

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CZ4032 Data Analytics and Mining Assignment

Group 9

Author:	Student ID:	Contribution
Derrick Peh Jia Hao	U1621219F	25%
Lim Boon Leng	U1621831E	25%
Tan Jin Ting	U1622629D	25%
Cheng Qian Yi	U1621150A	25%

Submission Date: 16 / 10 / 2018

Abstract

This project makes use of several data processing techniques and machine learning algorithms to do a prediction on the sales prices of houses based on their characteristics. The data is sourced from Kaggle and contains a comprehensive list of houses with sales prices and their features. Such features include interior, exterior, architecture, location, land, accessibility, utilities and miscellaneous of the houses.

Knowledge of data preprocessing and analysis are applied in attempts to remove diluted data for more accurate predictions - data preparation and feature engineering are done before training and prediction. This project is coded in Python, and makes use of open source packages/toolkits such as Pandas, scipy, and sklearn for the statistical analysis, modeling algorithms and preprocessing.

The goal of this Kaggle project is to predict the sales price of each house in the given test set. Evaluation of the prediction is subjected to the Root-Mean-Squared-Error between the logarithm of the predicted value and the logarithm of the observed sales price.

Table of Contents

Abstract	2
Table of Contents	3
Problem Description	5
Motivation	5
Problem Definition	5
Related Work	5
Approach	6
Methodology	6
Algorithms	7
Implementations	9
Data Cleaning and Preprocessing	9
Removal of outliers	9
Handling of missing values	10
Data Analysis	12
Correlation Matrix	12
Feature Engineering	13
Target Variable - SalePrice	14
Treating Skewed Features	14
Training and Prediction	15
Training	15
Optimising	16
Stacking	18
Ensemble	18
Experimental Results and Analysis	19
Experimental Setup	19
Comparison Schemes	19
Results and Analysis	19
Discussion of Pros and Cons	20
Conclusions	21
Summary of project achievements	21
Directions for improvements	21
References	22

Appendix	23
Datasets	23
Scripts/Source Codes	23
Implementation Guidelines	23

1. Problem Description

1.1. Motivation

Physical features of houses and its vicinity are often thought as overlooked or not considered when homebuyers are making decisions in purchasing their dream homes. Such data features include interior, exterior, architecture, location, land, accessibility, utilities and more information that were thought unimportant.

1.2. Problem Definition

This project takes into consideration that physical features are actually as influential as factors like number of bedrooms during price negotiations. This project attempts to prove so by attempting to process these information accordingly to train and make predictions on the sales prices of houses.

1.3. Related Work

The kernel provided by Serigne [1] provides an overview of the process that he implemented to tackling the objective of the competition. His methodology consists of 3 main sections : Data cleaning, Data processing and Modelling. These sections are implemented as part of our experimental setup. A more in depth preparation on data processing including treating outliers, missing values and deriving new features is demonstrated in his kernel implementation which we have also adopted as part of our experimental setup.

Pedro Marcelino [2] demonstrates the relationship of features with the target variables using box plot graph and scatter plot graph. It has provided a clearer and in depth view of the values that are present in the features, such as the frequency of the values and the correlation present between features. Pedro Marcelino [3] also introduced the implementation of one hot encoding that can be processed on categorical features. This is implemented on all of our categorical features, together with other processed numerical features to ensure that it is accepted by the training models we implemented to predict the price of the house.

2. Approach

2.1. Methodology

1. Data Selection

For this problem, the dataset used was taken from Kaggle. It contains one training set (train.csv), one test set (test.csv) and a sample benchmark submission from a linear regression on year and month of sale, lot square footage, and number of bedrooms (sample_submission.csv).

2. Data Cleaning and Preprocessing

After identifying and understanding the datasets, it is important to ensure that the data stored is consistent as quality results are based on quality data. Thus, these methods ensure quality data is met:

- a. Removal of outliers
- b. Handling of missing values

3. Data Analysis

Once the data was organised, gaining useful information can be done through the following methods to support decision making:

- a. Correlation matrix
- b. Feature engineering
- c. Target variable
- d. Treating skews

4. Training

- a. Feature reduction
- b. Train
- c. Optimization
- d. Stacking
- e. Ensemble

5. Prediction

Based on the collective results gotten from stage 1 to 4, better house prices prediction can be done to drive better decision on which houses to buy or the selling price for the house.

