**HR Analytics Project**

****

**Introduction:**

This article in on the title ”HR Analytics Project- Understanding the Attrition in HR”, a project on machine learning which we will be doing in python. Human resource analytics (HR analytics) is an area in the field of analytics. It refers to the analyzing of the human resource department of an organization with an aim to improve the performance of the employees. We will be going through the entire steps that will be needed for the completion of the project and we will understand them thoroughly. The topics that we will be covering are:

* Problem Definition
* Data Analysis
* EDA Concluding Remarks
* Pre-Processing Pipeline
* Building Machine Learning models
* Concluding Remarks

**Problem Definition:**

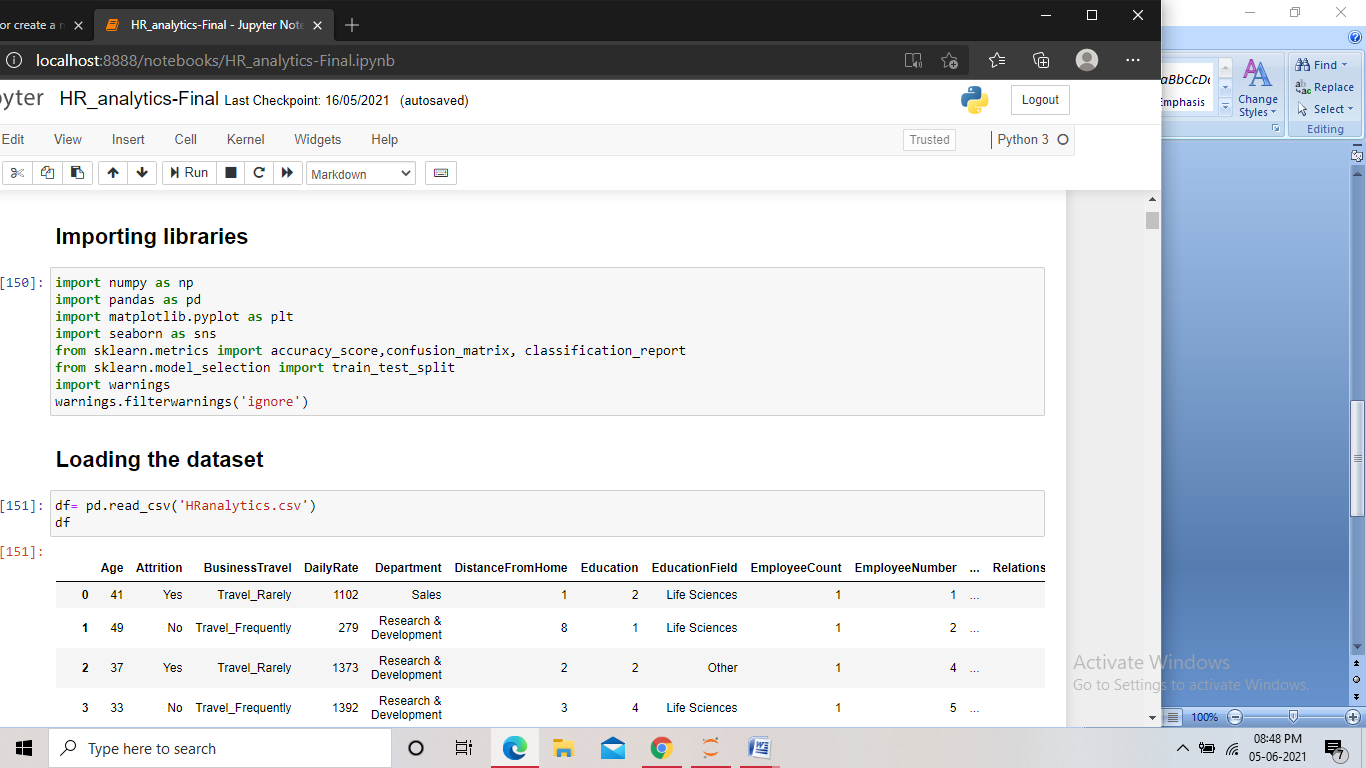
HR Analytics in an area under the field of analytics which aims to provide insights into the processes of an organization by collecting the data and using it to analyze the performance of the organization in various fields and then finding ways to improve the performance.

Attrition in HR generally refers to the gradual loss of employees overtime. This is problematic for the organization, as this increases the expense of the company with the new hiring, paper works, and training the new employee. It also affects the company in way that an experienced employee is replaced with a new hire, who will take time to get used to the company and gain experience and errors are more likely to occur if you constantly have new workers. Also it is more concerning if the business is related to facing the customers, as customers feel more comfortable interacting with familiar faces.

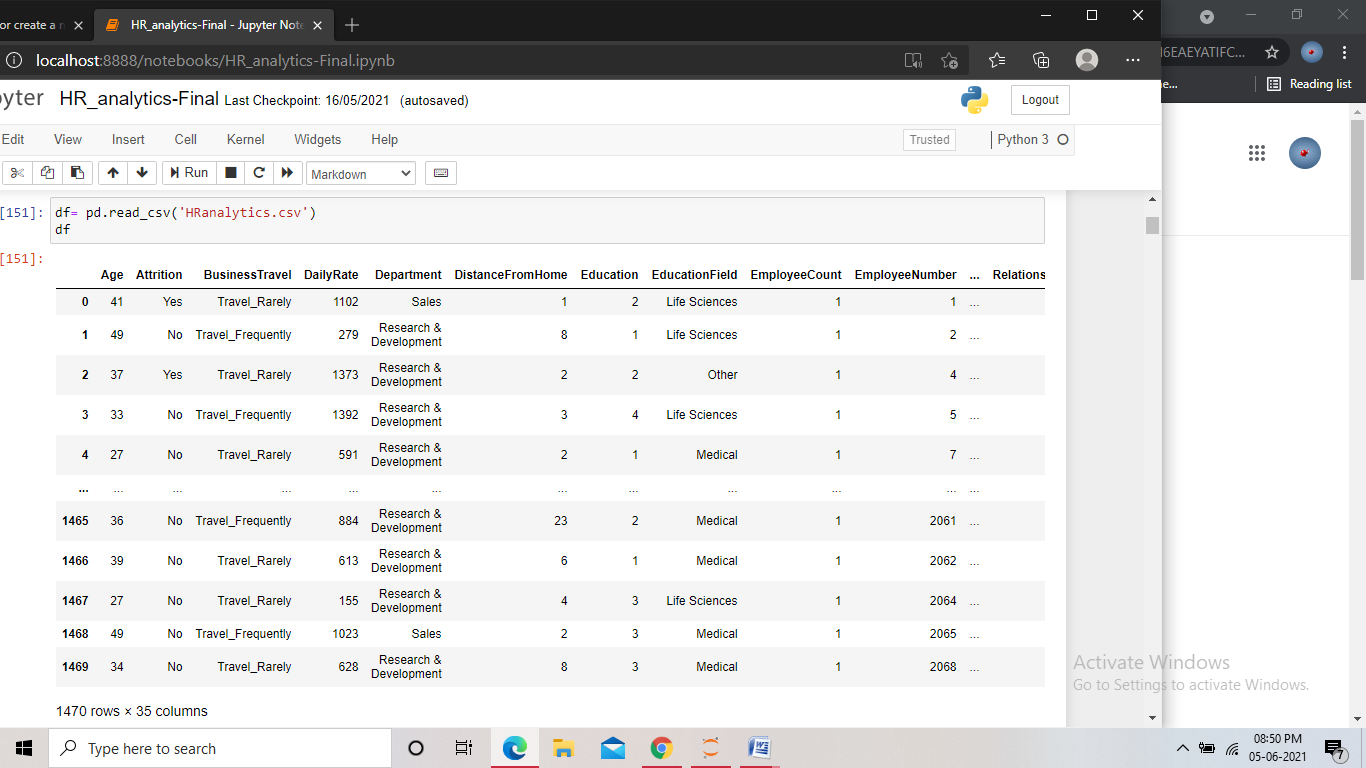
Here, we need the help of machine learning to find a solution for this problem. We can use all the previous employee data of the organization along with the attrition data, i.e. If the worker had attrited or not. The data can then be feed into the machine and we can process the data and get an understanding on when and why workers choose to attrite and create a model that can predict using the new employee data that whether the person is likely to attrite or not.

**Data Analysis:**

The project provides us with a dataset that contains various employee data, for example age, department, field, working hours etc. And along with all this data we have the attrition data, whether the employee has attrited or not. We first import all the required libraries for the project and load the dataset into the notebook.

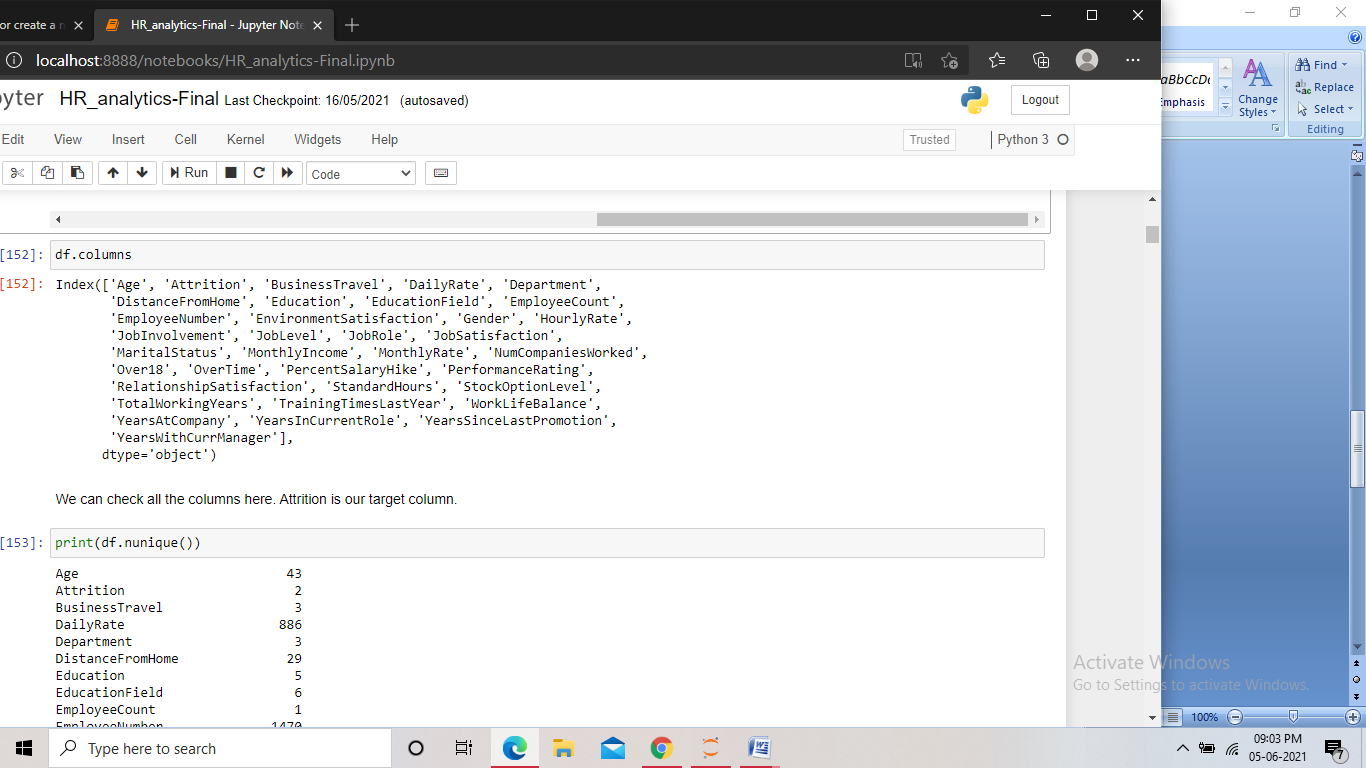


Screenshot of the code for importing the libraries



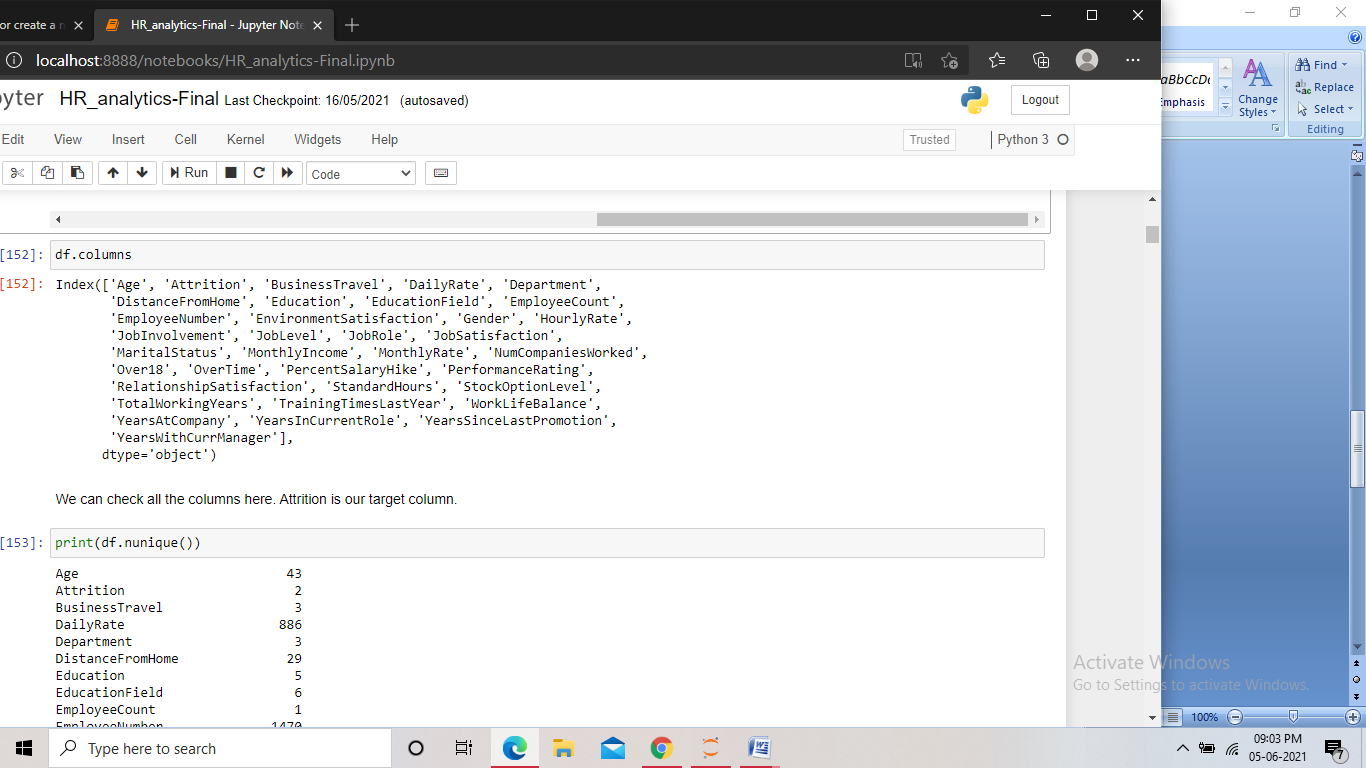
Screenshot of the dataset

The dataset is not that huge but contains many features, using which we have to build the model. The “Attrition” column is the target here. The target data has only two values that are yes or no, so it is clear that that the problem is a classification problem. We then check all the columns in the dataset.

