D214 – Data Analytics Graduate Capstone

Data Analytics Report

Western Governors University

December 27, 2024

Kevin Rupe
Student ID: XXXXXXX
Email: krupe6@wgu.edu
Phone: (864) 704-2340

RESEARCH QUESTION

The research question posed on this project is whether certain variables related to an alarm on a customer network device have a high correlation to the alarm itself being cleared. Internet Service Providers (ISP) have an objective to deliver fast and reliable service to their customers. In our ever-evolving world of technology with the exponentially expanding Internet of Things (IoT), as a society in the United States we have more devices online than ever before. In today's times, we all rely on our devices' connectivity to the internet in our day-to-day lives.

Having quality, fast, and reliable internet service is no exception to business customers who depend and rely on them in order for their businesses to operate efficiently (Torkildson, n.d.). When businesses are offline, they often can't function. This hampers their ability to run their business and ultimately, this costs them money. This can have a ripple effect on their customers as well.

For ISPs that are providing internet service to a business, it is imperative that when their customers are experiencing service affecting events that they be repaired and/or services are restored quickly, efficiently, and correctly. Getting their customers back online is the most important thing, and time is of the essence. Afterall, if the ISP struggles to provide excellent service to their customers, then in the end, their customers will find another ISP who can. Therefore, this has a negative impact on the ISPs revenue.

Businesses usually have multiple advanced network configurations deployed on their premises. Whenever one of these devices goes offline, an alarm case is created. By applying a Multiple Linear Regression model, we can gain insights into what factors might have a high correlation to the alarm clearing (Massaron, 2016). These insights would provide extremely useful intel to the ISP at helping them streamline processes, enhance their decision making, allocate resources, and even provide predictive insights. All of which would drive the company towards providing a better customer experience.

The null hypothesis of this research question is that there will not be any significant correlation in the related variables to the alarm being cleared. The alternative hypothesis is that there will be at least one variable that has significant correlation to the alarm clearing. If any variables are found to have significant correlation to the alarm clearing, then it would explain why the alarm cleared.

The alarm clearing means that the service affecting issue has been resolved. To understand on an analytical level why the alarm cleared would be very useful and valuable information to the ISP in determining how best to treat future service affecting issues, and possibly even how to proactively prevent them where they can. This ultimately would provide a better customer experience which would have a direct positive impact on ISP revenue.

DATA COLLECTION

When an alarm occurs on a customer's device an alarm case is generated. Each case has many variables that relate to a specific device that has an alarm event. Therefore, to determine if there are any factors that have a significant correlation to the alarm being cleared, we need to gather all of the variables on the alarm case. The dataset from the ISP was privately obtained, and contains 30 variables and 124,152 observations. The dataset has variables of Case_Duration, CreatedDate, ClosedDate, City, and State. The predictor variables are broken down and explained below:

Field	Type	Description
Arbitrary_ID	Nominal Categorical	Arbitrarily created ID to mask any sensitive information
Service_POD_Number	Nominal Categorical	Unique group number indicating the service level team that will handle the case
Customer_Comments	Discrete Quantitative	The number of times the customer interacted with the case
Assigned_Queue	Nominal Categorical	The queue in which the case is assigned
Origin	Nominal Categorical	The originating system of the case
Product	Nominal Categorical	The product of the service affecting issue
Priority	Ordinal Categorical	The priority of the case
Primary_Customer_Condition	Nominal Categorical	The condition of the customer
Service_Level	Ordinal Categorical	The service level of the customer
Customer_Impact	Ordinal Categorical	The impact level of the service affecting issue
Open_Tier_1	Nominal Categorical	First level tier related to the device and the alarm
Open_Tier_2	Nominal Categorical	Second level tier related to the device and the alarm
Open_Tier_3	Nominal Categorical	Third level tier related to the device and the alarm
Alarm_Status	Nominal Categorical	Clear (indicates the alarm has been resolved) Active (indicates the issue is still ongoing)
Issue	Nominal Categorical	Service is Down (indicates the customer has no service) Service is Impaired (indicates the customer has partial service)
Service_Status	Nominal Categorical	Out of Service (indicates that the customer is out of service) In Service (indicates that the customer is not out of service)
Closing_Fault	Nominal Categorical	The fault of the alarm
Closing_Issue	Nominal Categorical	The issue of the alarm
Closing_Resolution	Nominal Categorical	The resolution of the alarm
Action_Code	Nominal Categorical	The action of the alarm
Cause_Code	Nominal Categorical	The cause of the alarm
Class_Item_Code	Nominal Categorical	The class item of the alarm

One big advantage to using this dataset is the fact that I was able to pull the data directly from the source itself, which was located in a Snowflake database directly at the ISP. I used SQL to explore the alarm case table and all of its related column variables. Some of the data was missing, and since I had ample observations of data to use, I simply chose to remove any null values from the query before extracting the data into a dataset. I also chose to only find alarm cases that were closed in the calendar year 2024. Upon inspection, I noticed that several City and State names seemed to be bad data, so I decided to exclude any City name that did not begin with an alphabet letter, and I wrote an inclusive statement to only find States in the USA, Canada and US regions (e.g., Puerto Rico, Guam, Virgin Islands, etc.).

One disadvantage with this dataset is the fact that all the variables only point to what is known by the ISP. There are many factors that can cause an alarm that are not easily identifiable. For example, the internet is connected by all sorts of different ISPs, and the issue may be something related to another carrier's network, and therefore might not be fully known. Other issues with alarms could be related to Utility Company power outages, or even natural disasters.

The SQL query used to gather the data is below:

```
SELECT
    ROW_NUMBER() OVER (ORDER BY id) AS arbitrary_id,
    service_pod_number__c AS Service_POD_Number,
    customer_comments__c AS Customer_Comments,
    city__c AS City,
    state_province__c AS State,
    assigned_queue__c AS Assigned_Queue,
    origin AS Origin,
    CASE WHEN product__c = 'SD-WAN     ' THEN 'SD-WAN V'
        WHEN product__c = 'SD-WAN        ' THEN 'SD-WAN F'
        WHEN product__c = 'SD-WAN     ' THEN 'SD-WAN C'
        WHEN product__c = 'Security-    ' THEN 'Security-C'
        WHEN product__c = 'Security-    ' THEN 'Security-S'
        ELSE product__c END AS Product,
    priority AS Priority,
    primary_customer_condition__c AS Primary_Customer_Condition,
    service_level__c AS Service_Level,
    customer_impact__c AS Customer_Impact,
    case_duration__c AS Case_Duration,
    createddate AS CreatedDate,
    closeddate AS ClosedDate,
    CASE WHEN open_tier_1__c = 'SD-WAN    ' THEN 'SD-WAN V'
        WHEN open_tier_1__c = 'SD-WAN        ' THEN 'SD-WAN F'
        WHEN open_tier_1__c = 'SD-WAN     ' THEN 'SD-WAN C'
        ELSE open_tier_1__c END AS Open_Tier_1,
    open_tier_2__c AS Open_Tier_2,
    open_tier_3__c AS Open_Tier_3,
    alarm_status__c AS Alarm_Status,
    issue__c AS Issue,
    service_status__c AS Service_Status,
    closing_fault__c AS Closing_Fault,
    closing_issue__c AS Closing_Issue,
    closing_resolution__c AS Closing_Resolution,
    action_code__c AS Action_Code,
    cause_code__c AS Cause_Code,
    class_item_code__c AS Class_Item_Code
FROM PROD.MART.          _CASE
```

```sql
WHERE case_record_type_name__c = 'Service_Assurance'
    AND origin LIKE  'Alarm -%'
    AND origin != 'Alarm - ACO'
    AND createddate >= '2024-01-01 00:00:00.000'
    AND customer_division__c = 'ENT'
    AND account_flags__c = 'Strategic'
    AND market_type__c = 'Commercial'
    AND status = 'Closed'
    AND distribution_channel__c IS NOT NULL
    AND service_pod_number__c IS NOT NULL
    AND city__c IS NOT NULL
    AND city__c NOT LIKE '[0-9]%'
    AND state_province__c IN (
        -- USA States
        'AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'FL', 'GA',
        'HI', 'ID', 'IL', 'IN', 'IA', 'KS', 'KY', 'LA', 'ME', 'MD',
        'MA', 'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ',
        'NM', 'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC',
        'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY',
        'DC',
        -- Canadian Provinces and Territories
        'AB', 'BC', 'MB', 'NB', 'NL', 'NS', 'ON', 'PE', 'QC', 'SK',
        'NT', 'NU', 'YT',
        -- Other U.S. Regions
        'PR', 'GU', 'VI', 'MP', 'AS', 'FM', 'MH', 'PW'
        )
    AND product__c IS NOT NULL
    AND primary_customer_condition__c IS NOT NULL
    AND open_tier_1__c IS NOT NULL
    AND open_tier_2__c IS NOT NULL
    AND open_tier_3__c IS NOT NULL
    AND alarm_status__c IS NOT NULL
    AND issue__c IS NOT NULL
    AND closing_fault__c IS NOT NULL
    AND closing_issue__c IS NOT NULL
    AND closing_resolution__c IS NOT NULL
    AND action_code__c IS NOT NULL
    AND cause_code__c IS NOT NULL
    AND class_item_code__c IS NOT NULL
ORDER BY createddate
```

DATA EXTRACTION AND PREPARATION

Once the data has been gathered (above), the next step is to extract the data into a suitable format for reading into our Jupyter Notebook for analysis. The data is therefore extracted into a .csv file and loaded into Jupyter Notebook. Before we can begin, we must also import any relevant packages and libraries that will be necessary for this analysis.

```python
#import important libraries
import numpy as np
import pandas as pd
from pandas import DataFrame

#import statistics libraries
import statistics
from scipy import stats
from statsmodels.formula.api import ols

#import visualization libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings #to ignore warnings
warnings.simplefilter(action='ignore')
warnings.filterwarnings('ignore')
```

```python
#import CSV file
df = pd.read_csv("~/Documents/WGU/D214 - Data Analytics Graduate Capstone/Data Analytics Capstone Project.csv")
```

For this analysis, we will need to drop any columns that are not predictor variables. I also chose to drop each of the “Closing” variables, and each of the “Code” variables to reduce the dimensionality of the dataset. This will help with multicollinearity and will help prevent overfitting of the data (Geeks for Geeks, 2023).

```python
#Removing columns not needed for this exercise
df.drop(['ARBITRARY_ID', 'CITY', 'STATE', 'CASE_DURATION', 'CREATEDDATE', 'CLOSEDDATE',
        'CLOSING_FAULT', 'CLOSING_ISSUE', 'CLOSING_RESOLUTION', 'ACTION_CODE', 'CAUSE_CODE', 'CLASS_ITEM_CODE'],
               axis=1,
               inplace=True)
```

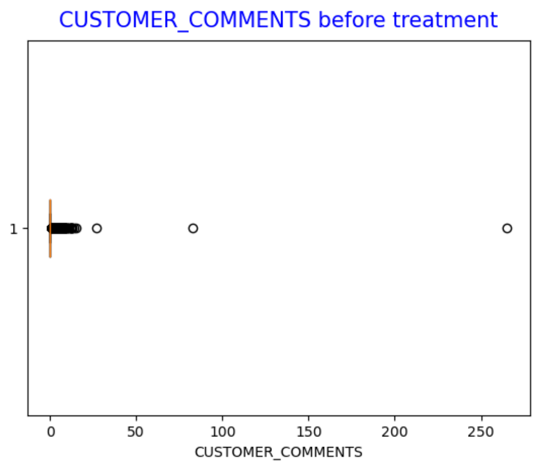
Next, we need to ensure that there are no null rows in this dataset. I did write the SQL query to reduce the possibility of null rows, but it’s still good to double-check anyway. The below output shows that we have no null data.

```python
# Calculate the number of null values in each column
null_counts = df.isnull().sum()

# Filter out columns with no null values (optional)
null_counts = null_counts[null_counts > 0]

# Display the result
print("Columns with null values and their counts:")
print(null_counts)
```

```
Columns with null values and their counts:
Series([], dtype: int64)
```

Looking at the dataset earlier, it seemed like the CUSTOMER_COMMENTS variable was going to have outliers, so to check for this I will visualize this using a boxplot chart from Matplotlib’s pyplot function.

