In [ ]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [ ]:
```
# First, we observe the dataset and determine which dataset is relevant to our anal
hdb_df = pd.read_csv("C:\\Users\\KAVIPRIYA\\Desktop\\Resaleflatpricesbasedonregistr
hdb_df.head(10)
```

Out[ ]:

	month	town	flat_type	block	street_name	storey_range	floor_area_sqm	flat_model
0	2017-01	ANG MO KIO	2 ROOM	406	ANG MO KIO AVE 10	10 TO 12	44.0	Improved
1	2017-01	ANG MO KIO	3 ROOM	108	ANG MO KIO AVE 4	01 TO 03	67.0	New Generation
2	2017-01	ANG MO KIO	3 ROOM	602	ANG MO KIO AVE 5	01 TO 03	67.0	New Generation
3	2017-01	ANG MO KIO	3 ROOM	465	ANG MO KIO AVE 10	04 TO 06	68.0	New Generation
4	2017-01	ANG MO KIO	3 ROOM	601	ANG MO KIO AVE 5	01 TO 03	67.0	New Generation
5	2017-01	ANG MO KIO	3 ROOM	150	ANG MO KIO AVE 5	01 TO 03	68.0	New Generation
6	2017-01	ANG MO KIO	3 ROOM	447	ANG MO KIO AVE 10	04 TO 06	68.0	New Generation
7	2017-01	ANG MO KIO	3 ROOM	218	ANG MO KIO AVE 1	04 TO 06	67.0	New Generation
8	2017-01	ANG MO KIO	3 ROOM	447	ANG MO KIO AVE 10	04 TO 06	68.0	New Generation
9	2017-01	ANG MO KIO	3 ROOM	571	ANG MO KIO AVE 3	01 TO 03	67.0	New Generation

In [ ]:
```
# In my analysis, I do not consider street name, block and  flat model is relevant
hdb_df = hdb_df.drop(['month','street_name','flat_model','lease_commence_date', 'bl
```

In [ ]:
```
# Let's rename the column so it will be clearer
hdb_df = hdb_df.rename(columns={'flat_type':'number_of_rooms','storey_range':'store
```

In [ ]:
```
# I assume EXECUTIVE is equal to a 6 room (5 room + 1 study room). MULTI-GENERATION
hdb_df['number_of_rooms'] = hdb_df['number_of_rooms'].str.replace(r'EXECUTIVE','6 R
hdb_df['number_of_rooms'] = hdb_df['number_of_rooms'].str.replace(r'MULTI-GENERATIO
hdb_df['number_of_rooms'] = hdb_df['number_of_rooms'].str.replace(r'ROOM','',regex=
```

In [ ]:
```
# I assume that rather we use floor range, I the possible highest floor within the
hdb_df['storey'] = hdb_df['storey'].str[-2:].astype('int')
```

In [ ]:
```
# I revise the format of the data in the remaining lease to be quantifiable (change
hdb_df['remaining_lease'] = hdb_df['remaining_lease'].str.split(' ')
hdb_df['remaining_lease'] = hdb_df['remaining_lease'].apply(lambda x: (float(x[0])+
hdb_df.head()
```

Out[ ]:

	town	number_of_rooms	storey	floor_area_sqm	remaining_lease	resale_price
0	ANG MO KIO	2	12	44.0	61.333333	232000.0
1	ANG MO KIO	3	3	67.0	60.583333	250000.0
2	ANG MO KIO	3	3	67.0	62.416667	262000.0
3	ANG MO KIO	3	6	68.0	62.083333	265000.0
4	ANG MO KIO	3	3	67.0	62.416667	265000.0

In [ ]:
```
# Observe whether there is missing data or not.
hdb_df.info()
```

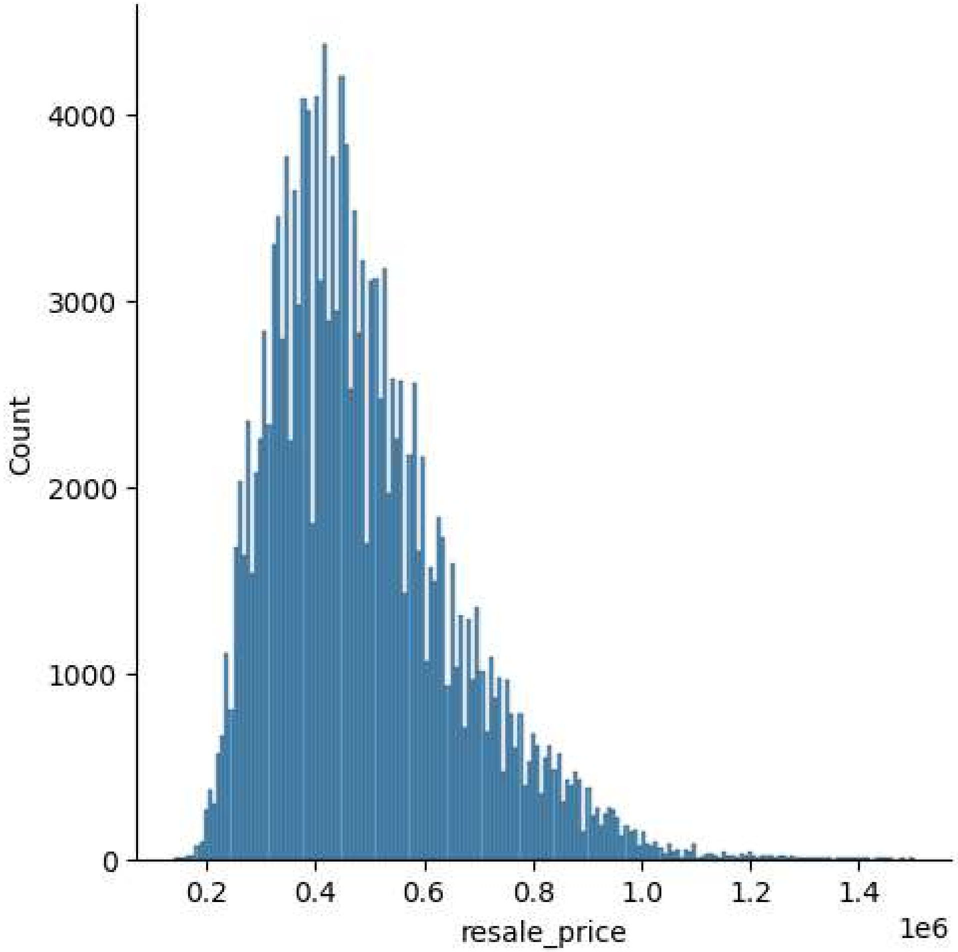
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 165159 entries, 0 to 165158
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   town             165159 non-null  object
 1   number_of_rooms  165159 non-null  int32
 2   storey           165159 non-null  int32
 3   floor_area_sqm   165159 non-null  float64
 4   remaining_lease  165159 non-null  float64
 5   resale_price     165159 non-null  float64
dtypes: float64(3), int32(2), object(1)
memory usage: 6.3+ MB
```

In [ ]:
```
hdb_df.isna().sum()
```

Out[ ]:
```
town               0
number_of_rooms    0
storey             0
floor_area_sqm     0
remaining_lease    0
resale_price       0
dtype: int64
```

In [ ]:
```
# First, we want to se ethe distribution of HDB resale price in Singapore
sns.displot(hdb_df['resale_price'])
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x27883ae6530>



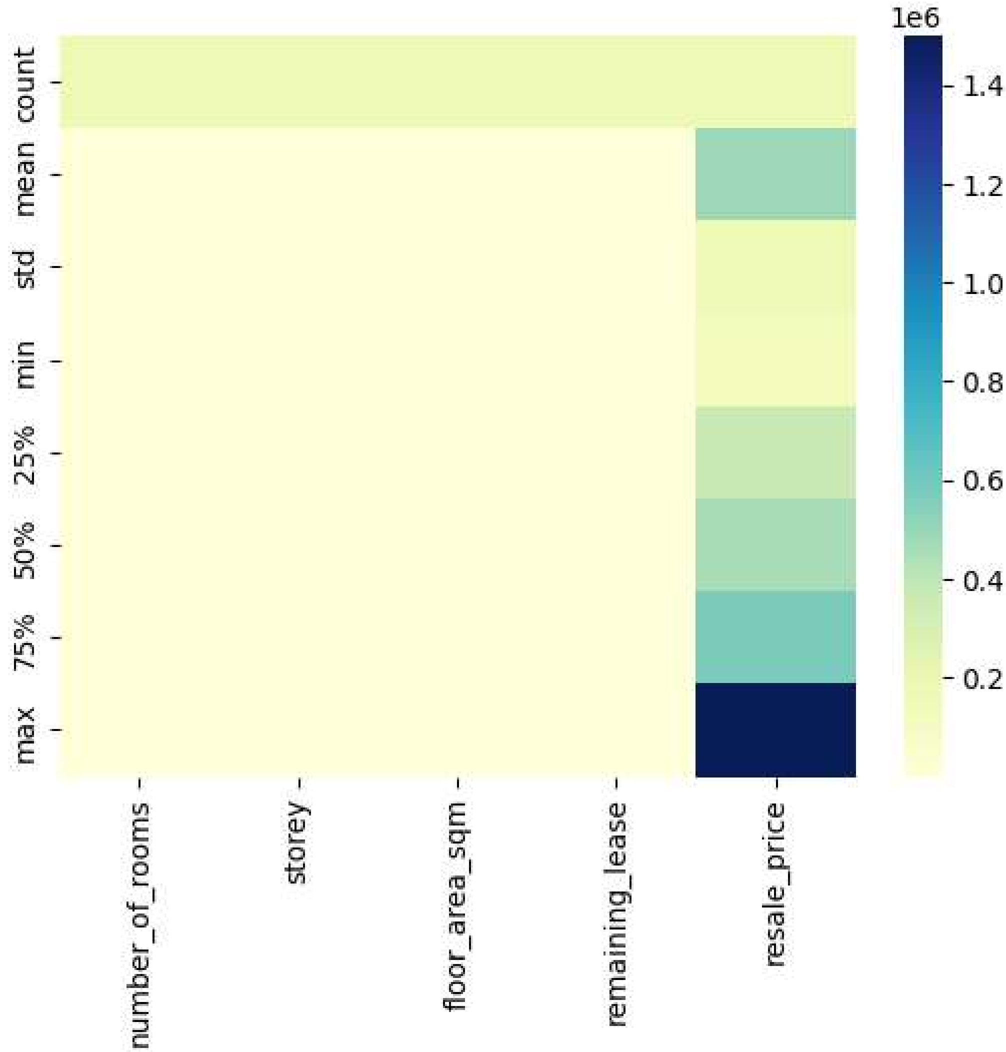
In [ ]:
```
# Let see the statistic information of the data
hdb_df.describe()
```

Out[ ]:

	number_of_rooms	storey	floor_area_sqm	remaining_lease	resale_price
count	165159.000000	165159.000000	165159.000000	165159.000000	1.651590e+05
mean	4.131510	9.768387	97.340378	74.703376	4.888170e+05
std	0.917013	5.948314	24.026982	13.821650	1.691271e+05
min	1.000000	3.000000	31.000000	42.250000	1.400000e+05
25%	3.000000	6.000000	82.000000	63.500000	3.650000e+05
50%	4.000000	9.000000	93.000000	74.666667	4.600000e+05
75%	5.000000	12.000000	112.000000	87.833333	5.800000e+05
max	6.000000	51.000000	249.000000	97.750000	1.500000e+06

In [ ]:
```
# Let us see the relation between each parameters
sns.heatmap(hdb_df.describe(), cmap="YlGnBu")
```

Out[ ]: <Axes: >



In [ ]:
```python
hdb_df['town'].unique()
```

Out[ ]:
```
array(['ANG MO KIO', 'BEDOK', 'BISHAN', 'BUKIT BATOK', 'BUKIT MERAH',
       'BUKIT PANJANG', 'BUKIT TIMAH', 'CENTRAL AREA', 'CHOA CHU KANG',
       'CLEMENTI', 'GEYLANG', 'HOUGANG', 'JURONG EAST', 'JURONG WEST',
       'KALLANG/WHAMPOA', 'MARINE PARADE', 'PASIR RIS', 'PUNGGOL',
       'QUEENSTOWN', 'SEMBAWANG', 'SENGKANG', 'SERANGOON', 'TAMPINES',
       'TOA PAYOH', 'WOODLANDS', 'YISHUN'], dtype=object)
```

In [ ]:
```python
hdb_df = hdb_df.replace(dict.fromkeys(['SEMBAWANG','SENGKANG','WOODLANDS','YISHUN']
hdb_df = hdb_df.replace(dict.fromkeys(['BUKIT MERAH','BUKIT TIMAH','QUEENSTOWN'], '
hdb_df = hdb_df.replace(dict.fromkeys(['BEDOK','GEYLANG','HOUGANG','KALLANG/WHAMPOA
hdb_df = hdb_df.replace(dict.fromkeys(['BUKIT BATOK','BUKIT PANJANG','CHOA CHU KANG
hdb_df = hdb_df.replace(dict.fromkeys(['ANG MO KIO','CENTRAL AREA','BISHAN','MARINE
```

In [ ]:
```python
# Let's check whether the data replacement was done properly
hdb_df['town'].unique()
```

Out[ ]:
```
array(['CENTRAL', 'EAST', 'WEST', 'SOUTH', 'NORTH'], dtype=object)
```

In [ ]:
```python
hdb_df = hdb_df.rename(columns={'town':'region'})
```

In [ ]:
```
X = hdb_df.iloc[:,:-1].values
y = hdb_df.iloc[:,-1].values
```

In [ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder=
X = np.array(ct.fit_transform(X))
```

In [ ]:
```
X[1,:]
```

Out[ ]:
```
array([1.0, 0.0, 0.0, 0.0, 0.0, 3, 3, 67.0, 60.583333333333336],
      dtype=object)
```

In [ ]:
```
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)
```

In [ ]:
```
# Import neccessary library to evaluate the performance of each machine learning mo
from sklearn.metrics import r2_score,mean_absolute_error, mean_squared_error
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
mlr = LinearRegression()
mlr.fit(X_train, y_train)
mlr_ypred = mlr.predict(X_test)
mlr_acc = r2_score(y_test,mlr_ypred)
mlr_acc
```

Out[ ]:
```
0.6669831160774795
```

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
# Let's determine the best degree for polynomial
for n in range(2,5):
    poly_reg = PolynomialFeatures(degree = n)
    X_poly = poly_reg.fit_transform(X_train)
    pr = LinearRegression()
    pr.fit(X_poly, y_train)
    poly_ypred = pr.predict(poly_reg.transform(X_test))
    poly_acc = r2_score(y_test,poly_ypred)
    poly_rmse = np.sqrt(mean_squared_error(y_test,poly_ypred))
    print(r'The accuracy of polynomial regression with degree of {} is {}'.format(n
    print(r'The RMSE of polynomial regression with degree of {} is {}'.format(n,pol
```

```
The accuracy of polynomial regression with degree of 2 is 0.7348388198462854
The RMSE of polynomial regression with degree of 2 is 86941.23746903137
The accuracy of polynomial regression with degree of 3 is 0.7475712228902434
The RMSE of polynomial regression with degree of 3 is 84828.20503583294
The accuracy of polynomial regression with degree of 4 is 0.7570782130310414
The RMSE of polynomial regression with degree of 4 is 83215.47171211109
```

In [ ]:
```
from sklearn.linear_model import Ridge
ridge_r = Ridge()
ridge_r.fit(X_train, y_train)
ridge_ypred = ridge_r.predict(X_test)
```

```
ridge_acc = r2_score(y_test,ridge_ypred)
ridge_acc
```

Out[ ]: 0.6669710360456889

In [ ]:
```
from sklearn.linear_model import Lasso
lasso_r = Lasso(max_iter=100000)
lasso_r.fit(X_train, y_train)
lasso_ypred = lasso_r.predict(X_test)
lasso_acc = r2_score(y_test,lasso_ypred)
lasso_acc
```

Out[ ]: 0.6669711175183721

In [ ]:
```
from sklearn.linear_model import ElasticNet
EN_r = ElasticNet()
EN_r.fit(X_train, y_train)
EN_ypred = EN_r.predict(X_test)
EN_acc = r2_score(y_test,EN_ypred)
EN_acc
```

