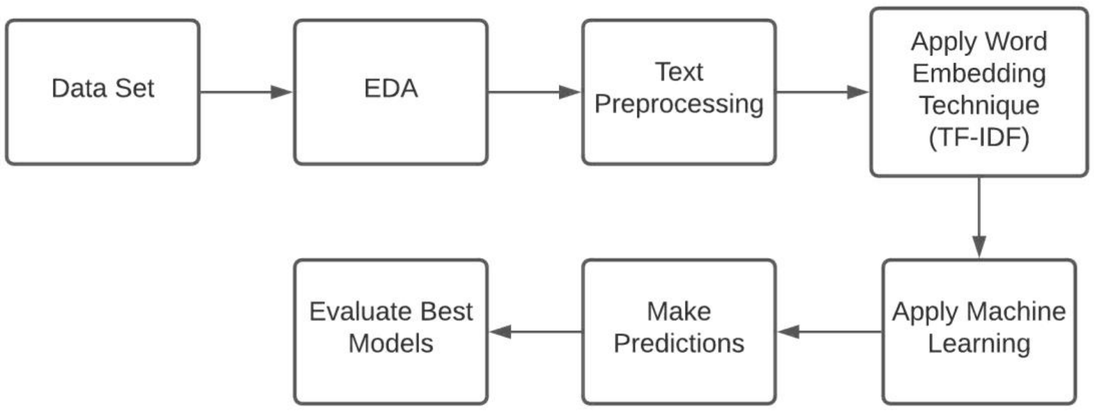
# Rental Listing Inquiry Prediction

## Objective:

To analyse a rental listing dataset, shared along with this assignment, and build a text analytics model to predict how much interest a new rental listing will receive.

## Methodology:



## Dataset:

The dataset contains **10857** samples and 10 features for each sample. The features are as following:

building_id

created

description

features

latitude

longitude

bathrooms

bedrooms

price

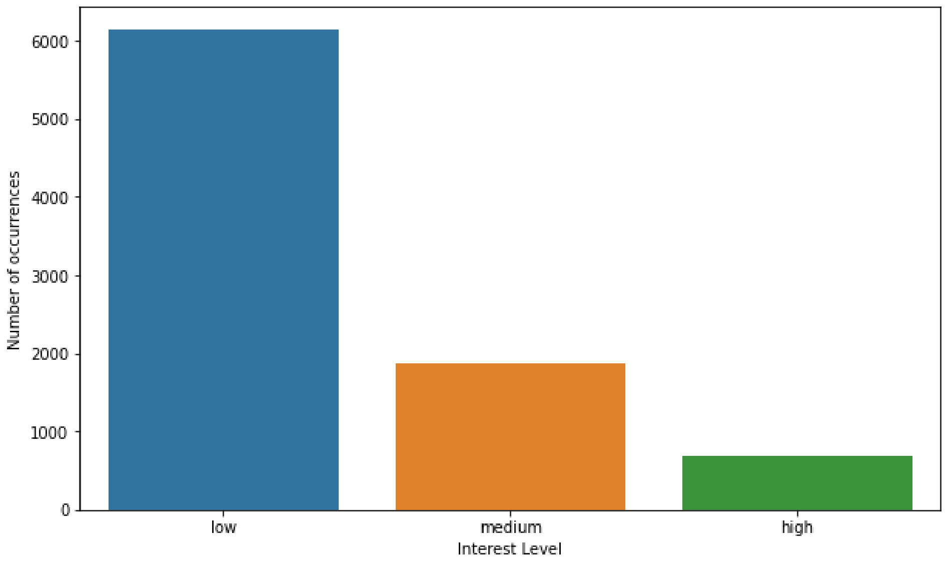
interest_level

Since there is no test data associated, the given file is manually random split into two files (80:20) there by creating a test file which is then again used for evaluating the model prediction on unseen data.

## Exploratory Data Analysis:

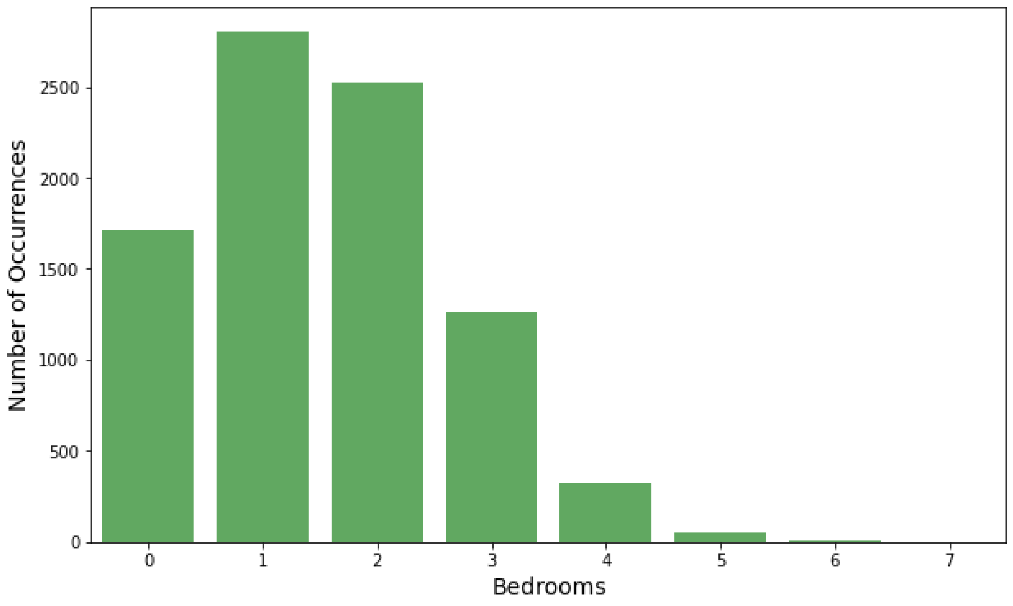
The given data set is being loaded into a python pandas data frame and the distribution of each of the available features as per the categorized labels are observed using a histogram.

**Interest level:**

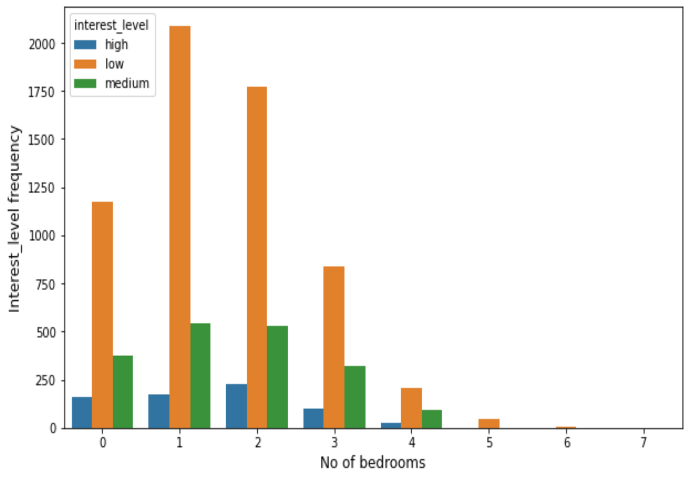


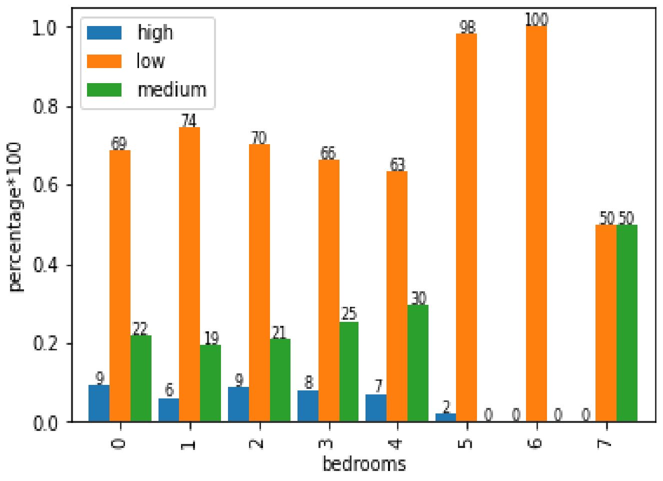
Interest level is low for most of the listings followed by medium and then high which is as expected.

**Bedrooms:**

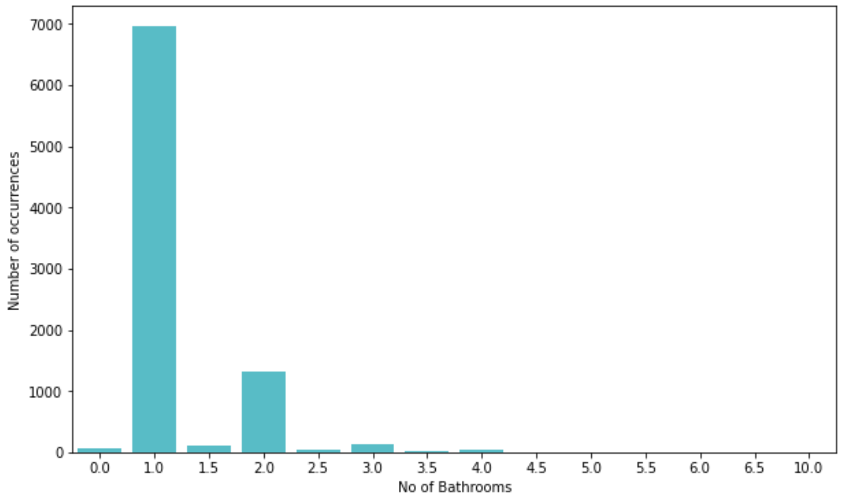


For each number of bedrooms (0, 1, 2...) we can count how many times high/low/medium occurs. From the graph below, the 1 bedroom occurs most, next by two bedrooms, and then 0 bedrooms. For each of these, low appears most, then medium and high appears least.