2.2. Algorithms

The method of measuring accuracy was chosen to be Root Equal Mean Squared Error which was mentioned in the competition details. We will use the inbuilt function in scikit learn. The description of each different algorithms will be shown below:

1. Kernel Ridge Regression

Kernel Ridge Regression (KRR) joins Ridge Regression together by using the kernel trick. In doing so, a linear function in the space made by the individual kernel and data was learned. For non-linear kernels, this relates to a non-linear function in the first space.

Kernel Ridge learned a type of the model that is similar to support vector regression (SVR). But, unlike loss functions are utilized: KRR utilizes squared error loss while SVR utilizes ϵ -insensitive loss, both combined with l_2 regularization. Rather than SVR, fitting Kernel Ridge is possible to do in closed-form and is usually faster for medium-sized datasets. Then again, the learned model is non-sparse and in this manner SVR will be faster, which learns a sparse model for, at prediction-time [4].

2. Elastic Net

Lasso is a regularization method for carrying out linear regression. It incorporates a forfeit term that limit the size of the evaluated coefficients. In this manner, it takes after ridge regression. Lasso is a shrinkage estimator: it creates coefficient estimates that are biased to be small. Nonetheless, a lasso estimator might have smaller mean squared error when applying it to new data as compared to a normal least-squares estimator.

Not like ridge regression, where lasso sets more coefficients to zero when the forfeit term rises. This implies that the lasso estimator have lesser indicators and it is a smaller model. Therefore, lasso is consider as a substitute in contrast to stepwise regression and other model selection or reduction techniques [5].

3. Lasso

Elastic net is a related technique. It is a combination of ridge regression and lasso regularization. By generating zero-valued coefficients, lasso can generate reduced models like lasso. Observational research have proposed that the elastic net method can beat lasso on data with highly correlated predictors [5].

4. Gradient Boosting

Gradient Boosting helps to reduce the loss when adding new models by using new created model to make assumptions of errors of the prior models, and then added collectively to make the final prediction.

5. Bayesian Ridge

Bayesian linear algorithm enables a reasonably ordinary system to survive deficient data, or poor disseminated data. It enables you to place a prior on the coefficients and on the noise for the priors to take over when the data is not present. Most significantly, you can enquire Bayesian linear which portions (assuming any) of its fit to the data is it certain about, and which portions are exceptionally dubious (maybe relied completely on the priors) [6].

6. Lasso Lars IC

Lasso Lars IC which is also known as Least Angle Regression is a model determination method for linear regression. It is basically forward stagewise made fast. Rather than making minor jumps toward one variable at any given moment, LARS makes ideally estimated jumps in optimal directions. These directions are selected to make equivalent angles with every variable present in the model. LARS also tend to give results that are very similar to both lasso and forward stagewise [7].

7. Random Forest Regressor

Random Forest Regressor has a feature of handling tabular data with numerical features, and uses collectively decisions from a sequence of base models to make prediction. The class of models can be written as: $g(x)=f_0(x)+f_1(x)+f_2(x)+...$ where the final model g is the sum of simple base models f_i . This would provide better performance in predicting as the base models are tested independently using different subsample of the data.

8. XGBoost

XGBoost (eXtreme Gradient Boosting), an enhanced version of Gradient Boosting, is a software library that uses advanced gradient boosting decision tree algorithm. The key of XGBoost is to use the available resource efficiently to train the model. This can be achieved by Sparse Aware to handle missing data values, Block Structure to support parallelization of tree construction, and Continued Training to boost on new data using the fitted model [8].

Implementations

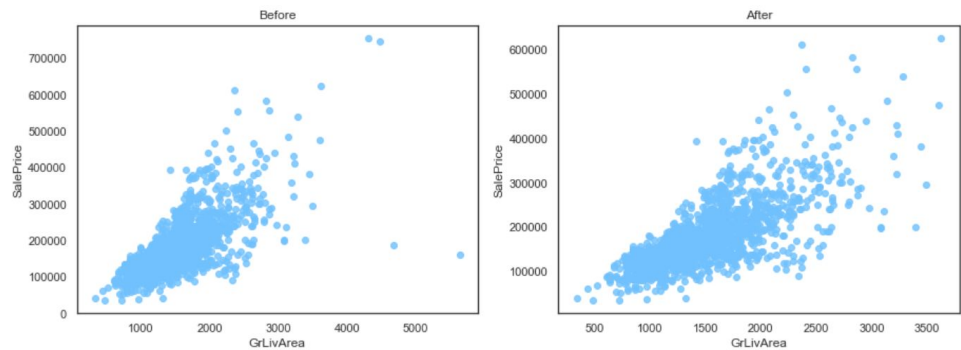
2.3. Data Cleaning and Preprocessing

2.3.1. Removal of outliers

```
plt.subplots(figsize=(15, 5))

plt.subplot(1, 2, 1)
g = sns.regplot(x=train['GrLivArea'], y=train['SalePrice'], fit_reg=False).set_title("Before")

# Delete outliers
plt.subplot(1, 2, 2)
train = train.drop(train[(train['GrLivArea']>4000)].index)
g = sns.regplot(x=train['GrLivArea'], y=train['SalePrice'], fit_reg=False).set_title("After")
```



The graph 'Before' shows the data distribution of the feature GrLivArea against SalePrice before removing the outliers. The outliers are clearly identified to be in the range of >4000 GrLivArea as the data points are located way off the distribution of other data points. As this may skew the distribution of the data and potential calculations, the outliers >4000 GrLivArea were dropped.

The graph 'After' shows the data distribution after removing the outliers. The data points are clearly seen to be more closely related and dense than the 'Before' graph.

2.3.2. Handling of missing values

Two ways of handling missing values were used in this project:

- Deletion of entire data row
- Fill the missing entry with an imputed value

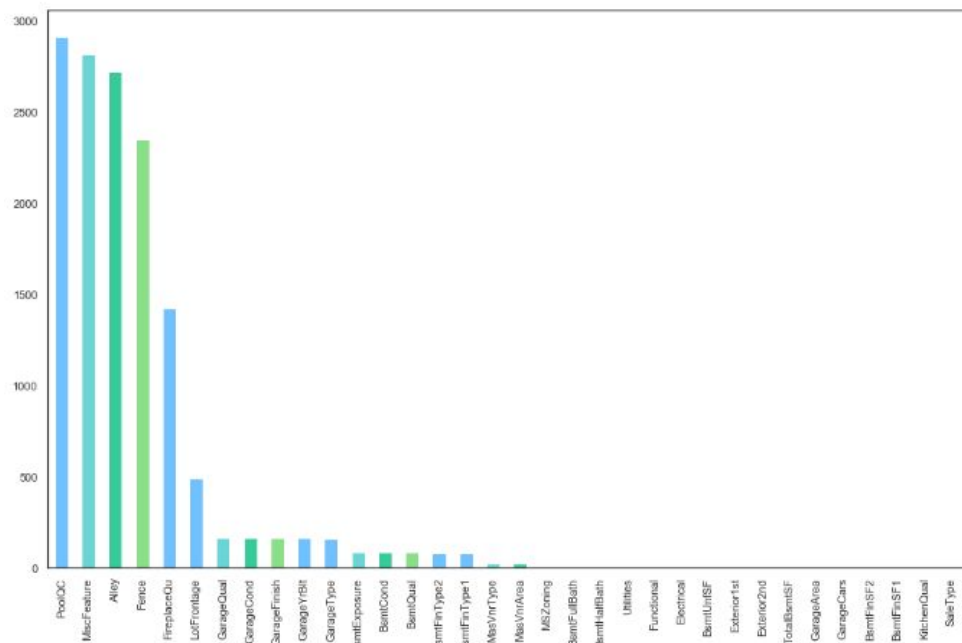
To ensure that modifications are consistent, both the training and test dataset are concatenated together before any changes are done.

```
# concatenate training and test data into all_data
all_data = pd.concat((train, test)).reset_index(drop=True)
all_data.drop(['SalePrice'], axis=1, inplace=True)
```

We then aggregate the null values of each feature to find out which one has how many missing entries. The features that have no missing entries at all are dropped as they need not be handled.

```
# aggregate all null values
all_data_na = all_data.isnull().sum()

# get rid of all the values with 0 missing values
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values(
ascending=False)
plt.subplots(figsize=(16, 10))
all_data_na.plot(kind='bar');
```



A plot of the features against the number of missing entries

For the missing entries, they are filled with an imputed value based on their data feature description. The features can be split into two categories:

- Features that cannot be quantified
 - *Alley* - Type of alley access
 - *Fence* - Fence quality
- Features that can be quantified
 - *GarageArea* - Size of garage in square feet
 - *GarageCars* - Size of garage in car capacity

For features that cannot be quantified, they are filled with “None”, while quantifiable features are filled with “0”. They are filled with null values - zeroes or nones, as the possibility of the features being not existent is there. *GarageArea* can be missing because this house has no garage at all.

For others features such as *LotFrontage*, *MSZoning*, *Electrical*, *KitchenQual*, *Exterior1st*, *Exterior2nd*, *SaleType* and *Functional*, the missing values are filled with the median or the mode of itself. This was done as these features cannot have null values such that they cannot contain zeroes or nones.

For example:

- *LotFrontage* - Linear feet of street connected to property
- *SaleType* - Type of Sale

The data would not make sense if such features contained “0” or “None”, since the linear feet of a street is not possible to be 0, nor the type of sale to be none - since this is a list of sold house prices.

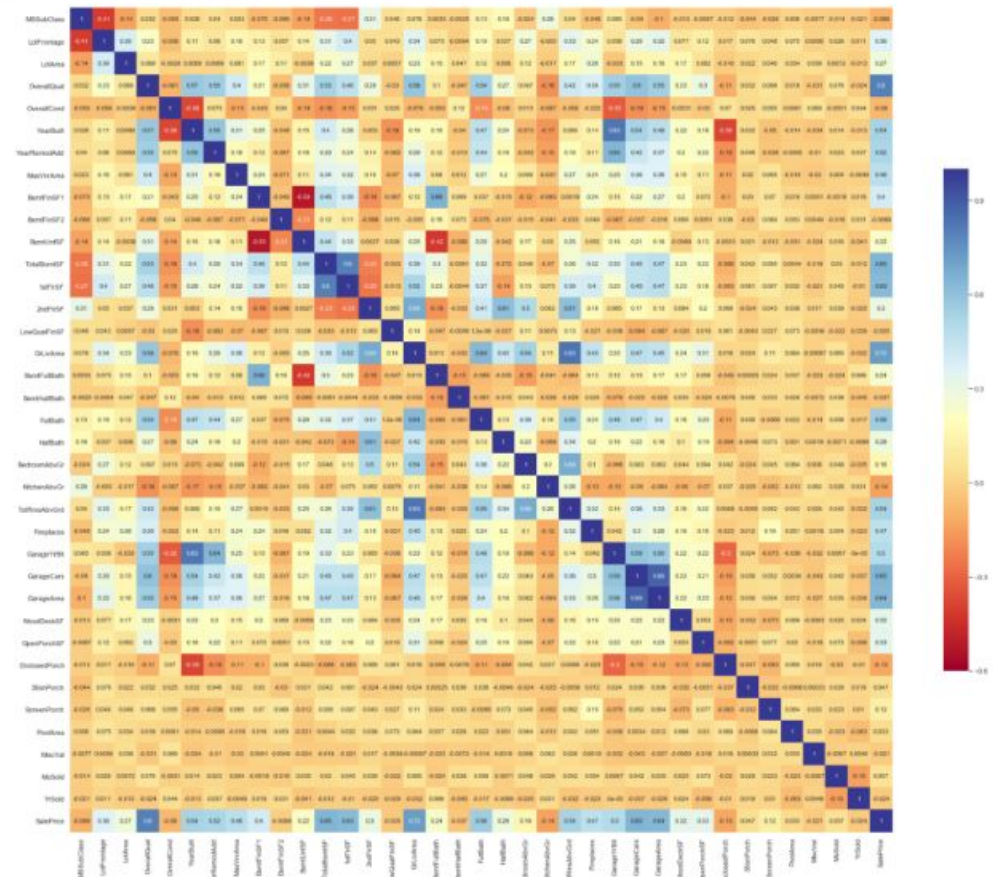
There is a binary feature, *Utilities*, that only has two values “AllPub” and “NoSeWa”. However, the test dataset only has data containing “AllPub” - constant, not useful for prediction. Thus Utiliti

2.4. Data Analysis

2.4.1. Correlation Matrix

To analyze the relationship of the features in more detail, we made use of Panda's `corr()` method and plotted the data onto a heatmap for us visualize the data better.

```
corr = train.corr()
plt.subplots(figsize=(30, 30))
cmap = sns.diverging_palette(150, 250, as_cmap=True)
sns.heatmap(corr, cmap="RdYlBu", vmax=1, vmin=-0.6, center=0.2, square=True, linewidths=0, cbar_kws={"shrink": .5}, annot = True);
```



It is clearly visible what are the highly influential features on *SalePrice* from the above heatmap.

2.4.2. Feature Engineering

Using the heatmap, we generate derived features from the highly correlated features to capture the complexity of non-linear relationships that are in the dataset.

