FLIP ROBO

Housing Price Prediction

Submitted by:
Rupali Bisen

ACKNOWLEDGMENT

"This report is dedicated to my parents,family members and friends, who always inspired me in every step to accomplish this study".

It is a great honor for me to work on the assigned topic
and I feel glad to accomplish the task. Along with my sincerity and interest, there are few people, who really helped me to make this endeavor to be a successful one.

At first, I would like to pass my appreciation, gratitude and thanks to the team
Flip Robo and DataTrained. Their valuable suggestions and ideas in every step
of my work helped me a lot to prepare this report.

I wish to extend my special thanks to my SME, Mentor, career coach for
clearing out my doubts.

Lastly, I would like to acknowledge my referencing the following Internet
resources during the execution of this project.

1. https://www.divaportal.org/smash/get/diva2:546238/FULLTEXT01.pdf
2. https://www8.gsb.columbia.edu/sites/socialenterprise/files/Mobile_Phones_Delivering_MF_200407.pdf
3. www.kaggle.com
4. towardsdatascience.com
5. www.pluralsight.com
6. www.analyticsvidhya.com
7. www.geeksforgeeks.org

INTRODUCTION

BUSINESS PROBLEM FRAMING

This is a real estate problem where a US based housing company named Surprise Housing has decided to invest in Australian Market. Their agenda is to buy houses in Australia at prices below their actual value in the market and sell them at high prices to gain profit. To do this this company uses data analytics to decide in which property they must invest.

Company has collected the data of previously sold houses in Australia and with the help of this data they want to know to the value of prospective properties to decide whether it will suitable to invest in the properties or not.

To know the value of properties company has provided data to us to do data analysis and to extract the information of attributes which are important to predict the price of the houses. They want a machine learning model which can predict the price of houses and also the significance of each important attribute in house prediction i.e, how and to what intensity each variable impacts the price of the house.

CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM

In real estate the value of property usually increases with time as seen in many countries. One of the causes for this is due to rising population. The value of property also depends on the proximity of the property, its size its neighbourhood and audience for which the property is subjected to be sold. For example if audience is mainly concerned of commercial purpose. Then the property which is located in densely populated area will be sold very fast and at high prices compared to the one located at remote place. Similarly if audience is concerned only on living place then property with less dense area having large area with all services will be sold at higher prices.
The company is looking at prospective properties to buy houses to enter the market.
We are required to build a model using Machine Learning in order to predict the actual
value of the prospective properties and decide whether to invest in them or not.
Flip Robo Technologies

REVIEW OF LITERATURE

Houses are one of the necessary needs of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world’s economy. A US-based housing

company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. We are required to build a model using Machine Learning in order to
predict the actual value of the prospective properties and decide whether to invest in
them or not.

With its great weather, cosmopolitan cities, diverse natural landscapes and relaxed lifestyle, it’s no wonder that Australia remains a top pick for expats. Living cost in Australia for one person: $2,835 per month. Average living expenses for a couple: $4,118 per month. Average monthly living expenses for a family of 4: $5,378. Australia currently has the 16th highest cost of living in the world, with the USA and UK well behind at 21st and 33rd place respectively. Sydney and Melbourne are popular choices for expats moving to Australia. House pricing in some of the top Australian
cities:-
Sydney - median house price A$1,142,212
Adelaide- median house price A$542,947
Hobbart (smaller city)- median house price A$530,570.

MOTIVATION FOR THE PROBLEM UNDERTAKEN
To understand real world problems where Machine Learning and Data Analysis can be applied to help organizations in various domains to make better decisions with the help of which they can gain profit or can be escaped from any loss which otherwise could be possible without the study of data .One of such domain is Real Estate.

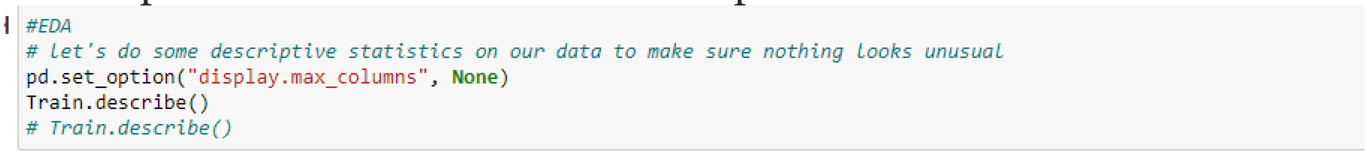
Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world’s economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool Flip Robo Technologies

To solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

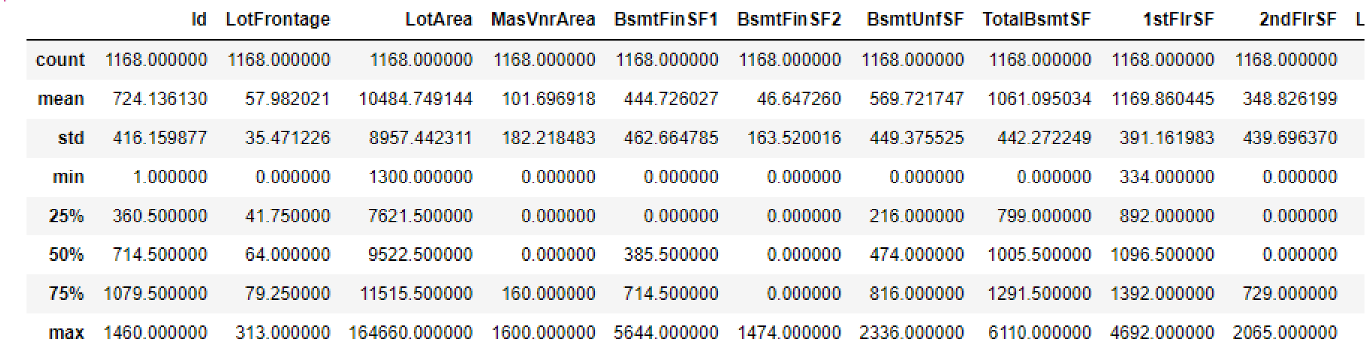
ANALYTICAL PROBLEM FRAMING

MATHEMATICAL/ ANALYTICAL MODELING OF THE PROBLEM

In this project we have performed various mathematical and statistical analysis such as we checked description or statistical summary of the data using describe, checked correlation using corr and also visualized it using heatmap. Then we have used Z-Score to plot outliers and remove them.



```
#EDA
# Let's do some descriptive statistics on our data to make sure nothing looks unusual
pd.set_option("display.max_columns", None)
Train.describe()
# Train.describe()
```



	Id	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	L
count	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	
mean	724.136130	57.982021	10484.749144	101.696918	444.726027	46.647260	569.721747	1061.095034	1169.860445	348.826199	
std	416.159877	35.471226	8957.442311	182.218483	462.664785	163.520016	449.375525	442.272249	391.161983	439.696370	
min	1.000000	0.000000	1300.000000	0.000000	0.000000	0.000000	0.000000	0.000000	334.000000	0.000000	
25%	360.500000	41.750000	7621.500000	0.000000	0.000000	0.000000	216.000000	799.000000	892.000000	0.000000	
50%	714.500000	64.000000	9522.500000	0.000000	385.500000	0.000000	474.000000	1005.500000	1096.500000	0.000000	
75%	1079.500000	79.250000	11515.500000	160.000000	714.500000	0.000000	816.000000	1291.500000	1392.000000	729.000000	
max	1460.000000	313.000000	164660.000000	1600.000000	5644.000000	1474.000000	2336.000000	6110.000000	4692.000000	2065.000000	

From this statistical analysis we make some of the interpretations that,

- Maximum standard deviation of 8957.44 is observed in LotArea column.
- Maximum SalePrice of a house observed is 755000 and minimum is 34900.
- In the columns Id, MSSubclass, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfsF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, HalfBath, TotRmsAbvGrd, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, Miscval, salePrice mean is considerably greater than median so the columns are positively skewed.
- In the columns FullBath, BedroomAbvGr, Fireplaces, Garagecars, GarageArea, YrSold Median is greater than mean so the columns are negatively skewed.
- In the columns Id, MSSubClass, LotFrontage, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtHalfBath, BedroomAbvGr, ToRmsAbvGrd, GarageArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal, SalePrice there is considerable difference between the 75 percentile and maximum so outliers are present.

Data Cleaning

Duplicates & NaNs: I started by removing duplicates from the data, checked for missing or NaN (not a number) values. It's important to check for NaNs (and not just because it's socially moral) because these cause errors in the machine learning models.

Categorical Features: There are a lot of categorical variables that are marked as N/A when a feature of the house is nonexistent. For example, when no alley is present. I identified all the cases where this was happening across the training and test data and replaced the N/As with something more descriptive. N/As can cause errors with machine learning later down the line so get rid of them.

Date Features: For this exercise dates would be better used as categories and not integers. After all, it's not so much the magnitude that we care about but rather that the dates represent different years. Solving this problem is simple, just convert the numeric dates to strings.

Decoded Variables: Some categorical variables had been number encoded. See the example below.

```
MSSubClass: Identifies the type of dwelling involved in the sale.

        20      1-STORY 1946 & NEWER ALL STYLES
        30      1-STORY 1945 & OLDER
        40      1-STORY W/FINISHED ATTIC ALL AGES
        45      1-1/2 STORY - UNFINISHED ALL AGES
        50      1-1/2 STORY FINISHED ALL AGES
        60      2-STORY 1946 & NEWER
        70      2-STORY 1945 & OLDER
        75      2-1/2 STORY ALL AGES
        80      SPLIT OR MULTI-LEVEL
        85      SPLIT FOYER
        90      DUPLEX - ALL STYLES AND AGES
       120      1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150      1-1/2 STORY PUD - ALL AGES
       160      2-STORY PUD - 1946 & NEWER
       180      PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190      2 FAMILY CONVERSION - ALL STYLES AND AGES
```

The problem here is that the machine learning algorithm could interpret the magnitude of the number to be important rather than just interpreting it as different categories of a feature. To solve the problem, I reverse engineered the categories and recoded them.

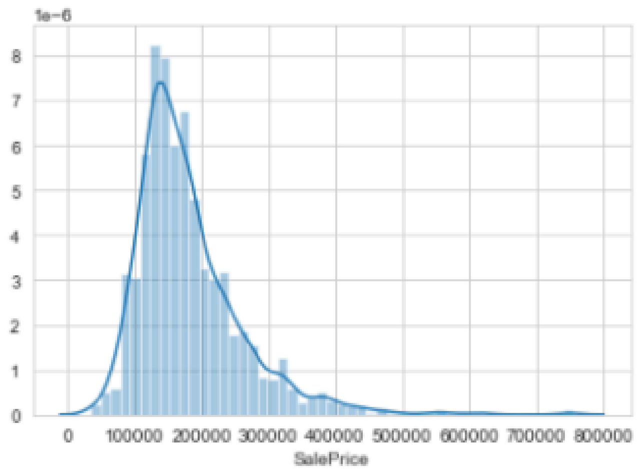
Exploratory Data Analysis (EDA)

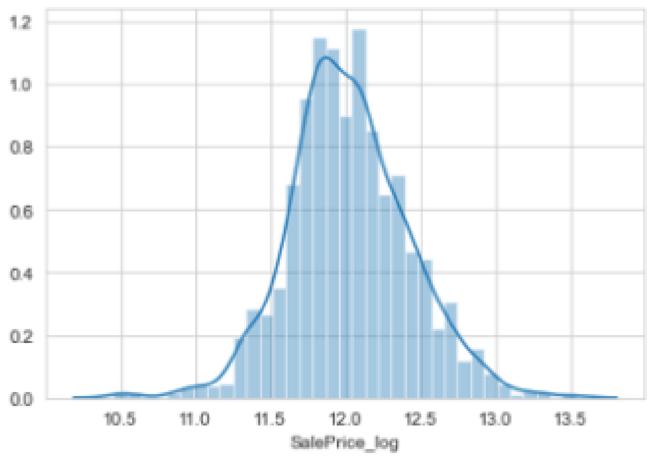
This is where our data visualisation journey often begins. The purpose of EDA in machine learning is to explore the quality of our data.

Here I perform my first bit of feature engineering. I'll apply a log transform to sales price to compress outliers making the distribution normal.

Outliers can have devastating effects on models that use loss functions minimising squared error. Instead of removing outliers try applying a transformation.

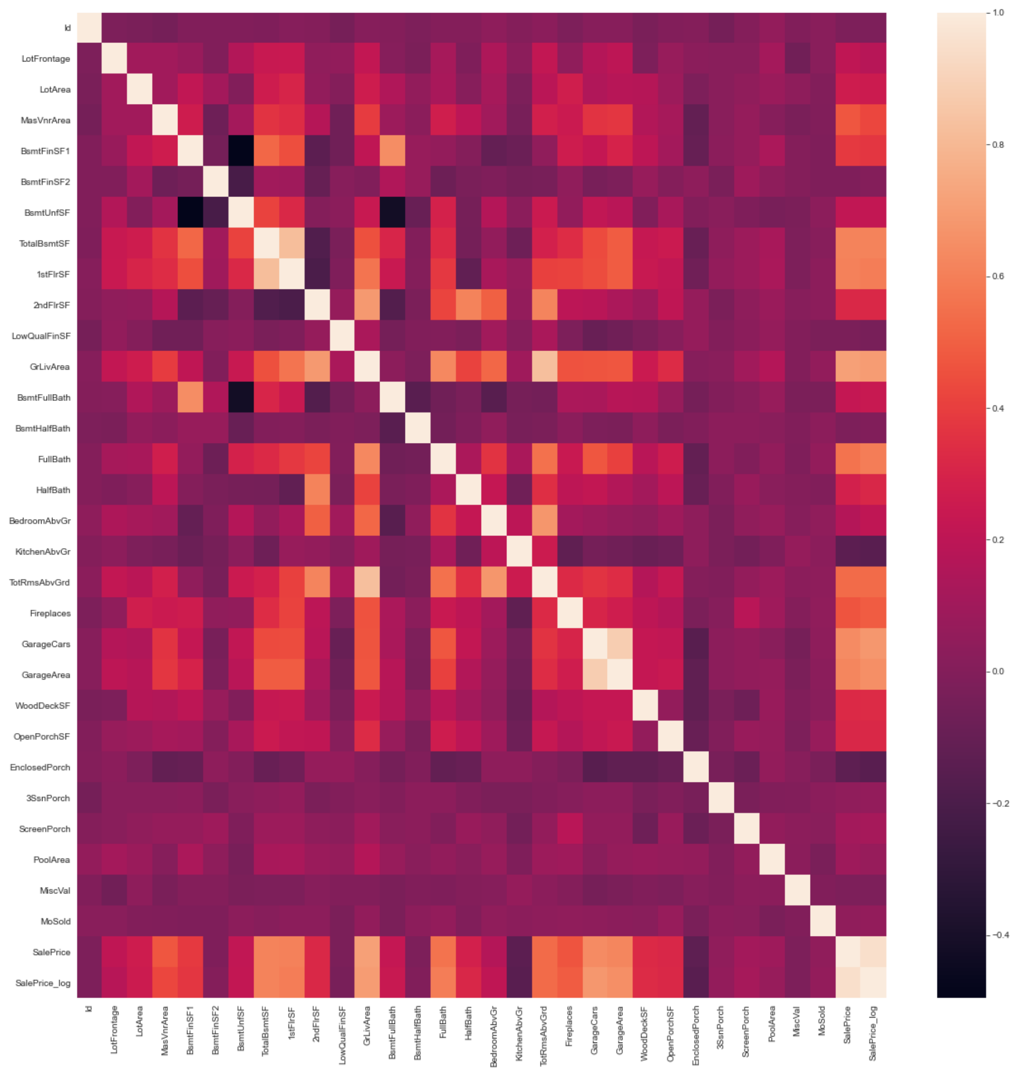
```python
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

x = Train.SalePrice
sns.set_style("whitegrid")
sns.distplot(x)
plt.show()

Train["SalePrice_log"] = np.log(Train.SalePrice)
x = Train.SalePrice_log
sns.distplot(x)
plt.show()
```





Correlations: It’s often good to plot a correlation matrix to give you an idea of relationships that exist in your data. It can also guide your model building. For example, if you see a lot of your features are correlated with each other you might want to avoid linear regression.

