In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.image as mpimg

from IPython.core.display    import HTML
from IPython.display         import Image
from tabulate                import tabulate
#from mpl_toolkits.basemap    import Basemap
from scipy.stats             import chi2_contingency

from sklearn.preprocessing   import RobustScaler, MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from boruta                  import BorutaPy
from sklearn.ensemble        import RandomForestRegressor
from sklearn.linear_model    import LinearRegression, Lasso
from sklearn.ensemble        import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import RandomizedSearchCV
import xgboost               as xgb
from sklearn import metrics

from sklearn.metrics         import mean_absolute_error, mean_squared_error
```

In [2]:
```
df = pd. read_csv('New_DB.csv',low_memory=False, error_bad_lines=False)
```

In [3]:
```
df.drop(['Unnamed: 0'],axis=1,inplace=True)
```

In [4]:
```
df.head(5)
```

Out[4]:

	region	price	condition	cylinders	fuel	odometer	title_status	transmission	drive	size
0	auburn	15000	excellent	6 cylinders	gas	128000.0	clean	automatic	rwd	rwd
1	auburn	27990	good	8 cylinders	gas	68696.0	clean	other	4wd	4wd
2	auburn	34590	good	6 cylinders	gas	29499.0	clean	other	4wd	4wd
3	auburn	35000	excellent	6 cylinders	gas	43000.0	clean	automatic	4wd	4wd
4	auburn	29990	good	6 cylinders	gas	17302.0	clean	other	4wd	4wd

In [5]:
```
df.drop(['region'],axis=1,inplace=True)
```

In [6]:
```
num_attributes = df.select_dtypes( include=['int64', 'float64'] )
cat_attributes = df.select_dtypes( exclude=['int64', 'float64'] )
num_attributes.sample()
```

Out[6]:

	price	odometer	lat	long	no_year
147631	11970	136536.0	41.366331	-82.259924	5

In [7]:
```
cat_attributes.sample()
```

Out[7]:

	condition	cylinders	fuel	title_status	transmission	drive	size	type	paint_color
39479	excellent	6 cylinders	gas	clean	automatic	4wd	4wd	wagon	silver

In [8]:
```
cat_attributes['size'].unique()
```

Out[8]:
```
array(['rwd', '4wd', 'fwd', 'mid-size'], dtype=object)
```

In [9]:
```
df['drive'].value_counts()
```

Out[9]:
```
4wd    91811
fwd    77104
rwd    42651
Name: drive, dtype: int64
```

In [10]:
```
df = df[df['size'] != 'mid-size']
```

In [11]:
```
df['size'].value_counts()
```

Out[11]:
```
4wd    92367
fwd    76454
rwd    42744
Name: size, dtype: int64
```

In [12]:
```
df.shape
```

Out[12]:
```
(211565, 14)
```

In [13]:
```
df1 = df[df['size'] == df['drive']]
```

In [14]:
```
df1.shape
```

Out[14]:
```
(209255, 14)
```

In [15]:
```
#Since both the drive type and size are matching I am gonna drop drive type
df1.drop(['drive'],axis=1,inplace=True)
df1.drop(['paint_color'],axis=1,inplace=True)
```

```
C:\Users\Elamathi\anaconda3\lib\site-packages\pandas\core\frame.py:4308: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  return super().drop(
```

In [16]:
```
df1.head(5)
```

Out[16]:

	price	condition	cylinders	fuel	odometer	title_status	transmission	size	type	lat
0	15000	excellent	6 cylinders	gas	128000.0	clean	automatic	rwd	truck	32.5920
1	27990	good	8 cylinders	gas	68696.0	clean	other	4wd	pickup	32.5900
2	34590	good	6 cylinders	gas	29499.0	clean	other	4wd	pickup	32.5900
3	35000	excellent	6 cylinders	gas	43000.0	clean	automatic	4wd	truck	32.6013
4	29990	good	6 cylinders	gas	17302.0	clean	other	4wd	pickup	32.5900

In [17]:
```
print(df1['condition'].unique())
print(df1['type'].unique())
print(df1['transmission'].unique())
print(df1['fuel'].unique())
print(df1['size'].unique())
print(df1['title_status'].unique())
```

```
['excellent' 'good' 'new' 'like new' 'fair' 'salvage']
['truck' 'pickup' 'other' 'coupe' 'SUV' 'mini-van' 'sedan' 'hatchback'
 'offroad' 'convertible' 'van' 'wagon' 'bus']
['automatic' 'other' 'manual']
['gas' 'other' 'diesel' 'hybrid' 'electric']
['rwd' '4wd' 'fwd']
['clean' 'rebuilt' 'salvage' 'missing' 'lien' 'parts only']
```

In [18]:
```
final_dataset=pd.get_dummies(df1 ,drop_first=True)
```

In [19]:
```
final_dataset.head(5)
```

Out[19]:

	price	odometer	lat	long	no_year	condition_fair	condition_good	condition_like
0	15000	128000.0	32.5920	-85.518900	7	0	0	
1	27990	68696.0	32.5900	-85.480000	8	0	1	
2	34590	29499.0	32.5900	-85.480000	4	0	1	
3	35000	43000.0	32.6013	-85.443974	1	0	0	
4	29990	17302.0	32.5900	-85.480000	4	0	1	

5 rows × 42 columns

In [20]:
```
final_dataset.corr()
```

Out[20]:

	price	odometer	lat	long	no_year	condition_fair
price	1.000000	-0.226282	-0.002266	-0.079190	-0.393922	-0.087591
odometer	-0.226282	1.000000	0.004877	0.017566	0.186131	0.081269
lat	-0.002266	0.004877	1.000000	-0.159362	0.009114	0.036649
long	-0.079190	0.017566	-0.159362	1.000000	0.013879	-0.020917
no_year	-0.393922	0.186131	0.009114	0.013879	1.000000	0.142656
condition_fair	-0.087591	0.081269	0.036649	-0.020917	0.142656	1.000000
condition_good	0.087880	-0.000021	0.001247	0.078867	0.028548	-0.152088
condition_like new	0.054695	-0.035296	-0.008602	-0.016890	-0.058402	-0.066348
condition_new	0.023222	-0.011646	-0.005678	0.000872	-0.008529	-0.011432
condition_salvage	0.017545	0.006546	0.006064	-0.027411	0.007221	-0.012835
cylinders_12 cylinders	0.004472	0.001175	-0.000309	-0.002293	0.020685	0.003924
cylinders_3 cylinders	-0.025749	-0.014266	0.000823	-0.018777	-0.014378	-0.007641
cylinders_4 cylinders	-0.292217	-0.044302	0.006011	-0.028540	-0.138115	-0.066278
cylinders_5 cylinders	-0.056329	0.011020	0.003576	0.000528	0.014609	-0.000322
cylinders_6 cylinders	0.029888	0.000652	-0.015285	0.022113	-0.031964	-0.024856
cylinders_8 cylinders	0.283856	0.045475	0.007468	0.012664	0.173516	0.095447
cylinders_other	0.005244	-0.009634	0.008985	-0.035855	-0.004443	-0.002068
fuel_electric	0.008544	-0.018505	-0.001870	-0.023587	-0.026693	-0.009367
fuel_gas	-0.223290	-0.017242	0.020502	-0.007927	0.048915	0.012660
fuel_hybrid	-0.034561	-0.002505	-0.008486	-0.014212	-0.026918	-0.014994
fuel_other	0.146355	-0.041157	-0.034000	0.035336	-0.094962	-0.015295
title_status_lien	0.015270	-0.001656	0.006016	0.003438	-0.015083	0.000086
title_status_missing	-0.037421	0.024445	-0.000119	0.004231	0.114554	0.058339
title_status_parts only	-0.019302	0.052650	-0.001828	0.000396	0.035785	0.015164
title_status_rebuilt	-0.048918	-0.018782	0.008860	0.012296	-0.031106	-0.007014
title_status_salvage	-0.060053	0.000195	-0.018382	-0.019906	0.000978	0.018272
transmission_manual	-0.081766	0.034345	0.014312	-0.030479	0.256741	0.040569
transmission_other	0.320595	-0.133124	-0.057134	0.091339	-0.192686	-0.053302
size_fwd	-0.361062	-0.017295	-0.086982	-0.004205	-0.106308	-0.051890
size_rwd	0.085089	-0.006955	-0.161920	0.012808	0.245564	0.015799
type_bus	-0.000040	0.009182	-0.003204	0.007793	0.022170	0.006421
type_convertible	-0.005110	-0.023477	-0.030829	0.018414	0.157080	-0.006008

	price	odometer	lat	long	no_year	condition_fair
type_coupe	0.056827	-0.036532	-0.034117	0.016425	0.135592	-0.002119
type_hatchback	-0.110587	-0.011640	-0.004330	-0.011995	-0.025442	-0.014061
type_mini-van	-0.083670	0.022466	0.008028	0.011274	0.012114	-0.000394
type_offroad	-0.003097	0.019371	-0.008055	-0.012995	0.063054	0.004115
type_other	0.112365	-0.035374	-0.030093	0.027988	-0.034294	-0.017012
type_pickup	0.253091	0.006591	-0.017007	0.002940	-0.022578	0.062555
type_sedan	-0.262984	-0.019652	-0.031775	0.006275	-0.038968	-0.040670
type_truck	0.199686	0.042234	0.053436	-0.028878	0.034941	0.063831
type_van	0.007622	0.000979	-0.014956	0.015888	-0.029824	-0.009380
type_wagon	-0.064513	0.006734	0.028521	-0.007594	-0.013040	-0.012822

42 rows × 42 columns

In [21]:

```
#get correlations of each features in dataset
plt.figure(figsize=(18,18))
sns.heatmap(final_dataset.corr(),annot=True,cmap='RdYlGn')
plt.show()
```

In [22]:
```
#upper_tri = corrmat.where(np.triu(np.ones(corrmat.shape),k=1).astype(np.bool
#print(upper_tri)
```

In [23]:
```
#to_drop = [column for column in upper_tri.columns if any(upper_tri[column] >
#print()
#print(to_drop)
```

In [24]:
```
#df3 = df.drop(df.columns[to_drop], axis=1)
#print()
#print(df3.head())
```

In [25]:
```
X=final_dataset.iloc[:,1:]
y=final_dataset.iloc[:,0]
```

In [26]:
```
X.head()
```

Out[26]:

	odometer	lat	long	no_year	condition_fair	condition_good	condition_like new	cond
0	128000.0	32.5920	-85.518900	7	0	0	0	
1	68696.0	32.5900	-85.480000	8	0	1	0	
2	29499.0	32.5900	-85.480000	4	0	1	0	
3	43000.0	32.6013	-85.443974	1	0	0	0	
4	17302.0	32.5900	-85.480000	4	0	1	0	

5 rows × 41 columns

In [27]:
```
y.head()
```

Out[27]:
```
0    15000
1    27990
2    34590
3    35000
4    29990
Name: price, dtype: int64
```

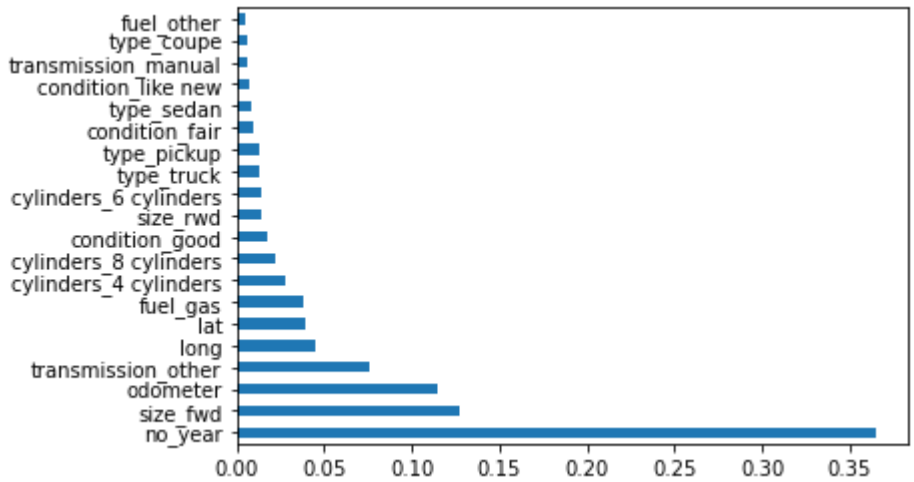
In [28]:
```
### Feature Importance
### improve the predictive accuracy and control over-fitting.
model = ExtraTreesRegressor()
model.fit(X,y)
```

Out[28]:
```
ExtraTreesRegressor()
```

In [29]:
```
print(model.feature_importances_)
```

```
[1.14102557e-01 3.88962406e-02 4.51506923e-02 3.64361190e-01
 9.30437806e-03 1.74143343e-02 6.98743993e-03 8.12940005e-04
 1.88928572e-03 4.50748947e-04 5.87485493e-04 2.74693326e-02
 6.37001699e-04 1.41523791e-02 2.19737991e-02 7.47204610e-04
 4.69976282e-04 3.80798824e-02 1.56856616e-03 4.93229312e-03
 7.63814763e-04 2.73028773e-04 7.22264956e-05 3.80692021e-03
 3.09340461e-03 6.21576805e-03 7.58082549e-02 1.27105690e-01
 1.45138969e-02 3.24270801e-04 4.09265767e-03 5.59074534e-03
 2.47812304e-03 1.19874609e-03 9.99204556e-04 4.24650519e-03
 1.26145793e-02 8.42077678e-03 1.32610222e-02 3.45951447e-03
 1.67312206e-03]
```

In [40]:
```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```



In [41]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
```

In [42]:
```
regressor=RandomForestRegressor()
```

In [43]:
```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 1
print(n_estimators)
```

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

In [44]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 1
# 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 [45]:
```
# 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,
1200], 'max_features': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 3
0], 'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5,
10]}
```

In [46]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
```

In [47]:
```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_g
```

In [48]:
```
rf_random.fit(X_train,y_train)
```

Out[48]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25,
30],
                                        'max_features': ['auto', 'sqr
t'],
                                        'min_samples_leaf': [1, 2, 5, 1
0],
                                        'min_samples_split': [2, 5, 10,
15,
                                                              100],
```

In [49]:
```
rf_random.best_params_
```

Out[49]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 25}
```

In [50]:
```
rf_random.best_score_
```

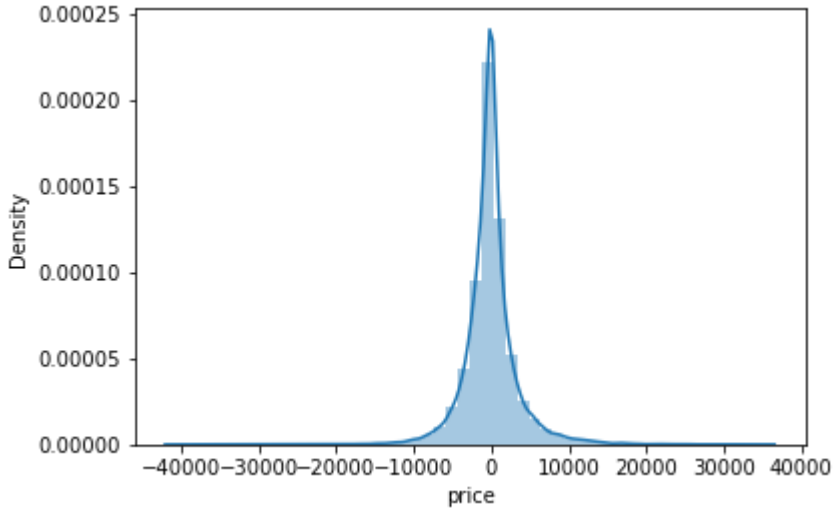
Out[50]:
```
-14669369.689400893
```

In [51]:
```
predictions=rf_random.predict(X_test)
```

In [52]:
```
sns.distplot(y_test-predictions)
```

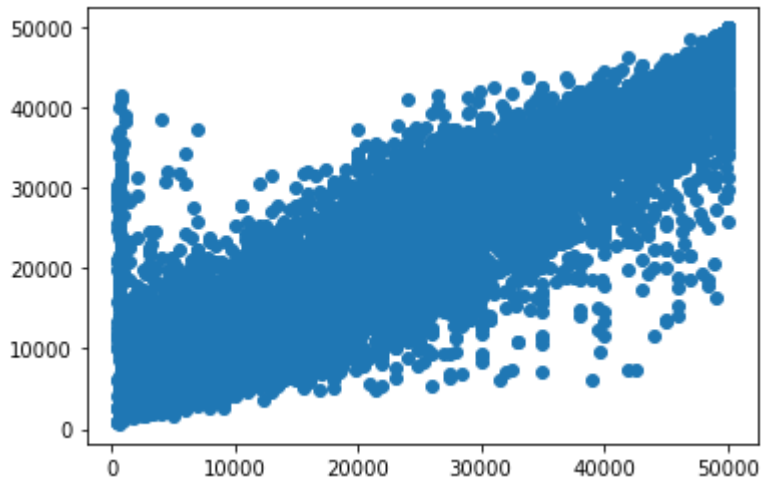
```
C:\Users\Elamathi\anaconda3\lib\site-packages\seaborn\distributions.py:255
1: FutureWarning: `distplot` is a deprecated function and will be removed i
n a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[52]: <AxesSubplot:xlabel='price', ylabel='Density'>



In [53]:
```
plt.scatter(y_test,predictions)
```

Out[53]: <matplotlib.collections.PathCollection at 0x1e532810b50>



In [54]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 2249.5769995673004
MSE: 13869288.071443995
RMSE: 3724.1493084252133
```

In [ ]:
```

```