Project: Explore the factors that imply if one house is occupied

1.Introduction:

How can we find that if a house has people living in it or not? This is an important question that can be used in the research of the housing vacancy rate. In this project report, we will explore three datasets, whose names are test data, validation data, and training data, that contain different features that are possibly connected with the occupancy of a house and find the most suitable model for our datasets.

2.Discussion about our features:

Looking to our datasets first, there are several features that can be chosen from in our study, and we will discuss the possibility and how much they can influence the housing occupancy.

'Date': Handling time data can be very difficult, and the time of our data is not very relevant to occupancy. Although workers and students may leave their houses in daylight and return at night, the occupancy still varies a lot on the same time as there are so many possibilities in it. So, we decide to drop this data in our data analysis.

'Temperature': Conventionally, temperature in an occupied house is higher than the average temperature overall as people will making a lot of CO2 and thus causing the greenhouse effect to raise the temperature. However, as there are so many ways for people to control the temperature, especially we have the ultimate controller—air conditioner, this feature is not that reliable.

'Humidity and Humidity Ratios': Those two features are similar to the temperature, as they tend to rise when people are in a house due to the breathing, but there are still many factors that can influence these features when we considering human activities.

'Light': Usually very reliable as most of the people need to open lights when they are living in a house, but there is a possibility that someone forgets to close the light when they are out, and thus causing the inaccuracy in prediction.

'CO2': Intuitively the most distinguishable feature as people always breathe and releasing tons of CO2. So occupied houses usually have a higher concentration of CO2.

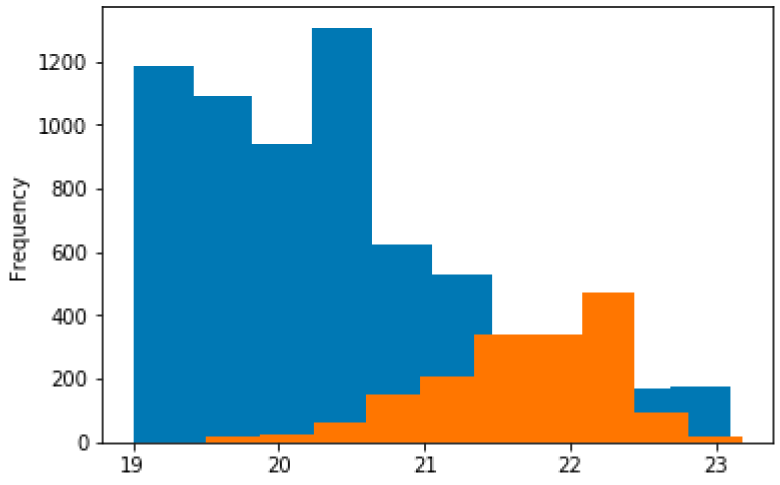
According to our discussion, we think that 'Light' and 'CO2' are the most helpful features in distinguishing the occupied houses and vacant houses. We need look in the data deeper to justify our discussion.

3.Simple EDA: Analysis about our data:

To find some features have great differences between occupied or not, we draw histogram based on the training dataset. Note that the blue histograms are histograms for the empty houses and the orange histograms are for occupied houses.

a) Temperature:

```
Occupancy
0    AxesSubplot(0.125,0.125;0.775x0.755)
1    AxesSubplot(0.125,0.125;0.775x0.755)
Name: Temperature, dtype: object
```

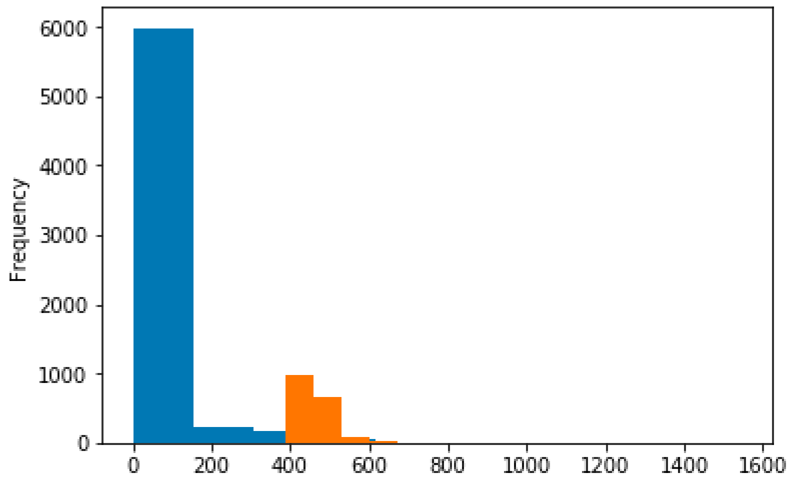


pic No.1

You can find that the average occupied temperature is higher than the empty temperature, and the occupied temperature is left skewed

b) Lightness

```
Occupancy
0    AxesSubplot(0.125,0.125;0.775x0.755)
1    AxesSubplot(0.125,0.125;0.775x0.755)
Name: Light, dtype: object
```

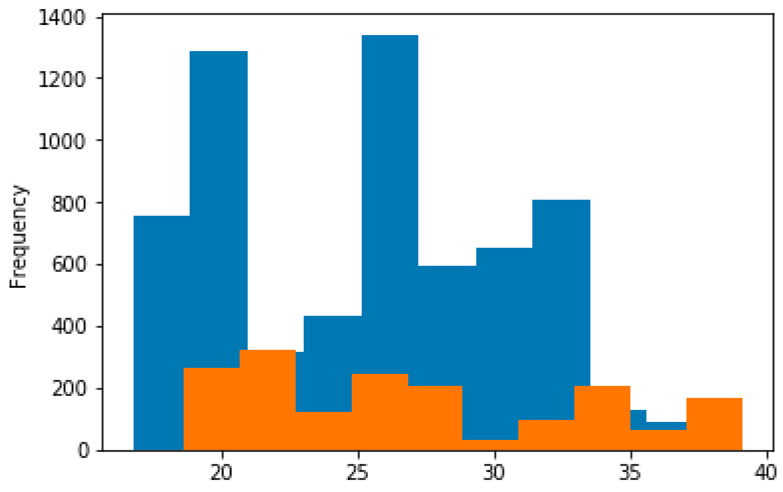


pic No.2

There is a clear boundary between empty and occupied, which is 400 Lux.

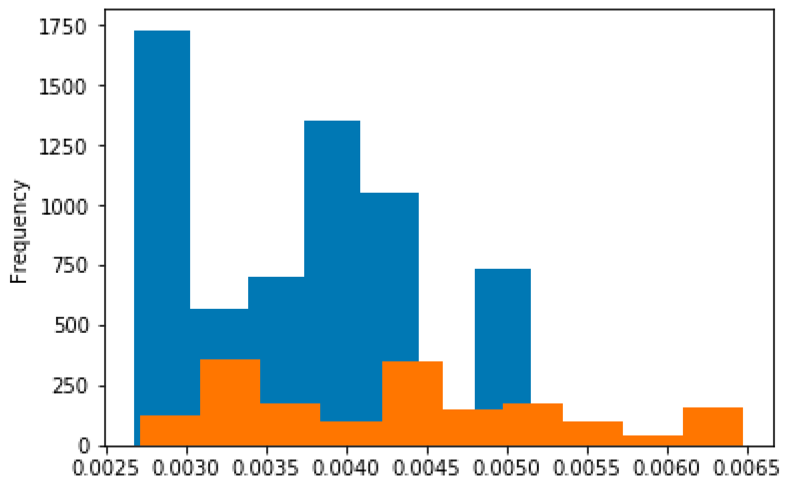
c) Humidity and Humidity Ratio:

```
Occupancy
0    AxesSubplot(0.125,0.125;0.775x0.755)
1    AxesSubplot(0.125,0.125;0.775x0.755)
Name: Humidity, dtype: object
```



pic No.3

```
Occupancy
0    AxesSubplot(0.125,0.125;0.775x0.755)
1    AxesSubplot(0.125,0.125;0.775x0.755)
Name: HumidityRatio, dtype: object
```