Out[ ]: 0.575702371290322

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor
tree_r = DecisionTreeRegressor()
tree_r.fit(X_train, y_train)
tree_ypred = tree_r.predict(X_test)
tree_acc = r2_score(y_test,tree_ypred)
tree_acc
```

Out[ ]: 0.7407413949535345

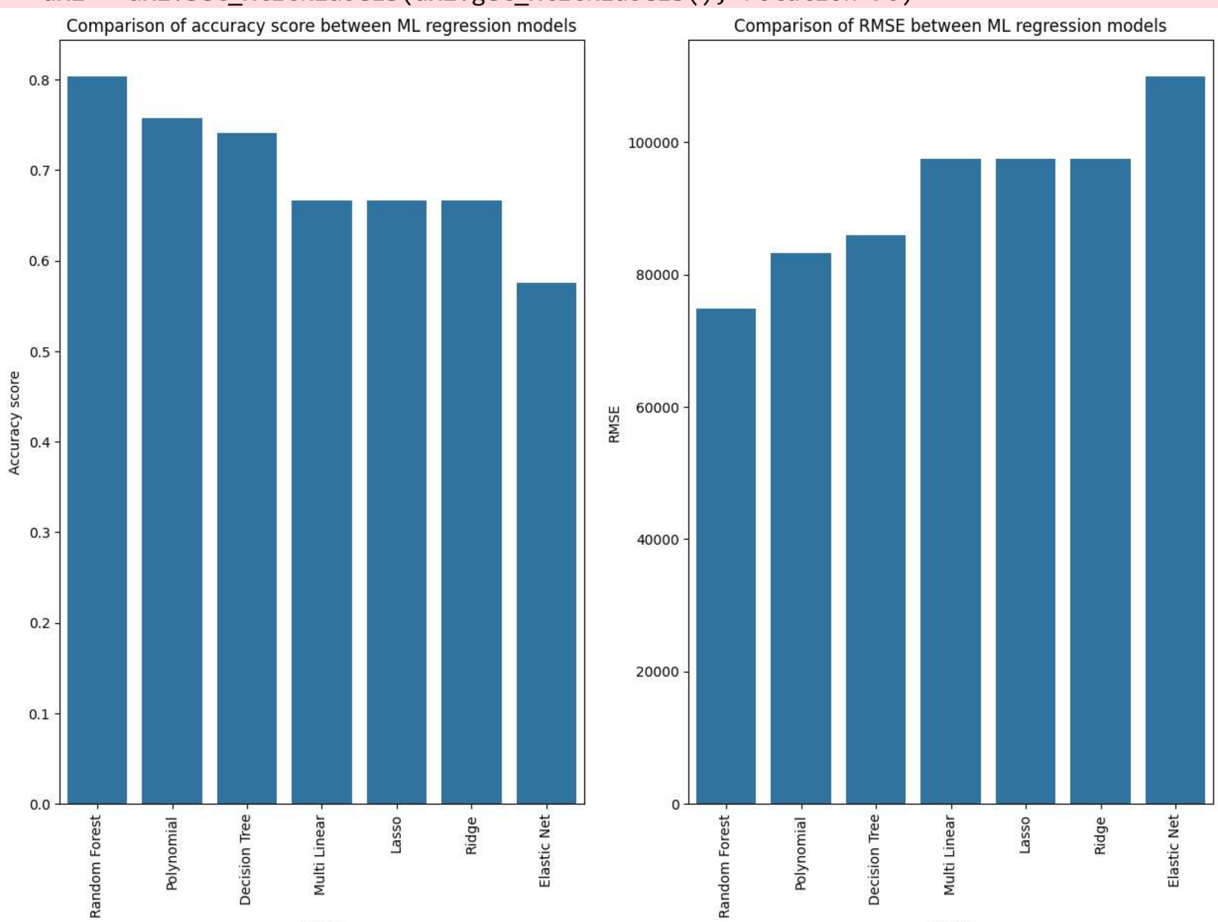
In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
forest_r = RandomForestRegressor(n_estimators = 10)
forest_r.fit(X_train, y_train)
forest_ypred = forest_r.predict(X_test)
forest_acc = r2_score(y_test,forest_ypred)
forest_acc
```

Out[ ]: 0.8037901498807709

In [ ]:
```
# Accuracy score for multi linear regression
mlr_acc = r2_score(y_test,mlr_ypred)
mlr_rmse = np.sqrt(mean_squared_error(y_test,mlr_ypred))
# Evaluation for polynomial regression has been calculated in finding the best degr
# Evaluation for ridge regression
ridge_acc = r2_score(y_test,ridge_ypred)
ridge_rmse = np.sqrt(mean_squared_error(y_test,ridge_ypred))
# Evaluation for lasso regression
lasso_acc = r2_score(y_test,lasso_ypred)
lasso_rmse = np.sqrt(mean_squared_error(y_test,lasso_ypred))
# Evaluation for elastic net regression
EN_acc = r2_score(y_test,EN_ypred)
EN_rmse = np.sqrt(mean_squared_error(y_test,EN_ypred))
# Evaluation for decision trees regression
```

```
tree_acc = r2_score(y_test,tree_ypred)
tree_rmse = np.sqrt(mean_squared_error(y_test,tree_ypred))
# Evaluation for elastic random forest regression
forest_acc = r2_score(y_test,forest_ypred)
forest_rmse = np.sqrt(mean_squared_error(y_test,forest_ypred))
# Let's put it as a list and compare it in a bar chart
model_acc_score = [mlr_acc,poly_acc, ridge_acc, lasso_acc, EN_acc, tree_acc, forest
model_rmse = [mlr_rmse, poly_rmse, ridge_rmse, lasso_rmse, EN_rmse, tree_rmse, fore
model_list = ['Multi Linear', 'Polynomial', 'Ridge', 'Lasso', 'Elastic Net', 'Decis
model_result_df = pd.DataFrame(
    {'Model': model_list,
     'Accuracy score': model_acc_score,
     'RMSE': model_rmse
    })
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,10))
sns.barplot(data=model_result_df, x='Model', y='Accuracy score', ax=ax1,order=model
ax1 = ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
sns.barplot(data=model_result_df, x='Model', y='RMSE', ax=ax2, order=model_result_d
ax2 = ax2.set_xticklabels(ax2.get_xticklabels(), rotation=90)
```

```
C:\Users\KAVIPRIYA\AppData\Local\Temp\ipykernel_11388\1401320706.py:31: UserWarning:
set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ti
cks() or using a FixedLocator.
  ax1 = ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
C:\Users\KAVIPRIYA\AppData\Local\Temp\ipykernel_11388\1401320706.py:33: UserWarning:
set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ti
cks() or using a FixedLocator.
  ax2 = ax2.set_xticklabels(ax2.get_xticklabels(), rotation=90)
```



In [ ]:

```
# We know that after One Hot Encoding, the value of Central, East, North, South, an
area = ['Central','East','North','South','West']
pred_price_central = forest_r.predict([[1.0, 0.0, 0.0, 0.0, 0.0, 4, 9, 95.0, 75]])[
pred_price_east = forest_r.predict([[0.0, 1.0, 0.0, 0.0, 0.0, 4, 9, 95.0, 75]])[0]
pred_price_north = forest_r.predict([[0.0, 0.0, 1.0, 0.0, 0.0, 4, 9, 95.0, 75]])[0]
pred_price_south = forest_r.predict([[0.0, 0.0, 0.0, 1.0, 0.0, 4, 9, 95.0, 75]])[0]
pred_price_west = forest_r.predict([[0.0, 0.0, 0.0, 0.0, 1.0, 4, 9, 95.0, 75]])[0]
resale_price = [pred_price_central,pred_price_east,pred_price_north,pred_price_sout
predict_df = pd.DataFrame(list(zip(area, resale_price)),
              columns=['Area','Predicted HDB price (SGD)'])
predict_df.round()
```

Out[ ]:

	Area	Predicted HDB price (SGD)
0	Central	663200.0
1	East	743189.0
2	North	477300.0
3	South	776139.0
4	West	515900.0