DATA SCIENCE PROJECT

DATE : 18 - AUG - 2024

Client: **ABC Tech** | Category: **ITSM - ML**

Project Ref: PM-PR-0012

Team Members :

Dipanjali Patra

Sandeep Chandra Sagar

Vinay C

ITSM Improvement through Machine Learning: Enhancing Incident Management at ABC Tech.

Problem Statement

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 the 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.

Business Case Description

ABC Tech is an established mid-sized organization operating in the IT-enabled business sector for over a decade. They manage a significant volume of IT incidents and tickets, averaging between 22,000 to 25,000 per year. ABC Tech follows best practices in **IT Service Management (ITSM),** including incident management, problem management, change management, and configuration management processes. These ITIL practices have matured over time, reaching a high level of process maturity.

Recently, ABC Tech conducted an audit that indicated that further improvement initiatives in their ITSM processes may not provide a sufficient return on investment (ROI). Despite their mature processes, customer feedback from recent surveys has revealed that incident management, in particular, is rated poorly, suggesting there is room for enhancement.

In response to these challenges, ABC Tech's management has decided to **explore the potential of machine learning (ML) to enhance their ITSM processes**. After attending a Machine Learning conference focused on IT Service Management (ITSM), they identified four key areas where ML can contribute to improving ITSM processes within the organization.

Tasks

1. Predicting High Priority Tickets: ABC Tech aims to develop an ML model that can predict high-priority tickets, specifically those categorized as priority 1 and 2. This prediction will allow them to take proactive measures to address issues or incidents before they escalate.

2. Forecasting Incident Volume: The organization plans to use ML to forecast the incident volume in different fields on a quarterly and annual basis. This predictive capability will help them better allocate resources and plan for the required technology upgrades.

3. Auto-Tagging Tickets: ABC Tech intends to implement a text classification ML model to automatically assign correct priorities and departments to incoming tickets. This automation will reduce reassignment and related delays in ticket handling.

4. Predicting RFC and ITSM Asset Misconfigurations: The organization aims to create predictive models for Request for Change (RFC) and detect potential failures or misconfigurations in ITSM assets. Identifying these issues in advance will help in preventing disruptions and improving overall ITSM asset management.

[The dataset that ABC Tech plans to use for these ML initiatives comprises a total of approximately 46,000 records spanning the years 2012, 2013, and 2014. The data is stored in a MySQL database with read-only access, and the relevant connection details are provided]

Summary of Dataset Fields

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

Priority Matrix

		Urgency					
		1	2	3	4	5	5 - very low
Impact	1	**1**	**2**	**3**	**3**	**3**	**3**
	2	**2**	**2**	**2**	**3**	**3**	**4**
	3	**2**	**2**	**3**	**3**	**4**	**4**
	4	**3**	**3**	**3**	**4**	**4**	**4**
	5	**3**	**3**	**4**	**4**	**5**	**5**

Data Extraction from SQL DB

Installing pyMySQL and mysql_connector to connect to DB

```
!pip install pymysql
!pip install mysql_connector
```

```
Collecting pymysql
  Downloading PyMySQL-1.1.1-py3-none-any.whl.metadata (4.4 kB)
Downloading PyMySQL-1.1.1-py3-none-any.whl (44 kB)
   ---------------------------------------- 45.0/45.0 kB 1.9 MB/s eta 0:00:00
Installing collected packages: pymysql
Successfully installed pymysql-1.1.1
Collecting mysql_connector
  Downloading mysql-connector-2.2.9.tar.gz (11.9 MB)
     ---------------------------------------- 11.9/11.9 MB 44.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: mysql_connector
  Building wheel for mysql_connector (setup.py) ... done
  Created wheel for mysql_connector: filename=mysql_connector-2.2.9-cp310-cp310-linux_x86_64.whl size=247948 sha256=dc4080de4f3
  Stored in directory: /root/.cache/pip/wheels/76/48/9b/da67ff1a18fe8e9d428f9b1a177716d4a7d363d2bbe83bf6cf
Successfully built mysql_connector
Installing collected packages: mysql_connector
Successfully installed mysql_connector-2.2.9
```

Importing necessary libraries and creating connection to server

```
[2] import mysql.connector
    import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    %matplotlib inline
    import warnings
    warnings.filterwarnings("ignore")
```

```
[3] #making connection to db server
    connection = mysql.connector.connect(host='18.136.157.135',
                                         user = 'dm_team',
                                         password = 'DM!$Team@&27920!',
                                         database = 'project_itsm' )
```

Once connected to Server, checking the databases and tables present in it.

```
[4] #check no.of db connected/available on server
    cursor = connection.cursor()
    cursor.execute('show databases')
    for i in cursor:
        print(i)
```

```
('information_schema',)
('project_itsm',)
```

```
[5] db_tables=pd.read_sql_query('show tables',connection)
    print(db_tables)
```

```
  Tables_in_project_itsm
0           dataset_list
```

Projecting all columns in the table - 'dataset_list'

```
[6] query = "select * from dataset_list"
    data = pd.read_sql(query,connection)
    data
```

	CI_Name	CI_Cat	CI_Subcat	WBS	Incident_ID	Status	Impact	Urgency	Priority	number_cnt	...	Reopen_Time	Resolved_Time
0	SUB000508	subapplication	Web Based Application	WBS000162	IM0000004	Closed	4	4	4	0.601292279	...		04-11-2013 13:50
1	WBA000124	application	Web Based Application	WBS000088	IM0000005	Closed	3	3	3	0.415049969	...	02-12-2013 12:31	02-12-2013 12:36
2	DTA000024	application	Desktop Application	WBS000092	IM0000006	Closed	NS	3	NA	0.517551335	...		13-01-2014 15:12
3	WBA000124	application	Web Based Application	WBS000088	IM0000011	Closed	4	4	4	0.642927218	...		14-11-2013 09:31
4	WBA000124	application	Web Based Application	WBS000088	IM0000012	Closed	4	4	4	0.345258343	...		08-11-2013 13:55
...	...	...	...	...	...	...	...	...	...	...	...	...	...
46601	SBA000464	application	Server Based Application	WBS000073	IM0047053	Closed	4	4	4	0.23189604	...		31-03-2014 16:29
46602	SBA000461	application	Server Based Application	WBS000073	IM0047054	Closed	4	4	4	0.805153085	...		31-03-2014 15:29
46603	LAP000019	computer	Laptop	WBS000091	IM0047055	Closed	5	5	5	0.917466294	...		31-03-2014 15:32
46604	WBA000058	application	Web Based Application	WBS000073	IM0047056	Closed	4	4	4	0.701278158	...		31-03-2014 15:42
46605	DCE000077	hardware	DataCenterEquipment	WBS000267	IM0047057	Closed	3	3	3	0.902319509	...		31-03-2014 22:47

46606 rows × 25 columns

Domain Analysis:

Certainly! Domain analysis is an essential step in any data science or machine learning project. It involves gaining a deep understanding of the domain-specific aspects of the problem you're trying to solve. In this case, we're analyzing the domain of IT Service Management (ITSM) within the context of ABC Tech's business case. Here's a domain analysis for ITSM:

1. IT Service Management (ITSM):

Definition:

IT Service Management (ITSM) refers to a set of practices and processes used by organizations to design, deliver, manage, and improve IT services for their customers and end-users.

Importance:

ITSM ensures that IT services are aligned with business goals, reliable, and efficiently delivered, leading to enhanced customer satisfaction and business performance.

2. Incident Management:

Definition:

Incident Management is a core ITSM process that involves identifying, categorizing, prioritizing, and resolving incidents to restore normal service operations as quickly as possible.

Challenges:

Common challenges in incident management include handling a high volume of incidents, determining incident priorities, minimizing response times, and reducing the impact on end-users.

3. Priority in ITSM:

Definition:

Priority is a classification system used to categorize incidents based on their severity and impact on business operations. In ITIL (IT Infrastructure Library) framework, there are typically four priority levels: Priority 1 (Critical), Priority 2 (High), Priority 3 (Medium), and Priority 4 (Low).

Importance:

Prioritizing incidents helps organizations allocate resources effectively and respond to critical issues promptly.

4. ITIL Framework:

Definition:

ITIL is a widely adopted framework for ITSM that provides best practices and guidelines for managing IT services, including incident management, problem management, change management, and configuration management.

Maturity Levels:

ITIL processes can mature over time, starting from ad-hoc practices and progressing to well-defined, controlled, and optimized processes. A mature ITIL framework leads to improved service quality and efficiency.

5. Machine Learning in ITSM:

Application:

Machine learning can be applied to ITSM processes to predict incidents, automate ticket classification, forecast resource needs, and detect anomalies or misconfigurations in IT assets.

Benefits:

ML can enhance incident response, reduce manual workload, improve service quality, and proactively identify issues before they impact operations.

Basic Checks

Displaying the first 5 rows of table which is stored in 'data'

```
[7] data.head()# shows top 5 records of dataset.
```

	CI_Name	CI_Cat	CI_Subcat	WBS	Incident_ID	Status	Impact	Urgency	Priority	number_cnt	...	Reopen_Time	Resolved_Time	Close_Time
0	SUB000508	subapplication	Web Based Application	WBS000162	IM0000004	Closed	4	4	4	0.601292279	...		04-11-2013 13:50	04-11-2013 13:51
1	WBA000124	application	Web Based Application	WBS000088	IM0000005	Closed	3	3	3	0.415049969	...	02-12-2013 12:31	02-12-2013 12:36	02-12-2013 12:36
2	DTA000024	application	Desktop Application	WBS000092	IM0000006	Closed	NS	3	NA	0.517551335	...		13-01-2014 15:12	13-01-2014 15:13
3	WBA000124	application	Web Based Application	WBS000088	IM0000011	Closed	4	4	4	0.642927218	...		14-11-2013 09:31	14-11-2013 09:31
4	WBA000124	application	Web Based Application	WBS000088	IM0000012	Closed	4	4	4	0.345258343	...		08-11-2013 13:55	08-11-2013 13:55

5 rows × 25 columns

Displaying the last 5 rows of table which is stored in 'data'

```
[8] data.tail() # shows last 5 records of dataset.
```

	CI_Name	CI_Cat	CI_Subcat	WBS	Incident_ID	Status	Impact	Urgency	Priority	number_cnt	...	Reopen_Time	Resolved_Time
46601	SBA000464	application	Server Based Application	WBS000073	IM0047053	Closed	4	4	4	0.23189604	...		31-03-2014 16:29
46602	SBA000461	application	Server Based Application	WBS000073	IM0047054	Closed	4	4	4	0.805153085	...		31-03-2014 15:29
46603	LAP000019	computer	Laptop	WBS000091	IM0047055	Closed	5	5	5	0.917466294	...		31-03-2014 15:32
46604	WBA000058	application	Web Based Application	WBS000073	IM0047056	Closed	4	4	4	0.701278158	...		31-03-2014 15:42
46605	DCE000077	hardware	DataCenterEquipment	WBS000267	IM0047057	Closed	3	3	3	0.902319509	...		31-03-2014 22:47

5 rows × 25 columns

Basic information of the table

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46606 entries, 0 to 46605
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   CI_Name                     46606 non-null  object
 1   CI_Cat                      46606 non-null  object
 2   CI_Subcat                   46606 non-null  object
 3   WBS                         46606 non-null  object
 4   Incident_ID                 46606 non-null  object
 5   Status                      46606 non-null  object
 6   Impact                      46606 non-null  object
 7   Urgency                     46606 non-null  object
 8   Priority                    46606 non-null  object
 9   number_cnt                  46606 non-null  object
 10  Category                    46606 non-null  object
 11  KB_number                   46606 non-null  object
 12  Alert_Status                46606 non-null  object
 13  No_of_Reassignments         46606 non-null  object
 14  Open_Time                   46606 non-null  object
 15  Reopen_Time                 46606 non-null  object
 16  Resolved_Time               46606 non-null  object
 17  Close_Time                  46606 non-null  object
 18  Handle_Time_hrs             46606 non-null  object
 19  Closure_Code                46606 non-null  object
 20  No_of_Related_Interactions  46606 non-null  object
 21  Related_Interaction         46606 non-null  object
 22  No_of_Related_Incidents     46606 non-null  object
 23  No_of_Related_Changes       46606 non-null  object
 24  Related_Change              46606 non-null  object
dtypes: object(25)
memory usage: 8.9+ MB
```

Insights

- The table contains 46606 Rows with 25 Columns. [Can also be checked with - ' **data.shape()**']
- All the columns are of Object type.

Removing the columns that do not impact the target variable from the dataset

```
[20] # dropping some unnecessary columns which does not impact on ticket priority.
     data.drop(["CI_Name", "WBS", "Incident_ID", "KB_number", "Related_Interaction","Related_Change"], axis=1, inplace=True)
```

```
[21] data.shape
```

```
(46606, 19)
```

Converting the columns to corresponding Data Types

```
[23] # Define categorical,numerical and datatime  columns:
     categorical_columns = ['CI_Cat', 'CI_Subcat', 'Status','Impact', 'Urgency' ,'Category',
            'Alert_Status', 'Closure_Code']

     numerical_columns=['Priority', 'number_cnt', 'No_of_Reassignments',
                       'No_of_Related_Interactions', 'No_of_Related_Incidents',
            'No_of_Related_Changes']

     datetime_columns=['Open_Time', 'Reopen_Time', 'Resolved_Time',
            'Close_Time', 'Handle_Time_hrs',]

     # Convert categorical columns to 'object' dtype
     data[categorical_columns] = data[categorical_columns].astype('object')

     # Convert numerical columns to 'float' or 'int' dtype (coerce errors for invalid parsing)
     data[numerical_columns] = data[numerical_columns].apply(pd.to_numeric, errors='coerce')

     # Convert time columns to 'datetime' dtype
     data[datetime_columns]=data[datetime_columns].apply(pd.to_datetime,errors='coerce')
```

```
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   CI_Cat                      46606 non-null  object
 1   CI_Subcat                   46606 non-null  object
 2   Status                      46606 non-null  object
 3   Impact                      46606 non-null  object
 4   Urgency                     46606 non-null  object
 5   Priority                    45226 non-null  float64
 6   number_cnt                  46606 non-null  float64
 7   Category                    46606 non-null  object
 8   Alert_Status                46606 non-null  object
 9   No_of_Reassignments         46605 non-null  float64
 10  Open_Time                   18612 non-null  datetime64[ns]
 11  Reopen_Time                 871 non-null    datetime64[ns]
 12  Resolved_Time               17577 non-null  datetime64[ns]
 13  Close_Time                  18333 non-null  datetime64[ns]
 14  Handle_Time_hrs             490 non-null    datetime64[ns]
 15  Closure_Code                46606 non-null  object
 16  No_of_Related_Interactions  46492 non-null  float64
 17  No_of_Related_Incidents     1222 non-null   float64
 18  No_of_Related_Changes       560 non-null    float64
dtypes: datetime64[ns](5), float64(6), object(8)
```

Displaying the unique values in each column

```python
# Print unique values for each column:
for i in categorical_columns:
    print(i,data[i].unique())
    print(data[i].value_counts())
    print("----------------------------")
```

```
CI_Cat ['subapplication' 'application' 'computer' '' 'displaydevice' 'software'
 'storage' 'database' 'hardware' 'officeelectronics' 'networkcomponents'
 'applicationcomponent' 'Phone']
CI_Cat
application             32900
subapplication           7782
computer                 3643
storage                   703
hardware                  442
software                  333
database                  214
displaydevice             212
officeelectronics         152
                          111
networkcomponents         107
applicationcomponent        5
Phone                       2
Name: count, dtype: int64
----------------------------
```

```
Status ['Closed' 'Work in progress']
Status
Closed              46597
Work in progress        9
Name: count, dtype: int64
----------------------------
Impact ['4' '3' 'NS' '5' '2' '1']
Impact
4     22556
5     16741
3      5234
NS     1380
2       692
1         3
Name: count, dtype: int64
----------------------------
Urgency ['4' '3' '5' '2' '1' '5 - Very Low']
Urgency
4               22588
5               16779
3                6536
2                 696
1                   6
5 - Very Low        1
Name: count, dtype: int64
----------------------------
```

```
CI_Subcat ['Web Based Application' 'Desktop Application' 'Server Based Application'
 'SAP' 'Client Based Application' 'Citrix' 'Standard Application'
 'Windows Server' 'Laptop' 'Linux Server' '' 'Monitor'
 'Automation Software' 'SAN' 'Banking Device' 'Desktop' 'Database'
 'Oracle Server' 'Keyboard' 'Printer' 'Exchange' 'System Software' 'VDI'
 'Encryption' 'Omgeving' 'MigratieDummy' 'Scanner' 'Controller'
 'DataCenterEquipment' 'KVM Switches' 'Switch' 'Database Software'
 'Network Component' 'Unix Server' 'Lines' 'ESX Cluster' 'zOS Server'
 'SharePoint Farm' 'NonStop Server' 'Application Server'
 'Security Software' 'Thin Client' 'zOS Cluster' 'Router' 'VMWare'
 'Net Device' 'Neoview Server' 'MQ Queue Manager' 'UPS' 'Number'
 'Iptelephony' 'Windows Server in extern beheer' 'Modem' 'X86 Server'
 'ESX Server' 'Virtual Tape Server' 'IPtelephony' 'NonStop Harddisk'
 'Firewall' 'RAC Service' 'zOS Systeem' 'Instance' 'NonStop Storage'
 'Protocol' 'Tape Library']
CI_Subcat
Server Based Application    18811
Web Based Application       15311
Desktop Application          3876
Laptop                       1921
SAP                          1199
                            ...
Security Software               1
Application Server              1
NonStop Storage                 1
Protocol                        1
Neoview Server                  1
Name: count, Length: 65, dtype: int64
----------------------------
```

```
Category ['incident' 'request for information' 'complaint' 'request for change']
Category
incident                   37748
request for information     8846
complaint                     11
request for change             1
Name: count, dtype: int64
----------------------------
Alert_Status ['closed']
Alert_Status
closed    46606
Name: count, dtype: int64
----------------------------
Closure_Code ['Other' 'Software' 'No error - works as designed' 'Operator error'
 'Unknown' 'Data' 'Referred' 'Hardware' 'Questions' 'User error' 'Inquiry'
 'User manual not used' 'Kwaliteit van de output' '' 'Overig']
Closure_Code
Other                           16470
Software                        13027
User error                       3554
No error - works as designed     3530
Hardware                         2999
Data                             2209
Unknown                          1590
Operator error                   1539
User manual not used              765
                                  460
Inquiry                           162
Referred                          158
Questions                         132
Kwaliteit van de output            10
Overig                              1
Name: count, dtype: int64
```

Transposing the data and describing it

```
data.describe().T
```

	count	mean	min	25%	50%	75%	max	std
Priority	45226.0	4.215805	1.0	4.0	4.0	5.0	5.0	0.705624
number_cnt	46606.0	0.499658	0.000023	0.248213	0.500269	0.749094	0.999997	0.288634
No_of_Reassignments	46605.0	1.131831	0.0	0.0	0.0	2.0	46.0	2.269774
Open_Time	18612	2013-12-10 03:34:32.927143680	2012-01-10 10:49:00	2013-06-11 11:40:45	2013-11-12 13:00:30	2014-06-02 18:22:30	2014-12-03 22:58:00	NaN
Reopen_Time	871	2013-12-11 16:45:50.493685504	2013-01-10 09:58:00	2013-06-12 13:19:30	2013-11-11 14:15:00	2014-06-03 13:45:30	2014-12-03 17:24:00	NaN
Resolved_Time	17577	2013-12-16 11:14:11.138419200	2013-01-10 06:45:00	2013-06-11 14:56:00	2013-12-11 08:38:00	2014-06-03 15:03:00	2014-12-03 17:56:00	NaN
Close_Time	18333	2013-12-15 14:00:49.795450880	2013-01-10 06:45:00	2013-06-11 13:59:00	2013-12-11 07:53:00	2014-06-03 14:27:00	2014-12-03 17:56:00	NaN
Handle_Time_hrs	490	2030-04-19 02:24:00.000000256	1974-01-01 00:00:00	1995-04-08 12:00:00	2040-01-01 00:00:00	2054-03-01 00:00:00	2073-05-01 00:00:00	NaN
No_of_Related_Interactions	46492.0	1.149897	1.0	1.0	1.0	1.0	370.0	2.556338
No_of_Related_Incidents	1222.0	1.669394	1.0	1.0	1.0	1.0	63.0	3.339687
No_of_Related_Changes	560.0	1.058929	1.0	1.0	1.0	1.0	9.0	0.403596

Dropping the column ‘ Alert’ as its having same value throughout data

```
[28] #Unique value will be 1 for constant column.Here, Alert_Status is the constant feature. so we drop the column.
     data.drop(["Alert_Status"],axis=1,inplace=True)
```

Basic Checks Report

Overview:

When working with IT Service Management (ITSM) data or any dataset, it is crucial to perform basic data checks to ensure the quality and integrity of the data. These checks help identify potential issues early, ensuring that the analysis or project is based on reliable data.

Data Shape:

We examined the dimensions of the dataset using `data.shape`, which revealed 46,606 rows and 25 columns.

Data Types:

We verified the data types of each column using `data.dtypes`. All columns were identified as object types. Afterward, we separated the categorical and numerical columns for further analysis.

Descriptive Statistics:

To obtain summary statistics for numerical columns, we used `data.describe().T` to get the mean, minimum, maximum, and other relevant statistics. For categorical columns, we used `data.describe(include='O').T` to summarize their characteristics.

Unique Values:

We checked the number of unique values in categorical columns using a loop (i.e., `data[i].unique()`). One constant column was identified, which was subsequently dropped.

Value Counts:

We examined the distribution of values in categorical columns using a loop with `data[i].value_counts()` to understand the frequency of each category.

The goal of these checks is to ensure that the data is clean, complete, and suitable for further analysis or processing.

Exploratory Data Analysis (EDA)

1. Univariate Analysis:

- **Overview:** Univariate analysis focuses on analyzing individual variables independently. The purpose is to understand the distribution and properties of each variable in isolation.
- **Numerical Variables:**
  - We analyze numerical columns using summary statistics (mean, median, mode, standard deviation) and visualizations like histograms, box plots, or density plots. This helps identify the spread, central tendency, and outliers in the data.
- **Categorical Variables:**
  - For categorical columns, we assess the frequency of each category using bar plots or pie charts.

2. Bivariate Analysis:

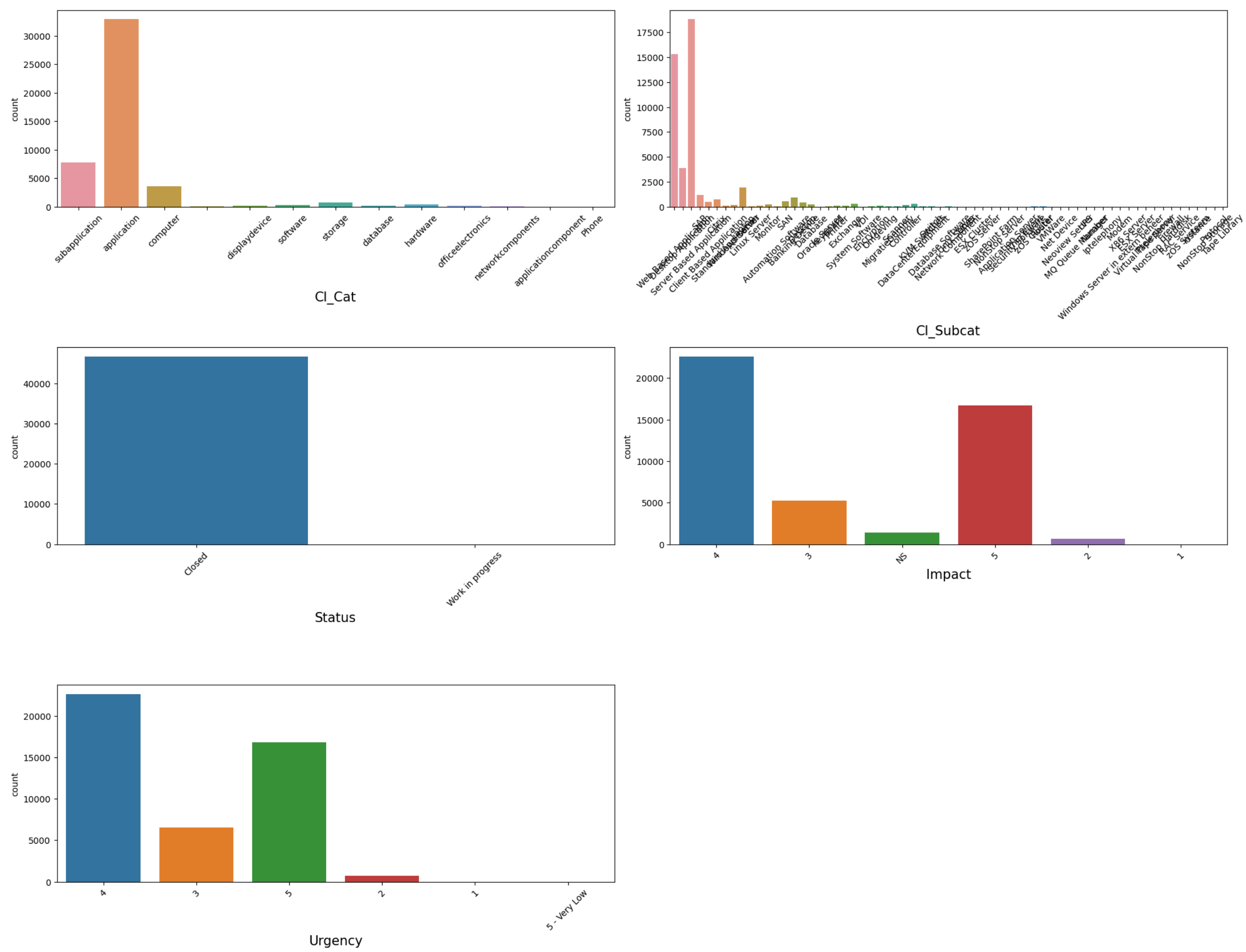
- **Overview:** Bivariate analysis looks at the relationship between two variables to understand how they interact.
- **Numerical-Numerical Relationships:**
  - We use scatter plots, correlation matrices, or pair plots to explore the relationship between two numerical variables. This helps assess trends or correlations.
- **Categorical-Numerical Relationships:**
  - Analyzing how a categorical variable affects a numerical variable can be done using box plots, violin plots, or grouped summary statistics.
- **Categorical-Categorical Relationships:**
  - The relationship between two categorical variables is explored using crosstabs or grouped bar plots.

3. Multivariate Analysis:

- **Overview:** Multivariate analysis involves examining the relationships between three or more variables to uncover complex patterns.
- **Techniques:**
  - **Heatmaps:** A correlation heatmap is used to visualize correlations between multiple numerical variables.
  - **Pair Plots:** For exploring multiple numerical relationships, pair plots give insights into interactions between variables.

Univariate Analysis:

```
[30] plt.figure(figsize=(20,25),facecolor='white')
     plotnumber = 1

     for column in data.columns:
         if plotnumber <= 5:
     # Check if the column is categorical
             if data[column].dtype == 'object':
                 ax = plt.subplot(5,2,plotnumber)
                 sns.countplot(x=data[column])
                 plt.xlabel(column,fontsize=15)
                 plt.xticks(rotation=45)
                 plotnumber+=1
     plt.tight_layout()
```

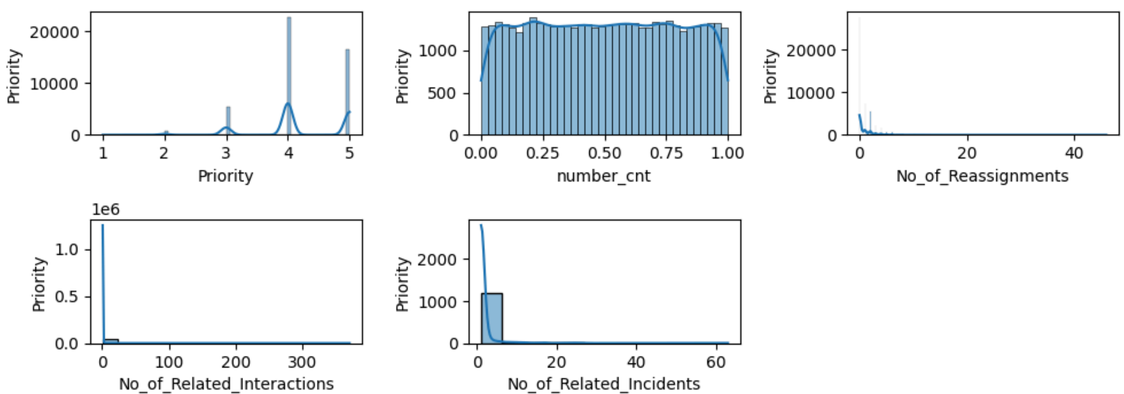


Insights:

1. **CI_cat Analysis:**
   - It was observed that the *Application* category has the highest count of tickets compared to other categories within the `CI_cat` column. This indicates that most tickets are related to application issues.
2. **Ticket Status:**
   - A majority of the tickets are in a *closed* state. This suggests that most of the reported issues have already been resolved.
3. **Impact and Urgency:**
   - In the `impact` and `urgency` columns, most tickets are assigned a priority of either 4 or 5. This indicates that the majority of issues are considered to have moderate to low urgency and impact.

Bivariate Analysis:

```
## This is for Numerical data correlated with target column:
plt.figure(figsize=(10, 5), facecolor='white')
plotnumber = 1

for column in data.columns:
    if plotnumber <= 5:
        # Check if the column is numerical
        if data[column].dtype in ['int64', 'float64']:
            ax = plt.subplot(3, 3, plotnumber)
            sns.histplot(x=data[column], kde=True)
            plt.xlabel(column, fontsize=10)
            plt.ylabel("Priority", fontsize=10)
            plotnumber += 1

plt.tight_layout()
```



Insights:

- The "No_of_Reassignments" column shows that most tickets are resolved on the first assignment. The number of reassignments decreases exponentially.

- The graph is skewed to the right, indicating that more tickets have fewer reassignments compared to those with many reassignments.

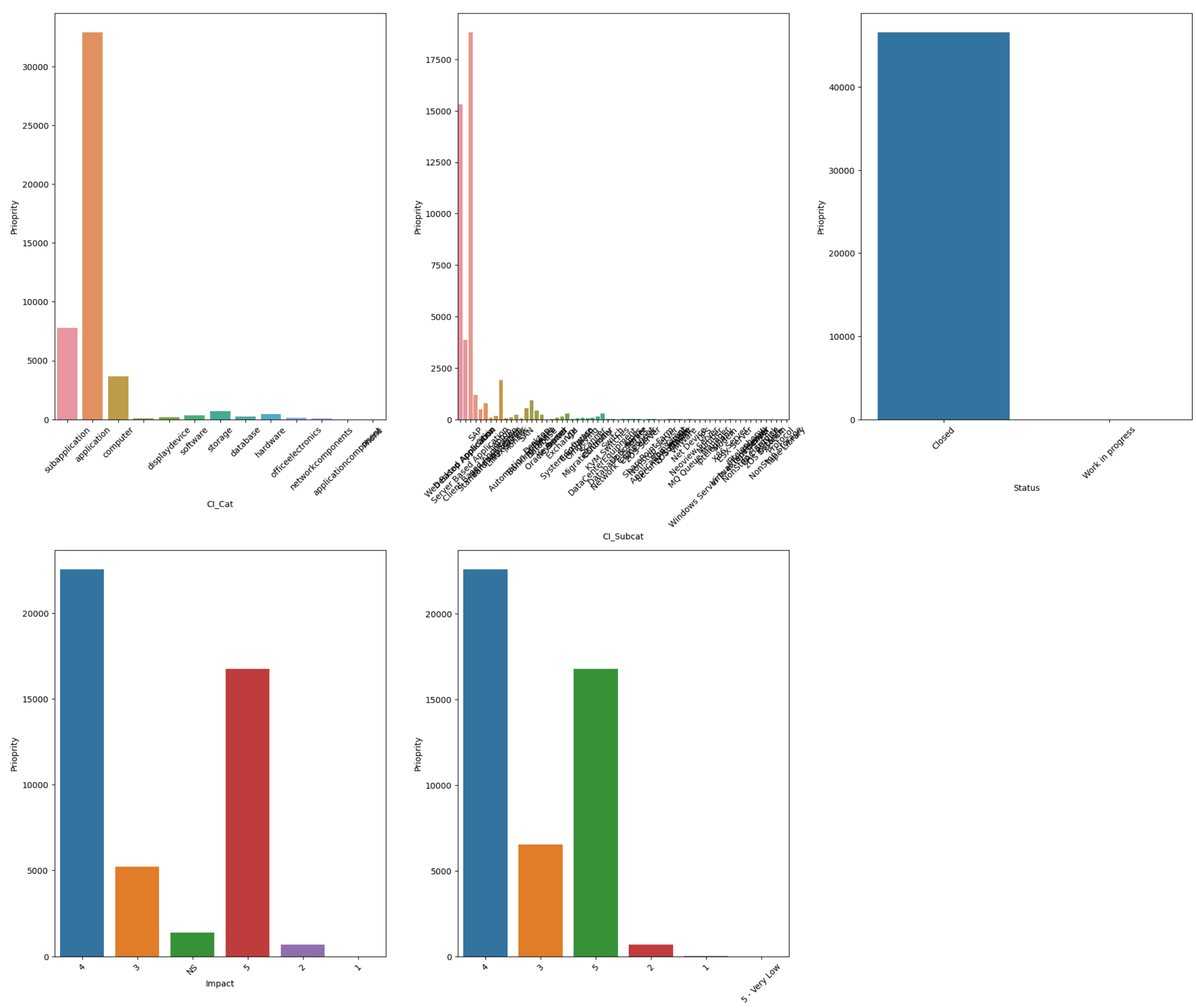
- The median number of related interactions is 1.

- 75% of tickets have 3 or fewer related interactions.

- There are a few outliers with a high number of related interactions, going up to 20.

- The distribution of the number of related incidents is skewed to the right, meaning that most incidents have fewer related incidents than those with many related incidents.

```python
## This is for categorical data correlated with target column:
plt.figure(figsize=(20, 25), facecolor='white')
plotnumber = 1

for column in data.columns:
    if plotnumber <= 5:
        # Check if the column is categorical
        if data[column].dtype == 'object':
            ax = plt.subplot(3, 3, plotnumber)
            sns.countplot(x=data[column])
            plt.xlabel(column, fontsize=10)
            plt.ylabel("Priopity", fontsize=10)
            plt.xticks(rotation=45)
            plotnumber += 1

plt.tight_layout()
```



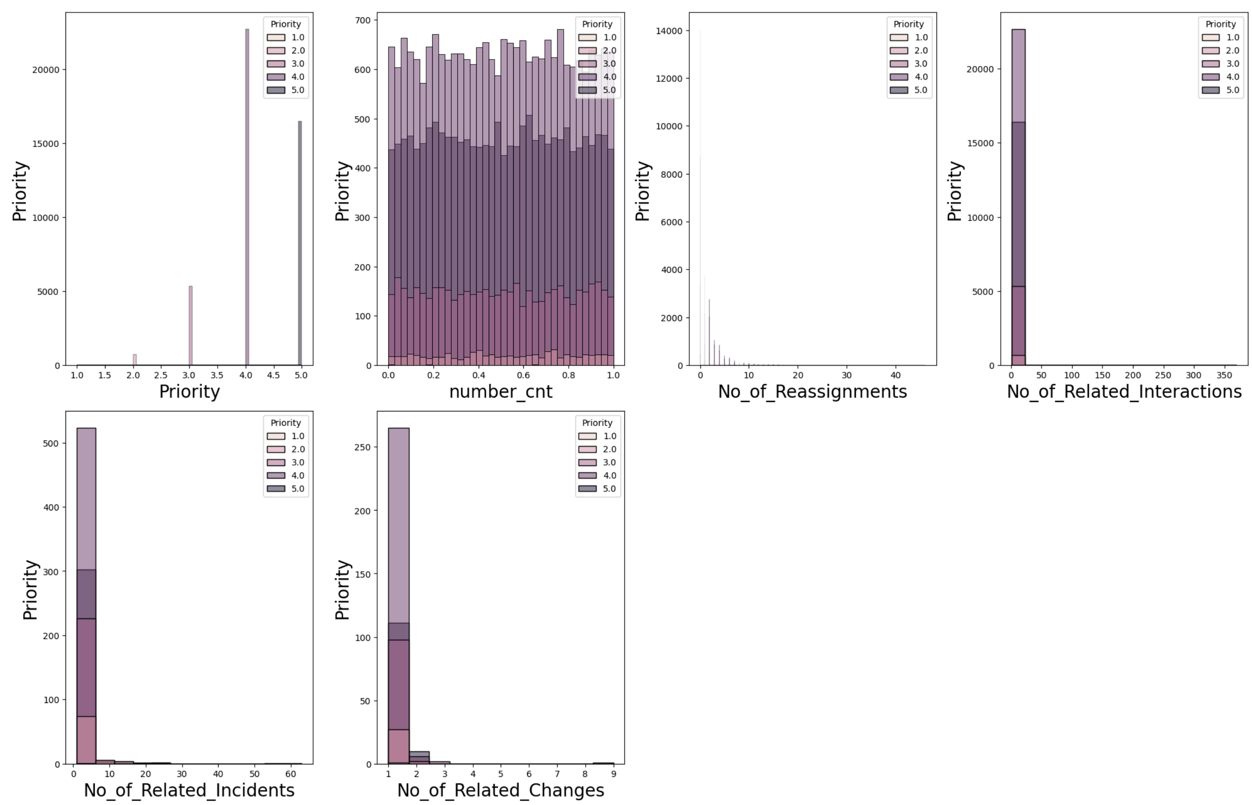
Insights:

- In the "CI_cat" column, it is observed that the application category has a higher priority count compared to others.

- The status of almost all tickets is in the closed state.

- In the "Impact" and "Urgency" columns, most tickets have an impact and urgency level of either 4 or 5.

[33]
```
plt.figure(figsize=(20, 25), facecolor='white')
plotnumber = 1

for column in data.columns:
    if plotnumber <= 16:
        # Check if the column contains numeric data
        if data[column].dtype in ['int64', 'float64']:
            ax = plt.subplot(4, 4, plotnumber)
            sns.histplot(x=data[column], hue=data.Priority)
            plt.xlabel(column, fontsize=20)
            plt.ylabel('Priority', fontsize=20)
            plotnumber += 1

plt.tight_layout()
```



Insights:

- The most common number of related changes is 1.

- There are more changes with fewer related changes than those with a higher number of related changes.

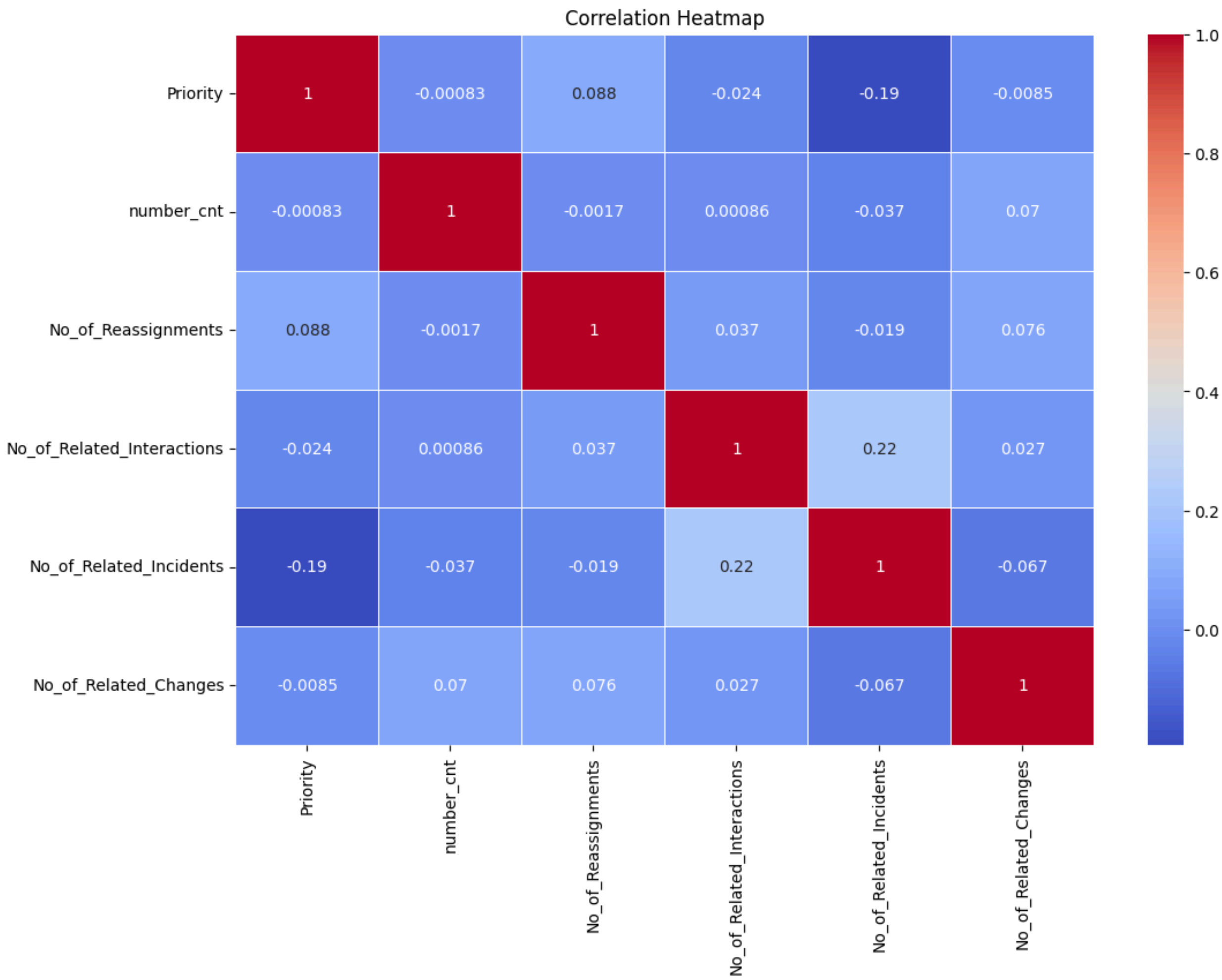
- The median number of related changes is 2.

Multivarite Analysis:

```
[34] plt.figure(figsize=(40,45))
     sns.pairplot(data)
     plt.tight_layout()
```

Checking correlation between columns

```
[35] # Select only the numeric columns for correlation analysis:
     numeric_data = data.select_dtypes(include=['int64', 'float64'])

     plt.figure(figsize=(12, 8))
     correlation = numeric_data.corr()
     sns.heatmap(correlation, annot=True, cmap='coolwarm', linewidths=0.5)
     plt.title('Correlation Heatmap')
```



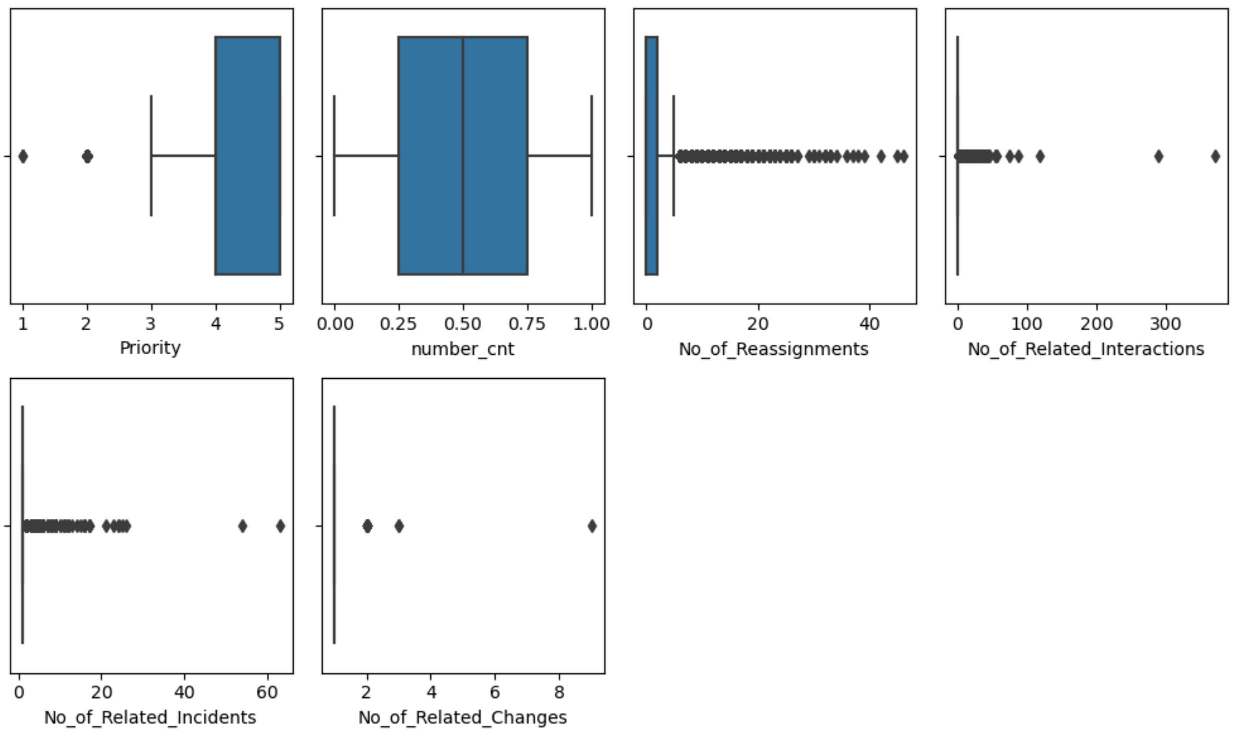
Insights:

- There is no feature highly correlated with the dependent feature (Priority), so no features are dropped.

- The correlation with Priority is very low across all features.

Check for Outliers:

```
[ ]  plt.figure(figsize=(10,6))
     plotnumber=1
     for col in numeric_data:
         if plotnumber<8:
             ax=plt.subplot(2,4,plotnumber)
             sns.boxplot(x=data[col])
             plt.xlabel(col,fontsize=10)
             plotnumber+=1
     plt.tight_layout()
```



Box Plot and Outliers Handle for :

1. No_of_Reassignments
2. No_of_Related_Interactions
3. No_of_Related_Incidents
4. No_of_Related_Changes

Box Plot for Column_names:

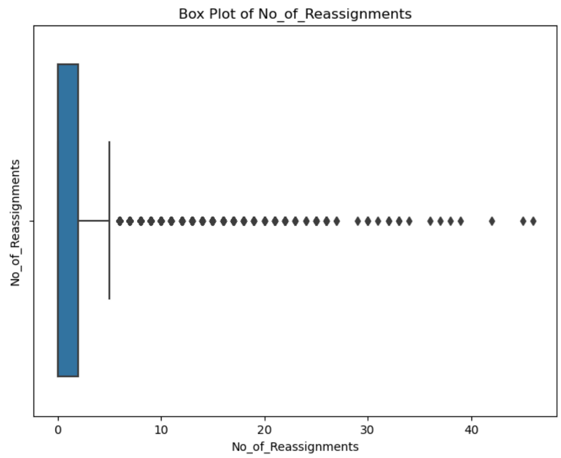
```
[ ]  plt.figure(figsize=(8, 6))
     sns.boxplot(x=data['Column_name'])
     plt.title('Box Plot of No_of_Reassignments')
     plt.ylabel('Column_name')
```

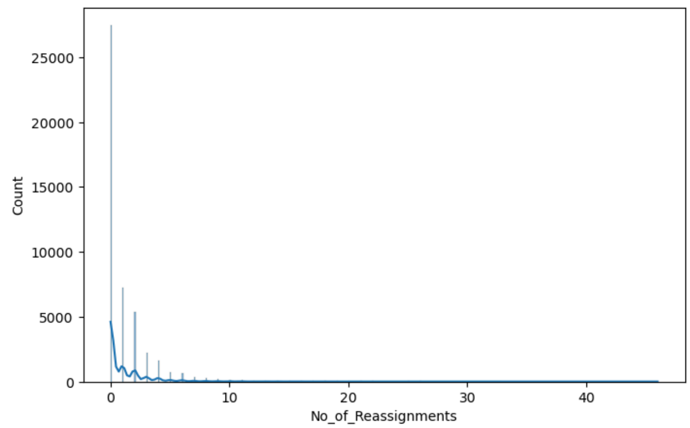
Outliers Handle for Column_names:

```
[ ]  plt.figure(figsize=(8,5))
     sns.histplot(data=data,x='Columns_names',kde=True)
```

```
Q1 = data["Column_names"].quantile(0.25)
print("lower_quartile", Q1)

Q3 = data["Column_names"].quantile(0.75)
print("upper_quartile", Q3)

IQR = Q3 - Q1
print("IQR", IQR)

lower_limit = Q1 - 1.5 * IQR
print("lower_limit is", lower_limit)

upper_limit = Q3 + 1.5 * IQR
print("upper_limit is", upper_limit)

# Identifying outliers
outliers = data.loc[data["Column_names"] > upper_limit]
print(outliers)

# Checking the proportion of outliers
outlier_proportion = len(outliers) / len(data)
print("Proportion of outliers:", outlier_proportion)

# Replacing outliers with the median
data.loc[data["Column_names"] > upper_limit, "Column_names"] = data["Column_names"].median()

