2nd Hand Car Price Prediction

In this project we analyze the 'Used Cars' dataset from kaggle (https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data?select=vehicles.csv).

The goal is to build a model that could estimate the price of second hand cars based on relevant features. We explore the data through cleanig and preprocessing, handling outliers, feature engineering and eventually testing different regression models to get the most accurate predictor.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
matplotlib.rcParams['figure.figsize']= (20,10)
```

In [2]:
```
df1 = pd.read_csv("/Users/marwa/Desktop/2ndHCP/model/vehicles.csv")
```

Data Cleaning

In [4]:
```
df1.head()
```

Out[4]:

	id	url	region	
0	7222695916	https://prescott.craigslist.org/cto/d/prescott...	prescott	https://pres
1	7218891961	https://fayar.craigslist.org/ctd/d/bentonville...	fayetteville	https://f
2	7221797935	https://keys.craigslist.org/cto/d/summerland-k...	florida keys	https://
3	7222270760	https://worcester.craigslist.org/cto/d/west-br...	worcester / central MA	https://worce
4	7210384030	https://greensboro.craigslist.org/cto/d/trinit...	greensboro	https://greens

5 rows × 26 columns

In [5]:
```
df1.shape
```

Out[5]: (426880, 26)

In [9]:
```
df1.describe()
```

Out[9]:

	id	price	year	odometer	county	
count	4.268800e+05	4.268800e+05	425675.000000	4.224800e+05	0.0	42033
mean	7.311487e+09	7.519903e+04	2011.235191	9.804333e+04	NaN	3
std	4.473170e+06	1.218228e+07	9.452120	2.138815e+05	NaN	
min	7.207408e+09	0.000000e+00	1900.000000	0.000000e+00	NaN	-8
25%	7.308143e+09	5.900000e+03	2008.000000	3.770400e+04	NaN	3
50%	7.312621e+09	1.395000e+04	2013.000000	8.554800e+04	NaN	3
75%	7.315254e+09	2.648575e+04	2017.000000	1.335425e+05	NaN	4
max	7.317101e+09	3.736929e+09	2022.000000	1.000000e+07	NaN	8

Drop irrelevant features:

Region can be removed since state wil suffice

Note that county has no values

Descirption and type can be removed since we are relying on manufacturer

In [11]:
```
df2= df1.drop(['id', 'url', 'region_url', 'image_url', 'description', 'la
               'long','region', 'VIN', 'title_status', 'type', 'cylinders
```

Handling missing values

In [13]:
```
#check columns with nan>50%

df2.isnull().mean()*100
```

Out[13]:
```
price            0.000000
year             0.282281
manufacturer     4.133714
model            1.236179
condition       40.785232
fuel             0.705819
odometer         1.030735
transmission     0.598763
drive           30.586347
size            71.767476
paint_color     30.501078
state            0.000000
posting_date     0.015930
dtype: float64
```

In [14]:
```
#since size feature as 71% NaN we will remove this feature
#and since our dataset is large it is efficiet enough to remove all rows
df2= df2.drop('size', axis='columns')
df3 = df2.dropna()
df3.shape
```

Out[14]:
```
(152629, 12)
```

Feature Engineering

Checking categories of categorical variables

In [343…
```
print("\nmanufacturer:\n" , len(df3['manufacturer'].unique()),
     "\nmodel:\n", len(df3['model'].unique()),
      "\ncondition:\n", len(df3['condition'].unique()),
      "\nfuel:\n", len(df3['fuel'].unique()),
      "\n transmission:\n", len(df3['transmission'].unique()),
      "\n drive: \n", len(df3['drive'].unique()),
      "\n paintcolour\n", len(df3['paint_color'].unique()),
      "\n state: \n", len(df3['state'].unique()) )
```

```
manufacturer:
 41
model:
 12963
condition:
 6
fuel:
 5
 transmission:
 3
 drive:
 3
 paintcolour
 12
 state:
 51
```

In [18]:
```
#since model has a lot of different categories it can make interpretabili
#we will remove it since we assume that maufacturer provides enough info

df4=df3.drop(['model'], axis='columns')
```

In [19]:
```
#we will remove 'harley-davidson' from manufacturer as it is a motorcycle

df4= df4[~(df4.manufacturer== 'harley-davidson')]
```

Checking count and reduce if necessary to prevent the dimensionality curse

that arises when performing one-hot-encoding in high dimensional categorical variables

what we will do is check how many samples are present per category for all manufacturer,

paint color and state

In [20]:
```
manufacturer_count= df4.groupby('manufacturer')['manufacturer'].agg('coun
paint_count=df4.groupby('paint_color')['paint_color'].agg('count').sort_v
state_count= df4.groupby('state')['state'].agg('count').sort_values(ascen

print(manufacturer_count,
```

```
      '\n\n ',paint_count ,
      '\n\n', state_count)
```

```
manufacturer
ford             26172
chevrolet        21935
toyota           12902
honda             8599
nissan            7705
jeep              7345
gmc               5786
bmw               5223
dodge             5115
ram               4920
volkswagen        4293
hyundai           4030
mercedes-benz     3853
subaru            3340
kia               3206
lexus             3020
mazda             2382
chrysler          2368
cadillac          2365
buick             2163
acura             1977
lincoln           1828
infiniti          1811
audi              1755
mitsubishi        1450
volvo             1122
pontiac           1066
mini              1035
rover              681
jaguar             585
mercury            563
saturn             526
tesla              452
porsche            434
fiat               316
alfa-romeo         199
ferrari             25
datsun              24
land rover           9
aston-martin         8
Name: manufacturer, dtype: int64

  paint_color
white     37736
black     30835
silver    21870
blue      16865
red       16835
grey      14621
green      4473
brown      4035
custom     2759
yellow     1177
orange      980
purple      402
Name: paint_color, dtype: int64

 state
ca    17824
```

```
fl    10208
ny     8711
tx     7670
oh     7010
mi     6160
pa     5702
nc     5397
wi     5048
ma     3897
tn     3871
va     3665
or     3656
il     3641
co     3558
nj     3528
ia     3455
az     3285
mn     3274
in     2768
ok     2706
ga     2535
ks     2398
sc     2358
id     2241
ct     2062
ky     2056
wa     1867
nm     1698
mo     1677
al     1654
md     1530
vt     1512
ar     1497
mt     1423
nh     1171
ri     1138
me     1121
dc     1033
ak     1022
nv      965
la      912
hi      664
sd      505
de      444
wv      429
ms      402
ne      387
ut      354
wy      269
nd      230
Name: state, dtype: int64
```

In [21]:
```
manufacturer_less_than_100 = manufacturer_count[manufacturer_count<100]
len(df4['manufacturer'].unique())
```

Out[21]: 40

In [22]:
```
# we will group the manufacturers with counts <100 together as 'other'
```