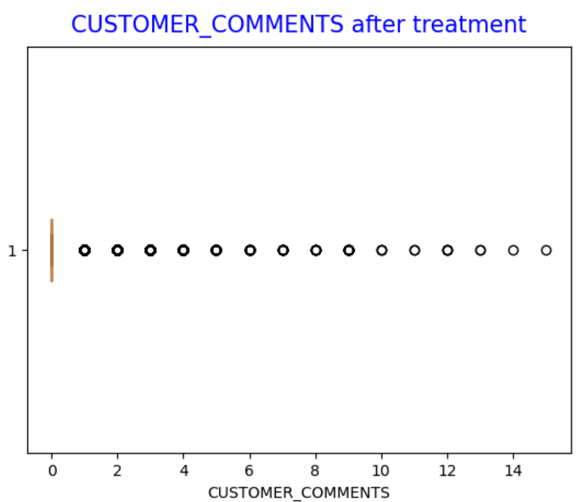
```python
#detect Income outliers before treatment
plt.boxplot(df['CUSTOMER_COMMENTS'], vert=False)
plt.xlabel('CUSTOMER_COMMENTS')
plt.title('CUSTOMER_COMMENTS before treatment', fontsize=15, pad=10, color='blue')
plt.show()
```



Looking at this boxplot graph, I would interpret this as having 3 outliers. Therefore, I will remove these 3. The remainder of the values look bunched very closely to the upper whisker, so even though they are outside the whisker I am going to leave them.

```python
#Treating outliers
df['CUSTOMER_COMMENTS'] = np.where(df['CUSTOMER_COMMENTS'] > 25, np.nan, df['CUSTOMER_COMMENTS'])
df['CUSTOMER_COMMENTS'].fillna(df['CUSTOMER_COMMENTS'].median(), inplace=True)
```

```python
#detect Income outliers after treatment
plt.boxplot(df['CUSTOMER_COMMENTS'], vert=False)
plt.xlabel('CUSTOMER_COMMENTS')
plt.title('CUSTOMER_COMMENTS after treatment', fontsize=15, pad=10, color='blue')
plt.show()
```



MLR requires all of the data to be numeric, therefore, I am going to need to convert most of these variables using One-hot encoding, Ordinal encoding, or use dummies (Pandas, 2024). Upon inspection, these variables will be Origin, Product, Priority, Primary_Customer_Condition, Service_Level, Customer_Impact, Open_Tier_1, Open_Tier_2, Open_Tier_3, Issue, Service_Status, Assigned_Queue, and Alarm_Status.

Statsmodels package in Python will help determine how closely the independent variables correlate to the dependent (Seabold, 2010). The slight drawback to the OLS function is that it requires variables to have no spaces, therefore, before converting these categorical variables into numerical ones, I will need to rename some of the variables to remove any unnecessary spaces in the names. The advantage of OLS far outweighs this slight disadvantage. In a very simple few lines of Python code, OLS will display statistical data that will explain a great deal of data including R-squared, Adjusted R-squared, F-statistic, AIC, BIC, p-values, and many other relevant values.

```
#Define a dictionary to rename ORIGIN

origin_dict = {
    'Alarm - SD WAN': 'SDWAN',
    'Alarm - Customer Netcool': 'Cust_Netcool',
    'Alarm - Core Netcool': 'Core_Netcool',
    'Alarm - SIEM': 'SIEM'
}

#rename values in the ASSIGNED_QUEUE column
df['ORIGIN'] = df['ORIGIN'].replace(origin_dict)
```

```
#Define a dictionary to rename SERVICE_LEVEL

svclvl_dict = {
    'Mid Market': 'Mid'
}

#rename values in the ASSIGNED_QUEUE column
df['SERVICE_LEVEL'] = df['SERVICE_LEVEL'].replace(svclvl_dict)
```

```
#Define a dictionary to rename PRODUCT

product_dict = {
    'LAN Services': 'LAN',
    'SD-WAN C': 'SDWAN_C',
    'SD-WAN F': 'SDWAN_F',
    'SD-WAN V': 'SDWAN_V',
    'Security-Firewall': 'FIREWALL',
    'Security-C': 'Security_C',
    'Security-S': 'SIEM'
}

#rename values in the ASSIGNED_QUEUE column
df['PRODUCT'] = df['PRODUCT'].replace(product_dict)
```

```
#Define a dictionary to rename SERVICE_STATUS

svcst_dict = {
    'Out of Service': 'OOS',
    'In Service': 'IS'
}

#rename values in the ASSIGNED_QUEUE column
df['SERVICE_STATUS'] = df['SERVICE_STATUS'].replace(svcst_dict)
```

```
#Define a dictionary to rename ISSUE

issue_dict = {
    'Service Impairment': 'Impaired',
    'Service Down': 'Down'
}

#rename values in the ASSIGNED_QUEUE column
df['ISSUE'] = df['ISSUE'].replace(issue_dict)
```

```
#Define a dictionary to rename OPEN_TIER_1

ot1_dict = {
    'LAN Services': 'LAN',
    'SD-WAN - Data': 'SDWAN_D',
    'SD-WAN Data': 'SDWAN_D',
    'SD-WAN C': 'SDWAN_C',
    'SD-WAN F': 'SDWAN_F',
    'SD-WAN V': 'SDWAN_V',
    'Security-CSOC': 'Security_CSOC'
}

#rename values in the ASSIGNED_QUEUE column
df['OPEN_TIER_1'] = df['OPEN_TIER_1'].replace(ot1_dict)
```

```
#Define a dictionary to rename OPEN_TIER_2

ot2_dict = {
    'Ethernet Circuit': 'ETH',
    'High Bandwidth': 'HB',
    'LAN - Switch': 'SWITCH',
    'LAN - Access Point': 'AP'
}

#rename values in the ASSIGNED_QUEUE column
df['OPEN_TIER_2'] = df['OPEN_TIER_2'].replace(ot2_dict)
```

