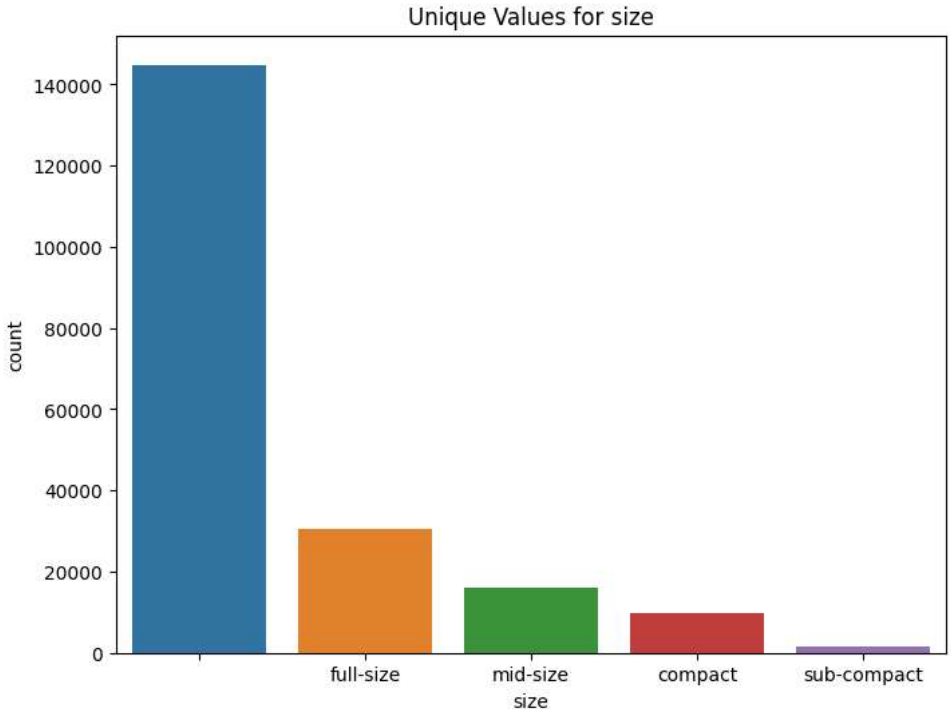

40000
20000
0
gas
other
diesel
hybrid
electric
fuel

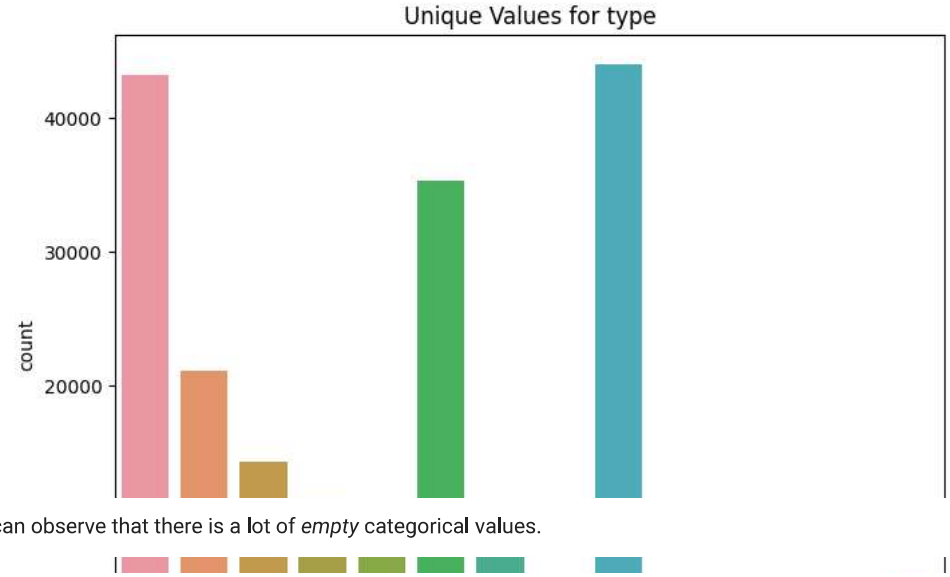

Unique Values for title_status
200000
175000
150000
125000
100000
75000
50000
25000
0
count
clean
rebuilt
lien
salvage
missing
parts only
title_status


Unique Values for transmission
160000
140000
120000
100000
80000
60000
40000
20000
0
count
other
automatic
manual
transmission


Unique Values for drive
60000
50000







We can observe that there is a lot of *empty* categorical values.

In order to reduce the dimension of our problem and making the computations easier and faster, We remove columns that do not provide any valuable information and also the county column since it's empty.

pickup truck other coupe SUV hatchback mini-van sedan offroad bus van convertible wagon

```
df.drop(columns=["url", "region_url", "image_url","county"], inplace=True)
```

Unique Values for paint_color

Focusing of our Target variable (price) we saw that it has a very skew distribution. More over there are some clear mistakes during the transcription of the data into our dataset by hand so we are gonna remove it:

```
# let's set the treshold for price to be 50 as minimum and 1000000 as maximum

df["price"] = pd.to_numeric(df["price"], errors="coerce")
min_price = df[df["price"] < 50].index
max_price = df[df["price"] > 1000000].index
df.drop(index=min_price, axis=0, errors="ignore", inplace=True)
df.drop(index=max_price, axis=0, errors="ignore", inplace=True)
```

Same goes for odometer, we are gonna remove the cars with a too many miles and the one with too little (since they should be labeled as new)

```
# tresholds for odometer: 500 minimum and 300 000 maximum

low_odometer = df[(df["odometer"] < 500) & (df["condition"] != "new")].index
high_odometer = df[df["odometer"] > 300000].index
df.drop(index=low_odometer, axis=0, errors="ignore", inplace=True)
df.drop(index=high_odometer, axis=0, errors="ignore", inplace=True)
```

FEATURE ENGINEERING

Now we can perform a better analysis about our outliers. Usually in a normal dataset we can find some outliers, but removing them usually brings a loss of information. That results in a worse performing model so we do not apply it. In this case, since we are talking about a real dataset with real data (not a syntetic one like we usually do), it may be worth to furtherly analyze how our model performs with/without outliers using different techniques.

Feature Scaling Feature scaling is important when we are using models with a distance metric. If our features are of different scales, they can be overcompensated in the models. We have:

- Absolute Max Scaling
- MinMax Scaling
- Z-Score Normalization (Standard Scaler)
- Robust Scaler

The first two tecniques are not robuste so it's useless to even try it, instead Z-score normalization assume a normal distribution that is inaccurate in our case since we have the Year variable => we apply the Robust Scaler.

```
from sklearn.model_selection import train_test_split

# split the features and the target (price)
X = df.drop('price',axis =1 )
y = df[['price']]

# divide dataset on train and test part
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


from sklearn.preprocessing import RobustScaler

numeric_cols = ['year', 'odometer']
df_rob = X_train.copy()

# Convert non-numeric values to NaN and then fill NaN with the median
df_rob[numeric_cols] = df_rob[numeric_cols].apply(pd.to_numeric, errors='coerce')
df_rob[numeric_cols] = df_rob[numeric_cols].fillna(df_rob[numeric_cols].median())

# Apply RobustScaler
df_rob[numeric_cols] = RobustScaler().fit_transform(df_rob[numeric_cols])
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

X_train[['year', 'odometer']] = X_train[['year', 'odometer']].apply(pd.to_numeric, errors='coerce')
X_test[['year', 'odometer']] = X_test[['year', 'odometer']].apply(pd.to_numeric, errors='coerce')

# Fill NaN values with the median
X_train[['year', 'odometer']] = X_train[['year', 'odometer']].fillna(X_train[['year', 'odometer']].median())
X_test[['year', 'odometer']] = X_test[['year', 'odometer']].fillna(X_test[['year', 'odometer']].median())

# model without rescaling
rf_no_scaling = RandomForestRegressor(n_estimators=100, random_state=42)
rf_no_scaling.fit(X_train[['year', 'odometer']], y_train)
pred_no_scaling = rf_no_scaling.predict(X_test[['year', 'odometer']])

# model using Robust Scaler
rf_rob = RandomForestRegressor(n_estimators=100, random_state=42)
rf_rob.fit(df_rob[['year', 'odometer']], y_train)
pred_rob = rf_rob.predict(X_test[['year', 'odometer']])
```

```
<ipython-input-23-06ef2849ee2a>:13: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  rf_no_scaling.fit(X_train[['year', 'odometer']], y_train)
<ipython-input-23-06ef2849ee2a>:18: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  rf_rob.fit(df_rob[['year', 'odometer']], y_train)
```

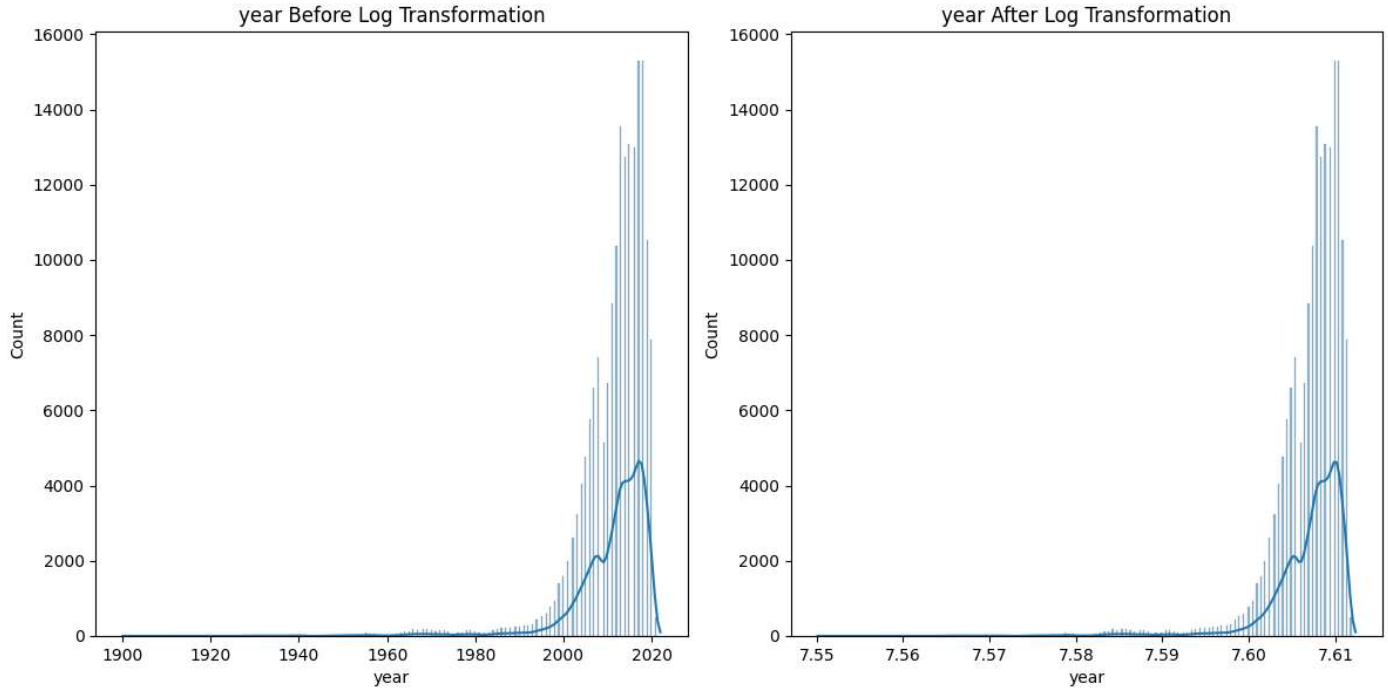
```
print('No Scaling: %.3f' % mean_absolute_error(y_test,pred_no_scaling))
print('Robust Scaler Score: %.3f' % mean_absolute_error(y_test,pred_rob))
```

