Predicting House Prices with Machine Learning

Understanding the Client and their Problem

A benefit to this study is that we can have two clients at the same time! (Think of being a divorce lawyer for both interested parties) However, in this case, we can have both clients with no conflict of interest!

Client Housebuyer: This client wants to find their next dream home with a reasonable price tag. They have their locations of interest ready. Now, they want to know if the house price matches the house value. With this study, they can understand which features (ex. Number of bathrooms, location, etc.) influence the final price of the house. If all matches, they can ensure that they are getting a fair price.

Client Houseseller: Think of the average house-flipper. This client wants to take advantage of the features that influence a house price the most. They typically want to buy a house at a low price and invest on the features that will give the highest return. For example, buying a house at a good location but small square footage. The client will invest on making rooms at a small cost to get a large return.

Loading Data and Packages

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
import warnings
import xgboost as xgb
import lightgbm as lgb
from scipy.stats import skew
from scipy import stats
from scipy.stats.stats import pearsonr
from scipy.stats import norm
from collections import Counter
from sklearn.linear_model import LinearRegression,LassoCV, Ridge, LassoLarsCV,ElasticNetCV
from sklearn.model_selection import GridSearchCV, cross_val_score, learning_curve
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor, G
radientBoostingRegressor
from sklearn.preprocessing import StandardScaler, Normalizer, RobustScaler
warnings.filterwarnings('ignore')
sns.set(style='white', context='notebook', palette='deep')
%config InlineBackend.figure_format = 'retina' #set 'png' here when working on notebook
%matplotlib inline
```

```
# Check the numbers of samples and features
print("The train data size before dropping Id feature is : {} ".format(trai
n.shape))
print("The test data size before dropping Id feature is : {} ".format(test.s
hape))

# Save the 'Id' column
train_ID = train['Id']
test_ID = test['Id']

# Now drop the 'Id' column since it's unnecessary for the prediction process.
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)

# Check data size after dropping the 'Id' variable
print("\nThe train data size after dropping Id feature is : {} ".format(trai
n.shape))
print("The test data size after dropping Id feature is : {} ".format(test.sh
ape))
```

```
The train data size before dropping Id feature is : (1460, 81)
```

So the training set has 1460 rows and 81 features. The test set has 1459 rows and 80 features.

```
train.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilitie
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub

```
test.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilitie
0	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub
1	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub
2	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub
3	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub
4	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub

2. Analyzing the Test Variable (Sale Price)

Let's check out the most interesting feature in this study: Sale Price. Important Note: This data is from Ames, Iowa. The location is extremely correlated with Sale Price. (I had to take a double-take at a point, since I consider myself a house-browsing enthusiast)

```
# Getting Description
train['SalePrice'].describe()
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

With an average house price of $180921, it seems like I should relocated to Iowa!

With an average house price of $180921, it seems like I should relocated to Iowa!

]:

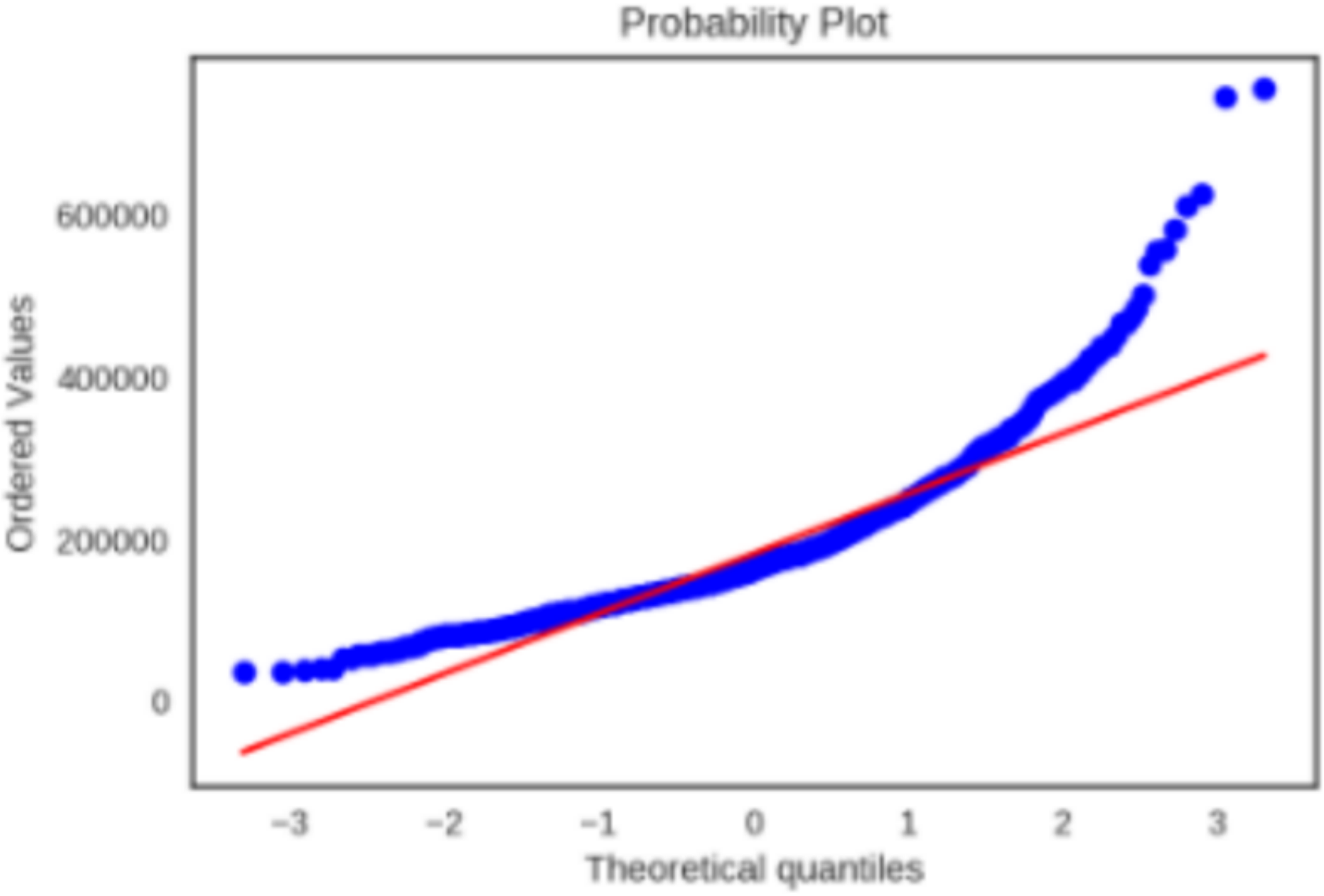
```
# Plot Histogram
sns.distplot(train['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()

print("Skewness: %f" % train['SalePrice'].skew())
print("Kurtosis: %f" % train['SalePrice'].kurt())
```

```
 mu = 180921.20 and sigma = 79415.29
```



Skewness: 1.882876

Kurtosis: 6.536282

Looks like a normal distribution? Not quite! Looking at the kurtosis score, we can see that there is a very nice peak. However, looking at the skewness score, we can see that the sale prices deviate from the normal distribution. Going to have to fix this later! We want our data to be as "normal" as possible.