Below is the list of highly influential features observed from the heatmap:

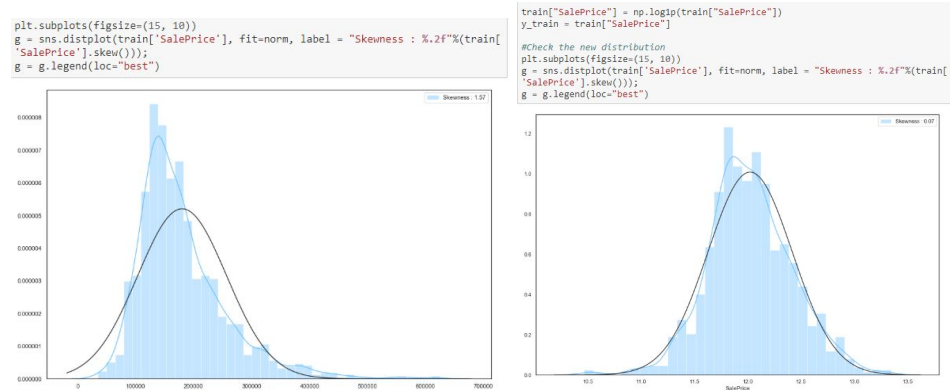
1. OverallQual
2. GrLivArea
3. GarageCars
4. GarageArea
5. TotalBsmtSF . . .

For different types of features, the data entries are handled differently:

- Ordinal categorical feature
 - *BsmtQual* - Height of basement
 - Values - None, Fa, TA, Gd, Ex
 - Positive correlation with SalePrice
 - Replaced with values 0, 1, 2, 3, 4 respectively
 - *BsmtFinType1* - Rating basement finished area
 - NO or minimal correlation with SalePrice
 - Create dummy features
 - *BsmtFinType1_ALQ*
 - *BsmtFinType1_BLQ* . . .
- Non-ordinal categorical feature
 - *Neighborhood* - Physical locations within Ames city limits
 - Create dummy features
- Continuous numerical features
 - *BsmtFinSF1* - Type 1 finished square feet
 - Create dummy features of different range groups
- Insubstantial unary features
 - *LowQualFinSF* - Low quality finished square feet (all floors)
 - Values - Largely 0 or other values
 - Replaced with binary flag - 0 or 1
- Redundant features
 - *BsmtHalfbath*, *BsmtFullBath*, *HalfBath*, *FullBath*
 - All represent number of bathrooms
 - Replaced with a total bathrooms feature

2.4.3. Target Variable - SalePrice

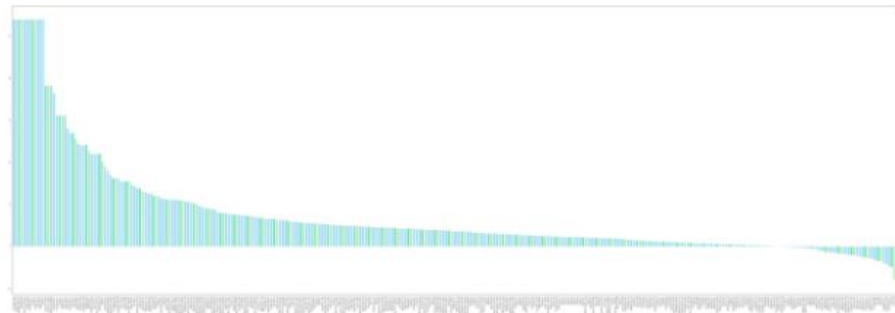
Machine learning algorithms work best with features that are normally distributed - symmetric and characteristic bell-shaped curve.



It is observed that there is a positive skew - mode is always less than mean and median. We transform the distribution using Python's numpy function log1p to all elements.

2.4.4. Treating Skewed Features

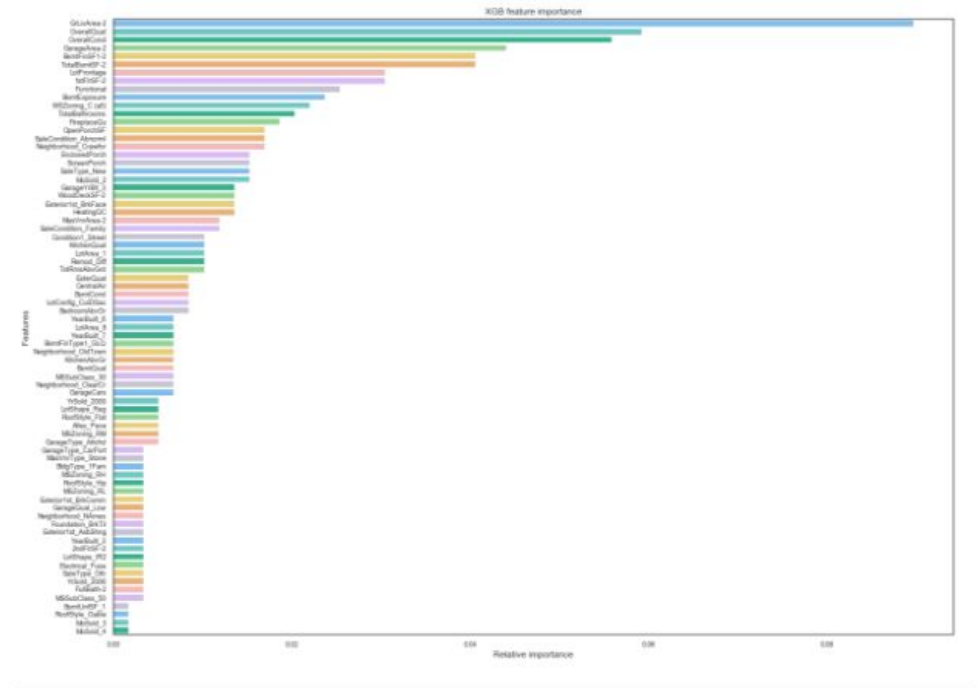
Same as the target variable, we need to treat the skewed features



It is observed that there is a mix of positive and negative skewed features. Box-Cox Transform is performed on the features that skew > 0.5 to follow the normal distribution more closely.

2.5. Training and Prediction

2.5.1. Training



Features that are deemed not important are not included in the training model. This is implemented using XGBoost's inbuilt feature importance functionality. The result depicts in descending order the important features calculated by the XGBoost - the first feature at the top being the most important feature and the last feature at the bottom the least important feature.

For analysis, we applied eight different algorithms as mentioned in Section 2.2. Python's scikit and xgboost is used to implement the machine learning models. The metric for result analysis is Root Mean Squared Error as used on Kaggle.

```
import xgboost as xgb
#Machine Learning Algorithm (MLA) Selection and Initialization
models = [KernelRidge(), ElasticNet(), Lasso(), GradientBoostingRegressor(), BayesianRidge(), LassoLarsIC(), RandomForestRegressor(), xgb.XGBRegressor()]
```

	Name	Parameters	Train Accuracy Mean	Test Accuracy
0	KernelRidge	{'alpha': 1, 'coef0': 1, 'degree': 3, 'gamma':....	30.763	32.917
1	ElasticNet	{'alpha': 1.0, 'copy_X': True, 'fit_intercept'...	22.081	22.351
2	Lasso	{'alpha': 1.0, 'copy_X': True, 'fit_intercept'...	27.211	27.333
3	GradientBoostingRegressor	{'alpha': 0.9, 'criterion': 'friedman_mse', 'i...	12.301	12.442
4	BayesianRidge	{'alpha_1': 1e-06, 'alpha_2': 1e-06, 'compute_...	11.229	11.759
5	LassoLarsIC	{'copy_X': True, 'criterion': 'aic', 'eps': 2....	12.552	12.511
6	RandomForestRegressor	{'bootstrap': True, 'criterion': 'mse', 'max_d...	14.433	15.116
7	XGBRegressor	{'base_score': 0.5, 'booster': 'gbtree', 'cols...	12.542	12.421

2.5.2. Optimising

```
gs_alg = GridSearchCV(alg, param_grid = params_grid[0], cv = shuff, scoring
= 'neg_mean_squared_error', n_jobs=-1)
params_grid.pop(0)

#set name and parameters
model_name = alg.__class__.__name__
after_model_compare.loc[row_index, 'Name'] = model_name

gs_alg.fit(X_train, Y_train)
gs_best = gs_alg.best_estimator_
after_model_compare.loc[row_index, 'Parameters'] = str(gs_alg.best_params_)

#score model with cross validation
after_training_results = np.sqrt(-gs_alg.best_score_)
after_test_results = np.sqrt(((Y_test-gs_alg.predict(X_test))**2).mean())

after_model_compare.loc[row_index, 'Train Accuracy Mean'] = (after_training_
results)*100
after_model_compare.loc[row_index, 'Test Accuracy'] = (after_test_results)*1
00

row_index+=1
print(row_index, alg.__class__.__name__, 'trained...')
```

We implemented sklearn's GridSearchCV to do optimization - selection of parameters that maximize the score of the models.

	Name	Parameters	Train Accuracy Mean	Test Accuracy
0	KernelRidge	{'alpha': 0.1, 'coef0': 100, 'degree': 1, 'gam...	11.212	11.911
1	ElasticNet	{'alpha': 0.001, 'copy_X': True, 'fit_intercep...	11.219	11.912
2	Lasso	{'alpha': 0.0005, 'copy_X': True, 'fit_interce...	11.195	11.774
3	GradientBoostingRegressor	{'learning_rate': 0.1, 'loss': 'huber', 'max_d...	12.123	12.154
4	BayesianRidge	{'alpha_1': 1e-08, 'alpha_2': 5e-06, 'copy_X':...	11.229	11.759
5	LassoLarsIC	{'copy_X': True, 'criterion': 'aic', 'eps': 1e...	12.552	12.511
6	RandomForestRegressor	{'max_depth': None, 'max_features': 'auto', 'm...	13.696	14.114
7	XGBRegressor	{'booster': 'gbtree', 'colsample_bylevel': 0.2...	12.073	11.789

2.5.3. Stacking

```
for alg in models:

    gs_alg = GridSearchCV(alg, param_grid = params_grid[0], cv = shuff, scoring
= 'neg_mean_squared_error', n_jobs=-1)
    params_grid.pop(0)

    gs_alg.fit(X_train, Y_train)
    gs_best = gs_alg.best_estimator_
    stacked_validation_train.insert(loc = row_index, column = names[0], value =
gs_best.predict(X_test))
    print(row_index+1, alg.__class__.__name__, 'predictions added to stacking va
lidation dataset...')

    stacked_test_train.insert(loc = row_index, column = names[0], value = gs_bes
t.predict(xgb_test))
    print(row_index+1, alg.__class__.__name__, 'predictions added to stacking te
st dataset...')
    print("-"*50)
    names.pop(0)

    row_index+=1

print('Done')
```

1 KernelRidge predictions added to stacking validation dataset...

1 KernelRidge predictions added to stacking test dataset...

2 ElasticNet predictions added to stacking validation dataset...

The best average performing model is used as the meta-model for stacking, and the rest are the base estimators.

The meta-model is then fit and used to generate predictions

2.5.4. Ensemble

The results of the individually optimised models are combined together with the meta-model to create an ensemble.

```
ensemble = meta_model_pred*(1/10) + final_predictions['XGBoost']*(1.5/10) + fina
l_predictions['Gradient Boosting']*(2/10) + final_predictions['Bayesian Ridge']*(
1/10) + final_predictions['Lasso']*(1/10) + final_predictions['KernelRidge']*(1
/10) + final_predictions['Lasso Lars IC']*(1/10) + final_predictions['Random For
est']*(1.5/10)

submission = pd.DataFrame()
submission['Id'] = test_ID
submission['SalePrice'] = ensemble
submission.to_csv('final_submission.csv', index=False)
```

3. Experimental Results and Analysis

3.1. Experimental Setup

Programming Language	Python
Data Visualization tool	Jupyter NoteBook
Python packages	As shown in Section 1 of code implementation - main.ipynb
Implementation	As shown in Implementation Guidelines of Appendix

3.2. Comparison Schemes

To determine the correlation matrix between the features and the target variable, a heatmap was implemented as shown in 3.2.1, where a more correlated feature is indicated with a darker shade of color.



In order to identify relationships between values in the features, three types of graphs were implemented:

1. Box Plot Graph
2. Scatter Plot Graph
3. Bar Graph

These graphs provide us with more insights of the frequency of data that appears in each feature, as well as the most appropriate measure to process the features as inferred from the graphs.

3.3. Results and Analysis

The results from the experimental setup has seen an achievement score of a 75th percentile placing (937 out of 4146) in the competition ranking

934	▼ 150	Min Kwon		0.12008	2	2mo
935	▼ 150	Anup Mandvariya		0.12009	8	2mo
936	▼ 29	Tuan Do		0.12011	10	7d
937	▼ 151	de441ck		0.12012	7	1h

4. Discussion of Pros and Cons

Data cleaning is implemented to increase the value of data in analytics and decision making. This is done through removing anomalies and errors in the data given. The process itself aids in improving the predictive model for any machine learning algorithm. However data cleaning is very time consuming, especially when dealing with datasets with large number of data and features. The optimal way of cleaning the data is to first look at outliers that are strongly correlated to the target variable of the data set. These outliers can spoil and mislead the training models in processing the features which may result in longer training times and less accurate models.

Filling missing data is later done as a group by several methods; using mean/median/mode of the data column, or filling the missing values with a constant value such as 0 or null if the cell holds no value to the data itself. Automated filling of these missing values may quicken the entire cleaning process instead of manually looking through the data individually row by row, however comes at the expense of potentially losing significance of a value in a particular data row.

Boosting techniques such as XGBoost have proven to aid in the improvement of predictions in training models. These techniques are implemented by converting a set of weak learners into strong learners. In our experiments, we implemented XGBoost as one of our many algorithms for ensemble learning. XGBoost have several components that many other boosting algorithms are lacking, which makes it a popular candidate for usage in many data science competitions. However, XGBoost does not allow categorical inputs and only numerical inputs. This gives a slight disadvantage over general machine learning algorithms as some preprocessing has to be done to the dataset prior to usage.

5. Conclusions

5.1. Summary of project achievements

Data Preparation techniques have been implemented, which includes removing outliers as seen in the documentation ¹, and treating missing values in the data sets given.

Data Analysis has been done by introducing correlation matrix to give guidance on how to prepare certain features for modeling. Feature engineering has also been implemented, such as

- Transforming some numerical variables that seem categorical
- Label encoding some categorical variables that may contain information in their ordering set
- Box cox transformation of skewed features (instead of log-transformation)
- Getting dummy variables for categorical features

One of the four feature engineering procedures was chosen upon the visualisation of several plot graphs, which includes box plot graphs, scatter plot graphs and bar graphs.

Modelling of the data has been implemented by introducing various machine learning algorithms and ensemble techniques to ensure a more robust model with reduced variance, bias, and improvement in predictions.

5.2. Directions for improvements

Outliers that were treated in the experimental setup was only done on one feature as indicated by the organiser[9]. A more in depth study can be done by treating outliers from other features that has a strong correlation to the target variable, SalePrice.

Models were built based on traditional learning-based algorithms coupled with senseable techniques. Another approach would be to implement deep learning algorithms that would most probably have an improvement in the score, but comes with huge cost factors such as the amount of training data that is being fed into the neural network as well as the number of neuron layers.

References

[1] Serigne Stacked Regressions to predict House Prices

<https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard>

[2] Pedro Marcelino Comprehensive Data Exploration with Python

<https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python>

[3] Pedro Marcelino Using Categorical Data with One Hot Encoding

<https://www.kaggle.com/dansbecker/using-categorical-data-with-one-hot-encoding/notebook>

[4] Kernel ridge regression

http://scikit-learn.org/stable/modules/kernel_ridge.html

[5] Lasso and Elastic Net

<https://www.mathworks.com/help/stats/lasso-and-elastic-net.html>

[6] Bayesian Ridge

<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/idauction2/.g/web/glossary/bayesian.html>

[7] Lasso Lars IC

<https://www.quora.com/What-is-Least-Angle-Regression-and-when-should-it-be-used>

[8] Jason Brownlee. A Gentle Introduction to XGBoost for Applied Machine Learning

<https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>

[9] Ames, Iowa Assessor's Office. Alternative to the Boston Housing Data Set:

<http://ww2.amstat.org/publications/jse/v19n3/Decock/DataDocumentation.txt>

Appendix

The complete source code can be found in the following github repository:
<https://github.com/derrickpehjh/CZ4032-Data-Analytics-and-Mining-Project>

Datasets

The required datasets can be found in the following repository subfolder:
<https://github.com/derrickpehjh/CZ4032-Data-Analytics-and-Mining-Project/tree/master/inputs>

There are 4 files in total:

1. *data_description.txt* - This file contains detailed description of each feature that is present in the dataset. It also includes the description of specific values that are present in each feature.
2. *sample_submission.csv* - This file contains the required submission format that kaggle is able to accept so as to be evaluate the final score of the dataset.
3. *test.csv* - *This file is the test set used.*
4. *train.csv* - *This file is the training set used.*

Scripts/Source Codes

The entire script can be found in this IPython notebook :
<https://github.com/derrickpehjh/CZ4032-Data-Analytics-and-Mining-Project/tree/master/codes>

Implementation Guidelines

1. Installation of Python and Jupyter Notebook
2. Download/Clone the entire github repository
3. In the command prompt, type the following command to open up the graphical interface on the browser:
`python -m notebook`
4. Navigate to the folder of main.ipynb and open the file
5. Ensure you have installed all the python packages as stated in the first few lines of the file
6. Run the entire program