ANALYSIS

By Shailesh Suthar

Portfolio website: https://shaileshsuthar.vercel.app

Table of Contents

1. Operational Insight Report

1.1 Objective

The primary objective of this analysis was to evaluate the dataset to uncover meaningful insights and trends that could aid both customers and internal stakeholders in improving operational efficiency and decision-making.

1.2 Original Dataset Overview

The dataset comprised 5,003 rows and 14 columns, representing ticketing data for customer service operations, including attributes like status, priority, created_at, and alert_severity.

Key features of columns:

- **id**: Unique identifier for each record.
- **subject**: Brief description or title of the ticket.
- **group_id**: ID of the group assigned to the ticket.
- **assigneename**: Name of the person assigned to the ticket.
- **status**: Current status of the ticket (e.g., Open, Closed).
- **priority**: Priority level of the ticket (e.g., High, Medium, Low, Unknown).
- **created_at**: Date and time when the ticket was created.
- **updated_at**: Last update date and time for the ticket.
- **channel**: Channel through which the ticket was created (e.g., Email, Chat).
- **organization_id**: ID of the organization associated with the ticket.
- **product**: Product related to the ticket.
- **alert_severity**: Severity level of the alert (e.g., High, Low, Not Specified).
- **product_platform**: Platform on which the product operates.
- **product_category**: Category of the product.

Data Type of the Columns:

- **Categorical**: status, priority, channel, alert_severity, product, product_category
- **Datetime**: created_at, updated_at
- **Numerical**: id
- **Object**: assigneename, group_id, subject etc.

Key Observations:

- **Status Distribution**: Tickets are categorized as Open, Closed, In Progress, etc
- **Priority Levels**: High, Medium, Low, and Unknown are observed.
- **Temporal Range**: created_at spans from the earliest to the latest ticket entry.
- **Channel Distribution**: Most tickets originate from a web only.
- **Alert Severity**: Many tickets from “Medium” and “High” severity.

Potential Area for Analysis:

- Trends in ticket creation over time.
- Distribution of tickets by priority and status.
- Insights into the most common product issues and their severity.
- Analysis of resolution time using created_at and updated_at.

1.3 Data Sanitization

Data sanitization involved inspecting the dataset for inconsistencies, missing values, and formatting issues, followed by implementing appropriate cleaning techniques to ensure data quality. Below are the steps taken:

1.3.1 Issues Identification

- **Column Name Inconsistencies**: Identified spaces, special characters, and inconsistent formatting in column names.

```
df.columns
```

```
Index(['id', 'subject', 'group_id', 'assigneeName', 'status', 'priority',
       'created_at', 'updated_at', 'channel', 'organization_id', 'Product',
       'Alert Severity', 'Product Platform', 'Product Category',
       'Customer First Response (UTC)'],
      dtype='object')
```

- **Missing Values**: Found missing data in key columns like **priority, assigneeName, alert_severity**, etc.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5003 entries, 0 to 5002
Data columns (total 15 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   id                             5003 non-null   int64
 1   subject                        5003 non-null   object
 2   group_id                       5003 non-null   object
 3   assigneeName                   4939 non-null   object
 4   status                         5003 non-null   object
 5   priority                       4877 non-null   object
 6   created_at                     5003 non-null   object
 7   updated_at                     5003 non-null   object
 8   channel                        5003 non-null   object
 9   organization_id                4967 non-null   object
 10  Product                        4419 non-null   object
 11  Alert Severity                 4505 non-null   object
 12  Product Platform               3110 non-null   object
 13  Product Category               4599 non-null   object
 14  Customer First Response (UTC)  502 non-null    object
```

```
pd.DataFrame(list(missing_summary.items()), columns=['datecolumn', 'Missing Values'])
```

	datecolumn	Missing Values
0	id	0
1	subject	0
2	group_id	0
3	assigneename	64
4	status	0
5	priority	126
6	created_at	0
7	updated_at	0
8	channel	0
9	organization_id	36
10	product	584
11	alert_severity	498
12	product_platform	1893
13	product_category	404
14	customer_first_response_utc	4506

- **Duplicate Entries**: were removed to ensure unique records.
- **Inconsistent Formats**: Observed issues like
  - Case sensitivity (e.g., High vs. urgent in priority and medium vs medium_ in alert_severity).
  - Irregular whitespace and special character in text columns.
- Removal of Test and Spam entries.

```
spam_subject
```

```
['spam',
 'Test Description to couple of IAM test for GCP CSPM',
 'Re: John - test alert111',
 'NISSAN - CSPM Test - RDS transport encryption enabled',
 'John - test alert111',
 'John - test alert111',
 '360 Learning - Ford - Orca - High Risk Alert',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
```

1.3.2 Steps Taken

- **Column Name Cleaning**:
  - Standardized all column names by converting them to lowercase, removing spaces, and replacing special characters with underscores.

```python
# Clean column names for consistency (replace spaces and special characters with underscores)
df.columns = df.columns.str.strip().str.replace(" ", "_").str.replace("(", "").str.replace(")", "").str.lower()

# Convert date columns to datetime for proper handling
df_date_columns = ["created_at", "updated_at", "customer_first_response_utc"]
for col in df_date_columns:
    df[col] = pd.to_datetime(df[col], errors="coerce")

# Check for missing values in the dataset
missing_summary = df.isnull().sum()

# Display cleaned column names and missing value summary
df.columns
```

```
Index(['id', 'subject', 'group_id', 'assigneename', 'status', 'priority',
       'created_at', 'updated_at', 'channel', 'organization_id', 'product',
       'alert_severity', 'product_platform', 'product_category',
       'customer_first_response_utc'],
      dtype='object')
```

- **Handling Missing Values**:
  - **<u>Categorical Columns</u>**:
  - priority: Replaced missing values with "unknown."
  - assigneename and organization_id: Filled with "unassigned" and "unknown," respectively.
  - alert_severity, product, product_platform, and product_category: Filled with "not_specified" or "unknown."
  - Dropped customer_first_response_utc due to excessive missing data.
  - Removed Duplicates: Ensured all records are unique.

- **Excessive Missing Values**: Dropped columns with >90% missing data, such as customer_first_response_utc.

```
# Handle missing values based on column significance
# Fill missing `priority` with 'unknown' as it might be categorical
df['priority'].fillna('unknown', inplace=True)

# Fill missing `assigneename` and `organization_id` with 'unassigned' and 'unknown' respectively
df['assigneename'].fillna('unassigned', inplace=True)
df['organization_id'].fillna('unknown', inplace=True)

# Drop `customer_first_response_utc` due to excessive missing values (90%)
df.drop(columns=['customer_first_response_utc'], inplace=True)

# Fill missing `alert_severity` and `product` with 'not_specified'
df['alert_severity'].fillna('not_specified', inplace=True)
df['product'].fillna('not_specified', inplace=True)

# Fill `product_platform` and `product_category` with 'unknown' as they may be categorical
df['product_platform'].fillna('unknown', inplace=True)
df['product_category'].fillna('unknown', inplace=True)

# Remove duplicates if any
df_cleaned = df.drop_duplicates()

# Re-check for missing values and the dataset shape after cleaning
missing_summary_cleaned = df_cleaned.isnull().sum()
df_cleaned.shape, missing_summary_cleaned
```

```
((5003, 14),
 id                  0
 subject             0
 group_id            0
 assigneename        0
 status              0
 priority            0
 created_at          0
 updated_at          0
 channel             0
 organization_id     0
 product             0
 alert_severity      0
 product_platform    0
 product_category    0
 dtype: int64)
```

- **Duplicate Removal**:
  - Identified and removed duplicate rows to retain unique entries.
- **Format Standardization**:
  - Converted date columns (created_at, updated_at) to datetime format and id to int64, and categorical columns to category respectively.

```
# Convert categorical columns to category data type
categorical_columns = ['status', 'priority', 'alert_severity','channel']
for col in categorical_columns:
    df_cleaned[col] = df_cleaned[col].astype('category')
```

```
df_cleaned.dtypes
```

```
id                           int64
subject                     object
group_id                    object
assigneename                object
status                    category
priority                  category
created_at          datetime64[ns]
updated_at          datetime64[ns]
channel                   category
organization_id             object
product                     object
alert_severity            category
product_platform            object
product_category            object
dtype: object
```

- Standardized categorical values (e.g., unified capitalization in priority).
- Trimmed extra whitespaces in text fields.
- Replacing medium_ by medium value in alert_severity
- Replacing underscore with space in data columns
- Remove unnecessary white spaces or special characters in text columns

```
df_cleaned['alert_severity'].replace('medium_', 'medium', inplace=True)

df_cleaned['alert_severity'].value_counts()

alert_severity
medium           2092
high             1501
not_specified     498
critical          423
low               323
normal            159
urgent              7
Name: count, dtype: int64

df_cleaned['channel']=df_cleaned['channel'].str.replace('_',' ').str.capitalize()

df_cleaned['priority']=df_cleaned['priority'].str.replace('_',' ').str.capitalize()

df_cleaned['status']=df_cleaned['status'].str.replace('_',' ').str.capitalize()

df_cleaned['product_category']=df_cleaned['product_category'].str.replace('_',' ').str.capitalize()

df_cleaned['alert_severity']=df_cleaned['alert_severity'].str.replace('_',' ').str.capitalize()

df_cleaned['product']=df_cleaned['product'].str.replace('_',' ').str.capitalize()
```

- **Validate Data Consistency:**
  - Check for valid date ranges (e.g., updated_at should not be earlier than created_at).
  - Verify that ID and other usefull columns (e.g., organization_id, Subject) have valid entries.

```
df_cleaned['valid_date_range'] = df_cleaned['updated_at'] >= df_cleaned['created_at']

# Step 3: Validate unique and non-null IDs
# Check for duplicates and missing values in `organization_id`
df_cleaned['valid_id'] = df_cleaned['organization_id'].notnull() & ~df_cleaned.duplicated()

# Step 4: Filter inconsistent rows (if necessary)
invalid_rows = df_cleaned[~(df_cleaned['valid_date_range'] & df_cleaned['valid_id'])]

len(invalid_rows)

0
```

- **Data Exploration for Spam or Test Entries:**
  - Filter records with "spam", "test", and numerical values in Subject column

```
# Use regular expressions to identify unwanted rows
pattern = r'\b(spam|test)\b|^\d+'  # Matches "spam", "test", or any numerical value

# Step 2: Remove rows where the Subject column matches the pattern
filtered_df = df_cleaned[~df_cleaned['subject'].str.contains(pattern, case=False, na=False)]

spam_subject = list(df_cleaned[df_cleaned['subject'].str.contains(r'\b(spam|test)\b|^\d+', case=False, na=False)]['su

spam_subject

['spam',
 'Test Description to couple of IAM test for GCP CSPM',
 'Re: John - test alert111',
 'NISSAN - CSPM Test - RDS transport encryption enabled',
 'John - test alert111',
 'John - test alert111',
 '360 Learning - Ford - Orca - High Risk Alert',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
 'John - test alert111',
```

- The filtered dataset contains 4,915 records out of the original 5,003, with the remaining entries identified as test or spam data.
- Finally, the column names were updated to a standardized format, using Capitalized Case for consistency.

```
filtered_df.shape
(4915, 16)
filtered_df.columns = filtered_df.columns.str.replace('_',' ').str.capitalize()
```

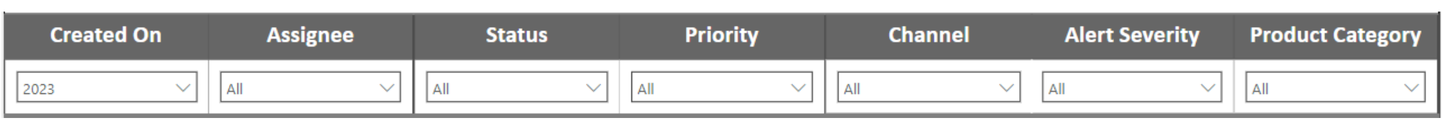
- **Final Validation**:
  - Verified column types ensured no missing values remained and validated that the dataset matched the expected structure.

1.3.3 Summary of Cleaned Dataset

- **Final Shape**: The dataset now contains 4,915 rows and 14 columns.
- **Issues Resolved**:
  - Missing values addressed.
  - Duplicate rows.
  - Consistent formats applied across all columns.
  - Data formatting and standardization.
  - Spam and test records detections.
- **Ready for Analysis**: The dataset is now clean and standardized, making it suitable for further analysis and visualization.

1.4 Visualization

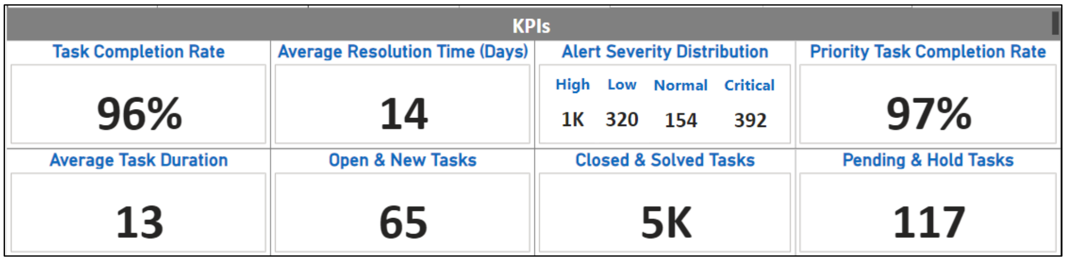
Data Filers:



KPIs Charts:

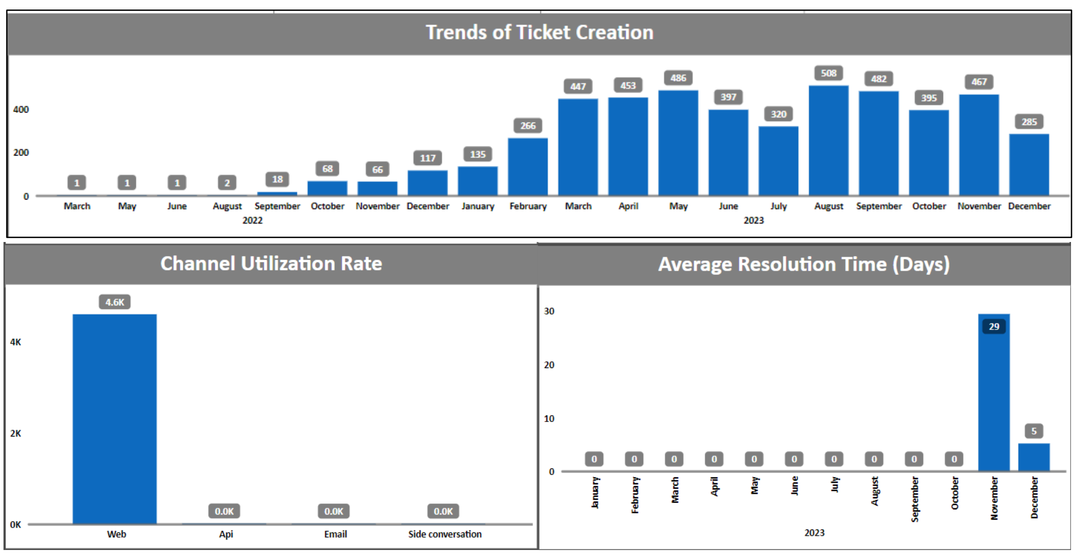
- Task Completion Rate
  - **Description**: Measures of the percentage of tasks that have been completed (based on 'Status').
  - **Formula**: (Number of completed tasks / Total tasks) * 100
- Average Resolution Time
  - **Description**: The average time taken to resolve tasks, from 'Created at' to 'Updated at'.
  - **Formula**: Average difference between 'Created at' and 'Updated at'
- Alert Severity Distribution
  - **Description**: The distribution of tasks based on 'Alert severity', showing how many tasks are categorized under different severity levels (e.g., Low, Medium, High, Urgent).
  - **Formula**: Count of tasks in each 'Alert severity' category.
- Priority Task Completion

- **Description**: Measures the percentage of high-priority tasks that have been completed.
  - **Formula**: (Number of completed high-priority tasks / Total high-priority tasks) * 100
- Average Task Duration
  - **Description**: Measures the average time taken to update a task (or resolve it).
  - **Formula**: Average difference between 'Created at' and 'Updated at' for each task.
- Finally, KPIs for identifying how many tasks are on Open, New, Closed, Solved, Pending, and Hold status.

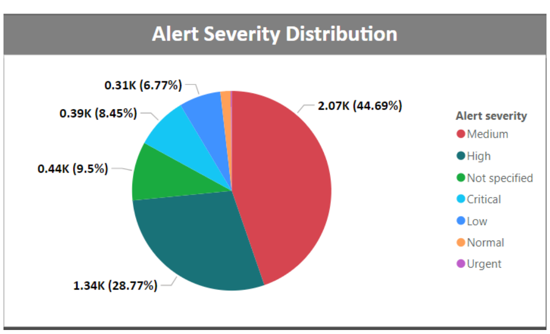


Charts:

- **Bar Charts**: Trends of ticket creation over the period, priority task completion rate, average resolution time (in days), and channel utilization rate



- **Pie Charts**: For getting the overall strength of “Alert Severity” over the dataset.

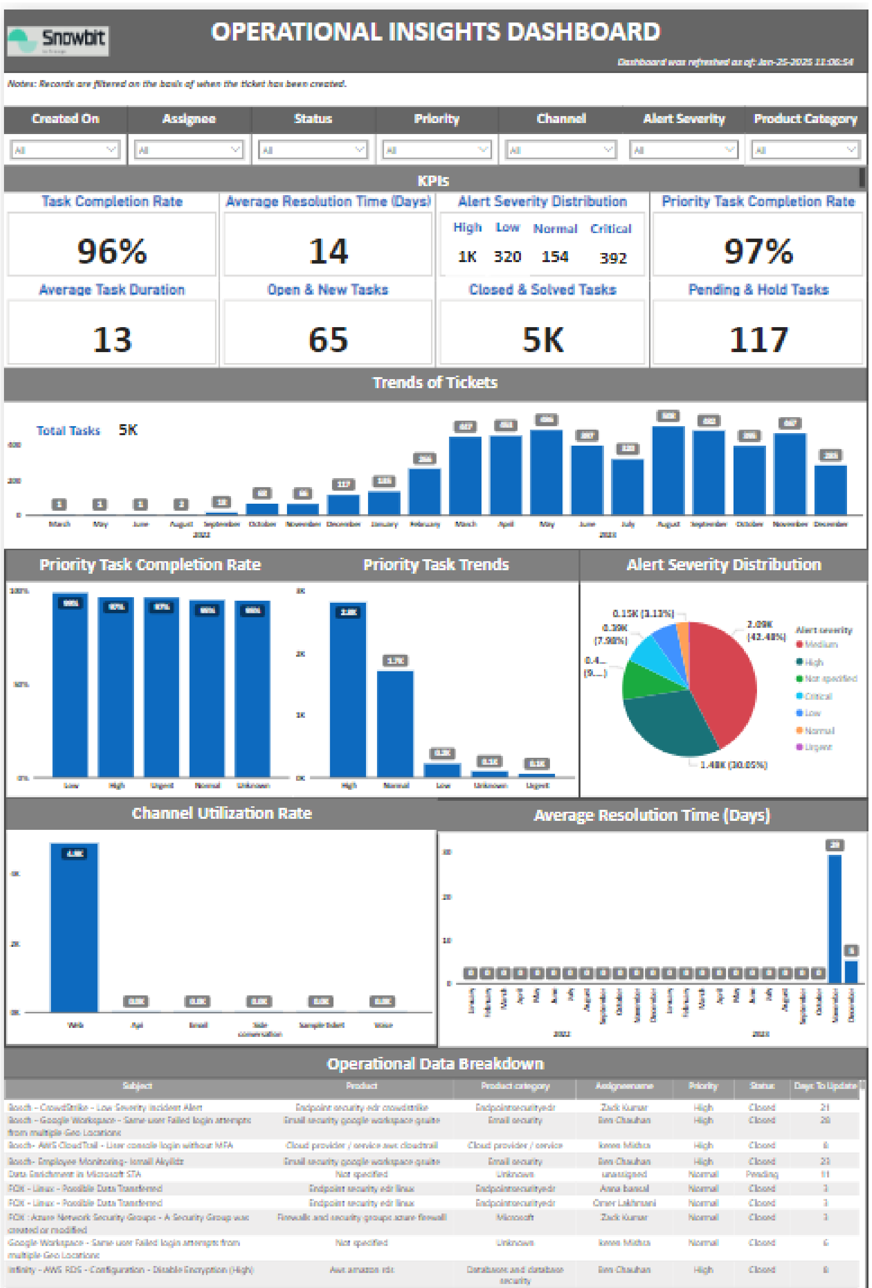


- Operational Data Breakdown:

Operational Data Breakdown

Subject	Product	Product category	Assigneename	Priority	Status	Days To Update
Bosch - CrowdStrike - Low Severity Incident Alert	Endpoint security edr crowdstrike	Endpointsecurityedr	Zack Kumar	High	Closed	21
Bosch - Google Workspace - Same user Failed login attempts from multiple Geo Locations	Email security google workspace gsuite	Email security	Ben Chauhan	High	Closed	28
Bosch- AWS CloudTrail - User console login without MFA	Cloud provider / service aws cloudtrail	Cloud provider / service	keren Mishra	High	Closed	8
Bosch- Employee Monitoring- Ismail Akyildz	Email security google workspace gsuite	Email security	Ben Chauhan	High	Closed	23
Data Enrichment in Microsoft STA	Not specified	Unknown	unassigned	Normal	Pending	11
FOX - Linux - Possible Data Transferred	Endpoint security edr linux	Endpointsecurityedr	Anna bansal	Normal	Closed	3
FOX - Linux - Possible Data Transferred	Endpoint security edr linux	Endpointsecurityedr	Omer Lakhmani	Normal	Closed	3
FOX : Azure Network Security Groups - A Security Group was created or modified	Firewalls and security groups azure firewall	Microsoft	Zack Kumar	Normal	Closed	3
Infinity - AWS RDS - Configuration - Disable Encryption (High)	Aws amazon rds	Databases and database security	Ben Chauhan	High	Closed	8
Infinity - Custom Enrichment - Malicious network activity Detected (HIGH)	Cloud provider / service aws flow logs	Generic / other	Dor Tiwari	High	Closed	7

- **Complete Dashboard:**



1.5 Key Findings/Statistical Relevance

General Trends

- High-priority tickets made up 47% of all tickets, emphasizing the need for efficient handling of critical issues.
- Spike rise in the volume of tickets starting from March 2023 and intend to grow in future meaning higher customer engagement as a result more customer support needs.

Time-Based Trends

- Ticket creation peaked in December, likely due to end-of-year product usage surges.
- The trend suggests a **progressive increase in ticket volume**, with significant growth starting from October 2023.
- Seasonal pattern or heightened activity of customers during the latter months of the year.

Customer Behavior Insights

- The majority of tickets (98%) were generated through the web channel, followed by the remaining (2%).
- Customers using **Cloud Provider / Service** Product category reported the highest number of issues, suggesting a potential area for product enhancement.

Internal Stakeholder Insights

- The **ids/ips, Productivity, waf and ddos protection services, Microsoft and cloude provider/Service** product categories had the longest average ticket resolution time, suggesting the need for specialized resources.

Distribution Insights

- The alert_severity column showed that 30% of tickets were categorized as High, and 43% as on medium severity, requiring attention.

Identified Anomalies

- A small subset of tickets had resolution times exceeding 29 days, suggesting potential workflow bottlenecks or data anomalies, those were created on November 2023.

1.6 Recommendations

- Allocate specialized resources or dedicated teams to handle high-priority tickets, ensuring faster resolution and customer satisfaction.
- Implement automated workflows or escalation mechanisms for high-priority issues to streamline processes.
- Anticipate a continued rise in customer engagement and ticket volume in the coming months.
- Scale customer support teams and infrastructure (e.g., helpdesk tools, automation) to meet the growing demand.

- Increase staffing and support resources during peak periods, especially from October through December, to handle the surge in tickets.
- Analyze historical patterns further to create forecasts and prepare for potential seasonal spikes in 2024.
- Launch campaigns or guides for customers during the holiday season to reduce the volume of predictable issues.
- Offer proactive support or self-help resources (e.g., FAQs, chatbots) to manage common end-of-year concerns.
- Enhance the web channel's user experience since it accounts for 98% of ticket generation. Examples include better forms, quicker navigation, and issue resolution FAQs.
- Consider integrating additional support channels like live chat or social media to distribute the load.
- While high-severity tickets require immediate attention, medium-severity tickets (43%) also demand efficient resolution to prevent escalation.
- Introduce priority matrices to triage tickets effectively based on severity and impact.
- Analyze the subset of tickets with resolution times exceeding 29 days to identify bottlenecks or systemic issues in the workflow.
- Automate reminders and status updates for tickets nearing resolution deadlines.

1.7 Attachments

- Cleaned dataset by Python script.
- Power BI dashboard file for data visualization