2.1 For splitting the dataset, I have used a random sample generator where random.sample gives the element of the passed list for specified numbers. In this case the list used is the index of the dataset and number is integer value of 80% of length of the dataset which is 365410. Here the train dataset contains 365410 rows and the test dataset contains 91353.
Here we can also use the train_test_split function of sklearn library, which splits the dataset in train and test dataset based on the ratio given.

2.2 Here, I have used XGBoost, KNN and Random Forest classifiers. XGBoost is known to reduce overfitting and during execution I find out that it takes less time then Adaboost. The reason for XGBoost to run faster is the parallelization used in the code which helps in reducing run time while modelling. Among SVM, KNN, Decision Trees, Random Forest and Naive Bayes, SVM uses a kernel method to find out the classes, as the dataset provided is quite large it takes more time to train the model and is likely to overfit. KNN is a distance based algorithm and as we have both numeric and categorical features in the dataset, we can use this algorithm for modelling. Random Forest is based on decision trees and it is seen to outperform independent decision trees in cases of large dataset and in case of overfitting. Naive Bayes algorithm uses assumption of independence between the features which is unlikely in given data as number of confirmed cases, active cases etc are highly correlated
Based on above mentioned reasons, the algorithms selected are XGBoost, Random Forest and KNN..

2.3 The Metrics used in the problem are Accuracy score, Confusion metric, Precision, Recall and F1-score. Accuracy gives the percentage of true classification. Confusion metric tells the count of True Positive classifications, False Positive classifications, True Negative classifications and False Negative classifications. In case of multiclass problems, True positives are those data points which are correctly classified to that class, True Negatives are data points which do not belong to that class and also classified to classes other than that. False Positives are those data points which belong to another class but classified to that class and False Negatives are those data points which belong to that class but classified to another class. Lets the classes are A,B and C, Then A classed as A is True positive, A classified as B/C are False Negative, B/C classified as A are False Positive and B/C classified as B/C are True Negative. Precision is the rate of True Positive over Predicted Positive while Recall is rate of True Positive over Actual Positive. F1-Score is the harmonic mean of precision and recall.
Analysis of all the metrics mentioned here other than accuracy is important for the given problem as it contains multiple classes which are not balanced. Due to the imbalanced nature of the dataset there is high probability that the model will be biased toward the higher classes and so if we calculated the accuracy will not give true measure.

Metrics value calculated

```
############XGBoost: n_estimators = 20 ############
Train: Accuracy 78.27%                          Test: Accuracy 78.14%
     precision   recall  f1-score   support         precision  recall   f1-score   support
      0      0.00      0.00      0.00      4080         0.00        0.00      0.00      1034
      1      0.67      1.00      0.80    157685         0.67        1.00      0.80     39499
```

```
        2      1.00      1.00      1.00    128262            1.00         1.00      1.00    31870
        3      0.74      0.00      0.00     75383             0.74        0.00      0.00     18950
############Random Forest: n_estimators = 20   ############
Train: Accuracy 81.11%                                 Test: Accuracy 80.93%
          precision    recall  f1-score   support          precision    recall  f1-score   support
        0      0.89      0.06      0.11     4080             0.55      0.04      0.07      1034
        1      0.70      0.98      0.82    157685            0.70      0.97      0.82     39499
        2      1.00      1.00      1.00    128262            1.00      1.00      1.00     31870
        3      0.78      0.19      0.30     75383             0.78      0.19      0.30     18950
############KNN: n_neighbors = 20        ############
Train: Accuracy 80.71%                                 Test: Accuracy 80.61%
          precision    recall  f1-score   support          precision    recall  f1-score   support
        0      0.70      0.03      0.06     4080             0.61      0.03      0.06      1034
        1      0.71      0.96      0.81    157685            0.71      0.95      0.81     39499
        2      1.00      1.00      1.00    128262            1.00      1.00      1.00     31870
        3      0.69      0.21      0.32     75383             0.69      0.22      0.33     18950
```

Based On accuracy we don’t see overfitting but as we have imbalanced classes in the dataset we will check for the other evaluation metrics. Here we can see that metric values for both the train and test are the same in case of XGBoost and the values are discouraging compared to other two models. Although the metric values are similar for train and test cases in Random Forest but if we see the first class, compared to Random Forest, KNN seems to perform better in test cases.

2.4 The models trained in this case seem to be a good fit model and neither overfitted nor underfitted. To find out whether the model is overfitted or not, we check the metric value for the test dataset and compare them with evaluation scores of the train dataset. If the performance of the model is very good on train data but its performs poorly on test data then the model is said to be an overfitted model. As in our cases all the models are equally performing good on both the train and test dataset, we can say that the trained models are a good fit model. Also we have achieved similar metric values for different values of hyperparameters.

Random Forest - F1 score:

	n_estimators=10	n_estimators=20	n_estimators=100	n_estimators=20,depth= 5
	Train,test	train,test	train,test	train,test
0	0.11, 0.08	0.11, 0.08	0.11, 0.07	0.00, 0.00
1	0.82, 0.82	0.82, 0.82	0.82, 0.82	0.80, 0.80
2	1.00, 1.00	1.00, 1.00	1.00, 1.00	1.00, 1.00
3	0.31, 0.30	0.31, 0.30	0.30, 0.30	0.01, 0.01

From the above score, we can say that changing the n_estimatator doesn’t have much effect on the evaluation metric while if we fix the max_depth of trees to 5 then performance is decreased for the minor classes.