Phase 4 Deliverable

The Jupyter Notebooks are attached to the deliverable.

I. **Mention the algorithms you used:** For classification: Decision Tree and Random Forest. For clustering: K-Means and DBSCAN.

II. **Show the models you constructed:**

Decision Tree

```
In [17]:
1 #Fit the training data onto the decision tress classifier
2 dt = DecisionTreeClassifier(criterion='entropy')
3 dt.fit(X_train, y_train)

Out[17]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_dept
h=None,
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
                splitter='best')
```

Random Forest

```
In [21]:
1 rf = RandomForestClassifier(n_estimators=600)
```

```
In [22]:
1 rf.fit(X_train,y_train)

Out[22]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
                max_depth=None, max_features='auto', max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, n_estimators=600, n_jobs=Non
e,
                oob_score=False, random_state=None, verbose=0,
                warm_start=False)
```

K-Means

```
In [16]:
1 # Fit the model to all the data except for the Is-Nighttime label.
2 kmeans.fit(df.drop('Is-Nighttime',axis=1))

Out[16]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

DBSCAN

```
In [25]:
1 clustering = DBSCAN().fit(df.values)
```

III. **Include the evaluation metrics, i.e. the accuracies of the models, the times taken to build the models and the confusion matrices, and**

Decision Tree

In [95]:
```
after = datetime.datetime.now()
time_taken = after - before
```

In [96]:
```
y_pred = dt.predict(X_test)
recall = recall_score(y_pred, y_test) * 100
precision = precision_score(y_pred, y_test) * 100
print("Recall of Decision Tree {:.2f} %".format(recall))
print("precision of Decision Tree {:.2f} %".format(precision))
print("Time taken:", time_taken.total_seconds())
```

```
Recall of Decision Tree 64.88 %
precision of Decision Tree 63.52 %
Time taken: 0.248331
```

In [97]:
```
predictions = dt.predict(X_test)
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.52      0.54      0.53      4867
           1       0.65      0.64      0.64      6559

   micro avg       0.59      0.59      0.59     11426
   macro avg       0.59      0.59      0.59     11426
weighted avg       0.59      0.59      0.59     11426
```

In [98]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[2612 2255]
 [2393 4166]]
```

Random Forest

In [102]:
```
after = datetime.datetime.now()
time_taken = after - before
```

In [103]:
```
y_pred = rf.predict(X_test)
recall = recall_score(y_pred, y_test) * 100
precision = precision_score(y_pred, y_test) * 100
print("Recall of Decision Tree {:.2f} %".format(recall))
print("precision of Decision Tree {:.2f} %".format(precision))
print("Time taken:", time_taken.total_seconds())
```

```
Recall of Decision Tree 64.51 %
precision of Decision Tree 69.26 %
Time taken: 23.884336
```

In [104]:
```
predictions = rf.predict(X_test)
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.54      0.49      0.51      4867
           1       0.65      0.69      0.67      6559

   micro avg       0.60      0.60      0.60     11426
   macro avg       0.59      0.59      0.59     11426
weighted avg       0.60      0.60      0.60     11426
```

In [105]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[2368 2499]
 [2016 4543]]
```

K-Means

```
[[ 9399  6824]
 [10555 11307]]
              precision    recall  f1-score   support

           0       0.47      0.58      0.52     16223
           1       0.62      0.52      0.57     21862

   micro avg       0.54      0.54      0.54     38085
   macro avg       0.55      0.55      0.54     38085
weighted avg       0.56      0.54      0.55     38085

Time taken: 0.989418
```

DBSCAN

In [195]:
```
clustering = DBSCAN().fit(df.values)
```

In [196]:
```
after = datetime.datetime.now()
time_taken = after - before
print("Time taken:", time_taken.total_seconds())
```

```
Time taken: 8.23498
```

IV. Contain a section on "lessons learned from applying these two techniques to the data".

1. It was quite fast to do it. We already had an organized dataset and we didn't have class imbalance, so training the model was simply following a few steps.
2. The answers were not as good as we expected. Probably because it is a simple exercise, we could have made more effort in some parts of the project in order to improve the accuracy.
3. The clustering part is harder to understand. For example, we don't know how to exactly use the clustering algorithm and how to exactly evaluate it.
4. Python and Pandas have lots of people working on it, so finding information about it is quite fast.
5. Having a good and organized dataset is fundamental. Having good quality data generates good quality results.