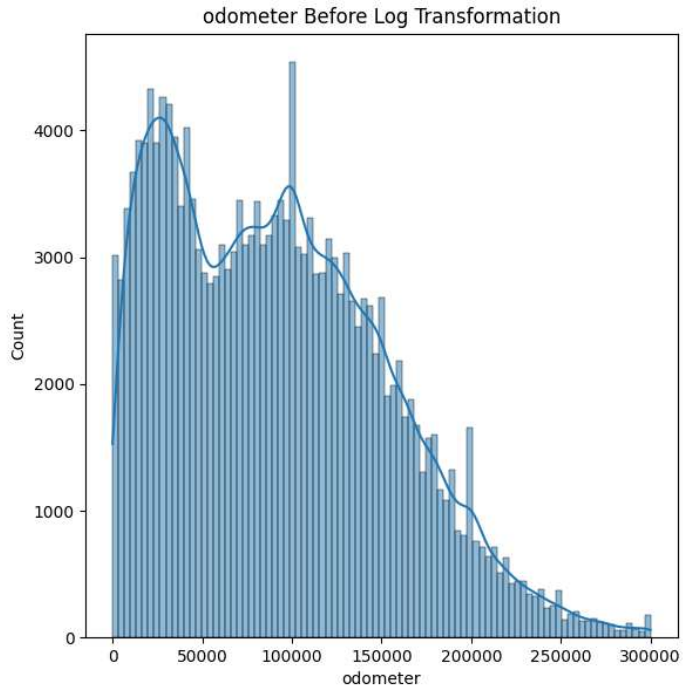
```
No Scaling: 5203.541
Robust Scaler Score: 11019.798
```

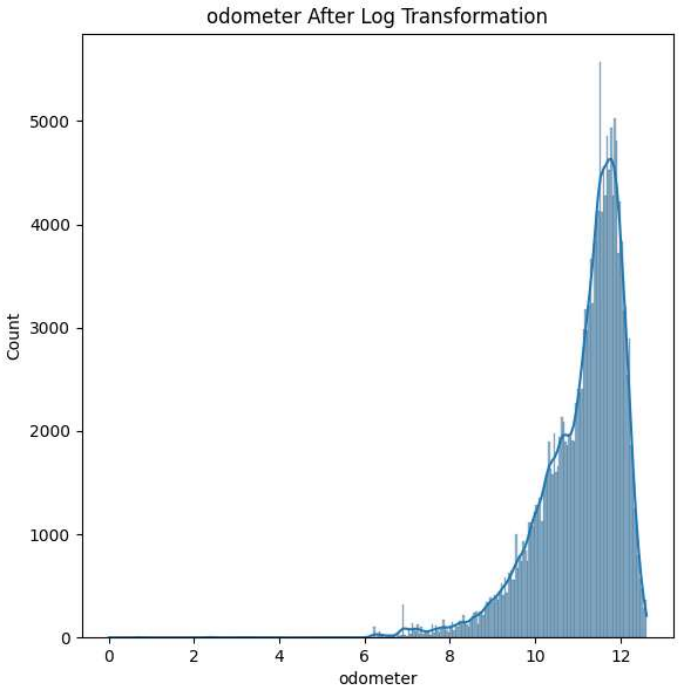
Since our results are better without scaling we are gonna keep it like this. Since we're very worried about the outliers in this dataset and we want to be 100% sure about it before starting to train our model, we want to see how it performs if we apply a log transformation to the skewed variables.

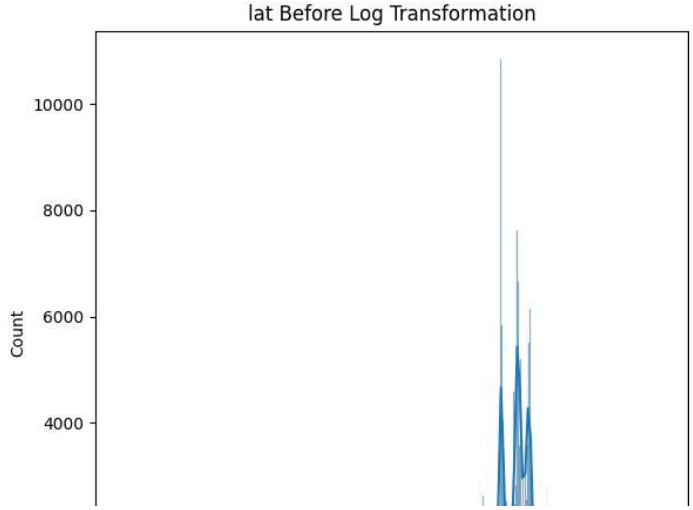
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df_log_transformed = df.copy()

# Apply log transformation to all numeric columns with a small constant (e.g., 1e-8) since the log of 0 will give us a NAN value and the mod
df_log_transformed[num_col] = df_log_transformed[num_col].apply(lambda x: np.log1p(x + 1e-8))

# Plot the distributions before and after log transformation
fig, axes = plt.subplots(nrows=len(num_col), ncols=2, figsize=(12, 6 * len(num_col)))

for i, col in enumerate(num_col):
    # Plot before log transformation
    sns.histplot(df[col], kde=True, ax=axes[i, 0])
    axes[i, 0].set_title(f'{col} Before Log Transformation')

    # Plot after log transformation
    sns.histplot(df_log_transformed[col], kde=True, ax=axes[i, 1])
    axes[i, 1].set_title(f'{col} After Log Transformation')

plt.tight_layout()
plt.show()
```

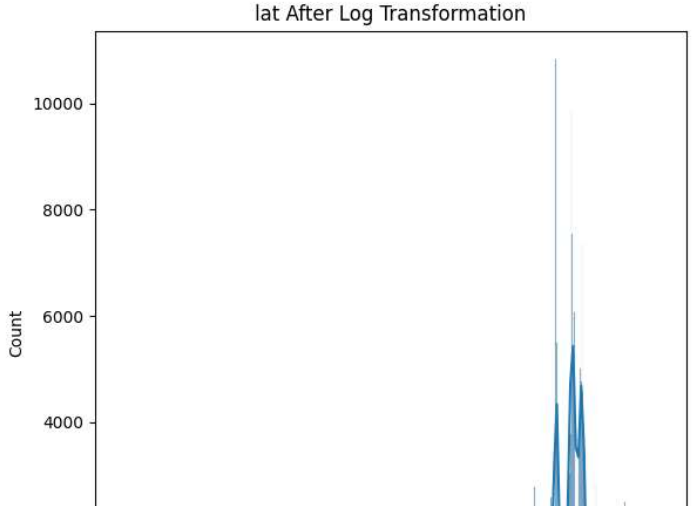
```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:402: RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

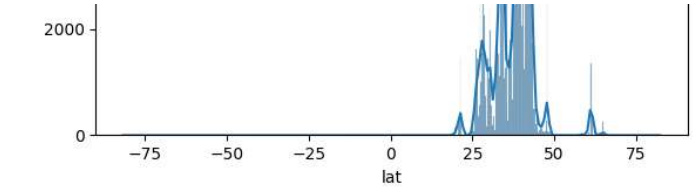


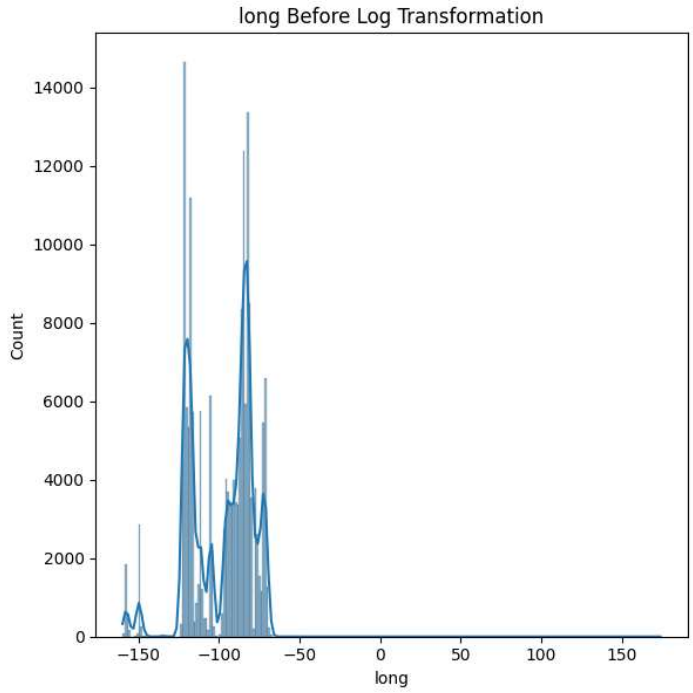


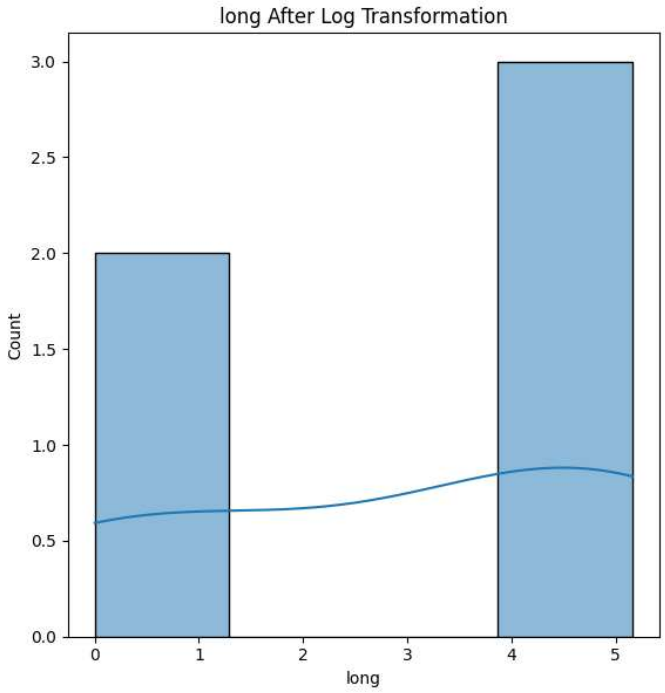













```
from sklearn.ensemble import RandomForestRegressor

# Train Random Forest model on the original dataset
rf_original = RandomForestRegressor(n_estimators=100, random_state=42)
rf_original.fit(X_train[['year', 'odometer']], y_train)
pred_original = rf_original.predict(X_test[['year', 'odometer']])
mae_original = mean_absolute_error(y_test, pred_original)

# Train Random Forest model on the log-transformed dataset
X_train_log = X_train.copy()
X_test_log = X_test.copy()

# Apply log transformation
X_train_log[['odometer']] = X_train_log[['odometer']].apply(np.log1p)
X_test_log[['odometer']] = X_test_log[['odometer']].apply(np.log1p)

rf_log_transformed = RandomForestRegressor(n_estimators=100, random_state=42)
rf_log_transformed.fit(X_train_log[['year', 'odometer']], y_train)
pred_log_transformed = rf_log_transformed.predict(X_test_log[['year', 'odometer']])
mae_log_transformed = mean_absolute_error(y_test, pred_log_transformed)

# Print Mean Absolute Errors
print(f'MAE on Original Dataset: {mae_original:.3f}')
print(f'MAE on Log-Transformed Dataset: {mae_log_transformed:.3f}')
```