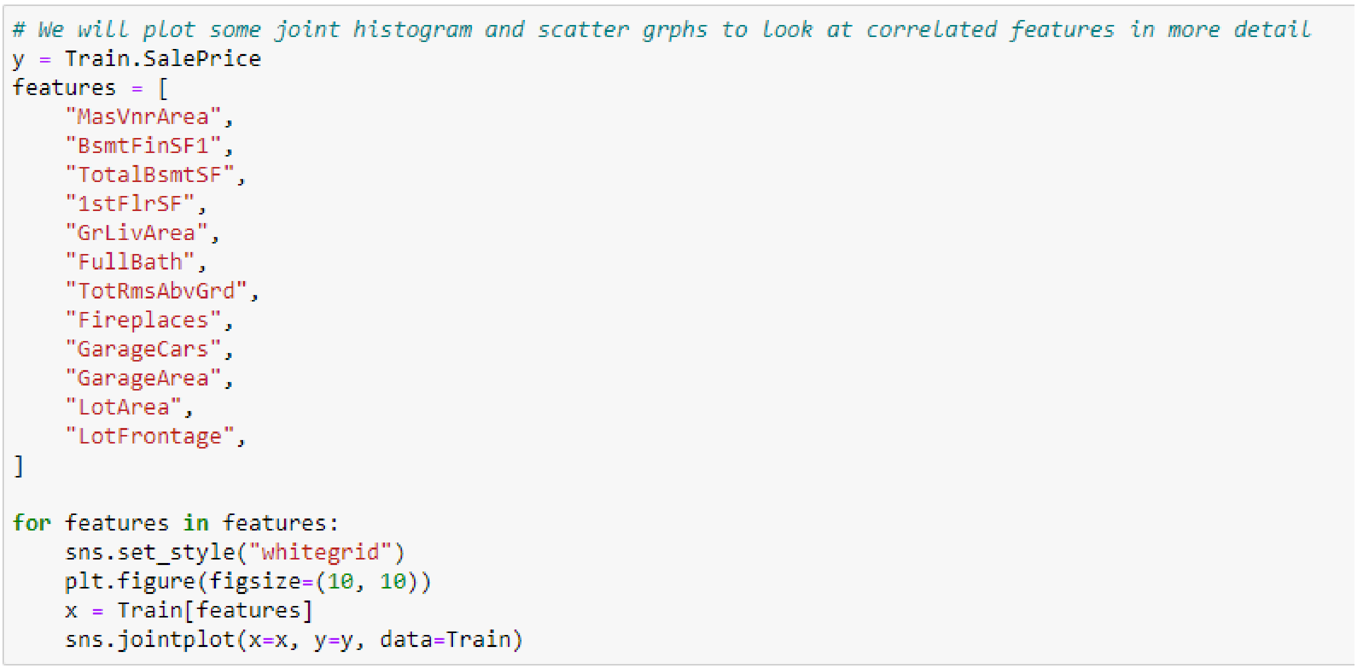
```
# Lets explore the correlations in our data set
plt.figure(figsize=(20, 20))
sns.heatmap(Train.corr())
plt.savefig("pearsonsmep.png")
```



The correlation measure used here is Pearson's correlation. In our case the lighter the square the stronger the correlation between two variables.

Features related to space such as lot frontage, garage area, ground living area were all positively correlated with sale price as one might expect. The logic being that larger properties should be more expensive. No correlations look suspicious here.

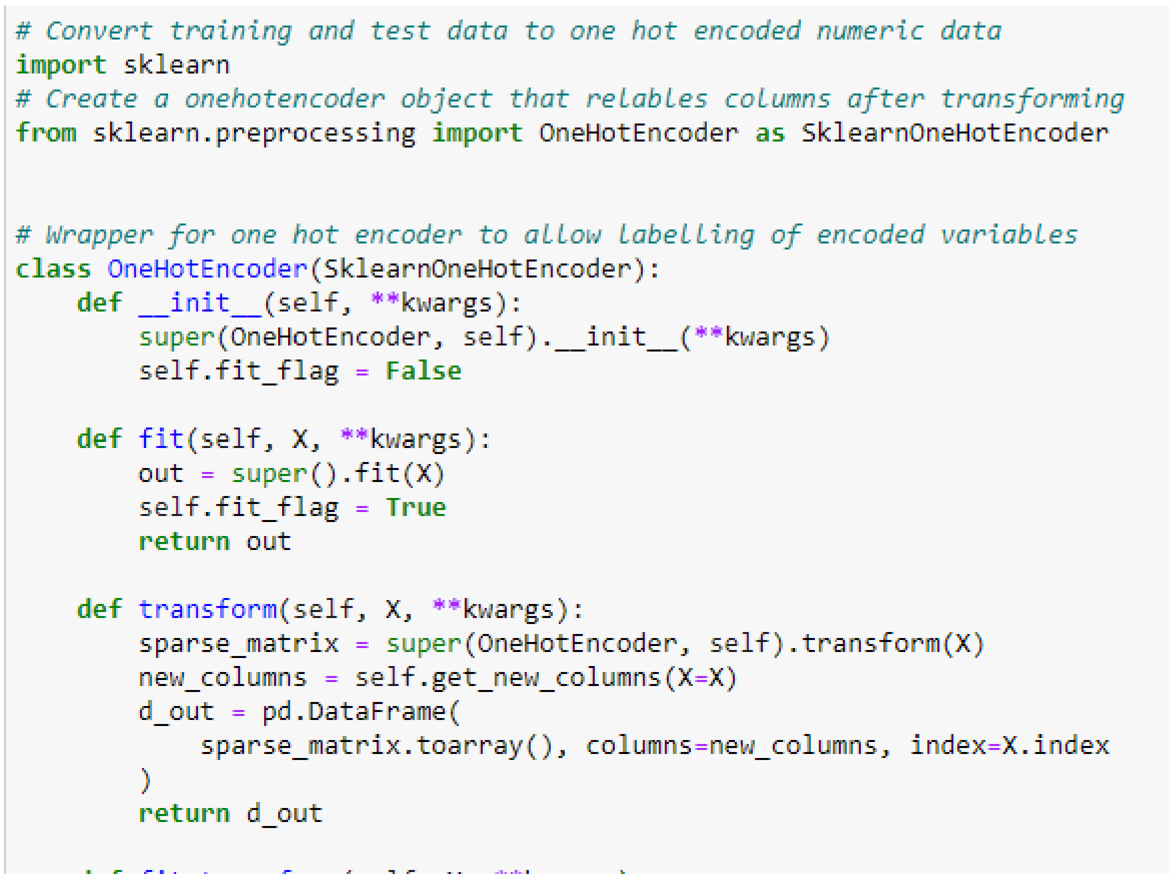
Categorical Relations: Sales price appears to be approximately normally distributed within each level of each category. No observations appear, untoward. Some categories contain little to no data, whilst other show little to no distinguishing ability between sales class. See full project on GitHub for data visualisation.



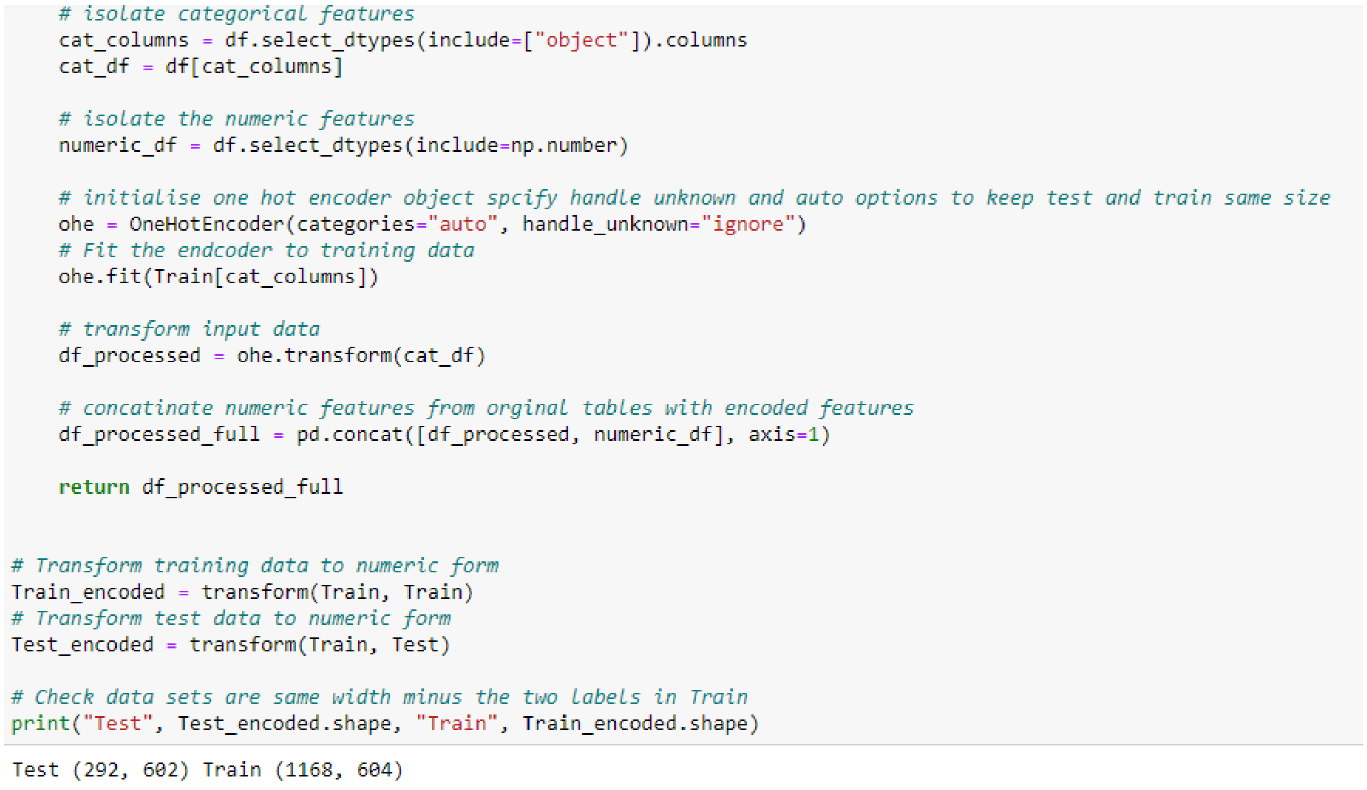
```
# We will plot some joint histogram and scatter grphs to look at correlated features in more detail
y = Train.SalePrice
features = [
    "MasVnrArea",
    "BsmtFinSF1",
    "TotalBsmtSF",
    "1stFlrSF",
    "GrLivArea",
    "FullBath",
    "TotRmsAbvGrd",
    "Fireplaces",
    "GarageCars",
    "GarageArea",
    "LotArea",
    "LotFrontage",
]

for features in features:
    sns.set_style("whitegrid")
    plt.figure(figsize=(10, 10))
    x = Train[features]
    sns.jointplot(x=x, y=y, data=Train)
```

Feature Engineering

Machine learning models can't understand categorical data. Therefore we will need to apply transformations to convert the categories into numbers. The best practice for doing this is via one hot encoding.



```python
# Convert training and test data to one hot encoded numeric data
import sklearn
# Create a onehotencoder object that relables columns after transforming
from sklearn.preprocessing import OneHotEncoder as SklearnOneHotEncoder


# Wrapper for one hot encoder to allow labelling of encoded variables
class OneHotEncoder(SklearnOneHotEncoder):
    def __init__(self, **kwargs):
        super(OneHotEncoder, self).__init__(**kwargs)
        self.fit_flag = False

    def fit(self, X, **kwargs):
        out = super().fit(X)
        self.fit_flag = True
        return out

    def transform(self, X, **kwargs):
        sparse_matrix = super(OneHotEncoder, self).transform(X)
        new_columns = self.get_new_columns(X=X)
        d_out = pd.DataFrame(
            sparse_matrix.toarray(), columns=new_columns, index=X.index
        )
        return d_out
```



```python
    # isolate categorical features
    cat_columns = df.select_dtypes(include=["object"]).columns
    cat_df = df[cat_columns]

    # isolate the numeric features
    numeric_df = df.select_dtypes(include=np.number)

    # initialise one hot encoder object spcify handle unknown and auto options to keep test and train same size
    ohe = OneHotEncoder(categories="auto", handle_unknown="ignore")
    # Fit the endcoder to training data
    ohe.fit(Train[cat_columns])

    # transform input data
    df_processed = ohe.transform(cat_df)

    # concatinate numeric features from orginal tables with encoded features
    df_processed_full = pd.concat([df_processed, numeric_df], axis=1)

    return df_processed_full


# Transform training data to numeric form
Train_encoded = transform(Train, Train)
# Transform test data to numeric form
Test_encoded = transform(Train, Test)

# Check data sets are same width minus the two labels in Train
print("Test", Test_encoded.shape, "Train", Train_encoded.shape)
```

```
Test (292, 602) Train (1168, 604)
```

Model Selection

As mentioned at the start of the article the task is supervised machine learning. We know it's a regression task because we are being asked to predict a numerical outcome (sale price).

Therefore, I approached this problem with three machine learning models. Decision tree, random forest and gradient boosting machines. I used the decision tree as my baseline model then built on this experience to tune my candidate models. This approach saves a lot of time as decision trees are quick to train and can give you an idea of how to tune the hyperparameters for my candidate models.

Model mechanics: I will not go into too much detail about how each model works here. Instead I'll drop a one-liner and link you to articles that describe what they do "under the hood".

Decision Tree — A tree algorithm used in machine learning to find patterns in data by learning decision rules.

Random Forest — A type of bagging method that plays on 'the wisdom of crowds' effect. It uses multiple independent decision trees in parallel to learn from data and aggregates their predictions for an outcome.

Gradient Boosting Machines — A type of boosting method that uses a combination of decision tree in series. Each tree is used to predict and correct the errors by the preceding tree additively.

Random forests and gradient boosting can turn individually weak decision trees into strong predictive models. They're great algorithms to use if you have small training data sets like the one we have.

Training

In machine learning training refers to the process of teaching your model using examples from your training data set. In the training stage, you'll tune your model hyperparameters.

Before we get into further detail, I wish to briefly introduce the bias-variance trade-off. Complexity can be thought of as the number of features in the model. Model variance and model bias have an inverse relationship leading to a trade-off. There is an optimal point for model complexity that minimizes the error. We seek to establish that by tuning our hyper parameters.

Hyperparameters: Hyperparameters help us adjust the complexity of our model. There are some best practices on what hyperparameters one should tune for each of the models. I'll first detail the hyperparameters, then I'll tell you which I've chosen to tune for each model.

max_depth — The maximum number of nodes for a given decision tree.

max_features — The size of the subset of features to consider for splitting at a node.

n_estimators — The number of trees used for boosting or aggregation. This hyperparameter only applies to the random forest and gradient boosting machines.

learning_rate — The learning rate acts to reduce the contribution of each tree. This only applies for gradient boosting machines.

Decision Tree — Hyperparameters tuned are the max_depth and the max_features

Random Forest — The most important hyperparameters to tune are n_estimators and max_features

Gradient boosting machines — The most important hyperparameters to tune are n_estimators, max_depth and learning_rate.

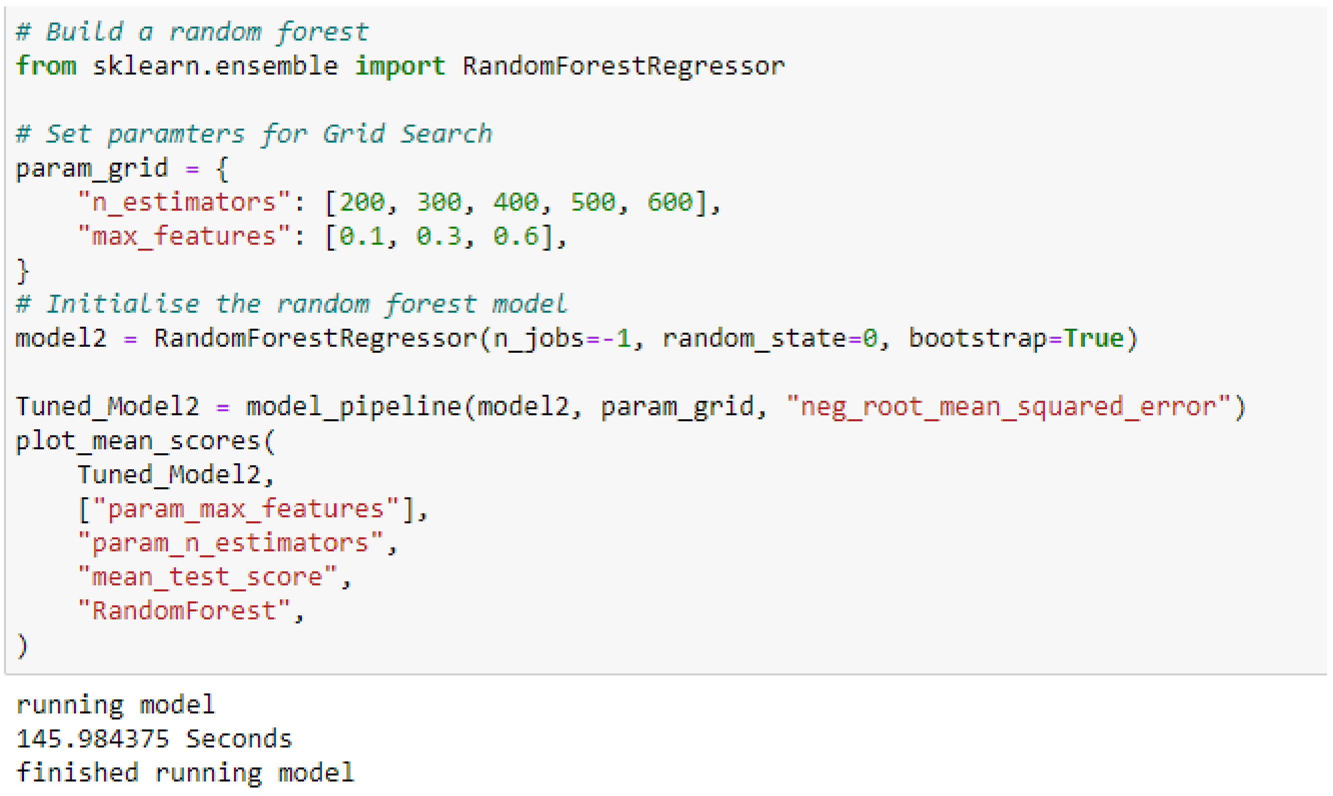
Grid search: Choosing the range of your hyperparameters is an iterative process. With more experience you'll begin to get a feel for what ranges to set. The good news is once you've chosen your possible hyperparameter ranges, grid search allows you to test the model at every combination of those ranges. I'll talk more about this in the next section.

Cross validation: Models are trained with a 5-fold cross validation. A technique that takes the entirety of your training data, randomly splits it into train and validation data sets over 5 iterations.

You end up with 5 different training and validation data sets to build and test your models. It's a good way to counter overfitting.

Implementation: SciKit Learn helps us bring together hyperparameter tuning and cross validation with ease in using GridSearchCv. It gives you options to view the results of each of your training runs.

Here's a run through of the code to build the model for random forests.

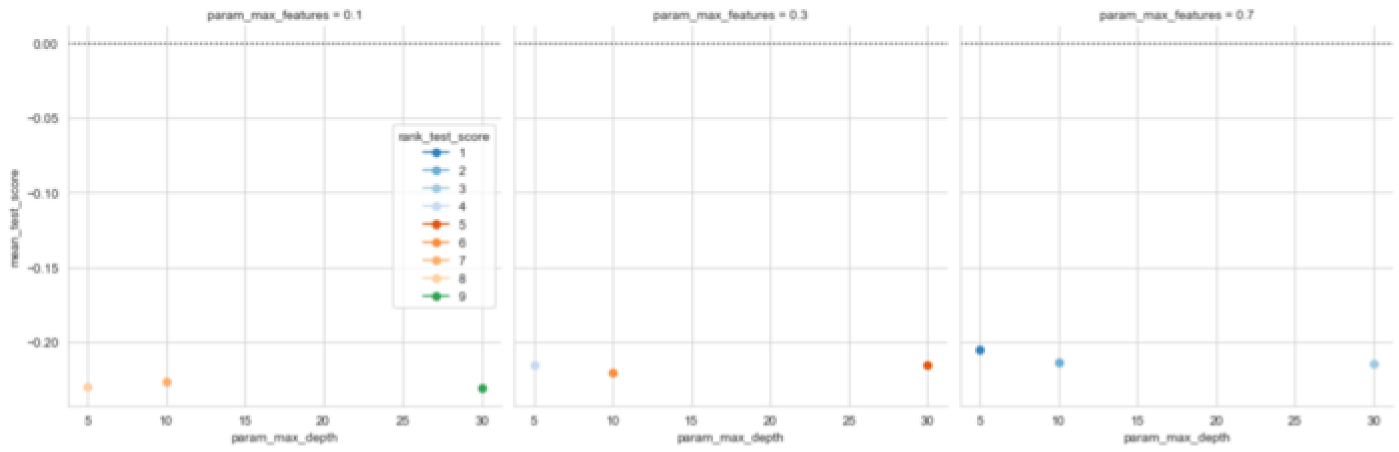


```
# Build a random forest
from sklearn.ensemble import RandomForestRegressor

# Set paramters for Grid Search
param_grid = {
    "n_estimators": [200, 300, 400, 500, 600],
    "max_features": [0.1, 0.3, 0.6],
}
# Initialise the random forest model
model2 = RandomForestRegressor(n_jobs=-1, random_state=0, bootstrap=True)

Tuned_Model2 = model_pipeline(model2, param_grid, "neg_root_mean_squared_error")
plot_mean_scores(
    Tuned_Model2,
    ["param_max_features"],
    "param_n_estimators",
    "mean_test_score",
    "RandomForest",
)
```

```
running model
145.984375 Seconds
finished running model
```

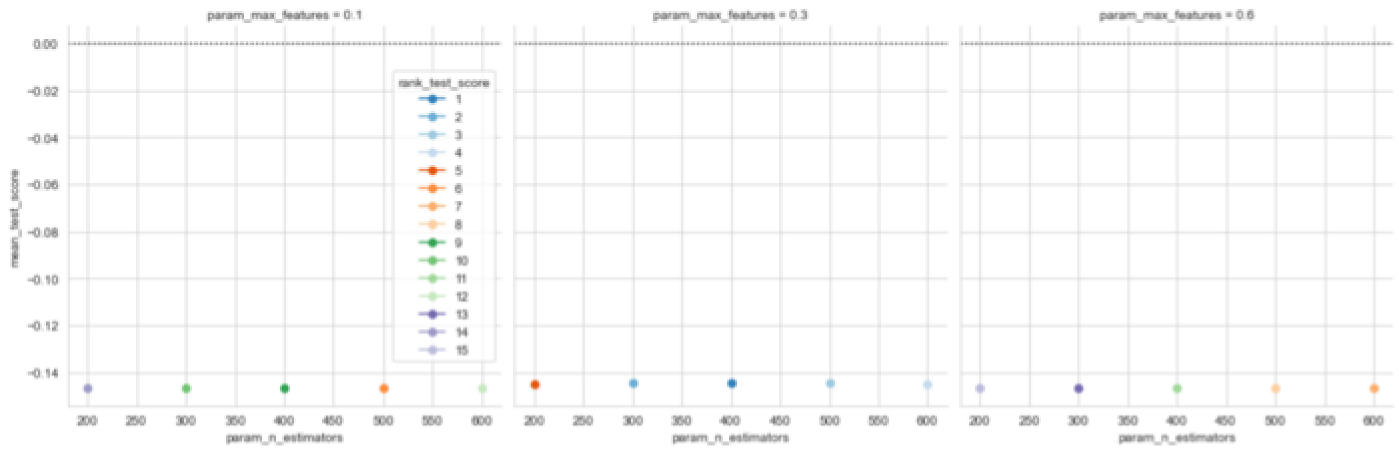
Decision tree

Predictably this was our worst performing method. Our best decision tree score -0.205 NRMSE. Tuning the hyperparameters didn't appear to make much of a difference to the model's however it trained in under 2 seconds. There is definitely some scope to assess a wider range of hyperparameters.



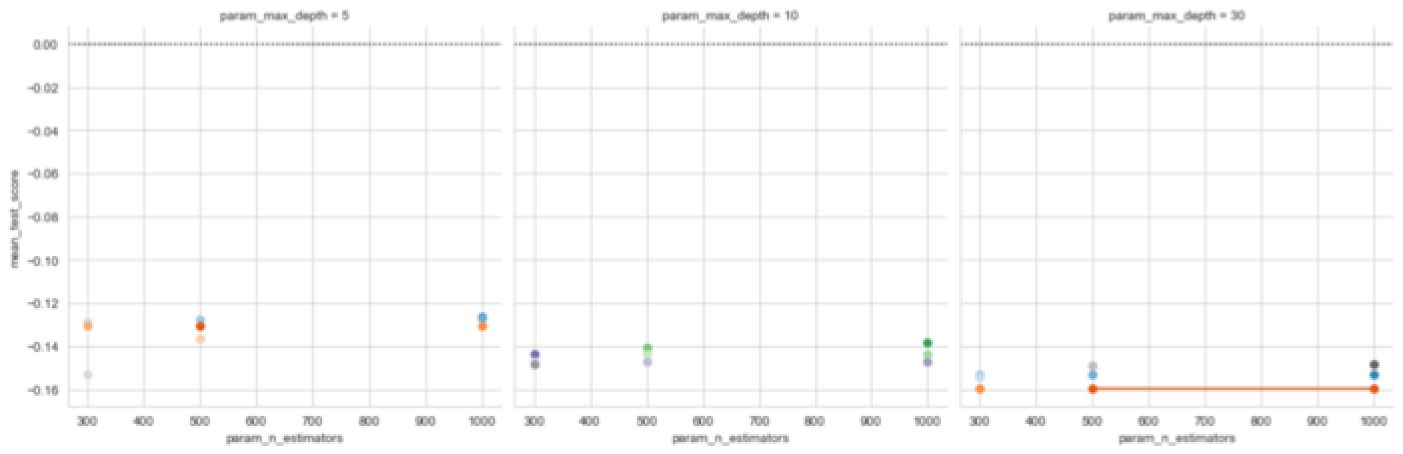
Random Forest

Our random forest model was a marked improvement on our decision tree with a NRMSE of -0.144. The model took around 75 seconds to train.



Gradient Boosting Machine

This was our best performer by a significant amount with a NRMSE of -0.126. The hyperparameters significantly impact the results illustrating that we must be incredibly careful in how we tune these more complex models. The model took around 196 seconds to train.


param_max_depth = 5
param_max_depth = 10
param_max_depth = 30
mean_test_score
param_n_estimators
0.00
-0.02
-0.04
-0.06
-0.08
-0.10
-0.12
-0.14
-0.16
300
400
500
600
700
800
900
1000