```
df4.manufacturer= df4.manufacturer.apply(lambda x: 'other' if x in manufa

len(df4['manufacturer'].unique())
```

Out[22]: 37

Extracting age of each vehicle

In [24]:
```
# want the feature post_date to be a year only

#first we convert it to a date time for easier manipulation
#using to_datetime from pandas and setting utc=True for correct handling
df4['posting_date'] = pd.to_datetime(df4['posting_date'], utc=True)

df4['posting_date'].dtypes
```

Out[24]: datetime64[ns, UTC]

In [25]:
```
#then we extract the year out of it using dt.year and create a new column
df4['posting_year'] = df4['posting_date'].dt.year

#remove postiing date column
df4= df4.drop('posting_date', axis= 'columns')
df4.year.dtypes
```

Out[25]: dtype('float64')

In [26]:
```
#turn year into int32 to allow subtraction
df4['year']= df4['year'].astype('int32')
```

In [27]:
```
#now we want to create a column of age of the car since purchase

df5 = df4.copy()
df5['vehicle_age'] = df5['posting_year']-df5['year']
```

In [28]:
```
#clearly vehicle_age and year now are highly correlated and the posting_y
#2021 for all its entries which does not provide much information
#we will keep vehicle age as it provides enough info about the 2

df5=df5.drop(['year', 'posting_year'], axis= 'columns')
```

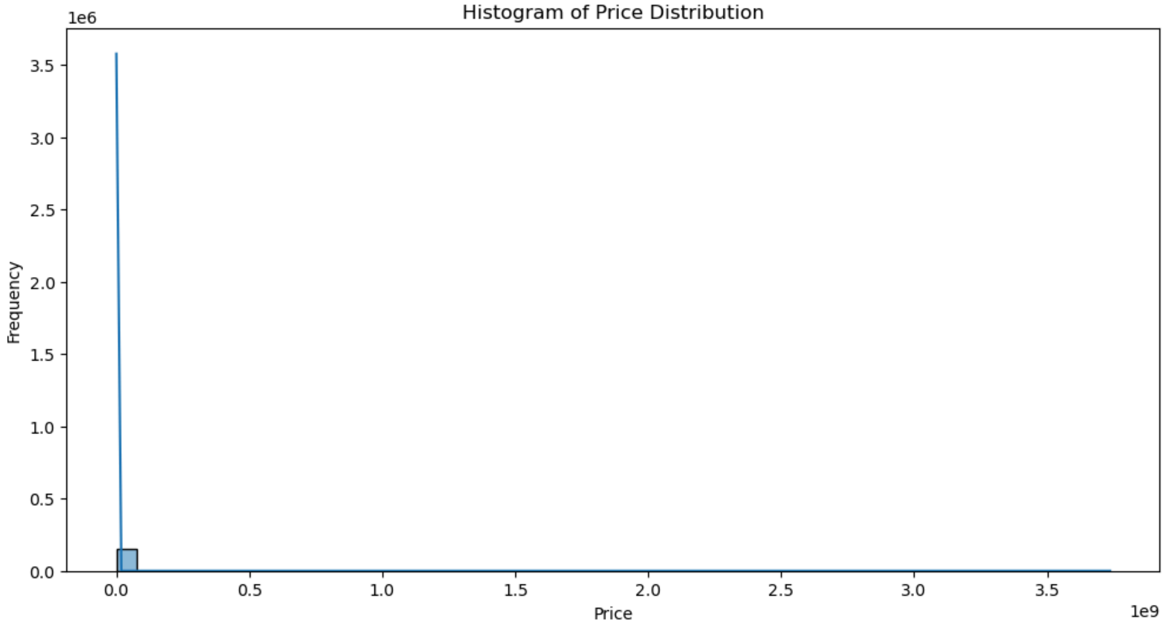
Outlier Removal

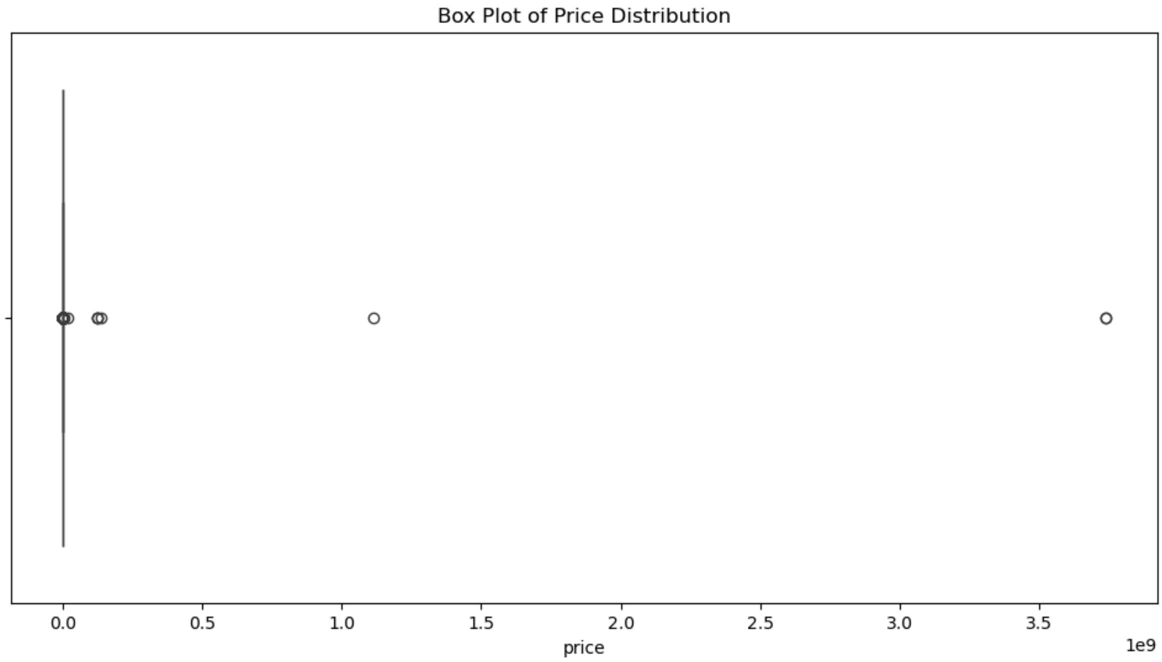
target variable price

In [32]:
```
#Visualising target variable price

plt.figure(figsize=(12, 6))
sns.histplot(df5['price'], bins=50, kde=True)
plt.title('Histogram of Price Distribution')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

```
plt.figure(figsize=(12, 6))
sns.boxplot(x=df5['price'])
plt.title('Box Plot of Price Distribution')
plt.show()
```





In [33]:
```
# check price range

df5.price.describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1])
```

Out[33]:
```
count     1.525880e+05
mean      7.598327e+04
std       1.383998e+07
min       0.000000e+00
25%       6.000000e+03
50%       1.290000e+04
75%       2.399500e+04
85%       3.099000e+04
90%       3.499500e+04
100%      3.736929e+09
max       3.736929e+09
Name: price, dtype: float64
```

mean price= 75,967 std dev= 13,838,120 which is extremely high relative to the mean.