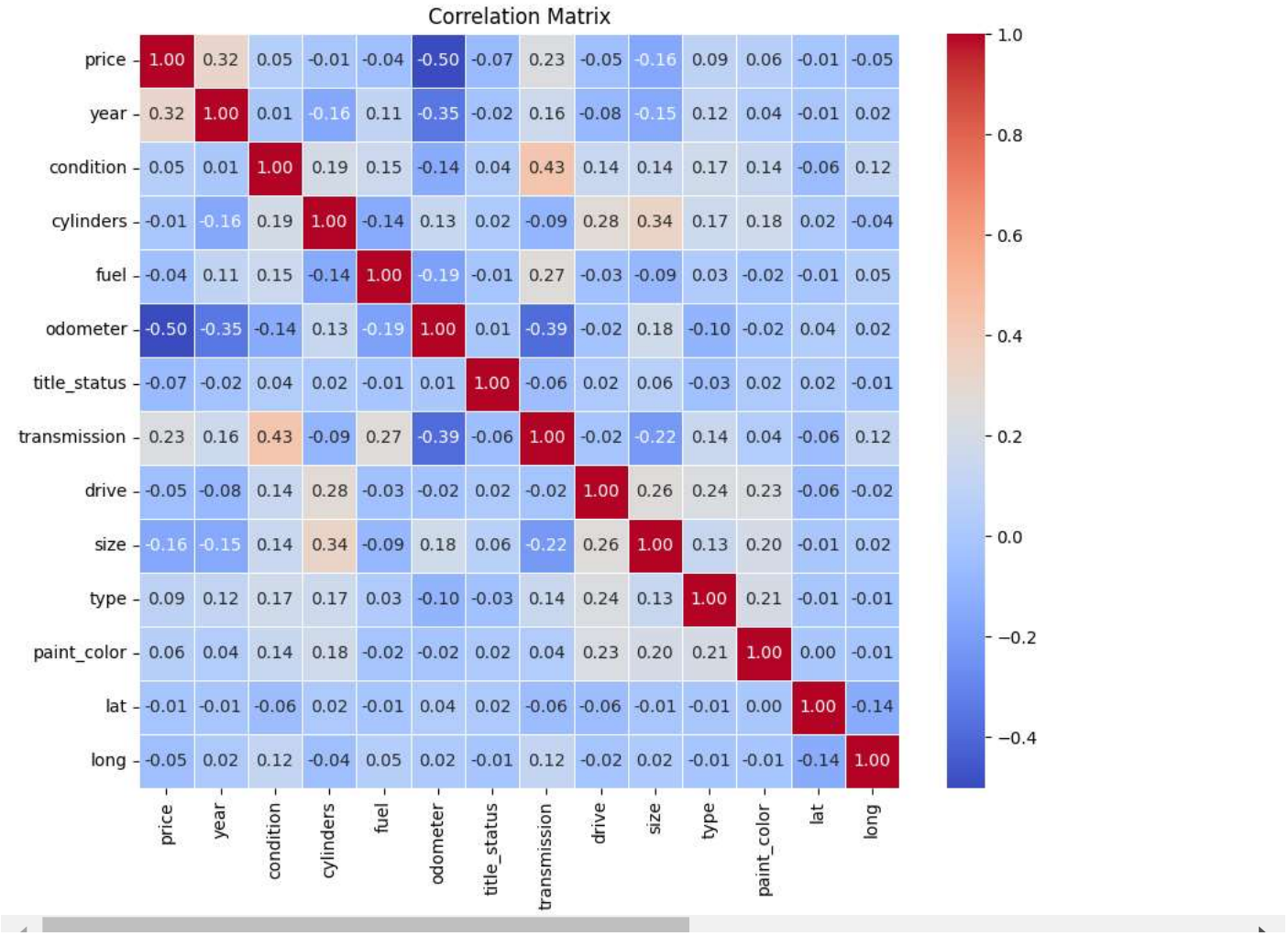
```
<ipython-input-26-491b5f29fc90>:5: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the s
  rf_original.fit(X_train[['year', 'odometer']], y_train)
<ipython-input-26-491b5f29fc90>:18: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  rf_log_transformed.fit(X_train_log[['year', 'odometer']], y_train)
MAE on Original Dataset: 5203.541
MAE on Log-Transformed Dataset: 5206.693
```

The results are more or less the same, since we don't have any improvement we'll keep the original dataset. Just to clarify we used the Random Forest Regressor since it usually outperform all the basic models (only Bootstrap method are better, but it would've taken even more time)

CORRELATION MATRIX

```
from sklearn.preprocessing import LabelEncoder

categorical_cols = df.select_dtypes(include='object').columns
cols_to_label_encode = [col for col in categorical_cols if df[col].nunique() < 15]

df_encoded = df.copy()

label_encoder = LabelEncoder()
for col in cols_to_label_encode:
    df_encoded[col] = label_encoder.fit_transform(df[col])

correlation_matrix = df_encoded.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

```
<ipython-input-27-c5fbf9be1328>:12: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versic
  correlation_matrix = df_encoded.corr()
```



Most significant relations related to *price* are *year* and *odometer* which is not surprising. No other important relation to furtherly analyze.

DUPLICATE VALUES

```
print("Number of duplicated data ", df.duplicated().sum())
```

```
Number of duplicated data  0
```

Despite the value of duplied data seams 0 we found a problem about the VIN variable, let's get into it

```
df["VIN"].value_counts()
```

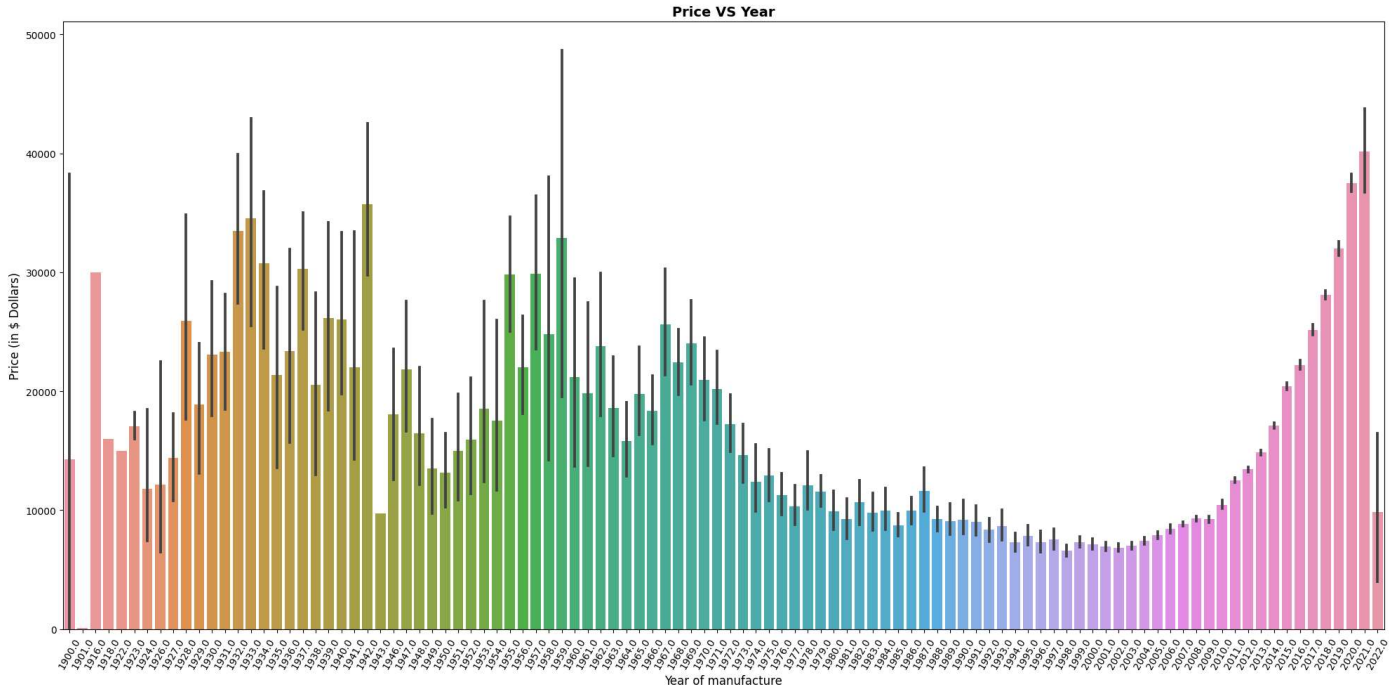
```
                    68515
1FMJU1JT1HEA52352     118
3C6JR6DT3KG560649     112
1GCHTCE37G1186784     101
5TFTX4CN3EX042751      98
                    ...
WBAFU9C51BC783337       1
WBA3C1C5XDF444049       1
JTMWFREVXFJ034466       1
JN1AR5EF9GM290250       1
2B3CJ5DTXBH604375       1
Name: VIN, Length: 57277, dtype: int64
```

As we can see, there are cars that have supposedly been "sold" 118 times. This indicates that the professional seller has posted the same advertisement in each of the regions or states to maximize visibility for the ad. Therefore, using the manufacturer, model, price, year, and VIN as reference values, we will proceed to remove all duplicates we find:

```
df.drop_duplicates(subset=["price", "year", "manufacturer", "model", "VIN"], keep="first", inplace=True)
```

FOCUS ON: PRICE vs YEAR relation

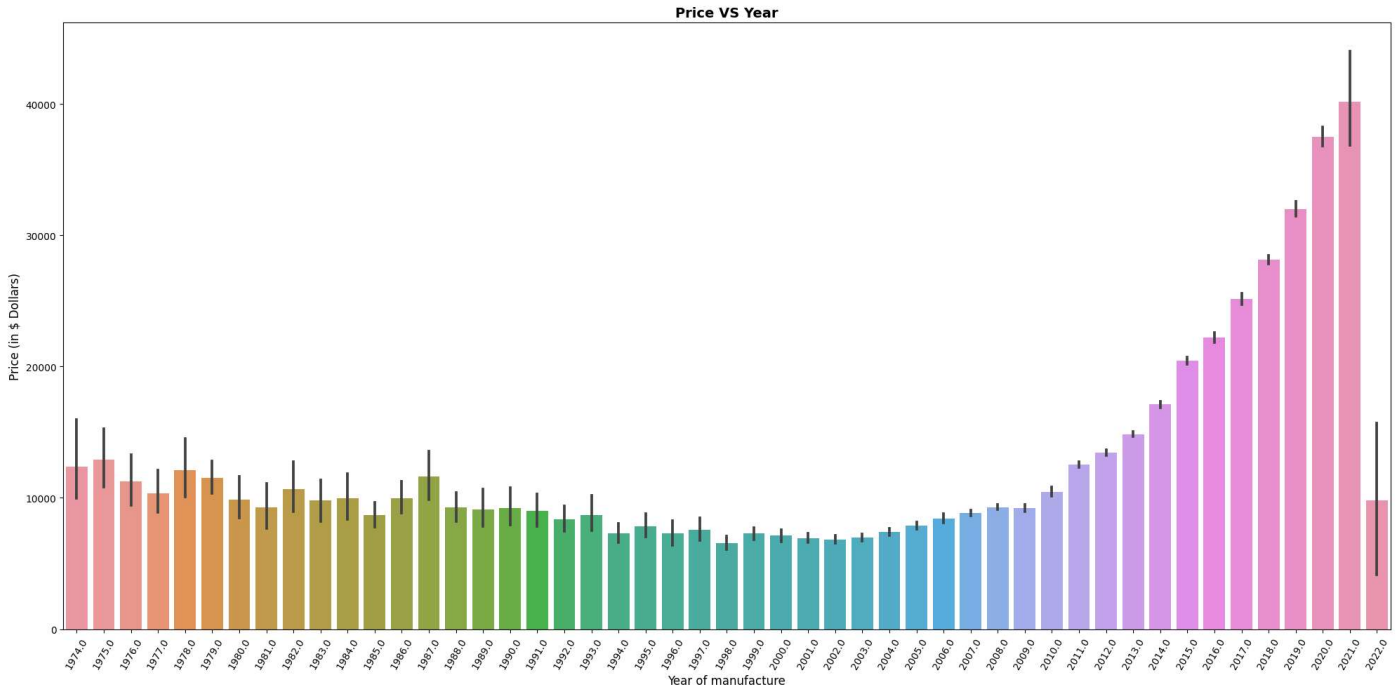
```
sns.set_palette('summer_r')
plt.figure(figsize =(20,10))
sns.barplot(data=df, y='price',x='year')
plt.title("Price VS Year",fontsize=14,fontweight ='bold')
plt.xlabel('Year of manufacture',fontsize = 12)
plt.xticks(rotation=60)
plt.ylabel('Price (in $ Dollars)',fontsize = 12)
plt.tight_layout()
plt.show()
```



Looking at the relation between price and year we can clearly see how the Car Market work. After a certain year (just by looking we would say 1974) we take more concern about the originality, rarity and such characteristic of old cars, that's why we need to divide our analysis in two group: classics and nowadays cars

Post 1976

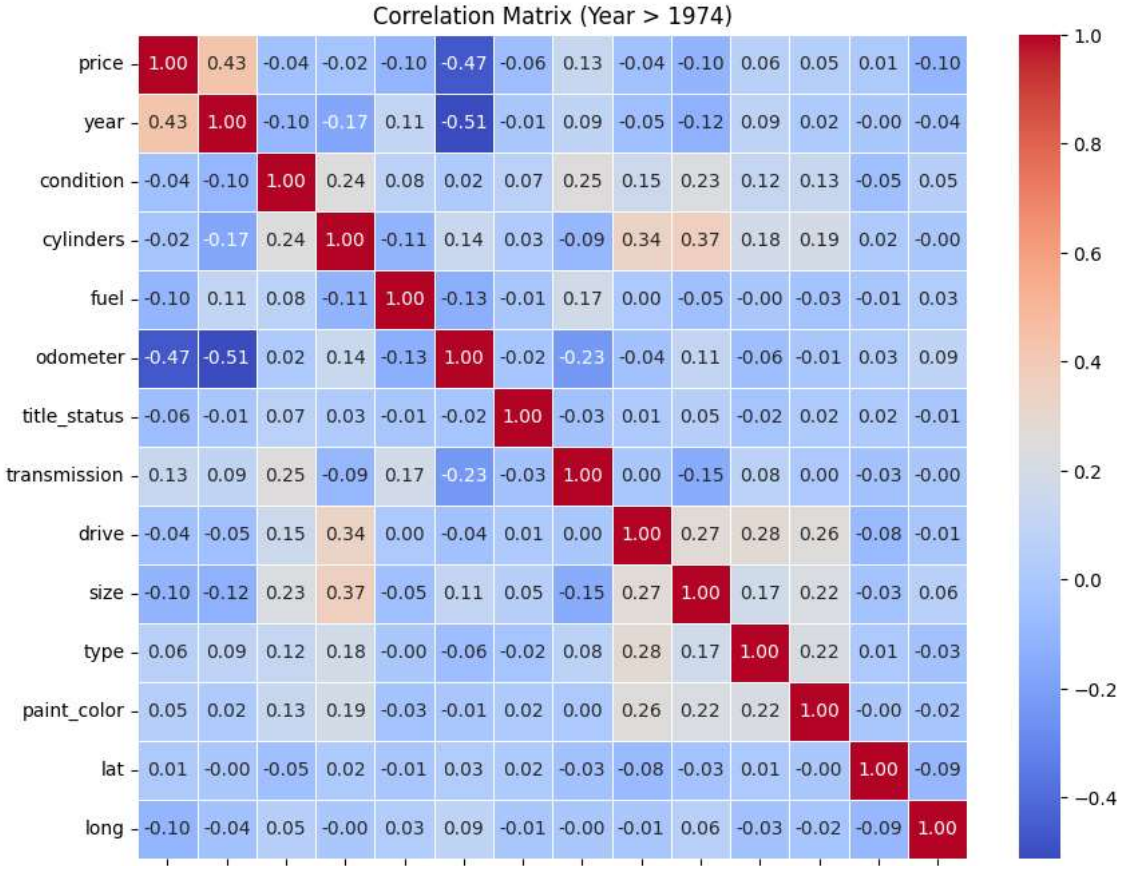
```
sns.set_palette('summer_r')
plt.figure(figsize =(20,10))
sns.barplot(data=df[df["year"] >= 1974], y='price',x='year')
plt.title("Price VS Year",fontsize=14,fontweight ='bold')
plt.xlabel('Year of manufacture',fontsize = 12)
plt.xticks(rotation=60)
plt.ylabel('Price (in $ Dollars)',fontsize = 12)
plt.tight_layout()
plt.show()
```



Let's see the correlation matrix for this subset of the dataset.

```
from sklearn.preprocessing import LabelEncoder
df_filtered = df[df['year'] > 1974]

categorical_cols = df_filtered.select_dtypes(include='object').columns
cols_to_label_encode = [col for col in categorical_cols if df_filtered[col].nunique() < 15]

df_encoded = df_filtered.copy()

label_encoder = LabelEncoder()
for col in cols_to_label_encode:
    df_encoded[col] = label_encoder.fit_transform(df_filtered[col])

correlation_matrix = df_encoded.corr()

# Plot a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix (Year > 1974)")
plt.show()
```

```
<ipython-input-33-e821674314f9>:13: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versic
  correlation_matrix = df_encoded.corr()
```



Pre 1976

```
sns.set_palette('summer_r')
plt.figure(figsize =(20,10))
sns.barplot(data=df[df["year"] <= 1974], y='price',x='year')
plt.title("Price VS Year",fontsize=14,fontweight ='bold')
plt.xlabel('Year of manufacture',fontsize = 12)
plt.xticks(rotation=60)
plt.ylabel('Price (in $ Dollars)',fontsize = 12)
plt.tight_layout()
plt.show()
```


```
df_filtered = df[df['year'] < 1974]

categorical_cols = df_filtered.select_dtypes(include='object').columns
cols_to_label_encode = [col for col in categorical_cols if df_filtered[col].nunique() < 15]

df_encoded = df_filtered.copy()

label_encoder = LabelEncoder()
for col in cols_to_label_encode:
    df_encoded[col] = label_encoder.fit_transform(df_filtered[col])

correlation_matrix = df_encoded.corr()

# Plot a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix (Year > 1974)")
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-94e58c534e40> in <cell line: 8>()
      6 df_encoded = df_filtered.copy()
      7
----> 8 label_encoder = LabelEncoder()
      9 for col in cols_to_label_encode:
     10     df_encoded[col] = label_encoder.fit_transform(df_filtered[col])

NameError: name 'LabelEncoder' is not defined
```

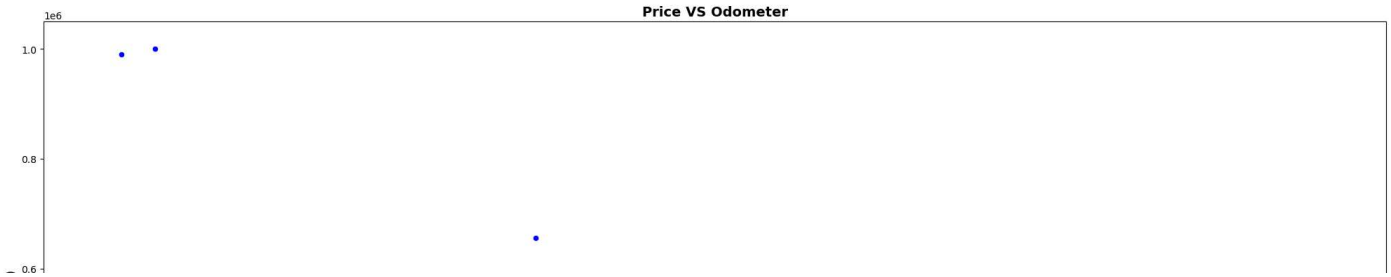
RICERCA SU STACK OVERFLOW

It's even more clear looking at the Correlation Matrix: by dividing our dataset into two epochs we have two complitely different relations, that's why while performing our model we'll remove every rows where the year is fewer than 1974

```
df_filtered = df[df['year'] > 1974]
```

FOCUS ON: Price vs Odometer

```
plt.figure(figsize =(20,10))
sns.scatterplot(data=df_filtered, y='price',x='odometer', color="blue")
plt.title("Price VS Odometer",fontsize=14,fontweight ='bold')
plt.xlabel('Odometer (in Miles)',fontsize = 12)
plt.xticks(rotation=60)
plt.ylabel('Price (in $ Dollars)',fontsize = 12)
plt.tight_layout()
plt.show()
```



Preparation for our Model:

```
# remove columns that do not give us useful info
columns_to_remove = ['id', 'VIN', 'description', 'lat', 'long', 'posting_date']
df_filtered = df_filtered.drop(columns=columns_to_remove)
df = df.drop(columns=columns_to_remove)
```

```
df_filtered
```

	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	drive	size
27	auburn	33590	2014.0	gmc	sierra 1500 crew cab slt	good	8 cylinders	gas	57923.0	clean	other		
28	auburn	22590	2010.0	chevrolet	silverado 1500	good	8 cylinders	gas	71229.0	clean	other		
29	auburn	39590	2020.0	chevrolet	silverado 1500 crew	good	8 cylinders	gas	19160.0	clean	other		
30	auburn	30990	2017.0	toyota	tundra double cab sr	good	8 cylinders	gas	41124.0	clean	other		
31	auburn	15000	2013.0	ford	f-150 xlt	excellent	6 cylinders	gas	128000.0	clean	automatic	rwd	full-size
...	...	...	...	...	...	...	...	...	...	...	...	...	...
202699	grand rapids	14900	2012.0	chevrolet	camaro	excellent	6 cylinders	gas	94000.0	clean	automatic	rwd	
202700	grand rapids	10500	2013.0	bmw	x1 awd 4dr xdrive28i	excellent	4 cylinders	gas	107763.0	clean	automatic	4wd	
202705	grand rapids	16000	2013.0	toyota	venza limited awd	excellent	6 cylinders	gas	91534.0	clean	automatic	fwd	mid-size
202706	grand rapids	2000	1984.0	ford	f-150	good		gas	100000.0	clean	manual		
202707	grand rapids	15155	2011.0	dodge	challenger			gas	129601.0	clean	automatic	rwd	compact

109602 rows × 16 columns

We can see that we reduced the dimension of the problem from 26 columns to 16 colums. Of course it is a partial lose of information, but we believe that not a significant one.

