Open University Learning Analytics Dataset- Adobe Challenge

Utsav Patel, MS in Data Science at Indiana University

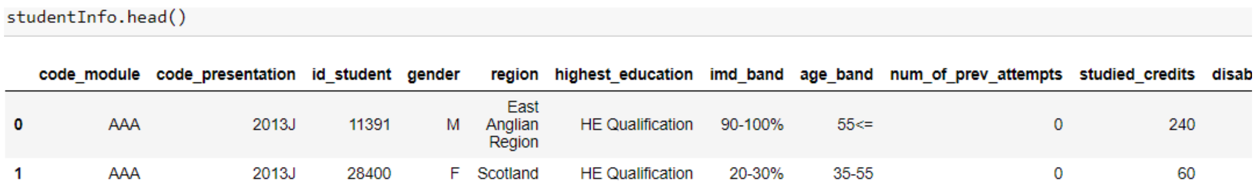
(312)774-3172, utpatel@iu.edu

Data Preprocessing(Combining 7 Tables + Feature Engineering):

- ➔ Main task for Data Preprocessing which occupied nearly more than 50% of time was Combining information from all 7 tables given to us so that no replication occurs and we need to impute missing values also along with doing all Feature Engineering.

Step 1 : Combining studentInfo, Courses and StudentRegistration tables into student table.

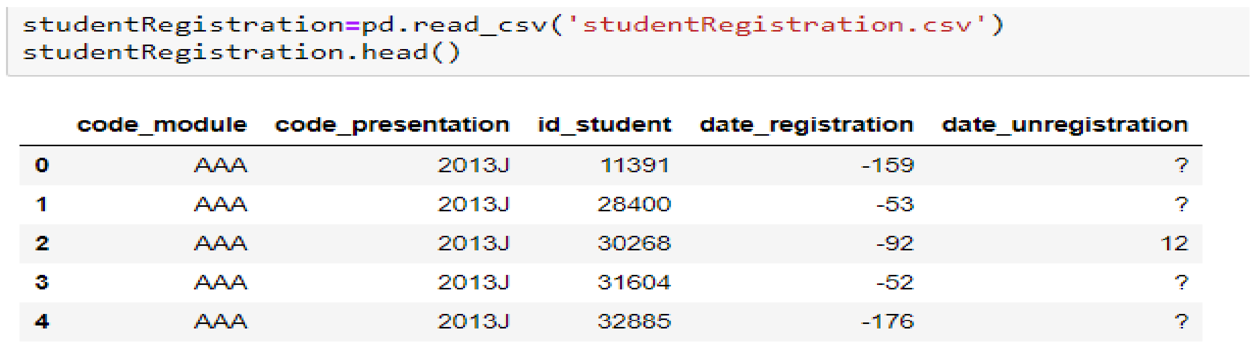
StudentInfo Table



```
studentInfo.head()
```

	code_module	code_presentation	id_student	gender	region	highest_education	imd_band	age_band	num_of_prev_attempts	studied_credits	disab
0	AAA	2013J	11391	M	East Anglian Region	HE Qualification	90-100%	55<=	0	240	
1	AAA	2013J	28400	F	Scotland	HE Qualification	20-30%	35-55	0	60	

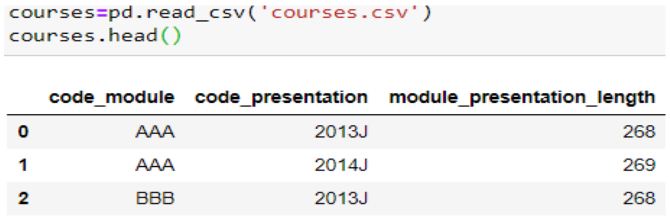
studentRegistration Table



```
studentRegistration=pd.read_csv('studentRegistration.csv')
studentRegistration.head()
```

	code_module	code_presentation	id_student	date_registration	date_unregistration
0	AAA	2013J	11391	-159	?
1	AAA	2013J	28400	-53	?
2	AAA	2013J	30268	-92	12
3	AAA	2013J	31604	-52	?
4	AAA	2013J	32885	-176	?

Courses table:



```
courses=pd.read_csv('courses.csv')
courses.head()
```

	code_module	code_presentation	module_presentation_length
0	AAA	2013J	268
1	AAA	2014J	269
2	BBB	2013J	268

```
student = pd.merge(studentInfo, courses,  how='left', on=['code_module','code_presentation'])
```

```
student = pd.merge(student, studentRegistration,  how='left', on=['code_module','code_presentation','id_student'])
student.head()
```

- First of all, we performed Left Join on tables: studentInfo and courses on code_module and code_presentation into one table called student.
- Then we merged student and studentRegistration on code_module, code_presentation and id_student, which led us to successful completion of combining 3 tables into student table.

Step 2: Combination of studentVle table and vle table :

```
studentVle=pd.read_csv('studentVle.csv')
studentVle.head()
```

	code_module	code_presentation	id_student	id_site	date	sum_click
0	AAA	2013J	28400	546652	-10	4
1	AAA	2013J	28400	546652	-10	1
2	AAA	2013J	28400	546652	-10	1
3	AAA	2013J	28400	546614	-10	11
4	AAA	2013J	28400	546714	-10	1

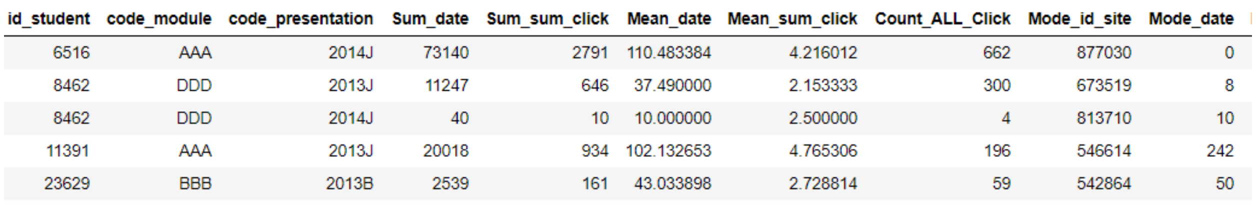
- We are having many duplicate rows in this table and I felt that it might be a deliberate different entry in studentVle for each corresponding feature.

```
vle=pd.read_csv('vle.csv')
vle.head()
```

	id_site	code_module	code_presentation	activity_type	week_from	week_to
0	546943	AAA	2013J	resource	?	?
1	546712	AAA	2013J	oucontent	?	?
2	546998	AAA	2013J	resource	?	?
3	546888	AAA	2013J	url	?	?
4	547035	AAA	2013J	resource	?	?

ONE TIME EXPLANATION OF CONCEPT BEHIND JOINING OF ANY TWO TABLES :

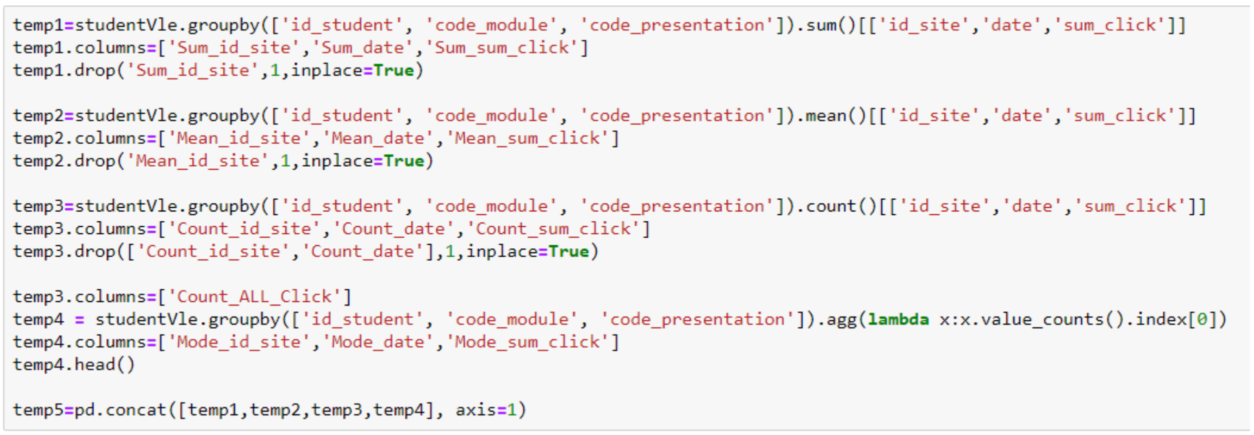
- We can see that in order to combine vle and studentVle, we have id_site, code_module and code_presentation in common and that is in fact a good news!! So how to combine both tables on these 3 common columns?
- The problem is in studentVle table, id_student, id_site, code_module and code_presentation has common values. It means for given 4 features of id_student, id_site, code_module and code_presentation, we will have many corresponding different entries for rest of the columns in studentVle table.
- But there is 100% unique feature called id_site in vle and we have id_student, code_module and code_presentation columns common in most of the tables. So How will it be if in Studentvle table we groupby id_student, code_module and code_presentation columns and perform operations on id_site such that we have only one entry for id_site instead of multiple. If we have only one entry of id_site, then we can easily combine vle and studentVle based on id_site, code_module and code_presentation.
- Thus, we need to groupby 3 columns in studentVle : id_student, code_module and code_presentation
- But What happens after doing groupby? How to get information from id_site, date and sum_click feature as they are multiple entries for same set of groupedby features.