This indicates that there's a wide spread in the price and the possible presence of outliers.

min=0 indicates that there're samples with no price!! this can't be marketable. max= 3.74 billion! is extremely high for a car price! Could be an outlier

First Remove cars with price=0 since they are not marketable and dont provide any insight to our analysis

In [36]:
```
df6= df5[~(df5.price==0)]
df6[df5['price']==0]
```

```
/var/folders/s3/24r6s08x3pg9xyqf7659_57h0000gn/T/ipykernel_88375/52173903
3.py:2: UserWarning: Boolean Series key will be reindexed to match DataFra
me index.
  df6[df5['price']==0]
```

Out[36]:

price	manufacturer	condition	fuel	odometer	transmission	drive	paint_color

since prices vary in different states and the figures are highly

right skewed it is best to perform the IQ method on price per state,

so for every location we will get the bounds of the price in that

location then remove outliers for each location

In [40]:
```
def remove_outliers(df):
    df_out= pd.DataFrame()
    #group by state
    for key, subdf in df.groupby('state'):

        Q1= subdf['price'].quantile(0.25)
        Q3= subdf['price'].quantile(0.75)
        IQR= Q3-Q1

        #define bounds
        lower= Q1 - 1.5 * IQR
        upper= Q3 + 1.5 * IQR
```

```
        #select prices per state within the bounds
        reduced_df= subdf[(subdf.price>= lower) & (subdf.price<= upper) ]
        df_out= pd.concat([df_out, reduced_df], ignore_index=True)

    return df_out

df7=remove_outliers(df6)
df7.shape
```

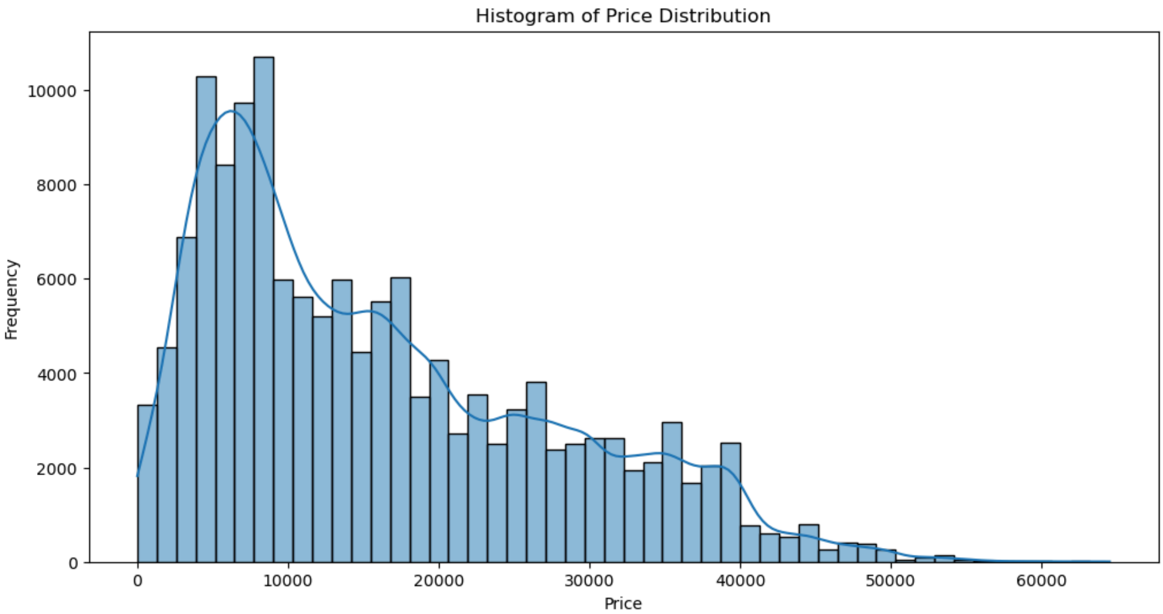
Out[40]: (144078, 10)

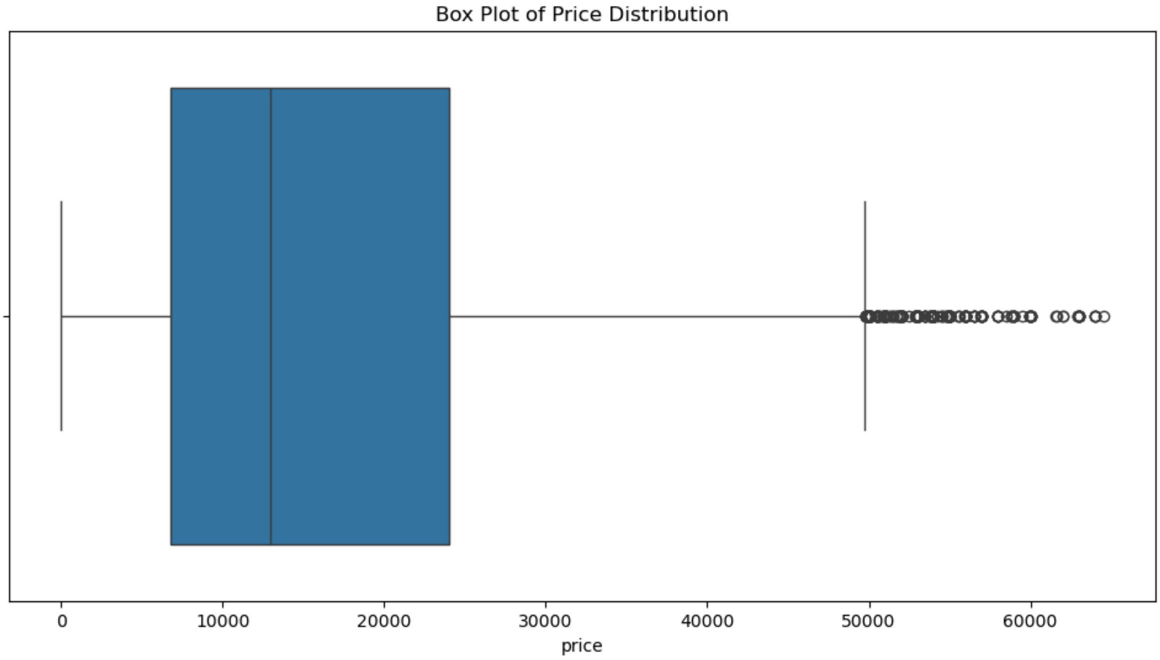
Price target variable

In [42]:
```
# visualise Price again

plt.figure(figsize=(12, 6))
sns.histplot(df7['price'], bins=50, kde=True)
plt.title('Histogram of Price Distribution')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 6))
sns.boxplot(x=df7['price'])
plt.title('Box Plot of Price Distribution')
plt.show()
```