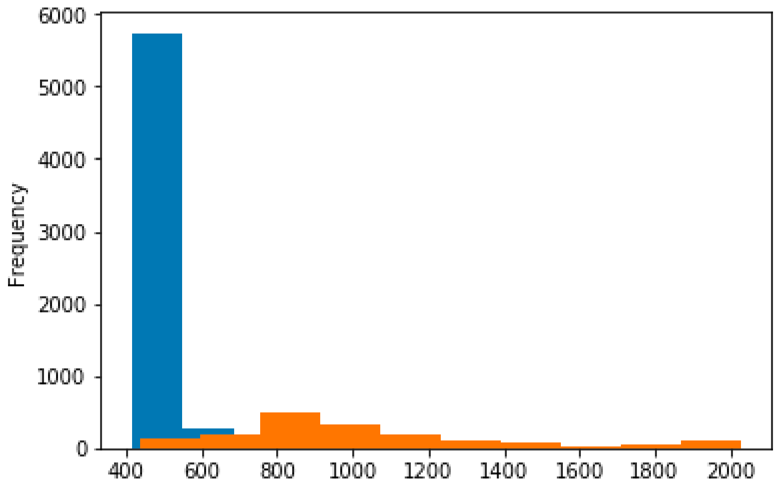


pic No.4

These two properties are not very good in differentiating occupancy, as occupied houses have those two approximately uniform distributed and empty houses tend to have lower data for those two.

d) CO2:

```
: Occupancy
  0    AxesSubplot(0.125,0.125;0.775x0.755)
  1    AxesSubplot(0.125,0.125;0.775x0.755)
  Name: CO2, dtype: object
```



pic No.5

As we can see, while occupied houses have their data left-skewed distributed, the empty houses C02 concentrate between 400 and 600 ppm

e) Summary:

We calculated the ratio of occupancy which is 0.79(empty) :0.21(occupied), so there is no need to worry about FPR or FNR in classification. From the graphs above, we can find that the influence of every parameter is ranked by:

Light>CO2>Temperature>Humidity Ratio≈Humidity.

Among of this, Light has a clear boundary at 400 Lux while CO2 has a concentration of empty houses between 400ppm and 600 ppm. We should pay attention to these two parameters if we want to do classification.

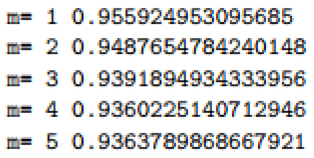
4. Modeling and Tuning parameters:

Now we are beginning to build our own model to make prediction about occupation according to our training data set. In this part, we are using sklearn package in python to build our basic models and tuning parameters according to the accuracy scores obtained by testing with validation data set. Note that the sklearn functions have a lot of default parameters, so if we select a sklearn model, we are selecting a series of parameters based on the sklearn documentations. You can check those parameters by checking the corresponding documentations.

1) Random Forest:

This is a go-to model if you want to do a classification from multiple possible features. The theorem of this model is simple. You build a forest of decision trees (decision tree is a tree model classifier that give the classification result based on a series of features given by the training set) based on a limited number of the features and find the best result according to the results from the trees. We used the default Random Forest Classifier first with 100 estimators (trees), and the result is very good at the first glance: the accuracies are 0.94 for the validation data set and 0.96 for the test data set. However, we are wondering if we can go to further if we tuning some important parameters here. At first, we try to use more estimators, and we try 200, 500, and 1000 estimators, but the result does not change much. Then, the parameter we choose to tune is the max_features, also known as m in the theorem. In a random forest classifier, a proper number of max_features can help the classifier to reduce the correlation between each tree as it forces every tree to make classification based on m random features (m is less than the total number of the features) to make more variation and thus the model become more reliable.

To make this selection, we write a loop in python that calculate the mean accuracy for every different m by 100 times. In other words, we build 100 random forest model for every possible m here (there are 5 in total) and train them by training data set and then calculate the accuracies using validation data and then calculate the means at last. Our result is shown below:



```
m= 1 0.955924953095685
m= 2 0.9487654784240148
m= 3 0.9391894934333956
m= 4 0.9360225140712946
m= 5 0.9363789868667921
```