id_student	code_module	code_presentation	Sum_date	Sum_sum_click	Mean_date	Mean_sum_click	Count_ALL_Click	Mode_id_site	Mode_date
6516	AAA	2014J	73140	2791	110.483384	4.216012	662	877030	0
8462	DDD	2013J	11247	646	37.490000	2.153333	300	673519	8
8462	DDD	2014J	40	10	10.000000	2.500000	4	813710	10
11391	AAA	2013J	20018	934	102.132653	4.765306	196	546614	242
23629	BBB	2013B	2539	161	43.033898	2.728814	59	542864	50

- In order to deal with Numerical features of id_site, sum_click and date, we will groupby with id_student, code_module and code_presentation and then perform Mean, Mode, Sum, Max, Min type of operations on id_site, date(no_of_days) and click.
- As you can see we have following type of code for each and every combination of Sum, Mean, Count, Mode. It is very important to note that we extract a lot of information when we perform these many operations and make them as new column and use it.

Now, this all discussed above (doing Sum, Mean, Count, Mode, MAX and MIN on features after doing groupby and using them as new features) is considered as Feature Engineering.

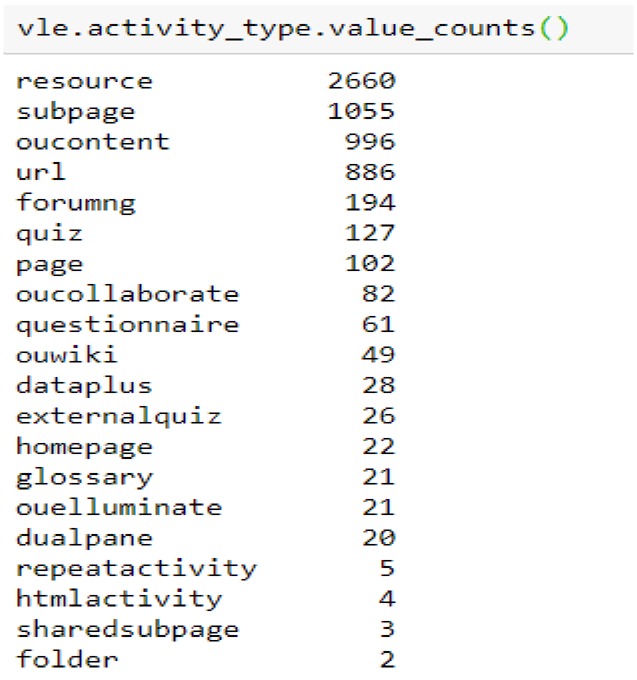


```
temp1=studentVle.groupby(['id_student', 'code_module', 'code_presentation']).sum()[['id_site','date','sum_click']]
temp1.columns=['Sum_id_site','Sum_date','Sum_sum_click']
temp1.drop('Sum_id_site',1,inplace=True)

temp2=studentVle.groupby(['id_student', 'code_module', 'code_presentation']).mean()[['id_site','date','sum_click']]
temp2.columns=['Mean_id_site','Mean_date','Mean_sum_click']
temp2.drop('Mean_id_site',1,inplace=True)

temp3=studentVle.groupby(['id_student', 'code_module', 'code_presentation']).count()[['id_site','date','sum_click']]
temp3.columns=['Count_id_site','Count_date','Count_sum_click']
temp3.drop(['Count_id_site','Count_date'],1,inplace=True)

temp3.columns=['Count_ALL_Click']
temp4 = studentVle.groupby(['id_student', 'code_module', 'code_presentation']).agg(lambda x:x.value_counts().index[0])
temp4.columns=['Mode_id_site','Mode_date','Mode_sum_click']
temp4.head()

temp5=pd.concat([temp1,temp2,temp3,temp4], axis=1)
```

studentVle_vle is formed by smartly performing a series of operations like Max, Mean, Mode, Count and Sum on lots of features.

Feature Engineering included:

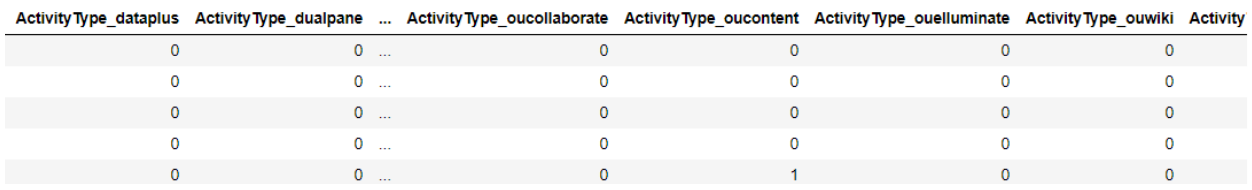
1. Making Some features One Hot Encoded. Take for example: ActivityType feature has following values:



```
vle.activity_type.value_counts()
```

```
resource          2660
subpage           1055
oucontent          996
url                886
forumng            194
quiz               127
page               102
oucollaborate       82
questionnaire       61
ouwiki              49
dataplus            28
externalquiz        26
homepage            22
glossary            21
ouelluminate        21
dualpane            20
repeatactivity       5
htmlactivity         4
sharedsubpage        3
folder               2
```

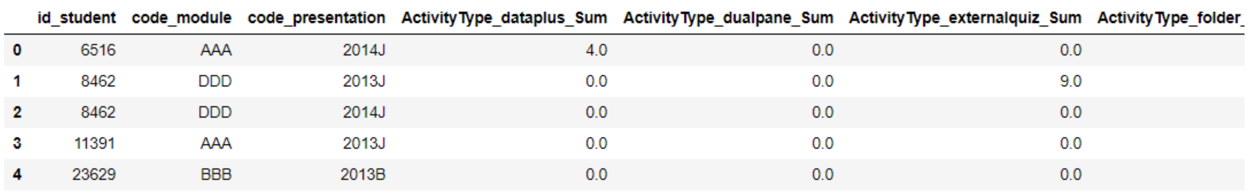
Now, these categorical values could not be used, so we used One Hot Encoding of this feature and that lead us to 20 new features from one feature. This is called One Hot Encoding where we make new feature and give that feature a value of 1 if that features name was corresponding category in Activity_type column.



ActivityType_dataplus	ActivityType_dualpane	...	ActivityType_oucollaborate	ActivityType_oucontent	ActivityType_ouelluminate	ActivityType_ouwiki	Activity
0	0	...	0	0	0	0	
0	0	...	0	0	0	0	
0	0	...	0	0	0	0	
0	0	...	0	0	0	0	
0	0	...	0	1	0	0	

2. Taking Mean, Max, Count , Sum and Mode of Numerical features and using it (already explained above).

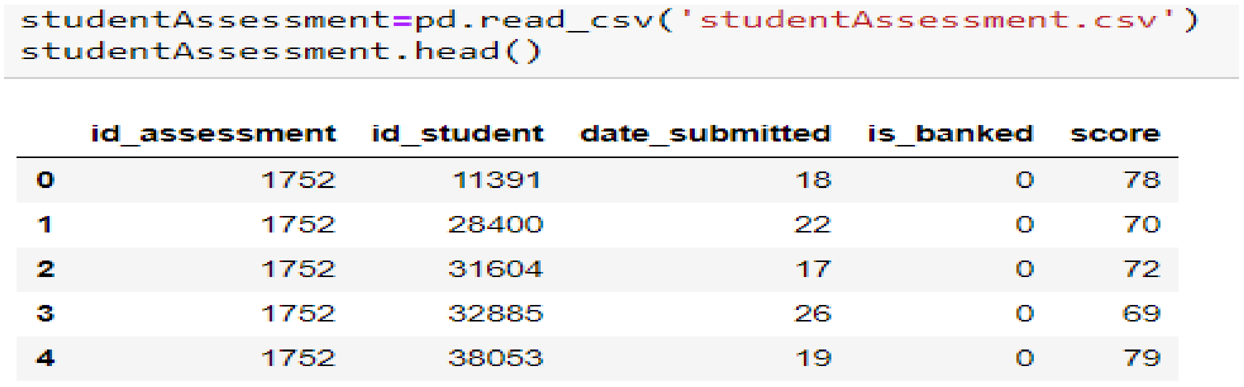
At last StudentVle_vle was made with combination of vle and studentVle performing lots of operations.



	id_student	code_module	code_presentation	ActivityType_dataplus_Sum	ActivityType_dualpane_Sum	ActivityType_externalquiz_Sum	ActivityType_folder_
0	6516	AAA	2014J	4.0	0.0	0.0	
1	8462	DDD	2013J	0.0	0.0	9.0	
2	8462	DDD	2014J	0.0	0.0	0.0	
3	11391	AAA	2013J	0.0	0.0	0.0	
4	23629	BBB	2013B	0.0	0.0	0.0	

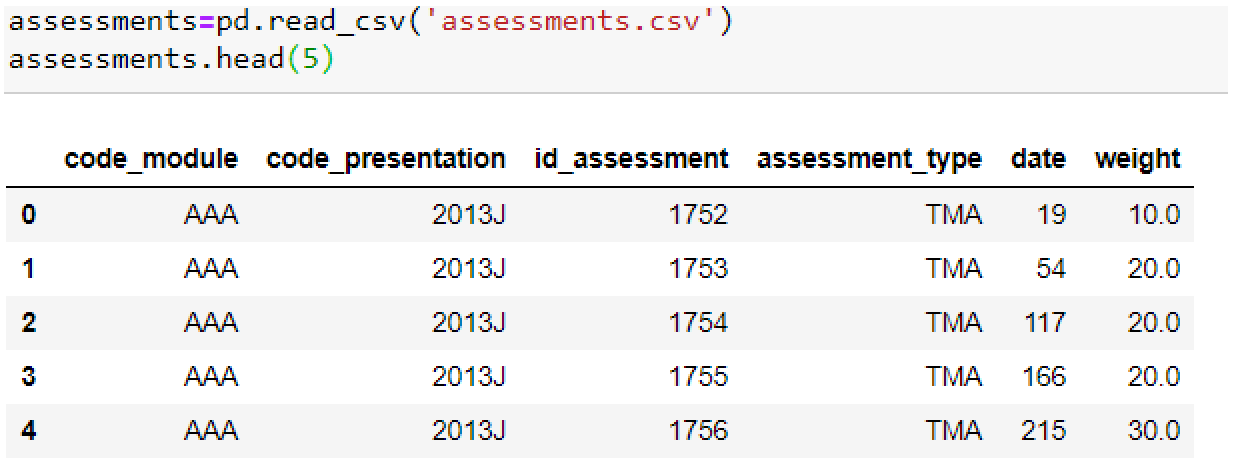
3. At later stage, Feature Engineering included Imputation of Missing Values and Dropping off Least important features.

Step 3: Merging studentAssessment and assessments table :



```
studentAssessment=pd.read_csv('studentAssessment.csv')
studentAssessment.head()
```

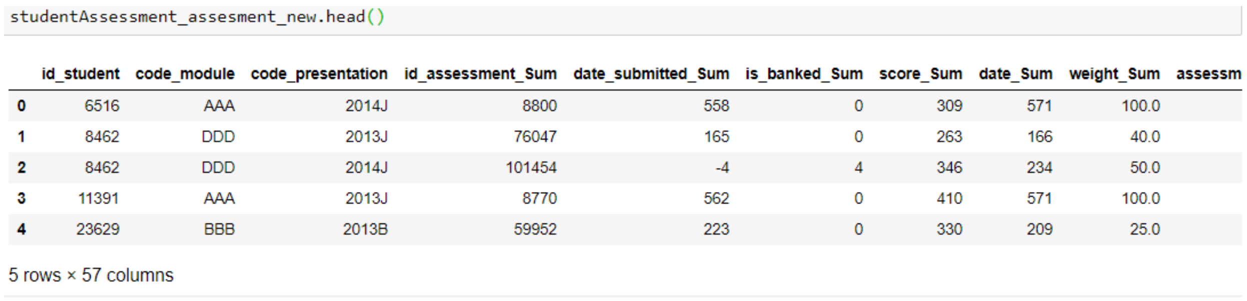
	id_assessment	id_student	date_submitted	is_banked	score
0	1752	11391	18	0	78
1	1752	28400	22	0	70
2	1752	31604	17	0	72
3	1752	32885	26	0	69
4	1752	38053	19	0	79



```
assessments=pd.read_csv('assessments.csv')
assessments.head(5)
```

	code_module	code_presentation	id_assessment	assessment_type	date	weight
0	AAA	2013J	1752	TMA	19	10.0
1	AAA	2013J	1753	TMA	54	20.0
2	AAA	2013J	1754	TMA	117	20.0
3	AAA	2013J	1755	TMA	166	20.0
4	AAA	2013J	1756	TMA	215	30.0

- We combined both tables on **id_assessment** key.
- In combined table, we replaced '?' in **date** feature with average of all other dates that were already present.
- In combined table, we replaced '?' in **score** feature with average of all other dates that were already present.
- One Hot encoding of **assessment_type** in combined table
- We grouped by 3 features : ['id_student','code_module','code_presentation'] and performed **Mean, Max, Mode, Count and Sum Operations** on the rest features after making every feature numerical.
- Combined table was named **studentAssessment_assesment_new**



```
studentAssessment_assesment_new.head()
```

	id_student	code_module	code_presentation	id_assessment_Sum	date_submitted_Sum	is_banked_Sum	score_Sum	date_Sum	weight_Sum	assessm
0	6516	AAA	2014J	8800	558	0	309	571	100.0	
1	8462	DDD	2013J	76047	165	0	263	166	40.0	
2	8462	DDD	2014J	101454	-4	4	346	234	50.0	
3	11391	AAA	2013J	8770	562	0	410	571	100.0	
4	23629	BBB	2013B	59952	223	0	330	209	25.0	

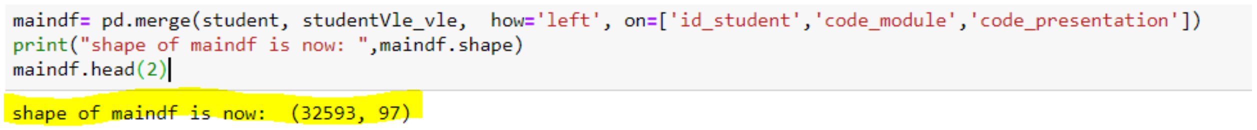
5 rows × 57 columns

Step 4: Combination of student and studentVle_vle and studentAssessment_assesment_new Tables:

Recap: student(studentInfo, courses and studentRegistration)

studentVle_vle(vle + studentVle)

Combined student and studentVle_vle on ['id_student','code_module','code_presentation'] – Did Left Join



```
maindf= pd.merge(student, studentVle_vle,  how='left', on=['id_student','code_module','code_presentation'])
print("shape of maindf is now: ",maindf.shape)
maindf.head(2)
```

```
shape of maindf is now:  (32593, 97)
```

Shape is now 32593 X 97

- Combined maindf and studentAssessment_assesment_new, how=left, on=['id_student', 'code_module', 'code_presentation']
- Above step was done first and then we found a trend in data and then we reverted back to do same join of tables but with Inner Join. This all is explained in below section.

TWIST IN DATA:

A Twist in Data Insight that could have been used to Cheat the System of Prediction, but I did not cheat. Find out Why?

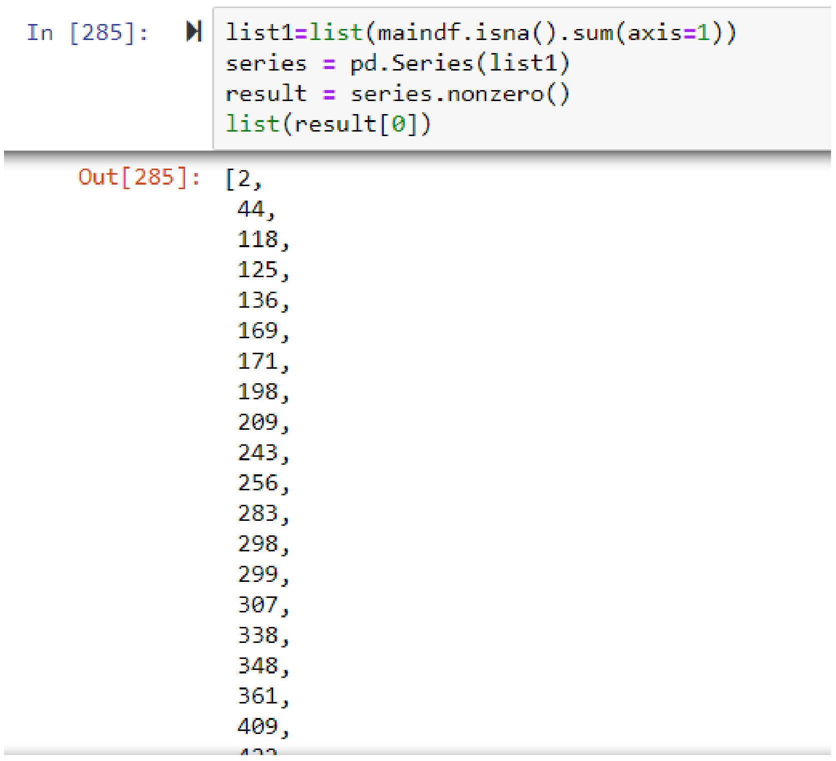
DO NOT HAVE TIME TO READ 2 pages? : This might take a little longer insight to study and understand, but if reader of this report is having short of time here is what is in this whole section of TWIST. I dropped all rows that were having Fail Status and had missing entries in assignments done later after they dropped or failed the course. I dropped them because those missing values were almost 100% of failing people and If I impute it with any value, my Machine Learning Model can capture it so easily and I don't want to capture people whom I already know they have failed . **GO TO TWIST ENDED SECTION.**

WANT TO KNOW IN DETAIL WHAT HAPPENED?

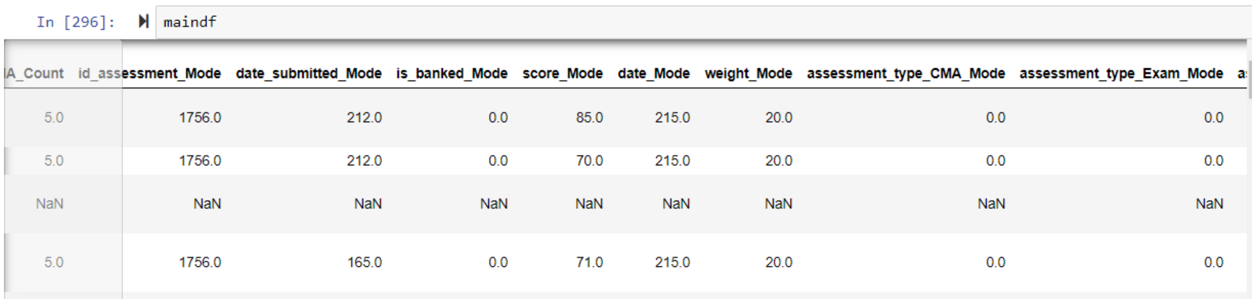
Here is what has happened:

➔ After combining all 3 final dataframes into one gave us unexpected results. There were many rows which had some or the other thing missing. In all, most of the features were missing for a particular code_module, code_presentation and id_student.

➔ Below is a look of index of rows which were having some or other feature missing and guess what ? There were nearly 6800 of them !!



```
In [285]:   list1=list(maindf.isna().sum(axis=1))
            series = pd.Series(list1)
            result = series.nonzero()
            list(result[0])

Out[285]: [2,
            44,
            118,
            125,
            136,
            169,
            171,
            198,
            209,
            243,
            256,
            283,
            298,
            299,
            307,
            338,
            348,
            361,
            409,
```



```
In [296]:   maindf
```

A_Count	id_assessment_Mode	date_submitted_Mode	is_banked_Mode	score_Mode	date_Mode	weight_Mode	assessment_type_CMA_Mode	assessment_type_Exam_Mode	a
5.0	1756.0	212.0	0.0	85.0	215.0	20.0	0.0	0.0	
5.0	1756.0	212.0	0.0	70.0	215.0	20.0	0.0	0.0	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5.0	1756.0	165.0	0.0	71.0	215.0	20.0	0.0	0.0	

➔ Missing lot of features on same row. Likewise there were 6800 rows.

➔ With these 6800 rows missing data there was a very important characteristic found. See Below:

➔ **Nearly 6795 from 6800 were Fail or Withdrawn and only 5 were Pass.** This was an amazing characteristic that came to our notation while exploring data.

```
maindf.iloc[list(result[0]),:]['final_result'].value_counts()
```

```
Withdrawn    5483
Fail         1312
Pass            5
Name: final_result, dtype: int64
```

- ➔ Very Important Intuition Behind this effect of getting NA values after joining tables and Why I did not consider it to be used as a feature:
- ➔ It is very important to note that there were more Withdrawn than Fail and that too with almost 5 times more. This led me to further exploration of the data and I got to know one more trend, which is not a good news!!
- ➔ So a further exploration led me to result that :"If a student has withdrawn from mid semester or in between of running course, then what happens?"
- ➔ YES, you are right!! We will have missing values of marks for later assignments after student has withdrawn from class. Take a example: If I am student at Open University and If I am taking 1 course. I completed half of the course and decided to drop that course. I completed 3 assessments from 10 assessments. So What happens now? I will have no entry after 3rd assessment in student_assessment table. Thus, when we are combining all tables, we are in sort HACKING if we are making imputation or making new feature that indicates those missing values and we will have nearly 98-99% Accuracy, which would be of no use as this type of model would not be used in real life as we will never know what assessments entries are going to missed for any student because if we know that it means the student has already failed.

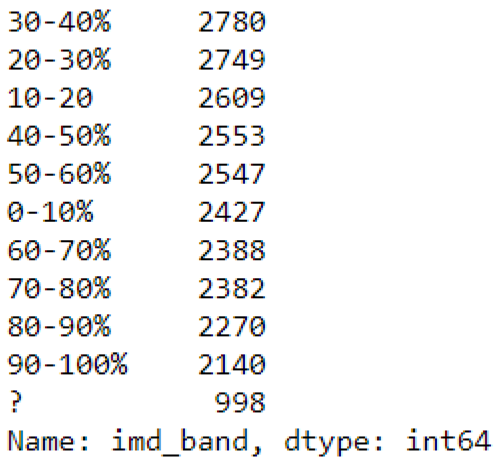
You should notice one thing that there were more withdrawn as compared to failed because students withdraw from a course during starting or mid of semester mostly and generally have more missing entries for assessments as compared to failing students, who might have been failed nearly at end of semester after trying a lot, which leads to less number of missing assessments for such failed students.!!!

- What happens if we do Imputation- filling with any other value or making new feature from this missing data?
- Our Model will capture this and will lead us to fake high accuracies- which we do not want as this type of model has no real life usefulness.
- Thus, I decided to drop all the rows that were having missing values because simply there is no meaning of hacking into the data and increasing accuracy to 99%.

TWIST IN DATA SECTION ENDED

DATA PREPROCESSING CONTINUED:

- Solution to our problem was to drop those rows by doing Inner Join.
- Now, all colmns derived from ActivityType column (All One Hot Encoded * (Mean, Mode,Max,Sum,Count)) was missing and I imputed them to 0.
- Maindf is having imd_band which was Integer Encoded. It means we gave 0-10% as 1 and 70-80% as 8 and so on. (0 to 10 coded)



```
30-40%     2780
20-30%     2749
10-20      2609
40-50%     2553
50-60%     2547
0-10%      2427
60-70%     2388
70-80%     2382
80-90%     2270
90-100%    2140
?           998
Name: imd_band, dtype: int64
```

- After this, Features which were having constant values were dropped.
- Features with having 100% correlation with another feature included in dataframe were dropped.
- Opendf was **achieved** after doing all sorts of normal cleaning of data further by combining effectively almost all 7 tables.

Model Building Phase:

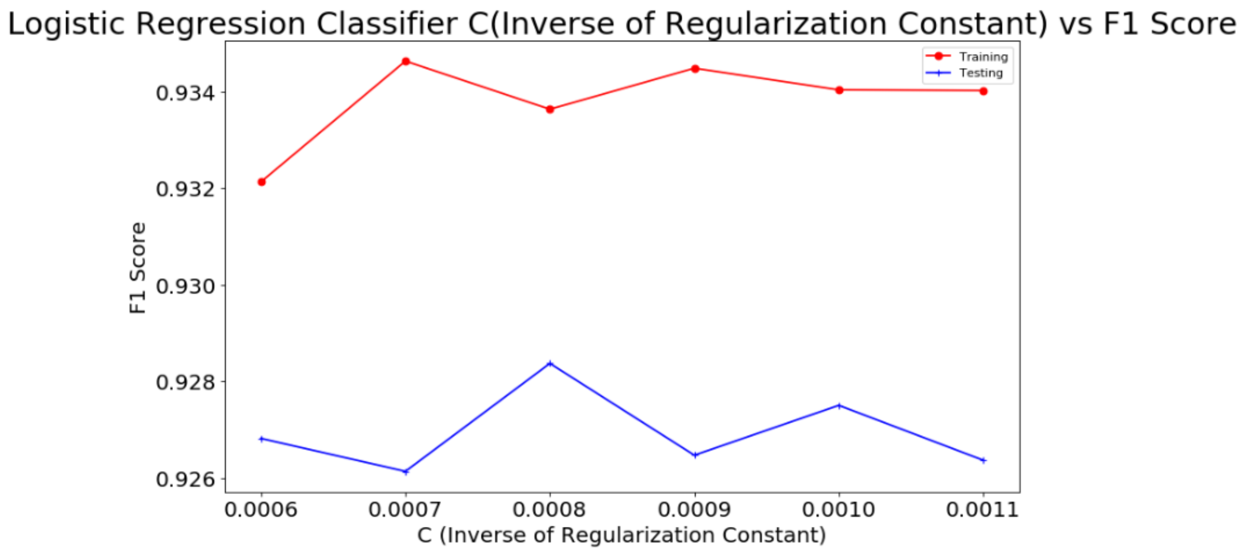
Accuracy Metric:

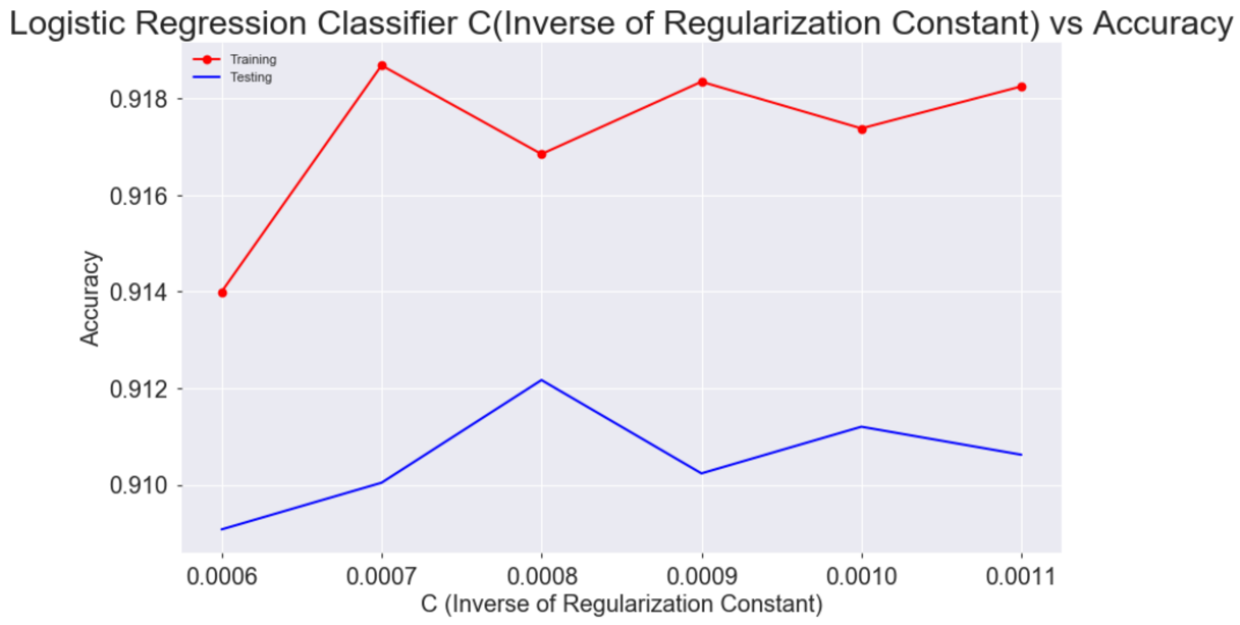
- Our Accuracy Metric would be F1 Score and We can also have a look at Accuracy for corresponding model as it is not highly imbalanced data, so for this classification model, F1 score is "good" to use and accuracy is also "good" to use.
- F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Thus is it always best to have a look at our False Positives and False Negatives.

✓ **Train Test Split of Data: 80% Training and 20% Testing**

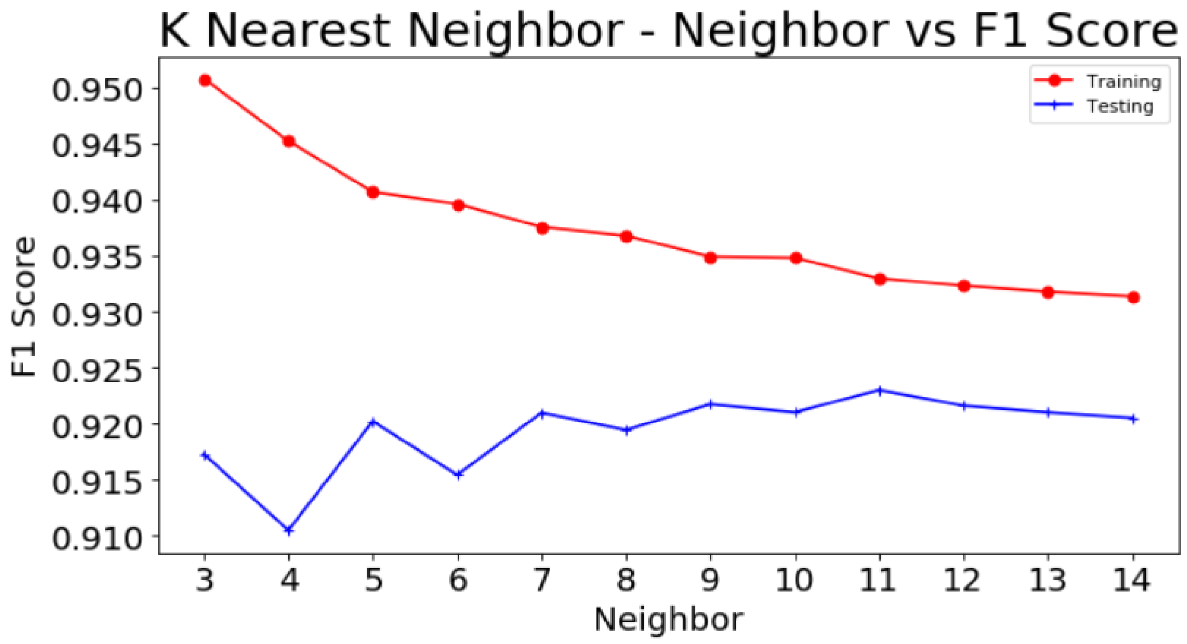
✓ **For Each and Every Model from Now onwards, General trend was 80% Training Data (Final model was build after doing 5 Fold Cross Validation) and 20% Testing.**

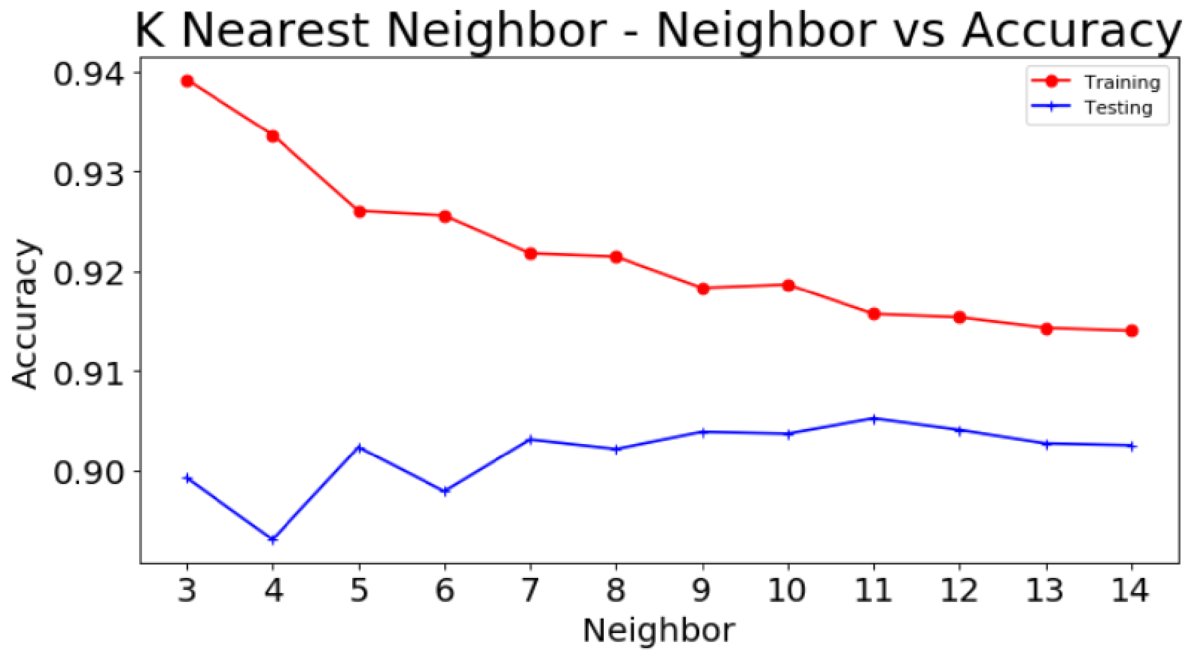
Logistic Regression : F1 Score of 0.9283 and Testing Accuracy : 91.2%



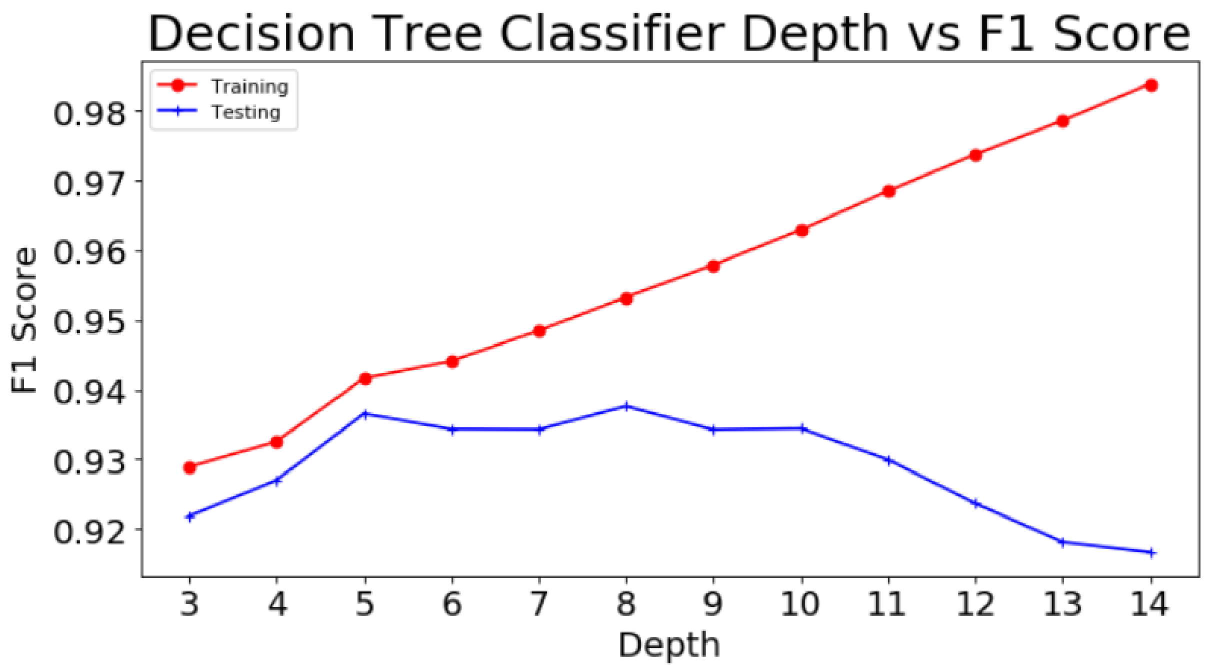


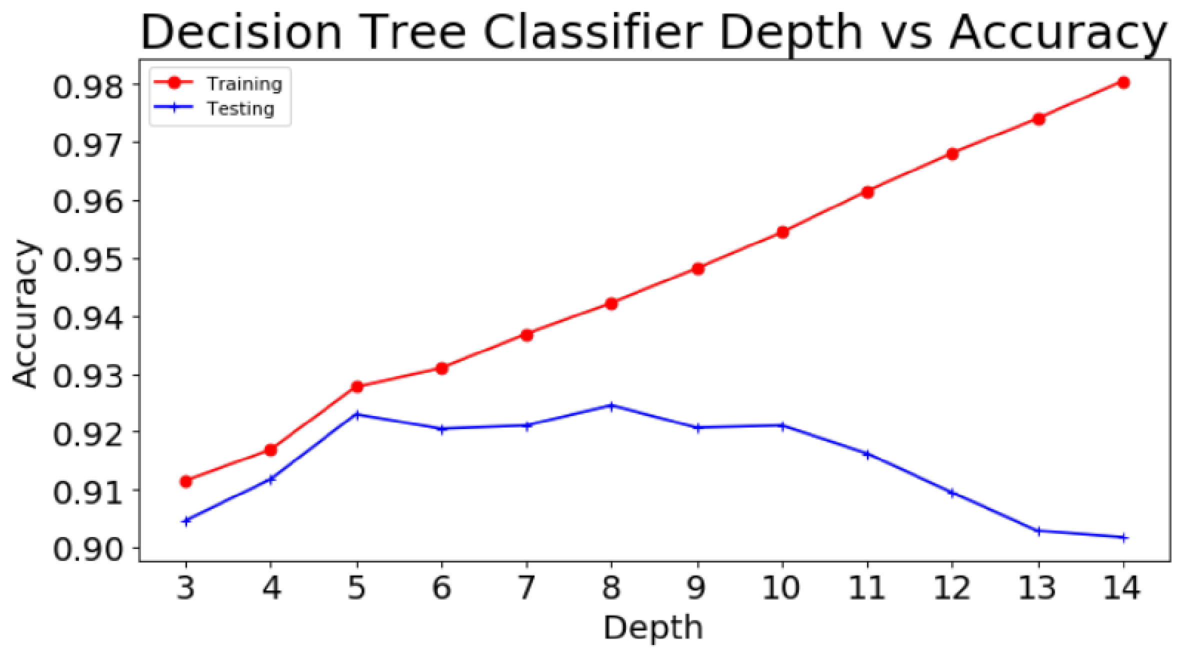
K Neighbors Classifier KNN : F1 Score : 0.923 and Accuracy : 90.52%



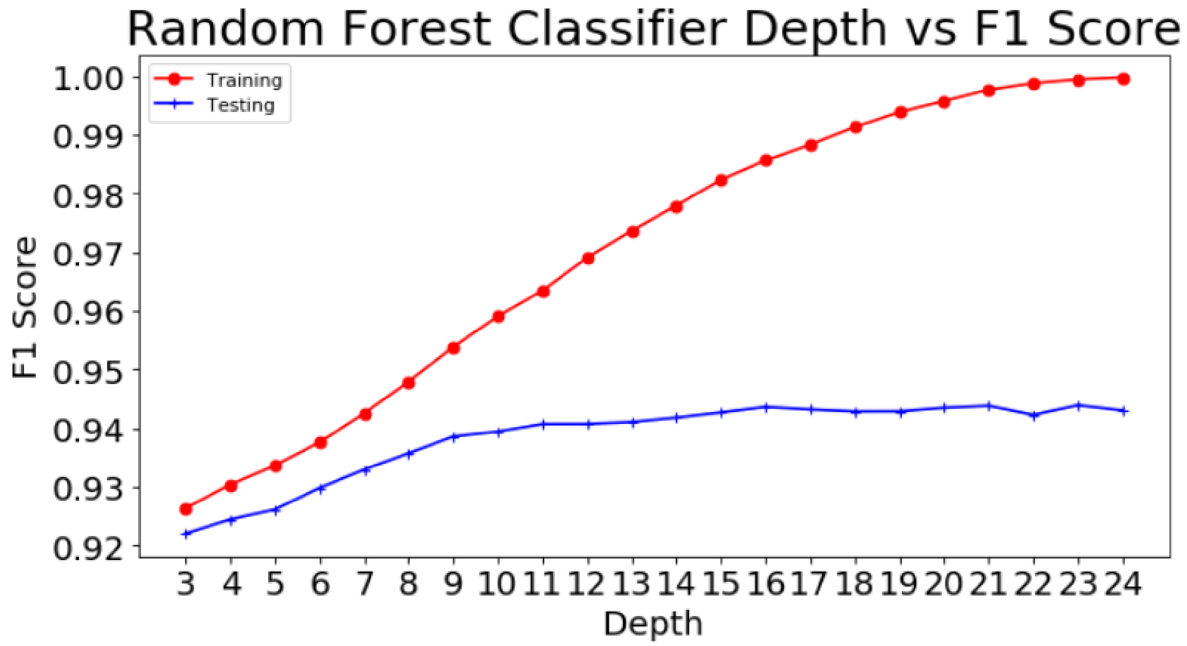


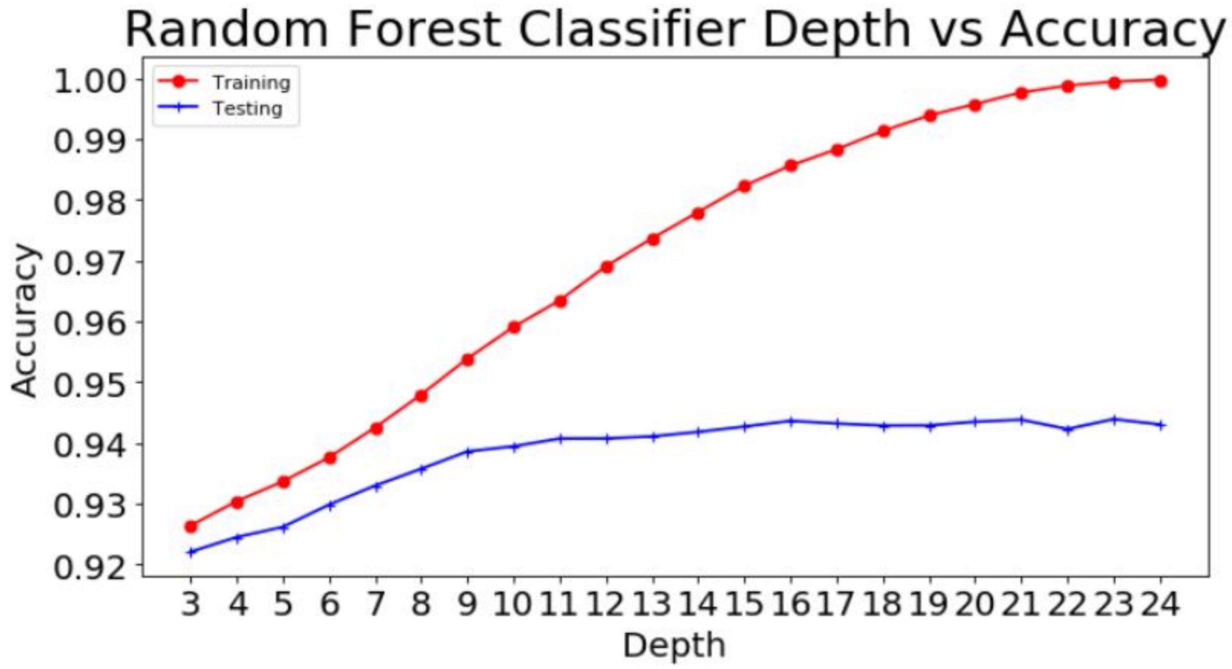
Decision Tree Classifier : F1 Score is : 0.9375 Test Accuracy is : 92.45% at Depth =8



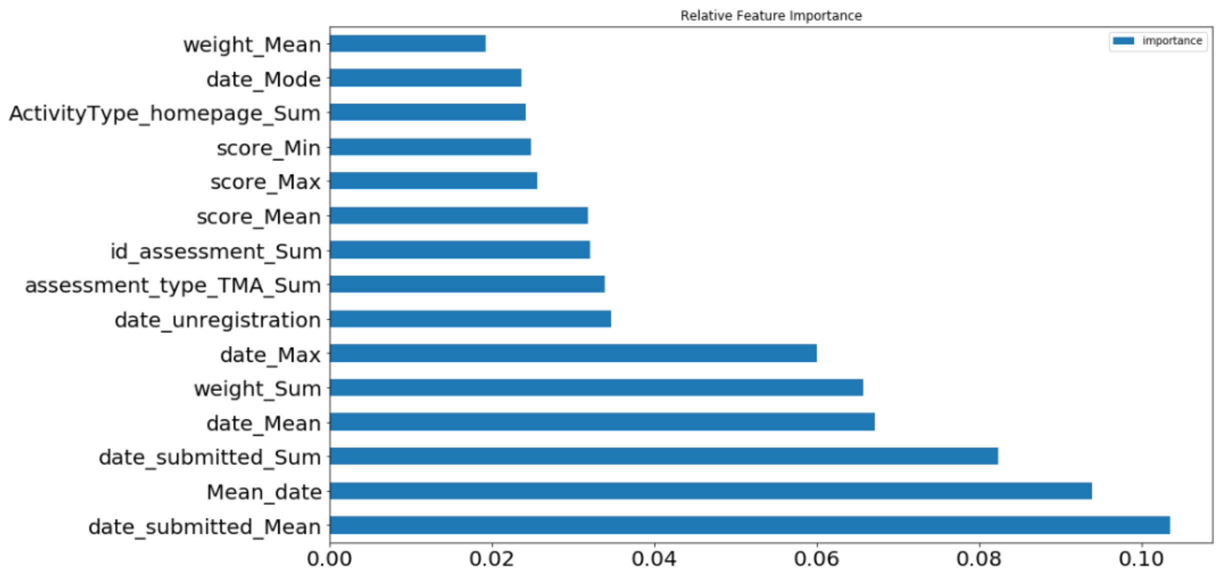


Random Forest Classifier: At Depth = 16, f1 score of 0.9436 and Testing Accuracy of 93.13



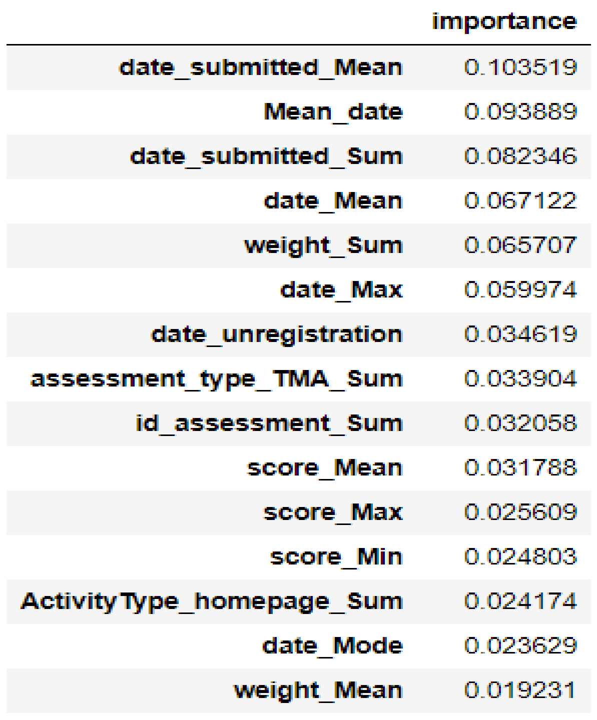


Relative Feature Importances of Features: Best 15 Features for Random Forest Model



PROOF OF BEST FEATURE ENGINEERING:

In top 15 Features that are best for our Random Forest Model of 0.9436 F1 Score and Accuracy of 93.13 %, we have 14 of our derived features. This means that operations of SUM, MAX, MODE, MEAN, MIN on numerical features really really turned out to be well and ALL CREDIT of this much Good F1 Score and Accuracy goes to Feature Engineering Section.



	importance
date_submitted_Mean	0.103519
Mean_date	0.093889
date_submitted_Sum	0.082346
date_Mean	0.067122
weight_Sum	0.065707
date_Max	0.059974
date_unregistration	0.034619
assessment_type_TMA_Sum	0.033904
id_assessment_Sum	0.032058
score_Mean	0.031788
score_Max	0.025609
score_Min	0.024803
ActivityType_homepage_Sum	0.024174
date_Mode	0.023629
weight_Mean	0.019231

XGBOOST MODEL BEST :F1 Score = 0.9421 and Accuracy = 0.9315

Parameters: Best Learning rate 0.081 , Depth = 10

Note that XGBOOST was our best model as it achieved Highest F1 Score and Highest Accuracy.

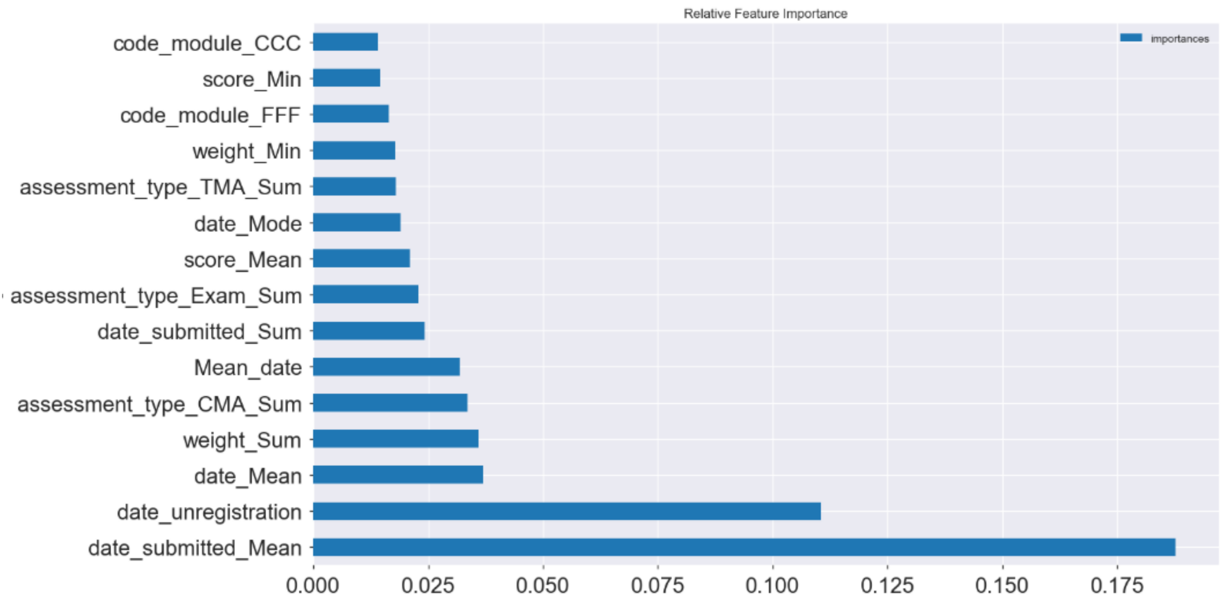
HOW XGBOOST WAS TRAINED:

90% Training and 10% Testing

Untouched until testing phase

90% Training → 80% Training and 20% Validation Set

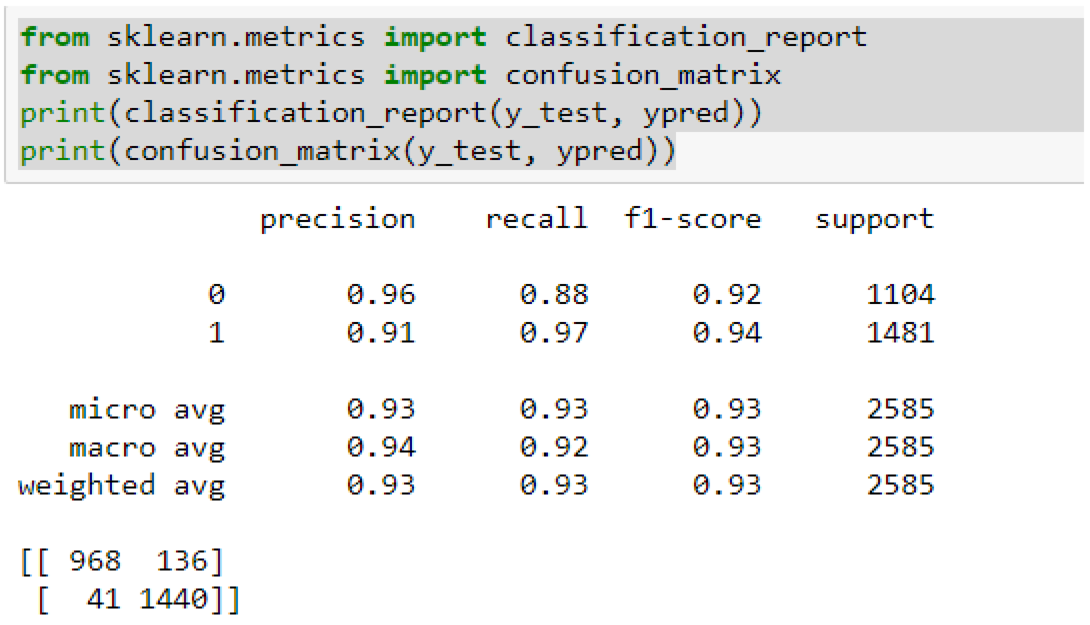
80% Training → Done with 5 Fold- Cross Validation



PROOF OF BEST FEATURE ENGINEERING:

In top 15 Features that are best for our XGBOOST MODEL of 0.9421 F1 Score and Accuracy of 93.15 %, we have 12 of our derived features.

Confusion Matrix and Classification Report:



```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test, ypred))
print(confusion_matrix(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.88      0.92      1104
           1       0.91      0.97      0.94      1481

   micro avg       0.93      0.93      0.93      2585
   macro avg       0.94      0.92      0.93      2585
weighted avg       0.93      0.93      0.93      2585

[[ 968  136]
 [  41 1440]]
```

Interpretation of confusion matrix:

→ Predicted →	0	1
Actual ⬇		
0	968	136
1	41	1440

True Positives (TP): we correctly predicted that student will Pass : 1440

True Negatives (TN): we correctly predicted that Student will Fail : 968

False Positives (FP): we incorrectly predicted that Student will Pass (a "Type I error"): 136

False Negatives (FN): we incorrectly predicted that Student will Fail (a "Type II error") : 41

XGBOOST was chosen to be out final model as it was having High F1 Score of 0. 9421 and highest accuracy of 0.9315.

Conclusions:

- ✓ **XGBOOST was best to use having 93.15% Accuracy and 0.9421 F1 core.**
- ✓ **Derived Features during Feature Engineering Phase were really important for the model.**
- ✓ **It is much important to efficiently use as much data as we have using best appropriate logic.**

Thanks and Kind Regards,
Utsav