Importing the libraries

In [1]:
```python
#important libraries to import.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from pandas_profiling import ProfileReport
```

1. EDA

In [2]:
```python
#read the dataset.
df = pd.read_csv('student_activity_data.csv')
```

In [3]:
```python
#shows first 5 rows.
df.head()
```

Out[3]:

	user	Grade	Active_subjects	Activated_subjects	Live_time_spent	Replay_time_spent	Total_time_spent	Live_sessions	Replay_sessions	Competition	Breakouts	Avg_ranking	Is_Converted	Is_Activated	Paid_amount
0	1	6	1	1	81.816667	1.483333	83.300000	2	1	0	2	3.857143	0	1	0.0
1	2	12	1	0	0.000000	13.800000	13.800000	0	1	0	0	0.000000	0	0	0.0
2	3	12	1	0	0.000000	55.183333	55.183333	0	2	0	0	0.000000	0	0	0.0
3	4	12	1	0	7.216667	0.000000	7.216667	0	0	2	0	0.000000	0	0	0.0
4	5	9	2	0	0.000000	0.250000	0.250000	0	1	0	0	0.000000	0	0	0.0

In [4]:
```python
#shows last 5 rows.
df.tail()
```

Out[4]:

	user	Grade	Active_subjects	Activated_subjects	Live_time_spent	Replay_time_spent	Total_time_spent	Live_sessions	Replay_sessions	Competition	Breakouts	Avg_ranking	Is_Converted	Is_Activated	Paid_amount
9995	9996	12	2	1	0.000000	106.200000	106.200000	0	3	0	0	0.0	0	1	0.0
9996	9997	10	4	0	68.316667	0.000000	68.316667	3	0	1	3	1.0	0	1	0.0
9997	9998	12	2	1	0.000000	175.466667	175.466667	0	6	0	0	0.0	0	1	0.0
9998	9999	11	1	0	0.000000	1.400000	1.400000	0	2	0	0	0.0	0	0	0.0
9999	10000	11	3	0	16.200000	13.350000	29.550000	0	2	1	0	0.0	0	0	0.0

In [5]:
```python
#shows each column and its count and if there is null values or not and the data type.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   user                10000 non-null  int64
 1   Grade               10000 non-null  int64
 2   Active_subjects     10000 non-null  int64
 3   Activated_subjects  10000 non-null  int64
 4   Live_time_spent     10000 non-null  float64
 5   Replay_time_spent   10000 non-null  float64
 6   Total_time_spent    10000 non-null  float64
 7   Live_sessions       10000 non-null  int64
 8   Replay_sessions     10000 non-null  int64
 9   Competition         10000 non-null  int64
 10  Breakouts           10000 non-null  int64
 11  Avg_ranking         10000 non-null  float64
 12  Is_Converted        10000 non-null  int64
 13  Is_Activated        10000 non-null  int64
 14  Paid_amount         10000 non-null  float64
dtypes: float64(5), int64(10)
memory usage: 1.1 MB
```

In [6]:
```python
#shows column names.
df.columns
```

Out[6]:
```
Index(['user', 'Grade', 'Active_subjects', 'Activated_subjects',
       'Live_time_spent', 'Replay_time_spent', 'Total_time_spent',
       'Live_sessions', 'Replay_sessions', 'Competition', 'Breakouts',
       'Avg_ranking', 'Is_Converted', 'Is_Activated', 'Paid_amount'],
      dtype='object')
```

In [6]:
```python
#Shows Statistics for each column.
df.describe()
```

Out[6]:

	user	Grade	Active_subjects	Activated_subjects	Live_time_spent	Replay_time_spent	Total_time_spent	Live_sessions	Replay_sessions	Competition	Breakouts	Avg_ranking	Is_Converted	Is_Activated	Paid_amount
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	9.797800	3.191000	0.795200	306.072680	122.378147	428.450827	10.960600	3.333000	4.25540	2.614200	1.405977	0.031200	0.459500	8.179270
std	2886.89568	2.631921	2.275618	1.341953	1364.086468	2921.344010	3308.242143	40.082811	8.391481	17.66681	13.687932	3.198166	0.173867	0.498382	56.673568
min	1.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.016667	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2500.75000	8.000000	1.000000	0.000000	1.800000	0.000000	4.879167	1.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	5000.50000	11.000000	3.000000	0.000000	24.275000	1.550000	38.491667	2.000000	1.000000	1.00000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	7500.25000	12.000000	4.000000	1.000000	172.837500	13.516667	226.658333	8.000000	3.000000	3.00000	1.000000	1.000000	0.000000	1.000000	0.000000
max	10000.00000	13.000000	18.000000	12.000000	50791.683330	222230.850000	224290.716700	997.000000	380.000000	491.00000	516.000000	43.000000	1.000000	1.000000	1949.000000

In [7]:
```python
#From pandas profil we get a full analysis over the dataset such as correlation, count values and description of each column.
profile = ProfileReport(df)
profile.to_file(output_file='reporting.html')
```

```
R:\Anaconda\lib\site-packages\pandas_profiling\visualisation\plot.py:166: MatplotlibDeprecationWarning: You are modifying the state of a globally registered colormap. In future versions, you will not be able to modify a registere
d colormap in-place. To remove this warning, you can make a copy of the colormap first. cmap = copy.copy(mpl.cm.get_cmap("RdBu"))
  cmap.set_bad(cmap_bad)
```

1.1 Data Cleaning

In [8]:
```python
#we remove values that exist only after the activation is done such as 'Activated_subjects' and the values that highly correlated to each other.
df.drop(['Live_time_spent','Replay_time_spent','Activated_subjects','Paid_amount'], axis=1 ,inplace=True)
```

In [9]:
```python
#remove zero values in grade.
df = df[df['Grade']!=0]
```

In [10]:
```python
#remove zero values in active subject.
df = df[df['Active_subjects']!=0]
```

In [11]:
```python
#shows first 5 rows after cleaning.
df.head()
```

Out[11]:

	user	Grade	Active_subjects	Total_time_spent	Live_sessions	Replay_sessions	Competition	Breakouts	Avg_ranking	Is_Converted	Is_Activated
0	1	6	1	83.300000	2	1	0	2	3.857143	0	1
1	2	12	1	13.800000	0	1	0	0	0.000000	0	0
2	3	12	1	55.183333	0	2	0	0	0.000000	0	0
3	4	12	1	7.216667	0	0	2	0	0.000000	0	0
4	5	9	2	0.250000	0	1	0	0	0.000000	0	0

In [12]:
```python
#shows last 5 rows after cleaning.
df.tail()
```

Out[12]:

	user	Grade	Active_subjects	Total_time_spent	Live_sessions	Replay_sessions	Competition	Breakouts	Avg_ranking	Is_Converted	Is_Activated
9995	9996	12	2	106.200000	0	3	0	0	0.0	0	1
9996	9997	10	4	68.316667	3	0	1	3	1.0	0	1
9997	9998	12	2	175.466667	0	6	0	0	0.0	0	1
9998	9999	11	1	1.400000	0	2	0	0	0.0	0	0
9999	10000	11	3	29.550000	0	2	1	0	0.0	0	0

In [13]:
```python
#Shows Statistics for each column.
df.describe()
```

Out[13]:

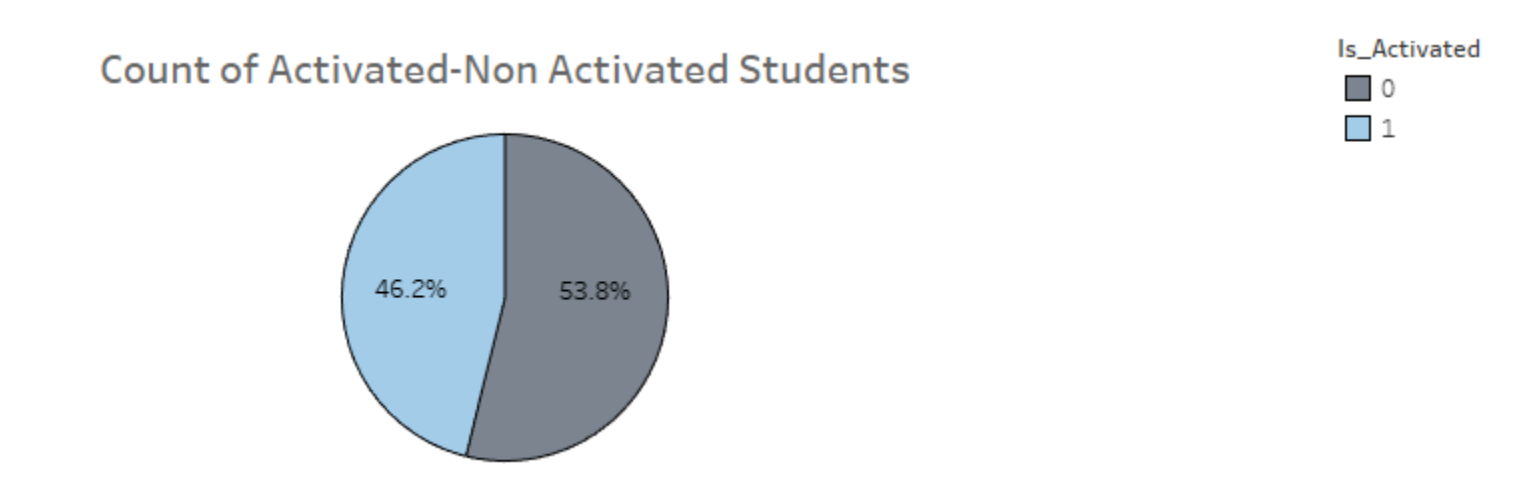
	user	Grade	Active_subjects	Total_time_spent	Live_sessions	Replay_sessions	Competition	Breakouts	Avg_ranking	Is_Converted	Is_Activated
count	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000	9800.000000
mean	4995.696224	9.986939	3.206735	429.922175	10.965612	3.314898	4.252653	2.614184	1.408028	0.030918	0.461531
std	2886.456393	2.274291	2.262522	3335.934656	40.113788	8.350657	17.744460	13.687987	3.206991	0.173105	0.498543
min	1.000000	1.000000	1.000000	0.016667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2498.750000	9.000000	1.000000	5.050000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	4999.500000	11.000000	3.000000	38.833333	2.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
75%	7492.250000	12.000000	4.000000	226.658333	8.000000	3.000000	3.000000	1.000000	1.000000	0.000000	1.000000
max	10000.000000	13.000000	18.000000	224290.716700	997.000000	380.000000	491.000000	516.000000	43.000000	1.000000	1.000000

In [14]:
```python
#From pandas profil we get a full analysis over the dataset such as correlation, count values and description of each column after cleaning.
profile = ProfileReport(df)
profile.to_file(output_file='reporting1.html');
```

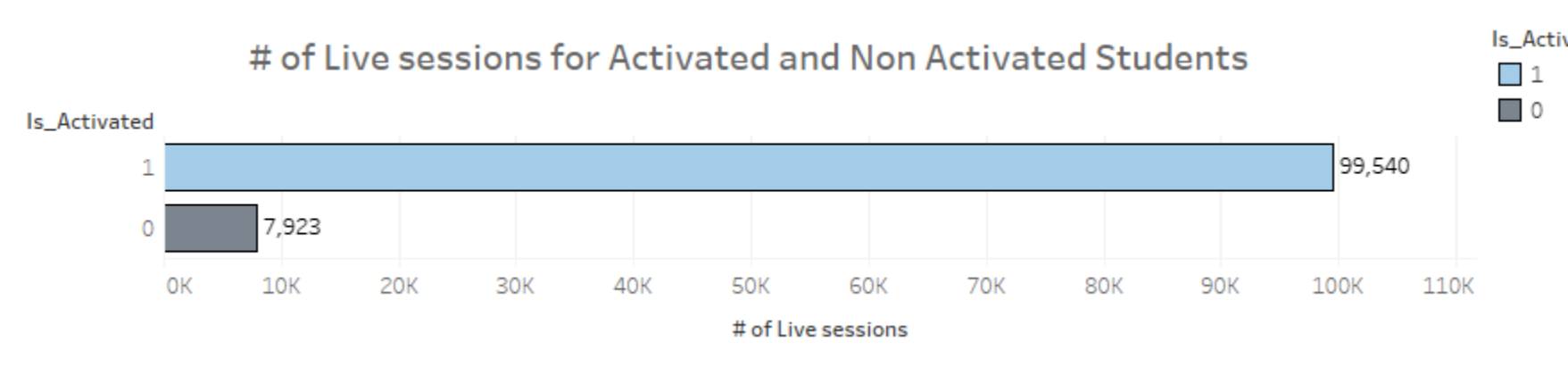
```
R:\Anaconda\lib\site-packages\pandas_profiling\visualisation\plot.py:166: MatplotlibDeprecationWarning: You are modifying the state of a globally registered colormap. In future versions, you will not be able to modify a registere
d colormap in-place. To remove this warning, you can make a copy of the colormap first. cmap = copy.copy(mpl.cm.get_cmap("RdBu"))
  cmap.set_bad(cmap_bad)
```

1.2 Data visualization

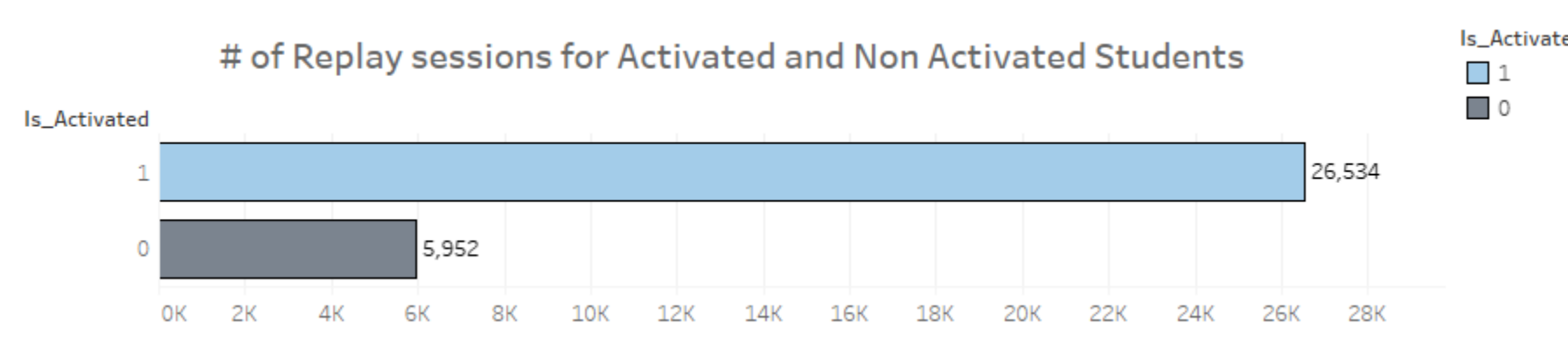
In [15]:
```python
#get our Feature (X) and our Target (y).
X = df.drop(['Is_Activated','user','Grade','Total_time_spent'], axis=1)
y = df['Is_Activated']
```

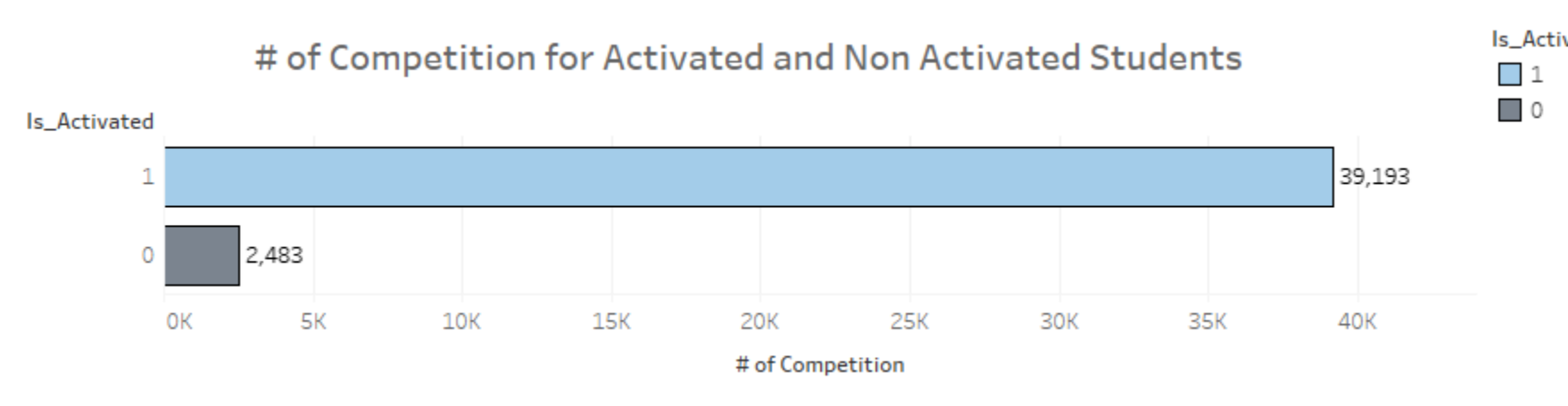


- here is the split of our activated/non activated students

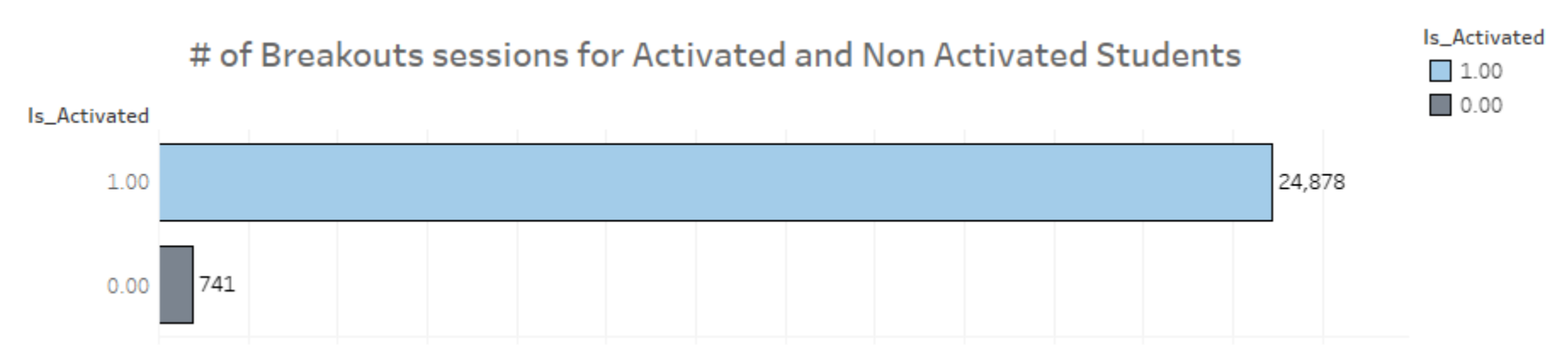


- comparison between activated/non activated students in number of live sessions
- comparison between activated/non activated students in number of replay sessions

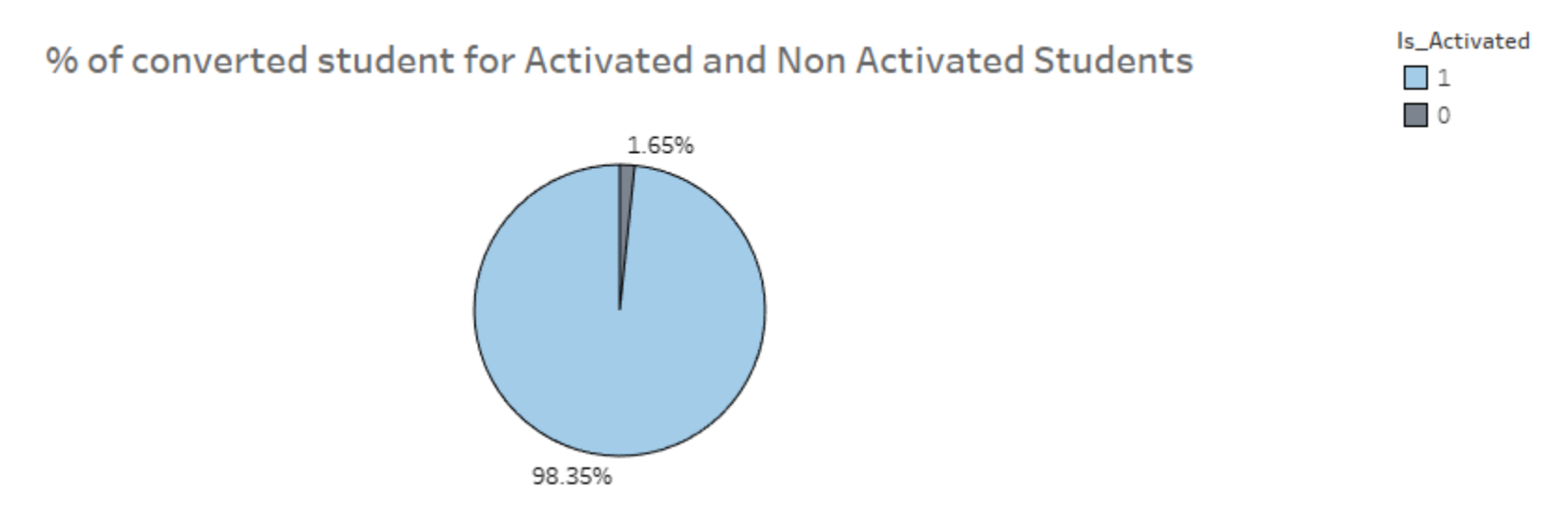




- comparison between activated/non activated students in number of competitions



- comparison between activated/non activated students in number of breakouts



- all converted student splitted over bieng activated/non activated

2. Data Pre-Processing

In [16]:
```python
#importing important libraries for feature scaling and splitting the data into test data and train data.
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [17]:
```python
# splitting the data into test data and a training data.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# scaling our feature using the standardscaler.
sc = StandardScaler()
X_train[X_train.columns] = sc.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = sc.fit_transform(X_test[X_test.columns])
```

3. Prediction Models

3.1 Logistic Regression

In [19]:
```python
# importing logistic regression model.
from sklearn.linear_model import LogisticRegression

#calling the logistic regression model.
lr = LogisticRegression(random_state = 42)

# fitting the model.
lr.fit(X_train, y_train)
```

Out[19]:
```
LogisticRegression(random_state=42)
```

In [20]:
```python
# predicting using the test data (x_test).
y_pred = lr.predict(X_test)
```

In [21]:
```python
# importing important libraries for evaluating classification models.
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error
from sklearn.metrics import classification_report

# printing the confusion matrix.
print(confusion_matrix(y_test, y_pred))

# printing the accuracy score.
print(accuracy_score(y_test, y_pred))

# printing the classification report.
print(classification_report(y_test,y_pred))
```

```
[[979  84]
 [160 737]]
0.8755102040816326
              precision    recall  f1-score   support

           0       0.86      0.92      0.89      1063
           1       0.90      0.82      0.86       897

    accuracy                           0.88      1960
   macro avg       0.88      0.87      0.87      1960
weighted avg       0.88      0.88      0.87      1960
```

3.2 Random Forest

In [22]:
```python
# importing Random Forest Classifier model.
from sklearn.ensemble import RandomForestClassifier

# by trial and error i found n_estimators = 1200 gives the best results for random forest.
rfc = RandomForestClassifier(n_estimators=1200)

# fitting the model with the training data.
rfc.fit(X_train,y_train)
```

Out[22]:
```
RandomForestClassifier(n_estimators=1200)
```

In [23]:
```python
# predicting using the test data (x_train).
y_pred1 = rfc.predict(X_test)
```

In [24]:
```python
# printing the confusion matrix.
print(confusion_matrix(y_test, y_pred1))

# printing the accuracy score.
print(accuracy_score(y_test, y_pred1))

# printing the classification report.
print(classification_report(y_test,y_pred1))
```

```
[[911 152]
 [134 763]]
0.8540816326530613
              precision    recall  f1-score   support

           0       0.87      0.86      0.86      1063
           1       0.83      0.85      0.84       897

    accuracy                           0.85      1960
   macro avg       0.85      0.85      0.85      1960
weighted avg       0.85      0.85      0.85      1960
```

3.3 XGBoost

In [30]:
```python
# importing xgboost model.
import xgboost as xgb
```

In [31]:
```python
# i tried multiple values for n_estimators but it did not make any difference so i set it to match the random forest
gbm = xgb.XGBClassifier(
                        n_estimators=1200,
                        max_depth=4,
                        objective='binary:logistic',
                        learning_rate=.05,
                        subsample=.8,
                        min_child_weight=3,
                        colsample_bytree=.8,
                        use_label_encoder=False
                       );

eval_set=[(X_train,y_train),(X_test,y_test)]
# fitting the model with the training data, the evaluation set and the evaluation metric (error)
fit_model = gbm.fit(
                    X_train, y_train,
                    eval_set=eval_set,
                    eval_metric='error', #new evaluation metric: classification error
                    early_stopping_rounds=50,
                    verbose=False
                   );
```

In [32]:
```python
# predicting using the test data (x_test).
y_pred2 = gbm.predict(X_test)
```

In [33]:
```python
# printing the confusion matrix
print(confusion_matrix(y_test, y_pred2))

# printing the accuracy score
print(accuracy_score(y_test, y_pred2))

# printing the classification report
print(classification_report(y_test,y_pred2))
```

```
[[971  92]
 [145 752]]
0.8790816326530613
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      1063
           1       0.89      0.84      0.86       897

    accuracy                           0.88      1960
   macro avg       0.88      0.88      0.88      1960
weighted avg       0.88      0.88      0.88      1960
```

In [29]:
```python
# importance plot to show which featue had the most affect on the model.
feature_important = gbm.get_booster().get_score(importance_type='weight')
keys = list(feature_important.keys())
values = list(feature_important.values())
df = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_values(by = "score", ascending=False)
df.nlargest(10, columns="score").sort_values(by='score',ascending=True).plot(kind='barh', figsize = (20,10));
```