In [43]:
```
df7['price'].describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1])
```

Out[43]:
```
count    144078.000000
mean      16103.381821
std       11566.195793
min           1.000000
25%        6795.000000
50%       12995.000000
75%       23990.000000
85%       29990.000000
90%       33990.000000
100%      64500.000000
max       64500.000000
Name: price, dtype: float64
```

The range looks better now it is still right skewed but that is to be expected in car prices

odometer/ mileage feature

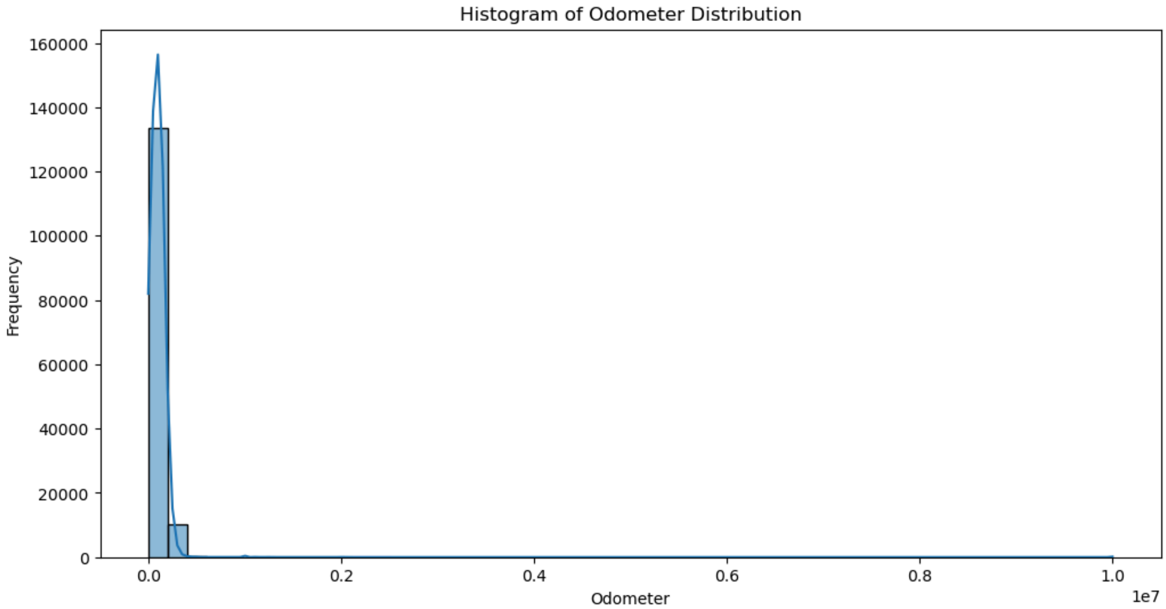
In [46]:
```
df7['odometer'].describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1])
```

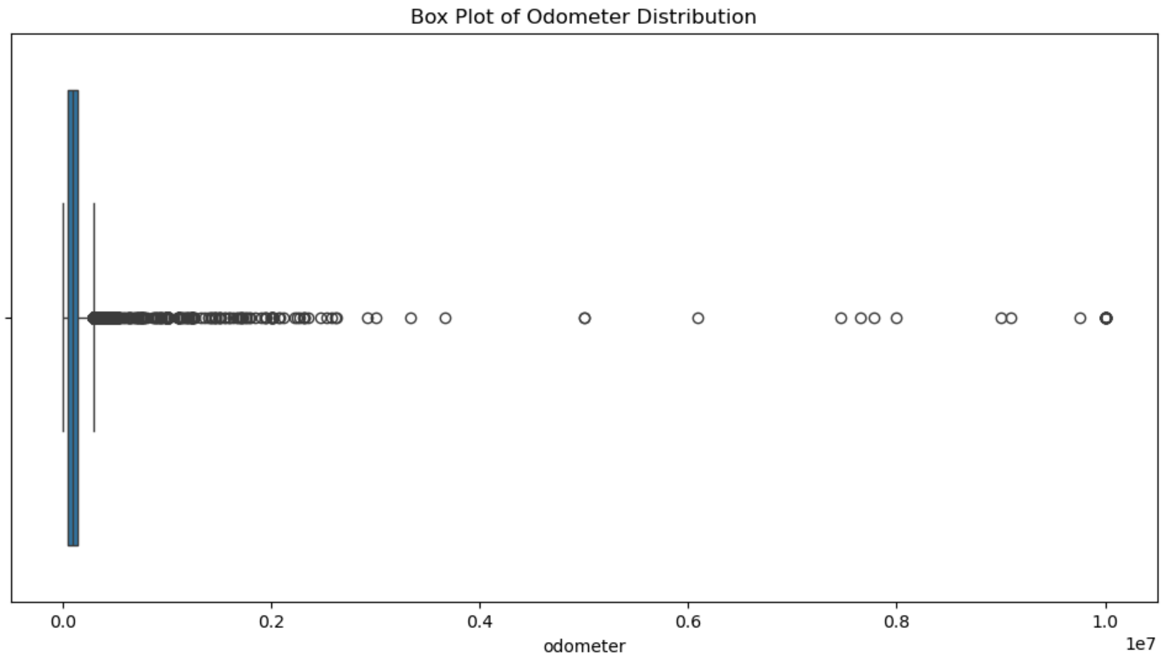
Out[46]:
```
count    1.440780e+05
mean     1.039443e+05
std      1.742401e+05
min      0.000000e+00
25%      4.397425e+04
50%      9.700000e+04
75%      1.435270e+05
85%      1.684450e+05
90%      1.860000e+05
100%     1.000000e+07
max      1.000000e+07
Name: odometer, dtype: float64
```

In [47]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df7['odometer'], bins=50, kde=True)
plt.title('Histogram of Odometer Distribution')
```

```
plt.xlabel('Odometer')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 6))
sns.boxplot(x=df7['odometer'])
plt.title('Box Plot of Odometer Distribution')
plt.show()
```





Notice the distribution is also highly right skewed.

The odometer desciption shows minimum mileage=0 which is unlikely for second hand cars.

Also the maximum= 10 million which is extremely high.

In [49]:
```
# first we check for odometer = 0
df7[(df7['odometer']==0)]
```

Out[49]:

	price	manufacturer	condition	fuel	odometer	transmission	drive	pair
2173	4500	gmc	good	gas	0.0	automatic	rwd	
2269	4250	ford	good	gas	0.0	automatic	rwd	
2355	10500	chevrolet	good	gas	0.0	other	rwd	
2524	7000	ford	good	gas	0.0	automatic	4wd	
5916	9999	jeep	excellent	gas	0.0	automatic	fwd	
...	...	...	...	...	...	...	...	
134351	650	jeep	fair	gas	0.0	automatic	4wd	
137120	20998	mitsubishi	new	gas	0.0	automatic	fwd	
137640	4999	kia	excellent	gas	0.0	automatic	fwd	
139817	6250	chrysler	good	gas	0.0	automatic	rwd	
142597	3500	cadillac	good	gas	0.0	automatic	rwd	

226 rows × 10 columns

In [50]:

```
#even though some of those vehicles with odometer=0 are aged=0, our analy
#hence we will remove all samples with odometer<1000 as anything less tha
#considered a new car
# we will also remove all samples with odometer> 500,000 as they are unma
df8= df7[(df7.odometer>1000) & (df7.odometer<=500000)]

df8.odometer.describe()
```

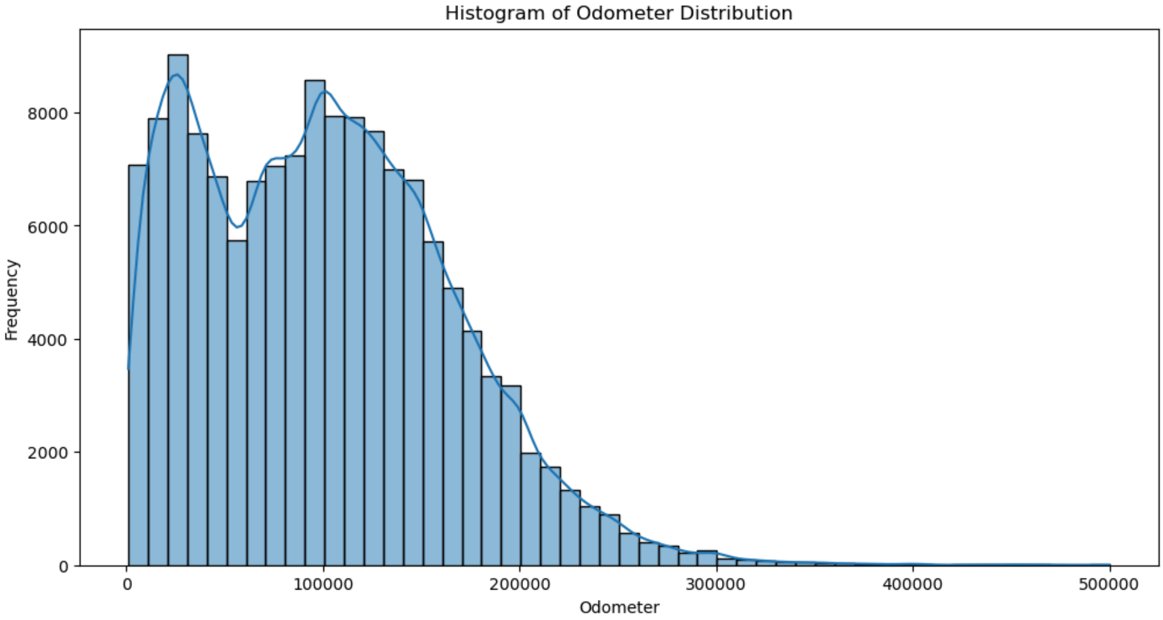
Out[50]:

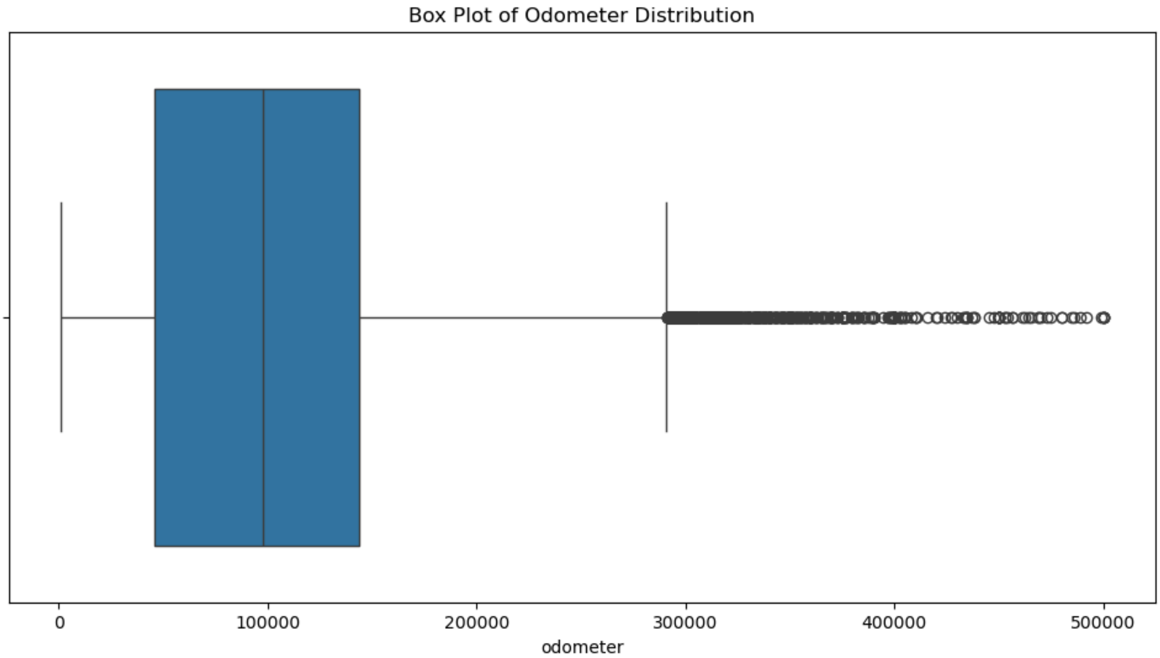
```
count    141853.000000
mean     100955.105074
std       64332.264191
min        1001.000000
25%       45996.000000
50%       98000.000000
75%      144000.000000
max      500000.000000
Name: odometer, dtype: float64
```

In [51]:

```
plt.figure(figsize=(12, 6))
sns.histplot(df8['odometer'], bins=50, kde=True)
plt.title('Histogram of Odometer Distribution')
plt.xlabel('Odometer')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 6))
sns.boxplot(x=df8['odometer'])
plt.title('Box Plot of Odometer Distribution')
plt.show()
```





age of vehicle feature

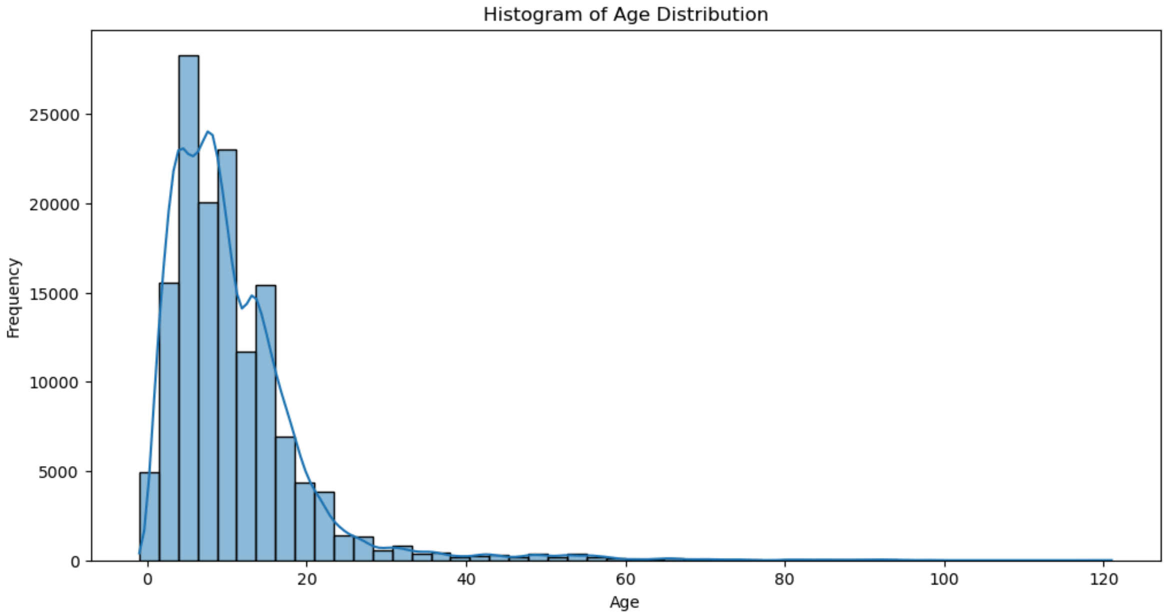
In [53]:
```
df8['vehicle_age'].describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1])
```

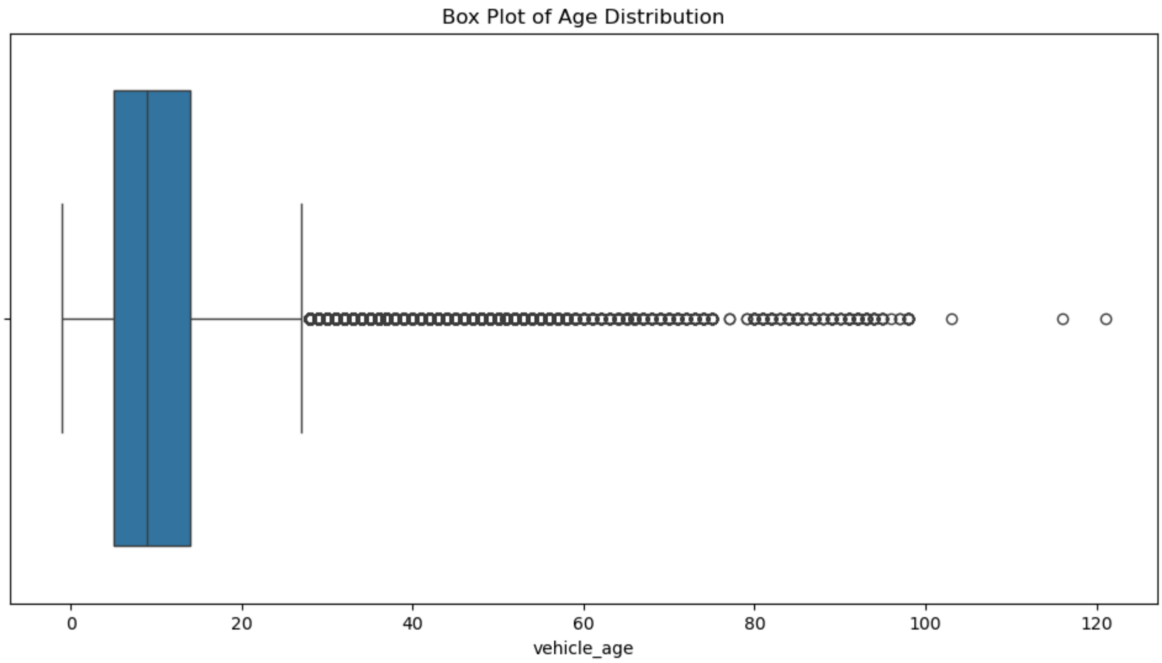
Out[53]:
```
count    141853.000000
mean         10.708184
std           8.918363
min          -1.000000
25%           5.000000
50%           9.000000
75%          14.000000
85%          17.000000
90%          19.000000
100%        121.000000
max         121.000000
Name: vehicle_age, dtype: float64
```

In [54]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df8['vehicle_age'], bins=50, kde=True)
plt.title('Histogram of Age Distribution')
```

```
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 6))
sns.boxplot(x=df8['vehicle_age'])
plt.title('Box Plot of Age Distribution')
plt.show()
```





In [55]:
```
#remove cars with age<=0 as they can be errors or new cars and hence irre
# we will also split our data into vintage-classic cars (age>=30) and non

#non vintage
df_nonvintage= df8[(df8.vehicle_age>0) & (df8.vehicle_age<30)]

#vintage
df_vintage= df8[(df8.vehicle_age>=30) & (df8.vehicle_age<=100)]


df_nonvintage['vehicle_age'].describe(percentiles = [0.25,0.50,0.75,0.85,
```

Out[55]:
```
count    137114.000000
mean          9.555122
std           5.826401
min           1.000000
25%           5.000000
50%           8.000000
75%          13.000000
85%          16.000000
90%          18.000000
100%         29.000000
max          29.000000
Name: vehicle_age, dtype: float64
```

Checking for Duplicates

In [59]:
```
print('Duplicates in nonvintage data:\n ', df_nonvintage.duplicated().sum
      '\nDuplicates in vintage data:\n ', df_vintage.duplicated().sum())
```

```
Duplicates in nonvintage data:
  34681
Duplicates in vintage data:
  249
```

In [60]:
```
#removing duplicates
df_nonvintage= df_nonvintage.drop_duplicates()
df_vintage= df_vintage.drop_duplicates()

print('Duplicates in nonvintage data:\n ', df_nonvintage.duplicated().sum
      '\nDuplicates in vintage data:\n ', df_vintage.duplicated().sum())
```

```
Duplicates in nonvintage data:
  0
Duplicates in vintage data:
  0
```

One hot encoding

In [67]:
```
#non vintage cars

#to ensure our dummies are not boolean (True/False) set dtype=int


df_nonvintage_encoded= pd.get_dummies(df_nonvintage, columns=['manufactur
                                       'paint_color', 'state'],  dtype=int



df_nonvintage_encoded.head()
```

Out[67]:

	price	odometer	vehicle_age	acura	alfa-romeo	audi	bmw	buick	cadillac	chev
0	55000	167000.0	8	0	0	0	0	0	0	
2	16000	53111.0	3	0	0	0	0	0	0	
4	29000	98000.0	11	0	0	0	0	0	0	
5	23000	94252.0	8	0	0	0	0	0	0	
7	13950	193121.0	14	0	0	0	0	0	0	

5 rows × 120 columns

In [68]:
```
#to avoide dummy variable trap, we will drop one of the columns from each
#we will choose columns with the least count

#for state= nd (least count), pain_color=custom, fuel= other, transmissio
#condition= salvage (least count), manufacturer= other
df_nonvintage_encoded.drop(['nd','other','custom','other','other','rwd','
```

In [70]:
```
# vintage cars

df_vintage_encoded= pd.get_dummies(df_vintage, columns=['manufacturer', '
                                    'paint_color', 'state'],   dtype=in

df_vintage_encoded.drop(['nd','other','custom','other','other','fwd','sal
```

Model

In [72]:
```
#split into x(input) and y (target/output)

#nonvintage
x_nonvintage=df_nonvintage_encoded.drop('price', axis='columns')
y_nonvintage= df_nonvintage_encoded.price

#vintage
x_vintage=df_vintage_encoded.drop('price', axis='columns')
y_vintage= df_vintage_encoded.price
```

split to train and test

In [76]:
```
from sklearn.model_selection import train_test_split

#since nonvintage cars dataset contains 102,433 samples an 80/20 split wi
x_nonvintage_train, x_nonvintage_test,y_nonvintage_train, y_nonvintage_te

#similarily the vintage cars dataset contains 4,359 samples an 80/20 spli
x_vintage_train, x_vintage_test,y_vintage_train, y_vintage_test= train_te
```

Linear regression

In [78]:
```
from sklearn.linear_model import LinearRegression

lr_nonvintage= LinearRegression()
lr_vintage= LinearRegression()

#fit for both vintage and nonvintage
lr_nonvintage.fit(x_nonvintage_train, y_nonvintage_train)
lr_vintage.fit(x_vintage_train, y_vintage_train)
```

Out[78]:
```
▾ LinearRegression
LinearRegression()
```

In [79]:
```
#evaluete the nonvintage model

lr_nonvintage.score(x_nonvintage_test, y_nonvintage_test)
```

Out[79]:
```
0.6904030226735173
```

69% is not bad but still need to improve the model

In [81]:
```
#evaluete the vintage model

lr_vintage.score(x_vintage_test, y_vintage_test)
```

Out[81]:
```
0.3941757946715474
```

39% is pretty low which is to be expected froom vintage cars since we have a smaller sample and it is harder to predict the prices of vintage cars

we will stick to modelling non vintage cars

k-fold Cross Validation

In [85]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

#shufflesplit will randomise the sample to ensure each fold will have equ
#of each of the data samples and is not targeted to one area
cv= ShuffleSplit(n_splits= 5, test_size=0.2, random_state=0)

cross_val_score(LinearRegression(), x_nonvintage, y_nonvintage, cv=cv)
```

Out[85]:
```
array([0.68721057, 0.69514816, 0.6933332 , 0.69261292, 0.69100681])
```

In each fold results remain roughly arond 69%

Testing different regression models

In [88]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
```

In [89]:
```
def find_best_model_using_gridsearchcv(X, y):
    algorithms = {
        'linear_regression': {
            'model': LinearRegression(),
            'params': {
                'fit_intercept': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [0.1, 1],  # Adjusted alpha values
                'selection': ['random', 'cyclic']
            }
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion': ['squared_error', 'friedman_mse'],
                'splitter': ['best', 'random']
            }
        },
    }

    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

    for algo_name, config in algorithms.items():
        try:
            gs = GridSearchCV(config['model'], config['params'], cv=cv,
                              return_train_score=False, n_jobs=-1)
            gs.fit(X, y)
            scores.append({
                'model': algo_name,
                'best_score_': gs.best_score_,
                'best_params_': gs.best_params_
            })
        except Exception as e:
            print(f"Error in {algo_name}: {e}")

    return pd.DataFrame(scores)

# Call the function with your data
results = find_best_model_using_gridsearchcv(x_nonvintage, y_nonvintage)
results
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.536e+12, toleran
ce: 9.964e+08
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.463e+12, toleran
ce: 9.993e+08
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.542e+12, toleran
ce: 1.002e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.533e+12, toleran
ce: 1.001e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.539e+12, toleran
ce: 1.002e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.861e+10, toleran
ce: 1.001e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.302e+12, toleran
ce: 9.964e+08
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.285e+11, toleran
ce: 1.002e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.017e+12, toleran
ce: 9.993e+08
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.539e+12, toleran
ce: 9.964e+08
  model = cd_fast.enet_coordinate_descent(
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.535e+12, toleran
ce: 1.001e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 8.169e+11, toleran
ce: 1.002e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.477e+12, toleran
ce: 1.002e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.544e+12, toleran
ce: 1.002e+09
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_coordina
te_descent.py:697: ConvergenceWarning: Objective did not converge. You mig
ht want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 1.741e+12, toleran
ce: 1.249e+09
  model = cd_fast.enet_coordinate_descent(
```

Out[89]:

	model	best_score_	best_params_
0	linear_regression	0.691862	{'fit_intercept': True}
1	lasso	0.691769	{'alpha': 0.1, 'selection': 'cyclic'}
2	decision_tree	0.683318	{'criterion': 'friedman_mse', 'splitter': 'ran...

In [90]:

```
def find_best_model_using_gridsearchcv(X, y):
    algorithms = {

        'random_forest': {
            'model': RandomForestRegressor(),
            'params': {
                'n_estimators': [50,100]




            }
        }
        ,

        'xgboost': {
            'model': xgb.XGBRegressor(),
            'params':{
                'n_estimators': [100, 200],
                'max_depth': [3, 5]
            }
        }
```

```
    }

    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

    for algo_name, config in algorithms.items():
        try:
            gs = GridSearchCV(config['model'], config['params'], cv=cv,
                              return_train_score=False, n_jobs=-1)
            gs.fit(X, y)
            scores.append({
                'model': algo_name,
                'best_score_': gs.best_score_,
                'best_params_': gs.best_params_
            })
        except Exception as e:
            print(f"Error in {algo_name}: {e}")

    return pd.DataFrame(scores)

# Call the function with your data
results = find_best_model_using_gridsearchcv(x_nonvintage, y_nonvintage)
results
```

Out[90]:

	model	best_score_	best_params_
0	random_forest	0.822767	{'n_estimators': 100}
1	xgboost	0.803931	{'max_depth': 5, 'n_estimators': 200}

Random Forest has the best performance with a score of 82%

Random Forest had the best performance

In [93]:

```
rf= RandomForestRegressor(n_jobs=-1)
rf.fit(x_nonvintage_train, y_nonvintage_train)
```

Out[93]:

```
RandomForestRegressor
RandomForestRegressor(n_jobs=-1)
```

In [204…

```
rf.score(x_nonvintage_test, y_nonvintage_test)
```

Out[204…

```
0.8257530771207827
```

In [152…

```
imp=rf.feature_importances_

col= x_nonvintage_train.columns

imp_df= pd.DataFrame([imp], columns=col)
imp_df
```

Out[152…

	odometer	vehicle_age	acura	alfa-romeo	audi	bmw	buick	cadil
0	0.190751	0.418049	0.002148	0.000122	0.001265	0.002702	0.000931	0.0(

1 rows × 112 columns

In [202…

```
#we will sum all the levels within each category. since get_dummies arran
#in alphabetic order


#manufacturer starts with Acura and ends with Volvo
manufacturer= imp_df.loc[:,'acura':'volvo'].sum(axis=1).values[0]

#similarily for the rest of the features
paint_color= imp_df.loc[:,'black':'yellow'].sum(axis=1).values[0]
state= imp_df.loc[:,'ak':'wy'].sum(axis=1).values[0]
transmission= imp_df.loc[:,'automatic':'manual'].sum(axis=1).values[0]
drive= imp_df.loc[:,'4wd':'fwd'].sum(axis=1).values[0]
fuel= imp_df.loc[:,'diesel':'hybrid'].sum(axis=1).values[0]
condition= imp_df.loc[:,'excellent':'new'].sum(axis=1).values[0]

imp= pd.DataFrame({ 'manufacturer': [manufacturer], 'paint_color' : [pain
                   'drive': [drive], 'fuel': [fuel], 'condition': [condit

#add the first two columns from imp_df to include odometer and vehicle_ag
pd.concat([imp, imp_df.iloc[:,:2] ], axis=1)
```

Out[202…

	manufacturer	paint_color	state	transmission	drive	fuel	conditio
0	0.087551	0.030071	0.062539	0.014202	0.124431	0.050871	0.02153

We will remove features with less than 5% significancce: paint_color, transmission, condition

In [207…

```
#first select columns to be removed
trans= x_nonvintage_train.loc[:,'automatic':'manual'].columns.tolist()
color= x_nonvintage_train.loc[:,'black':'yellow'].columns.tolist()
cond= x_nonvintage_train.loc[:,'excellent':'new'].columns.tolist()

#combine those colums
col_remove= trans+color+cond

#modify both train and test sets

x_nonvintage_train_reduced =x_nonvintage_train.drop(columns= col_remove)
x_nonvintage_test_reduced= x_nonvintage_test.drop(columns= col_remove)
y_nonvintage_train_reduced =y_nonvintage_train.drop(columns= col_remove)
y_nonvintage_test_reduced= y_nonvintage_test.drop(columns= col_remove)
```

Implement Random Forest on the reduced data

In [253…

```
rf_reduced= RandomForestRegressor(n_jobs=-1)
rf_reduced.fit(x_nonvintage_train_reduced, y_nonvintage_train_reduced)
```

Out[253…

```
▾ RandomForestRegressor
RandomForestRegressor(n_jobs=-1)
```

In [254…

```
rf_reduced.score(x_nonvintage_test_reduced, y_nonvintage_test_reduced)
```

Out[254…

```
0.8046099029041316
```

Model performance did not improve. try with cross validation

In [216…

```
x_nonvintage_reduced =x_nonvintage.drop(columns= col_remove)
y_nonvintage_reduced =y_nonvintage.drop(columns= col_remove)
```

In [229…

```
cv= ShuffleSplit(n_splits= 5, test_size=0.2, random_state=0)

cross_val_score(RandomForestRegressor(n_jobs=-1), x_nonvintage_reduced, y
```

Out[229…

```
array([0.80388894, 0.80362325, 0.80208947, 0.79945176, 0.80429128])
```

In [230…

```
# try with the second best model: xgboost

cv= ShuffleSplit(n_splits= 5, test_size=0.2, random_state=0)

cross_val_score(xgb.XGBRegressor(n_jobs=-1), x_nonvintage_reduced, y_nonv
```

Out[230…

```
array([0.78439391, 0.78865314, 0.78627592, 0.78047609, 0.78332841])
```

Removing features did not improve our model. We will keep all features

In [233…

```
rf= RandomForestRegressor(n_jobs=-1)
rf.fit(x_nonvintage_train, y_nonvintage_train)
rf.score(x_nonvintage_test, y_nonvintage_test)
```

Out[233…

```
0.8258058553370891
```

In [305…

```
def predict_price(manufacturer, condition, fuel, odometer, transmission,
    # Initialize an array of the size of all the columns in x_nonvintage
    x = np.zeros(len(x_nonvintage.columns))

    #set first two indices to our numeric features
    x[0] = odometer
    x[1] = vehicle_age

    # list of categorical features
    features = [state, manufacturer, condition, fuel, transmission, paint

    # for all categores in a categorical feature, loop over the categorie
    for category in features:

        # Check if the category is in our data x_nonvintage
        if category in x_nonvintage.columns:
```

```
            # Get the index for the one hot encoded category in that data
            idx = np.where(x_nonvintage.columns == category)[0][0]

            #set that category index=1 (categories for that categorical f
            x[idx] = 1

    # prediction
    return rf.predict([x])[0]
```

In [307…

```
predict_price('bmw', 'like new', 'hybrid', 120000, 'automatic','fwd', 'gr
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarni
ng: X does not have valid feature names, but RandomForestRegressor was fit
ted with feature names
  warnings.warn(
```

Out[307… 9723.135

In [309…

```
predict_price('bmw', 'like new', 'hybrid', 100000, 'automatic','fwd', 'gr
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarni
ng: X does not have valid feature names, but RandomForestRegressor was fit
ted with feature names
  warnings.warn(
```

Out[309… 15187.78

In [311…

```
predict_price('bmw', 'good', 'hybrid', 100000, 'hybrid','fwd', 'grey','ny
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493: UserWarni
ng: X does not have valid feature names, but RandomForestRegressor was fit
ted with feature names
  warnings.warn(
```

Out[311… 16578.1

In [320…

```
import pickle
with open('/Users/marwa/Desktop/2ndHCP/model/rf_model', 'wb') as f:
    pickle.dump(rf, f)
```

In [321…

```
import json
columns = {
    'data_columns' : [col.lower() for col in x_nonvintage.columns]
}

with open('/Users/marwa/Desktop/2ndHCP/model/columns', 'w') as f:
    f.write(json.dumps(columns))
```