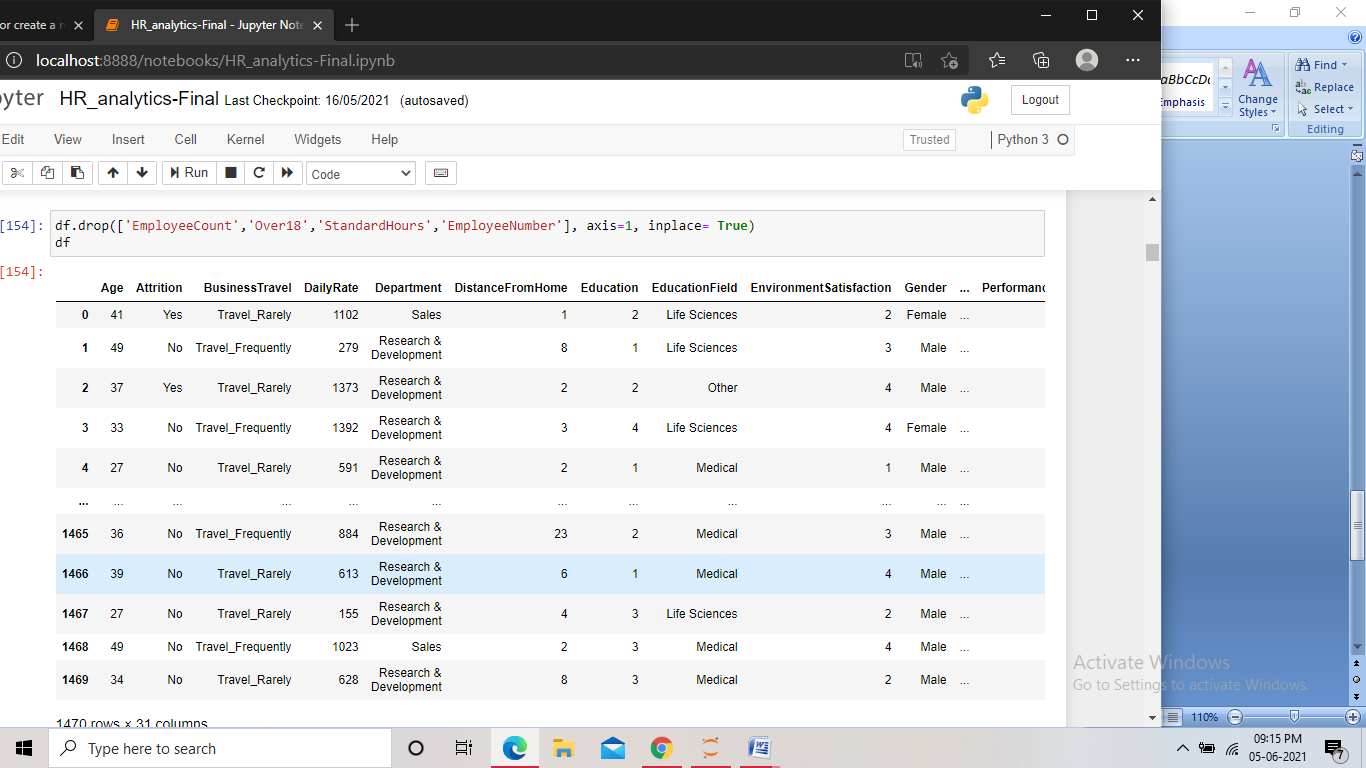


Screenshot of the code

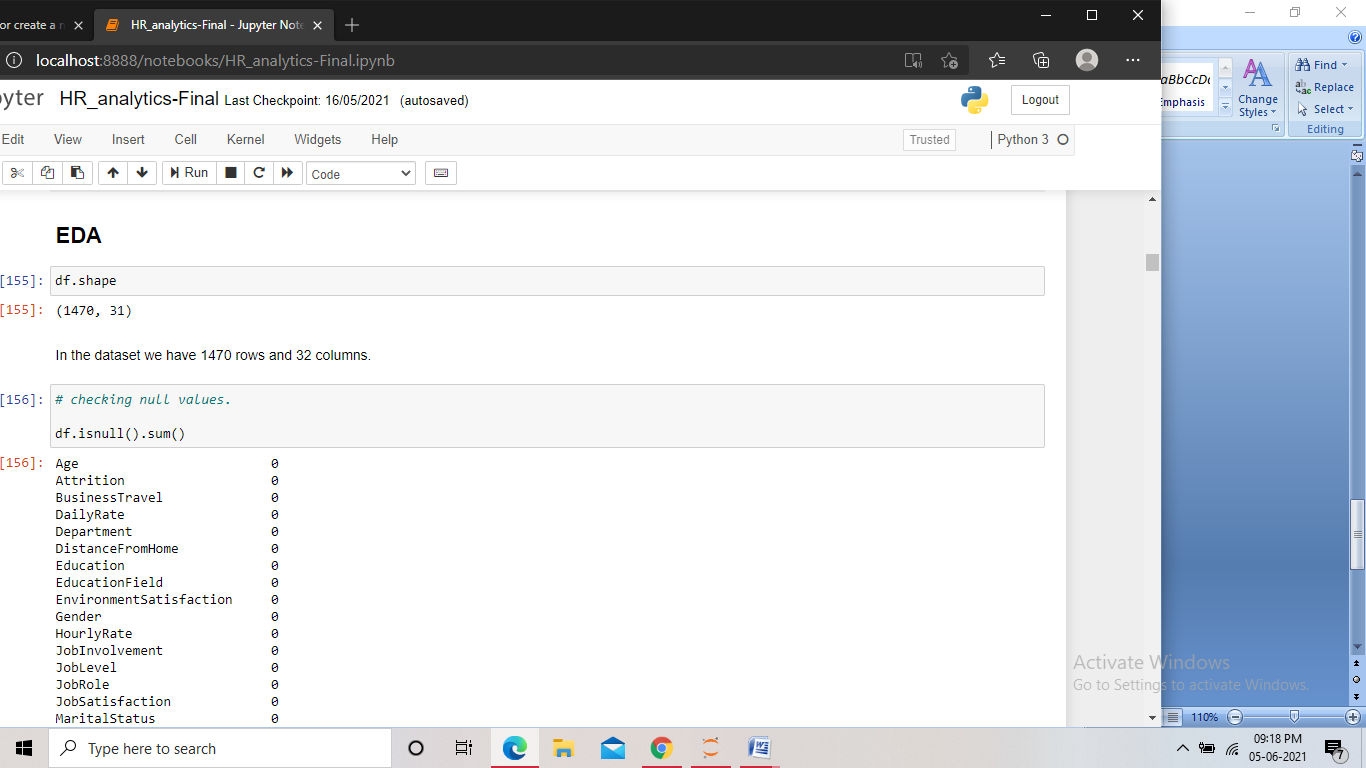
**EDA Concluding Remarks:**

* First of all we try to get an idea about the data present in the columns, we start by checking the number of unique values present in them.

Screenshot of the code

We found some columns having just one value for all the records. As these columns have no impact on the model creation, so we drop those columns. 

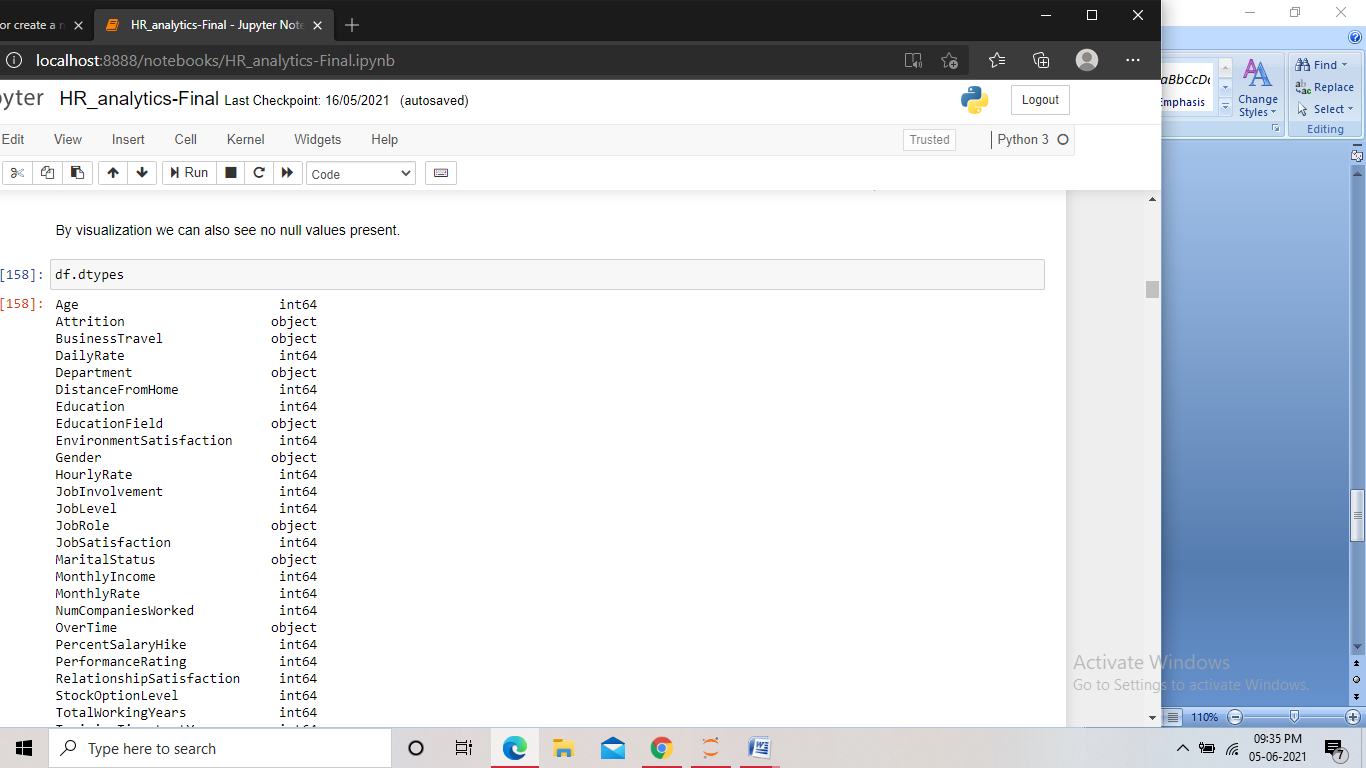
Screenshot of the code

* Now we check the shape of the dataset and also check if there are missing values present.

Screenshot of the code 1

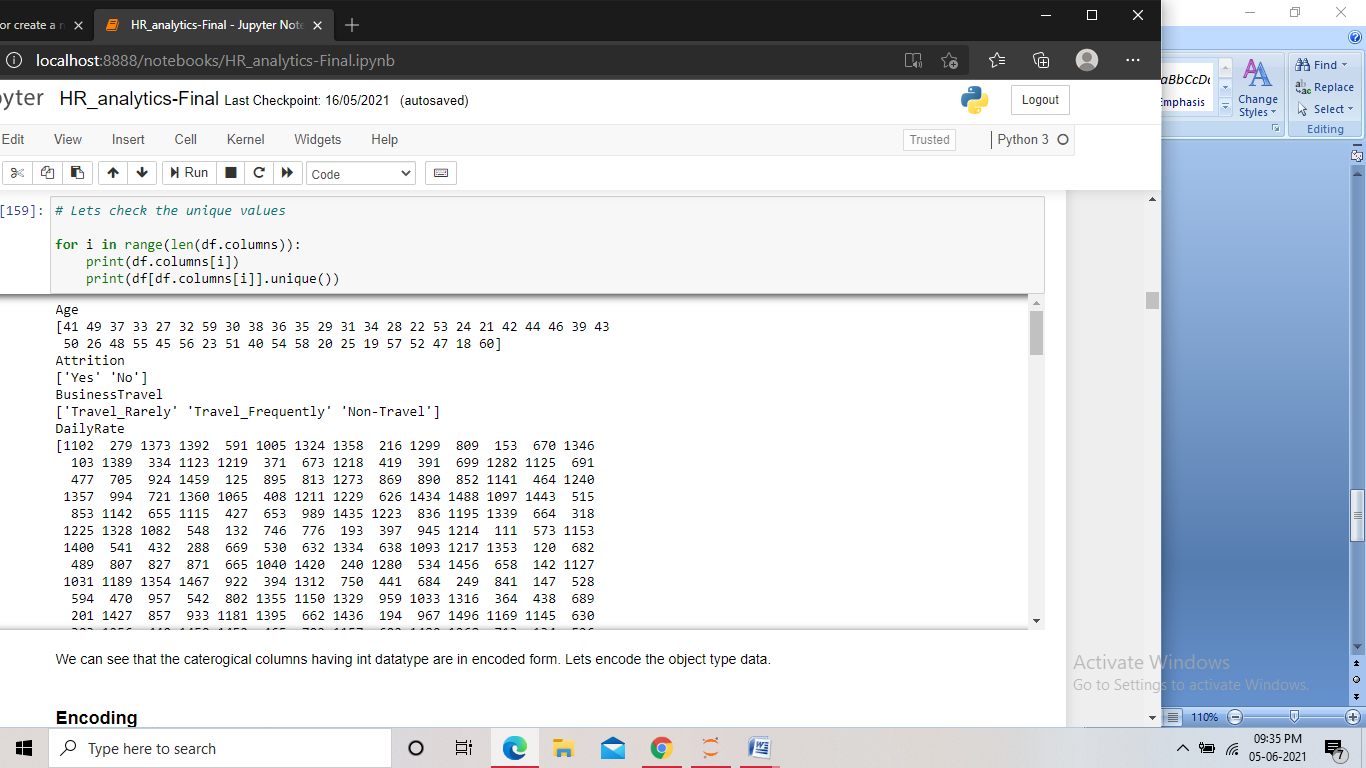
The dataset then contained 1470 rows and 31 columns. Any by check for missing values we found none.

* After this we checked the data types of the columns. And the unique values present in them.



Screenshot of the code

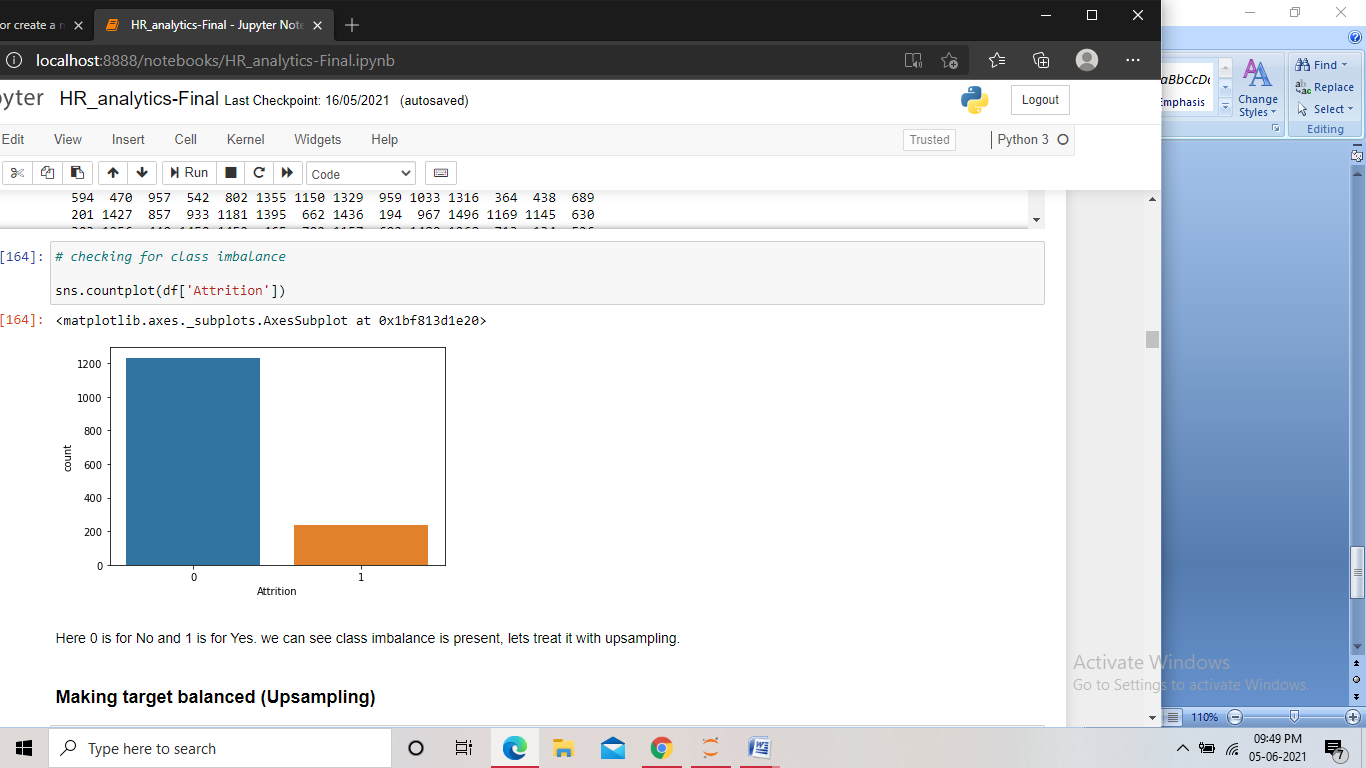
We found the many columns having datatype as object including the target column, we will need to encode them in upcoming steps.



Screenshot of the code for checking unique values for all columns

We found that the majority of the columns contain categorical type value and many are in encoded format.

* We then checked the distribution of the data in the columns using different plots, and checked for class imbalance in the target column.



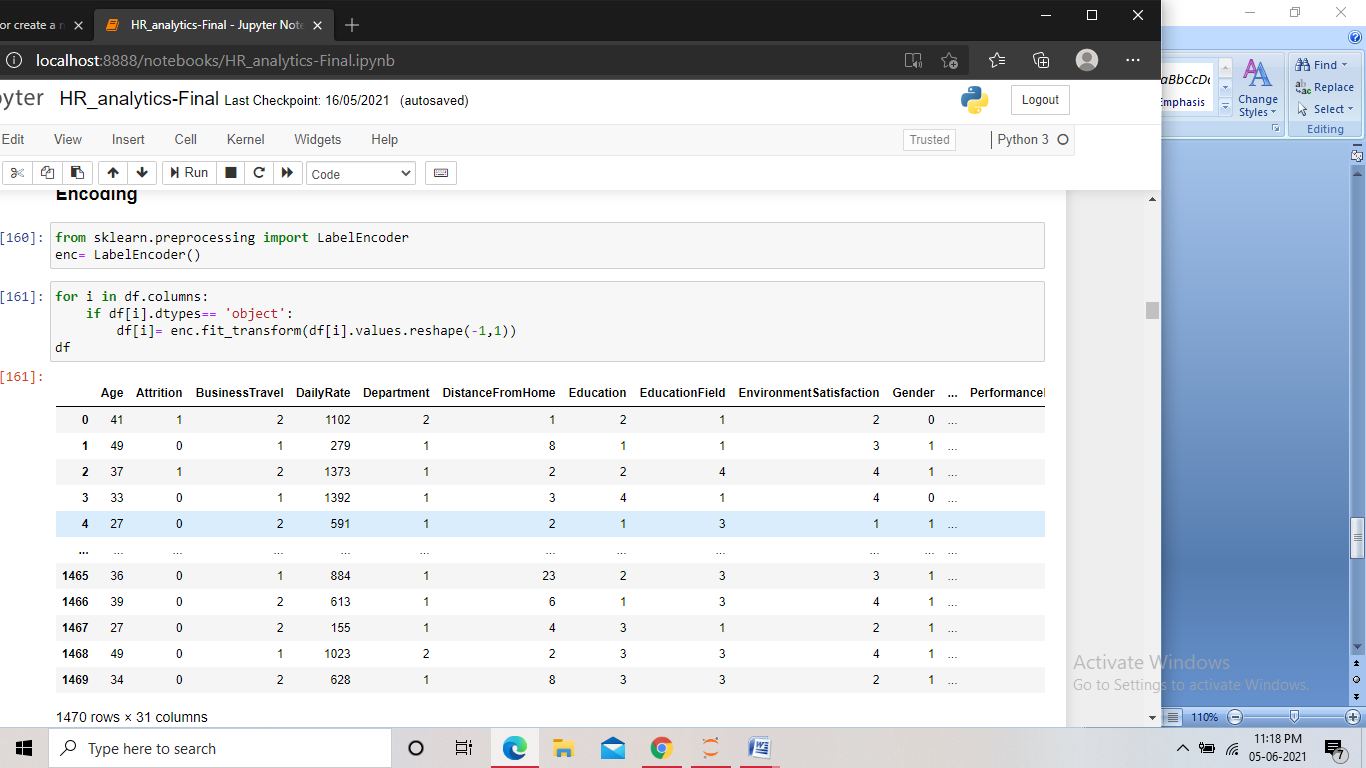
Screenshot of the code

We found some class imbalance in the target column, with the value “No” being very high. This needed to be treated.

**Pre Processing Pipeline:**

With the EDA, we got the information about all the features in the dataset and found some issues in the dataset that needed to be treated.

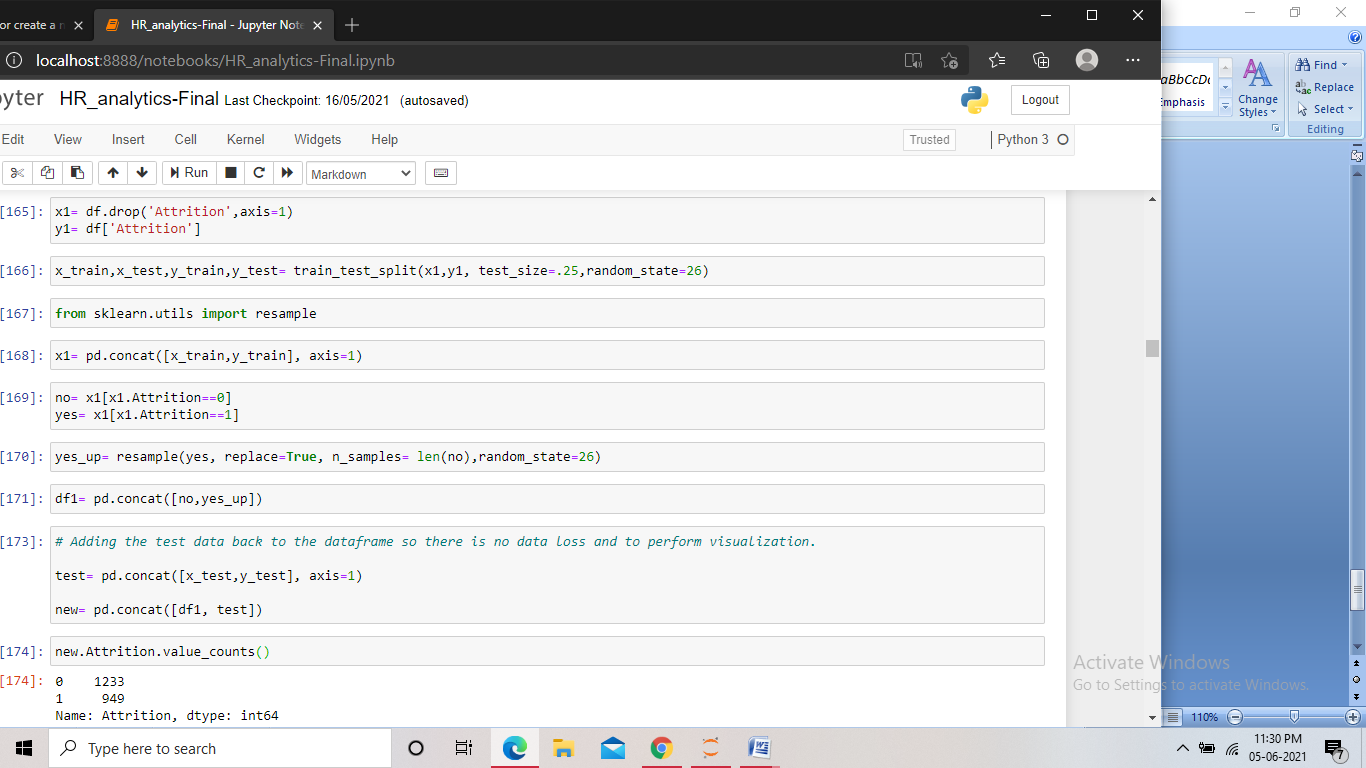
* As we found many columns, including the target have the datatype as object, we encoded them so that we can proceed further.



Screenshot of the code for encoding

Here we used the Label encoder for the encoding. We imported the encoder and assigned it to a variable and then using condition to select only the object datatype columns, we performed the encoding.

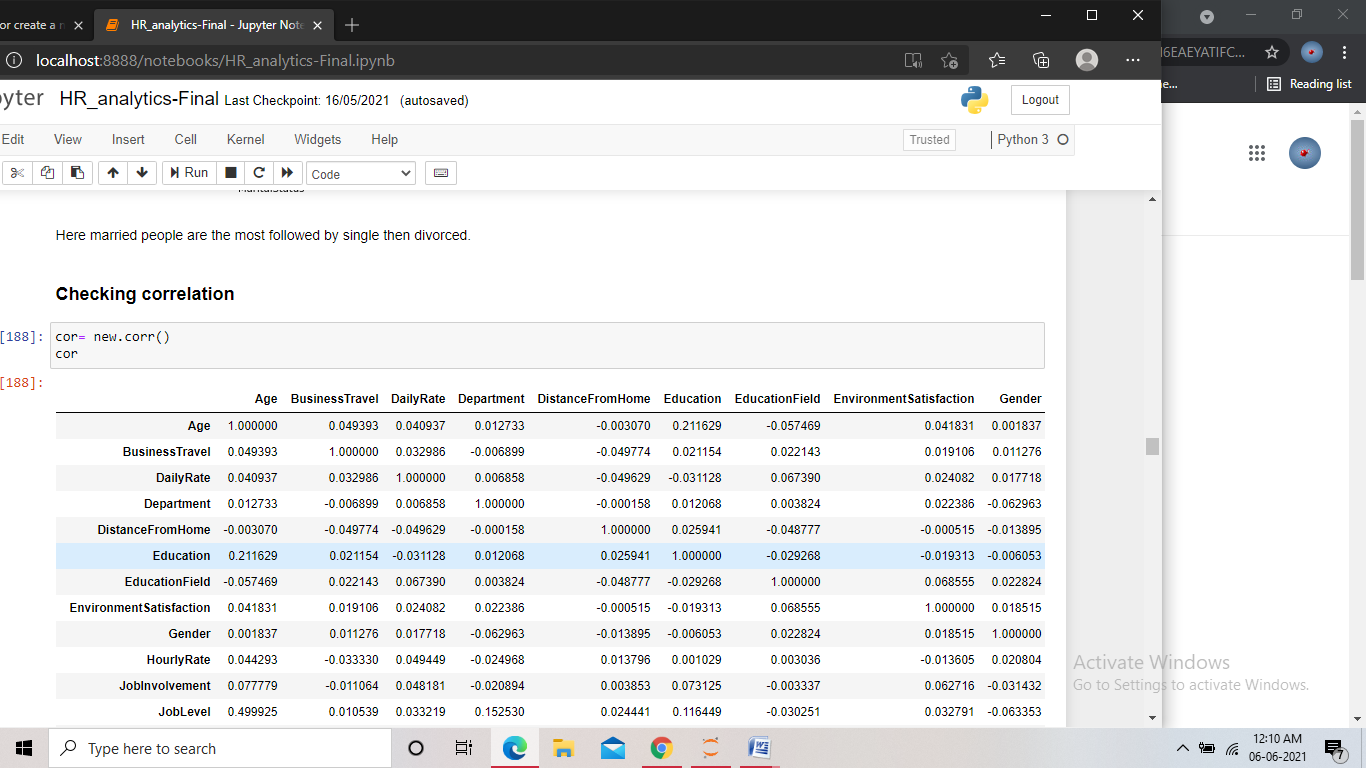
* We also found the target column having class imbalance problem, so we needed to make the target balanced, so we performed up sampling to the dataset.



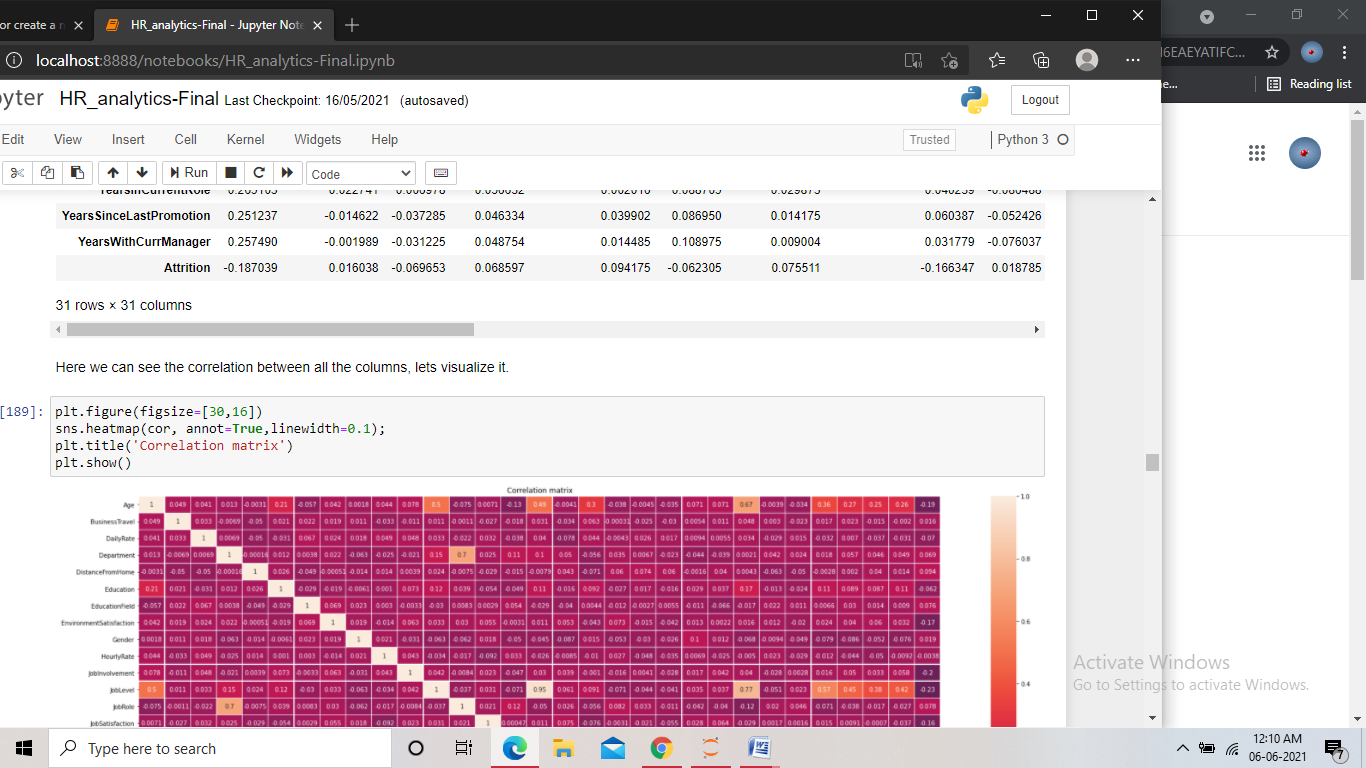
Screenshot of the code

At first we split the data into features and target and performed train-test split. We then imported the resample method. After this we concatenated the train feature and train target and kept the data in a variable and from that variable we extracted the data having target ‘yes’ and ‘no’ in two different variables. After that we up sampled the ‘yes’ data using the resample method to the size of the ‘no’ data. We then concatenated the up sampled data and the ‘no’ data. And finally we add the test data back to the dataframe so that there is no data loss, and as the test data is very small, the target will still be balanced.

* After this we checked the correlation in the dataset and visualized it.

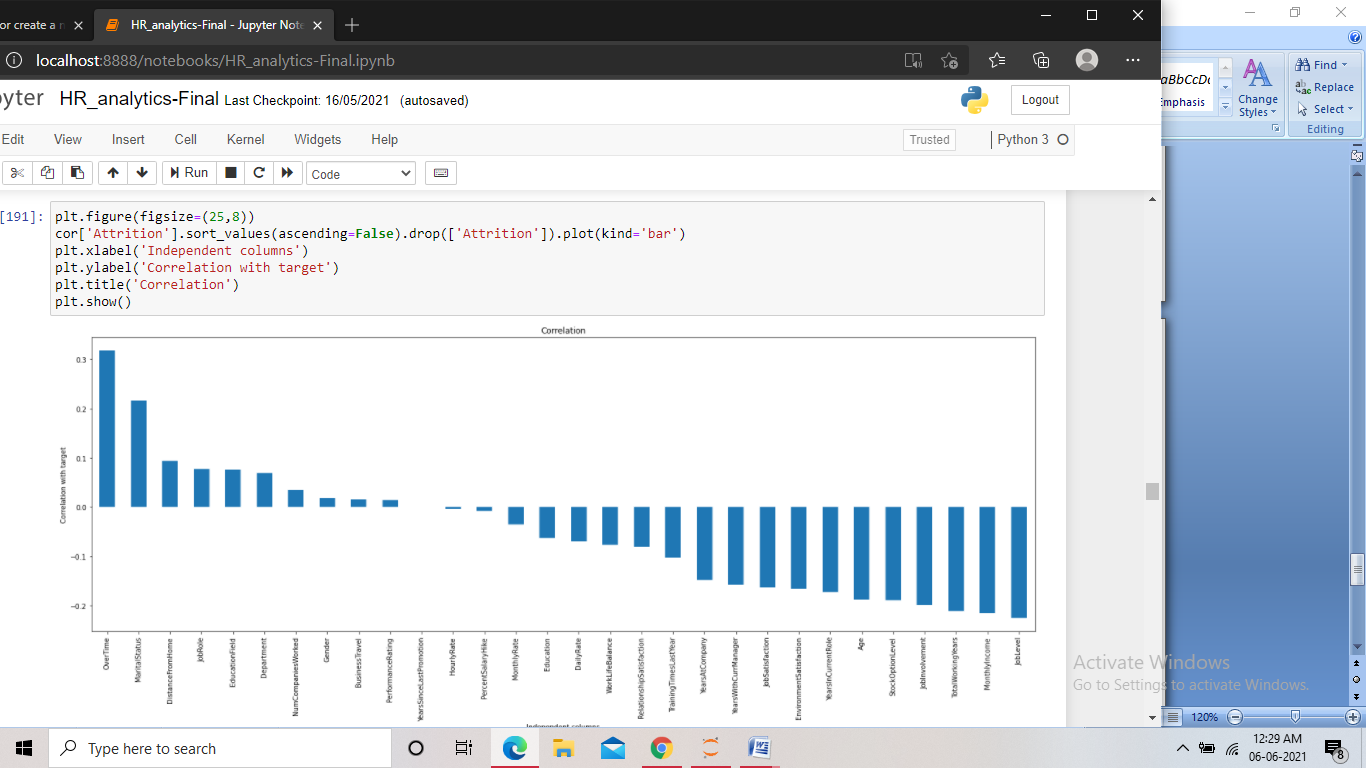


Screenshot of the code for checking correlation



Screenshot of the code for visualizing the correlation

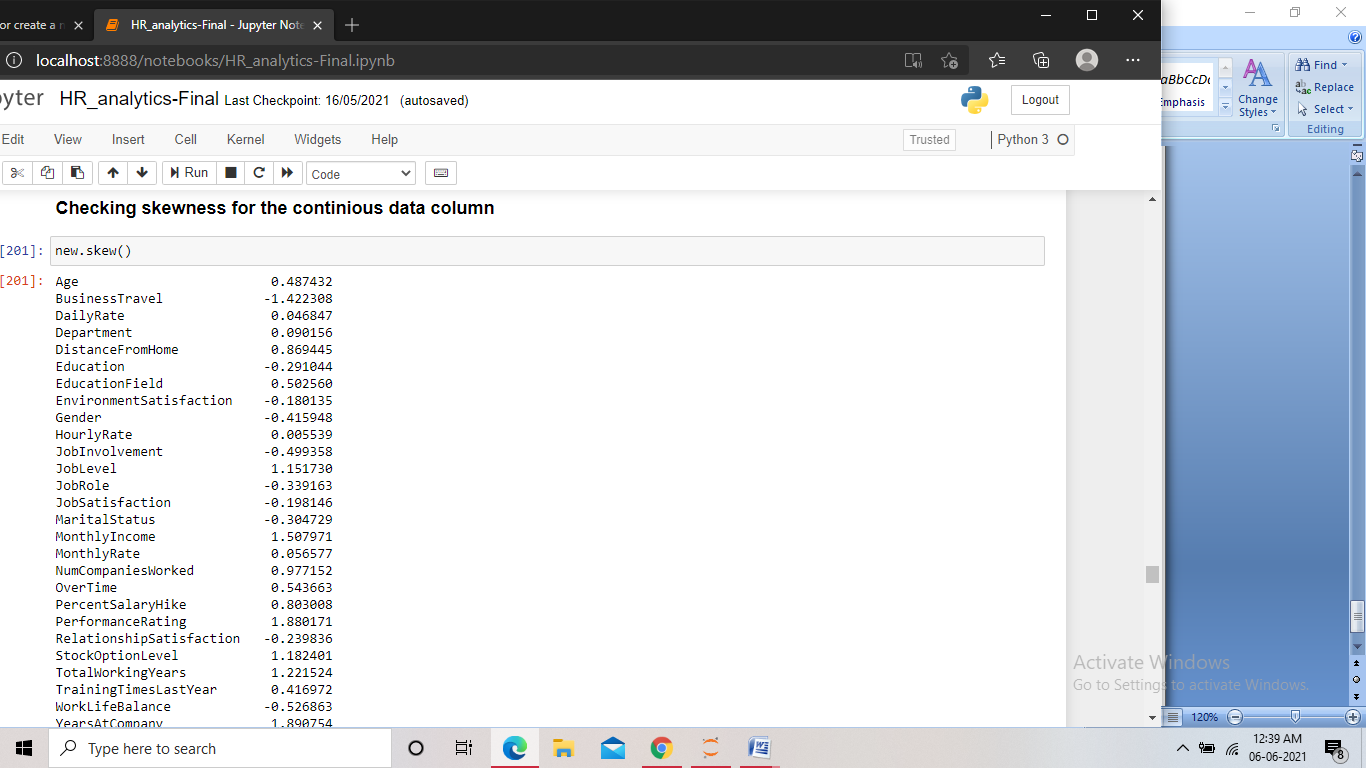
We checked the correlation of the dataset and then using heatmap we visualized the correlation. Then we separately checked the correlation the target with the independent columns and performed bi-variate analysis between the columns.



Screenshot of the code

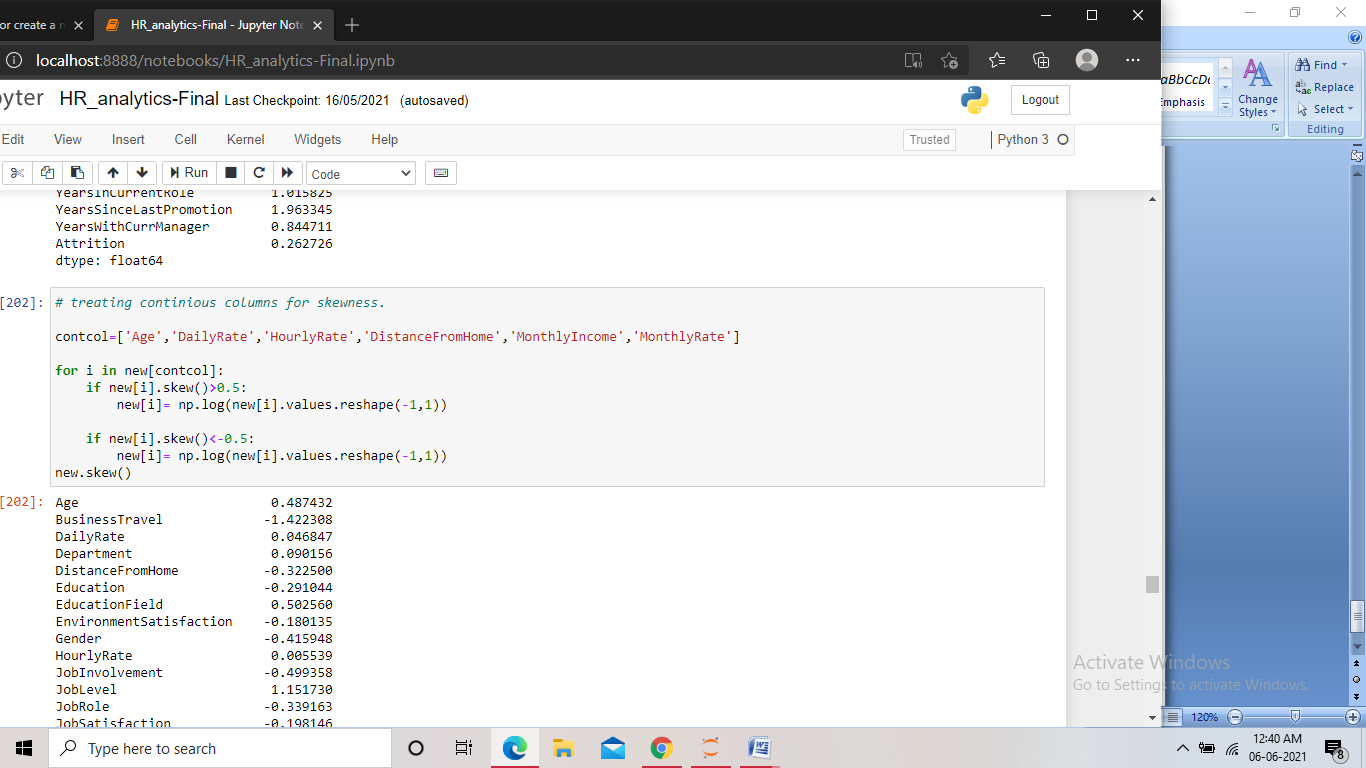
We found the “overtime” column giving the highest correlation with the target column, which showed that more the employees do overtime, more is the chance of attrition. On the other hand “Job level” showed the most negative correlation, which meant higher the job level, less is the chance of attrition.

* We then checked for skewness in the continuous data columns in the dataframe.



Screenshot of the code

Taking the threshold of (>-0.5 and <0.5), some skewness were found in few columns, we then treated them.

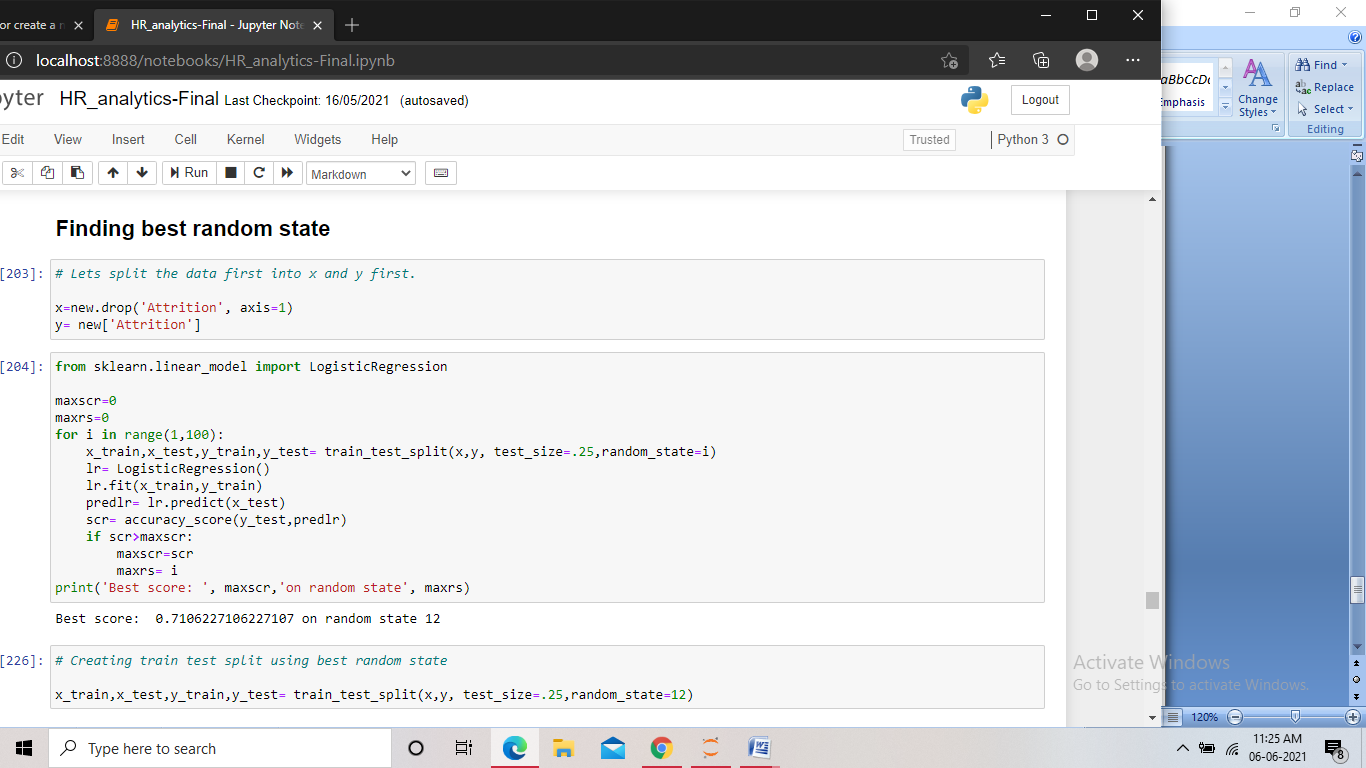


Screenshot of the code

For the continuous data columns where we found skewness, we used log transformation technique to remove the skewness in the data.

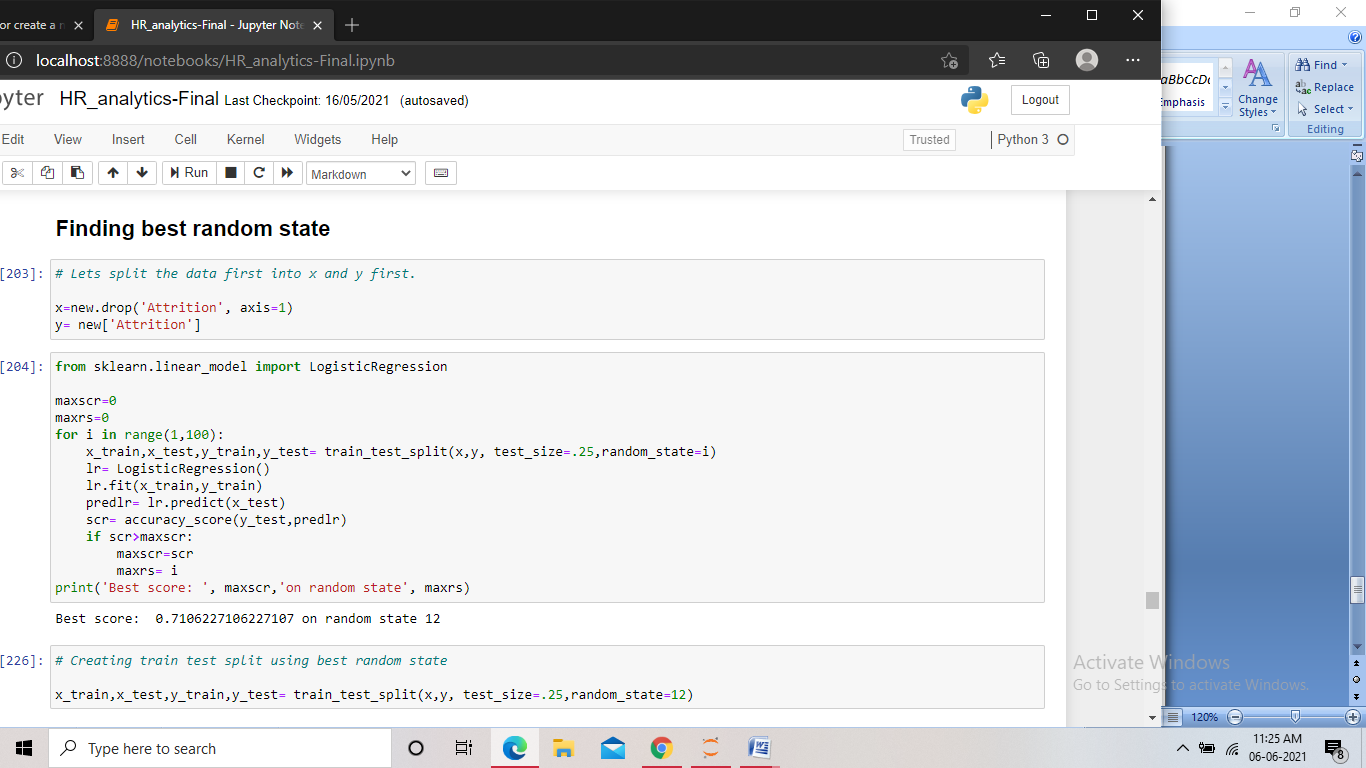
**Building Machine Learning Models:**

* At first we split the data into target and features



Screenshot of the code

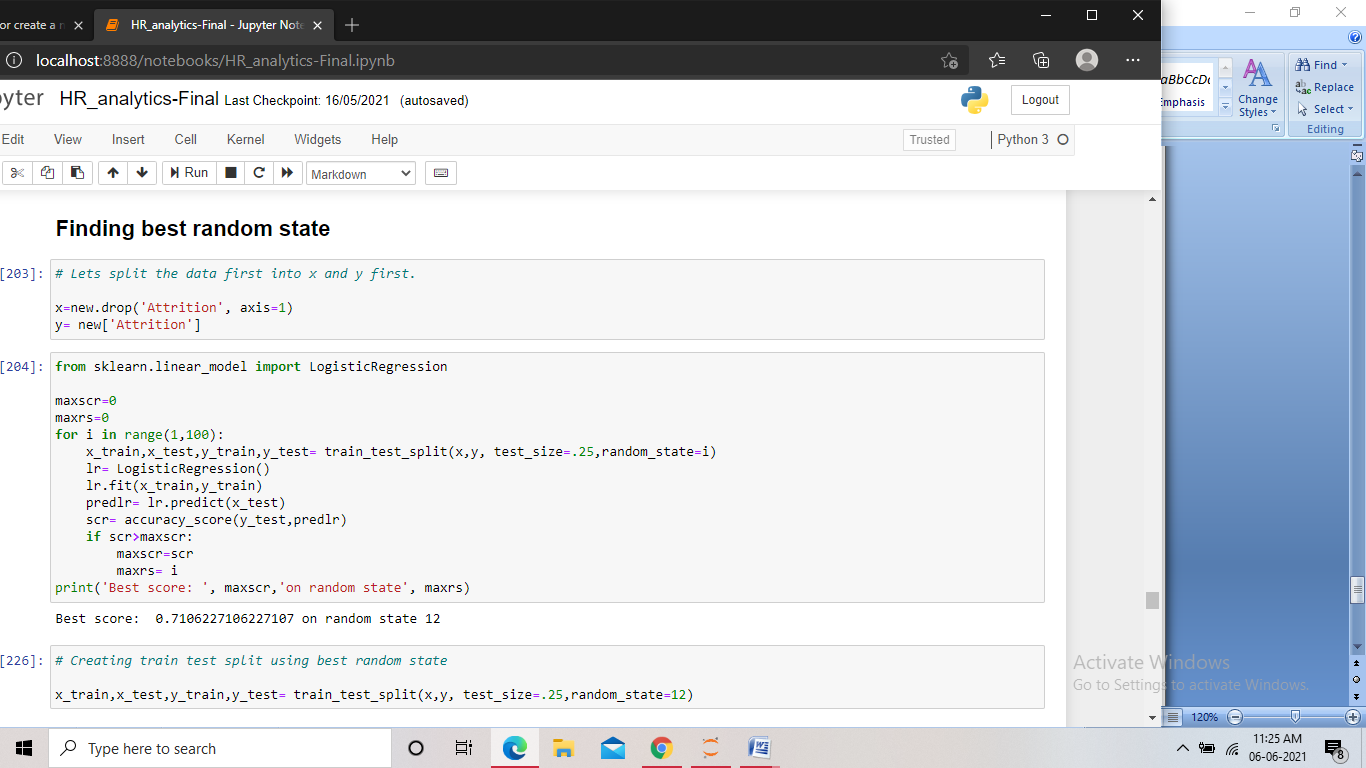
* After that we find the best random state for the dataset that can be used to make the train-test split.



Screenshot of the code

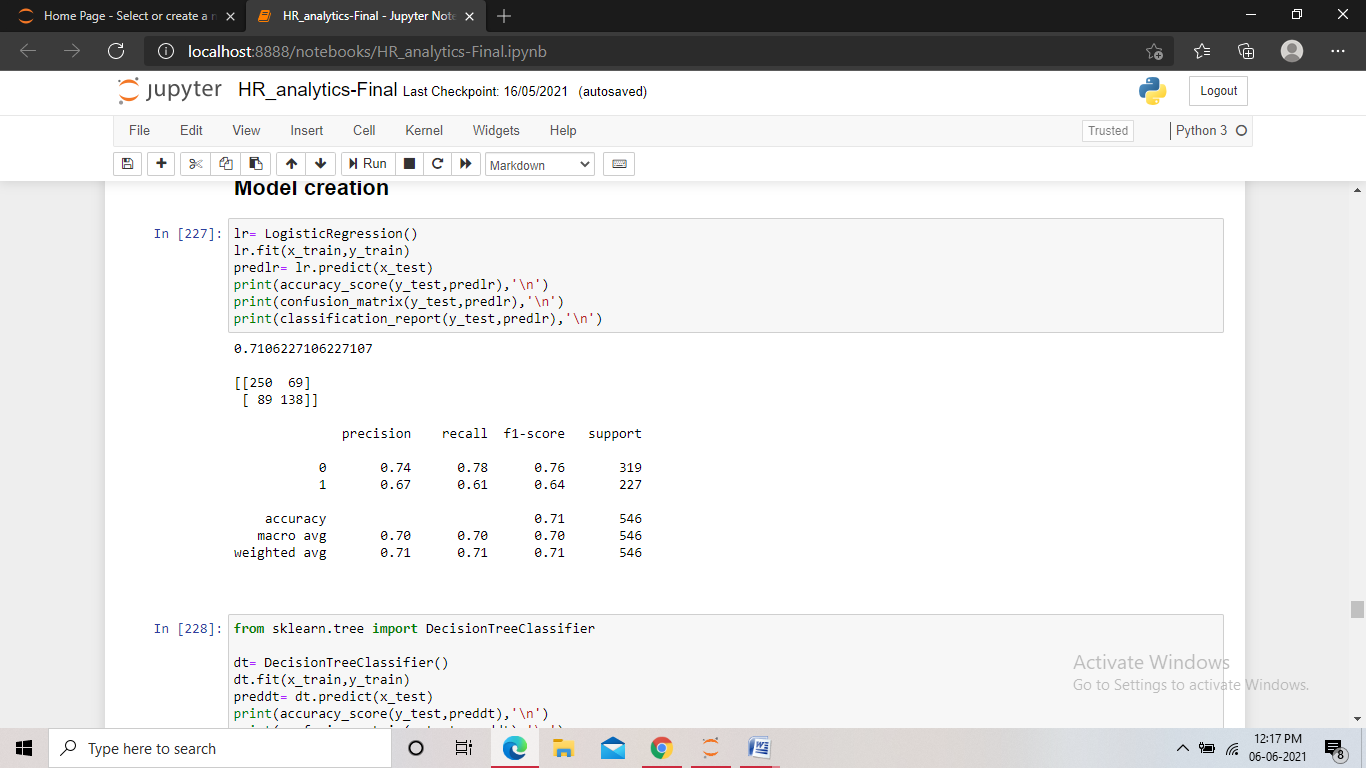
We first created two variables to store the score and the random state and assigned value 0 to them. After that we took a range of 1 to 100 for the random state and created the train test split and then performed model fitting using each random state. We then kept the highest score in one variable and the random state for that score in another. And finally we printed the score and the random state.

Now using the random state that we found giving highest score, we made the train and test split keeping the test size of 25%.



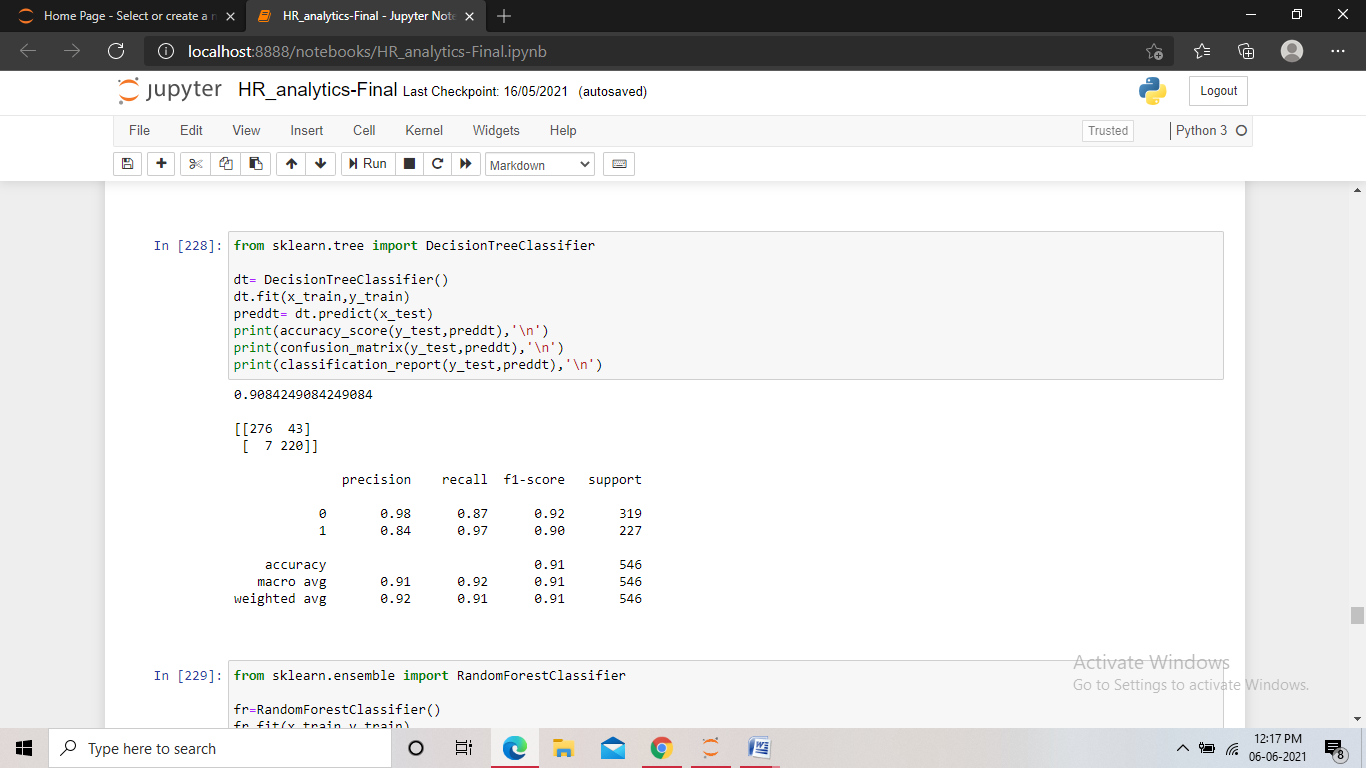
Screenshot of the code

* After the split we import different models from their libraries and assign them to a variable, then using the variable on the train data we perform the model fitting. After that we send the test features to the model for predicting the target and store the prediction in a variable. Finally we check the accuracy of the predictions by comparing the predicted and the actual target and then print the accuracy score. We also print the confusion matrix along with the classification report to get better understanding of the model performance.



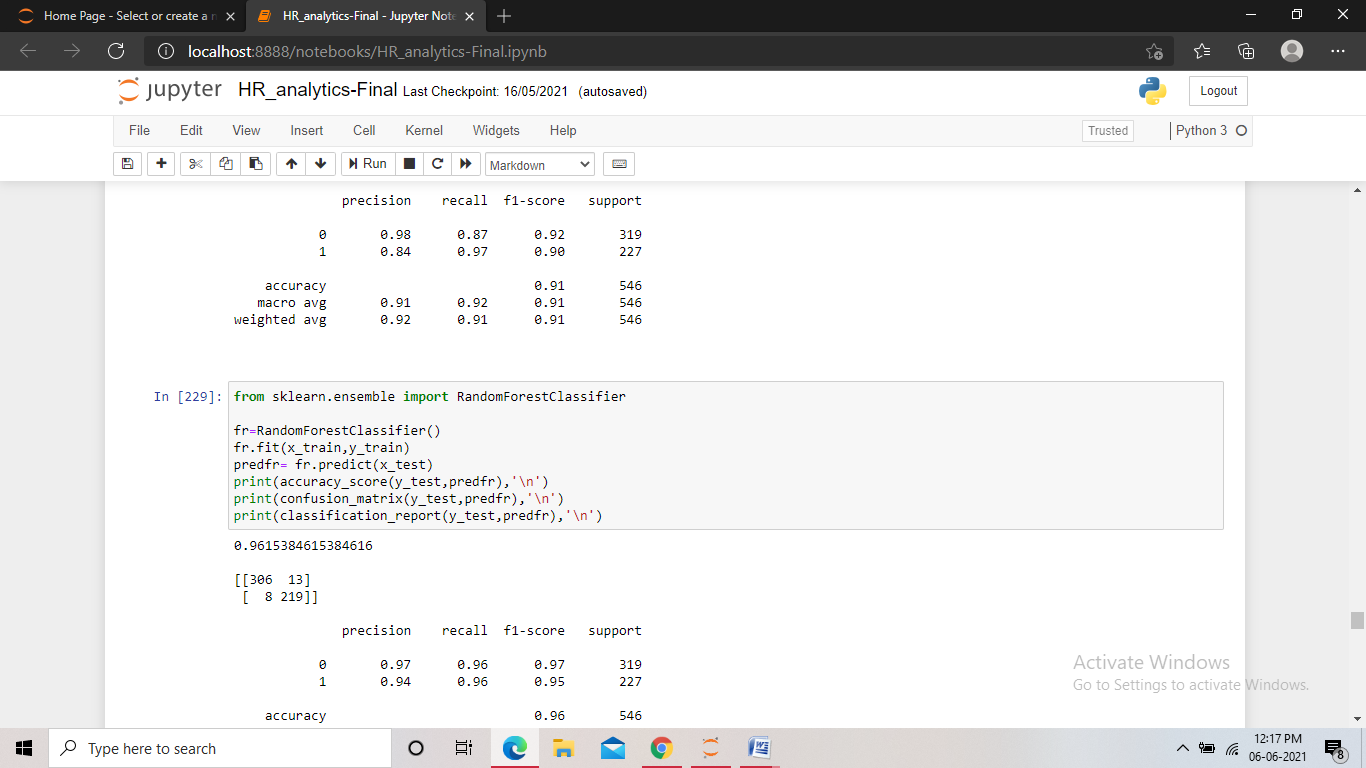
Screenshot of the code for logistic regression model

Using logistic regression model, we found an accuracy score of 71% and the f1 score of 76% for ‘no’ and 64% for ‘yes’.



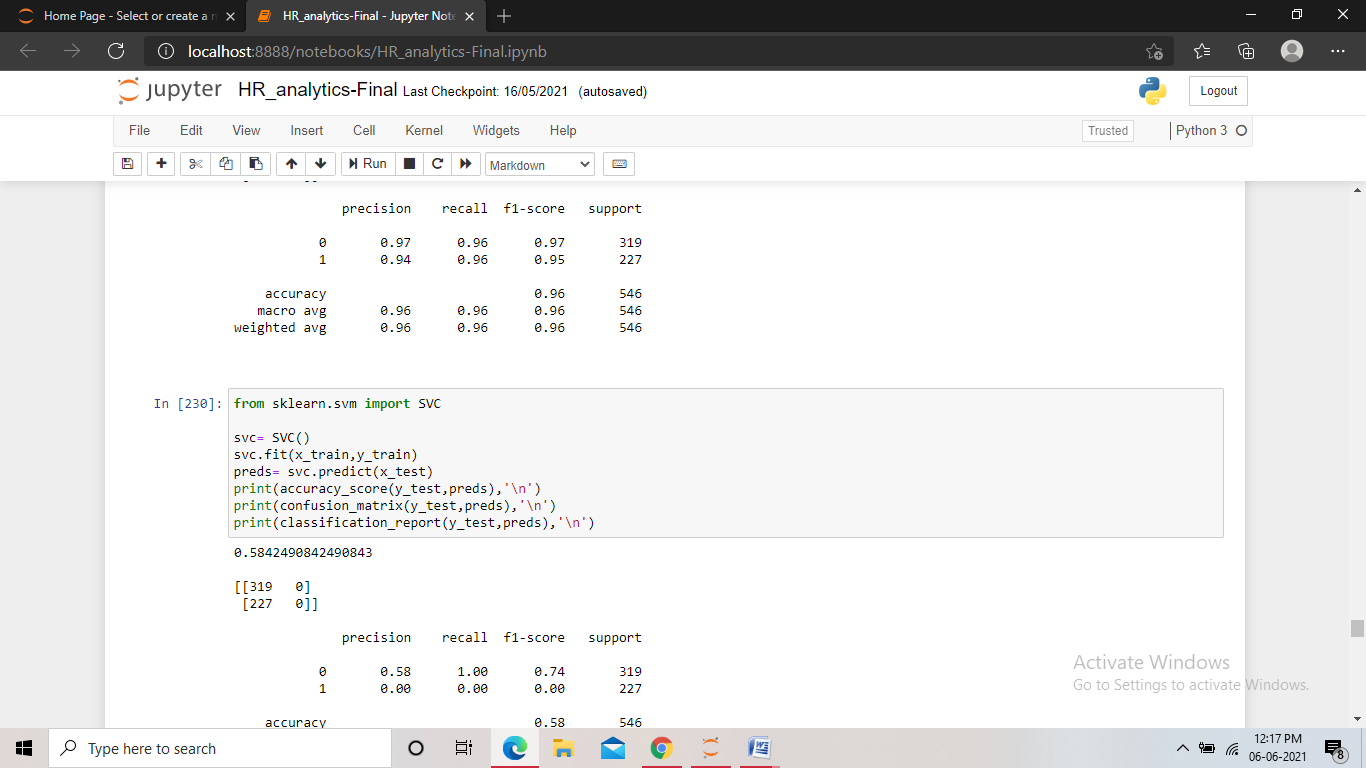
Screenshot of the code for Decision tree classifier

For the Decision tree model, we got an accuracy score of 90% and the f1 score of 92% for ‘no’ and 90% for ‘yes’.



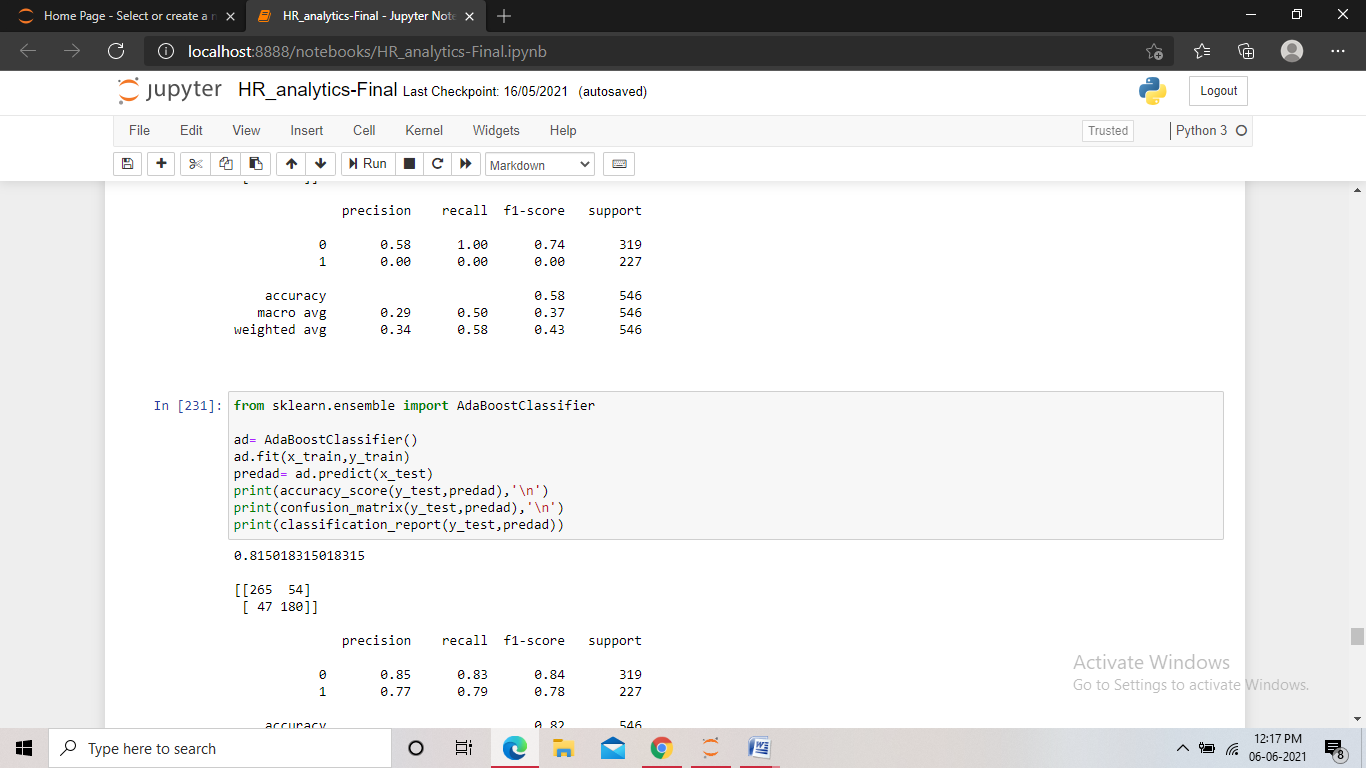
Screenshot of the code for Random Forest classifier

With the Random forest model, we got an accuracy score of 96% and the f1 score of 97% for ‘no’ and 95% for ‘yes’.



Screenshot of the code for SVC

Using Support vector classifier, we found an accuracy score of 58% and the f1 score of 74% for ‘no’ and 0% for ‘yes’.

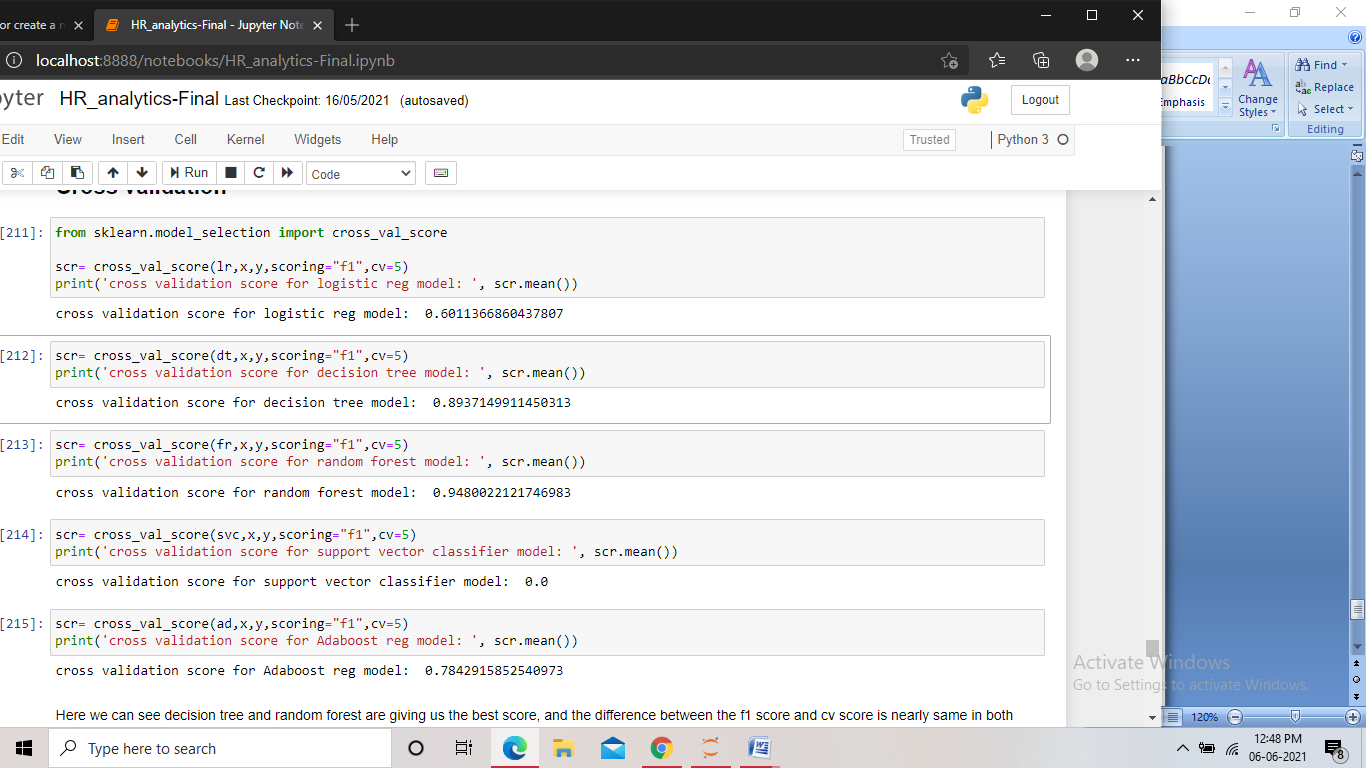


Screenshot of the code for AdaBoost classifier

For the Boosting model AdaBoost classifier, we found an accuracy score of 81% and the f1 score of 84% for ‘no’ and 78% for ‘yes’.

Among all the models we found the Random forest and the Decision tree model giving us the highest f1 scores and accuracy score, with very less errors in the confusion matrix.

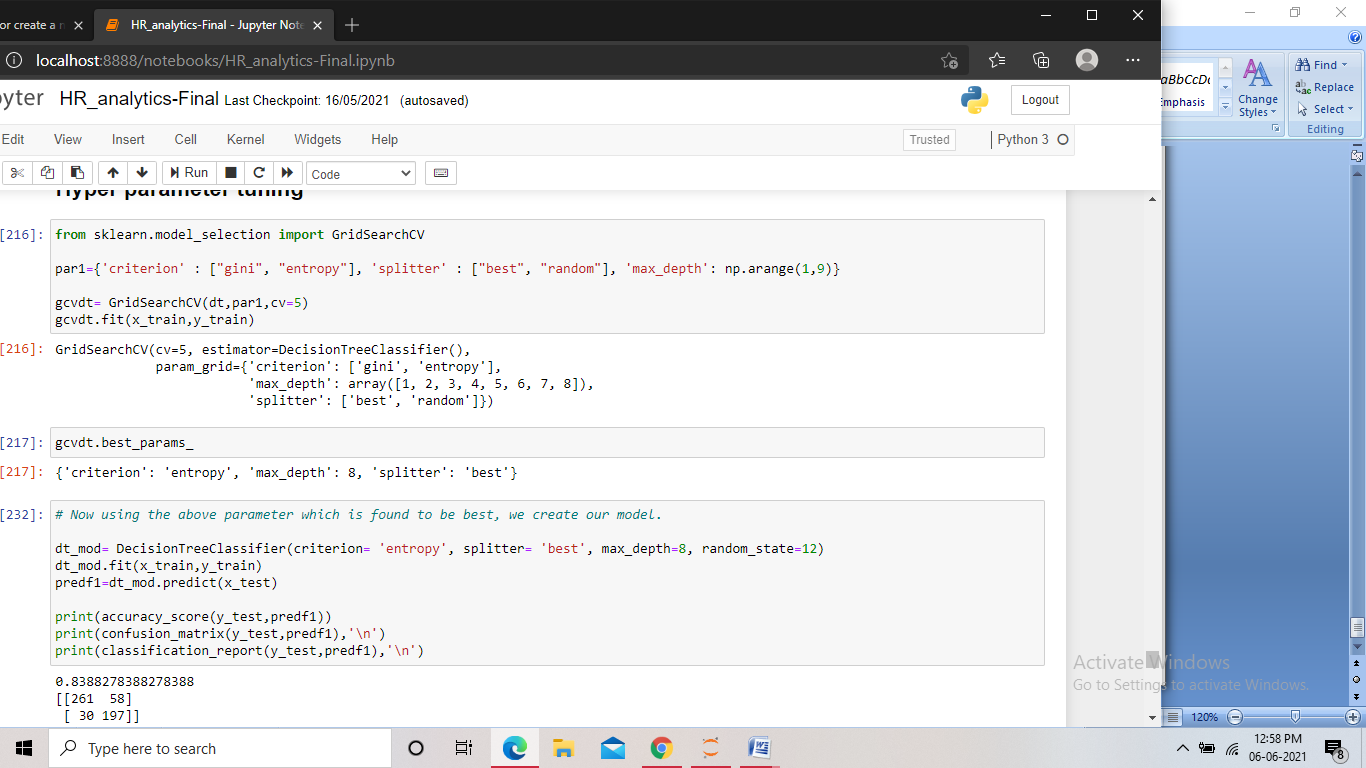
* A good score may be due to the overfitting of the model, and therefore we checked the cross validation score for all the models for any over fitting or under fitting problem.



Screenshot of the code

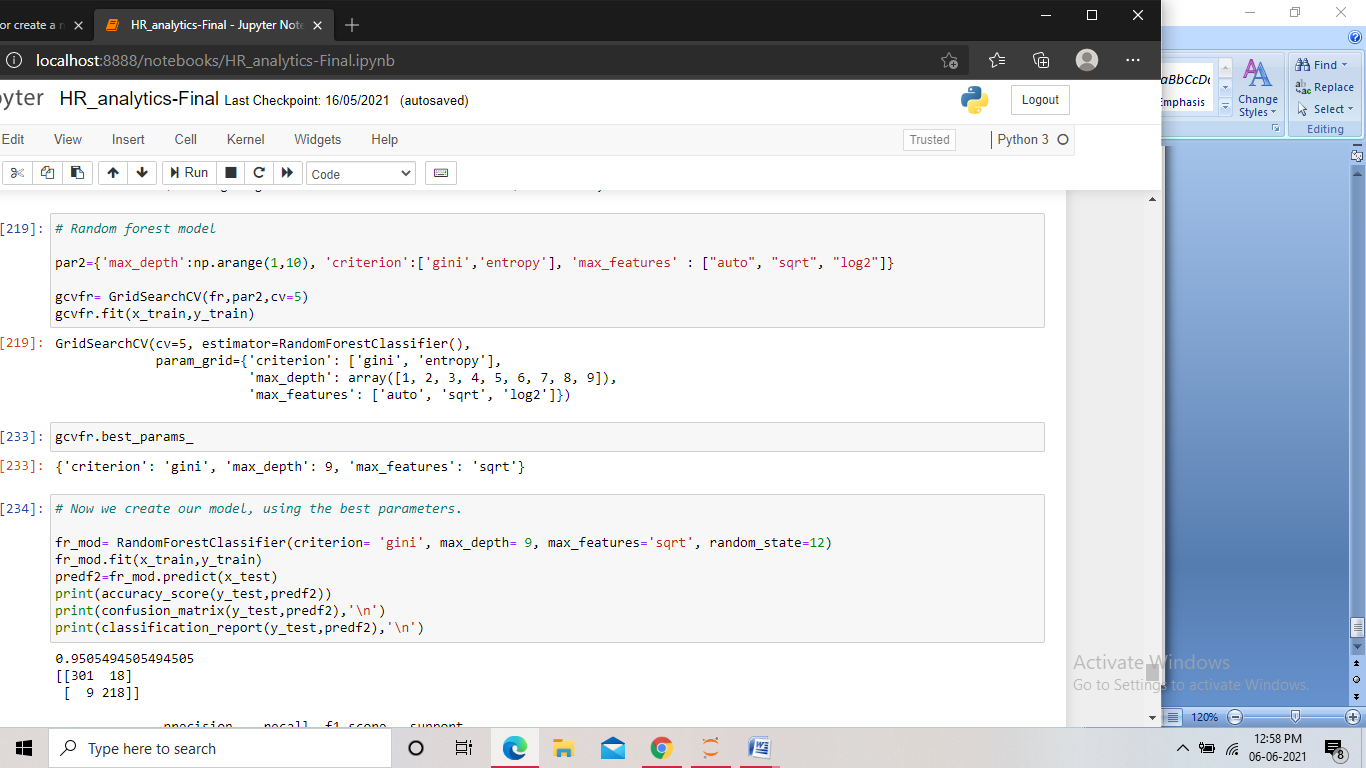
Here also we found the CV score for the random forest and decision tree the highest. Also they showed a very less difference between the f1 scores and the CV score.

* Now, as both the modes showed a good performance, we then hyper parameter tune both the models and find the best performing one among the two.



Screenshot of the code for hyperparameter tuning Decision tree

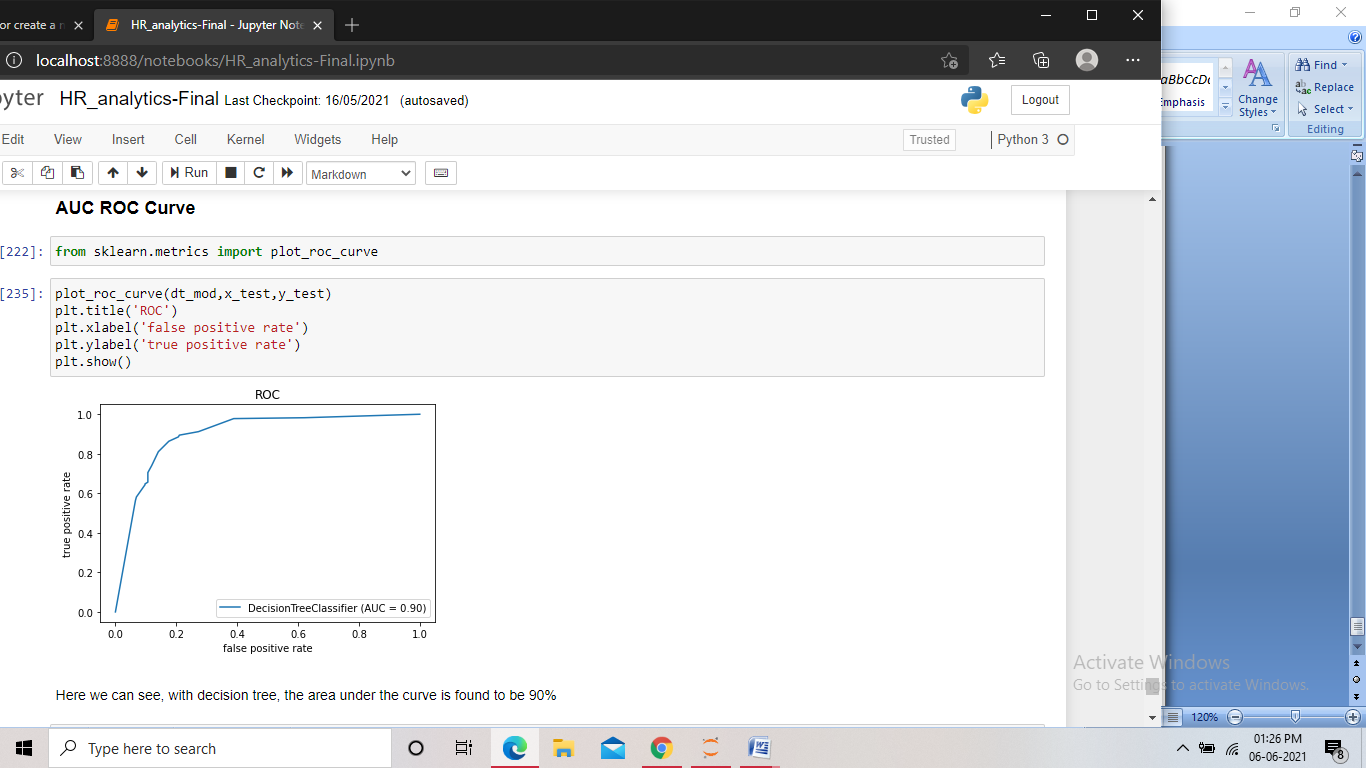
We imported GridSearchCV and passed different attributes with its different parameters to it along with the model, and we got the best parameters for the model. Then using those parameters we created our model.



Screenshot of the code for hyperparameter tuning Random forest

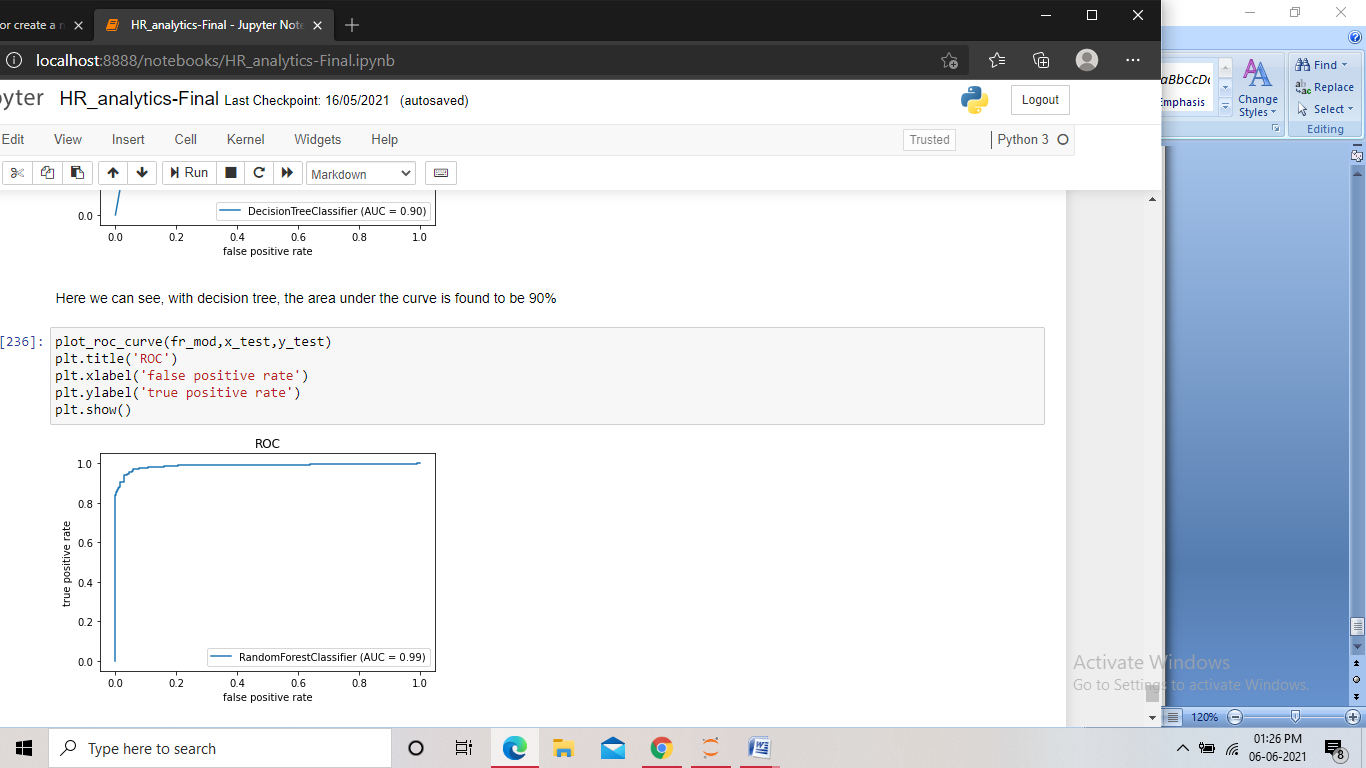
We applied the same steps for random forest and created our model using the parameters we got.

* As this was a classification problem, we then checked the AUC ROC curve.



Screenshot of the code for decision tree

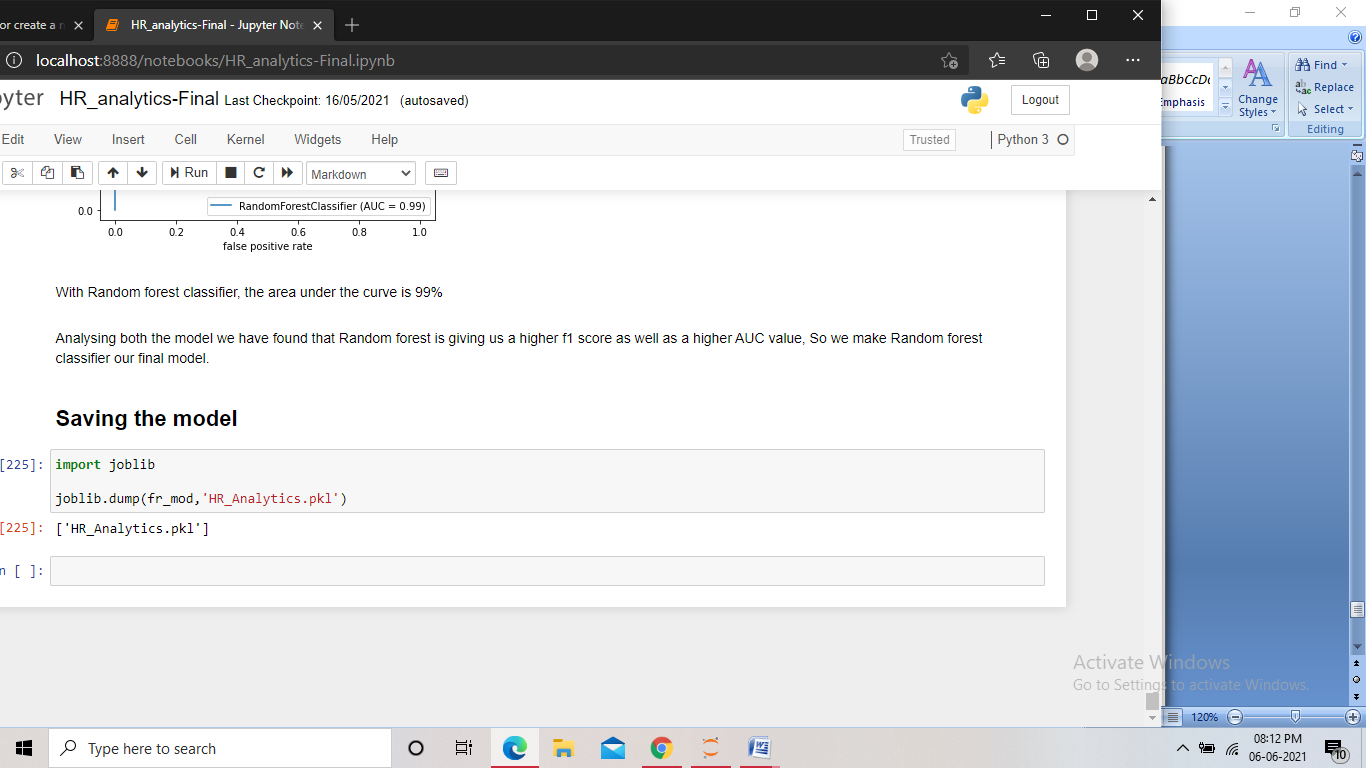
First we imported the plot and fed the model to it, and we got the area under the curve as 90%.



Screenshot of the code for random forest

With the random forest we got the area under the curve as 99%.

* Analyzing both the model’s performance, we found that the Random forest model was performing better than the Decision tree model. Hence we made the former our final model and then saved the model.



Screenshot of the code for saving the model

Finally we imported joblib and saved the model in pkl format.

**Concluding Remarks:**

With this project, we got the idea about what type of data we can work with in building a model and what type data we should avoid. We also found how balancing the target values in a classification problem play a crucial part.

By analyzing the dataset provided in this project, we found that the attrition has a high positive correlation with overtime, as overtime increases chances of attrition also increases. We also found that attrition has a negative correlation with the job level and monthly income, which meant that with a higher job level and a monthly income, the chance of attrition goes down.

We found how tuning the right parameters increase the performance of the model, and saw role of ROC curve in analyzing the performance and finalizing a model in a classification problem.