Import Libraries

In [16]:

```
#Import all libraries

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from sklearn import tree

from matplotlib import pyplot as plt
```

Import Data

In [17]:

```
#Import the event log of BPIC 2013
data = pd.read_csv("VINST cases incidents modified excel.csv")
data.head()
```

Out[17]:

	Case ID	ST	Line	Complete Timestamp	Variant	Variant index	Status	Sub Status	Involved ST Function Div	Invo Org
0	364285768	V5	3	04:00.0	Variant 520	520	Accepted	In Progress	A2_5	Org
1	364285768	V30	1	04:00.0	Variant 520	520	Queued	Awaiting Assignment	A2_4	Org
2	364285768	V13	2	04:00.0	Variant 520	520	Accepted	In Progress	A2_5	Org
3	364285768	V13	2	04:00.0	Variant 520	520	Completed	Resolved	A2_5	Org
4	364285768	V30	1	04:00.0	Variant 520	520	Queued	Awaiting Assignment	A2_4	Org

Drop columns

In [18]:

```
#Drop the unnecessary columns
data = data.drop(["Complete Timestamp", "Variant", "Variant index", "Involved
                  "Involved Org line 3", "SR Latest Impact", "Product", "Coun
data
```

Out[18]:

	Case ID	ST	Line	Status	Sub Status
0	364285768	V5	3	Accepted	In Progress
1	364285768	V30	1	Queued	Awaiting Assignment
2	364285768	V13	2	Accepted	In Progress
3	364285768	V13	2	Completed	Resolved
4	364285768	V30	1	Queued	Awaiting Assignment

	Case ID	ST	Line	Status	Sub Status
...	...	...	...	...	...
65528	740866691	C9	1	Completed	In Call
65529	740866708	C9	1	Accepted	In Progress
65530	740866708	C9	1	Accepted	In Progress
65531	740866708	C9	1	Completed	In Call
65532	740866821	N36	1	Accepted	In Progress

65533 rows × 5 columns

Remove null values

In [19]:
```
#remove null values
data = data.dropna()
```

In [20]:
```
#Check the data types of all columns in the data
data.dtypes
```

Out[20]:
```
Case ID         int64
ST             object
Line            int64
Status         object
Sub Status     object
dtype: object
```

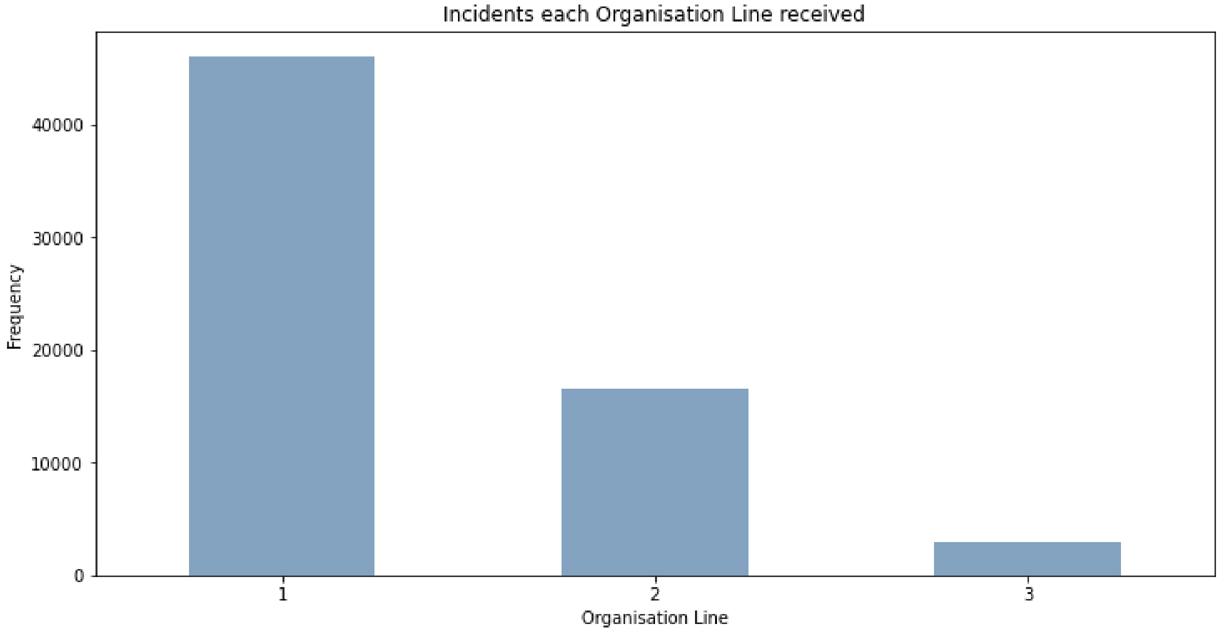
In [21]:
```
# Making the data classification ready
#one hot encoding
data = pd.get_dummies(data)
data.dtypes
```

Out[21]:
```
Case ID                                int64
Line                                   int64
ST_A1                                  uint8
ST_A10                                 uint8
ST_A11                                 uint8
                                       ...
Sub Status_Wait                        uint8
Sub Status_Wait - Customer             uint8
Sub Status_Wait - Implementation       uint8
Sub Status_Wait - User                 uint8
Sub Status_Wait - Vendor               uint8
Length: 617, dtype: object
```

Initial exploration

In [22]:
```
# Plot for number of cases handled by each organisation line

ax = data['Line'].value_counts().plot(kind='bar', figsize=(12,6),
                                      title="Incidents each Organisation Line
ax.set_xlabel("Organisation Line")
ax.set_ylabel("Frequency")
plt.xticks(rotation=360)
plt.show()
```



Analysis using Decision Tree Classification method

Split feature and target columns -> split data into training and test data-> train the model

In [23]:
```
#Separate the feature columns and the target column, that is, dependent and i
X = data.drop(['Line'], axis=1) #Feature columns
y = data['Line']  #Target column
```

In [24]:
```
#Split the data into training and testing datasets
rs = 100
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stra
#Split input data into 70% train data and 30% test data
```

In [25]:
```
#Defining DecisionTreeClassifier
model = DecisionTreeClassifier()

#Train the model using a fit function
model.fit(X_train, y_train)
```

Out[25]:
```
DecisionTreeClassifier()
```

In [26]:
```
#Predict the test data set values using the model above
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     13813
           2       0.99      1.00      1.00      4974
           3       1.00      0.99      0.99       873

    accuracy                           1.00     19660
   macro avg       1.00      1.00      1.00     19660
weighted avg       1.00      1.00      1.00     19660
```

In [27]:
```
# Calculating the accuracy of the model using .score function
```

```
print("Train accuracy:", model.score(X_train, y_train)*100)
print("Test accuracy:", model.score(X_test, y_test)*100)
```

```
Train accuracy: 100.0
Test accuracy: 99.75076297049847
```

.score() is used to understand the accuracy of the model against both the training and the test data, that is to understand how well or with what accuracy the model predicts correctly.

We use classification_report to understand the performance of the classifier. It gives us a number of statistics - Preciison, recall, f1-score and support. They are further discussed and explained in the Analysis section of the report.

Feature Importance

It is important to understand the input variables or the features that have the most impact on the decision making process in the model. For this purpose, we use feature*importances* available in sklearn. We then generate the results along the feature names and arrange them in value of importance.