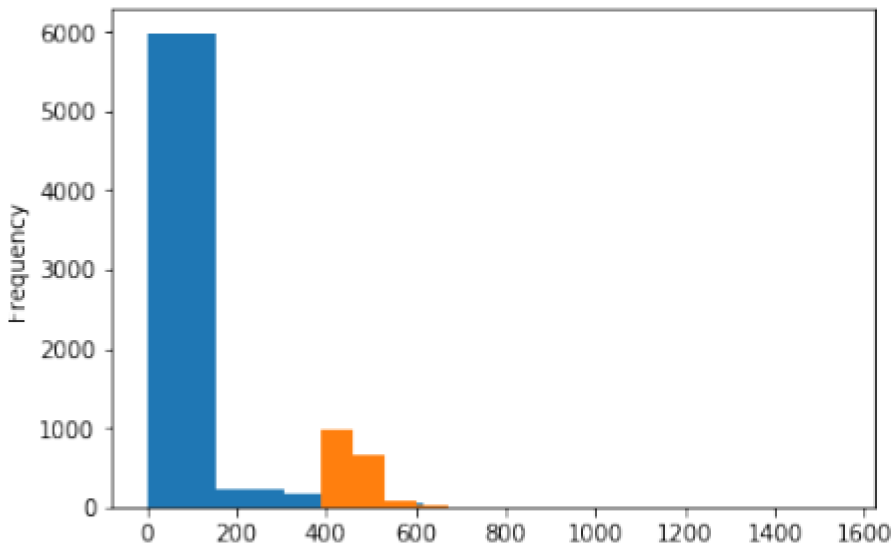
pic No.6

It is clear that when m=1 we have the best result. So, our final random forest model uses 100 estimators (to save time) and max_features of 1, and the final accuracy for test data set is 0.973.

2) Naïve Approach: Dividing the data according to the lightness

The accuracy we get from the random forest model seems very good, but can we have a better one? Wait, if we look into the data set clearly, we can find that the occupancy looks like to have a very clear boundary based on the lightness of the room!

```
: Occupancy
  0    AxesSubplot(0.125,0.125;0.775x0.755)
  1    AxesSubplot(0.125,0.125;0.775x0.755)
  Name: Light, dtype: object
```



pic No.7

Looking back to the lightness histogram, as we find in our previous analysis, it seems that all houses with lightness< 400 Lux are empty and houses with higher lightness are occupied! So, what if we build a naïve classifier based on the discovery?

We write our naïve classifier by just looking into the lightness column and state that lightness that are less than 400 Lux is empty and lightness that at least 400 Lux is occupied. The accuracy scores are surprisingly good, 0.9767 for our validation set and 0.9929 for test data set! It seems that we do not need to make effort to make fancy models for this classification!

3) LDA:

According to our naïve model, we can find that there some clear boundaries in some of our features, and this indicates that this training data set is easy to be discriminated. Thus, we can try to use LDA to classifier the data set. LDA is making a decision from a linear boundary, and this boundary is somehow clear in our data set. So, though our data is not perfectly normally distributed, which is not the ideal situation that LDA assumes, we can still use LDA to achieve a nice classification result.

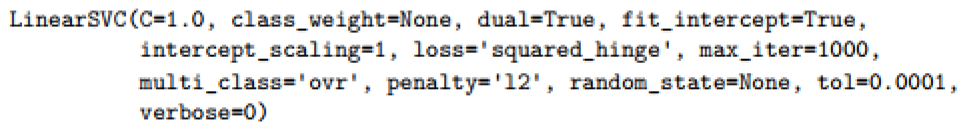
We use the sklearn default LDA first, and use least square solutions. The result is better than the random forest classifier, as the accuracies are 0.979 for our validation set and 0.988 for our training set. It seems that this model is a little worse than the naïve model, but we can improve it by tuning the parameter.

We consider the prior probabilities between the empty and occupied data. We try multiple priors, but the influences we make for the validation data set is very small. However, when we adjust our priors to a extreme portion, which is 0.99 empty : 0.01 occupied, the accuracy for test data set increase to the 0.9921, which means that LDA is now almost the same accurate as the naïve model. We are surprised that as we use the training probabilities (which is 0.79:0.21), the results do not improve, but when we use these extreme prior probabilities, our classification improves. It may because that occupied houses are very easy to be recognized than the empty houses so we need extreme priors to reach a better result.

4) Linear SVM:

Talking about linear differentiable data, SVM is another choice when we doing the binary classification. SVM find the hyperplane and its margin in the data set to divide them into the two categories we want, and it is quite robust to the behavior of observations that far away from the plane compared to LDA. In this case, we use the sklearn linearSVC model to build our linear SVM model.

We try the default setting first, which is:



```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Pic No.8

The accuracy we get is very good at our first try, which is 0.978 for the validation set and 0.9926 for the test set. It seems that it can be the best model if we tuning the parameters properly!

We raise our C to find if we can improve our result. C is the max number of violations of our dividing of hyperplane, which means no more than C observations are allowed to at the wrong side of the hyperplane. When C is large, our tolerance is high and the margin of our dividing is large. We carefully raise the C to 2 first, and the accuracies for validation set and test set rise to 0.9786 and 0.9929. However, as we keep raising the C, the accuracies start to fall, and thus we choose C=2 to be the best C here.

5) Conclusion:

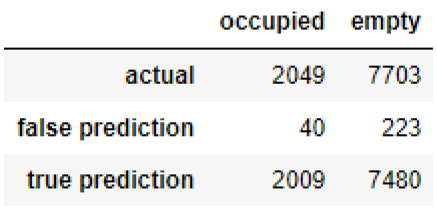
Comparing all the models we try above, we can find that LinearSVC with C=2 has the best accuracy. This is because that SVM is more robust to the observations that are far away from the plane, and C allows more adjustment for our margin to make a better classification. However, the LDA, naïve model, and the linearSVC actually have very close performs that are all very good, and this is because that our data sets have some very clear boundary over the lightness feature (400 Lux). Therefore, naïve classification, supervised linear boundary division and linear support vectors classification all perform very good on this classification task. In the other words, all of 3 are very good model for our data sets.

If we need to choose the best model here, we prefer to choose the naïve model we made in 2). First of all, all our models are trained/obtained by training set, so they can only apply to the data set with the similar distribution of the training set. If we randomly reassign the variable names in our test data set, all our models will be useless, so the naïve model is at least reliable as the other models. Secondly, the naïve model has a very good accuracy, very close to those gotten by LinearSVC and LDA. Thirdly and the most important one, the naïve model is very easy in both theorem and implication and it can potentially save you a lot of time. So, we choose the naïve model which takes all houses with lightness that <400 Lux as empty and the others as occupied.

6) Extra information: table reports for every model after tuning parameters

Those are table reports for the test data set:

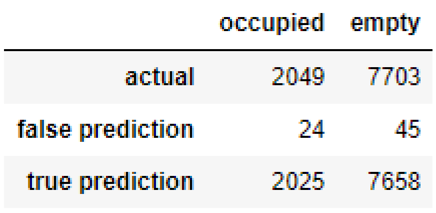
a) Random Forest, m=1



	occupied	empty
actual	2049	7703
false prediction	40	223
true prediction	2009	7480

pic No.9

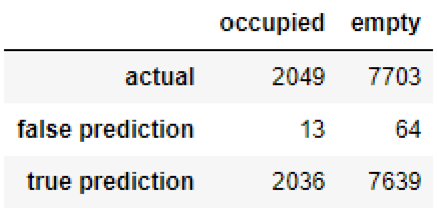
b) Naïve model



	occupied	empty
actual	2049	7703
false prediction	24	45
true prediction	2025	7658

pic No.10

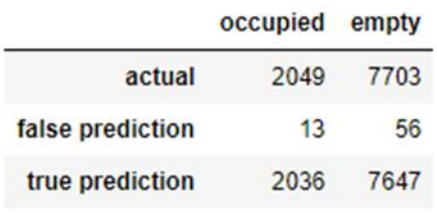
c) LDA, priors={0.99:0.01}



	occupied	empty
actual	2049	7703
false prediction	13	64
true prediction	2036	7639

pic No.11

d) LinearSVC (SVM), C=2



	occupied	empty
actual	2049	7703
false prediction	13	56
true prediction	2036	7647

pic No.12

5. Conclusion:

After we doing all of the analysis and modelling above, we now reach the conclusion of how to classify the occupancy of a house. The answer its simple: by looking at its light in the house. According to our analysis, this method reaches 0.9929 accuracy for test data set, among one of the best results in classification methods. So, next time, if you want to know if the house is occupied, you can just look if the light is on, and you can accurately find your result!

Reference

1. *sklearn.discriminant_analysis.LinearDiscriminantAnalysis*, https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html
2. *sklearn.svm.LinearSVC*, https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
3. *3.2.4.3.1. sklearn.ensemble.RandomForestClassifier,* https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
4. *pandas.DataFrame.plot,* https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html
5. *pandas.DataFrame.groupby,* https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html
6. *pandas.DataFrame,* https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
7. *numpy.array,* https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html