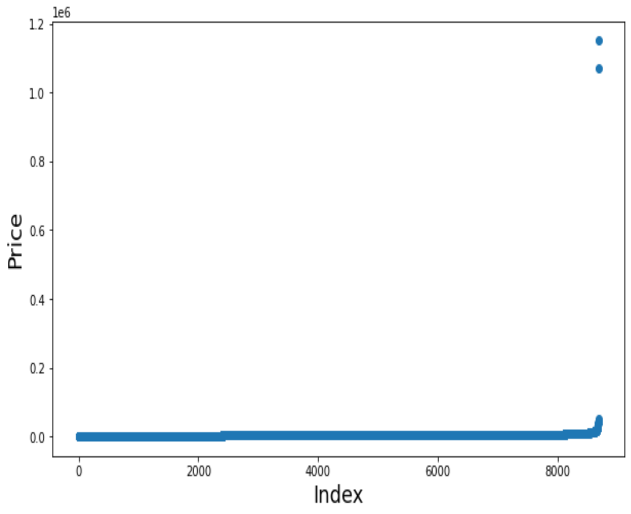


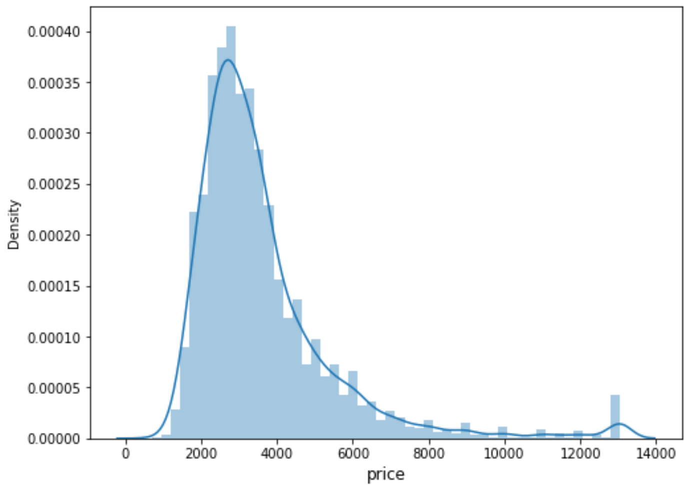


**Bathrooms**



**Price:**



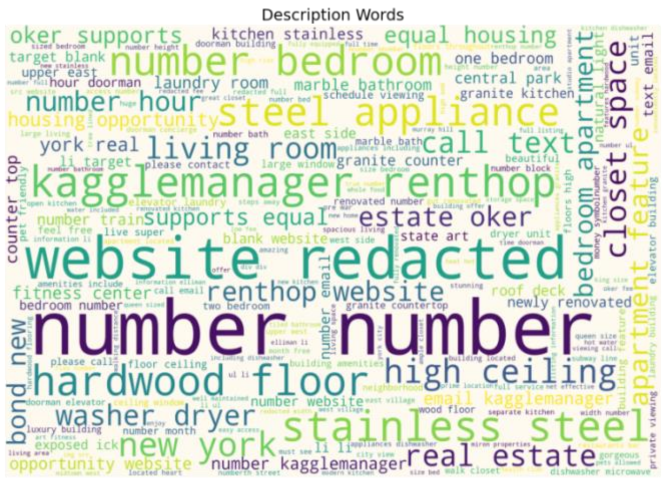


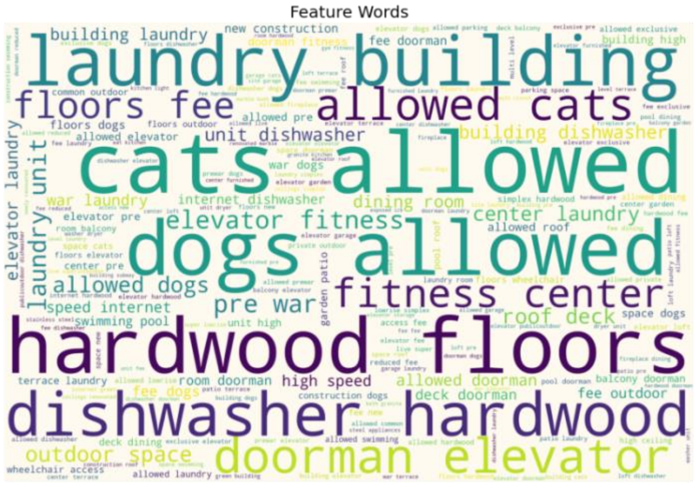
There were a couple of outliers observed when the distribution was plotted. These outliers were handled before going ahead with the model.

**<u>Text Pre-processing & Word Embedding:</u>**

Text pre-processing is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, text in a different case. When we talk about Human Language then, there are different ways to say the same thing, And this is only the main problem we have to deal with because machines will not understand words, they need numbers so we need to convert text to numbers in an efficient manner. Text data can be represented in vectorized format. Text pre-processing can be done by various NLP (Natural Language Processing) techniques. Machine Learning algorithms work with numbers only. So, there is a need to encode text data into the numeric format. Tokenization is a process of dividing text data into different parts. In-Text pre-processing stop words are removed. stop words are the words that are not useful for analysing the text data. For example, the words like is, was, that are stop words. After removing, stemming can also be applied. Stemming is a process of reducing the word to its stem. For example, the word “playing” can be changed as “play”. After that word embeddings can be done, where the words are changed as vectors of real values. Here Count Vectorizer technique has been used. CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

**Word clouds for description and features:**





<u>**Training and Validation data sets using Borderline SMOTE:**</u>

```
# Spliting data into train and cross validation
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(train_X,train_y, test_size=0.2, random_state=18)
from imblearn.over_sampling import BorderlineSMOTE
sm = BorderlineSMOTE(random_state=42)
x_train y_train = sm.fit_resample(x_train, y_train)
```

<u>**Machine Learning techniques:**</u>

After converting text into real valued vectors, various machine learning classifiers like naive bayes, random forest decision tree etc have been applied once the data has been split into training and testing sets. Since the data at hand is imbalanced, imblearn's Synthetic Minority Oversampling Technique has been used to up sample the minority data class.

**K-Nearest Neighbours:** K-NN is a simple but efficient machine learning classifier, which is based on distance calculation. It identifies the k-nearest neighbours and based on the count of neighbour’s class; the new data point is classified.

**Logistic Regression:** Logistic Regression uses logit function and sigmoid function for classification tasks. The output variable is predicted based on s-shaped curve.

**Random Forest classification:** Random Forest classifier takes the opinion of several decision trees to decide the class of a new data points. It is an ensemble approach.

**Gradient Boosting for classification:** GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions.

**XGBClassifier:** XGBoost provides a wrapper class to allow models to be treated like classifiers or regressors in the scikit-learn framework. This means we can use the full scikit-learn library with XGBoost models. The XGBoost model for classification is called XGBClassifier.

**Hyperparameter tuning and Cross Validation:**

Each of the above classifier models are preceded with hyperparameter tuning to get the best possible parameter that should be used for the given set of data. Also, each of the classifier models is cross validated with CalibratedClassifierCV. This class uses cross-validation to both estimate the parameters of a classifier and subsequently calibrate a classifier. With default ensemble=True, for each cv split it fits a copy of the base estimator to the training subset and calibrates it using the testing subset. For prediction, predicted probabilities are averaged across these individual calibrated classifiers.

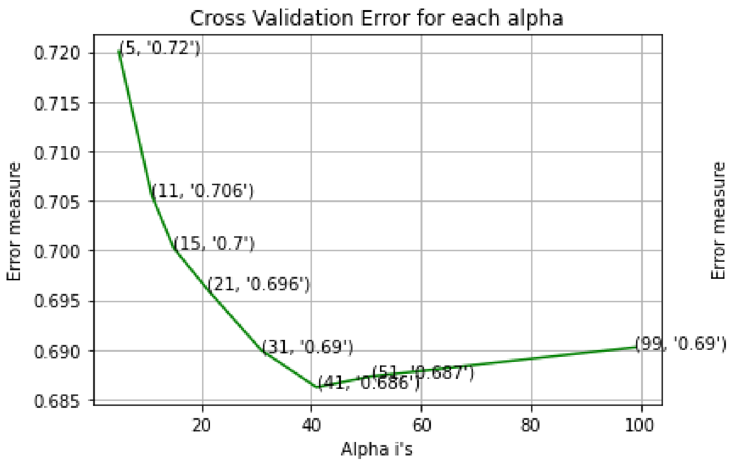
**Evaluation Metrics:**

**Log-loss:**

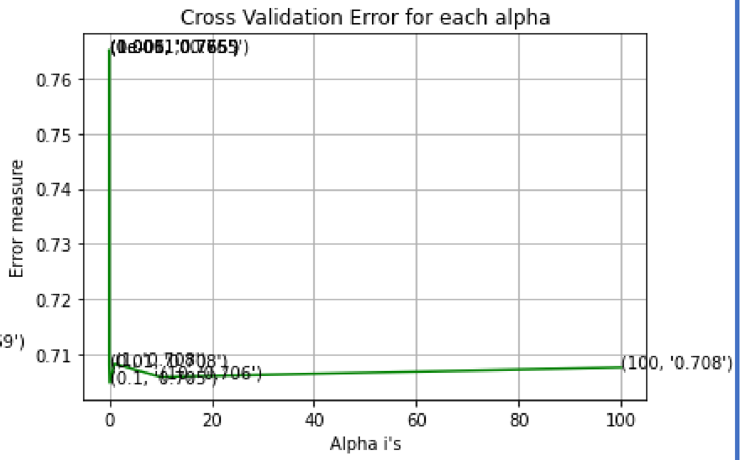
Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value.

$$Logloss_i = -[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$

*where i is the given observation/record, y is the actual/true value, p is the prediction probability, and ln refers to the natural logarithm (logarithmic value using base of e) of a number.*



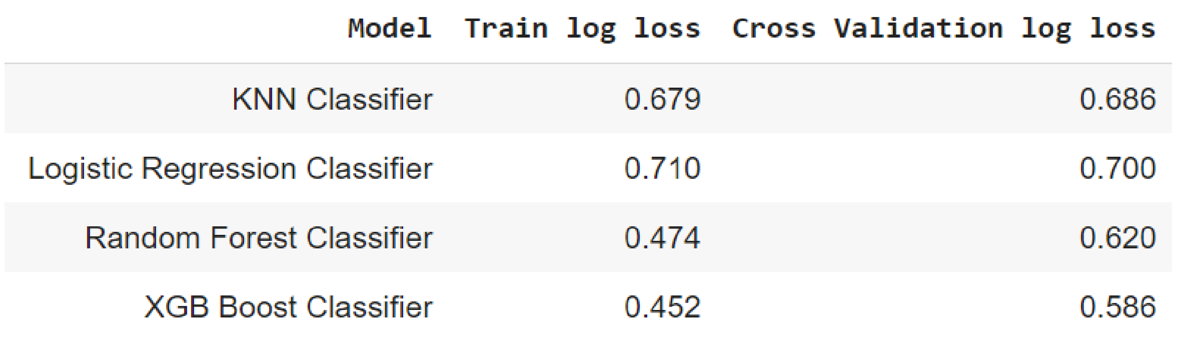
*KNN*



*Logistic Regression Classifier*

Each of the classifier model is run on the training data and tested on the validation data. Then it is cross validated with the test data set which was split from the original file earlier.

**<u>Results of Experiments:</u>**



| Model | Train log loss | Cross Validation log loss |
|---:|---:|---:|
| KNN Classifier | 0.679 | 0.686 |
| Logistic Regression Classifier | 0.710 | 0.700 |
| Random Forest Classifier | 0.474 | 0.620 |
| XGB Boost Classifier | 0.452 | 0.586 |

As we can see from the above results, XGB Boost classifier model gives the best result with the least log loss scores of 0.452 and 0.586.