In [28]:

```
# grab feature importances from the model and feature name from the original
importances = model.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]
for i in indices:
 print(feature_names[i], ':', importances[i])
```

```
ST_G230 : 0.06310667557337099
ST_G96 : 0.040349733003552234
ST_G97 : 0.0399177597058290З
ST_S42 : 0.03623763163949318
Case ID : 0.022623965040077496
ST_G51 : 0.019268911085004115
ST_V37 : 0.019181290583133916
ST_G140 : 0.017849248039123272
ST_L38 : 0.01725282979357193
ST_G271 : 0.014640179883620656
ST_G22 : 0.014157890470808673
ST_D8 : 0.013263035742649315
ST_D4 : 0.012625761920302946
ST_S56 : 0.012596107710338394
ST_D2 : 0.012208223988900474
ST_D5 : 0.012205090593867006
ST_N36 : 0.011957314045391437
ST_G92 : 0.011900688472574278
ST_S49 : 0.011749283636493562
ST_V17 : 0.011617891671560436
```

In [29]:

```
from sklearn import tree

a = tree.export_graphviz(model,
                         out_file="DecisionTree_default.dot",
                         feature_names=X.columns,
                         filled = True)
```

The dot files are later converted into png using the Terminal and the following syntax -

cd ~/(location of dot file)

dot -Tpng (dot file name).dot -o (png file name you want).png

However, the resulting model with the default values of the parameter is complex. Hence, we find out the optimal parameters and then generate the decision tree.

TUNING OF PARAMETERS - Model 1

Finding out optimal parameters using GridSearchCV

Hyperparameter tuning is done using the GridSearchCV function available in scikit-learn's model_selection package. We include the paraemters of interest in the parameter grid. It then evaluates the model for each combination of the parameters and then picks out the "best".

Before getting the best parameters, we have to determine the parameters along with the possible values that need to be explored. In our study, we have focussed on the following three hyperparameters – criterion, max_depth and min_samples_leaf.

- Criterion - the function to measure the quality of split. There are two – "gini" for Gini impurity and "entropy" for information gain.
- Max_depth – It controls the maximum depth of the decision tree. We set our range from 1-20.
- Min_samples_leaf – The minimum number of samples required to be at a leaf node. We set the range of 1-25 with step of 5.

One can look for more information on the other parameters on this documentation - https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

In this section, we find the optimal parameters for our decision tree using GridSearchCV. We set the values as follows:

- cv =10, which means it is a 10-fold cross validation
- estimator = DecisionTreeClassifier() with random state of 100 as defined above
- return_train_score = True to returnt he training score value

More information on GridSearchCV and its parameters and values can be found in the following link to the documentation- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

In [40]:
```
#GridSearchCV
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(1, 20),
          'min_samples_leaf': range(0, 25, 5)[1:]}

optimal_dt = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier
                          return_train_score=True, cv=10)
```

In [41]:
```
optimal_dt.fit(X_train, y_train)
```

Out[41]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 20),
                         'min_samples_leaf': range(5, 25, 5)},
             return_train_score=True)
```

In [42]:
```
print(optimal_dt.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 14, 'min_samples_leaf': 5}
```

In [43]:
```
print(optimal_dt.best_estimator_)
```

```
DecisionTreeClassifier(max_depth=14, min_samples_leaf=5, random_state=100)
```

Testing the performance of the model

In [105…
```
print("Train accuracy:", optimal_dt.score(X_train, y_train)*100)
print("Test accuracy:", optimal_dt.score(X_test, y_test)*100)
```

```
Train accuracy: 77.97397161729121
Test accuracy: 77.90437436419126
```

Building decision tree with parameter values as received above using GridSearchCV - 1

Define the parameters according to the results obtained from optimal tuning and then build a decision tree model.

In [55]:
```
#Defining DecisionTreeClassifier
final = DecisionTreeClassifier(criterion='gini', max_depth = 14, min_samples_

#Train the model using a fit function
final.fit(X_train, y_train)
```

Out[55]:
```
DecisionTreeClassifier(max_depth=14, min_samples_leaf=5)
```

Check the performance of the model built using accuracy and classification report

In [57]:
```
#Predict the test data set values using the model above
y_pred_final = final.predict(X_test)
print(classification_report(y_test, y_pred_final))
```

```
              precision    recall  f1-score   support

           1       0.76      1.00      0.86     13813
           2       1.00      0.27      0.42      4974
           3       1.00      0.20      0.33       873

    accuracy                           0.78     19660
   macro avg       0.92      0.49      0.54     19660
weighted avg       0.83      0.78      0.73     19660
```

In [106…
```
print("Train accuracy:", final.score(X_train, y_train)*100)
print("Test accuracy:", final.score(X_test, y_test)*100)
```

```
Train accuracy: 77.97397161729121
Test accuracy: 77.90437436419126
```

Feature importance analysis

In [59]:

```
# grab feature importances from the model and feature name from the original 
importances = final.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]
for i in indices:
 print(feature_names[i], ':', importances[i])
```

```
ST_G230 : 0.1902842299298085
ST_G96 : 0.12166570022417966
ST_G97 : 0.12036318023622795
ST_S42 : 0.10926656757547032
ST_G51 : 0.05810114182188254
ST_V37 : 0.05783694156774224
ST_G140 : 0.05382046173549192
ST_L38 : 0.05202209435930402
ST_G271 : 0.044144226104094554
ST_G22 : 0.04268998899389329
ST_D8 : 0.0399917523762968
ST_S56 : 0.03798077832491711
ST_D5 : 0.03680175264782172
ST_V17 : 0.03503118410286944
ST_G33 : 0.0
ST_G331 : 0.0
ST_G329 : 0.0
ST_G328 : 0.0
ST_G327 : 0.0
ST_G325 : 0.0
```

Visualising the Decision Tree with optimal parameters

In [75]:

```
from sklearn import tree

a = tree.export_graphviz(final,
                         out_file="DecisionTreeFinal.dot",
                         feature_names=X.columns,
                         filled = True)
```

TUNING OF PARAMETERS - Model 2

Finding optimal parameters using GridSearchCV but max depth range 1-10

In [63]:

```
#GridSearchCV
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(1, 10),
          'min_samples_leaf': range(0, 25, 5)[1:]}

optimal_dt_2 = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifi
                   return_train_score=True, cv=10)
```

In [64]:

```
optimal_dt_2.fit(X_train, y_train)
```

Out[64]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 10),
                         'min_samples_leaf': range(5, 25, 5)},
             return_train_score=True)
```

In [66]:
```
print(optimal_dt_2.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 5}
```

In [67]:
```
print(optimal_dt_2.best_estimator_)
```

```
DecisionTreeClassifier(max_depth=9, min_samples_leaf=5, random_state=100)
```

Building decision tree with parameter values as received above using GridSearchCV - 2

In [69]:
```
#Defining DecisionTreeClassifier
final2 = DecisionTreeClassifier(criterion='gini', max_depth = 9, min_samples_

#Train the model using a fit function
final2.fit(X_train, y_train)
```

Out[69]:
```
DecisionTreeClassifier(max_depth=9, min_samples_leaf=5)
```

Check the performance of the model built using accuracy and classification report

In [107…
```
#Predict the test data set values using the model above
y_pred_final2 = final2.predict(X_test)
print(classification_report(y_test, y_pred_final2))
```

```
              precision    recall  f1-score   support

           1       0.75      1.00      0.86     13813
           2       1.00      0.24      0.39      4974
           3       1.00      0.11      0.20       873

    accuracy                           0.77     19660
   macro avg       0.92      0.45      0.48     19660
weighted avg       0.83      0.77      0.71     19660
```

In [108…
```
print("Train accuracy:", final2.score(X_train, y_train)*100)
print("Test accuracy:", final2.score(X_test, y_test)*100)
```

```
Train accuracy: 76.93196433631984
Test accuracy: 76.83621566632756
```

Feature importance analysis

In [72]:
```
# grab feature importances from the model and feature name from the original
importances = final2.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]
```

```
for i in indices:
 print(feature_names[i], ':', importances[i])
```

```
ST_G230 : 0.23564477927551886
ST_G96 : 0.15066875003411437
ST_G97 : 0.1490557312612184
ST_S42 : 0.13531387339882633
ST_G51 : 0.07195147356837459
ST_V37 : 0.07162429243205871
ST_G140 : 0.06665035158638627
ST_L38 : 0.06442328377538369
ST_G271 : 0.05466746466811859
ST_G335 : 0.0
ST_G334 : 0.0
ST_G332 : 0.0
ST_G331 : 0.0
Sub Status_Wait - Vendor : 0.0
ST_G338 : 0.0
ST_G33 : 0.0
ST_G329 : 0.0
ST_G328 : 0.0
ST_G327 : 0.0
ST_G325 : 0.0
```

Visualising the Decision Tree with optimal parameters

In [74]:

```
from sklearn import tree

a = tree.export_graphviz(final2,
                         out_file="DecisionTreeFinal2.dot",
                         feature_names=X.columns,
                         filled = True)
```

Comparing the models obtained above

Comparing the accuracies of both the models

In [109…

```
y_pred_dt_final = final.predict(X_test)
y_pred_dt_final2 = final2.predict(X_test)

print("Accuracy score on test for Model 1:", accuracy_score(y_test, y_pred_dt_
print("Accuracy score on test for Model 2:", accuracy_score(y_test, y_pred_dt_
```

```
Accuracy score on test for Model 1: 77.90437436419126
Accuracy score on test for Model 2: 76.83621566632756
```

Based on the accuracy scores, we observe that the model 1 performs better than model 2.

Plot Confusion Matrices

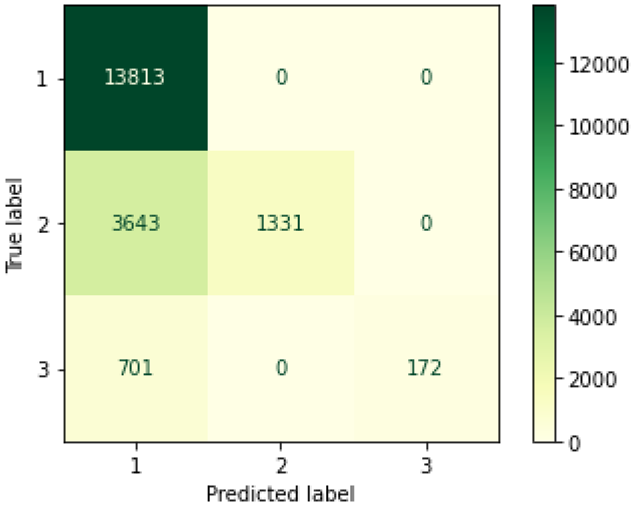
Scikit-learn library has plot_confusion_matrix function that enables us plot the true positives and false positives.

In [100…

```
from sklearn.svm import SVC
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(final, X_test, y_test, cmap=plt.cm.YlGn)
print("Confusion matrix for Model 1")
plt.show()
```

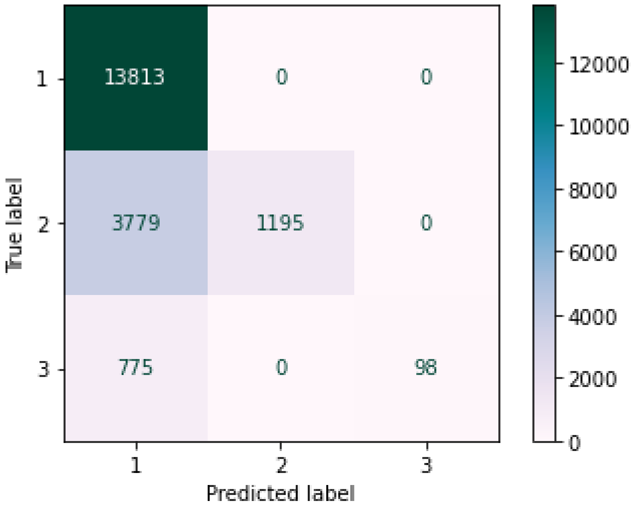
```
Confusion matrix for Model 1
```



In [101…

```
plot_confusion_matrix(final2, X_test, y_test, cmap=plt.cm.PuBuGn)
print("Confusion matrix for Model 2")
plt.show()
```

```
Confusion matrix for Model 2
```



In [ ]: