Data Science PROJECT  
Client: ABC Tech | Category: ITSM - ML  
Project Ref: PM-PR-0012

**Business Case:**

ABC Tech is an mid-size organisation operation in IT-enabled business segment over a decade. On an average ABC Tech receives 22-25k IT incidents/tickets, which were handled to best practice ITIL framework with incident management, problem management, change management and configuration management processes. These ITIL practices attained matured process level and a recent audit confirmed that further improvement initiatives may not yield return of investment.

ABC Tech management is looking for ways to improve the incident  
management process as recent customer survey results shows that  
incident management is rated as poor.

Machine Learning as way to improve ITSM processes ABC Tech management recently attended Machine Learning conference on ML for ITSM.

Machine learning looks prospective to improve ITSM processes through  
prediction and automation. They came up with 4 key areas, where ML can  
help ITSM process in ABC Tech.

1. Predicting High Priority Tickets: To predict priority 1 & 2 tickets, so  
that they can take preventive measures or fix the problem before  
it surfaces.

2. Forecast the incident volume in different fields, quarterly and  
annual. So that they can be better prepared with resources and technology planning.

3. Auto tag the tickets with right priorities and right departments so  
that reassigning and related delay can be reduced.

4. Predict RFC (Request for change) and possible failure / misconfiguration of ITSM assets.

**Data Set Fields:**

Total of about 46k records from year 2012,2013,2014  
Data needs to be queried from MYSQL data base (Read Only Access)  
Host: 18.136.56.185  
Port: 3306  
Username : dm\_team  
Password: dm\_team123#

|  |  |
| --- | --- |
| CI\_Name | SUB000508 |
| CI\_Cat | subapplication |
| CI\_Subcat | Web Based Application |
| WBS | WBS000162 |
| Incident\_ID | IM0000004 |
| Status | Closed |
| Impact | 4 |
| Urgency | 4 |
| Priority | 4 |
| Category | incident |
| KB\_number | KM0000553 |
| Alert\_Status | closed |
| No\_of\_Reassignments | 26 |
| Open\_Time | 05/02/2012 13:32:57 |
| Reopen\_Time |  |
| Resolved\_Time | 04/11/2013 13:50:27 |
| Close\_Time | 04/11/2013 13:51:17 |
| Handle\_Time\_hrs | 3871,691111 |
| Closure\_Code | Other |
| No\_of\_Related\_Interactions | 1 |
| Related\_Interaction | SD0000007 |
| No\_of\_Related\_Incidents | 2 |
| No\_of\_Related\_Changes | 1 |
| Related\_Change | C00000056 |

**CI\_Name** : Configuration Item Number

**CI\_Cat** : Configuration Item Category

**CI\_Subcat** : Configuration Item sub-category

The above three fields are provided by Configuration Management Database(CMDB) <https://en.wikipedia.org/wiki/Configuration_management_database>

**WBS** : Work breakdown structure - A work-breakdown structure in project management and systems engineering, is a deliverable-oriented breakdown of a project into smaller components. <https://en.wikipedia.org/wiki/Work_breakdown_structure>

**Incident\_ID** : Incident Identification

Who detected the incident and how? How soon was the incident detected after it occurred? Could the incident have been identified earlier? Could any tools or technologies have aided in the prompt or pre-emptive detection of the incident? <https://www.manageengine.com/products/service-desk/itil-incident-management-guide.html>

**Status** : Status of the item whether it is open or closed.

**Impact** : Impact is a measure of the effect of an incident, problem, or change on business processes. Impact is often based on how service levels will be affected.

**Urgency** : Urgency is a measure of how long it will be until an incident, problem, or change has a significant business impact. For example, a high impact incident may have low urgency if the impact will not affect the business until the end of the financial year.

**Priority** : Priority is a category that identifies the relative importance of an incident, problem, or change. Priority is based on impact and urgency, and it identifies required times for actions to be taken. Impact and urgency are used to assign priority.

<https://docs.bmc.com/docs/display/public/rondsubscriber/Impact%2C+urgency%2C+and+priority+criteria#:~:text=Impact%20is%20a%20measure%20of,has%20a%20significant%20business%20impact>.

**Category** : Whether the ticket is incident or request for information

**KB\_number** : Knowledge base number

**Alert\_Status** : Status of the ticket whether it is closed or not closed

**No\_of\_Reassignments** : Number of reassigned tickets

**Open\_Time** : Time of opening

**Reopen\_Time** : Time of re-opening

**Resolved\_Time** : Time of resolving

**Close\_Time** : Time of closing

**Handle\_Time\_hrs** : Time of handling the incidents

**Closure\_Code** : Code of closure

**No\_of\_Related\_Interactions** : Number of related interactions

**Related\_Interaction** :Use of a separate Interaction Management process allows for a separation of Incident metrics from Requests for Information.

**No\_of\_Related\_Incidents** :Number of related incidents

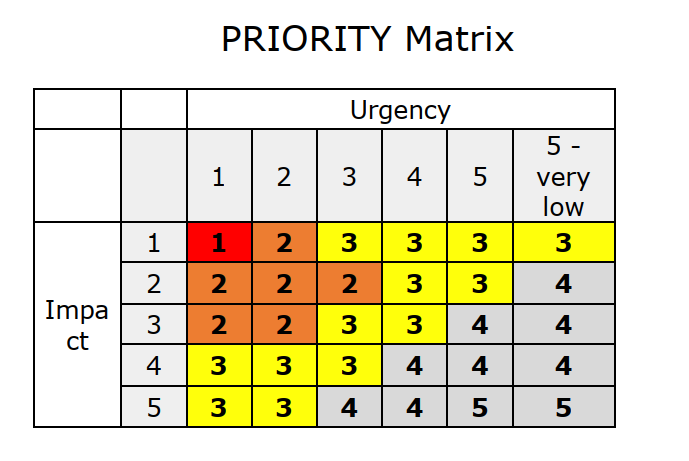
**No\_of\_Related\_Changes** : Number of related changes

**Related\_Change** :A change request is a document containing a call for an adjustment of a system.

The goal of this project is to improve the ITSM process.

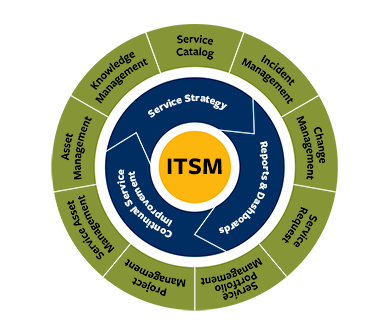
**5. Features**

The feature of the datasets were provided by ***Datamites*** company.



## ITSM Description

IT service management (ITSM) refers to the entirety of activities – directed by policies, organized and structured in processes and supporting procedures – that are performed by an organization to design, plan, deliver, operate and control information technology (IT) services offered to customers. <https://en.wikipedia.org/wiki/IT_service_management>.



**Figure 1: ITSM diagram**

**Assumptions**:

* Used forward fill and backward fill to fill in the null values.
* For the first part drop the following fields: 'Urgency','Impact','Alert\_Status','No\_of\_Related\_Incidents','No\_of\_Related\_Changes'.
* For the first part use priority as target variable.
* For second part, used Open\_Time and Incident\_ID fields for forecasting incidents using ARIMA and SARIMA
* Used Rolling forecast on ARIMA and SARIMA model to improve the predicted data.
* Used more data in train dataset than the test dataset in forecasting incidents.
* For the third part, drop the following fields: 'Urgency','Impact','Alert\_Status','Open\_Time','Reopen\_Time','Close\_Time','Resolved\_Time'
* Used No\_of\_Reassignments as target variable for finding the tickets with right priorities and right departments.
* For the fourth part, drop the following fields: 'Urgency','Impact','Alert\_Status','No\_of\_Related\_Incidents','Status','Open\_Time','Reopen\_Time','Close\_Time','Resolved\_Time
* Used No\_of\_Related\_Changes as target variable for predicting RFC (Request for change) and possible failure misconfiguration of ITSM assets.

**Import the dataset from the server**

!pip install sqlalchemy

!pip install pymysql

!pip install --upgrade pipb

!pip install imblearn

from sqlalchemy import create\_engine

import pandas as pd

db\_host= '18.136.56.185:3306'

username = 'dm\_team'

user\_pass= 'dm\_team123#'

db\_name='project\_itsm'

conn=create\_engine('mysql+pymysql://dm\_team:dm\_team123#@18.136.56.185:3306/project\_itsm')

conn.table\_names()

query = 'select \* from dataset\_list '

dataset\_list = pd.read\_sql(query,conn)

print(dataset\_list .shape)

dataset\_list

**Download the dataset in csv format**

dataset\_list.to\_csv('C:\\Users\DELL\Desktop\Rubixe projects\Mar2020\ITSM\_data.csv')

**Approach**:

1. **Predicting High Priority Tickets: To predict priority 1 & 2 tickets, so  
   that they can take preventive measures or fix the problem before  
   it surfaces.**
2. **Import the necessary packages**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import rcParams

%matplotlib inline

from collections import Counter

import warnings

warnings.filterwarnings("ignore")

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder,scale

from sklearn.metrics import

accuracy\_score,precision\_score,confusion\_matrix,classification\_report,f1\_score,recall\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.decomposition import PCA

1. **Load the dataset**

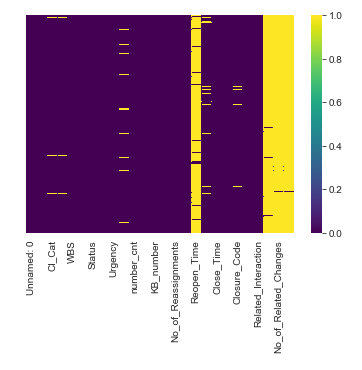
data\_parser=lambda c: pd.to\_Dataframe(c,format='%d/%m/%Y %H:%M:%s')

data=pd.read\_csv('C:\\Users\DELL\Desktop\Rubixe projects\Mar2020\ITSM\_data.csv', parse\_dates=['Open\_Time','Reopen\_Time','Close\_Time','Resolved\_Time'])

1. **Checking for outliers**

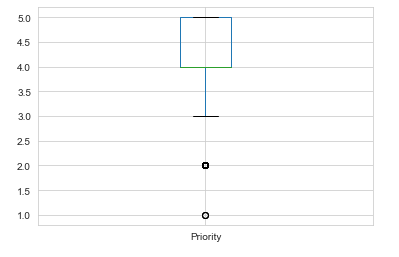
sns.set\_style('whitegrid')

sns.heatmap(data.isnull(),yticklabels=False,cbar=True,cmap='viridis')



**Figure 2: Heatmap to detect outliers in the data**

data[['Priority']].boxplot();



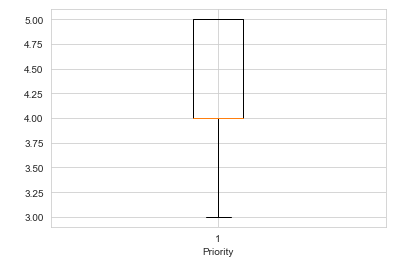
**Figure 3: Outliers using boxplot**

X=data.Priority

print(str(X[removed\_outliers\_Priority].size)+"/"+str(X.size)+" data points remain")

plt.boxplot(X[removed\_outliers\_Priority]);

plt.xlabel("Priority")



**Figure 4: Removed outliers**

figure,axis=plt.subplots(1,2,figsize=(16,5))

axis[0].boxplot(X);

axis[1].boxplot(X[removed\_outliers\_Priority]);

axis[0].set\_title("With outliers")

axis[0].set\_xlabel("Priority")

axis[1].set\_title("Removed outliers")

axis[1].set\_xlabel("Priority")



**Figure 5: Boxplots with outliers and after removing outliers**

data['clean\_Priority']=X[removed\_outliers\_Priority]

1. **Check for EDA**

data.info()

data.describe().T

data.shape

data.head()

data.isnull().sum().to\_frame().T

data.fillna(method='ffill',inplace=True)

data.isnull().sum().to\_frame().T

data.fillna(method='bfill',inplace=True)

data.isnull().sum().to\_frame()

Counter(data.clean\_Priority)

Drop the fields:

data\_new=data.drop(['Urgency','Impact','Alert\_Status','No\_of\_Related\_Incidents','No\_of\_Related\_Changes','Related\_Change','Reopen\_Time','Priority'],axis=1)

data\_new.head()

1. **Define X and y**

X=data\_new.iloc[:,data\_new.columns!='clean\_Priority']

y=data\_new.clean\_Priority

X.info()

y.describe().T

6) **Using Label Encoder**

enc=LabelEncoder()

X.CI\_Name=enc.fit\_transform(X.CI\_Name)

X.CI\_Cat=enc.fit\_transform(X.CI\_Cat)

X.CI\_Subcat=enc.fit\_transform(X.CI\_Subcat)

X.Status=enc.fit\_transform(X.Status)

X.Closure\_Code=enc.fit\_transform(X.Closure\_Code)

X.Category=enc.fit\_transform(X.Category)

X.WBS=enc.fit\_transform(X.WBS)

X.Incident\_ID=enc.fit\_transform(X.Incident\_ID)

X.Related\_Interaction=enc.fit\_transform(X.Related\_Interaction)

X.KB\_number=enc.fit\_transform(X.KB\_number)

X.Open\_Time=enc.fit\_transform(X.Open\_Time)

X.Resolved\_Time=enc.fit\_transform(X.Resolved\_Time)

X.Close\_Time=enc.fit\_transform(X.Close\_Time)

X.Handle\_Time\_hrs=enc.fit\_transform(X.Handle\_Time\_hrs)

X.number\_cnt=enc.fit\_transform(X.number\_cnt)

X.No\_of\_Related\_Interactions=enc.fit\_transform(X.No\_of\_Related\_Interactions)

X.No\_of\_Reassignments=enc.fit\_transform(X.No\_of\_Reassignments)

X.head()

X.info()

1. **Train-test split**

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=20,test\_size=0.3) print("X\_train shape = ",X\_train.shape)

print("X\_test shape = ",X\_test.shape)

print("y\_train shape = ",y\_train.shape)

print("y\_test shape = ",y\_test.shape)

1. **Random-Forest Classifier**

Define the model

model=RandomForestClassifier(n\_estimators=250,random\_state=10,criterion='gini')

model.fit(X\_train,y\_train)

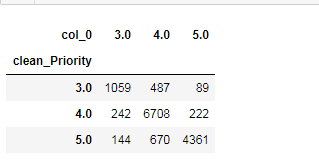
print(model.feature\_importances\_)

pd.DataFrame(model.feature\_importances\_,index=X.columns).sort\_values(0,ascending=False)

y\_train\_predict=model.predict(X\_train)

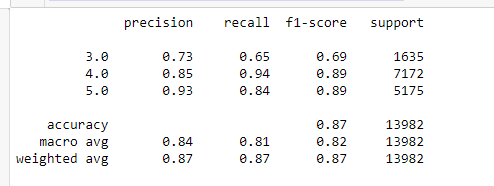
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 6: Confusion matrix using Random Forest classifier**

print(classification\_report(y\_test,y\_predict))



**Figure 7: Classification report using Random Forest classifier**

**8)** **PCA**

pca=PCA()

X=pd.DataFrame(pca.fit\_transform(X))

X.head()

pca.explained\_variance\_

pca.explained\_variance\_ratio\_

Follow the same procedure like define x and y, label encoder, train-test split, as in Random-Forest for PCA, XGBoost, ANN, KNN, Logistic regression

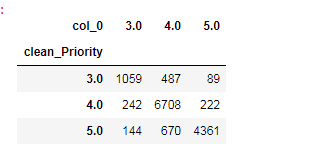
model=RandomForestClassifier(n\_estimators=250,random\_state=10,criterion='gini')

model.fit(X\_train,y\_train)

print(model.feature\_importances\_)

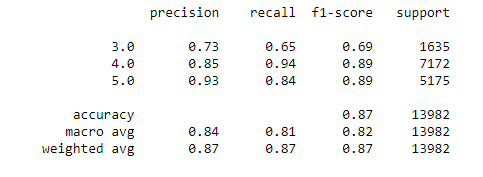
pd.DataFrame(model.feature\_importances\_,index=X.columns).sort\_values(0,ascending=False)

pd.crosstab(y\_test,y\_predict)



**Figure 8: Confusion matrix using PCA**

print(classification\_report(y\_test,y\_predict))

****

**Figure 9: Classification report using PCA**

**9) XGBoost**

from xgboost import XGBClassifier

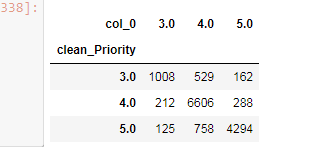
model=XGBClassifier(learning\_rate=0.5,random\_state=5,n\_estimators=50)

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

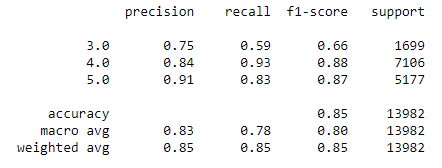
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 10: Confusion matrix using XGBoost**

print(classification\_report(y\_test,y\_predict))



**Figure 11: Classification report using XGBoost**

**10)** **ANN**

from sklearn.neural\_network import MLPClassifier

Standardize the dataset along X--axis

X=scale(X)

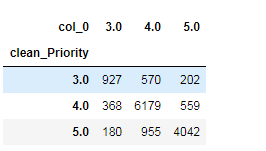
model=MLPClassifier(hidden\_layer\_sizes=(55,67,50),random\_state=10)

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

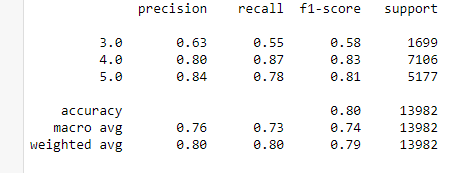
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 12: Confusion matrix using ANN**

print(classification\_report(y\_test,y\_predict))



**Figure 13: Classification report using ANN**

**11) KNN**

from sklearn.neighbors import KNeighborsClassifier

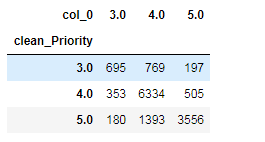
model= KNeighborsClassifier(n\_neighbors=10)

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

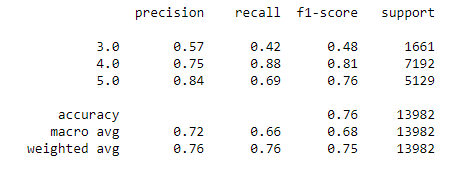
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 14: Confusion matrix using KNN**

print(classification\_report(y\_test,y\_predict))



**Figure 15: Classification report using KNN**

**12)** **Logistic regression**

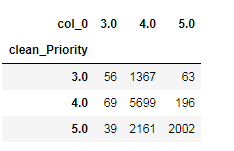
model=LogisticRegression()

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

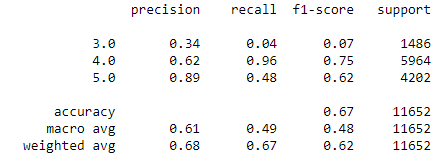
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 16: Confusion matrix using Logistic Regression**

print(classification\_report(y\_test,y\_predict))



**Figure 17: Classification report using Logistic Regression**

1. **Forecast the incident volume in different fields, quarterly and  
   annual. So that they can be better prepared with resources and technology planning.**
2. **Check for EDA**

Counter(data.Priority)

data.isnull().sum().to\_frame().T

# Create a new Dataframe containing Open\_Time as parameter

data\_new=data.groupby(data.Open\_Time).sum()

data\_new

data\_new.info()

data\_new.head().T

data\_new=data.loc[:,['Open\_Time','Incident\_ID','CI\_Name','CI\_Cat','CI\_Subcat','WBS','Category','Priority']]

data\_new

data\_new.dropna(inplace=True)

data\_new.isna().sum().to\_frame().T

data\_new['Open\_Date']=data\_new['Open\_Time'].apply(lambda x:x.date())

data\_new.Open\_Date

data\_new['No\_Incidents']=data\_new.groupby('Open\_Date')['Incident\_ID'].transform('count')

data\_new.head()

data\_new.corr()

incidents=data\_new.loc[:,['Open\_Date','No\_Incidents']]

incidents.head()

incidents.drop\_duplicates(inplace=True)

incidents.shape

incidents.head()

incidents=incidents.set\_index('Open\_Date')

incidents.index=pd.to\_datetime(incidents.index)

incidents.index

print(incidents.index.min(),'to',incidents.index.max())

data=incidents['No\_Incidents']

data=incidents.asfreq('D')

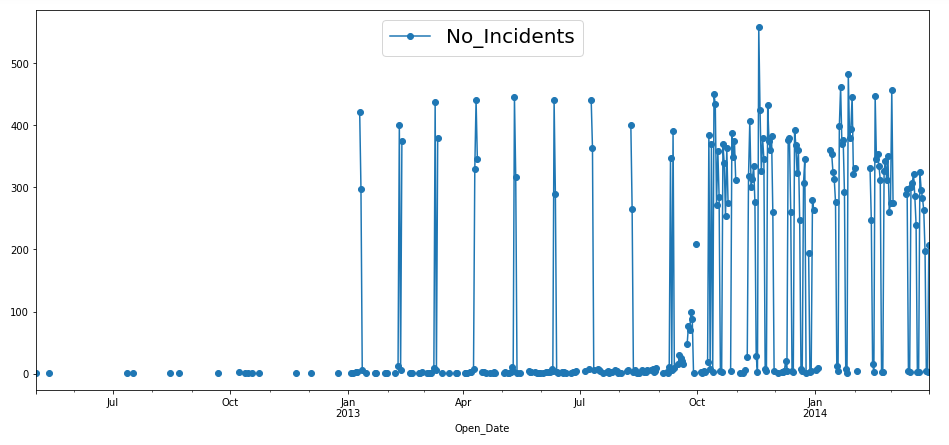
data.index

incidents.head()

data.plot(figsize=(16,7),marker='o')

plt.legend(loc='upper center',frameon=True, labelspacing=1,fontsize=20)

plt.show()



**Figure 18: Line plot for Open\_Date v/s No\_of\_Incidents**

infrom2013=incidents[incidents.index>dt.datetime(2013,10,1)]

infrom2013

infrom2014=incidents[incidents.index>dt.datetime(2014,10,1)]

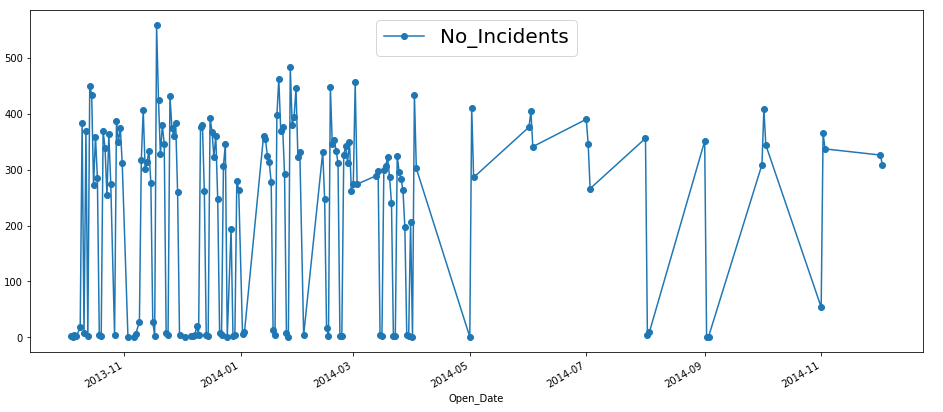
infrom2014

data=infrom2013['No\_Incidents']

data.plot(figsize=(16,7),marker='o')

plt.legend(loc='upper center',frameon=True, labelspacing=1,fontsize=20)

plt.show()



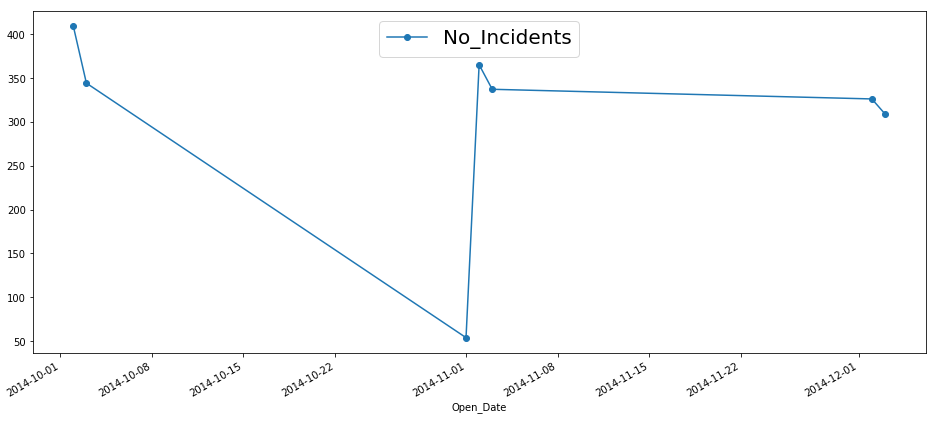
**Figure 19: Line plot for Open\_Date v/s No\_of\_Incidents from 2013**

data=infrom2014['No\_Incidents']

data.plot(figsize=(16,7),marker='o')

plt.legend(loc='upper center',frameon=True, labelspacing=1,fontsize=20)

plt.show()



**Figure 20: Line plot for Open\_Date v/s No\_of\_Incidents from 2014**

data=data.asfreq('D')

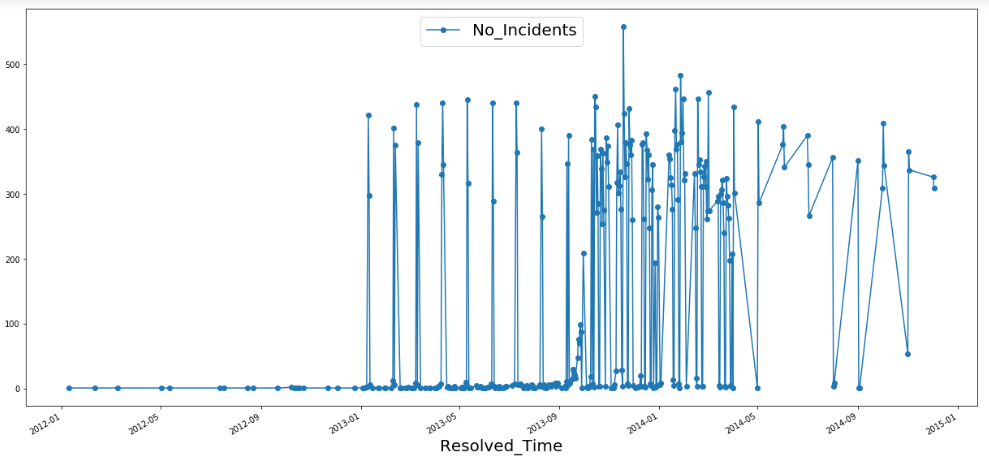
data.index

1. **Time series Forecasting**

incidents.plot(figsize=(21,10),marker='o')

plt.legend(loc='upper center',frameon=True, labelspacing=1,fontsize=20)

plt.xlabel(xlabel='Resolved\_Time',fontdict={'fontsize':20})

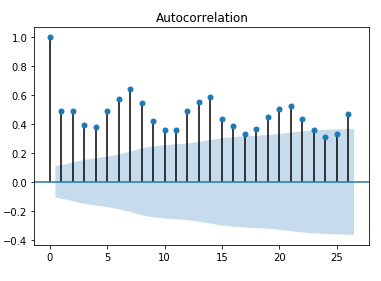


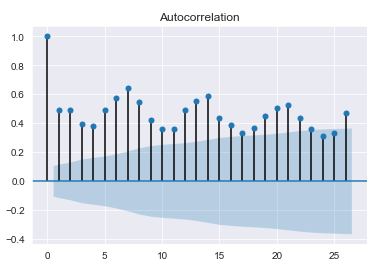
**Figure 21: Time series forecasting line plot for Resolved\_Time v/s No\_of\_Incidents**

1. **ACF and PACF**

from statsmodels.graphics.tsaplots import plot\_acf

plot\_acf(incidents)

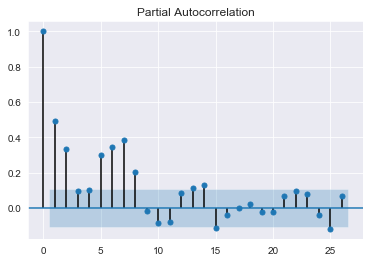


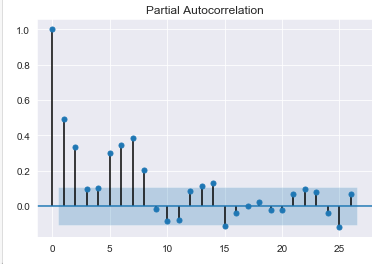


**Figure 22: ACF plot for incidents**

from statsmodels.graphics.tsaplots import plot\_pacf

plot\_pacf(incidents)





**Figure 23: PACF plot for incidents**

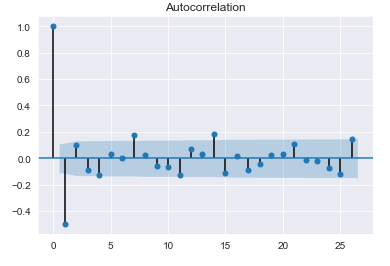
1. **Converting to stationary**

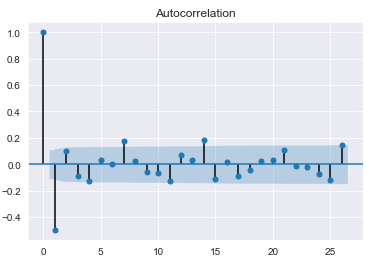
incidents\_diff=incidents.diff(periods=1)

incidents\_diff=incidents\_diff[1:]

incidents\_diff.head()

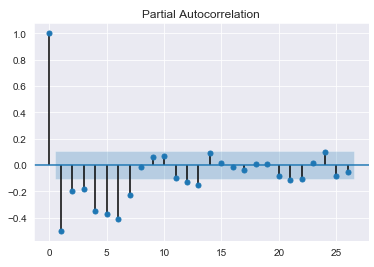
plot\_acf(incidents\_diff)

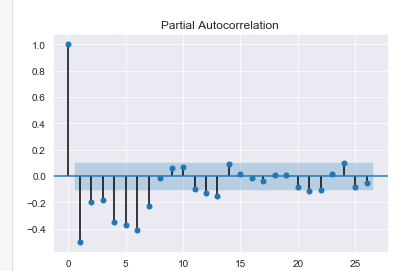




**Figure 24: ACF plot for incidents after converting data to stationary**

plot\_pacf(incidents\_diff)





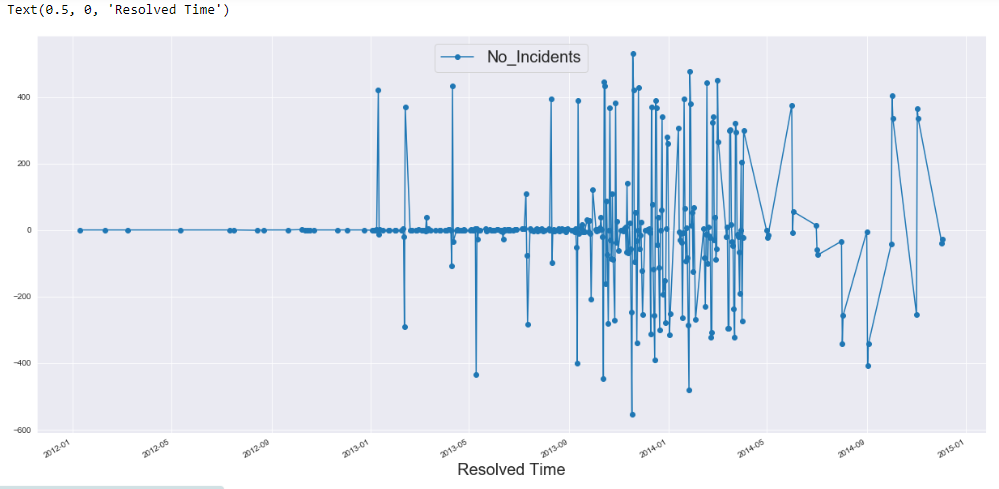
**Figure 25: PACF plot for incidents after converting data to stationary**

incidents\_diff.plot(figsize=(21,10),marker='o')

sns.set\_style("darkgrid")

plt.legend(loc='upper center',frameon=True, labelspacing=1,fontsize=20)

plt.xlabel(xlabel='Resolved Time',fontdict={'fontsize':20})



**Figure 26: Time series forecasting line plot for Resolved\_Time v/s No\_of\_Incidents for stationary data**

incidents\_diff.shape

incidents\_diff.head()

incidents\_diff.describe()

incidents\_diff.info()

incidents\_diff.isna().sum().to\_frame().T

## Auto Regressive Integrated Moving Average(ARIMA) Model

## X=incidents\_diff.values

## train=X[:300]

## test=X[301:]

## predictions=[]

## endog=train

from statsmodels.tsa.arima\_model import ARIMA

from sklearn.metrics import mean\_squared\_error

model\_arima=ARIMA(train,order=(10,1,1))

model\_arima\_fit=model\_arima.fit()

print(model\_arima\_fit.aic)

predictions = model\_arima\_fit.forecast(steps=24,alpha=0.05)[0]

predictions

plt.figure(figsize=(16,5))

sns.set\_style("darkgrid")

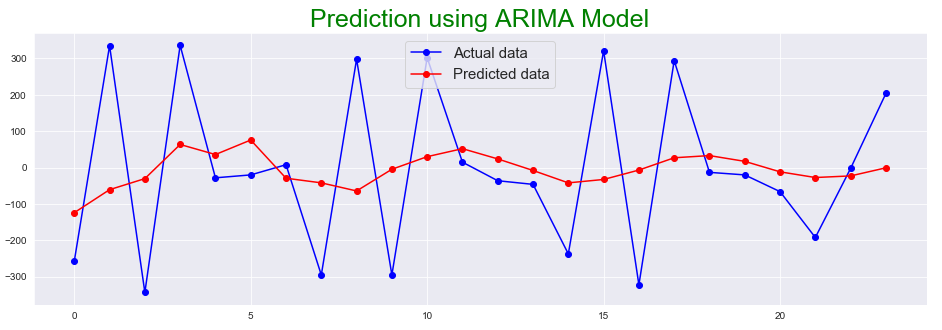
plt.plot(test,color='blue',label='Actual data',marker='o')

plt.plot(predictions,color='red',label="Predicted data",marker='o')

plt.legend(loc='upper center',frameon=True,fontsize=15)

plt.title('Prediction using ARIMA Model',fontdict={'fontsize':25},color='green')

plt.show()



**Figure 27: Prediction using ARIMA model**

mean\_squared\_error(test,predictions)

np.sqrt(mean\_squared\_error(test,predictions))

import itertools

p=d=q=range(0,2)

pdq=list(itertools.product(p,d,q))

pdq

import warnings

warnings.filterwarnings('ignore')

for param in pdq:

try:

model\_arima=ARIMA(train,order=param)

model\_arima\_fit=model\_arima.fit()

print(param,model\_arima\_fit.aic)

except:

continue

1. **Rolling Forecast on ARIMA**

Using ARIMA model, the plot for predicted data was deviating from the actual data at many points. So I had used Rolling forcast technique to correct the graph for the predicted data.

A rolling window model involves calculating a statistic on a fixed contiguous block of prior observations and using it as a forecast.

It is much like the expanding window, but the window size remains fixed and counts backwards from the most recent observation.

It may be more useful on time series problems where recent lag values are more predictive than older lag values.

history = [x for x in train]

predictions = list()

for t in range(len(test)):

model = ARIMA(history, order=(4,2,0))

model\_fit = model.fit()

output = model\_fit.forecast()

y\_predict= output[0]

predictions.append(y\_predict)

observation = test[t]

history.append(observation)

print('predicted=%f, expected=%f' % (y\_predict, observation))

error = mean\_squared\_error(test, predictions)

print('Test MSE: %.3f' % error)

print('Square root of Test MSE: %.3f' %np.sqrt(error))

plt.figure(figsize=(16,7))

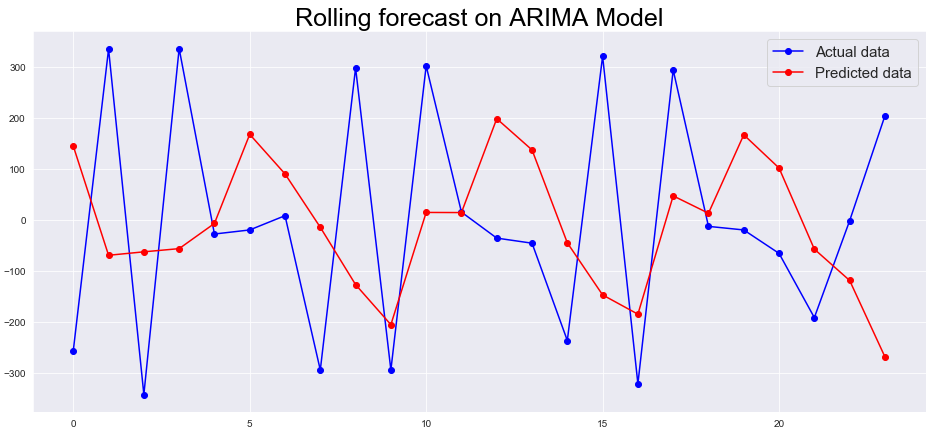
plt.plot(test,color='blue',label='Actual data',marker='o')

plt.plot(predictions, color='red',label='Predicted data',marker='o')

plt.legend(loc='upper right',frameon=True,fontsize=15)

plt.title('Rolling forecast on ARIMA Model',fontdict={'fontsize':25},color='black')

plt.show()



**Figure 28: Rolling Forecast on ARIMA model**

## Seasonal Auto Regressive Integrated Moving Average(SARIMA) Model

from statsmodels.tsa.statespace.sarimax import SARIMAX

from sklearn.metrics import mean\_squared\_error

model\_sarimax = SARIMAX(train,order=(2,2,1), seasonal\_order=(2,2,1,2),mle\_regression=True,

enforce\_stationarity=True,enforce\_invertibility=True)

print(model\_sarimax)

model\_sarimax\_fit = model\_sarimax.fit()

print(model\_sarimax\_fit.aic)plt.figure(figsize=(16,5))

sns.set\_style("darkgrid")

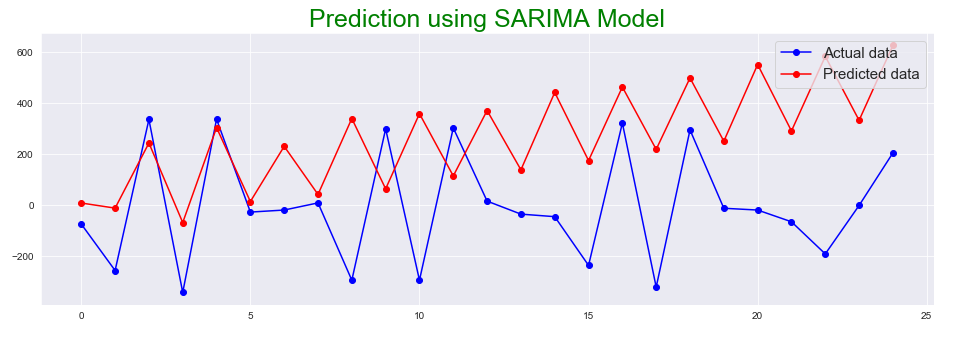
plt.plot(test,color='blue',label='Actual data',marker='o')

plt.plot(predictions,color='red',label="Predicted data",marker='o')

plt.legend(loc='upper right',frameon=True,fontsize=15)

plt.title('Prediction using SARIMA Model',fontdict={'fontsize':25},color='green')

plt.show()



**Figure 29: Prediction using SARIMA model**

mean\_squared\_error(test,predictions)

np.sqrt(mean\_squared\_error(test,predictions))

import itertools

p=d=q=s=range(0,2)

pdqs=list(itertools.product(p,d,q,s))

pdqs

import warnings

warnings.filterwarnings('ignore')

for a in pdqs:

try:

model\_sarimax=SARIMAX(train,order=a,seasonal\_order=a)

model\_sarimax\_fit=model\_sarimax.fit()

print(a,model\_sarimax\_fit.aic)

print(model\_fit.summary())

except:

continue

**Rolling forecast on SARIMA**

history = [x for x in train]

predictions = list()

for t in range(len(test)):

model = SARIMAX(history, order=(2,1,1),seasonal\_order=(1,1,1,1))

model\_fit = model.fit()

output = model\_fit.forecast()

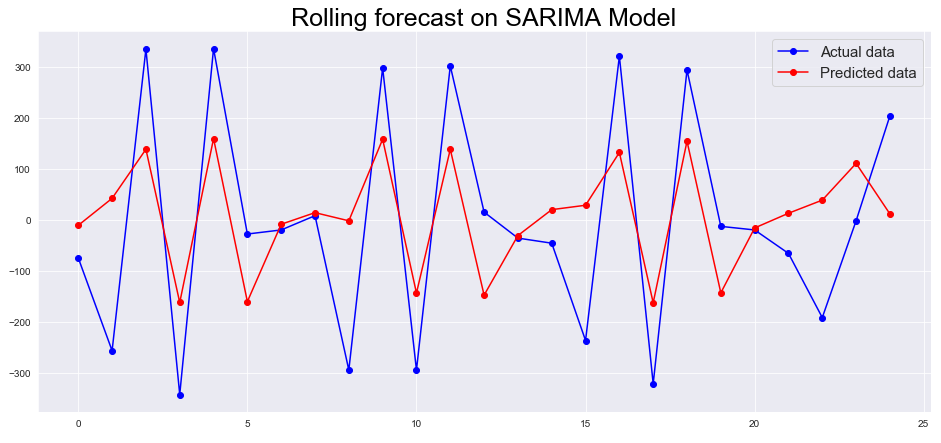
y\_predict= output[0]

predictions.append(y\_predict)

observation = test[t]

history.append(observation)

print('predicted=%f, expected=%f' % (y\_predict, observation))

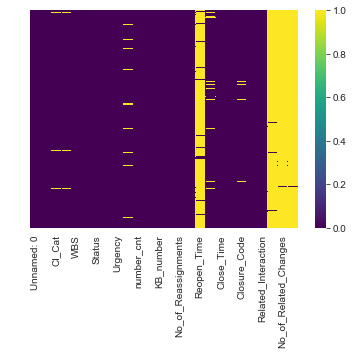


**Figure 30: Rolling Forecast on SARIMA model**

1. **Auto tag the tickets with right priorities and right departments so  
   that reassigning and related delay can be reduced.**
2. **Checking for outliers**

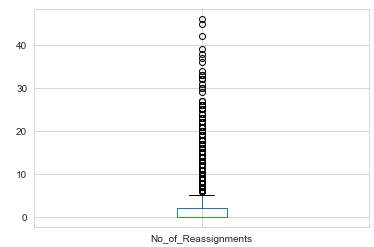
sns.set\_style('whitegrid')

sns.heatmap(data.isnull(),yticklabels=False,cbar=True,cmap='viridis')



**Figure 31: Heatmap for detecting outliers**

data[['No\_of\_Reassignments']].boxplot();



**Figure 32: Boxplot to detect outliers**

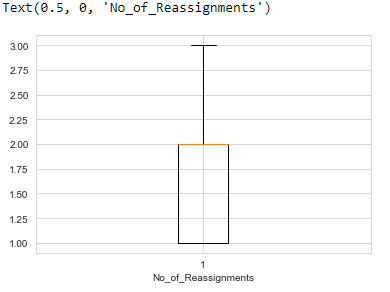
X=data.No\_of\_Reassignments

removed\_outliers\_Reassignments=X.between(X.quantile(0.7),X.quantile(0.9))

print(str(X[removed\_outliers\_Reassignments].size)+"/"+str(X.size)+" data points remain")

plt.boxplot(X[removed\_outliers\_Reassignments]);

plt.xlabel("No\_of\_Reassignments")



**Figure 33: Boxplot to remove outliers**

figure,axis=plt.subplots(1,2,figsize=(16,5))

axis[0].boxplot(X);

axis[1].boxplot(X[removed\_outliers\_Reassignments]);

axis[0].set\_title("With outliers")

axis[0].set\_xlabel("No\_of\_Reassignments")

axis[1].set\_title("Removed outliers")

axis[1].set\_xlabel("No\_of\_Reassignments")



**Figure 33: Boxplot representing data with outliers and after removing outliers**

data['clean\_Reassignments']=X[removed\_outliers\_Reassignments]

1. **Check for EDA**

data=data.drop(['Urgency','Impact','Alert\_Status','Open\_Time','Reopen\_Time','Close\_Time','Resolved\_Time','No\_of\_Reassignments'],axis=1)

Counter(data.clean\_Reassignments).most\_common()

data.head()

dataset=data.loc[:,['CI\_Cat','CI\_Subcat','WBS','Category']]

dataset.head()

dataset.shape

dataset.dropna(inplace=True)

dataset.isna().sum().to\_frame().T

dataset.info()

dataset.describe()

Create a new field tickets

dataset.loc[data.clean\_Reassignments>4.0,'tickets']='high'

dataset.loc[(data.clean\_Reassignments>2.0)& (data.clean\_Reassignments<=4.0),'tickets']='medium'

dataset.loc[data.clean\_Reassignments<=2.0,'tickets']='low'

dataset.head(34)

1. **Define X and y**

X=dataset.loc[:,dataset.columns!='tickets']

y=dataset.tickets

X.head()

y.fillna(method='ffill',inplace=True)

y.fillna(method='bfill',inplace=True)

y.isna().sum()

4) **Using Label Encoder**

enc=LabelEncoder()

X.CI\_Cat=enc.fit\_transform(X.CI\_Cat)

X.CI\_Subcat=enc.fit\_transform(X.CI\_Subcat)

X.WBS=enc.fit\_transform(X.WBS)

X.Category=enc.fit\_transform(X.Category)

X.head()

X.info()

1. **Using train-test split**

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=20,test\_size=0.3)

print("X\_train shape = ",X\_train.shape)

print("X\_test shape = ",X\_test.shape)

print("y\_train shape = ",y\_train.shape)

print("y\_test shape = ",y\_test.shape)

1. **Random-forest classifier**

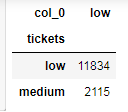
model=RandomForestClassifier(n\_estimators=300,random\_state=10,max\_depth=4,criterion='gini')

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

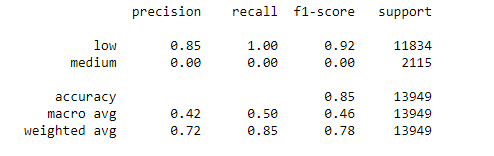
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 34: Confusion matrix using Random-Forest Classifier**

print(classification\_report(y\_test,y\_predict))



**Figure 35: Classification report using Random-Forest Classifier**

1. **XGBoost**

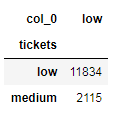
model=XGBClassifier(max\_depth=3,learning\_rate=0.5,random\_state=50,n\_estimators=50)

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

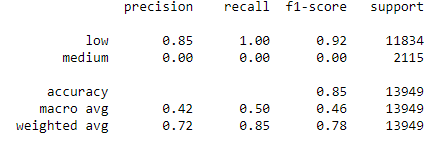
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 36: Confusion matrix using XGBoost**

print(classification\_report(y\_test,y\_predict))



**Figure 37: Classification report using XGBoost**

1. **Predict RFC (Request for change) and possible failure / misconfiguration of ITSM assets.**
2. **Import the necessary packages**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import rcParams

%matplotlib inline

from collections import Counter

import warnings

warnings.filterwarnings("ignore")

from sklearn.preprocessing import LabelEncoder,scale

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score,precision\_score,confusion\_matrix,classification\_report,f1\_score,recall\_score

import datetime as dt

from scipy import stats

import pandas.util.testing as tm

1. **Load the dataset**

data\_parser=lambda c: pd.to\_Dataframe(c,format='%d/%m/%Y %H:%M:%s')

data=pd.read\_csv('C:\\Users\DELL\Desktop\Rubixe projects\Mar2020\ITSM\_data.csv', parse\_dates=['Open\_Time','Reopen\_Time','Close\_Time','Resolved\_Time'])

data.head()

data.shape

data.info()

data.describe()

1. **Check for EDA**

data=data.drop(['Urgency','Impact','Alert\_Status','No\_of\_Related\_Incidents','Status','Open\_Time','Reopen\_Time','Close\_Time','Resolved\_Time'],axis=1)

data=data.iloc[:,:]

data.head()

data.shape

data.isna().sum().to\_frame().T

data['No\_of\_Related\_Changes'].fillna(method='ffill',inplace=True)

data['No\_of\_Related\_Changes'].fillna(method='bfill',inplace=True)

data['Related\_Change'].fillna(method='ffill',inplace=True)

data['Related\_Change'].fillna(method='bfill',inplace=True)

data.isna().sum().to\_frame().T

data.dropna(inplace=True,how='any')

data.info()

data.describe()

1. **Define X and y**

X=data.loc[:,data.columns!='No\_of\_Related\_Changes']

y=data.No\_of\_Related\_Changes

X.info()

1. **Using Label Encoder**

enc=LabelEncoder()

X.CI\_Name=enc.fit\_transform(X.CI\_Name)

X.CI\_Cat=enc.fit\_transform(X.CI\_Cat)

X.Incident\_ID=enc.fit\_transform(X.Incident\_ID)

X.CI\_Subcat=enc.fit\_transform(X.CI\_Subcat)

X.WBS=enc.fit\_transform(X.WBS)

X.Category=enc.fit\_transform(X.Category)

X.KB\_number=enc.fit\_transform(X.KB\_number)

X.Related\_Interaction=enc.fit\_transform(X.Related\_Interaction)

X.Handle\_Time\_hrs=enc.fit\_transform(X.Handle\_Time\_hrs)

X.Closure\_Code=enc.fit\_transform(X.Closure\_Code)

X.Related\_Change=enc.fit\_transform(X.Related\_Change)

X.head()

X.info()

**6)**  **Using train-test split**

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=20,test\_size=0.3)

print("X\_train shape = ",X\_train.shape)

print("X\_test shape = ",X\_test.shape)

print("y\_train shape = ",y\_train.shape)

print("y\_test shape = ",y\_test.shape)

**7)** **Random-Forest classifier**

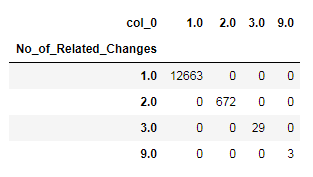
model=RandomForestClassifier(n\_estimators=250,random\_state=10,max\_depth=4)

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

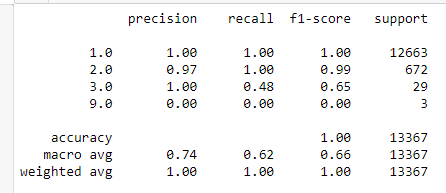
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 38: Confusion matrix using Random-Forest Classifier**

print(classification\_report(y\_test,y\_predict))



**Figure 39: Classification report using Random-Forest Classifier**

**8) XGBoost**

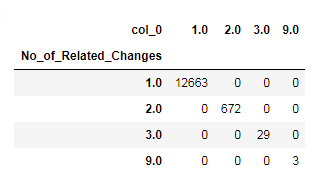
model=XGBClassifier(max\_depth=3,learning\_rate=0.5,random\_state=50,n\_estimators=50)

model.fit(X\_train,y\_train)

y\_train\_predict=model.predict(X\_train)

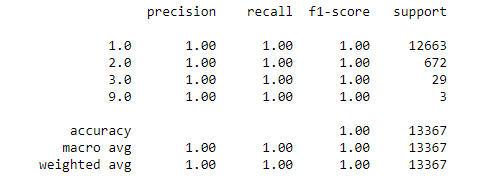
y\_predict=model.predict(X\_test)

pd.crosstab(y\_test,y\_predict)



**Figure 40: Confusion matrix using XGBoost**

print(classification\_report(y\_test,y\_predict))



**Figure 41: Classification report using XGBoost**