```
data_types = df_filtered.dtypes
print("Data Types:")
print(data_types)
```

```
Data Types:
region          object
price            int64
year           float64
manufacturer    object
model           object
condition       object
cylinders       object
fuel            object
odometer       float64
```

```
title_status     object
transmission     object
drive            object
size             object
type             object
paint_color      object
state            object
dtype: object
```

```
common_columns = df.columns.intersection(df_filtered.columns)  # Get common columns

# Update df to only include common columns
df = df[common_columns]
```

We use LabelEncoder to transform the categorical variables into numerical to run our model. Usually we prefer to use Dummies because the Label Method give us an "order" that in reality doesn't exist but in this case using the Dummy method means having more than 1500 variables so our model would be too heavy.

```
from sklearn.preprocessing import LabelEncoder
numerics = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64']
categorical_columns = []
features = df_filtered.columns.values.tolist()
for col in features:
    if df_filtered[col].dtype in numerics: continue
    categorical_columns.append(col)
for col in categorical_columns:
    if col in df_filtered.columns:
        le = LabelEncoder()
        le.fit(list(df_filtered[col].astype(str).values))
        df_filtered[col] = le.transform(list(df_filtered[col].astype(str).values))
```

```
from sklearn.preprocessing import LabelEncoder
numerics = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64']
categorical_columns = []
features = df.columns.values.tolist()
for col in features:
    if df[col].dtype in numerics: continue
    categorical_columns.append(col)
for col in categorical_columns:
    if col in df.columns:
        le = LabelEncoder()
        le.fit(list(df[col].astype(str).values))
        df[col] = le.transform(list(df[col].astype(str).values))
```

```
df = df.dropna()
df
```

	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	drive	size	type	p
27	8	33590	2014.0	15	14269	3	7	3	57923.0	1	3	0	0	9	
28	8	22590	2010.0	8	14498	3	7	3	71229.0	1	3	0	0	9	
29	8	39590	2020.0	8	14521	3	7	3	19160.0	1	3	0	0	9	
30	8	30990	2017.0	40	16338	3	7	3	41124.0	1	3	0	0	9	
31	8	15000	2013.0	14	8196	1	6	3	128000.0	1	1	3	2	11	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
202699	64	14900	2012.0	8	5203	1	6	3	94000.0	1	1	3	0	3	
202700	64	10500	2013.0	5	16898	1	4	3	107763.0	1	1	1	0	5	
202705	64	16000	2013.0	40	16544	1	6	3	91534.0	1	1	2	3	1	
202706	64	2000	1984.0	14	8043	3	0	3	100000.0	1	2	0	0	0	
202707	64	15155	2011.0	11	5521	0	0	3	129601.0	1	1	3	1	4	

111333 rows × 16 columns

```
nan_counts = df_filtered.isna().sum()
print("NaN Counts:")
print(nan_counts)
```

```
NaN Counts:
region            0
price             0
year              0
manufacturer      0
model             0
condition         0
cylinders         0
fuel              0
odometer        660
title_status      0
transmission      0
drive             0
size              0
type              0
paint_color       0
state             0
dtype: int64
```

```
df_filtered = df_filtered.dropna(subset=['odometer'])
df = df.dropna(subset=['odometer'])

any_nan_after_removal = df_filtered.isna().any().any()
any_nan_after_removal2 = df.isna().any().any()

print("Any NaN Values After Removal:", any_nan_after_removal)
print("Any NaN Values After Removal2:", any_nan_after_removal2)

#I forgot to remove the rows when I removed the outliers from odometer
```

```
Any NaN Values After Removal: False
Any NaN Values After Removal2: True
```

Regression

Types of reggresion examined in this part:

- Linear
- RandomForrestRegressor
- Polynomial
- Support vector machine
- Stochastic gradient descent

1. Linear Regression

First we will try to use the linear regression which is a fast and interpretable way how to perform regression. The disadvantage however is that it assumes a linear relationship between the independent variables and the target variable which can result in inaccurateness.tučný text

```
# import libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

X = df_filtered.drop('price', axis=1)
y = df_filtered['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# use the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# predict
predictions = model.predict(X_test)

# compute metrics
mse = mean_squared_error(y_test, predictions)
r_squared = r2_score(y_test, predictions)
```

```
mae = mean_absolute_error(y_test,predictions)
rmse = np.sqrt(mean_squared_error(y_test,predictions))


print("Mean Squared Error:", mse)
print("R-squared:", r_squared)
print("mean_absolute_error ", mae)
print("Root Mean Squared Error", rmse)
```

```
Mean Squared Error: 204347831.55998525
R-squared: 0.2761941077688832
mean_absolute_error  7630.385498082568
Root Mean Squared Error 14295.028211234327
```

Since R-squared is around 32%, the model does not explain much of the variance in the dependent variable and might not be suitable for making predictions.

2. Random Forrest Regressor

Now we try to use the Random Forest Regressor because it is suitable for predicting continuous numeric values, making it suitable for regression problems such as predicting prices which is our case.

```
# import libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

X = df_filtered.drop('price', axis=1)
y = df_filtered['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# use the random forest regressor
# TODO: USE THE BEST PARAMS OBTAINED FROM PREVIOUS SECTION!!
model = RandomForestRegressor(n_estimators=500, random_state=42, bootstrap=True)
model.fit(X_train, y_train)

# predict
predictions = model.predict(X_test)

# compute metrics
mse = mean_squared_error(y_test, predictions)
r_squared = r2_score(y_test, predictions)
mae = mean_absolute_error(y_test,predictions)
rmse = np.sqrt(mean_squared_error(y_test,predictions))

print("Mean Squared Error:", mse)
print("R-squared:", r_squared)
print("mean_absolute_error ", mae)
print("Root Mean Squared Error", rmse)
```

We can see that now we are getting better results in R-squared metric.

3. Polynomial Regression

If there are nonlinear relationships between features and the target variable by introducing polynomial terms, we might be able to achieve good results with this method.

```
# import libraries
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error

X = df_filtered.drop('price', axis=1)
y = df_filtered['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Degree of the polynomial // TO BE DETERMINED WITH SOME PARAM TUNER (OPTUNA)
degree = 2
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Creating and fitting the Polynomial regression model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# predict
predictions = model.predict(X_test_poly)

# compute metrics
mse_pl = mean_squared_error(y_test, predictions)
r_squared_pl = r2_score(y_test, predictions)
mae_pl = mean_absolute_error(y_test,predictions)
rmse_pl = np.sqrt(mean_squared_error(y_test,predictions))


print("Mean Squared Error:", mse_pl)
print("R-squared:", r_squared_pl)
print("mean_absolute_error ", mae_pl)
print("Root Mean Squared Error", rmse_pl)
```

```
Mean Squared Error: 169825969.84814182
R-squared: 0.39847153409178115
mean_absolute_error  5920.40433775554
Root Mean Squared Error 13031.729349865343
```

We can see that there is an improvement compared to just the Linear regression but the results are not as good as in case of Random Forest Regression.

4. Support Vector Machines (SVM)

SVM Regression can capture nonlinear relationships and is effective in high-dimensional spaces which is our case becuase we have many features. Usually there is a drawback in its long computation run, but the results are often very good.

We use rbf as kernel because it is useful for non-linear relationships. We also need to scale the features so that they contribute equally to the computation.

```
# import libraries
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SVR(kernel='rbf')  # rbf kernel for non-linear relationships
model.fit(X_train_scaled, y_train)

# predict
predictions = model.predict(X_test_scaled)

# evaluate metrics
mse_svm = mean_squared_error(y_test, predictions)
r_squared_svm = r2_score(y_test, predictions)
mae_svm = mean_absolute_error(y_test,predictions)
rmse_svm = np.sqrt(mean_squared_error(y_test,predictions))


print("Mean Squared Error:", mse_svm)
print("R-squared:", r_squared_svm)
print("mean_absolute_error ", mae_svm)
print("Root Mean Squared Error", rmse_svm)
```

```
Mean Squared Error: 273556891.92183137
R-squared: 0.03105362693643321
mean_absolute_error  9098.544764407008
Root Mean Squared Error 16539.555372555555
```

OK, this does not seem good. Let's change the kernel type to poly.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SVR(kernel='poly', degree=3)  # rbf kernel for non-linear relationships
model.fit(X_train_scaled, y_train)

# predict
predictions = model.predict(X_test_scaled)

# evaluate metrics
mse_svm_pl = mean_squared_error(y_test, predictions)
r_squared_svm_pl = r2_score(y_test, predictions)
mae_svm_pl = mean_absolute_error(y_test,predictions)
rmse_svm_pl = np.sqrt(mean_squared_error(y_test,predictions))


print("Mean Squared Error:", mse_svm_pl)
print("R-squared:", r_squared_svm_pl)
print("mean_absolute_error ", mae_svm_pl)
print("Root Mean Squared Error", rmse_svm_pl)
```

```
Mean Squared Error: 274991442.81959677
R-squared: 0.025972406428334982
mean_absolute_error  9159.062673327306
Root Mean Squared Error 16582.865941072934
```

Ok still bad. Let's try linear kernel since the linear regression had a bigger success.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SVR(kernel='linear', C=10)  # rbf kernel for linear relationships
model.fit(X_train_scaled, y_train)

# predict
predictions = model.predict(X_test_scaled)

# evaluate metrics
mse_svm_lm = mean_squared_error(y_test, predictions)
r_squared_svm_lm = r2_score(y_test, predictions)
mae_svm_lm = mean_absolute_error(y_test,predictions)
rmse_svm_lm = np.sqrt(mean_squared_error(y_test,predictions))


print("Mean Squared Error:", mse_svm_lm)
print("R-squared:", r_squared_svm_lm)
print("mean_absolute_error ", mae_svm_lm)
print("Root Mean Squared Error", rmse_svm_lm)
```

```
Mean Squared Error: 214066474.01692292
R-squared: 0.24177039687790625
mean_absolute_error  7351.838316279684
Root Mean Squared Error 14631.010697040821
```

Let's try to tune the 'C' parameter, by default 1.0.

Recap:

- Smaller C value allows for a wider margin => potentially misclassifying some training points.
- Larger C value emphasizes classifying all training data correctly => potentially leading to overfitting.

5. SGDRegressor

We can also try to use SGDRegressor, a linear model that uses stochastic gradient descent for training. Again, scaling is necessary.

```
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# feature scaling (important for SGD)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SGDRegressor(max_iter=100000, random_state=42)
model.fit(X_train_scaled, y_train)

# predict
predictions = model.predict(X_test_scaled)

# evaluate metrics
mse_sdg = mean_squared_error(y_test, predictions)
r_squared_sdg = r2_score(y_test, predictions)
mae_sdg = mean_absolute_error(y_test,predictions)
rmse_sdg = np.sqrt(mean_squared_error(y_test,predictions))


print("Mean Squared Error:", mse_sdg)
print("R-squared:", r_squared_sdg)
print("mean_absolute_error ", mae_sdg)
print("Root Mean Squared Error", rmse_sdg)
```

```
Mean Squared Error: 204687144.30369055
R-squared: 0.27499225227901514
mean_absolute_error  7589.830868477312
Root Mean Squared Error 14306.891496886756
```

Random Forest using Optuna, the same library that we used in our previous research about classification, for tuning the model's hyperparameters.

```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-3.4.0-py3-none-any.whl (409 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 409.6/409.6 kB 6.4 MB/s eta 0:00:00
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.0-py3-none-any.whl (230 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 230.6/230.6 kB 26.3 MB/s eta 0:00:00
Collecting colorlog (from optuna)
  Downloading colorlog-6.7.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (23.2)
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.0.23)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna) (4.66.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna) (6.0.1)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.0-py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.6/78.6 kB 9.9 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna) (4.5.0)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0->optuna) (3.0.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0->optuna) (2.1.3)
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.0 alembic-1.13.0 colorlog-6.7.0 optuna-3.4.0
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
import optuna
X = df_filtered.drop('price', axis=1)
y = df_filtered['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

def objective_rf(trial):
    """Define the objective function for Random Forest"""

    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 5, 30),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 20),
        'max_features': trial.suggest_float ('max_features', 0.1, 1.0),
        'random_state': 42,
        'n_jobs': -1,
    }

    model_rf = RandomForestRegressor(**params)
    model_rf.fit(X_train, y_train)
    y_pred = model_rf.predict(X_test)

    # Calculate R2 score
    r2 = r2_score(y_test, y_pred)

    return r2

# Optimize hyperparameters for Random Forest using Optuna
study_rf = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler(seed=42))
study_rf.optimize(objective_rf, n_trials=50)

# Get the best hyperparameters
best_params_rf = study_rf.best_params

# Print the best hyperparameters and the corresponding score
print("Best Hyperparameters for Random Forest:", best_params_rf)
print("Best Score (R2) for Random Forest:", study_rf.best_value)
```

```
23-12-01 20:18:12,142] A new study created in memory with name: no-name-294e13a0-a85a-49ab-bb58-1af8c924fe52
23-12-01 20:18:57,657] Trial 0 finished with value: 0.4992826670305567 and parameters: {'n_estimators': 437, 'max_depth': 29, 'min_s
23-12-01 20:19:30,437] Trial 1 finished with value: 0.42695496880417283 and parameters: {'n_estimators': 240, 'max_depth': 6, 'min_s
23-12-01 20:19:42,280] Trial 2 finished with value: 0.5158153934046369 and parameters: {'n_estimators': 118, 'max_depth': 30, 'min_s
23-12-01 20:20:13,366] Trial 3 finished with value: 0.4992829752590463 and parameters: {'n_estimators': 265, 'max_depth': 12, 'min_s
23-12-01 20:21:31,945] Trial 4 finished with value: 0.4626566637302022 and parameters: {'n_estimators': 651, 'max_depth': 8, 'min_sa
23-12-01 20:22:12,177] Trial 5 finished with value: 0.44216072623677627 and parameters: {'n_estimators': 807, 'max_depth': 10, 'min_
23-12-01 20:24:48,933] Trial 6 finished with value: 0.47404025400213323 and parameters: {'n_estimators': 647, 'max_depth': 9, 'min_s
23-12-01 20:27:05,934] Trial 7 finished with value: 0.49743526960653484 and parameters: {'n_estimators': 828, 'max_depth': 12, 'min_
23-12-01 20:27:34,045] Trial 8 finished with value: 0.49477436846418044 and parameters: {'n_estimators': 209, 'max_depth': 17, 'min_
23-12-01 20:28:39,792] Trial 9 finished with value: 0.48957394373302476 and parameters: {'n_estimators': 696, 'max_depth': 13, 'min_
23-12-01 20:29:06,402] Trial 10 finished with value: 0.4896347088207328 and parameters: {'n_estimators': 447, 'max_depth': 29, 'min_
23-12-01 20:29:29,223] Trial 11 finished with value: 0.5255481045663737 and parameters: {'n_estimators': 128, 'max_depth': 24, 'min_
23-12-01 20:29:57,805] Trial 12 finished with value: 0.5298283491190232 and parameters: {'n_estimators': 114, 'max_depth': 24, 'min_
23-12-01 20:30:26,379] Trial 13 finished with value: 0.5362601836298577 and parameters: {'n_estimators': 109, 'max_depth': 23, 'min_
23-12-01 20:32:23,728] Trial 14 finished with value: 0.5476234542958052 and parameters: {'n_estimators': 354, 'max_depth': 23, 'min_
23-12-01 20:34:29,215] Trial 15 finished with value: 0.5509702251138534 and parameters: {'n_estimators': 330, 'max_depth': 22, 'min_
23-12-01 20:39:24,408] Trial 16 finished with value: 0.5505492169280405 and parameters: {'n_estimators': 1000, 'max_depth': 19, 'min
23-12-01 20:43:52,437] Trial 17 finished with value: 0.5294346613498169 and parameters: {'n_estimators': 999, 'max_depth': 19, 'min_
23-12-01 20:48:29,525] Trial 18 finished with value: 0.510872915588191 and parameters: {'n_estimators': 1000, 'max_depth': 18, 'min_
23-12-01 20:50:53,604] Trial 19 finished with value: 0.5523378112138374 and parameters: {'n_estimators': 528, 'max_depth': 20, 'min_
23-12-01 20:54:03,421] Trial 20 finished with value: 0.5495250090565429 and parameters: {'n_estimators': 517, 'max_depth': 26, 'min_
23-12-01 20:55:24,311] Trial 21 finished with value: 0.5444372103611792 and parameters: {'n_estimators': 347, 'max_depth': 16, 'min_
23-12-01 20:57:56,745] Trial 22 finished with value: 0.533708380655902 and parameters: {'n_estimators': 519, 'max_depth': 21, 'min_s
23-12-01 21:01:23,504] Trial 23 finished with value: 0.5447055231102642 and parameters: {'n_estimators': 816, 'max_depth': 20, 'min_
23-12-01 21:04:39,014] Trial 24 finished with value: 0.5538226263555679 and parameters: {'n_estimators': 585, 'max_depth': 21, 'min_
23-12-01 21:06:52,534] Trial 25 finished with value: 0.5399569832442781 and parameters: {'n_estimators': 421, 'max_depth': 15, 'min_
23-12-01 21:07:02,775] Trial 26 failed with parameters: {'n_estimators': 597, 'max_depth': 22, 'min_samples_split': 9, 'min_samples_
back (most recent call last):
e "/usr/local/lib/python3.10/dist-packages/optuna/study/_optimize.py", line 200, in _run_trial
alue_or_values = func(trial)
e "<ipython-input-26-d8aafff9825d>", line 25, in objective_rf
odel_rf.fit(X_train, y_train)
e "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py", line 473, in fit
rees = Parallel(
e "/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py", line 63, in __call__
eturn super().__call__(iterable_with_config)
e "/usr/local/lib/python3.10/dist-packages/joblib/parallel.py", line 1952, in __call__
eturn output if self.return_generator else list(output)
e "/usr/local/lib/python3.10/dist-packages/joblib/parallel.py", line 1595, in _get_outputs
ield from self._retrieve()
e "/usr/local/lib/python3.10/dist-packages/joblib/parallel.py", line 1707, in _retrieve
ime.sleep(0.01)
ardInterrupt
23-12-01 21:07:02,778] Trial 26 failed with value None.
---------------------------------------------------------------------------
ardInterrupt                         Traceback (most recent call last)
hon-input-26-d8aafff9825d> in <cell line: 35>()
33 # Optimize hyperparameters for Random Forest using Optuna
34 study_rf = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler(seed=42))
35 study_rf.optimize(objective_rf, n_trials=50)
36
37 # Get the best hyperparameters

---------------------------------------- 10 frames ----------------------------------------
local/lib/python3.10/dist-packages/joblib/parallel.py in _retrieve(self)
05                 (self._jobs[0].get_status(
06                     timeout=self.timeout) == TASK_PENDING)):
07                 time.sleep(0.01)
08                 continue
09

ardInterrupt:
```

```
import optuna
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Assuming df_filtered is our DataFrame and 'price' is the target variable
X = df_filtered.drop('price', axis=1)
y = df_filtered['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

def objective_xg(trial):
    """Define the objective function"""

    params = {
        'booster': trial.suggest_categorical('booster', ['gbtree']),
        'max_depth': trial.suggest_int('max_depth', 5, 30),
        'max_leaves': trial.suggest_int('max_leaves', 8, 64),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.5),
        'n_estimators': trial.suggest_int('n_estimators', 400, 3000),
        'min_child_weight': trial.suggest_int('min_child_weight', 5, 30),
        'subsample': trial.suggest_float('subsample', 0.4, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.1, 0.5),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.5, 1.0),
        'colsample_bylevel': trial.suggest_float('colsample_bylevel', 0.1, 0.8),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.4, 1.0),
        'colsample_bynode': trial.suggest_float('colsample_bynode', 0.2, 0.8),
        "random_state" : trial.suggest_categorical('random_state', [42]),
        'objective': trial.suggest_categorical('objective', ['reg:absoluteerror']),
        "n_jobs" : trial.suggest_categorical('n_jobs', [-1]),
        'verbosity':1,
        'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide']),
    }

    model_xgb = XGBRegressor(**params)
    model_xgb.fit(X_train, y_train)
    y_pred = model_xgb.predict(X_test)

    # Calculate R2 score
    r2 = r2_score(y_test, y_pred)

    return r2

# Optimize hyperparameters using Optuna
study = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler(seed=42))
study.optimize(objective_xg, n_trials=50)

# Get the best hyperparameters
best_params = study.best_params

# Print the best hyperparameters and the corresponding score
print("Best Hyperparameters:", study.best_params)
print("Best Score (R2):", study.best_value)
```

```
:udy created in memory with name: no-name-c26fe8d4-cf94-4da6-a286-10afbfe49122
 finished with value: 0.6066706799911278 and parameters: {'booster': 'gbtree', 'max_depth': 14, 'max_leaves': 62, 'learning_rate': 0.3686
 finished with value: 0.581551887075396 and parameters: {'booster': 'gbtree', 'max_depth': 10, 'max_leaves': 18, 'learning_rate': 0.09986
 finished with value: 0.606739420592971 and parameters: {'booster': 'gbtree', 'max_depth': 10, 'max_leaves': 37, 'learning_rate': 0.30028
 finished with value: 0.5633313784356404 and parameters: {'booster': 'gbtree', 'max_depth': 16, 'max_leaves': 14, 'learning_rate': 0.2526
 finished with value: 0.6249743387776258 and parameters: {'booster': 'gbtree', 'max_depth': 29, 'max_leaves': 59, 'learning_rate': 0.3029
 finished with value: 0.6040502127438923 and parameters: {'booster': 'gbtree', 'max_depth': 19, 'max_leaves': 16, 'learning_rate': 0.4030
 finished with value: 0.6100130965125821 and parameters: {'booster': 'gbtree', 'max_depth': 14, 'max_leaves': 14, 'learning_rate': 0.4329
 finished with value: 0.6192214740999643 and parameters: {'booster': 'gbtree', 'max_depth': 23, 'max_leaves': 51, 'learning_rate': 0.2850
 finished with value: 0.6227322235819099 and parameters: {'booster': 'gbtree', 'max_depth': 28, 'max_leaves': 22, 'learning_rate': 0.2110
 finished with value: 0.6269527032973135 and parameters: {'booster': 'gbtree', 'max_depth': 9, 'max_leaves': 58, 'learning_rate': 0.27427
) finished with value: 0.6181086398983924 and parameters: {'booster': 'gbtree', 'max_depth': 5, 'max_leaves': 40, 'learning_rate': 0.4850
. finished with value: 0.6418417110623178 and parameters: {'booster': 'gbtree', 'max_depth': 29, 'max_leaves': 64, 'learning_rate': 0.177
! finished with value: 0.6179887361807586 and parameters: {'booster': 'gbtree', 'max_depth': 23, 'max_leaves': 50, 'learning_rate': 0.161
3 finished with value: 0.5943526764758161 and parameters: {'booster': 'gbtree', 'max_depth': 5, 'max_leaves': 52, 'learning_rate': 0.0366
4 finished with value: 0.6195286751123075 and parameters: {'booster': 'gbtree', 'max_depth': 22, 'max_leaves': 63, 'learning_rate': 0.167
5 finished with value: 0.6253787218888169 and parameters: {'booster': 'gbtree', 'max_depth': 10, 'max_leaves': 27, 'learning_rate': 0.210
6 finished with value: 0.6199204836082646 and parameters: {'booster': 'gbtree', 'max_depth': 27, 'max_leaves': 44, 'learning_rate': 0.346
7 finished with value: 0.6142382869413227 and parameters: {'booster': 'gbtree', 'max_depth': 19, 'max_leaves': 55, 'learning_rate': 0.114
8 finished with value: 0.6076638573259925 and parameters: {'booster': 'gbtree', 'max_depth': 25, 'max_leaves': 45, 'learning_rate': 0.239
9 finished with value: 0.6223742342709443 and parameters: {'booster': 'gbtree', 'max_depth': 8, 'max_leaves': 56, 'learning_rate': 0.3302
0 finished with value: 0.6115816695188961 and parameters: {'booster': 'gbtree', 'max_depth': 30, 'max_leaves': 30, 'learning_rate': 0.261
. finished with value: 0.6249650519480399 and parameters: {'booster': 'gbtree', 'max_depth': 10, 'max_leaves': 30, 'learning_rate': 0.190
! finished with value: 0.6050274012484 94 and parameters: {'booster': 'gbtree', 'max_depth': 12, 'max_leaves': 8, 'learning_rate': 0.22210
```

```
3 finished with value: 0.6200291888994326 and parameters: {'booster': 'gbtree', 'max_depth': 8, 'max_leaves': 29, 'learning_rate': 0.2067
4 finished with value: 0.62691017058194 and parameters: {'booster': 'gbtree', 'max_depth': 16, 'max_leaves': 64, 'learning_rate': 0.14367
5 finished with value: 0.6160698344235382 and parameters: {'booster': 'gbtree', 'max_depth': 16, 'max_leaves': 64, 'learning_rate': 0.131
5 finished with value: 0.6276430212220172 and parameters: {'booster': 'gbtree', 'max_depth': 21, 'max_leaves': 58, 'learning_rate': 0.072
7 finished with value: 0.6065889678102705 and parameters: {'booster': 'gbtree', 'max_depth': 26, 'max_leaves': 58, 'learning_rate': 0.073
8 finished with value: 0.5892371755388575 and parameters: {'booster': 'gbtree', 'max_depth': 21, 'max_leaves': 46, 'learning_rate': 0.017
9 finished with value: 0.623801544164406 and parameters: {'booster': 'gbtree', 'max_depth': 19, 'max_leaves': 60, 'learning_rate': 0.0774
0 finished with value: 0.6274482849756106 and parameters: {'booster': 'gbtree', 'max_depth': 25, 'max_leaves': 55, 'learning_rate': 0.152
1 finished with value: 0.632216043670591 and parameters: {'booster': 'gbtree', 'max_depth': 25, 'max_leaves': 55, 'learning_rate': 0.1559
2 finished with value: 0.6274716562740614 and parameters: {'booster': 'gbtree', 'max_depth': 25, 'max_leaves': 53, 'learning_rate': 0.179
3 finished with value: 0.6281273645459309 and parameters: {'booster': 'gbtree', 'max_depth': 24, 'max_leaves': 53, 'learning_rate': 0.107
4 finished with value: 0.6275244329421921 and parameters: {'booster': 'gbtree', 'max_depth': 21, 'max_leaves': 49, 'learning_rate': 0.123
5 finished with value: 0.6219668312885163 and parameters: {'booster': 'gbtree', 'max_depth': 30, 'max_leaves': 40, 'learning_rate': 0.097
6 finished with value: 0.5913260598507092 and parameters: {'booster': 'gbtree', 'max_depth': 28, 'max_leaves': 61, 'learning_rate': 0.097
7 finished with value: 0.6259216628109184 and parameters: {'booster': 'gbtree', 'max_depth': 24, 'max_leaves': 48, 'learning_rate': 0.066
8 finished with value: 0.6214160008018017 and parameters: {'booster': 'gbtree', 'max_depth': 28, 'max_leaves': 60, 'learning_rate': 0.137
9 finished with value: 0.6268420059041774 and parameters: {'booster': 'gbtree', 'max_depth': 21, 'max_leaves': 56, 'learning_rate': 0.178
0 finished with value: 0.6061968154608258 and parameters: {'booster': 'gbtree', 'max_depth': 27, 'max_leaves': 41, 'learning_rate': 0.052
1 finished with value: 0.6306720008234439 and parameters: {'booster': 'gbtree', 'max_depth': 21, 'max_leaves': 50, 'learning_rate': 0.128
2 finished with value: 0.62543777937852 and parameters: {'booster': 'gbtree', 'max_depth': 23, 'max_leaves': 54, 'learning_rate': 0.11115
3 finished with value: 0.6285435461221336 and parameters: {'booster': 'gbtree', 'max_depth': 19, 'max_leaves': 57, 'learning_rate': 0.094
4 finished with value: 0.6291957277456766 and parameters: {'booster': 'gbtree', 'max_depth': 18, 'max_leaves': 51, 'learning_rate': 0.149
5 finished with value: 0.633726620816932 and parameters: {'booster': 'gbtree', 'max_depth': 18, 'max_leaves': 49, 'learning_rate': 0.1523
6 finished with value: 0.627995202847097 and parameters: {'booster': 'gbtree', 'max_depth': 14, 'max_leaves': 37, 'learning_rate': 0.1483
7 finished with value: 0.6401925491034439 and parameters: {'booster': 'gbtree', 'max_depth': 17, 'max_leaves': 43, 'learning_rate': 0.161
8 finished with value: 0.6410825100543616 and parameters: {'booster': 'gbtree', 'max_depth': 17, 'max_leaves': 43, 'learning_rate': 0.190
9 finished with value: 0.6358771767343323 and parameters: {'booster': 'gbtree', 'max_depth': 14, 'max_leaves': 42, 'learning_rate': 0.193
;btree', 'max_depth': 29, 'max_leaves': 64, 'learning_rate': 0.1778328369315007, 'n_estimators': 2896, 'min_child_weight': 5, 'subsample'
```

Light GBM using Optuna for tuning hyperparameters

```
import optuna
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Assuming df_filtered is our DataFrame and 'price' is the target variable
X = df_filtered.drop('price', axis=1)
y = df_filtered['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

def objective(trial):
    params = {
        "objective": "regression",
        "metric": "r2",
        "verbosity": -1,
        "boosting_type": "gbdt",
        "num_leaves": trial.suggest_int("num_leaves", 2, 256),
        "learning_rate": trial.suggest_float("learning_rate", 1e-5, 0.1),
        "feature_fraction": trial.suggest_float("feature_fraction", 0.1, 1.0),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.1, 1.0),
        "bagging_freq": trial.suggest_int("bagging_freq", 1, 10),
        "min_child_samples": trial.suggest_int("min_child_samples", 5, 100),
    }

    model = LGBMRegressor(**params)

    # Train the model
    model.fit(X_train, y_train)

    # Predict using the model
    y_pred = model.predict(X_test)

    # Calculate R2 score
    r2 = r2_score(y_test, y_pred)

    return r2

# Optimize hyperparameters using Optuna
study = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler(seed=42))
study.optimize(objective, n_trials=50)

# Get the best hyperparameters
best_params = study.best_params

# Print the best hyperparameters and the corresponding score
print("Best Hyperparameters:", study.best_params)
print("Best Score (R2):", study.best_value)
```

```
[I 2023-12-01 02:05:04,586] A new study created in memory with name: no-name-eaa5acc5-1a75-44c2-bb4a-8d22669c376c
[I 2023-12-01 02:05:07,250] Trial 0 finished with value: 0.6347201790278958 and parameters: {'num_leaves': 97, 'learning_rate': 0.095071
[I 2023-12-01 02:05:10,909] Trial 1 finished with value: 0.5761838260408692 and parameters: {'num_leaves': 16, 'learning_rate': 0.086618
[I 2023-12-01 02:05:19,118] Trial 2 finished with value: 0.49423596942330883 and parameters: {'num_leaves': 214, 'learning_rate': 0.0212
[I 2023-12-01 02:05:22,722] Trial 3 finished with value: 0.5720088890720485 and parameters: {'num_leaves': 112, 'learning_rate': 0.02913
[I 2023-12-01 02:05:26,503] Trial 4 finished with value: 0.6088108347215906 and parameters: {'num_leaves': 118, 'learning_rate': 0.07851
[I 2023-12-01 02:05:31,987] Trial 5 finished with value: 0.348981729909827 and parameters: {'num_leaves': 156, 'learning_rate': 0.017060
[I 2023-12-01 02:05:38,600] Trial 6 finished with value: 0.43591771826718817 and parameters: {'num_leaves': 79, 'learning_rate': 0.00977
[I 2023-12-01 02:05:41,064] Trial 7 finished with value: 0.553784265817664 and parameters: {'num_leaves': 10, 'learning_rate': 0.0909329
[I 2023-12-01 02:05:49,345] Trial 8 finished with value: 0.5407772450531033 and parameters: {'num_leaves': 141, 'learning_rate': 0.01849
[I 2023-12-01 02:05:53,145] Trial 9 finished with value: 0.5859078916971192 and parameters: {'num_leaves': 154, 'learning_rate': 0.09218
[I 2023-12-01 02:05:56,488] Trial 10 finished with value: 0.6387360902421494 and parameters: {'num_leaves': 243, 'learning_rate': 0.0684
[I 2023-12-01 02:05:59,501] Trial 11 finished with value: 0.6378479592209902 and parameters: {'num_leaves': 251, 'learning_rate': 0.0638
[I 2023-12-01 02:06:02,446] Trial 12 finished with value: 0.6263972587827742 and parameters: {'num_leaves': 250, 'learning_rate': 0.0615
[I 2023-12-01 02:06:05,423] Trial 13 finished with value: 0.5944428468020612 and parameters: {'num_leaves': 211, 'learning_rate': 0.0606
[I 2023-12-01 02:06:09,854] Trial 14 finished with value: 0.6179683331873462 and parameters: {'num_leaves': 254, 'learning_rate': 0.0442
[I 2023-12-01 02:06:13,284] Trial 15 finished with value: 0.6152845937910375 and parameters: {'num_leaves': 201, 'learning_rate': 0.0735
[I 2023-12-01 02:06:16,027] Trial 16 finished with value: 0.612485749417909 and parameters: {'num_leaves': 188, 'learning_rate': 0.04903
[I 2023-12-01 02:06:18,925] Trial 17 finished with value: 0.6333759352236079 and parameters: {'num_leaves': 233, 'learning_rate': 0.0665
[I 2023-12-01 02:06:21,890] Trial 18 finished with value: 0.5991645074993268 and parameters: {'num_leaves': 170, 'learning_rate': 0.0388
[I 2023-12-01 02:06:26,063] Trial 19 finished with value: 0.6199915042213981 and parameters: {'num_leaves': 233, 'learning_rate': 0.0554
[I 2023-12-01 02:06:28,798] Trial 20 finished with value: 0.6009211365152904 and parameters: {'num_leaves': 51, 'learning_rate': 0.07269
[I 2023-12-01 02:06:30,997] Trial 21 finished with value: 0.6303596678860135 and parameters: {'num_leaves': 86, 'learning_rate': 0.09597
[I 2023-12-01 02:06:33,169] Trial 22 finished with value: 0.637347420389726 and parameters: {'num_leaves': 76, 'learning_rate': 0.082796
[I 2023-12-01 02:06:34,964] Trial 23 finished with value: 0.6022897389645372 and parameters: {'num_leaves': 45, 'learning_rate': 0.08146
[I 2023-12-01 02:06:37,004] Trial 24 finished with value: 0.613313444368395 and parameters: {'num_leaves': 57, 'learning_rate': 0.068561
[I 2023-12-01 02:06:40,087] Trial 25 finished with value: 0.6475661860618609 and parameters: {'num_leaves': 181, 'learning_rate': 0.0815
[I 2023-12-01 02:06:44,200] Trial 26 finished with value: 0.6379908916892952 and parameters: {'num_leaves': 183, 'learning_rate': 0.0772
[I 2023-12-01 02:06:47,797] Trial 27 finished with value: 0.6379871304313295 and parameters: {'num_leaves': 189, 'learning_rate': 0.0762
[I 2023-12-01 02:06:50,849] Trial 28 finished with value: 0.6394911094921196 and parameters: {'num_leaves': 177, 'learning_rate': 0.0873
```

```
[I 2023-12-01 02:06:53,978] Trial 29 finished with value: 0.6453505372439231 and parameters: {'num_leaves': 227, 'learning_rate': 0.0997
[I 2023-12-01 02:06:57,388] Trial 30 finished with value: 0.6422296695745516 and parameters: {'num_leaves': 217, 'learning_rate': 0.0973
[I 2023-12-01 02:07:01,609] Trial 31 finished with value: 0.6454127750197897 and parameters: {'num_leaves': 209, 'learning_rate': 0.0999
[I 2023-12-01 02:07:05,508] Trial 32 finished with value: 0.6471569315682263 and parameters: {'num_leaves': 219, 'learning_rate': 0.0997
[I 2023-12-01 02:07:08,933] Trial 33 finished with value: 0.6424197615149199 and parameters: {'num_leaves': 225, 'learning_rate': 0.0966
[I 2023-12-01 02:07:12,020] Trial 34 finished with value: 0.6375919077492588 and parameters: {'num_leaves': 204, 'learning_rate': 0.0981
[I 2023-12-01 02:07:15,719] Trial 35 finished with value: 0.6420940282098309 and parameters: {'num_leaves': 197, 'learning_rate': 0.0884
[I 2023-12-01 02:07:20,080] Trial 36 finished with value: 0.6405812168807519 and parameters: {'num_leaves': 160, 'learning_rate': 0.0857
[I 2023-12-01 02:07:23,285] Trial 37 finished with value: 0.6439586786757132 and parameters: {'num_leaves': 223, 'learning_rate': 0.0994
[I 2023-12-01 02:07:26,025] Trial 38 finished with value: 0.6400894428204624 and parameters: {'num_leaves': 129, 'learning_rate': 0.0935
[I 2023-12-01 02:07:29,985] Trial 39 finished with value: 0.6380489824105663 and parameters: {'num_leaves': 237, 'learning_rate': 0.0908
[I 2023-12-01 02:07:34,359] Trial 40 finished with value: 0.6364972665367727 and parameters: {'num_leaves': 210, 'learning_rate': 0.0996
[I 2023-12-01 02:07:38,543] Trial 41 finished with value: 0.6458131674014467 and parameters: {'num_leaves': 223, 'learning_rate': 0.0919
[I 2023-12-01 02:07:41,691] Trial 42 finished with value: 0.6579358013847103 and parameters: {'num_leaves': 225, 'learning_rate': 0.0916
[I 2023-12-01 02:07:44,666] Trial 43 finished with value: 0.6541481256981967 and parameters: {'num_leaves': 193, 'learning_rate': 0.0840
[I 2023-12-01 02:07:47,543] Trial 44 finished with value: 0.6528994621356492 and parameters: {'num_leaves': 167, 'learning_rate': 0.0828
[I 2023-12-01 02:07:51,424] Trial 45 finished with value: 0.6462907775584796 and parameters: {'num_leaves': 143, 'learning_rate': 0.0819
[I 2023-12-01 02:07:55,571] Trial 46 finished with value: 0.6525650593173191 and parameters: {'num_leaves': 170, 'learning_rate': 0.0854
[I 2023-12-01 02:07:58,459] Trial 47 finished with value: 0.6578843868375055 and parameters: {'num_leaves': 167, 'learning_rate': 0.0850
[I 2023-12-01 02:08:01,620] Trial 48 finished with value: 0.6523626177268043 and parameters: {'num_leaves': 167, 'learning_rate': 0.0883
[I 2023-12-01 02:08:04,408] Trial 49 finished with value: 0.6477703841571075 and parameters: {'num_leaves': 144, 'learning_rate': 0.0857
Best Hyperparameters: {'num_leaves': 225, 'learning_rate': 0.09169706300454003, 'feature_fraction': 0.7668823142614607, 'bagging_fractio
Best Score (R2): 0.6579358013847103
```

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor,AdaBoostRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

lr = LinearRegression()
tree = DecisionTreeRegressor()
rf = RandomForestRegressor()
gb = GradientBoostingRegressor()
xgb = XGBRegressor(booster = 'gbtree',
                   max_depth = 29,
                   max_leaves = 64,
                   learning_rate = 0.1778328369315007, n_estimators = 2896,
                   min_child_weight = 5,
                   subsample = 0.6101113784180874,
                   reg_alpha = 0.10189088776007482,
                   reg_lambda = 0.6886799927976344,
                   colsample_bylevel = 0.3478057819644239,
                   colsample_bytree = 0.6280156166165987,
                   colsample_bynode = 0.6908881411335419,
                   random_state = 42, objective = 'reg:absoluteerror', n_jobs = -1, grow_policy = 'lossguide')
ada_boost = AdaBoostRegressor(base_estimator = DecisionTreeRegressor(max_depth = 61),n_estimators = 304, learning_rate = 0.18276027831785724

Models = [lr,tree,rf,gb,xgb,ada_boost]

for model in Models:
    print('Model is: {}'.format(model))
    m = model.fit(X_train,y_train)
    print('Training score : {}'.format(m.score(X_train,y_train)))
    prediction = m.predict(X_test)

    r2score = r2_score(y_test,prediction)
    print('R2 score is : {}'.format(r2score))

    mae = mean_absolute_error(y_test,prediction)
    mse = mean_squared_error(y_test,prediction)
    rmse = np.sqrt(mean_squared_error(y_test,prediction))
    print('MAE : {}'.format(mae))
    print('MSE : {}'.format(mse))
    print('RMSE : {}'.format(rmse))
    print("#" * 30)
```

```
Model is: LinearRegression()
Training score : 0.3297971482795481
R2 score is : 0.31593663592038645
MAE : 7499.037941135838
MSE : 165602218.4826964
RMSE : 12868.652551168534
##############################
Model is: DecisionTreeRegressor()
Training score : 0.9997210860749772
R2 score is : 0.2704983363593646
```

```
MSE : 176602198.31864446
RMSE : 13289.1759834327
##############################
Model is: RandomForestRegressor()
Training score : 0.9463812378399835
R2 score is : 0.5868233505129772
MAE : 3991.4386902957076
MSE : 100024315.54328226
RMSE : 10001.215703267391
##############################
Model is: GradientBoostingRegressor()
Training score : 0.5981416427980499
R2 score is : 0.537963246401209
MAE : 4784.456802197475
MSE : 111852666.63045235
RMSE : 10576.04210612138
##############################
Model is: XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=0.3478057819644239,
             colsample_bynode=0.6908881411335419,
             colsample_bytree=0.6280156166165987, device=None,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=None,
             grow_policy='lossguide', importance_type=None,
             interaction_constraints=None, learning_rate=0.1778328369315007,
             max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=29, max_leaves=64,
             min_child_weight=5, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=2896, n_jobs=-1,
             num_parallel_tree=None, objective='reg:absoluteerror', ...)
Training score : 0.7565264416875656
R2 score is : 0.6418417110623178
MAE : 3384.4341779792285
MSE : 86705136.29369552
RMSE : 9311.559283691187
##############################
Model is: AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=61),
                  learning_rate=0.18276027831785724, n_estimators=304)
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` i
  warnings.warn(
Training score : 0.9997100887467486
R2 score is : 0.6113068997604987
MAE : 3608.093845835029
MSE : 94097189.07426697
RMSE : 9700.370563760282
```

Model is: XGBRegressor(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, device=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=None, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=None, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, multi_strategy=None, n_estimators=None, n_jobs=None, num_parallel_tree=None, random_state=None, ...) Training score : 0.836642238489476 R2 score is : 0.6170597239925779 MAE : 3792.8915473537595 MSE : 92704510.4053031 RMSE : 9628.318150399014