In [1]:
```
#Random Forest Implementation for 16 variables
# packages to be used
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("df_cleaned.csv")
```

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

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	20141013T000000	221900	3	1	1180	5650	1	
1	6414100192	20141209T000000	538000	3	2	2570	7242	2	
2	5631500400	20150225T000000	180000	2	1	770	10000	1	
3	2487200875	20141209T000000	604000	4	3	1960	5000	1	
4	1954400510	20150218T000000	510000	3	2	1680	8080	1	

5 rows × 21 columns

In [4]:
```
df.shape
```

Out[4]:
```
(17498, 21)
```

In [5]:
```
#print highly correlated variables
corr_features =[]

for i , r in df.corr().iterrows():
    k=0
    for j in range(len(r)):
        if i!= r.index[k]:
            if r.values[k] >=0.5:
                corr_features.append([i, r.index[k], r.values[k]])
        k += 1
corr_features
```

Out[5]:
```
[['price', 'sqft_living', 0.5681513121388794],
 ['price', 'grade', 0.5865202142141626],
 ['price', 'sqft_living15', 0.5053581518565498],
 ['bedrooms', 'sqft_living', 0.6075464596376176],
 ['bathrooms', 'sqft_living', 0.5895526844843015],
 ['bathrooms', 'floors', 0.5101526374668407],
 ['bathrooms', 'grade', 0.5094153760693566],
 ['bathrooms', 'sqft_above', 0.5328737348429328],
 ['sqft_living', 'price', 0.5681513121388794],
 ['sqft_living', 'bedrooms', 0.6075464596376176],
 ['sqft_living', 'bathrooms', 0.5895526844843015],
 ['sqft_living', 'grade', 0.671228606324547],
 ['sqft_living', 'sqft_above', 0.8407618549316075],
 ['sqft_living', 'sqft_living15', 0.7322234781398173],
 ['sqft_lot', 'sqft_lot15', 0.6976557104966404],
 ['floors', 'bathrooms', 0.5101526374668407],
 ['floors', 'sqft_above', 0.5323963159598748],
 ['floors', 'yr_built', 0.6146183068331962],
 ['grade', 'price', 0.5865202142141626],
 ['grade', 'bathrooms', 0.5094153760693566],
 ['grade', 'sqft_living', 0.671228606324547],
 ['grade', 'sqft_above', 0.6754280632662011],
 ['grade', 'sqft_living15', 0.640666653952043],
 ['sqft_above', 'bathrooms', 0.5328737348429328],
 ['sqft_above', 'sqft_living', 0.8407618549316075],
 ['sqft_above', 'floors', 0.5323963159598748],
 ['sqft_above', 'grade', 0.6754280632662011],
 ['sqft_above', 'sqft_living15', 0.7094855810853424],
 ['yr_built', 'floors', 0.6146183068331962],
 ['sqft_living15', 'price', 0.5053581518565498],
 ['sqft_living15', 'sqft_living', 0.7322234781398173],
 ['sqft_living15', 'grade', 0.640666653952043],
 ['sqft_living15', 'sqft_above', 0.7094855810853424],
 ['sqft_lot15', 'sqft_lot', 0.6976557104966404]]
```

In [6]:
```
#let us remove highly correlated features that is above 0.8
feat =[]
for i in corr_features:
    if i[2] >= 0.8:
        feat.append(i[0])
        feat.append(i[1])

df.drop(list(set(feat)), axis=1, inplace=True)
df.head()
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	vie
0	7129300520	20141013T000000	221900	3	1	5650	1	0	
1	6414100192	20141209T000000	538000	3	2	7242	2	0	
2	5631500400	20150225T000000	180000	2	1	10000	1	0	
3	2487200875	20141209T000000	604000	4	3	5000	1	0	
4	1954400510	20150218T000000	510000	3	2	8080	1	0	

In [7]:
```
df.shape
```

Out[7]: (17498, 19)

In [8]:
```
df = df.iloc[0:1000]
```

In [9]:
```
predictors = ['bedrooms','bathrooms','sqft_lot','floors','waterfront','view','
              'grade','sqft_basement','yr_built','yr_renovated','zipcode','lat
outcome = 'price'
```

In [10]:
```
#Partition Data
X = pd.get_dummies(df[predictors],drop_first=True)
y = df[outcome]

# Split dataset into train and test

train_X, valid_X, train_y, valid_y = train_test_split(X,y, test_size=0.2, rand
```

In [11]:
```
#Importing RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

#Creating Random Forest Regressor
rfc = RandomForestRegressor(n_estimators = 1)
```

In [12]:
```
#We’ll use x_train and y_train to fit our model.
rfc.fit(train_X,train_y)
```

Out[12]: RandomForestRegressor(n_estimators=1)

In [13]:
```
#Let’s calculate our model’s score using valid_x and valid_y.
rfc.score(valid_X,valid_y)
```

Out[13]: 0.5659536291705745

In [14]:
```
#We have fitted the model and seen its performance. Let us predict the prices
rfc_pred = rfc.predict(valid_X)
print(rfc_pred)
```

```
[ 357000.  770000.  224500.  500000.  986000.  565000.  335000.  227950.
  485000.  555000.  194000.  597750.  229000.  435000.  387000.  785000.
  326100.  285000.  920000.  856600.  365000.  256883.  791500.  791500.
  360000.  194000.  341500.  532500.  565000.  226500.  819900.  365000.
  619000.  277000.  332500.  245000.  218500.  265000.  550000.  341500.
  585000.  214000.  365000.  390000.  410000.  860000.  315000.  275000.
  510000.  390000.  218000.  245000.  305000.  445000.  235000.  410000.
  170000.  516500.  423000.  375000.  940000.  252500.  245000.  485000.
  260000.  194000.  441000.  315000.  430000.  245000.  553500.  430000.
  510000.  232000.  516500.  550000.  324000.  385000.  194000.  880000.
  341500.  655000.  510000.  610750.  252500.  490000.  315000.  385000.
  255000.  496500.  604000.  324000.  245000.  210000.  469950.  380000.
  542000.  460000.  532500.  237500.  725000.  531000.  320000.  173000.
  699800.  237000.  226500.  551000.  240000.  669950.  280300.  365000.
  600000.  834000.  360000.  538200.  196500.  824000.  153000.  153000.
  560000.  237000.  699000.  245000.  609900.  360000.  360000.  860000.
  341500.  240000.  791500.  510000.  245000.  415000.  531000.  328000.
  385000.  259950.  648000.  510000.  252500.  610750.  485000.  358000.
  510000.  560000.  642450.  269950.  819900.  430000.  615000.  880000.
  252350.  648000.  485000.  345000.  196000.  245000.  245000.  320000.
  265000.  232000.  365000.  289950.  423000.  330000.  290000.  256883.
  180000.  245000.  648000.  375000.  218000.  400000. 1088000.  289950.
  290000.  673000.  467000.  610750.  490000.  335000.  249950.  250000.
  610000.  194000.  856600.  610000.  325000.  196000.  447000.  335000.
  669950.  378500.  253000.  360000.  720000.  510000.  205000.  252500.]
```

In [15]:
```
rsquared = metrics.r2_score(valid_y,rfc_pred)
adjusted_r_squared = 1 - (1-rsquared)*(len(valid_y)-1)/(len(valid_y)-valid_X.s
print('Mean absolute error: {}'.format(metrics.mean_absolute_error(valid_y, rf
print('Mean squared error: {}'.format(metrics.mean_squared_error(valid_y, rfc_
print('Root mean squared error: {}'.format(np.sqrt(metrics.mean_squared_error(
print('R Squared value: {}'.format(rsquared))
print('Adjusted R Squared Value: {}'.format(adjusted_r_squared))
```

```
Mean absolute error: 88835.635
Mean squared error: 16621930676.065
Root mean squared error: 128926.06670516633
R Squared value: 0.5659536291705745
Adjusted R Squared Value: 0.5280042196991492
```

In [16]:
```
from dmba import regressionSummary
```

In [17]:
```
regressionSummary(valid_y, rfc_pred)
```

```
Regression statistics

                      Mean Error (ME) : 15312.4650
       Root Mean Squared Error (RMSE) : 128926.0667
            Mean Absolute Error (MAE) : 88835.6350
          Mean Percentage Error (MPE) : 1.0657
Mean Absolute Percentage Error (MAPE) : 19.5696
```

In [18]:
```
#ran the model 50 times with estimators ranging from 1 to 50. The root mean sq
RMSE_rfc = []
for i in range(1,50):
    rfc = RandomForestRegressor(n_estimators=i)
    rfc.fit(train_X,train_y)
    pred_i = rfc.predict(valid_X)
    RMSE_rfc.append((np.sqrt(metrics.mean_squared_error(valid_y, pred_i))))
print('Minimum Root Mean Squared Error is {} with {} estimators'.format(round(
```

```
Minimum Root Mean Squared Error is 92397.524 with 14 estimators
```

In [19]:
```
rfc = RandomForestRegressor(n_estimators=50)
rfc.fit(train_X,train_y)
pred_p = rfc.predict(valid_X)
rsquared_p = metrics.r2_score(valid_y,pred_p)
adjusted_r_squared_p = 1 - (1-rsquared_p)*(len(valid_y)-1)/(len(valid_y)-valid
print('Mean absolute error: {}'.format(metrics.mean_absolute_error(valid_y, pr
print('Mean squared error: {}'.format(metrics.mean_squared_error(valid_y, pred
print('Root mean squared error: {}'.format(np.sqrt(metrics.mean_squared_error(
print('R squared value: {}'.format(rsquared_p))
print('Adjusted squared value: {}'.format(adjusted_r_squared_p))
```

```
Mean absolute error: 61322.900733333336
Mean squared error: 8442685228.973589
Root mean squared error: 91884.08583086403
R squared value: 0.779537229753463
Adjusted squared value: 0.7602617962892848
```

In [27]:
```
regressionSummary(valid_y, pred_p)
```

```
Regression statistics

                      Mean Error (ME) : -2229.2796
       Root Mean Squared Error (RMSE) : 91884.0858
            Mean Absolute Error (MAE) : 61322.9007
          Mean Percentage Error (MPE) : -4.5280
Mean Absolute Percentage Error (MAPE) : 14.4825
```

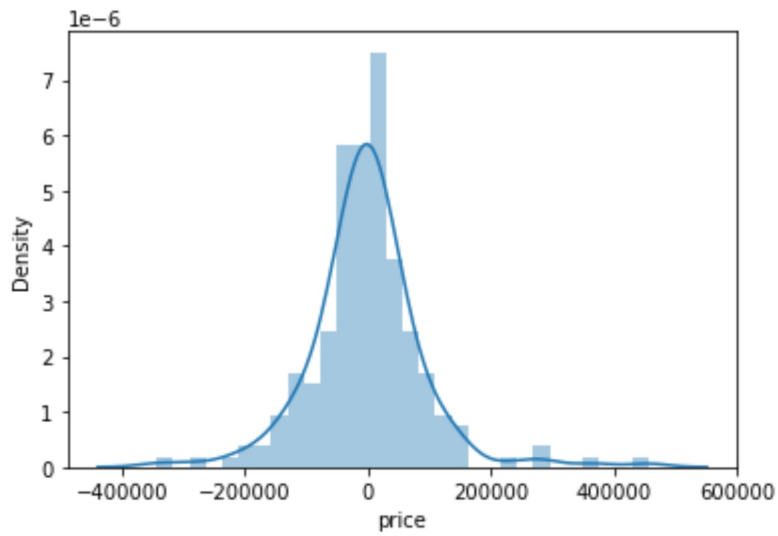
In [28]:
```
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.offline as ply
import plotly.graph_objs as go
from plotly import tools
init_notebook_mode(connected=True)
from  plotly.offline import plot
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode()
```

In [22]:
```
trace0 = go.Scatter(
x = valid_X.iloc[:,2],
y = valid_y,
mode = 'markers',
name = 'Test Set'
)
trace1 = go.Scatter(
x = valid_X.iloc[:,2],
y = pred_p,
opacity = 0.75,
mode = 'markers',
name = 'Predictions',
marker = dict(line = dict(color = 'black', width = 0.5))
)
data = [trace0, trace1]
ply.iplot(data)
```

In [23]:
```
residualr = (valid_y- pred_p)
sns.distplot(residualr);
```

```
C:\Users\kadam\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



In [ ]:

In [ ]: