CAR DETAILS V3

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import re
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv("car details v3.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	

In [4]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [5]:
```
1 df.describe()
```

Out[5]:

	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

In [6]:
```
1 df.shape
```

Out[6]: (8128, 13)

In [7]:
```
1 df.describe(include="all")
```

Out[7]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owne
count	8128	8128.000000	8.128000e+03	8.128000e+03	8128	8128	8128	812
unique	2058	NaN	NaN	NaN	4	3	2	
top	Maruti Swift Dzire VDI	NaN	NaN	NaN	Diesel	Individual	Manual	Firs Owne
freq	129	NaN	NaN	NaN	4402	6766	7078	528
mean	NaN	2013.804011	6.382718e+05	6.981951e+04	NaN	NaN	NaN	Nal
std	NaN	4.044249	8.062534e+05	5.655055e+04	NaN	NaN	NaN	Nal
min	NaN	1983.000000	2.999900e+04	1.000000e+00	NaN	NaN	NaN	Nal
25%	NaN	2011.000000	2.549990e+05	3.500000e+04	NaN	NaN	NaN	Nal
50%	NaN	2015.000000	4.500000e+05	6.000000e+04	NaN	NaN	NaN	Nal
75%	NaN	2017.000000	6.750000e+05	9.800000e+04	NaN	NaN	NaN	Nal
max	NaN	2020.000000	1.000000e+07	2.360457e+06	NaN	NaN	NaN	Nal

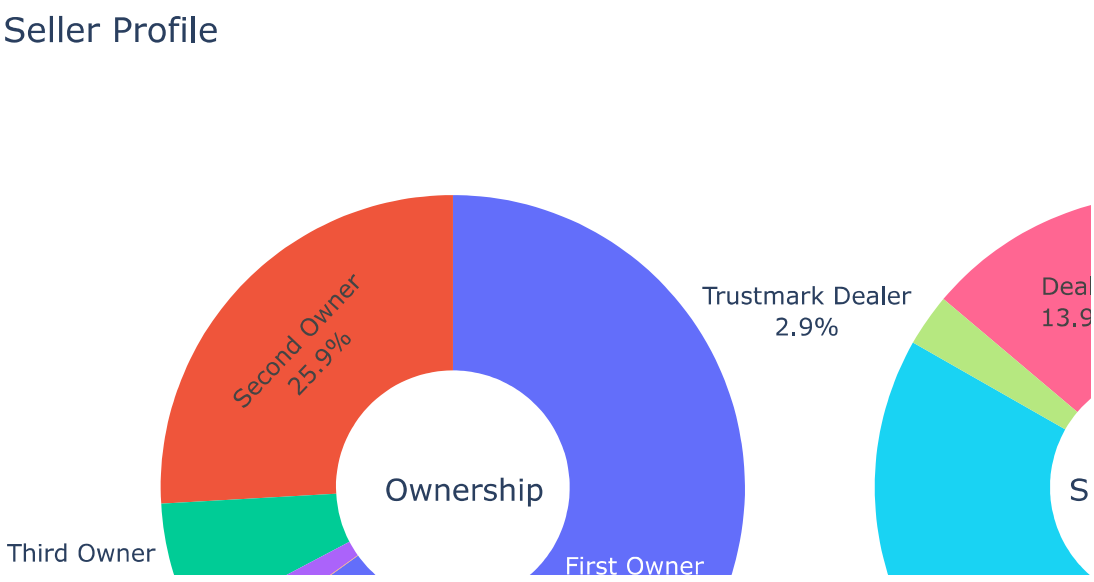
In [8]:
```
1 df.isna().sum()
```

Out[8]:
```
name              0
year              0
selling_price     0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         221
engine          221
max_power       215
torque          222
seats           221
dtype: int64
```

EDA

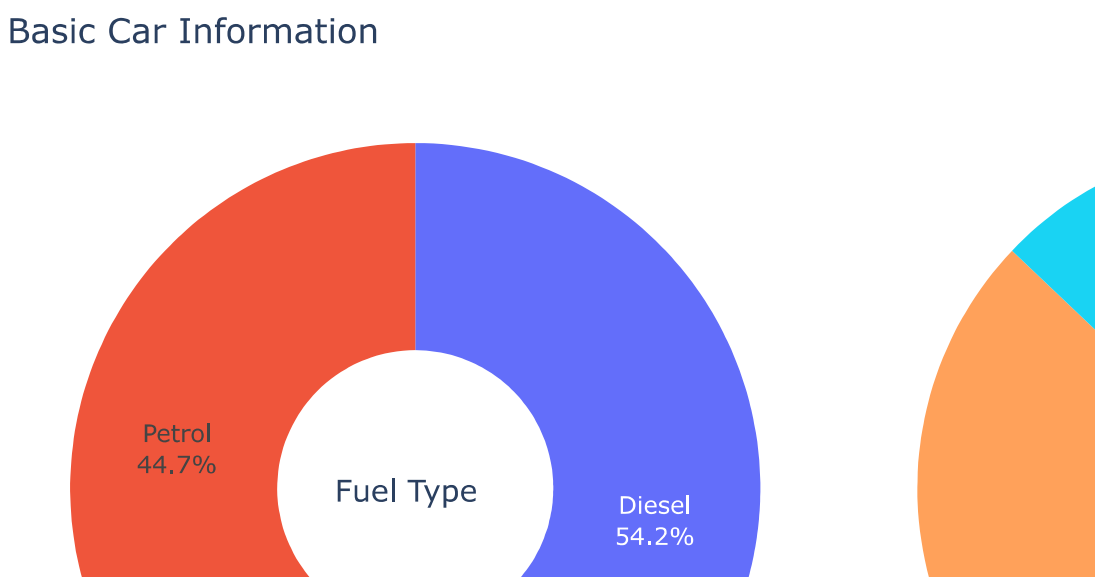
In [9]:

```
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'d
fig.add_trace(go.Pie(labels=df['owner'], name="Ownership", textinfo='label
              1, 1)
fig.add_trace(go.Pie(labels=df['seller_type'], name="Seller Type",textinf
              1, 2)

fig.update_traces(hole=.4, hoverinfo="label+percent+name")

fig.update_layout(
    title_text="Seller Profile",
    annotations=[dict(text='Ownership', x=0.17, y=0.5, font_size=15, showa
                 dict(text='Seller Type', x=0.83, y=0.5, font_size=15, sh
fig.show()
```

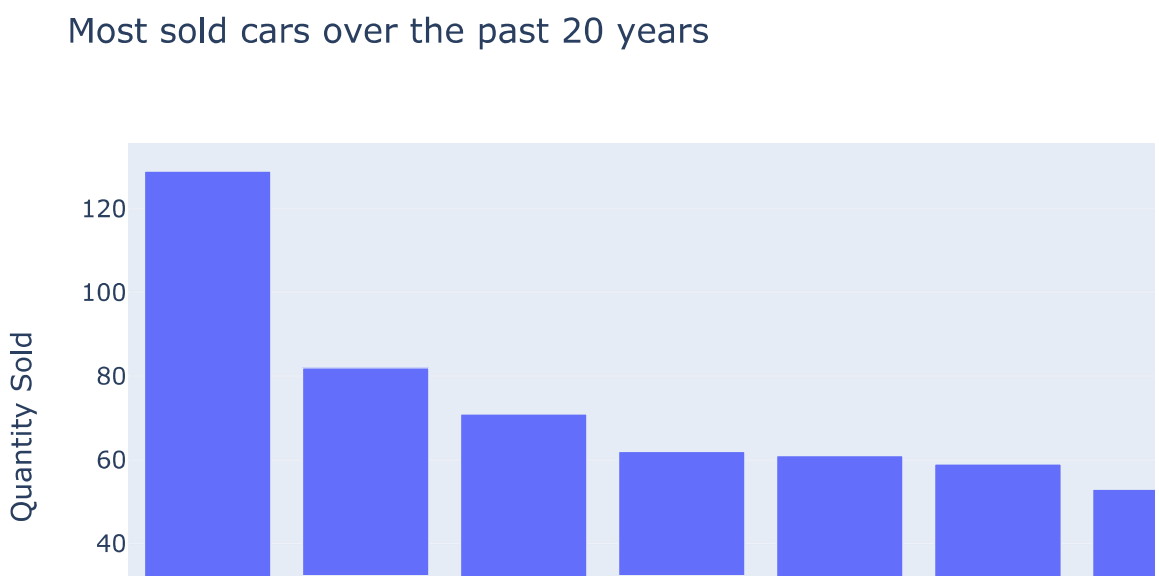


In [10]:

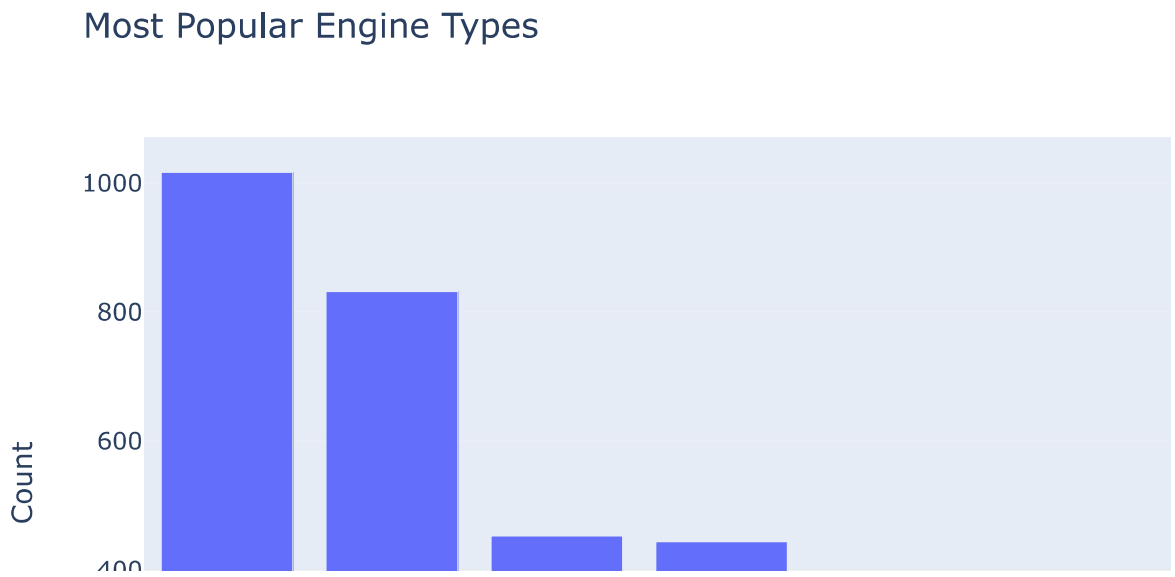
```
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'d
fig.add_trace(go.Pie(labels=df['fuel'], name="Fuel Type",textinfo='label+
              1, 1)
fig.add_trace(go.Pie(labels=df['transmission'], name="Transmission", text
              1, 2)

fig.update_traces(hole=.4, hoverinfo="label+percent+name")

fig.update_layout(
    title_text="Basic Car Information",

    annotations=[dict(text='Fuel Type', x=0.17, y=0.5, font_size=15, show
                 dict(text='Transmission', x=0.83, y=0.5, font_size=15, sl
fig.show()
```



In [11]:
```
most_sold = df.name.value_counts()[:10]
px.bar(data_frame = most_sold, x = most_sold.index, y = most_sold, labels=
       title = 'Most sold cars over the past 20 years')
```

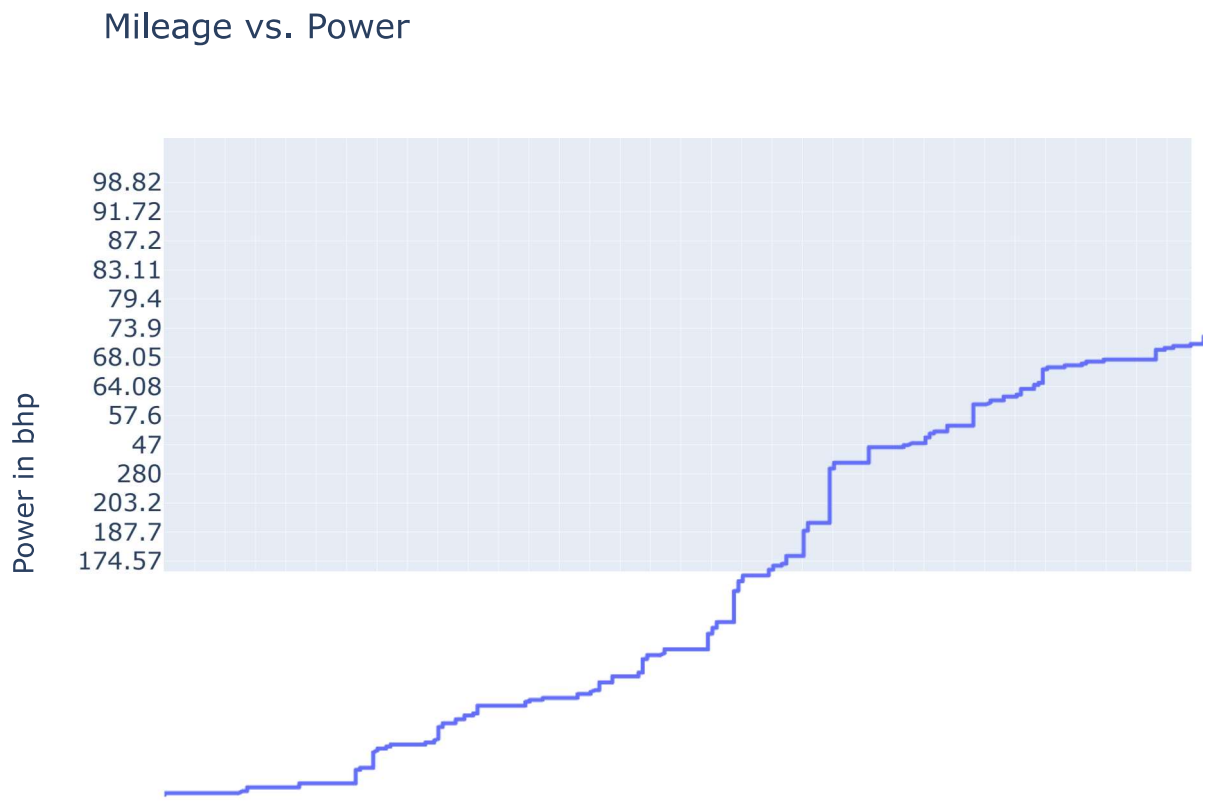


In [12]:
```
1 engine = df.engine.value_counts()[:10]
2 px.bar(x = engine.index, y = engine, labels = {'x': 'Engine Type', 'y': '
```

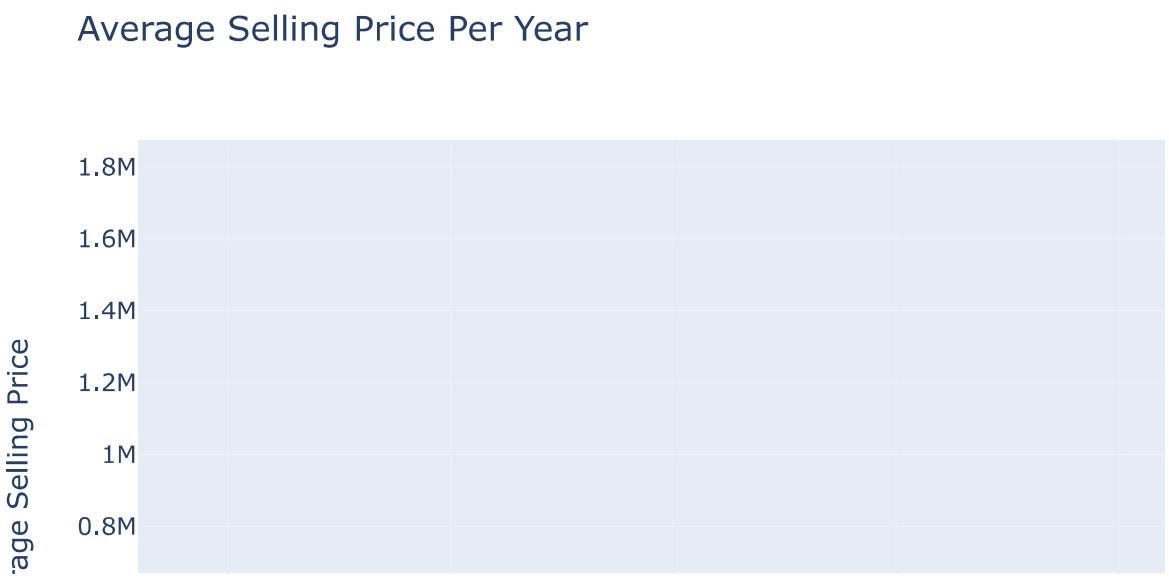


In [13]:

```
power = []
mileage = []

for i in range(0, 8128):
    temp = str(df['mileage'][i])
    temp = re.sub('[^0-9.]', '', temp)
    mileage.append(temp)
while('' in mileage) :
    mileage.remove('')
    mileage.sort()

for i in range(0, 8128):
    temp = str(df['max_power'][i])
    temp = re.sub('[^0-9.]', '', temp)
    power.append(temp)
while('' in power) :
    power.remove('')
    power.sort()

power = power[:len(power)-5]
px.line(x = mileage, y = power, title = "Mileage vs. Power", labels = {'x
```



In [14]:
```
data = df.groupby(['year']).mean()
px.line(data_frame = data, x = data.index, y = 'selling_price', labels =
       title = 'Average Selling Price Per Year')
```



In [15]:
```
px.line(data_frame = data, x = data.index, y = 'km_driven', labels = {'ye
       title = 'Average Distance Travelled Per Year')
```

In [ ]:
```
```

Feature Engineering

In [16]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
df['fuel'] = labelEncoder.fit_transform(df['fuel'])
df['transmission'] = labelEncoder.fit_transform(df['transmission'])
df['owner'] = labelEncoder.fit_transform(df['owner'])
df['seller_type'] = labelEncoder.fit_transform(df['seller_type'])
```

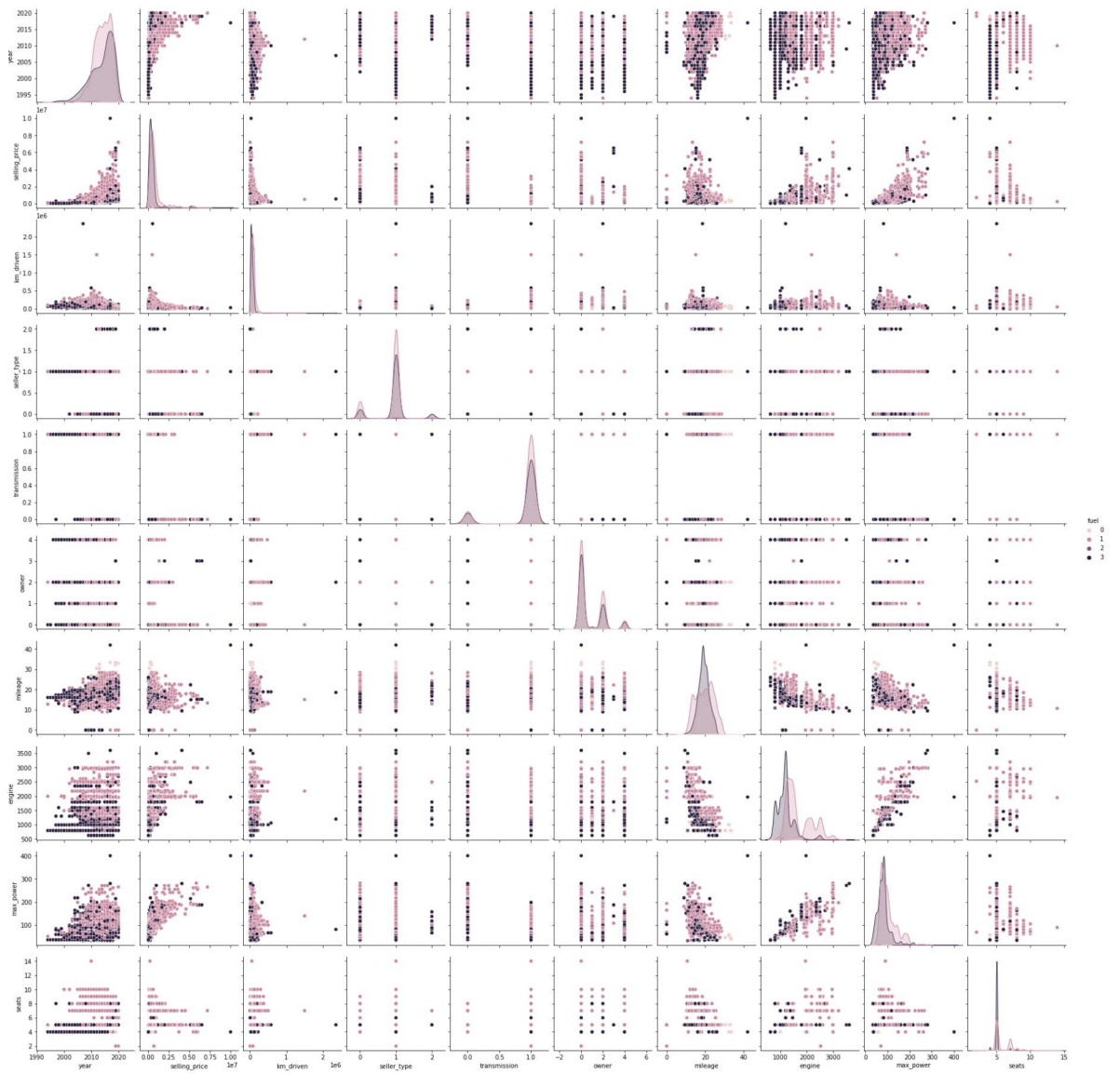
In [17]:
```
df.dropna(inplace = True)
df.reset_index(inplace = True, drop = True)
df.drop(['name', 'torque'], inplace = True, axis = 1)
```

In [18]:
```
lst, lst1, lst2 = [], [], []
for i in range(0, 7906):
    lst.append(re.sub('[^0-9.]', '', str(df['mileage'][i])))
    lst1.append(re.sub('[^0-9.]', '', str(df['engine'][i])))
    lst2.append(re.sub('[^0-9.]', '', str(df['max_power'][i])))
new_lst = list(map(float, lst))
new_lst1 = list(map(float, lst1))
new_lst2 = list(map(float, lst2))
df['mileage'] = new_lst
df['engine'] = new_lst1
df['max_power'] = new_lst2
df.head()
```

Out[18]:

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_
0	2014	450000	145500	1	1	1	0	23.40	1248.0	
1	2014	370000	120000	1	1	1	2	21.14	1498.0	
2	2006	158000	140000	3	1	1	4	17.70	1497.0	
3	2010	225000	127000	1	1	1	0	23.00	1396.0	
4	2007	130000	120000	3	1	1	0	16.10	1298.0	

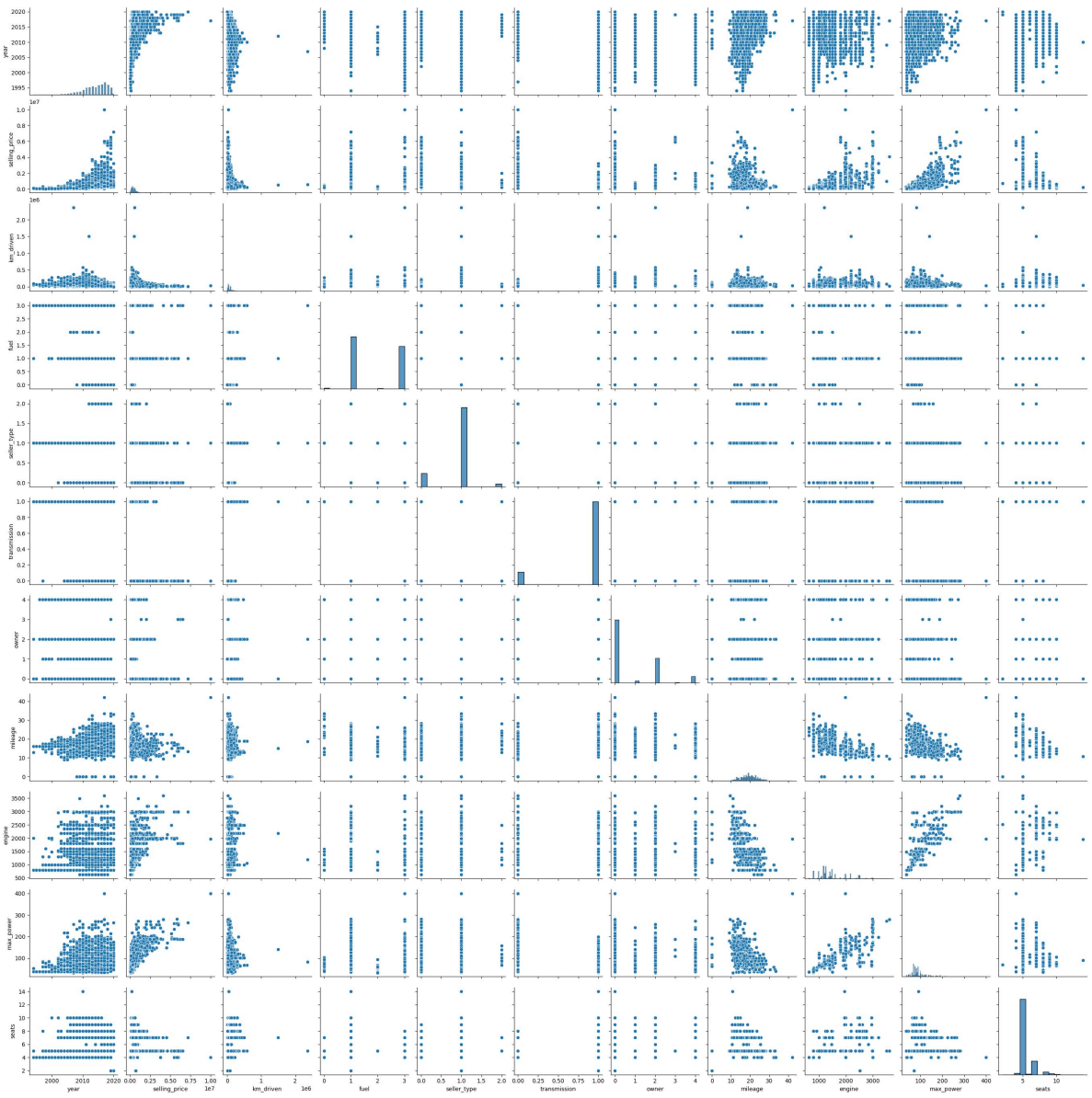
In [19]:
```
sns.pairplot(df, hue = 'fuel')
plt.show()
```



In [ ]:
```
1
```

In [20]:
```
1 sns.pairplot(df)
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x2772cd2bd00>



In [21]:
```
plt.figure(figsize=(18, 10))
sns.heatmap(df.corr(), linecolor = 'white', linewidths = 1, cmap = 'coolwa
plt.show()
```

Spitting the Dataset

In [22]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(np.array(df.drop('sel
                                                    test_size = 0.25, ran
```

In [23]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [24]:
```
regressor=RandomForestRegressor()

```

In [25]:
```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num
print(n_estimators)

```

```
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]
```

In [26]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [27]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [28]:
```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 12
00], 'max_features': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 30],
'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

In [29]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
```

In [30]:
```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = rand
```

In [31]:
```
1 rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=900; total time=   2.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=900; total time=   2.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=900; total time=   2.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=900; total time=   2.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=900; total time=   3.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=1100; total time=   4.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=1100; total time=   3.8s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=1100; total time=   3.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=1100; total time=   3.8s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=1100; total time=   3.8s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=100, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=100, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=100, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=100, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=100, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=400; total time=   3.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=400; total time=   3.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=400; total time=   3.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=400; total time=   3.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_spl
it=5, n_estimators=400; total time=   3.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_sp
lit=5, n_estimators=700; total time=   5.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_sp
lit=5, n_estimators=700; total time=   4.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_sp
lit=5, n_estimators=700; total time=   5.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_sp
lit=5, n_estimators=700; total time=   4.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_sp
lit=5, n_estimators=700; total time=   4.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=2, n_estimators=1000; total time=   5.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=2, n_estimators=1000; total time=   5.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=2, n_estimators=1000; total time=   5.5s
```

```
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=2, n_estimators=1000; total time=   5.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=2, n_estimators=1000; total time=   5.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_spl
it=15, n_estimators=1100; total time=   2.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_spl
it=15, n_estimators=1100; total time=   2.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_spl
it=15, n_estimators=1100; total time=   2.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_spl
it=15, n_estimators=1100; total time=   2.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_spl
it=15, n_estimators=1100; total time=   2.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=300; total time=   1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=300; total time=   1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=300; total time=   1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=300; total time=   1.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=300; total time=   1.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spli
t=10, n_estimators=700; total time=   1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spli
t=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spli
t=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spli
t=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spli
t=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=700; total time=   6.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=700; total time=   5.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=700; total time=   5.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=700; total time=   5.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_spl
it=15, n_estimators=700; total time=   5.8s
```

Out[31]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [32]:
```
1 rf_random.best_params_
```

Out[32]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 25}
```

In [33]:
```
1 rf_random.best_score_
2
```

Out[33]:
```
-29545027091.252174
```

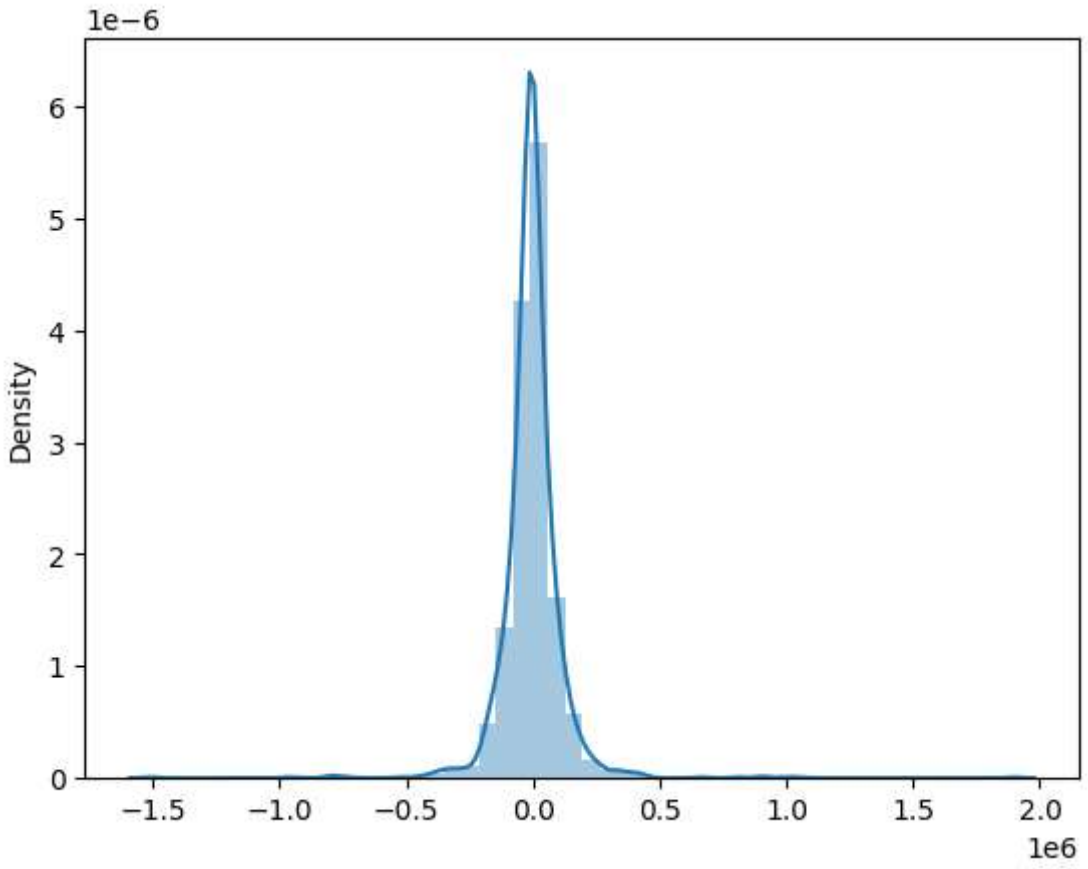
In [34]:
```
1 predictions=rf_random.predict(X_test)
```

In [35]:
```
1 predictions
```

Out[35]:
```
array([ 425691.995     ,  779000.        ,  197197.72936147, ...,
        464358.895     ,  227019.96428571, 4098140.        ])
```

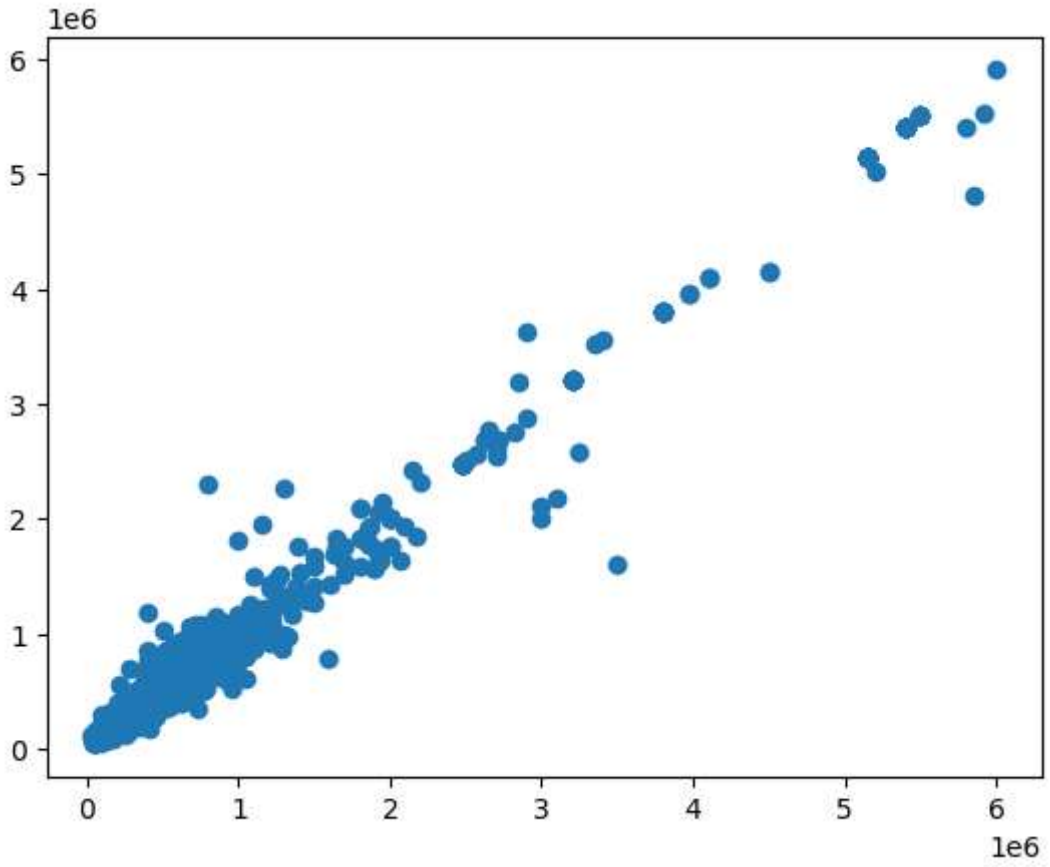
In [36]:
```
1 sns.distplot(y_test-predictions)
```

Out[36]:
```
<Axes: ylabel='Density'>
```



In [37]:
```
1 plt.scatter(y_test,predictions)
```

Out[37]: <matplotlib.collections.PathCollection at 0x277261fc0a0>



In [ ]:
```
1
```

In [38]:
```
1 from sklearn import metrics
```

In [39]:
```
1 print('MAE:', metrics.mean_absolute_error(y_test, predictions))
2 print('MSE:', metrics.mean_squared_error(y_test, predictions))
3 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 67427.24464088162
MSE: 15670028784.805992
RMSE: 125179.9855600167
```

In [40]:
```
1 import pickle
2 # open a file, where you want to store the data
3 file = open('random_forest_regression_model.pklnaik', 'wb')
4
5 # dump information to that file
6 pickle.dump(rf_random, file)
7
```

In [41]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report
```

In [42]:
```
X_train, X_test, y_train, y_test = train_test_split(X_train,y_train,test_
```

In [43]:
```
logreg=LogisticRegression()
logreg.fit(X_train,y_train)
```

Out[43]:
```
LogisticRegression()
```

In [44]:
```
y_pred=logreg.predict(X_test)
```

In [45]:
```
accuracy=accuracy_score(y_test,y_pred)
```

In [46]:
```
report=classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

       31000       0.00      0.00      0.00         1
       40000       0.00      0.00      0.00         3
       45000       0.00      0.00      0.00         3
       45957       0.00      0.00      0.00         1
       50000       0.00      0.00      0.00         3
       60000       0.00      0.00      0.00         1
       65000       0.00      0.00      0.00         3
       68000       0.00      0.00      0.00         1
       70000       0.00      0.00      0.00         2
       75000       0.00      0.00      0.00         2
       80000       0.00      0.00      0.00         5
       81000       0.00      0.00      0.00         1
       85000       0.00      0.00      0.00         4
       90000       0.00      0.00      0.00         4
       92000       0.00      0.00      0.00         1
       95000       0.00      0.00      0.00         1
       98000       0.00      0.00      0.00         1
```

In [ ]:

In [ ]: