

# Code and Output

December 17, 2020

Importing Required Libraries

```
[33]: import pandas
import pandas as pd
import numpy as np
import datetime as dt

import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn import preprocessing

from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

#import pandas as pd
#import numpy as np
#import matplotlib.pyplot as plt
#from sklearn.linear_model import LinearRegression, Ridge, Lasso
#from sklearn.model_selection import train_test_split, cross_val_score
#from statistics import mean

#import matplotlib.pyplot as plt
#import pandas as pd
#import numpy as np
```

Loading and Cleaning Data

```
[34]: data = pd.read_csv('kc_house_data.csv')
```

Data Analysis

```
[35]: format(data.shape)
```

```
[35]: '(21613, 21)'
```

```
[36]: data.dtypes
```

```
[36]: id                int64
      date              object
      price             float64
      bedrooms          int64
      bathrooms         float64
      sqft_living        int64
      sqft_lot           int64
      floors            float64
      waterfront         int64
      view              int64
      condition          int64
      grade              int64
      sqft_above         float64
      sqft_basement      int64
      yr_built           int64
      yr_renovated       int64
      zipcode           int64
      lat               float64
      long              float64
      sqft_living15      int64
      sqft_lot15         int64
      dtype: object
```

```
[37]: data.hist(figsize=(30,20))
      plt.show()
```



```
[38]: data.describe()
```

```
[38]:
```

	id	price	bedrooms	bathrooms	sqft_living \
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000

	sqft_lot	floors	waterfront	view	condition \
count	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000
mean	1.510697e+04	1.494309	0.007542	0.234303	3.409430
std	4.142051e+04	0.539989	0.086517	0.766318	0.650743
min	5.200000e+02	1.000000	0.000000	0.000000	1.000000
25%	5.040000e+03	1.000000	0.000000	0.000000	3.000000
50%	7.618000e+03	1.500000	0.000000	0.000000	3.000000
75%	1.068800e+04	2.000000	0.000000	0.000000	4.000000
max	1.651359e+06	3.500000	1.000000	4.000000	5.000000

	grade	sqft_above	sqft_basement	yr_built	yr_renovated \
count	21613.000000	21611.000000	21613.000000	21613.000000	21613.000000

mean	7.656873	1788.396095	291.509045	1971.005136	84.402258
std	1.175459	828.128162	442.575043	29.373411	401.679240
min	1.000000	290.000000	0.000000	1900.000000	0.000000
25%	7.000000	1190.000000	0.000000	1951.000000	0.000000
50%	7.000000	1560.000000	0.000000	1975.000000	0.000000
75%	8.000000	2210.000000	560.000000	1997.000000	0.000000
max	13.000000	9410.000000	4820.000000	2015.000000	2015.000000

	zipcode	lat	long	sqft_living15	sqft_lot15
count	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
mean	98077.939805	47.560053	-122.213896	1986.552492	12768.455652
std	53.505026	0.138564	0.140828	685.391304	27304.179631
min	98001.000000	47.155900	-122.519000	399.000000	651.000000
25%	98033.000000	47.471000	-122.328000	1490.000000	5100.000000
50%	98065.000000	47.571800	-122.230000	1840.000000	7620.000000
75%	98118.000000	47.678000	-122.125000	2360.000000	10083.000000
max	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

```
[39]: #Reference 1
data[["price","bedrooms","bathrooms","sqft_living","sqft_lot","sqft_above","yr_built","sqft_li
↪describe()
```

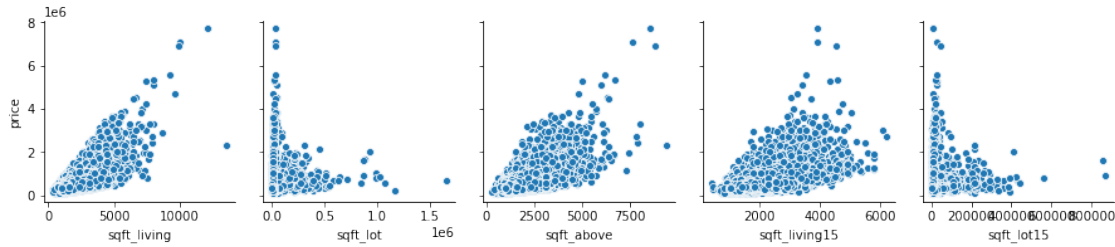
```
[39]:
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot \
count	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04
mean	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04
std	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04
min	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

	sqft_above	yr_built	sqft_living15	sqft_lot15
count	21611.000000	21613.000000	21613.000000	21613.000000
mean	1788.396095	1971.005136	1986.552492	12768.455652
std	828.128162	29.373411	685.391304	27304.179631
min	290.000000	1900.000000	399.000000	651.000000
25%	1190.000000	1951.000000	1490.000000	5100.000000
50%	1560.000000	1975.000000	1840.000000	7620.000000
75%	2210.000000	1997.000000	2360.000000	10083.000000
max	9410.000000	2015.000000	6210.000000	871200.000000

```
[40]: sns.pairplot(data=data,
↪x_vars=['sqft_living','sqft_lot','sqft_above','sqft_living15','sqft_lot15'],
↪y_vars=["price"])
```

```
[40]: <seaborn.axisgrid.PairGrid at 0x7feec1d928b0>
```



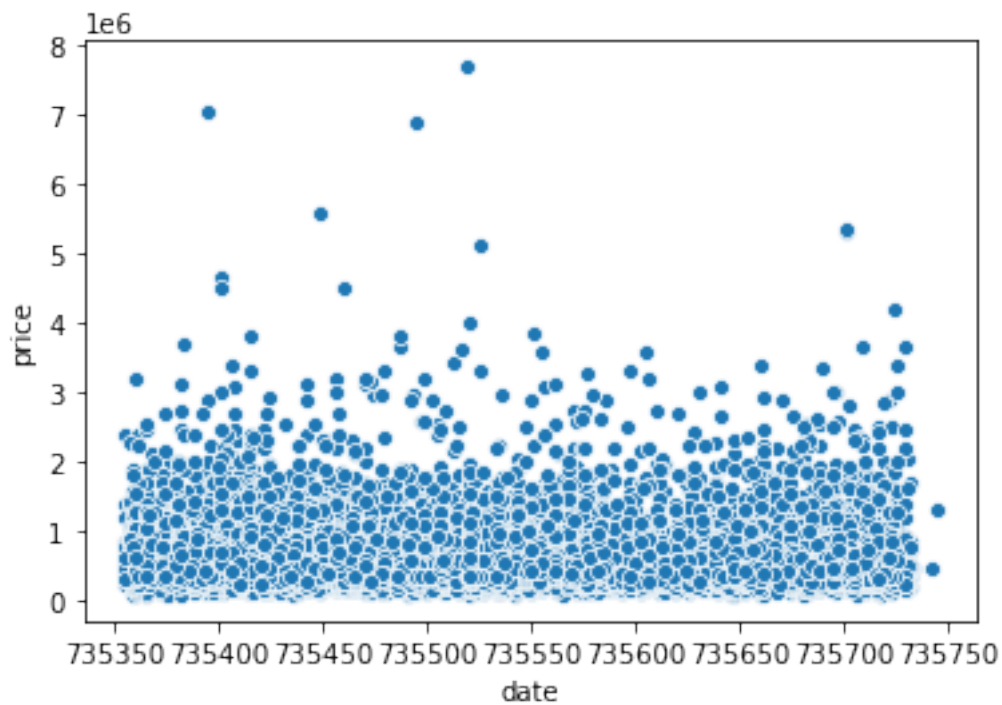
We will check to see which columns we should drop by testing everything against the column price

```
[41]: dateObj = dt.datetime.strptime('20140623T000000', '%Y%m%dT%H%M%S')

data['date'] = data['date'].apply(lambda x: dt.datetime.strptime(x,
↪ '%Y%m%dT%H%M%S'))
data['date'] = data['date'].map(dt.datetime.toordinal)
```

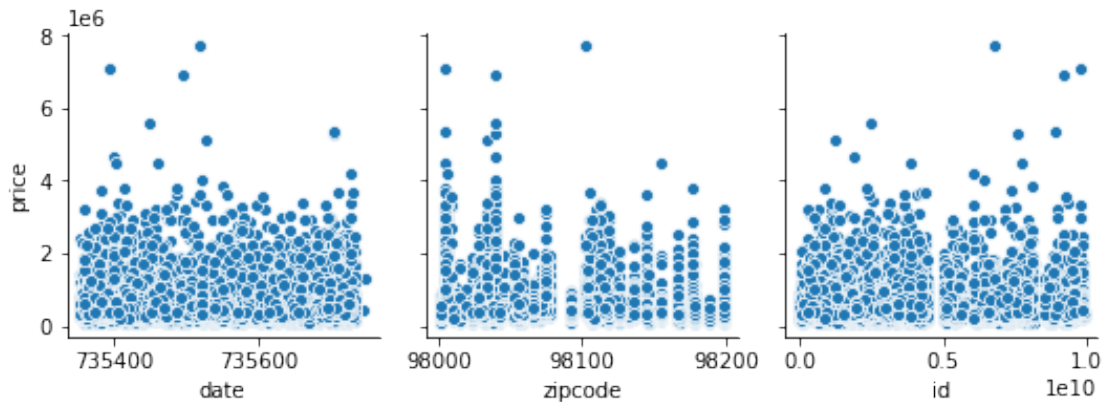
```
[42]: sns.scatterplot(data=data, x="date", y="price")
```

```
[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7feec2200430>
```



```
[43]: sns.pairplot(data=data, x_vars=['date', 'zipcode', 'id'], y_vars=["price"])
```

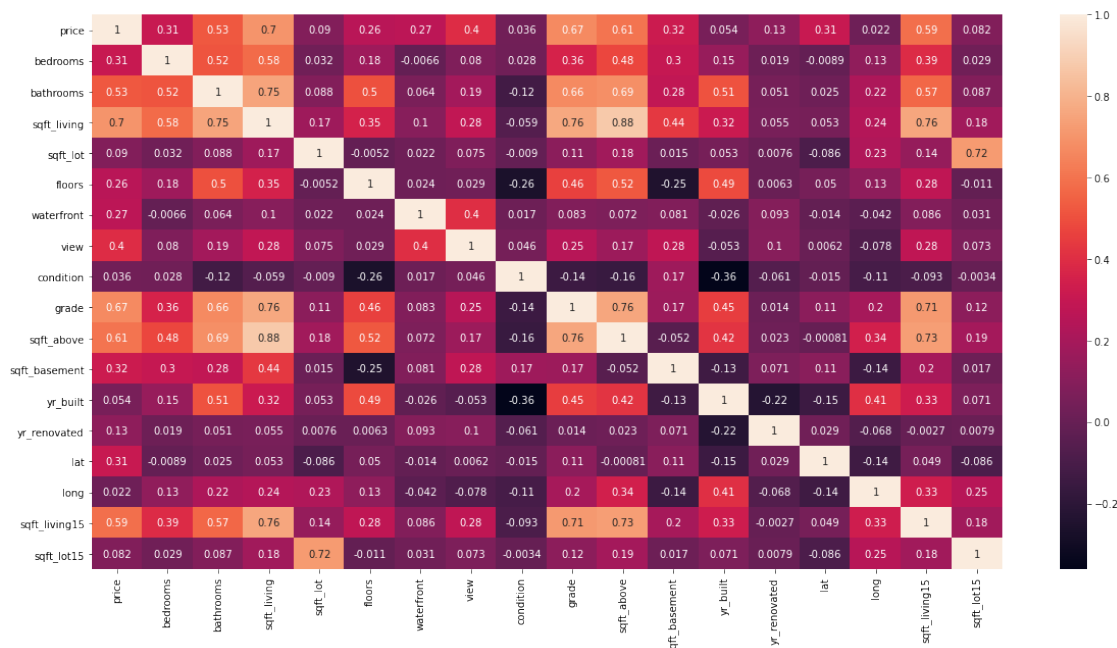
[43]: <seaborn.axisgrid.PairGrid at 0x7feec0e8b2b0>



```
[44]: #We could conclude that columns: id, date, and zipcode will not benefit us here,
      ↪so we will drop them
dropColumns = ['id', 'date', 'zipcode']
data = data.drop(dropColumns, axis = 1)
```

```
[45]: plt.figure(figsize=(20,10))
      sns.heatmap(data.corr(), annot=True)
```

[45]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7feec211cb20>



```
[46]: #From this, we see that columns: 'sqft_living', 'grade', 'sqft_above',
      ↪ 'sqft_living15', 'sqft_lot15', 'yr_built' are all highly correlated
def remove_collinear_features(x, threshold):

    # Create correlation matrix:
    corr_matrix = x.corr()
    iters = range(len(corr_matrix.columns) - 1)
    drop_cols = []

    # Work through the iterations setup:
    for i in iters:
        for j in range(i+1):
            items = corr_matrix.iloc[j:(j+1), (i+1):(i+2)]
            col = items.columns
            row = items.index
            val = abs(items.values)

            # Compare against threshold:
            if val >= threshold:
                print(col.values[0], "|", row.values[0], "|", round(val[0][0],
                ↪ 2))

                drop_cols.append(col.values[0])

    cols_to_drop = set(drop_cols)
    x = x.drop(columns = cols_to_drop, axis=1)

    return x
features = data.drop('price', axis=1)
remove_collinear_features(features, 0.7)
features = features.drop(columns = ['sqft_living', 'grade', 'sqft_above',
      ↪ 'sqft_living15', 'sqft_lot15', 'yr_built'], axis=1)
```

```
sqft_living | bathrooms | 0.75
grade | sqft_living | 0.76
sqft_above | sqft_living | 0.88
sqft_above | grade | 0.76
sqft_living15 | sqft_living | 0.76
sqft_living15 | grade | 0.71
sqft_living15 | sqft_above | 0.73
sqft_lot15 | sqft_lot | 0.72
```

```
[47]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn import preprocessing
```

```

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
import numpy as np
import statsmodels.api as sm
x = pd.DataFrame({
    'sqft_living': data.sqft_living,
    'grade': data.grade,
    'sqft_above': data.sqft_above,
    'sqft_living15' : data.sqft_living15,
    'bathrooms': data.bathrooms,
    'lat' : data.lat,
    'waterfront': data.waterfront,
    'floors': data.floors,
    'yr_renovated':data.yr_renovated,
    'sqft_lot':data.sqft_lot,
    'sqft_lot15':data.sqft_lot15,
    'yr_built':data.yr_built,
    'condition':data.condition,
    'long':data.long,

})

scaler = preprocessing.MinMaxScaler()
minmax_scaled_data = scaler.fit_transform(x)
minmax_scaled_data = pd.DataFrame(minmax_scaled_data, columns=['sqft_living',
    ↪ 'grade', 'sqft_above', 'sqft_living15', 'bathrooms', 'lat','waterfront',
    ↪ 'floors', 'yr_renovated', 'sqft_lot', 'sqft_lot15','yr_built','condition',
    ↪ 'long' ])

scaler = preprocessing.StandardScaler()
scaled_data = scaler.fit_transform(x)
scaled_data = pd.DataFrame(scaled_data, columns=['sqft_living', 'grade',
    ↪ 'sqft_above', 'sqft_living15', 'bathrooms', 'lat','waterfront', 'floors',
    ↪ 'yr_renovated', 'sqft_lot', 'sqft_lot15','yr_built','condition', 'long'])

scaler = preprocessing.RobustScaler()
robust_scaled_data = scaler.fit_transform(x)
robust_scaled_data = pd.DataFrame(robust_scaled_data, columns=['sqft_living',
    ↪ 'grade', 'sqft_above', 'sqft_living15', 'bathrooms', 'lat','waterfront',
    ↪ 'floors', 'yr_renovated', 'sqft_lot', 'sqft_lot15','yr_built','condition',
    ↪ 'long'])

fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, figsize=(9, 5))

```



```

ax1.set_title('Before Scaling')
sns.kdeplot(x['sqft_living'], ax=ax1)
sns.kdeplot(x['sqft_above'], ax=ax1)
sns.kdeplot(x['sqft_living15'], ax=ax1)
sns.kdeplot(x['bathrooms'], ax=ax1)
sns.kdeplot(x['lat'], ax=ax1)
sns.kdeplot(x['waterfront'], ax=ax1)
sns.kdeplot(x['floors'], ax=ax1)
sns.kdeplot(x['yr_renovated'], ax=ax1)
sns.kdeplot(x['sqft_lot15'], ax=ax1)
sns.kdeplot(x['yr_built'], ax=ax1)
sns.kdeplot(x['sqft_lot'], ax=ax1)
sns.kdeplot(x['condition'], ax=ax1)
sns.kdeplot(x['long'], ax=ax1)

ax2.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_scaled_data['sqft_living'], ax=ax2)
sns.kdeplot(minmax_scaled_data['sqft_above'], ax=ax2)
sns.kdeplot(minmax_scaled_data['sqft_living15'], ax=ax2)
sns.kdeplot(minmax_scaled_data['bathrooms'], ax=ax2)
sns.kdeplot(minmax_scaled_data['lat'], ax=ax2)
sns.kdeplot(minmax_scaled_data['waterfront'], ax=ax2)
sns.kdeplot(minmax_scaled_data['floors'], ax=ax2)
sns.kdeplot(minmax_scaled_data['yr_renovated'], ax=ax2)
sns.kdeplot(minmax_scaled_data['sqft_lot15'], ax=ax2)
sns.kdeplot(minmax_scaled_data['yr_built'], ax=ax2)
sns.kdeplot(minmax_scaled_data['sqft_lot'], ax=ax2)
sns.kdeplot(minmax_scaled_data['condition'], ax=ax2)
sns.kdeplot(minmax_scaled_data['long'], ax=ax2)

ax3.set_title('After Standard Scaler')
sns.kdeplot(scaled_data['sqft_living'], ax=ax3)
sns.kdeplot(scaled_data['sqft_above'], ax=ax3)
sns.kdeplot(scaled_data['sqft_living15'], ax=ax3)
sns.kdeplot(scaled_data['bathrooms'], ax=ax3)
sns.kdeplot(scaled_data['lat'], ax=ax3)
sns.kdeplot(scaled_data['waterfront'], ax=ax3)
sns.kdeplot(scaled_data['floors'], ax=ax3)
sns.kdeplot(scaled_data['yr_renovated'], ax=ax3)
sns.kdeplot(scaled_data['sqft_lot15'], ax=ax3)
sns.kdeplot(scaled_data['yr_built'], ax=ax3)
sns.kdeplot(scaled_data['sqft_lot'], ax=ax3)
sns.kdeplot(scaled_data['condition'], ax=ax3)
sns.kdeplot(scaled_data['long'], ax=ax3)

ax4.set_title('After Robust Scaling')

```

```

sns.kdeplot(robust_scaled_data['sqft_living'], ax=ax4)
sns.kdeplot(robust_scaled_data['sqft_above'], ax=ax4)
sns.kdeplot(robust_scaled_data['sqft_living15'], ax=ax4)
sns.kdeplot(robust_scaled_data['bathrooms'], ax=ax4)
sns.kdeplot(robust_scaled_data['lat'], ax=ax4)
sns.kdeplot(robust_scaled_data['waterfront'], ax=ax4)
sns.kdeplot(robust_scaled_data['floors'], ax=ax4)
sns.kdeplot(robust_scaled_data['yr_renovated'], ax=ax4)
sns.kdeplot(robust_scaled_data['sqft_lot15'], ax=ax4)
sns.kdeplot(robust_scaled_data['yr_built'], ax=ax4)
sns.kdeplot(robust_scaled_data['sqft_lot'], ax=ax4)
sns.kdeplot(robust_scaled_data['condition'], ax=ax4)
sns.kdeplot(robust_scaled_data['long'], ax=ax4)

plt.show()

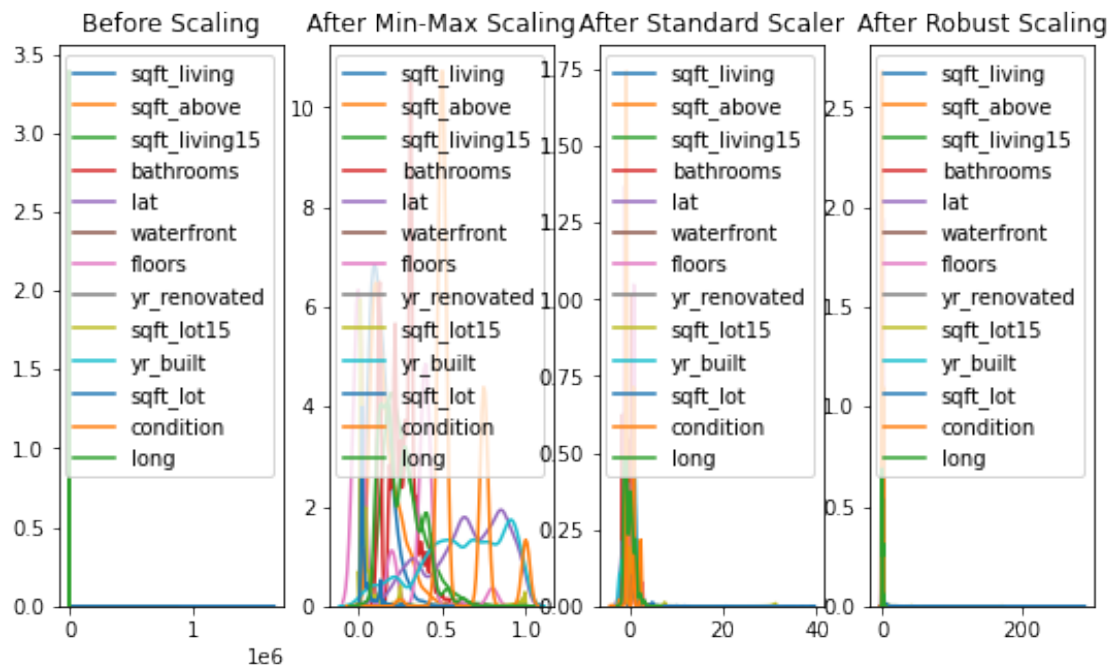
```

```

/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/SherienHassan/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)

```

```
warnings.warn(msg, UserWarning)
```



```
[20]: # Normalization
# 'waterfront', 'floors', 'yr_renovated', 'sqft_lot', '
→ 'sqft_lot15', 'yr_built', 'condition', 'long'
x_array = np.array(data['sqft_living'])
normalized_sqft_living = preprocessing.normalize([x_array])

x_array = np.array(data['grade'])
normalized_grade = preprocessing.normalize([x_array])

x_array = np.array(data['sqft_above'])
normalized_sqft_above = preprocessing.normalize([x_array])

x_array = np.array(data['sqft_living15'])
normalized_sqft_living15 = preprocessing.normalize([x_array])

x_array = np.array(data['bathrooms'])
normalized_bathrooms = preprocessing.normalize([x_array])

x_array = np.array(data['lat'])
normalized_lat = preprocessing.normalize([x_array])

x_array = np.array(data['waterfront'])
normalized_waterfront = preprocessing.normalize([x_array])
```

```

x_array = np.array(data['floors'])
normalized_floors = preprocessing.normalize([x_array])

x_array = np.array(data['yr_renovated'])
normalized_yr_renovated = preprocessing.normalize([x_array])

x_array = np.array(data['sqft_lot'])
normalized_sqft_lot = preprocessing.normalize([x_array])

x_array = np.array(data['sqft_lot15'])
normalized_sqft_lot15 = preprocessing.normalize([x_array])

x_array = np.array(data['yr_built'])
normalized_yr_built = preprocessing.normalize([x_array])

x_array = np.array(data['condition'])
normalized_condition = preprocessing.normalize([x_array])

x_array = np.array(data['long'])
normalized_long = preprocessing.normalize([x_array])

```

```

↳ -----
ValueError                                Traceback (most recent call↳
↳ last)

<ipython-input-20-b9333458e3cf> in <module>
      8
      9 x_array = np.array(data['sqft_above'])
----> 10 normalized_sqft_above = preprocessing.normalize([x_array])
      11
      12 x_array = np.array(data['sqft_living15'])

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py↳
↳ in inner_f(*args, **kwargs)
      71                                     FutureWarning)
      72         kwargs.update({k: arg for k, arg in zip(sig.parameters,↳
↳ args)})
----> 73         return f(**kwargs)
      74     return inner_f
      75

```

```

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/preprocessing/_data.
py in normalize(X, norm, axis, copy, return_norm)
1708         raise ValueError("'d' is not a supported axis" % axis)
1709
-> 1710     X = check_array(X, accept_sparse=sparse_format, copy=copy,
1711                        estimator='the normalize function',
dtype=FLOAT_DTYPES)
1712     if axis == 0:

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py
in inner_f(*args, **kwargs)
71         FutureWarning)
72         kwargs.update({k: arg for k, arg in zip(sig.parameters,
args)})
---> 73         return f(**kwargs)
74     return inner_f
75

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py
in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples,
ensure_min_features, estimator)
643
644     if force_all_finite:
--> 645         _assert_all_finite(array,
646                             allow_nan=force_all_finite ==
'allow-nan')
647

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py
in _assert_all_finite(X, allow_nan, msg_dtype)
95         not allow_nan and not np.isfinite(X).all()):
96         type_err = 'infinity' if allow_nan else 'NaN, infinity'
---> 97         raise ValueError(
98             msg_err.format
99             (type_err,

ValueError: Input contains NaN, infinity or a value too large for
dtype('float64').

```

```
[48]: #There is an issue with this:Input contains NaN, infinity or a value too large,  
      →for dtype('float64') So we will fix this here  
      #x_array = np.array(df1['sqft_above'])  
      #normalized_sqft_above = preprocessing.normalize([x_array])  
  
      #Lets check for null values!  
      data['sqft_above'].isnull().values.any()
```

[48]: True

```
[49]: #There are various ways we can go about this, but because the data is so large,  
      →I do not think it would matter which approach we take  
      #I decided to fill it in with the random number: 1200  
      data['sqft_above'] = data['sqft_above'].fillna(1200)  
      data['sqft_above'].isnull().values.any()
```

[49]: False

```
[53]: # Normalization  
      #'waterfront', 'floors', 'yr_renovated', 'sqft_lot',  
      →'sqft_lot15', 'yr_built', 'condition', 'long'  
      x_array = np.array(data['sqft_living'])  
      normalized_sqft_living = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['grade'])  
      normalized_grade = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['sqft_above'])  
      normalized_sqft_above = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['sqft_living15'])  
      normalized_sqft_living15 = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['bathrooms'])  
      normalized_bathrooms = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['lat'])  
      normalized_lat = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['waterfront'])  
      normalized_waterfront = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['floors'])  
      normalized_floors = preprocessing.normalize([x_array])  
  
      x_array = np.array(data['yr_renovated'])  
      normalized_yr_renovated = preprocessing.normalize([x_array])
```

```

x_array = np.array(data['sqft_lot'])
normalized_sqft_lot = preprocessing.normalize([x_array])

x_array = np.array(data['sqft_lot15'])
normalized_sqft_lot15 = preprocessing.normalize([x_array])

x_array = np.array(data['yr_built'])
normalized_yr_built = preprocessing.normalize([x_array])

x_array = np.array(data['condition'])
normalized_condition = preprocessing.normalize([x_array])

x_array = np.array(data['long'])
normalized_long = preprocessing.normalize([x_array])

#No errors this time!

```

```

[54]: # Dividing the data into training and testing set
X = features
y=data['price']
y=np.log(y)

#note, normalization and standarization were both attempted, but no effects_
→were seen on improving the model more than it is now

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

```

```

[55]: #Multivariate Linear Regression Model
Linear_Regression = LinearRegression()
Linear_Regression_Model = Linear_Regression.fit(X_train, y_train)

Linear_Regression_preds = Linear_Regression_Model.predict(X_test)

rmse1 = np.sqrt(mean_squared_error(y_test, Linear_Regression_preds))
print('RMSE: ', rmse1)
rsq1 = r2_score(y_test, Linear_Regression_preds)
print('R2 Score: ', rsq1)
mae1 = mean_absolute_error(y_test, Linear_Regression_preds)
print('MAE: ', mae1)
plt.scatter(y_test, Linear_Regression_preds)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Multivariate Linear Regression")

```

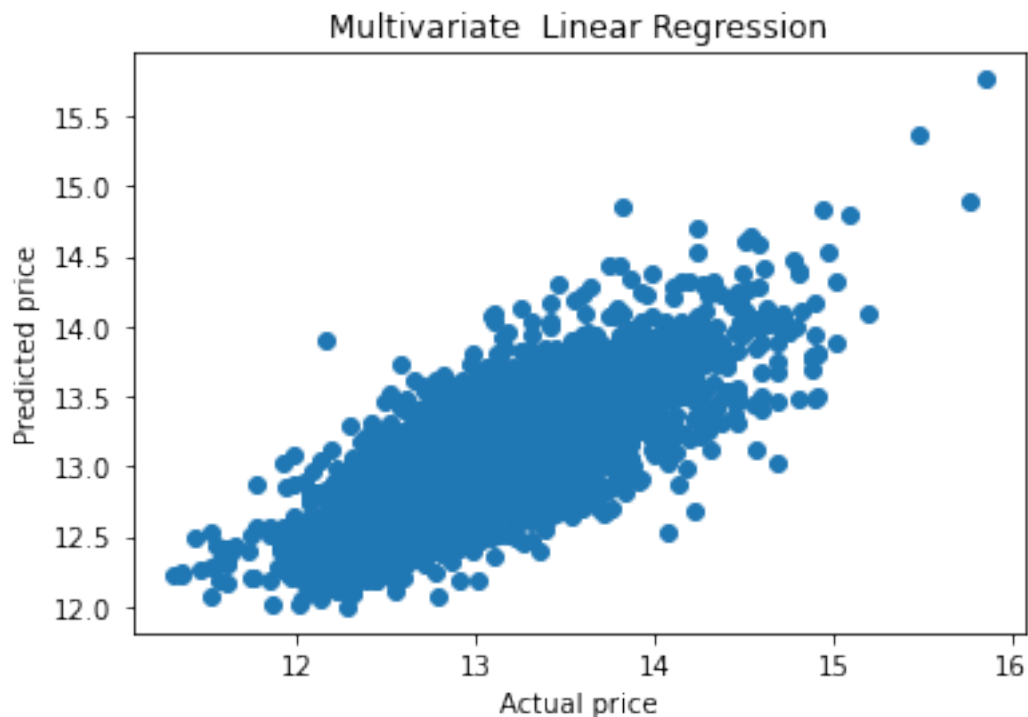
```

RMSE: 0.3361510551443762
R2 Score: 0.596933551393023

```

MAE: 0.25891335450495595

[55]: Text(0.5, 1.0, 'Multivariate Linear Regression')



```
[80]: #Ridge Regression Model(alpha 2.0)
Ridge_Regression = Ridge(alpha=2.0)
Ridge_Regression_model = Ridge_Regression.fit(X_train, y_train)

Ridge_Regression_preds = Ridge_Regression_model.predict(X_test)

rmse2 = np.sqrt(mean_squared_error(y_test, Ridge_Regression_preds))
print('RMSE: ', rmse2)
rsq2 = r2_score(y_test, Ridge_Regression_preds)
print('R2 Score: ', rsq2)
mae2 = mean_absolute_error(y_test, Ridge_Regression_preds)
print('MAE: ', mae2)

#Ridge Regression Model(alpha 1.0)
Ridge_Regression = Ridge(alpha=1.0)
Ridge_Regression_model = Ridge_Regression.fit(X_train, y_train)

Ridge_Regression_preds = Ridge_Regression_model.predict(X_test)

rmse21 = np.sqrt(mean_squared_error(y_test, Ridge_Regression_preds))
```



```

print('RMSE: ', rmse21)
rsq21 = r2_score(y_test, Ridge_Regression_preds)
print('R2 Score: ', rsq21)
mae21 = mean_absolute_error(y_test, Ridge_Regression_preds)
print('MAE: ', mae21)

#Ridge Regression Model(alpha 0.5)
Ridge_Regression = Ridge(alpha=0.5)
Ridge_Regression_model = Ridge_Regression.fit(X_train, y_train)

Ridge_Regression_preds = Ridge_Regression_model.predict(X_test)

rmse22 = np.sqrt(mean_squared_error(y_test, Ridge_Regression_preds))
print('RMSE: ', rmse22)
rsq22 = r2_score(y_test, Ridge_Regression_preds)
print('R2 Score: ', rsq22)
mae22 = mean_absolute_error(y_test, Ridge_Regression_preds)
print('MAE: ', mae22)

scores = {
    'Alpha': ['2', '1', '.5'],

    'RMSE': [rmse2,rmse21,rmse22],

    'R2 Score': [rsq2,rsq21,rsq22],

    'MAE': [mae2,mae21, mae22]
}

col = ['Alpha', 'RMSE', 'R2 Score', 'MAE']

error_matrix = pd.DataFrame(data=scores, columns=col).sort_values(by='MAE',
↪ascending=True).reset_index()
error_matrix.drop(columns=['index'], inplace=True)

error_matrix

```

```

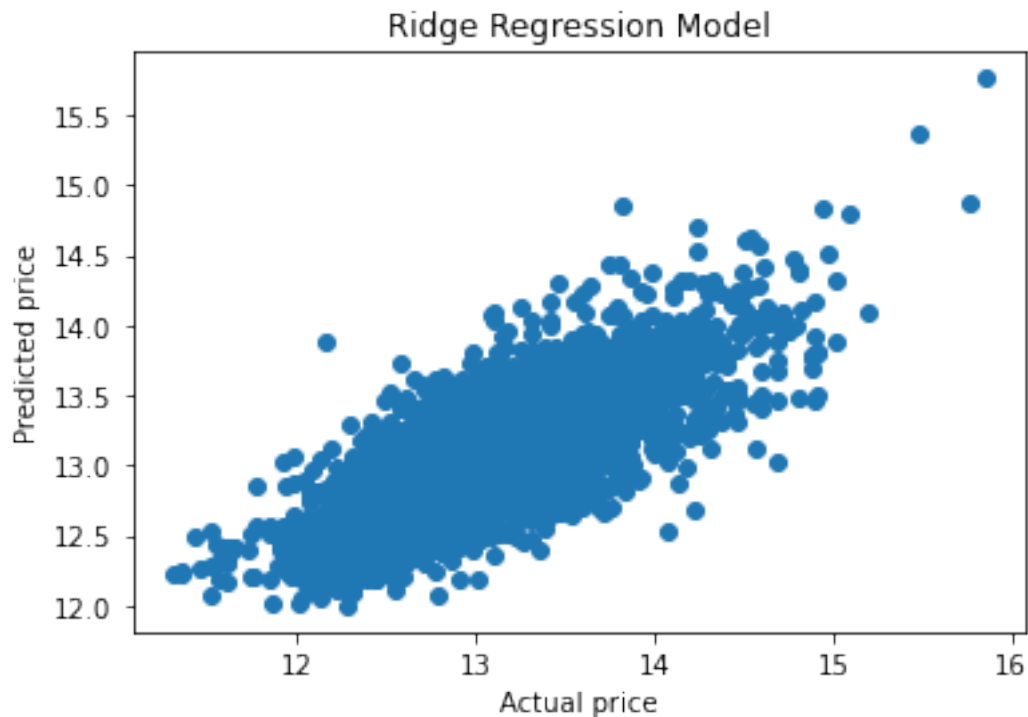
RMSE:  0.3347906397773514
R2 Score:  0.5950868383734451
MAE:  0.2578837705528117
RMSE:  0.33476958567120185
R2 Score:  0.5951377646277263
MAE:  0.25783359787683474
RMSE:  0.33475963841424705
R2 Score:  0.5951618242139389
MAE:  0.2578086387946662

```

```
[80]: Alpha      RMSE  R2 Score      MAE
      0    .5  0.334760  0.595162  0.257809
      1     1  0.334770  0.595138  0.257834
      2     2  0.334791  0.595087  0.257884
```

```
[57]: plt.scatter(y_test, Ridge_Regression_preds)
      plt.xlabel("Actual price")
      plt.ylabel("Predicted price")
      plt.title("Ridge Regression Model")
```

```
[57]: Text(0.5, 1.0, 'Ridge Regression Model')
```



```
[58]: #Random Forest Model
Random_Forest = RandomForestRegressor(max_depth=None, min_samples_split=2,
    ↪ min_samples_leaf=1)
Random_Forest_Model = Random_Forest.fit(X_train, y_train)

Random_Forest_preds = Random_Forest_Model.predict(X_test)

rmse3 = np.sqrt(mean_squared_error(y_test, Random_Forest_preds))
print('RMSE: ', rmse3)
rsq3 = r2_score(y_test, Random_Forest_preds)
print('R2 Score: ', rsq3)
```

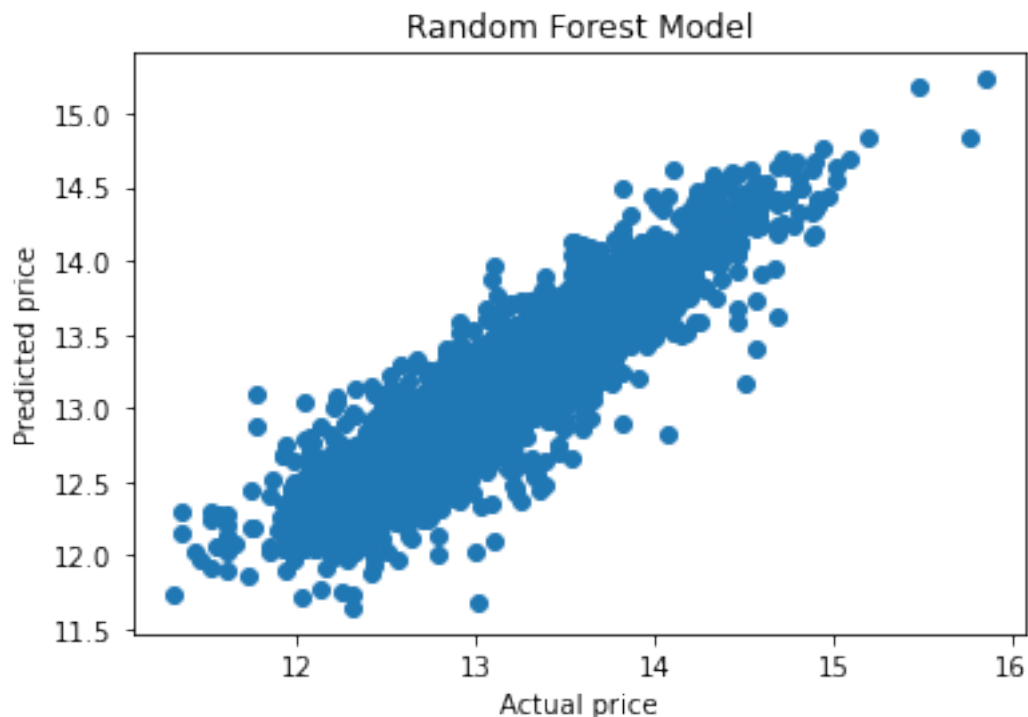
```

mae3 = mean_absolute_error(y_test, Random_Forest_preds)
print('MAE: ', mae3)
plt.scatter(y_test, Random_Forest_preds)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Random Forest Model")
#changing the parameters in this case did not change the accuracy. It might
→have to do with the actual data before being put into this model

```

RMSE: 0.21147053870365048  
 R2 Score: 0.8404828306363288  
 MAE: 0.153169285871099

[58]: Text(0.5, 1.0, 'Random Forest Model')



```

[59]: #Decision Tree Regression Model
Decision_Tree = DecisionTreeRegressor(criterion='mse', splitter='best',
    →max_depth=None, min_samples_split=2, min_samples_leaf=1,)
Decision_Tree_Model = Decision_Tree.fit(X_train, y_train)

Decision_Tree_Model_preds = Decision_Tree_Model.predict(X_test)

rmse4 = np.sqrt(mean_squared_error(y_test, Decision_Tree_Model_preds))

```

```

print('RMSE: ', rmse4)
rsq4 = r2_score(y_test, Decision_Tree_Model_preds)
print('R2 Score: ', rsq4)
mae4 = mean_absolute_error(y_test, Decision_Tree_Model_preds)
print('MAE: ', mae4)
plt.scatter(y_test, Decision_Tree_Model_preds)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Decision Tree Regression Model")
#changing the parameters in this case did not change the accuracy. It might
→have to do with the actual data before being put into this model

```

RMSE: 0.28883817062338357

R2 Score: 0.7024110078106671

MAE: 0.20902646373030037

[59]: Text(0.5, 1.0, 'Decision Tree Regression Model')



```

[69]: #Support Vector Regression Model(c=1.0)
Support_Vector_Regression = SVR(gamma=1, C=.001)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
→y_train)

```

```

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse5 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse5)
rsq5 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq5)
mae5 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae5)

#Support Vector Regression Model(c=1)
Support_Vector_Regression = SVR(gamma=1.0, C=.01)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse51 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse51)
rsq51 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq51)
mae51 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae51)

#Support Vector Regression Model(c=1)
Support_Vector_Regression = SVR(gamma=1, C=10.0)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse52 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse52)
rsq52 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq52)
mae52 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae52)

#Support Vector Regression Model(c=10)
Support_Vector_Regression = SVR(gamma=1.0, C=1.0)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

```

```
Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse53 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse53)
rsq53 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq53)
mae53 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae53)
```

```
RMSE: 366124.3075860474
R2 Score: -0.06144156011328139
MAE: 220129.51848259591
RMSE: 366124.3053745621
R2 Score: -0.06144154729051965
MAE: 220129.5162806445
RMSE: 366121.64338307123
R2 Score: -0.06142611243381335
MAE: 220127.10056806626
RMSE: 366124.0462249328
R2 Score: -0.061440044674702765
MAE: 220129.27679016447
```

```
[70]: scores = {
        'C': ['.001', '.01', '10.0', '1.0'],
        'RMSE': [rmse5, rmse51, rmse52, rmse53],
        'R2 Score': [rsq5, rsq51, rsq52, rsq53],
        'MAE': [mae5, mae51, mae52, mae53]
    }

col = ['C', 'RMSE', 'R2 Score', 'MAE']

error_matrix = pd.DataFrame(data=scores, columns=col).sort_values(by='MAE',
    ↪ascending=True).reset_index()
error_matrix.drop(columns=['index'], inplace=True)

error_matrix
```

```
[70]:
```

	C	RMSE	R2 Score	MAE
0	10.0	366121.643383	-0.061426	220127.100568
1	1.0	366124.046225	-0.061440	220129.276790
2	.01	366124.305375	-0.061442	220129.516281
3	.001	366124.307586	-0.061442	220129.518483

```

[71]: #Support Vector Regression Model(gamma=1)
Support_Vector_Regression = SVR(gamma=1, C=1.0)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse5 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse5)
rsq5 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq5)
mae5 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae5)

#Support Vector Regression Model(gamma=10.)
Support_Vector_Regression = SVR(gamma=10.0, C=1)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse51 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse51)
rsq51 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq51)
mae51 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae51)

#Support Vector Regression Model(gamma=.01)
Support_Vector_Regression = SVR(gamma=.01, C=1.0)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse52 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse52)
rsq52 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq52)
mae52 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae52)

#Support Vector Regression Model(gamma=.001)
Support_Vector_Regression = SVR(gamma=.001, C=1.0)

```

```

Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
↪predict(X_test)

rmse53 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse53)
rsq53 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq53)
mae53 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae53)

```

```

RMSE: 366124.0462249328
R2 Score: -0.061440044674702765
MAE: 220129.27679016447
RMSE: 366124.1522482343
R2 Score: -0.061440659424786936
MAE: 220129.4328591825
RMSE: 366123.761244609
R2 Score: -0.06143839228714154
MAE: 220128.4425958709
RMSE: 366122.8481854245
R2 Score: -0.0614330981491098
MAE: 220127.26865998548

```

```

[81]: scores = {
        'Gamma': ['1.0', '10.0', '.01', '.001'],

        'RMSE': [rmse5, rmse51, rmse52, rmse53],

        'R2 Score': [rsq5, rsq51, rsq52, rsq53],

        'MAE': [mae5, mae51, mae52, mae53]
    }

col = ['Gamma', 'RMSE', 'R2 Score', 'MAE']

error_matrix = pd.DataFrame(data=scores, columns=col).sort_values(by='MAE',
↪ascending=True).reset_index()
error_matrix.drop(columns=['index'], inplace=True)

error_matrix

```

```

[81]:   Gamma      RMSE  R2 Score      MAE
0    1.0    0.512341  0.051726    0.399704
1   .001  366122.848185 -0.061433  220127.268660

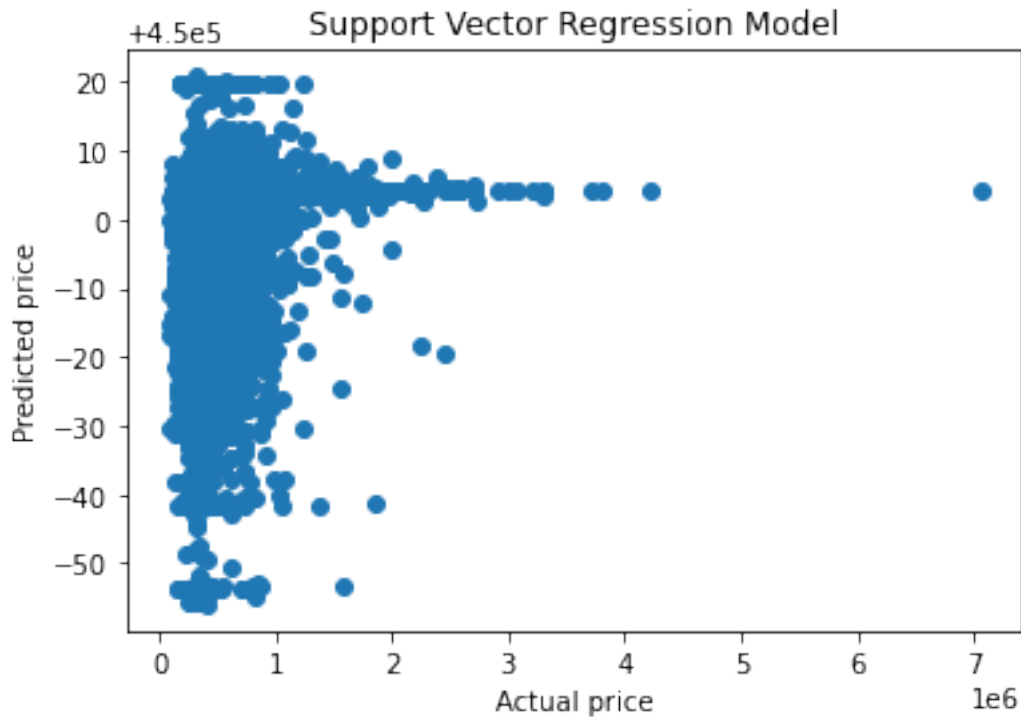
```



```
2 .01 366123.761245 -0.061438 220128.442596
3 10.0 366124.152248 -0.061441 220129.432859
```

```
[73]: plt.scatter(y_test, Support_Vector_Regression_preds)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Support Vector Regression Model")
```

```
[73]: Text(0.5, 1.0, 'Support Vector Regression Model')
```



```
[66]: #Trying without log(Y)
# Dividing the data into training and testing set
X = features
y=data['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

#Support Vector Regression Model
Support_Vector_Regression = SVR(gamma=100.00)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
    ↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)
```

```

rmse = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse)
rsq = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq)
mae = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae)
plt.scatter(y_test, Support_Vector_Regression_preds)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Support Vector Regression Model")

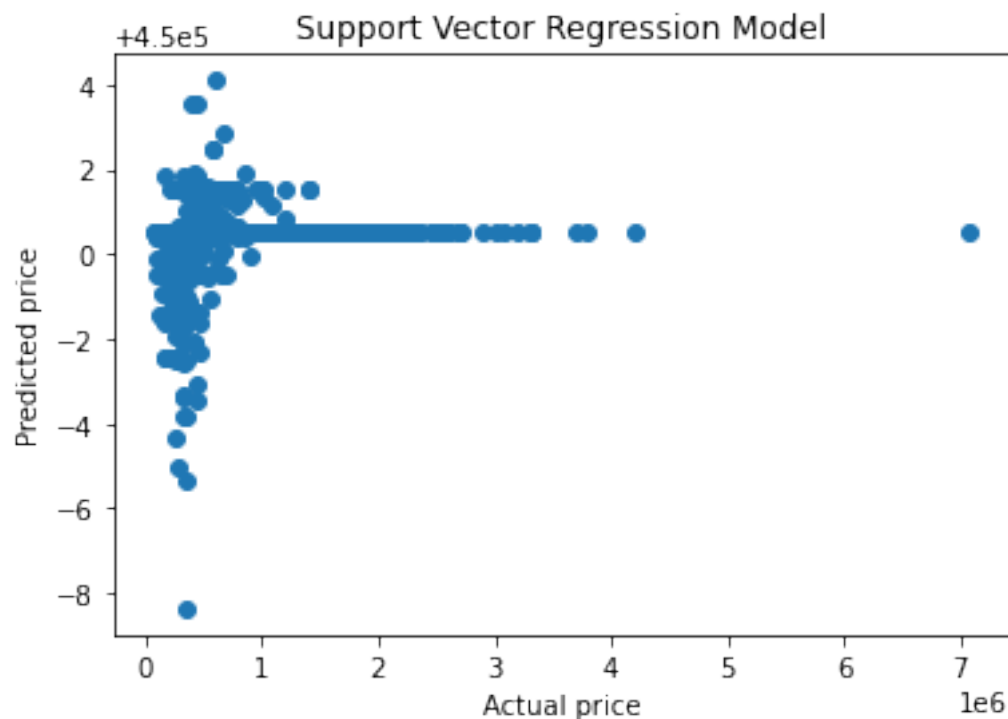
```

RMSE: 366124.17310173565

R2 Score: -0.06144078033871603

MAE: 220129.46686048718

[66]: Text(0.5, 1.0, 'Support Vector Regression Model')



```

[67]: #Support Vector Regression Model
Support_Vector_Regression = SVR(gamma=.0001)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
↪y_train)

```

```

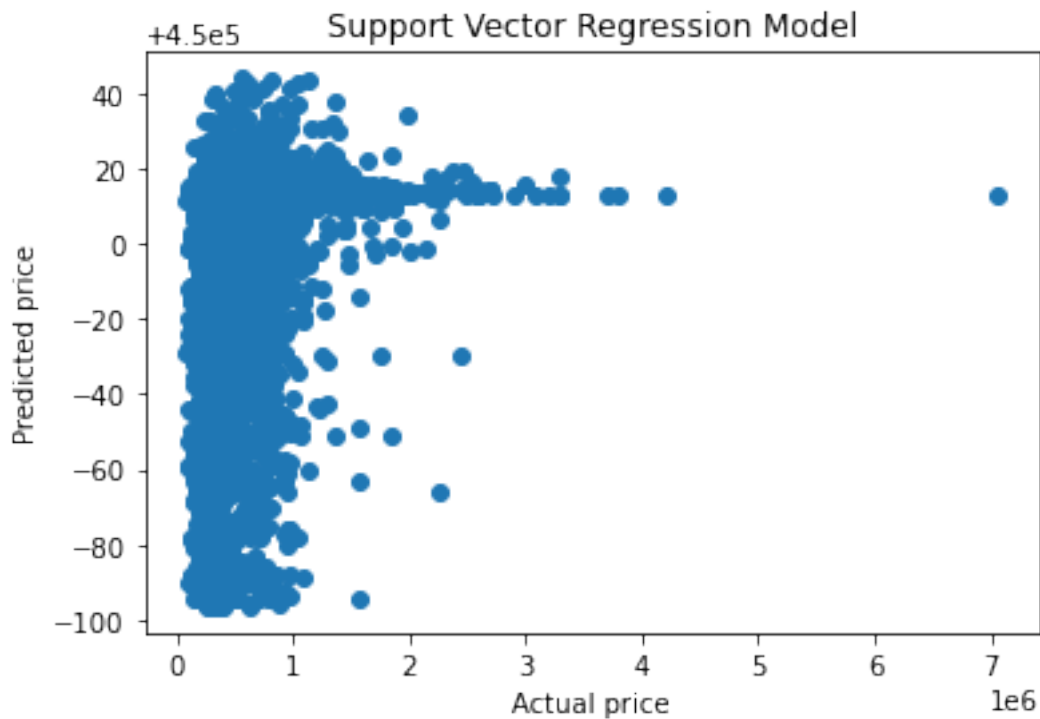
Support_Vector_Regression_preds = Support_Vector_Regression_Model.
    ↪predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse)
rsq = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq)
mae = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae)
plt.scatter(y_test, Support_Vector_Regression_preds)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Support Vector Regression Model")

```

RMSE: 366119.64435764274  
 R2 Score: -0.06141452168502837  
 MAE: 220123.24505602836

[67]: Text(0.5, 1.0, 'Support Vector Regression Model')



```

[77]: data = pd.read_csv('kc_house_data.csv')
      X = features
      y=data['price']

```

```

y=np.log(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
Support_Vector_Regression = SVR(gamma=10.0, C=1.0)
Support_Vector_Regression_Model = Support_Vector_Regression.fit(X_train,
↪y_train)

Support_Vector_Regression_preds = Support_Vector_Regression_Model.
↪predict(X_test)

rmse5 = np.sqrt(mean_squared_error(y_test, Support_Vector_Regression_preds))
print('RMSE: ', rmse5)
rsq5 = r2_score(y_test, Support_Vector_Regression_preds)
print('R2 Score: ', rsq5)
mae5 = mean_absolute_error(y_test, Support_Vector_Regression_preds)
print('MAE: ', mae5)

```

```

RMSE:  0.5123413269941175
R2 Score:  0.051725533801635803
MAE:  0.39970412487460555

```

```

[78]: scores = {
        'Model': ['Linear Regression', 'Ridge Regression', 'Random Forest',
↪Regression', 'Decision Tree Regression',
                'SVR'],

        'RMSE': [rmse1,rmse2,rmse3,rmse4,rmse5],

        'R2 Score': [rsq1,rsq2,rsq3,rsq4,rsq5],

        'MAE': [mae1,mae2, mae3,mae4,mae4]
    }

col = ['Model', 'RMSE', 'R2 Score', 'MAE']

error_matrix = pd.DataFrame(data=scores, columns=col).sort_values(by='MAE',
↪ascending=True).reset_index()
error_matrix.drop(columns=['index'], inplace=True)

error_matrix

```

```

[78]:

```

	Model	RMSE	R2 Score	MAE
0	Random Forest Regression	0.211471	0.840483	0.153169
1	Decision Tree Regression	0.288838	0.702411	0.209026
2	SVR	0.512341	0.051726	0.209026
3	Linear Regression	0.336151	0.596934	0.258913
4	Ridge Regression	0.336177	0.596872	0.258999