# Visualizing using a boxplot
sns.boxplot(x=data["Column_names"])
```

1 No_of_Ressignment





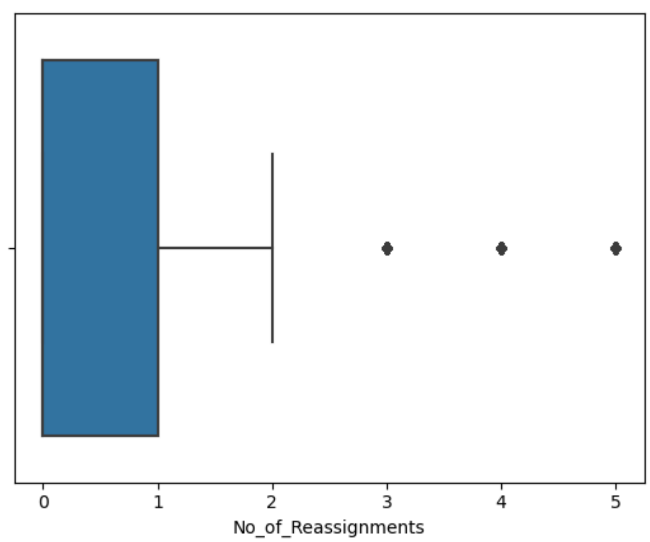
Insights:

- Most tickets with 0 reassignments are resolved without being reassigned.

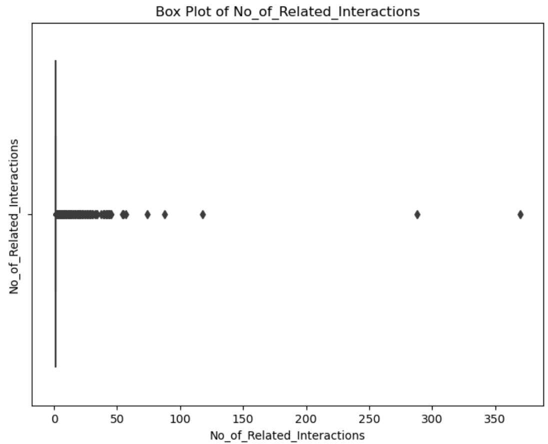
- The number of reassignments decreases exponentially.

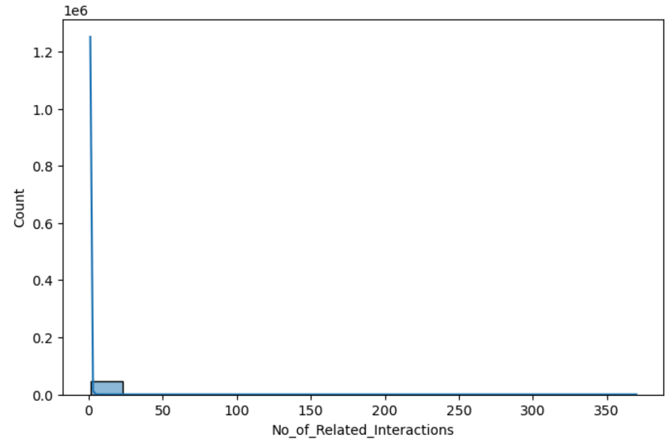
- There is a slight peak at 3 reassignments.

- The graph is skewed to the right, indicating that more tickets have fewer reassignments compared to those with many reassignments.



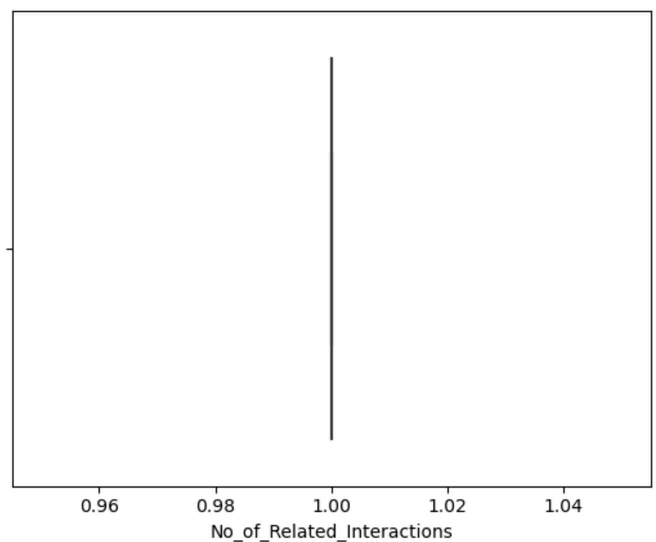
2 No_of_Related_Interactions





Insight:

- Most tickets with 0 interactions are resolved without any interaction.



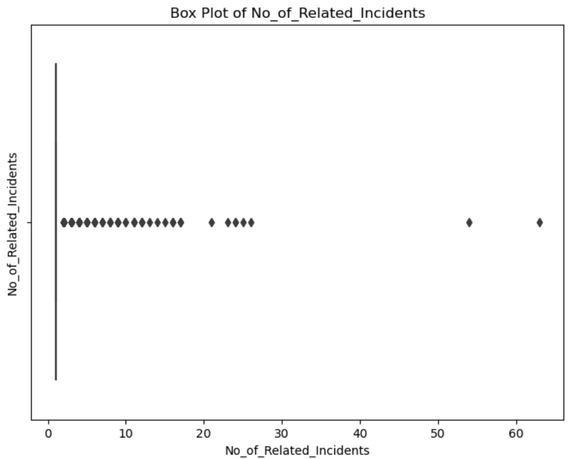
Insights:

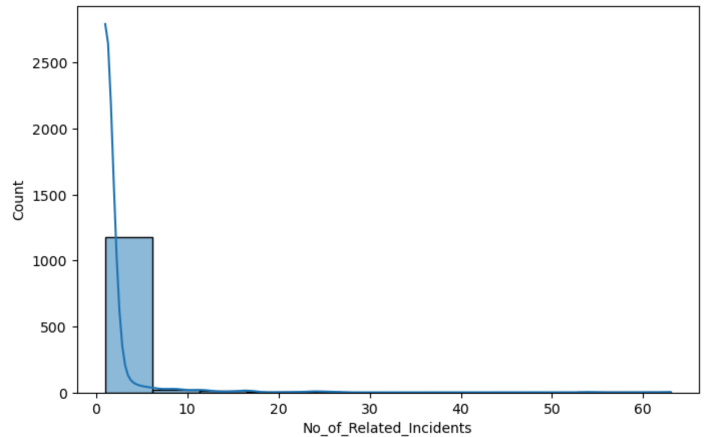
- The median number of related interactions is 1.

- 75% of tickets have 3 or fewer related interactions.

- There are a few outliers with a high number of related interactions, reaching up to 20.

3 No_of_Related_Incidents



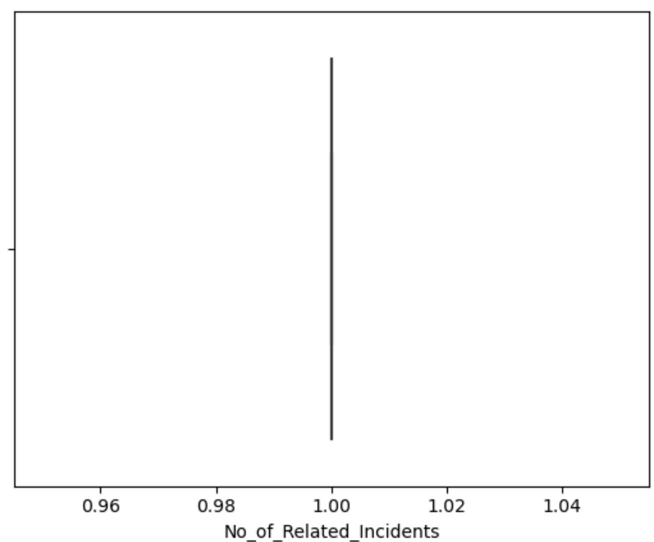


Insights:

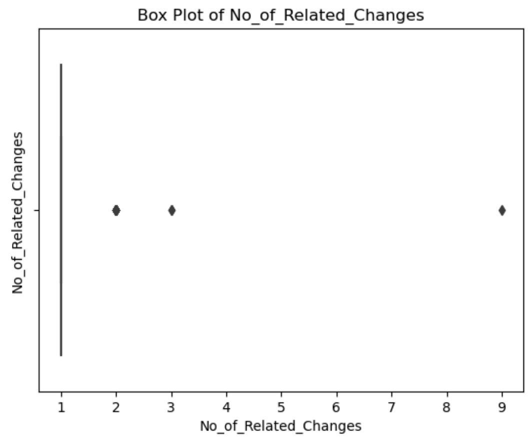
- The most common number of related incidents is 1.

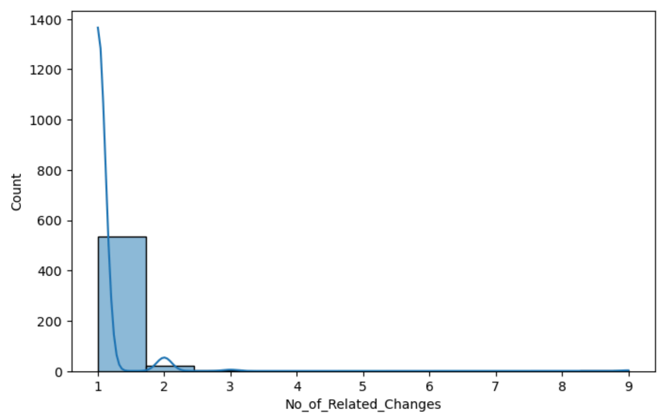
- The distribution of related incidents is skewed to the right, indicating that more incidents have fewer related incidents compared to those with a higher number.

- The maximum number of related incidents is 63.



4 No_of_Related_Changes

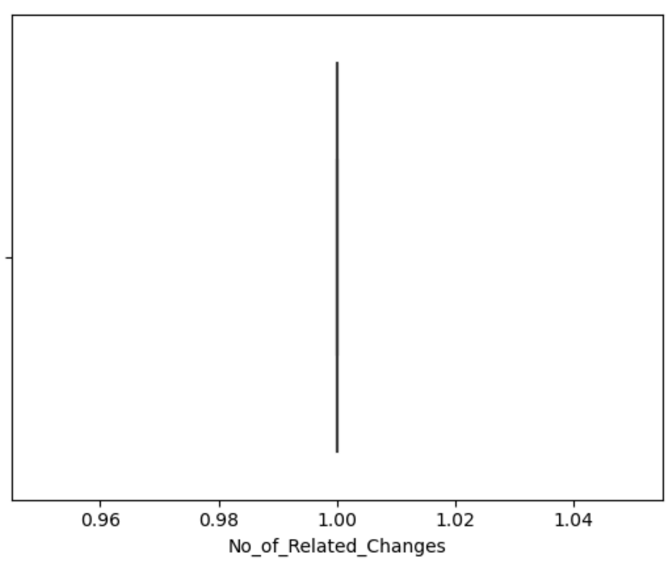




Insights:

- The most common number of related changes is 1.

- The distribution of related changes is skewed to the right, indicating that there are more changes with fewer related changes compared to those with a higher number.



Feature Engineering

```
Data columns (total 18 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   CI_Cat                        46606 non-null  object
 1   CI_Subcat                     46606 non-null  object
 2   Status                        46606 non-null  object
 3   Impact                        46606 non-null  object
 4   Urgency                       46606 non-null  object
 5   Priority                      45226 non-null  float64
 6   number_cnt                    46606 non-null  float64
 7   Category                      46606 non-null  object
 8   No_of_Reassignments           46605 non-null  float64
 9   Open_Time                     18612 non-null  datetime64[ns]
 10  Reopen_Time                   871 non-null    datetime64[ns]
 11  Resolved_Time                 17577 non-null  datetime64[ns]
 12  Close_Time                    18333 non-null  datetime64[ns]
 13  Handle_Time_hrs               490 non-null    datetime64[ns]
 14  Closure_Code                  46606 non-null  object
 15  No_of_Related_Interactions    46492 non-null  float64
 16  No_of_Related_Incidents       1222 non-null   float64
 17  No_of_Related_Changes         560 non-null    float64
dtypes: datetime64[ns](5), float64(6), object(7)
```

There are 18 columns that need preprocessing:

1. Print the sum of null values and empty names for each specific feature.
2. Use the mode of each feature to fill in the null values or empty names.

```
[ ] print("Null - ",data["CI_Cat"].isnull().sum()) # printing the sum of null values present for particular feature.
    print("Empty names - ",(data["CI_Cat"] == '').sum()) # printing the sum  of empty names  present for particular feature.

    Null -  0
    Empty names -  111

[ ] data['CI_Cat'].mode() # It gives the mode of particular feature.

    0    application
    Name: CI_Cat, dtype: object

[ ] # There were 111 empty values present in this column replacing those with mode i.e. application
    data.loc[data["CI_Cat"] == '',"CI_Cat"] = 'application'
```

3. Convert the categorical columns to numerical format using LabelEncoder.

```
[ ] # Transforming the column from categorical column to numerical column using label_encoder

    from sklearn.preprocessing import LabelEncoder
    encoder=LabelEncoder()

[ ] data['CI_Cat']=encoder.fit_transform(data['CI_Cat'])
```

*******Doing same for all the columns in table*******

The column ***data['Handle_Time_hrs']*** does not contain meaningful information, so we can manually create ***Handle_Time_hrs_conv*** and drop the original column.

To do this, we will calculate the difference between the ***open_time* and *close_time*** columns in days and convert it to hours.

```
data.drop("Handle_Time_hrs", axis=1, inplace=True)
```

```
[ ] data['Handle_Time_hrs_conv']=abs(data['Close_Time']-data['Open_Time'])

[ ] a=[]
    for i in data['Handle_Time_hrs_conv'].index:
        a.append((data['Handle_Time_hrs_conv'][i].total_seconds())/3600)

[ ] data['Handle_Time_hrs_conv']=a
```

Since the closure code does not determine the ticket priority and its importance is only considered at a later stage of ticket resolution, we can drop the column.

```
data.drop('Closure_Code', axis=1, inplace=True)
```

We are dropping the columns `No_of_Related_Changes` and `No_of_Related_Incidents` because they contain more than 50% null values.

data['No_of_Related_Incidents']

```
print("Null - ",data["No_of_Related_Incidents"].isnull().sum())
print("Empty names - ",(data["No_of_Related_Incidents"]=='').sum())
```

```
Null -  45383
Empty names -  0
```

data['No_of_Related_Changes']

```
[ ] print("Null - ",data["No_of_Related_Changes"].isnull().sum())
    print("Empty names - ",(data["No_of_Related_Changes"]=='').sum())
```

```
Null -  46045
Empty names -  0
```

```
data.drop('No_of_Related_Changes',axis=1,inplace=True)

data.drop('No_of_Related_Incidents',axis=1,inplace=True)
```

Since we have already used the columns `Close_Time`, `Open_Time`, and `Resolved_Time` to create `handle_time_hrs_conv`, we can drop these columns.

```
data.drop(['Open_Time','Resolved_Time','Close_Time',],axis=1,inplace=True)
```

Preprocessed Dataset for ML

```
data.head()
```

	CI_Cat	CI_Subcat	Status	Impact	Urgency	Priority	number_cnt	Category	No_of_Reassignments	No_of_Related_Interactions	Handle_Time_hrs_conv
0	11	57	0	4	4	4.0	0.601292	1	0.0	1	8256.316667
1	1	57	0	3	3	3.0	0.415050	1	0.0	1	1700.866667
2	1	10	0	4	3	4.0	0.517551	3	3.0	1	7291.733333
3	1	57	0	4	4	4.0	0.642927	1	0.0	1	7291.733333
4	1	57	0	4	4	4.0	0.345258	1	2.0	1	7370.900000

```
[ ] data.shape
```

```
(46605, 11)
```

Task 1: Predicting High Priority Tickets

The goal is to predict priority 1 and 2 tickets so that preventive measures can be taken or the problem can be fixed before it arises.

```
[ ] y=df_1['Priority'].map({1:1,2:1,3:0,4:0,5:0})
```

```
[ ] y.value_counts()
```

```
Priority
0    45905
1      700
Name: count, dtype: int64
```

Mapping the 1-5 level priorities to 1 and 0 as per Task 1.

Priority 1 and 2 will be mapped to 1 (high priority), while priorities 3, 4, and 5 will be mapped to 0 (low priority).

Train-Test Split:

This process involves dividing a dataset into two subsets: one for training a model and the other for testing its performance. Typically, the data is split into a training set (used to train the model) and a test set (used to evaluate the model's accuracy and generalization ability). This helps ensure that the model is tested on data it hasn't seen during training, providing a more realistic assessment of its performance.

Train test split

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,stratify=y)
```

```
[ ] print(X_train.shape)
    print(X_test.shape)
    print(y_train.shape)
    print(y_test.shape)
```

```
(32623, 9)
(13982, 9)
(32623,)
(13982,)
```

Logic Behind the Function

1. Create a dictionary named `model_summary` with keys and null values.
2. The function `model_selection_1` will take a model as a parameter.
3. Inside the function, the model will be initialized and stored in a variable called `model`.
4. The model will be trained using `x_train` and `y_train`.
5. The model will make predictions on the test data.
6. After prediction, evaluation metric values will be added to the dictionary with corresponding keys.
7. The function will print the confusion matrix and classification report for the model.
8. The same steps will be repeated for the training data.

Importing necessary libraries for Model creation and evaluation:

```
[ ] ## Model creation
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier,BaggingClassifier,GradientBoostingClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.linear_model import LogisticRegression
    from sklearn.svm import SVC

    # Model evaluation
    from sklearn.metrics import confusion_matrix,classification_report,ConfusionMatrixDisplay,f1_score,recall_score,accuracy_score
```

Creating Model and Evaluating

```
model_summary={'model_name_train':[],'f1_score_train':[],'recall_score_train':[],'accuracy_score_train':[],
               'model_name_test':[],'f1_score_test':[],'recall_score_test':[],'accuracy_score_test':[]}

def model_selction_1(model):

    # Model initialization ,fitting and predicting

    print(model)
    model=model()
    model.fit(X_train,y_train)
    model_pred=model.predict(X_test)

    # Appending the metrics to the dictionary created

    model_summary['model_name_test'].append(model.__class__.__name__)
    model_summary['f1_score_test'].append(f1_score(y_test,model_pred,average='macro'))
    model_summary['recall_score_test'].append(recall_score(y_test,model_pred,average='macro'))
    model_summary['accuracy_score_test'].append(accuracy_score(y_test,model_pred))

    # Printing the confusion metrics and classification report

    print('metrics on test data')
    print(confusion_matrix(y_test,model_pred))
    print('\n')
    print(classification_report(y_test,model_pred))

    # Predictions on train data

    model_pred1=model.predict(X_train)

    # Appending the metrics to the dictionary created

    model_summary['model_name_train'].append(model.__class__.__name__)
    model_summary['f1_score_train'].append(f1_score(y_train,model_pred1,average='macro'))
    model_summary['recall_score_train'].append(recall_score(y_train,model_pred1,average='macro'))
    model_summary['accuracy_score_train'].append(accuracy_score(y_train,model_pred1))

    # Printing the confusion metrics and classification report

    print('metrics on train data')
    print(confusion_matrix(y_train,model_pred1))
    print('\n')
    print(classification_report(y_train,model_pred1))
    print('==='*10)
```

Models are :

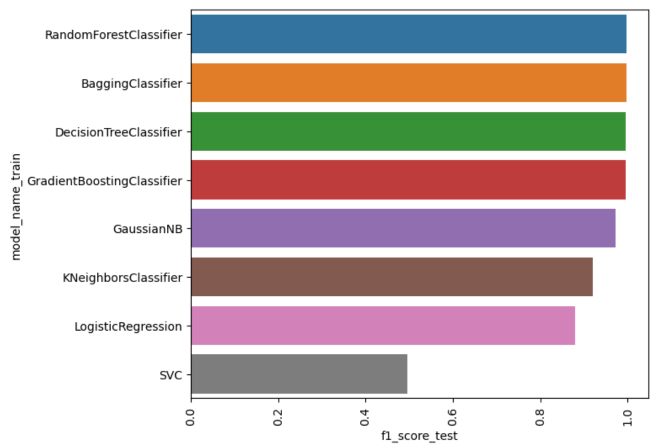
```
models = [LogisticRegression,DecisionTreeClassifier,RandomForestClassifier,
          BaggingClassifier,KNeighborsClassifier,GaussianNB,SVC,GradientBoostingClassifier]
```

Summary of Chosen Metrics for All Models

```
summary=pd.DataFrame(model_summary).sort_values('f1_score_test',ascending=False).drop('model_name_test',axis=1)
```

```
summary
```

	model_name_train	f1_score_train	recall_score_train	accuracy_score_train	f1_score_test	recall_score_test	accuracy_score_test
2	RandomForestClassifier	1.000000	1.000000	1.000000	0.997571	0.995238	0.999857
3	BaggingClassifier	1.000000	1.000000	1.000000	0.997571	0.995238	0.999857
1	DecisionTreeClassifier	1.000000	1.000000	1.000000	0.996366	0.995202	0.999785
7	GradientBoostingClassifier	0.998451	0.999953	0.999908	0.995165	0.995165	0.999714
5	GaussianNB	0.983977	0.969388	0.999080	0.971963	0.947619	0.998427
4	KNeighborsClassifier	0.950246	0.917222	0.997272	0.921192	0.887623	0.995709
0	LogisticRegression	0.892844	0.889291	0.993716	0.879137	0.879137	0.992848
6	SVC	0.496217	0.500000	0.984980	0.496217	0.500000	0.984981



Model Selection for Task 1

From the graph, it is observed that the `RandomForestClassifier`, `BaggingClassifier`, `DecisionTreeClassifier`, and `GradientBoostingClassifier` are performing well compared to other algorithms, achieving performance above 95%. Therefore, additional optimization techniques are not required separately.

We are choosing the `GradientBoostingClassifier` over the other models—`RandomForestClassifier`, `BaggingClassifier`, and `DecisionTreeClassifier`—as it has demonstrated better performance more consistently.

We will proceed with creating the `GradientBoostingClassifier` model for further use.

```
# Model creation
# Model initialization
high_priority_model=GradientBoostingClassifier()

# Fitting the model
high_priority_model.fit(X_train,y_train)

# Predicting using the model
high_priority_pred=high_priority_model.predict(X_test)

# Printing the confusion metrics and classification report
print('metrics on test data')
print('confusion matrix')
print(confusion_matrix(y_test,high_priority_pred))
print('\n')
print('classification report')
print(classification_report(y_test,high_priority_pred))
print('==='*10)
```

Accuracy: Measures the proportion of correctly predicted instances out of the total instances. It provides a general idea of the model's performance.

Precision: Indicates the proportion of true positive predictions among all positive predictions made by the model. It reflects how many of the predicted positives are actually positive.

Recall (Sensitivity): Shows the proportion of true positive predictions among all actual positive instances. It reflects how well the model captures the actual positives.

F1 Score: The harmonic mean of precision and recall. It balances the two metrics and provides a single score to evaluate the model's performance, especially when dealing with imbalanced classes.

Confusion Matrix: A table that shows the counts of true positives, true negatives, false positives, and false negatives. It helps in understanding the model's classification performance.

Classification Report: A detailed report that includes precision, recall, F1 score, and support for each class. It provides a comprehensive view of the model's performance in each class.

```
metrics on test data
confusion matrix
[[   0    0    0    0    0    0    0    0    0    0    0    0]
 [   0 9127    0    0    0    0    0    0    0    0    0  776]
 [   0    0    1    0    0    0    0    0    0    0    0    0]
 [   1    0    0 1092    0    0    0    0    0    0    0    0]
 [   0    0    0    0   63    0    0    1    0    0    0    0]
 [   0    0    0    0    0   64    0    0    0    0    0    0]
 [   0    0    0    1    0    0  132    0    0    0    0    0]
 [   0    0    0    2    1    0    0   26    0    0    3    0]
 [   0    0    0    0    0    0    0    0   46    0    0    0]
 [   0    0    0    0    0    0    0    0    0  100    0    0]
 [   0    1    0    0    0    1    0    0    0    0  209    0]
 [   0 1401    0    0    0    0    0    0    0    0    0  934]]


classification report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.87      0.92      0.89      9903
           2       1.00      1.00      1.00         1
           3       1.00      1.00      1.00      1093
           4       0.98      0.98      0.98        64
           5       0.98      1.00      0.99        64
           6       1.00      0.99      1.00       133
           7       0.96      0.81      0.88        32
           8       1.00      1.00      1.00        46
           9       1.00      1.00      1.00       100
          10       0.99      0.99      0.99       211
          11       0.55      0.40      0.46      2335

    accuracy                           0.84     13982
   macro avg       0.86      0.84      0.85     13982
weighted avg       0.83      0.84      0.83     13982
```

Task 2: Forecasting

Forecast the incident volume across different fields on a quarterly and annual basis. This will help in better resource allocation and technology planning.

```
# Imporing the necessary columns

# Importing the necessary columns from  dataset, which include
'Incident_ID' and 'Open_Time'.

df_2 = df.loc[:,['Incident_ID','Open_Time']]
```

Parsing the date to one format `[%Y-%m-%d]`

```
[ ] import datetime as dt
```

```
[ ] # Define a function to parse the date formats
    def parse_date(date_str):
        try:
            # Try to parse the date using one format
            return pd.to_datetime(date_str, format='%d/%m/%Y %H:%M')
        except ValueError:
            # If the first format fails, try the second format
            return pd.to_datetime(date_str, format='%d-%m-%Y %H:%M')

    # Apply the function to the 'Open_Time' column
    df_2['Open_Time'] = df_2['Open_Time'].apply(parse_date)

    # Convert the 'Open_Time' column to the desired string format
    df_2['Open_Time'] = df_2['Open_Time'].dt.strftime('%Y-%m-%d')

    # Print the modified DataFrame
    df_2
```

Exploratory Data Analysis

```
[ ] # Adding a new column which will have the number of tickets per day
    df_2['No_Incidents'] = df_2.groupby('Open_Time')['Incident_ID'].transform('count')
```

```
[ ] df_2.drop(['Incident_ID'],axis=1,inplace=True)
```

```
[ ] # After converting the dates to a consistent format, we created a new DataFrame with the 'Incident_ID' column removed.
    df_2.head()
```

	Open_Time	No_Incidents
0	2012-02-05	1
1	2012-03-12	1
2	2012-03-29	1
3	2012-07-17	1
4	2012-08-10	2

Data Cleaning and Indexing

1. Calculated the number of incidents per day and identified duplicate values in the dataset.
2. After removing the duplicates, set the `Open_Time` column as the index.
3. Checked the date range to ensure the data is complete and accurately represents the incidents over time.

```
[ ] df_2.duplicated().sum()
```

```
46275
```

```
[ ] df_2.drop_duplicates(inplace=True)
```

	Open_Time	No_Incidents
0	2012-02-05	1
1	2012-03-12	1
2	2012-03-29	1
3	2012-07-17	1
4	2012-08-10	2
...	...	...
45857	2014-03-27	269
46154	2014-03-28	205
46354	2014-03-29	5
46386	2014-03-30	3
46387	2014-03-31	217

331 rows × 2 columns

Set the `Open_Time` column as the index of the DataFrame `df_2` and convert it to datetime format:

df_2 = df_2.set_index('Open_Time')

df_2.index = pd.to_datetime(df_2.index)

Check the range of dates:

print(df_2.index.min(), 'to', df_2.index.max())

Create a new Series with a daily frequency:

data1 = df_2['No_Incidents']

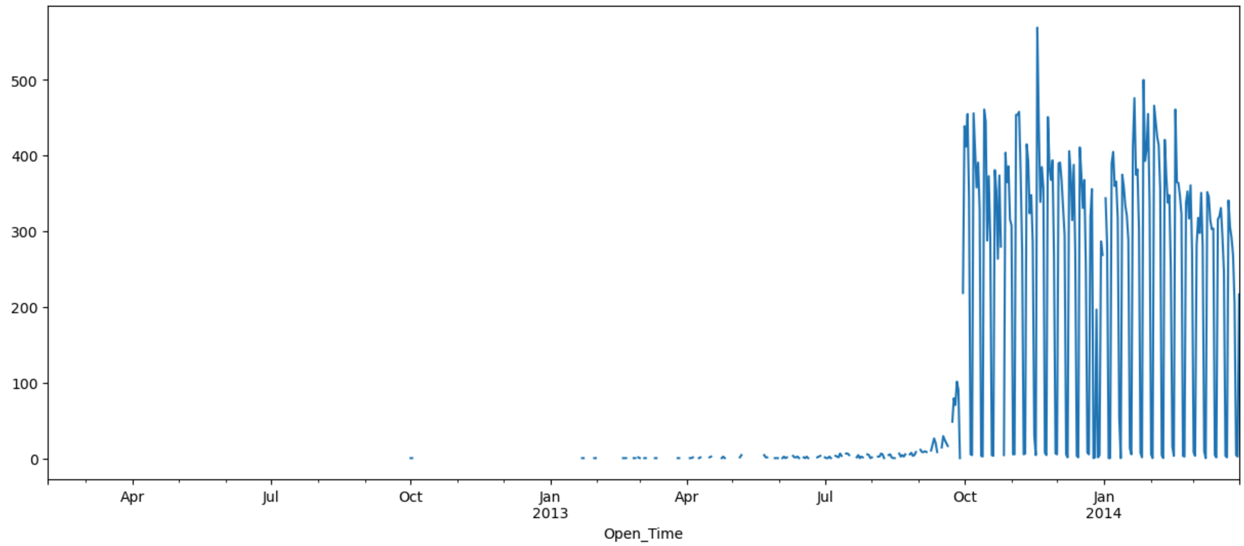
data1 = data1.asfreq('D')

Display the new Series index to verify:

data1.index

```
DatetimeIndex(['2012-02-05', '2012-02-06', '2012-02-07', '2012-02-08',
               '2012-02-09', '2012-02-10', '2012-02-11', '2012-02-12',
               '2012-02-13', '2012-02-14',
               ...
               '2014-03-22', '2014-03-23', '2014-03-24', '2014-03-25',
               '2014-03-26', '2014-03-27', '2014-03-28', '2014-03-29',
               '2014-03-30', '2014-03-31'],
              dtype='datetime64[ns]', name='Open_Time', length=786, freq='D')
```

Data Analysis and Visualization



Analyzing Incident Volume

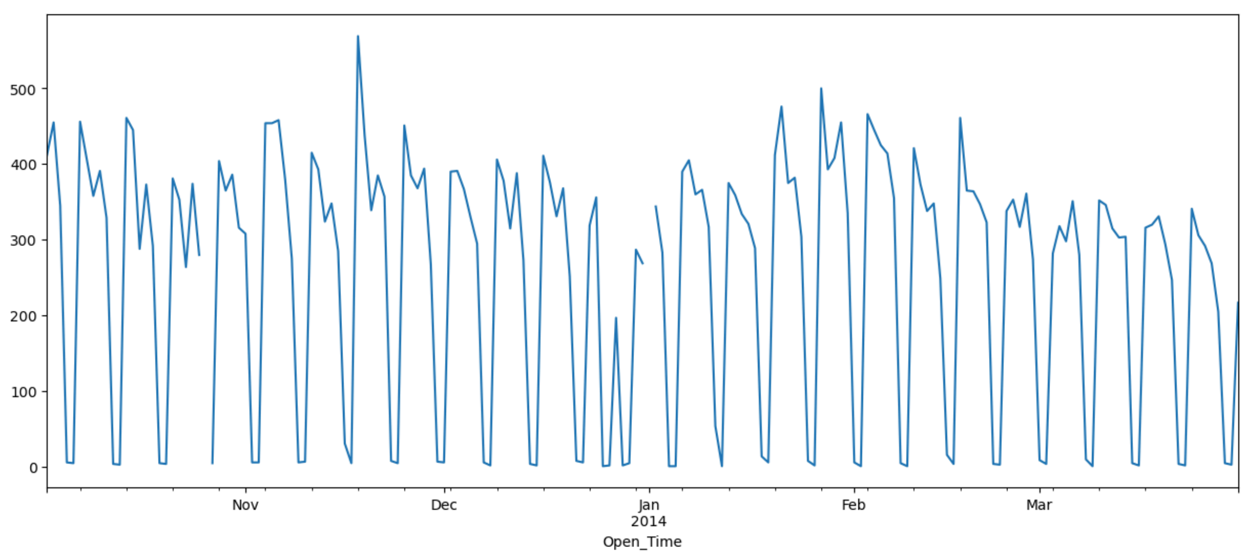
1. Created a time series plot of the number of tickets per day to better understand the incident volume.
2. Observed a significant increase in incidents after October 2013 from the plot.
3. Note that there were fewer tickets before October 2013, so we will focus on the data from after this date.

	No_Incidents
Open_Time	
2013-10-02	412
2013-10-03	455
2013-10-04	345
2013-10-07	456
2013-10-05	6

```
# new Series
data2 = incfrom2013['No_Incidents']
data2 = data2.asfreq('D')
data2.index
```

```
DatetimeIndex(['2013-10-02', '2013-10-03', '2013-10-04', '2013-10-05',
               '2013-10-06', '2013-10-07', '2013-10-08', '2013-10-09',
               '2013-10-10', '2013-10-11',
               ...
               '2014-03-22', '2014-03-23', '2014-03-24', '2014-03-25',
               '2014-03-26', '2014-03-27', '2014-03-28', '2014-03-29',
               '2014-03-30', '2014-03-31'],
              dtype='datetime64[ns]', name='Open_Time', length=181, freq='D')
```

Plotting Number of Tickets Per Day After October 2013



Time Series Forecasting

```
import itertools
import statsmodels.api as sm
```

```
# Making a list of values for p,d & q
p = d = q = range(0,2)
pdq = list(itertools.product(p,d,q))
```

```
# Choosing the ARIMA Model
# Checking the AIC values per pairs
for param in pdq:
    mod = sm.tsa.statespace.SARIMAX(data2,order=param,enforce_stationarity=False,enforce_invertibility=False)
    results = mod.fit()
    print('ARIMA{} - AIC:{}'.format(param, results.aic))
```

```
ARIMA(0, 0, 0) - AIC:2539.6180293605685
ARIMA(0, 0, 1) - AIC:2373.785382472209
ARIMA(0, 1, 0) - AIC:2371.128960804689
ARIMA(0, 1, 1) - AIC:2313.1363347365786
ARIMA(1, 0, 0) - AIC:2365.2916469365655
ARIMA(1, 0, 1) - AIC:2337.3125086933514
ARIMA(1, 1, 0) - AIC:2373.128068065154
ARIMA(1, 1, 1) - AIC:2294.43158124368
```

Time series forecasting involves predicting future values based on previously observed values

Choosing the ARIMA Model with Minimum AIC for Time Series Forecasting

1. **Model Selection Using AIC**:
    - Evaluate different ARIMA models by calculating their Akaike Information Criterion (AIC) values. The AIC helps in selecting the model that best balances fit and complexity.
    - Select the ARIMA model with the lowest AIC value as it indicates the best trade-off between model accuracy and complexity.
2. **Fitting the Selected ARIMA Model**:
    - Once the model with the minimum AIC is identified, fit the ARIMA model to the time series data using the chosen parameters.
3. **Forecasting**:
    - Use the fitted ARIMA model to generate forecasts for future values based on the historical data.

```
# Choosing the model with minimum AIC and the ARIMA Model for Time Series Forecasting
mod = sm.tsa.statespace.SARIMAX(data2,order=(1,1,1))
results = mod.fit()
print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3386      0.090      3.771      0.000       0.163       0.515
ma.L1         -0.9989      0.428     -2.332      0.020      -1.839      -0.159
sigma2       2.52e+04   9781.707      2.576      0.010    6023.561    4.44e+04
==============================================================================
```

Summary of the Selected ARIMA Model (1,1,1)

The summary of the selected ARIMA model with parameters (1,1,1) includes:

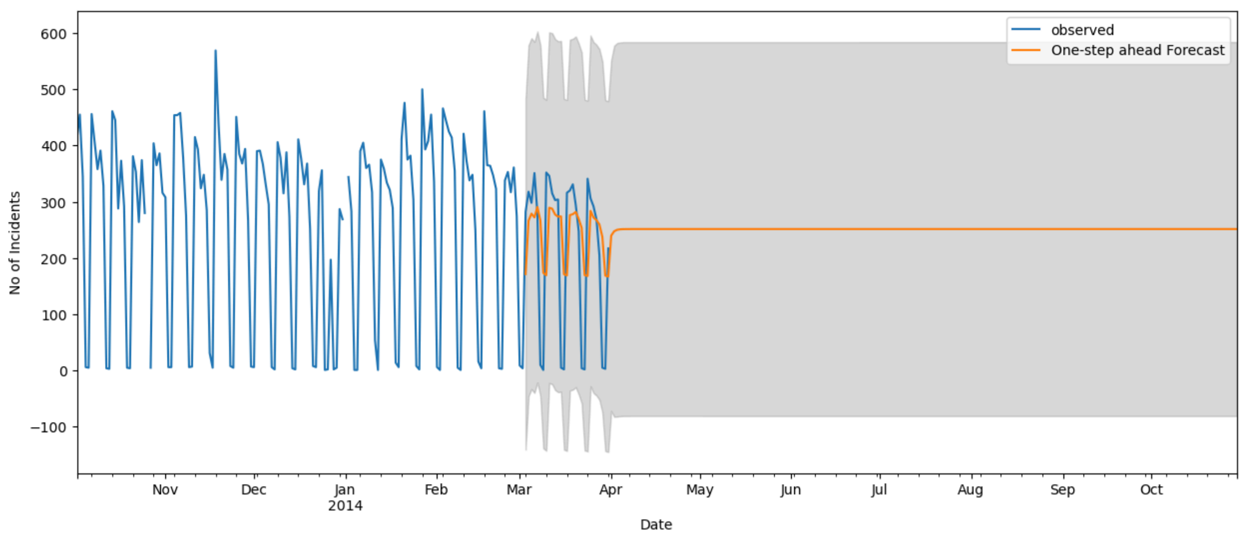
1. **Coefficients**:
   - **Autoregressive (AR) Term**: Coefficient value indicating the influence of past values on the current value.
   - **Moving Average (MA) Term**: Coefficient value reflecting the influence of past forecast errors on the current value.
2. **Sigma² Value**:
   - Represents the variance of the residuals (errors). A lower sigma² value indicates that the model's residuals are closer to zero, suggesting a better fit of the model to the data.

```
[ ] # Predicting the future values and the confidence interval
    pred = results.get_prediction(start=pd.to_datetime('2014-3-3'),end=pd.to_datetime('2014-10-30'),dynamic=False)
    pred_ci = pred.conf_int()
    pred.predicted_mean.round()
```

```
2014-03-03    172.0
2014-03-04    266.0
2014-03-05    279.0
2014-03-06    272.0
2014-03-07    291.0
              ...
2014-10-26    252.0
2014-10-27    252.0
2014-10-28    252.0
2014-10-29    252.0
2014-10-30    252.0
Freq: D, Name: predicted_mean, Length: 242, dtype: float64
```

The selected ARIMA model was used to predict future incident volumes. The forecast was made for the period from March 3, 2014, to October 30, 2014.

Visualization of Forecasted Incident Volumes



The predicted incident volumes were plotted alongside the observed data for visualization. This comparison allowed us to assess the model's performance in forecasting incident volumes.

Task 3: Tag Tickets

Automatically tag tickets with the appropriate priorities and departments to minimize reassignment and related delays.

Model Creation and Evaluation

```
[ ] model_summary_1={'model_name_train':[],'f1_score_train':[],'recall_score_train':[],'accuracy_score_train':[],
                 'model_name_test':[],'f1_score_test':[],'recall_score_test':[],'accuracy_score_test':[]}


    def model_selction_2(model):

        # Model init ialization ,fitting and predicting
        print(model)
        model=model()
        model.fit(X_train,y_train)
        model_pred=model.predict(X_test)

        # Appending the metrics to the dictionary created
        model_summary_1['model_name_test'].append(model.__class__.__name__)
        model_summary_1['f1_score_test'].append(f1_score(y_test,model_pred,average='macro'))
        model_summary_1['recall_score_test'].append(recall_score(y_test,model_pred,average='macro'))
        model_summary_1['accuracy_score_test'].append(accuracy_score(y_test,model_pred))

        # Printing the confusion metrics and classification report
        print('metrics on test data')
        print(confusion_matrix(y_test,model_pred))
        print('\n')
        print(classification_report(y_test,model_pred))

        # Predictions on train data
        model_pred1=model.predict(X_train)

        # Appending the metrics to the dictionary created
        model_summary_1['model_name_train'].append(model.__class__.__name__)
        model_summary_1['f1_score_train'].append(f1_score(y_train,model_pred1,average='macro'))
        model_summary_1['recall_score_train'].append(recall_score(y_train,model_pred1,average='macro'))
        model_summary_1['accuracy_score_train'].append(accuracy_score(y_train,model_pred1))

        # Printing the confusion metrics and classification report
        print('metrics on train data')
        print(confusion_matrix(y_train,model_pred1))
        print('\n')
        print(classification_report(y_train,model_pred1))
        print('==='*10)
```

Test train split

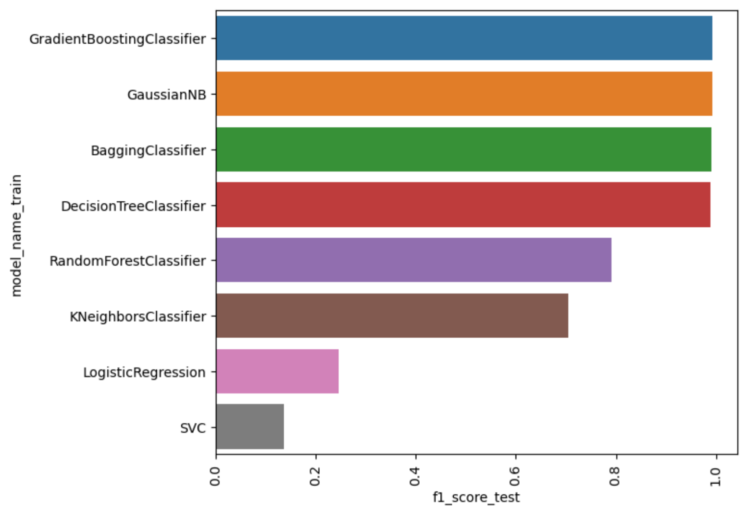
```
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.3,
random_state=42,stratify=y1)
```

Summary of Chosen Metrics for All Models

```
summary_1=pd.DataFrame(model_summary_1).sort_values('f1_score_test',ascending=False).drop('model_name_test',axis=1)
```

```
summary_1
```

	model_name_train	f1_score_train	recall_score_train	accuracy_score_train	f1_score_test	recall_score_test	accuracy_score_test
7	GradientBoostingClassifier	0.996350	0.995716	0.994452	0.993070	0.991536	0.991918
5	GaussianNB	0.992917	0.990533	0.992030	0.992912	0.990616	0.991775
3	BaggingClassifier	0.999137	0.998891	0.998988	0.990922	0.989827	0.989630
1	DecisionTreeClassifier	1.000000	1.000000	1.000000	0.988540	0.988371	0.986912
2	RandomForestClassifier	1.000000	1.000000	1.000000	0.792188	0.790622	0.990559
4	KNeighborsClassifier	0.738770	0.722150	0.935506	0.705238	0.688854	0.895437
0	LogisticRegression	0.244881	0.261841	0.598933	0.245027	0.262022	0.599414
6	SVC	0.136332	0.200000	0.517059	0.136325	0.200000	0.517022



Model Selection for Task 3: Auto-Tag Tickets with the Right Priority

From the analysis, it is observed that the `GradientBoostingClassifier`, `GaussianNB`, `BaggingClassifier`, and `DecisionTreeClassifier` models are performing well compared to other algorithms, achieving performance above 95%. Therefore, additional optimization techniques are not required.

We are selecting the `GradientBoostingClassifier` over `GaussianNB`, `BaggingClassifier`, and `DecisionTreeClassifier` because it has consistently demonstrated better performance.

We will proceed with creating the `GradientBoostingClassifier` model for further use.

```
[ ] # Model creation
    # Model initialization
    all_priority_model=GradientBoostingClassifier()

    # Fitting the model
    all_priority_model.fit(X_train,y_train)

    # Predicting using the model
    all_priority_pred=all_priority_model.predict(X_test)

    # Printing the confusion metrics and classification report
    print('metrics on test data')
    print('confusion matrix')
    print(confusion_matrix(y_test,all_priority_pred))
    print('\n')
    print('classification report')
    print(classification_report(y_test,all_priority_pred))
    print('==='*10)
```

```
metrics on test data
confusion matrix
[[   1    0    0    0    0]
 [   0  207    2    0    0]
 [   0    0 1562   33    2]
 [   0    0   13 7158   58]
 [   0    0    0    4 4942]]


classification report
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         1
           2       1.00      0.99      1.00       209
           3       0.99      0.98      0.98      1597
           4       0.99      0.99      0.99      7229
           5       0.99      1.00      0.99      4946

    accuracy                           0.99     13982
   macro avg       0.99      0.99      0.99     13982
weighted avg       0.99      0.99      0.99     13982

==============================
```

We will proceed with creating the `GradientBoostingClassifier` model for further use.

```
[ ] # Model creation
    # Model initialization
    department_classification_model=GradientBoostingClassifier()

    # Fitting the model
    department_classification_model.fit(X_train,y_train)

    # Predicting using the model
    department_classification_pred=department_classification_model.predict(X_test)

    # Printing the confusion metrics and classification report
    print('metrics on test data')
    print('confusion matrix')
    print(confusion_matrix(y_test,department_classification_pred))
    print('\n')
    print('classification report')
    print(classification_report(y_test,department_classification_pred))
    print('==='*10)
```

```
metrics on test data
confusion matrix
[[   0    0    0    0    0    0    0    0    0    0    0    0]
 [   0 9127    0    0    0    0    0    0    0    0    0  776]
 [   0    0    1    0    0    0    0    0    0    0    0    0]
 [   1    0    0 1092    0    0    0    0    0    0    0    0]
 [   0    0    0    0   63    0    0    1    0    0    0    0]
 [   0    0    0    0    0   64    0    0    0    0    0    0]
 [   0    0    0    1    0    0  132    0    0    0    0    0]
 [   0    0    0    2    1    0    0   26    0    0    3    0]
 [   0    0    0    0    0    0    0    0   46    0    0    0]
 [   0    0    0    0    0    0    0    0    0  100    0    0]
 [   0    1    0    0    0    1    0    0    0    0  209    0]
 [   0 1400    0    0    0    0    0    0    0    0    0  935]]


classification report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.87      0.92      0.89      9903
           2       1.00      1.00      1.00         1
           3       1.00      1.00      1.00      1093
           4       0.98      0.98      0.98        64
           5       0.98      1.00      0.99        64
           6       1.00      0.99      1.00       133
           7       0.96      0.81      0.88        32
           8       1.00      1.00      1.00        46
           9       1.00      1.00      1.00       100
          10       0.99      0.99      0.99       211
          11       0.55      0.40      0.46      2335

    accuracy                           0.84     13982
   macro avg       0.86      0.84      0.85     13982
weighted avg       0.83      0.84      0.83     13982

==============================
```

! Refer Page 36 for Metrics Definitions

Task 4

Predict Requests for Change (RFC) and potential failures or misconfigurations of ITSM (IT Service Management) assets.

Data Type Conversion

1. Convert categorical columns to the `'object'` data type.
2. Convert numerical columns to `'float'` or `'int'` data types, using coercion to handle errors for invalid parsing.

```
+ Code   + Text
[ ] # Define categorical,numerical columns:
    categorical_columns = ['CI_Subcat', 'WBS', 'Category']

    numerical_columns=['Priority',
                       'No_of_Related_Interactions', 'No_of_Related_Incidents','No_of_Related_Changes']


    # Convert categorical columns to 'object' dtype
    df_4[categorical_columns] = df_4[categorical_columns].astype('object')

    # Convert numerical columns to 'float' or 'int' dtype (coerce errors for invalid parsing)
    df_4[numerical_columns] = df_4[numerical_columns].apply(pd.to_numeric, errors='coerce')
```

After removing replacing the Null Values with .median() of specific column

```
df_4.head()
```

	CI_Subcat	WBS	Priority	Category	No_of_Related_Interactions	No_of_Related_Incidents	No_of_Related_Changes
0	Web Based Application	WBS000162	4.0	incident	1.0	2.0	1.0
1	Web Based Application	WBS000088	3.0	incident	1.0	1.0	1.0
2	Desktop Application	WBS000092	4.0	request for information	1.0	1.0	1.0
3	Web Based Application	WBS000088	4.0	incident	1.0	1.0	1.0
4	Web Based Application	WBS000088	4.0	incident	1.0	1.0	1.0

```
df_4["No_of_Related_Changes"].value_counts()
```

```
No_of_Related_Changes
1.0    46582
2.0       21
3.0        2
9.0        1
Name: count, dtype: int64
```

We observe that the column `No_of_related_changes` contains values 3.0 and 9.0, each with only 2 and 1 occurrence, respectively.

```
df_4.drop(df_4.loc[df_4['No_of_Related_Changes']==3.0].index,inplace=True)
# As there were 2 records in this category.

df_4.drop(df_4.loc[df_4['No_of_Related_Changes']==9.0].index,inplace=True)
# As there was only 1 record in this category.
```

```
df_4["No_of_Related_Changes"].value_counts()
```

```
No_of_Related_Changes
1.0    46582
2.0       21
Name: count, dtype: int64
```

Handling Imbalanced Data

1. **Data Imbalance**: The data in our dependent feature `No_of_Related_Changes` is highly imbalanced.
2. **Oversampling Solution**: To address this issue, we will use oversampling to balance the data.
3. **RandomOverSampler**: We will use the `RandomOverSampler` to handle the imbalance in the data.

This approach helps ensure that the classes in the dependent feature are balanced, improving the performance of the predictive model.

After balancing data :

```
y4_res.value_counts()
```

```
No_of_Related_Changes
1.0     46582
2.0     46582
Name: count, dtype: int64
```

```
# Splitting the Data into train and test for calculating the accuracy

X4_train, X4_test, y4_train, y4_test =
train_test_split(X4_res,y4_res,test_size=0.3,random_state=10)

# Standardization technique is used

sc = StandardScaler()

X4_train = sc.fit_transform(X4_train)

X4_test = sc.transform(X4_test)
```

Model Creation and Evaluation

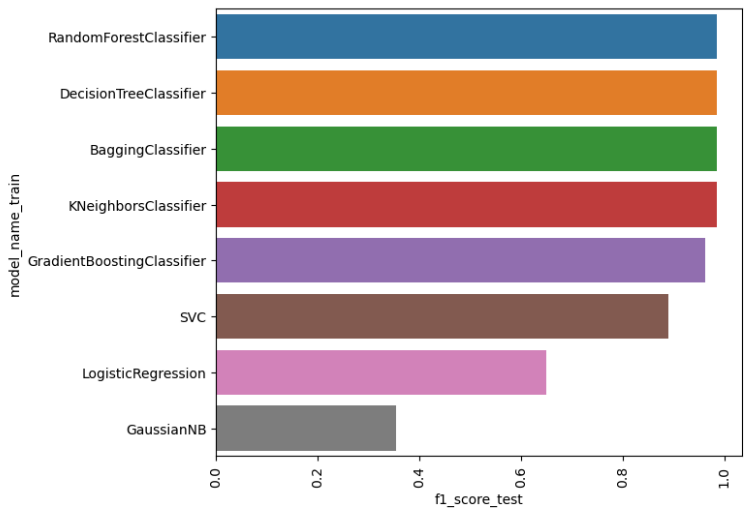
```
model_summary_4={'model_name_train':[],'f1_score_train':[],'recall_score_train':[],'accuracy_score_train':[],
               'model_name_test':[],'f1_score_test':[],'recall_score_test':[],'accuracy_score_test':[]}


def model_selction_4(model):

    # Model initialization ,fitting and predicting
    print(model)
    model=model()
    model.fit(X4_train,y4_train)
    model_pred=model.predict(X4_test)

    # Appending the metrics to the dictionary created
    model_summary_4['model_name_test'].append(model.__class__.__name__)
    model_summary_4['f1_score_test'].append(f1_score(y4_test,model_pred,average='macro'))
    model_summary_4['recall_score_test'].append(recall_score(y4_test,model_pred,average='macro'))
    model_summary_4['accuracy_score_test'].append(accuracy_score(y4_test,model_pred))

    # Printing the confusion metrics and classification report
    print('metrics on test data')
    print(confusion_matrix(y4_test,model_pred))
    print('\n')
    print(classification_report(y4_test,model_pred))

    # Predictions on train data
    model_pred1=model.predict(X4_train)

    # Appending the metrics to the dictionary created
    model_summary_4['model_name_train'].append(model.__class__.__name__)
    model_summary_4['f1_score_train'].append(f1_score(y4_train,model_pred1,average='macro'))
    model_summary_4['recall_score_train'].append(recall_score(y4_train,model_pred1,average='macro'))
    model_summary_4['accuracy_score_train'].append(accuracy_score(y4_train,model_pred1))

    # Printing the confusion metrics and classification report
    print('metrics on train data')
    print(confusion_matrix(y4_train,model_pred1))
    print('\n')
    print(classification_report(y4_train,model_pred1))
    print('==='*10)
```

Summary of Chosen Metrics for All Models

```
[ ] summary_4=pd.DataFrame(model_summary_4).sort_values('f1_score_test',ascending=False).drop('model_name_test',axis=1)
```

```
[ ] summary_4
```

	model_name_train	f1_score_train	recall_score_train	accuracy_score_train	f1_score_test	recall_score_test	accuracy_score_test
2	RandomForestClassifier	0.984893	0.984915	0.984896	0.984933	0.984893	0.984937
1	DecisionTreeClassifier	0.984893	0.984915	0.984896	0.984753	0.984714	0.984758
3	BaggingClassifier	0.984878	0.984900	0.984881	0.984718	0.984678	0.984723
4	KNeighborsClassifier	0.984801	0.984823	0.984804	0.984718	0.984678	0.984723
7	GradientBoostingClassifier	0.962382	0.962479	0.962431	0.961257	0.961210	0.961324
6	SVC	0.892235	0.892708	0.892630	0.890462	0.890704	0.890877
0	LogisticRegression	0.647724	0.651898	0.651762	0.650118	0.653996	0.654311
5	GaussianNB	0.351604	0.508469	0.507851	0.354848	0.509545	0.510984



Model Selection for Task 4

From the analysis, it is observed that the `RandomForestClassifier`, `BaggingClassifier`, `DecisionTreeClassifier`, and `KNeighborsClassifier` models are performing well compared to other algorithms, achieving performance above 95%. Therefore, additional optimization techniques are not required.

We are selecting the `RandomForestClassifier` over the `BaggingClassifier`, `DecisionTreeClassifier`, and `KNeighborsClassifier` because it has consistently demonstrated superior performance.

We will proceed with developing the `RandomForestClassifier` model for further use.

```
[ ]  # Model creation
     # Model initialization
     category_classification_model=RandomForestClassifier()

     # Fitting the model
     category_classification_model.fit(X4_train,y4_train)

     # Predicting using the model
     category_classification_pred=category_classification_model.predict(X4_test)

     # Printing the confusion metrics and classification report
     print('metrics on test data')
     print('confusion matrix')
     print(confusion_matrix(y4_test,category_classification_pred))
     print('\n')
     print('classification report')
     print(classification_report(y4_test,category_classification_pred))
     print('==='*10)
```

```
metrics on test data
confusion matrix
[[13512   422]
 [    0 14016]]


classification report
              precision    recall  f1-score   support

         1.0       1.00      0.97      0.98     13934
         2.0       0.97      1.00      0.99     14016

    accuracy                           0.98     27950
   macro avg       0.99      0.98      0.98     27950
weighted avg       0.99      0.98      0.98     27950

==============================
```

Files related to project

1. PRCL-0012.pdf
2. Project Code Notebook

Team Members Details

PTID-CDS-MAR-24-1870 Team Info:

1. Dipanjali Patra

- (01-MAY-23-CDS-ONL-BUN-021-WDE20)
- email: satyas.behera@gmail.com /dipanjalipatra@gmail.com

2. Vinay C

- (09-Oct-23-CDS-BUN-021-WDE20-ONL)
- email: vinayvinay9617@gmail.com

3. Sandeep Chandra Sagar R

- (09-Oct-23-CDS-BUN-021-WDE20-ONL)
- email: sanwithdeep@gmail.com

****