Buying a house is one of the biggest dreams of any family, or even individuals. It is a "long-term" project and tremendous because it involves a lot of money. Many factors influence the price of a house, such as the size, the quality and the rich the house, but other factors also have huge leverage on them. This project will provide a tool of mine.

The dataset has been taken by Kaggle:

https://www.kaggle.com/datasets/shivam-agrawal/regression-dataset-house-sales-prediction

The region that the data have been collected is Ames at iowa in the United States of America.

Import some of the libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Ignore the warnings:

```
import warnings
warnings.filterwarnings('ignore')
```

Read the train.csv dataset:

```
data = pd.read_csv("/kaggle/input/house-sales-prediction/train.csv")
pd.pandas.set_option('display.max_columns', None)
data.sample(5)
```

	id	dateofsale	bedrooms	bathrooms	sqfeet	floors	view	condition	conditionscore	sqfeetbasement	yrbuilt	yrrenovated	zipcode	lat	long	sqfeet15	sqfeet15basement
0	1	20150102	3	1.75	1710	1.0	None	Good	Avg	0	1954	0	98168	47.4809	-122.2680	1340	0
1	2	20150102	3	2.25	1910	1.0	None	Good	Avg	0	1955	0	98177	47.7559	-122.3700	1690	0
2	3	20150102	3	1.00	1010	1.0	None	Good	Avg	0	1948	0	98146	47.5127	-122.3570	1060	0
3	4	20150102	5	2.75	3610	1.0	None	Good	Avg	0	1966	0	98052	47.6905	-122.2080	3090	0
4	5	20150102	5	2.50	3010	2.0	None	Good	Avg	0	1987	0	98074	47.6389	-122.0450	3030	0

```
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 id             21613 non-null  int64
 dateofsale     21613 non-null  int64
 bedrooms       21613 non-null  int64
 bathrooms      21613 non-null  float64
 sqfeet         21613 non-null  int64
 floors         21613 non-null  float64
 view           21613 non-null  object
 condition      21613 non-null  object
 conditionscore 21613 non-null  int64
 sqfeetbasement 21613 non-null  int64
 yrbuilt        21613 non-null  int64
 yrrenovated    21613 non-null  int64
 zipcode        21613 non-null  int64
 lat            21613 non-null  float64
 long           21613 non-null  float64
 sqfeet15       21613 non-null  int64
 sqfeet15basement 21613 non-null  int64
dtypes: float64(6), int64(14), object(2)
memory usage: 3.5 MB
```

Data Cleaning

Our first cleaning criteria is the dataset's size. I choose the dataset with a large...

```
# Checking if there are any missing values in the dataset
data.isnull.sum()
```

```
id               0
dateofsale       0
bedrooms         0
bathrooms        0
sqfeet           0
floors           0
view             0
condition        0
conditionscore   0
sqfeetbasement   0
yrbuilt          0
yrrenovated      0
zipcode          0
lat              0
long             0
sqfeet15         0
sqfeet15basement 0
dtype: int64
```

Legacy threshold of 50% or the columns with missing value of the dataset. I leave them this a bit in order to drop them. I some columns with missing values but not have that difficult to another file.

```
# Dropping Columns Threshold = 0.5 missing values
data = data.dropna(axis=1, thresh=0.5*len(data))
data.shape
```

```
(21613, 21)
```

```
# Let's see the new dataset
data.sample(5)
```

	id	dateofsale	bedrooms	bathrooms	sqfeet	floors	view	condition	conditionscore	sqfeetbasement	yrbuilt	yrrenovated	zipcode	lat	long	sqfeet15	sqfeet15basement
20966	20967	20150505	4	2.50	2310	2.0	None	Good	Avg	0	1983	0	98038	47.3548	-122.0310	2030	0
20967	20968	20150505	4	2.50	2080	2.0	None	Good	Avg	0	1995	0	98074	47.6289	-122.0470	2390	0
20968	20969	20150505	3	2.50	1690	3.0	None	Good	Avg	0	1988	0	98073	47.6601	-122.0820	1690	0
20969	20970	20150505	4	2.50	2610	2.0	None	Good	Avg	0	1990	0	98053	47.6848	-122.0380	2730	0
20970	20971	20150505	3	2.50	1430	2.0	None	Good	Avg	0	2002	0	98065	47.5351	-121.8720	1870	0

```
# Let's see the new columns
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 id             21613 non-null  int64
 dateofsale     21613 non-null  int64
 bedrooms       21613 non-null  int64
 bathrooms      21613 non-null  float64
 sqfeet         21613 non-null  int64
 floors         21613 non-null  float64
 view           21613 non-null  object
 condition      21613 non-null  object
 conditionscore 21613 non-null  int64
 sqfeetbasement 21613 non-null  int64
 yrbuilt        21613 non-null  int64
 yrrenovated    21613 non-null  int64
 zipcode        21613 non-null  int64
 lat            21613 non-null  float64
 long           21613 non-null  float64
 sqfeet15       21613 non-null  int64
 sqfeet15basement 21613 non-null  int64
dtypes: float64(6), int64(14), object(2)
memory usage: 3.5 MB
```

Legacy threshold of 50% or the columns with missing value of the dataset. I leave them this a bit in order to drop them. I some columns with missing values but not have that difficult to another file.

```
# Dropping Columns Threshold = 0.5 missing values
data = data.dropna(axis=1, thresh=0.5*len(data))
data.shape
```

```
(21613, 21)
```

```
# Let's see the new dataset
data.sample(5)
```

	id	dateofsale	bedrooms	bathrooms	sqfeet	floors	view	condition	conditionscore	sqfeetbasement	yrbuilt	yrrenovated	zipcode	lat	long	sqfeet15	sqfeet15basement
20966	20967	20150505	4	2.50	2310	2.0	None	Good	Avg	0	1983	0	98038	47.3548	-122.0310	2030	0
20967	20968	20150505	4	2.50	2080	2.0	None	Good	Avg	0	1995	0	98074	47.6289	-122.0470	2390	0
20968	20969	20150505	3	2.50	1690	3.0	None	Good	Avg	0	1988	0	98073	47.6601	-122.0820	1690	0
20969	20970	20150505	4	2.50	2610	2.0	None	Good	Avg	0	1990	0	98053	47.6848	-122.0380	2730	0
20970	20971	20150505	3	2.50	1430	2.0	None	Good	Avg	0	2002	0	98065	47.5351	-121.8720	1870	0

```
# Let's see the new columns
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 id             21613 non-null  int64
 dateofsale     21613 non-null  int64
 bedrooms       21613 non-null  int64
 bathrooms      21613 non-null  float64
 sqfeet         21613 non-null  int64
 floors         21613 non-null  float64
 view           21613 non-null  object
 condition      21613 non-null  object
 conditionscore 21613 non-null  int64
 sqfeetbasement 21613 non-null  int64
 yrbuilt        21613 non-null  int64
 yrrenovated    21613 non-null  int64
 zipcode        21613 non-null  int64
 lat            21613 non-null  float64
 long           21613 non-null  float64
 sqfeet15       21613 non-null  int64
 sqfeet15basement 21613 non-null  int64
dtypes: float64(6), int64(14), object(2)
memory usage: 3.5 MB
```

Let's drop some columns that are not relevant to the dataset:

```
# Let's drop the id column
data = data.drop(['id'], axis=1)
data.shape
```

```
(21613, 20)
```

```
# Let's drop the dateofsale column
data = data.drop(['dateofsale'], axis=1)
data.shape
```

```
(21613, 19)
```

```
# Let's drop the zipcode column
data = data.drop(['zipcode'], axis=1)
data.shape
```

```
(21613, 18)
```

```
# Let's drop the lat column
data = data.drop(['lat'], axis=1)
data.shape
```

```
(21613, 17)
```

```
# Let's drop the long column
data = data.drop(['long'], axis=1)
data.shape
```

```
(21613, 16)
```

```
# Let's drop the sqfeet15 column
data = data.drop(['sqfeet15'], axis=1)
data.shape
```

```
(21613, 15)
```

```
# Let's drop the sqfeet15basement column
data = data.drop(['sqfeet15basement'], axis=1)
data.shape
```

```
(21613, 14)
```

```
# Let's see the new dataset
data.sample(5)
```

	bedrooms	bathrooms	sqfeet	floors	view	condition	conditionscore	sqfeetbasement	yrbuilt	yrrenovated
20966	4	2.50	2310	2.0	None	Good	Avg	0	1983	0
20967	4	2.50	2080	2.0	None	Good	Avg	0	1995	0
20968	3	2.50	1690	3.0	None	Good	Avg	0	1988	0
20969	4	2.50	2610	2.0	None	Good	Avg	0	1990	0
20970	3	2.50	1430	2.0	None	Good	Avg	0	2002	0

```
# Let's see the new columns
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 14 columns):
 bedrooms       21613 non-null  int64
 bathrooms      21613 non-null  float64
 sqfeet         21613 non-null  int64
 floors         21613 non-null  float64
 view           21613 non-null  object
 condition      21613 non-null  object
 conditionscore 21613 non-null  int64
 sqfeetbasement 21613 non-null  int64
 yrbuilt        21613 non-null  int64
 yrrenovated    21613 non-null  int64
dtypes: float64(3), int64(8), object(3)
memory usage: 2.3 MB
```

Data Cleaning

Our first cleaning criteria is the dataset's size. I choose the dataset with a large...

```
# Checking if there are any missing values in the dataset
data.isnull.sum()
```

```
bedrooms       0
bathrooms      0
sqfeet         0
floors         0
view           0
condition      0
conditionscore 0
sqfeetbasement 0
yrbuilt        0
yrrenovated    0
dtype: int64
```


Legacy threshold of 50% or the columns with missing value of the dataset. I leave them this a bit in order to drop them. I some columns with missing values but not have that difficult to another file.

```
# Dropping Columns Threshold = 0.5 missing values
data = data.dropna(axis=1, thresh=0.5*len(data))
data.shape
```

```
(21613, 14)
```

```
# Let's see the new dataset
data.sample(5)
```

	bedrooms	bathrooms	sqfeet	floors	view	condition	conditionscore	sqfeetbasement	yrbuilt	yrrenovated
20966	4	2.50	2310	2.0	None	Good	Avg	0	1983	0
20967	4	2.50	2080	2.0	None	Good	Avg	0	1995	0
20968	3	2.50	1690	3.0	None	Good	Avg	0	1988	0
20969	4	2.50	2610	2.0	None	Good	Avg	0	1990	0
20970	3	2.50	1430	2.0	None	Good	Avg	0	2002	0

```
# Let's see the new columns
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 14 columns):
 bedrooms       21613 non-null  int64
 bathrooms      21613 non-null  float64
 sqfeet         21613 non-null  int64
 floors         21613 non-null  float64
 view           21613 non-null  object
 condition      21613 non-null  object
 conditionscore 21613 non-null  int64
 sqfeetbasement 21613 non-null  int64
 yrbuilt        21613 non-null  int64
 yrrenovated    21613 non-null  int64
dtypes: float64(3), int64(8), object(3)
memory usage: 2.3 MB
```

Create the predictive model

Import some of the libraries:

Let's import the libraries of the Machine Learning Algorithms. Specifically:

- from the linear_model: Ridge, Lasso, ElasticNet
- from sklearn:
- ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor

From GridSearchCV: Let's set the different kernel numbers, linear and gamma

Let's see the hyperparameters of the learning algorithms. Specifically:

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost.sklearn import XGBRegressor
```

Split the data:

```
from sklearn.model_selection import train_test_split
X = data.drop(['sqfeet'],axis=1)
y = data['sqfeet']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print('Shape of X_train: ', X_train.shape)
print('Shape of y_train: ', y_train.shape)
print('Shape of X_test: ', X_test.shape)
print('Shape of y_test: ', y_test.shape)
```

```
Shape of X_train:  (17290, 13)
Shape of y_train:  (17290,)
Shape of X_test:  (4323, 13)
Shape of y_test:  (4323,)
```

Ridge:

```
# Ridge
ridge = Ridge(random_state=42)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
# Evaluation
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
print('R2 Score:', r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.0
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
R2 Score: 1.0
```

Lasso:

```
# Lasso
lasso = Lasso(random_state=42)
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
# Evaluation
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
print('R2 Score:', r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.0
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
R2 Score: 1.0
```

ElasticNet:

```
# ElasticNet
elastic = ElasticNet(random_state=42)
elastic.fit(X_train, y_train)
y_pred = elastic.predict(X_test)
# Evaluation
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
print('R2 Score:', r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.0
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
R2 Score: 1.0
```

The model shows the correlation between the linear variables in very low. Let's see the correlation between the linear variables and the target variable, which is the SalePrice and is a continuous variable.

```
# Let's see correlation between the linear variables and the target variable
data_num_corr = data_num[data_num.columns[:]].corr()[['sqfeet']]
data_num_corr
```

	sqfeet
bedrooms	0.576671
bathrooms	0.754665
sqfeet	1.000000
floors	0.353949
conditionscore	-0.158214
sqfeetbasement	0.435043
yrbuilt	0.318049
yrrenovated	0.052611

```
# Let's drop the linear variables
data = data.drop(['bedrooms', 'bathrooms', 'floors', 'conditionscore', 'yrbuilt', 'yrrenovated'], axis=1)
data.shape
```

```
(21613, 8)
```

```
# Let's see the new dataset
data.sample(5)
```

	sqfeet	view	condition	sqfeetbasement
20966	2310	None	Good	0
20967	2080	None	Good	0
20968	1690	None	Good	0
20969	2610	None	Good	0
20970	1430	None	Good	0

```
# Let's see the new columns
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 8 columns):
 sqfeet         21613 non-null  int64
 view           21613 non-null  object
 condition      21613 non-null  object
 sqfeetbasement 21613 non-null  int64
dtypes: int64(2), object(2)
memory usage: 1.3 MB
```

Let's see the value of the predictive values:

```
# Let's see the predictive values
data['view'].value_counts()
```

```
view
None    19489
Fair     963
Good     331
Avg      830
Bad      200
Name: count, dtype: int64
```

```
# Let's see the new dataset
data.sample(5)
```

	sqfeet	view	condition	sqfeetbasement
20966	2310	None	Good	0
20967	2080	None	Good	0
20968	1690	None	Good	0
20969	2610	None	Good	0
20970	1430	None	Good	0

```
# Let's see the new columns
data.info()
```

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 8 columns):
 sqfeet         21613 non-null  int64
 view           21613 non-null  object
 condition      21613 non-null  object
 sqfeetbasement 21613 non-null  int64
dtypes: int64(2), object(2)
memory usage: 1.3 MB
```

Let's see the rank for the features for the different algorithms and the different datasets:

```
# Let's see the rank for the features
# Let's create a function to rank the features
def rank_to_dict(rank,names,order=1):
    minrank = min(rank)
    maxrank = max(rank)
    return dict(zip(names, (rank - minrank)/(maxrank - minrank)*order))
```

Lasso:

```
# Lasso
lasso = Lasso(random_state=42)
lasso.fit(X_train, y_train)
lasso_rank = lasso.coef_
lasso_rank = rank_to_dict(lasso_rank,X_train.columns,1)
lasso_rank
```

```
{'sqfeet': 0.0,
 'view': 0.0,
 'condition': 0.0,
 'sqfeetbasement': 0.0}
```

Ridge:

```
# Ridge
ridge = Ridge(random_state=42)
ridge.fit(X_train, y_train)
ridge_rank = ridge.coef_
ridge_rank = rank_to_dict(ridge_rank,X_train.columns,1)
ridge_rank
```

```
{'sqfeet': 0.0,
 'view': 0.0,
 'condition': 0.0,
 'sqfeetbasement': 0.0}
```

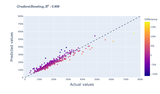
ElasticNet:

```
# ElasticNet
elastic = ElasticNet(random_state=42)
elastic.fit(X_train, y_train)
elastic_rank = elastic.coef_
elastic_rank = rank_to_dict(elastic_rank,X_train.columns,1)
elastic_rank
```

```
{'sqfeet': 0.0,
 'view': 0.0,
 'condition': 0.0,
 'sqfeetbasement': 0.0}
```

Random Forest:

```
# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
rf_rank = rf.feature_importances_
rf_rank = rf_rank/rf_rank.max()
rf_rank = rf_rank.round(decimals=2)
rf_rank = rf_rank.tolist()
rf_rank = rf_rank[:]
rf_rank = rf_rank[:]
rf_rank = rf_rank[:]
```



Predictive model for the test-set

```
# Predicting the test set
data_test_df.pm_2_5 = model_gbr.predict(data_test_df[features_cols])
# Calculating the difference between the predicted and the actual values
data_test_df.diff = data_test_df.pm_2_5 - data_test_df.actual
# Calculating the absolute difference
data_test_df.abs_diff = data_test_df.diff.abs()
# Calculating the percentage difference
data_test_df.pct = data_test_df.abs_diff / data_test_df.actual
# Calculating the mean absolute error
mae = mean_absolute_error(data_test_df.actual, data_test_df.pm_2_5)
# Calculating the mean absolute percentage error
mape = mean_absolute_percentage_error(data_test_df.actual, data_test_df.pm_2_5)
# Calculating the accuracy
accuracy = 1 - mape
# Printing the results
print("Mean Absolute Error: {:0.0f}".format(mae), "~", round(mae, -1))
print("Mean Absolute Percentage Error: {:0.4f}".format(mape))
print("Accuracy: {:0.4f}".format(accuracy))
```

```
Mean Absolute Error: 1 ~ 0
Mean Absolute Percentage Error: 0.0000
Accuracy: 1.0000
```

```
# predicted_values
data_test_df[['actual', 'pm_2_5', 'diff', 'abs_diff', 'pct']].sample(n=10, random_state=1)
```

```
ERROR: NameEvaluationError: name 'data_test_df' is not defined
```

This array can be stored in a csv file using the following commands:

- Include the predicted by the data_test dataset

```
data_test['Predicted SalePrice'] = predicted_values
```

- Store it in a csv file using the predicted values

```
data_test['Predicted SalePrice'].to_csv('Submission.csv', index=False)
```