Certain variables, such as Assigned_Queue and Open_Tier_3, have many unique options. This will certainly add dimensionality and the possibility of multicollinearity. Therefore, I chose to combine certain factors. I have spent time exploring this data, and my role at my employer gives me a closeup view of this data on a daily basis. I feel my experience affords me the knowledge of performing this task, but I would not do this on just any dataset.

```
#Define a dictionary to rename ASSIGNED_QUEUE -- this will help to reduce dimensionality

queue_dict = {
    'SA Advanced OS Support': 'AdvOSSpt',
    'SA Automation Hold': 'Automation',
    'SA Customer Owned Automation Hold': 'Automation',
    'SA Customer Power Outage': 'Automation',
    'SA Customer Request Automation Hold': 'Automation',
    'SA Disco Automation Hold': 'Automation',
    'SA Order Related Automation Hold': 'Automation',
    'SA Partner Standard Automation Hold': 'Automation',
    'SA SDWAN Automation Hold': 'Automation',
    'SA Simplex Automation Hold': 'Automation',
    'SA Unresponsive Customer Automation Hold': 'Automation',
    'SA Wireless Automation Hold': 'Automation',
    'SA Transport HB': 'TransHB',
    'SA Complex NOC': 'Complex',
    'SA SMB Complex': 'Complex',
    'SA CSOC SASE': 'CSOC',
    'SA CSOC SIEM': 'CSOC',
    'SA CSOC Support': 'CSOC',
    'SA Outage Child': 'Outage',
    'SA Outage Lead': 'Outage',
    'SA Premier Managed Network Solutions': 'MNS',
    'SA Prime Managed Network Solutions': 'MNS',
    'SA Select Managed Network Solutions': 'MNS',
    'SA Proactive Alarm': 'Proactive',
    'SA Proactive Hold': 'Proactive',
    'SA Proactive Support': 'Proactive',
    'SA TSM': 'TSM',
    'SA WSL Event': 'WSL',
    'SA WSL Waves': 'WSL'
}

#rename values in the ASSIGNED_QUEUE column
df['ASSIGNED_QUEUE'] = df['ASSIGNED_QUEUE'].replace(queue_dict)
```

```
#Define a dictionary to rename OPEN_TIER_3

ot3_dict = {
    'Circuit Jitter': 'JL_PL',
    'Circuit Latency': 'JL_PL',
    'Circuit Packet Loss': 'JL_PL',
    'Circuit Proactive Down': 'Pro_Ckt',
    'Circuit Down': 'Ckt_Down',
    'Customer Owned Circuit Degraded': 'Cust_Deg',
    'Customer Owned Circuit Down': 'Cust_Down',
    'Customer Power Outage': 'Cust_Power',
    'Customer-Ordered Circuit Degraded': 'Cust_Deg',
    'Customer-Ordered Circuit Down': 'Cust_Down',
    'Device Down': 'Dev_Down',
    'Impaired - w/ Redundant Link Down': 'SDWAN_HA',
    'Outage Data': 'Outage_Data',
    'Proactive 4g Wireless Circuit': 'Pro_4GW',
    'Proactive Virtual VCE Down': 'Pro_VCE',
    'Proactive Circuit Down': 'Pro_Ckt',
    'Proactive Down': 'Pro',
    'Proactive Down - HA': 'Pro_HA',
    'Proactive Down - Multiple Underlay': 'Pro_Multi',
    'Service Proactive Down': 'Pro_Svc',
    'Severity 3-Alarm': 'Sev3'
}

#rename values in the ASSIGNED_QUEUE column
df['OPEN_TIER_3'] = df['OPEN_TIER_3'].replace(ot3_dict)
```

To simplify the names of these columns more succinctly, which will make them better for the next step of re-expression, I am choosing to shorten the names of several of these categories.

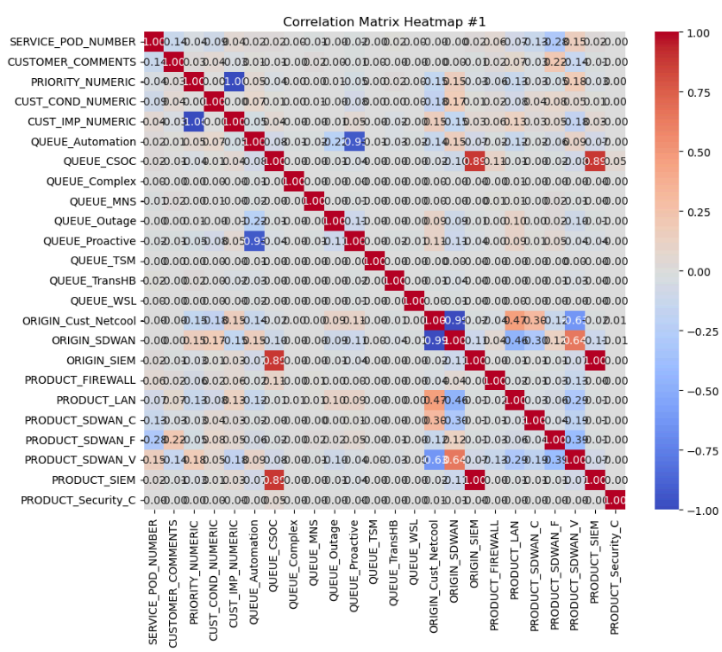
```python
df.rename(columns={'OPEN_TIER_1': 'OT1',
                   'OPEN_TIER_2': 'OT2',
                   'OPEN_TIER_3': 'OT3',
                  'ASSIGNED_QUEUE': 'QUEUE',
                  'ALARM_STATUS': 'ALARM',
                  'SERVICE_LEVEL': 'SVCLVL',
                  'SERVICE_STATUS': 'SVCST'}, inplace=True)
```

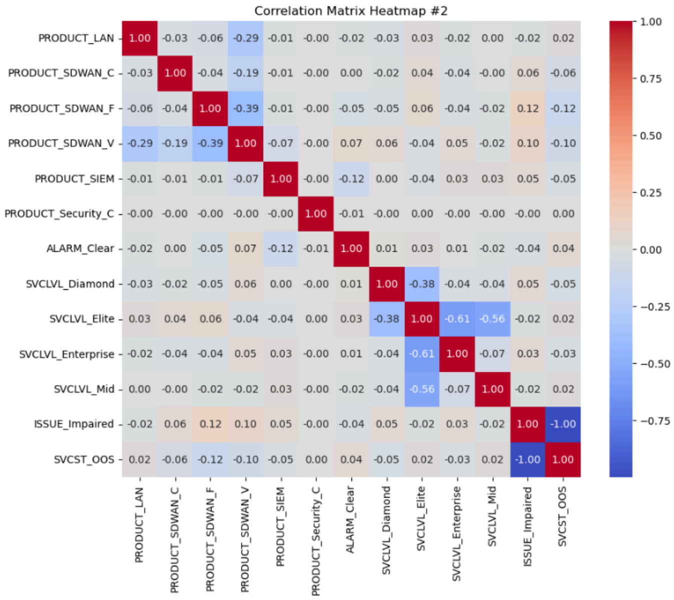
Ordinal Encoding is used for Priority, Primary_Customer_Condition, and Customer_Impact. The remainder variables I will use dummy variables using a prefix list. Given all my efforts thus far in reducing dimensionality, this process still expands our dataset from 18 original columns to 69!

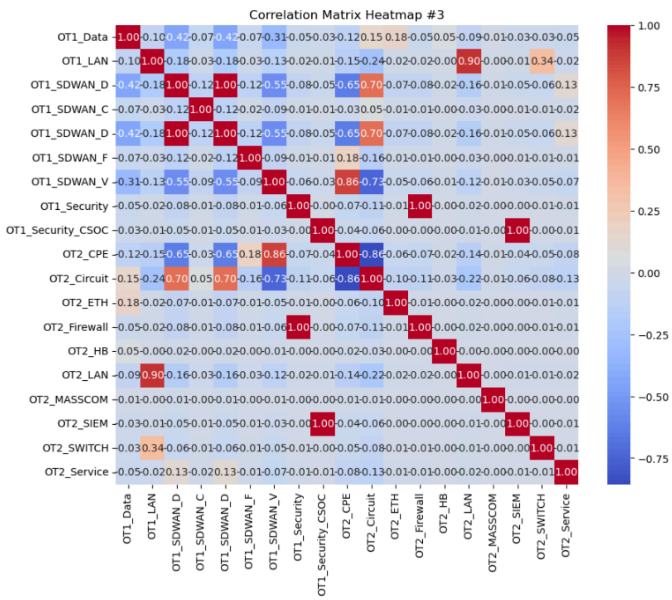
```
PRIORITY_NUMERIC unique values are [4 3 2 1]
{'PRIORITY_NUMERIC': {'Low': 1, 'Medium': 2, 'High': 3, 'Critical': 4}}

CUST_COND_NUMERIC unique values are [2 1 5 4 3]
{'CUST_COND_NUMERIC': {'Lost': 1, 'At Risk': 2, 'Save Motion': 3, 'Stable': 4, 'Growth': 5}}

CUST_IMP_NUMERIC unique values are [1 2 3 4]
{'CUST_IMP_NUMERIC': {'Sev 1': 1, 'Sev 2': 2, 'Sev 3': 3, 'Sev 4': 4}}

Orig Rows:  124738 Columns:  18
New Rows:   124738 Columns:  69
```

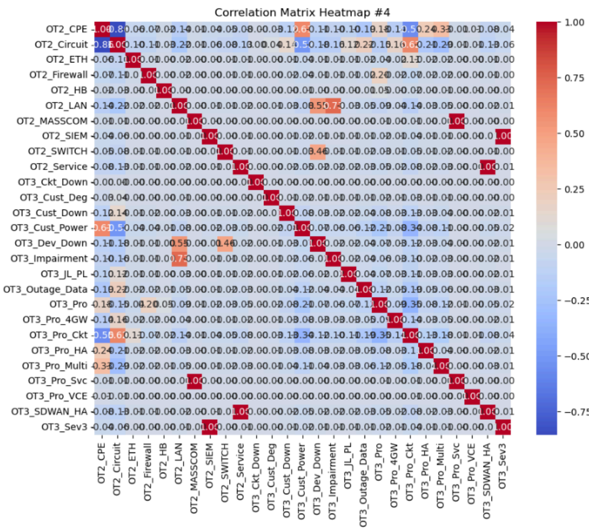
The next thing to do in our data preparation is to check for multicollinearity. Before I began this analysis, I expected some multicollinearity, but nothing prepared me for just how many variables I would find. This phase of the process was very tedious and repetitive which given time and resources, typically would be viewed as a disadvantage. However, nonetheless was it also very important to ensure our MLR model was appropriate and not overfit.

To first check for multicollinearity, I decided to view a correlation matrix using Seaborn's heatmap function (Waskom, 2021). There were simply too many rows to be able to visualize it, so I had to split up the data as meaningfully as I could 4 different times. Each chart provided certain variables that were an exact 1-to-1 match for multicollinearity.




Correlation Matrix Heatmap #2
PRODUCT_LAN
PRODUCT_SDWAN_C
PRODUCT_SDWAN_F
PRODUCT_SDWAN_V
PRODUCT_SIEM
PRODUCT_Security_C
ALARM_Clear
SVCLVL_Diamond
SVCLVL_Elite
SVCLVL_Enterprise
SVCLVL_Mid
ISSUE_Impaired
SVCST_OOS
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


Correlation Matrix Heatmap #3
OT1_Data
OT1_LAN
OT1_SDWAN_D
OT1_SDWAN_C
OT1_SDWAN_D
OT1_SDWAN_F
OT1_SDWAN_V
OT1_Security
OT1_Security_CSOC
OT2_CPE
OT2_Circuit
OT2_ETH
OT2_Firewall
OT2_HB
OT2_LAN
OT2_MASSCOM
OT2_SIEM
OT2_SWITCH
OT2_Service
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



I then removed all of the values in the heatmap that showed 1.0. This shows that there is a perfect multicollinearity in these variables, so therefore, they are not needed.

```
#Removing multicollinear columns not needed for this exercise
df.drop(['ORIGIN_SIEM', 'ISSUE_Impaired', 'OT2_Firewall', 'OT2_SIEM',
         'OT3_Pro_Svc', 'OT2_Service'],
                axis=1,
                inplace=True)
```

Furthering on in my search for multicollinearity, I next decided to use Statsmodel's variance_inflaction_factor (VIF) function. Any values over 10 will need to be removed, however, I also need to only remove a few variables at a time as removing them may lessen the multicollinearity in the remaining variables. Running this VIF function the first few times showed "inf" output which means that the variable's multicollinearity is *infinitely large*. Thus, these "inf" variables will need to be removed first.

On the first run, these variables were removed:

	variables	VIF
22	PRODUCT_Security_C	inf
43	OT3_Ckt_Down	inf
28	OT1_Data	inf
4	CUST_IMP_NUMERIC	inf
41	OT2_MASSCOM	inf

On the second run, these next variables were removed:

	variables	VIF
19	PRODUCT_SDWAN_V	inf
15	PRODUCT_FIREWALL	inf
28	OT1_SDWAN_D	inf
29	OT1_SDWAN_F	inf
18	PRODUCT_SDWAN_F	inf
30	OT1_SDWAN_V	inf
31	OT1_Security	inf

The limitation (or disadvantage) to using this VIF function is that when using too many variables, often it will only find the first few "inf" values. It's imperative to iteratively check for new variables, if any are found, remove them, and then check again. Repeat this process until there is no longer any multicollinearity remaining in the dataset.

On the third run, these next variables were removed:

	variables	VIF
46	OT3_Sev3	inf
25	OT1_Security_CSOC	inf
35	OT3_Dev_Down	inf
23	OT1_LAN	inf
36	OT3_Impairment	inf
17	PRODUCT_SIEM	inf

On the fourth run, I finally had no more "inf" values, but I now had plenty of values larger than 10. Rather than removing all values larger than 10 instantly, I chose to remove them in smaller sets, and re-checking each time for multicollinearity.

	variables	VIF
2	PRIORITY_NUMERIC	5683.720479
4	QUEUE_Automation	4243.925362
14	ORIGIN_SDWAN	2376.608636
22	OT1_SDWAN_C	2293.267971
16	PRODUCT_SDWAN_C	2292.224141
9	QUEUE_Proactive	2155.332384
36	OT3_Pro_Ckt	983.611441
34	OT3_Pro	469.744599
13	ORIGIN_Cust_Netcool	468.629337
31	OT3_Cust_Power	432.657602
23	OT2_CPE	219.493366
8	QUEUE_Outage	173.225665
33	OT3_Outage_Data	171.502499
38	OT3_Pro_Multi	149.820317
15	PRODUCT_LAN	109.586797
24	OT2_Circuit	105.287157
35	OT3_Pro_4GW	94.851083
19	SVCLVL_Elite	89.029540
37	OT3_Pro_HA	80.494979
30	OT3_Cust_Down	76.113297
32	OT3_JL_PL	57.554427
40	OT3_SDWAN_HA	32.216352
5	QUEUE_CSOC	17.076892
41	SVCST_OOS	16.282899
27	OT2_LAN	14.544733
3	CUST_COND_NUMERIC	10.452864

Even though all of these values are > 10, I only removed these variables first.

On the fifth run, I again chose to only remove the largest values, not necessarily all values larger than 10. For example, before the fourth run, OT3_Pro_Ckt had a VIF of 983, but afterwards the VIF dropped to just 92. This VIF is now less than 1/10th what it was before. This is why it's important to slowly and methodically remove multicollinear variables, choosing the largest ones first.

	variables	VIF
30	OT3_Pro_Ckt	92.622603
18	OT2_Circuit	88.488767
14	SVCLVL_Elite	82.515465
17	OT2_CPE	44.562840
28	OT3_Pro	44.502187
25	OT3_Cust_Power	43.707233
11	PRODUCT_LAN	20.877999
27	OT3_Outage_Data	17.157481
32	OT3_Pro_Multi	15.914117
35	SVCST_OOS	15.462234
21	OT2_LAN	14.322301
2	CUST_COND_NUMERIC	10.347226

On the sixth and final run, I only had 2 remaining variables larger than 10. I chose to just remove PRODUCT_LAN and leave OT2_LAN in the dataset.

	variables	VIF
11	PRODUCT_LAN	16.450187
18	OT2_LAN	14.063872

Checking one last time for multicollinearity I now have 29 remaining non-multicollinear variables out of the original 69.

	variables	VIF
2	CUST_COND_NUMERIC	5.223035
11	ALARM_Clear	4.833569
16	OT2_HB	3.419195
0	SERVICE_POD_NUMBER	3.377103
8	QUEUE_TransHB	3.352351
28	SVCST_OOS	2.559592
10	ORIGIN_Cust_Netcool	1.838377
14	SVCLVL_Mid	1.380652
17	OT2_LAN	1.328388
25	OT3_Pro_Multi	1.229302
13	SVCLVL_Enterprise	1.182810
22	OT3_Outage_Data	1.161530
24	OT3_Pro_HA	1.111908
1	CUSTOMER_COMMENTS	1.089271
12	SVCLVL_Diamond	1.070740
20	OT3_Cust_Down	1.068238
23	OT3_Pro_4GW	1.065628
15	OT2_ETH	1.065304
18	OT2_SWITCH	1.065145
9	QUEUE_WSL	1.054515
6	QUEUE_Outage	1.052872
21	OT3_JL_PL	1.042832
27	OT3_SDWAN_HA	1.029571
3	QUEUE_CSOC	1.016138
4	QUEUE_Complex	1.011143
19	OT3_Cust_Deg	1.004640
5	QUEUE_MNS	1.000817
26	OT3_Pro_VCE	1.000425
7	QUEUE_TSM	1.000356

ANALYSIS

Now that the data is prepared, it is time to run Statsmodels' MLR OLS function. Using ALARM_Clear as the dependent variable, I add all 29 independent variables to the function.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            ALARM_Clear   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.044
Method:                 Least Squares   F-statistic:                     208.2
Date:                Wed, 18 Dec 2024   Prob (F-statistic):               0.00
Time:                        17:13:48   Log-Likelihood:                -50835.
No. Observations:              124738   AIC:                         1.017e+05
Df Residuals:                  124709   BIC:                         1.020e+05
Df Model:                          28
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
Intercept               0.7780      0.004    194.800      0.000       0.770       0.786
CUST_COND_NUMERIC       0.0178      0.001     20.695      0.000       0.016       0.019
OT2_HB                 -0.1231      0.085     -1.452      0.146      -0.289       0.043
SERVICE_POD_NUMBER     -0.0033      0.000    -10.102      0.000      -0.004      -0.003
QUEUE_TransHB           0.0597      0.086      0.695      0.487      -0.109       0.228
SVCST_OOS               0.0653      0.003     25.965      0.000       0.060       0.070
ORIGIN_Cust_Netcool    -0.1034      0.003    -29.715      0.000      -0.110      -0.097
SVCLVL_Mid              0.0058      0.005      1.125      0.261      -0.004       0.016
OT2_LAN                 0.0225      0.006      3.514      0.000       0.010       0.035
OT3_Pro_Multi          -0.0762      0.005    -16.033      0.000      -0.086      -0.067
SVCLVL_Enterprise       0.0238      0.004      5.489      0.000       0.015       0.032
OT3_Outage_Data         0.0585      0.004     13.091      0.000       0.050       0.067
OT3_Pro_HA             -0.0405      0.006     -6.551      0.000      -0.053      -0.028
CUSTOMER_COMMENTS      -0.0084      0.003     -3.288      0.001      -0.013      -0.003
SVCLVL_Diamond         -0.0028      0.007     -0.423      0.673      -0.016       0.010
OT3_Cust_Down           0.1023      0.006     15.834      0.000       0.090       0.115
OT3_Pro_4GW            -0.0720      0.006    -12.356      0.000      -0.083      -0.061
OT2_ETH                -0.1000      0.013     -7.968      0.000      -0.125      -0.075
OT2_SWITCH              0.0672      0.015      4.492      0.000       0.038       0.096
QUEUE_WSL               0.1279      0.093      1.370      0.171      -0.055       0.311
QUEUE_Outage            0.1442      0.007     22.079      0.000       0.131       0.157
OT3_JL_PL              -0.1592      0.008    -21.014      0.000      -0.174      -0.144
OT3_SDWAN_HA           -0.1680      0.009    -17.799      0.000      -0.186      -0.149
QUEUE_CSOC             -0.7130      0.018    -40.048      0.000      -0.748      -0.678
QUEUE_Complex           0.2497      0.164      1.526      0.127      -0.071       0.570
OT3_Cust_Deg            0.1249      0.024      5.115      0.000       0.077       0.173
QUEUE_MNS               0.0295      0.073      0.405      0.685      -0.113       0.172
OT3_Pro_VCE            -0.2112      0.121     -1.741      0.082      -0.449       0.027
QUEUE_TSM              -0.4385      0.257     -1.704      0.088      -0.943       0.066
==============================================================================
Omnibus:                    34008.766   Durbin-Watson:                   1.768
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            69478.317
Skew:                          -1.721   Prob(JB):                         0.00
Kurtosis:                       4.236   Cond. No.                     1.73e+03
==============================================================================
```

I can see that there are several p-values larger than 0.05 which shows that they are not statistically significant and could hinder our model. Therefore, I will remove them from the function one by one, but will not necessarily remove them from the dataset. I will start with the largest values first until all p-values are less than or equal to 0.05.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            ALARM_Clear   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.044
Method:                 Least Squares   F-statistic:                     306.1
Date:                Fri, 27 Dec 2024   Prob (F-statistic):               0.00
Time:                        23:40:07   Log-Likelihood:                -50842.
No. Observations:              124738   AIC:                         1.017e+05
Df Residuals:                  124718   BIC:                         1.019e+05
Df Model:                          19
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
Intercept               0.7776      0.004    195.782      0.000       0.770       0.785
CUST_COND_NUMERIC       0.0177      0.001     20.821      0.000       0.016       0.019
SERVICE_POD_NUMBER     -0.0031      0.000    -10.942      0.000      -0.004      -0.003
SVCST_OOS               0.0652      0.003     25.933      0.000       0.060       0.070
ORIGIN_Cust_Netcool    -0.1032      0.003    -29.658      0.000      -0.110      -0.096
OT2_LAN                 0.0227      0.006      3.553      0.000       0.010       0.035
OT3_Pro_Multi          -0.0761      0.005    -16.017      0.000      -0.085      -0.067
SVCLVL_Enterprise       0.0229      0.004      5.397      0.000       0.015       0.031
OT3_Outage_Data         0.0584      0.004     13.068      0.000       0.050       0.067
OT3_Pro_HA             -0.0405      0.006     -6.560      0.000      -0.053      -0.028
CUSTOMER_COMMENTS      -0.0082      0.003     -3.247      0.001      -0.013      -0.003
OT3_Cust_Down           0.1021      0.006     15.800      0.000       0.089       0.115
OT3_Pro_4GW            -0.0720      0.006    -12.360      0.000      -0.083      -0.061
OT2_ETH                -0.1003      0.013     -7.989      0.000      -0.125      -0.076
OT2_SWITCH              0.0673      0.015      4.501      0.000       0.038       0.097
QUEUE_Outage            0.1439      0.007     22.037      0.000       0.131       0.157
OT3_JL_PL              -0.1593      0.008    -21.029      0.000      -0.174      -0.144
OT3_SDWAN_HA           -0.1681      0.009    -17.817      0.000      -0.187      -0.150
QUEUE_CSOC             -0.7127      0.018    -40.035      0.000      -0.748      -0.678
OT3_Cust_Deg            0.1249      0.024      5.113      0.000       0.077       0.173
==============================================================================
Omnibus:                    34012.934   Durbin-Watson:                   1.768
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            69491.997
Skew:                          -1.721   Prob(JB):                         0.00
Kurtosis:                       4.236   Cond. No.                         164.
==============================================================================
```

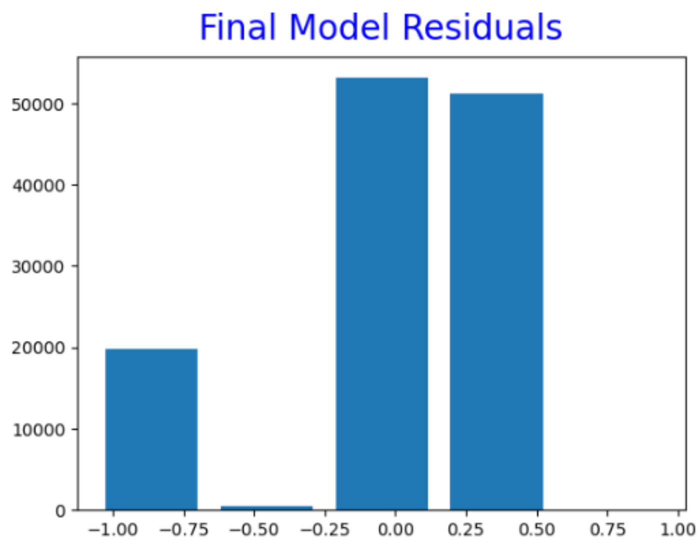
This is the final model that will be used to summarize and interpret the data. All insignificant p-values have been removed which lowered the condition number from 1,730 to 164. All numerical issues have been removed from the model.

DATA SUMMARY AND IMPLICATIONS

A relatively low R-squared value shows that just 4.5% of the variability in ALARM_Clear is explained in the model. The same came be said for the low 4.4% Adjusted R-Squared value. This suggests plainly that the independent variables do not account for much of the variability in ALARM_Clear. In other words, you would not be wise to attempt to predict if the alarm were to clear based on the independent variables in this model.

Several notable variables have a strong impact on the dependent variable. QUEUE_CSOC has the strongest negative effect which indicates that increases in QUEUE_CSOC drastically decreases ALARM_Clear. Another queue, QUEUE_Outage has the opposite effect on ALARM_Clear, though not nearly as strong with a coefficient of 0.1439. This queue has the highest positive coefficient.

Both the Omnibus and Jarque-Bera tests have significantly large values which indicate that there is a non-normality in the residuals. Viewing these residuals is evident that there is a negative left skew, which is also shown in the -1.721 Skew value. This is a limitation in the dataset.



Given a high F-statistic of 306.1, and a significantly low p-value for this F-statistic of 0.00, we are assured that this model bears an overall significance with respect to the independent variables explaining the variance in the dependent variable. Therefore, we must reject the null hypothesis and rather accept the alternative hypothesis that there is significant correlation between at least one of the independent variables and the alarm being cleared.

The variables that are significantly positively correlated are Primary Customer Condition, Service Status, Service Level of Enterprise, Assigned Queue of Outage, Open Tier 2 codes of LAN and Switch, and Open Tier 3 codes of Outage Data, Customer Down, and Customer Degraded.

The variables that are significantly negatively correlated are Service POD Number, Origin of Customer Netcool, Customer Comments, Assigned Queue of CSOC, Open Tier 2 code of Ethernet, and Open Tier 3 codes of Proactive Multiple Underlay, Proactive HA, Proactive 4G Wireless, Jitter, Latency, Packet Loss, and SD-WAN HA.

The recommended course of action is for the ISP to simplify the level of detail the alarm cases are given. This dataset shows that too much information is not a “good thing.” Given the complexity of the data, it makes it extremely difficult to analyze properly without first removing multicollinear variables as well as statistically insignificant ones. Teams within the ISP are working hard to determine how to cut down on the amount of time an alarm stays active, and how to proactively clear alarms. Using this dataset will make this task impossible given that this model is only able to predict 4.5% of the variability in the alarm clearing. Predictive Analytics can greatly help an ISP become more proactive in its approach to network optimization than strictly being reactionary (Shillingsburg, 2024). Steps should be taken to reduce the dimensionality of this dataset which would aid the analysts who are looking to predict certain behaviors from the data.

The first direction of approach on a future study would be to remove the Open Tier Codes and replace them with the previously removed “Closing” codes to evaluate if this model is able to better predict the alarm clearing. For this ISP, the “open” codes are determined when the case is opened, and the “closing” codes are assigned once the case is closed. Since more is known about the issue at the end of the case rather than the beginning, this could improve the model’s explanatory power.

Another direction to take would be to change the dependent variable to SERVICE_STATUS and see how well the independent variables explain the variance there instead. The ALARM_STATUS variable is updated during the life of the case once the alarm clears, but the SERVICE_STATUS is a historical object only. Therefore, we could learn which variables are statistically significant to SERVICE_STATUS which would be useful information for the ISP that might explain more about why some products, queues, origins, etc., tend to correlate to “Out of Service” events.

SOURCES FOR THIRD-PARTY CODE

Massaron, L. & Boschetti, A. (2016). Regression Analysis with Python: Learn the Art of Regression Analysis with Python. Packt Publishing.

Pandas (2024, September 20). Pandas documentation. Retrieved December 10, 2024, from https://pandas.pydata.org/docs/index.html.

Seabold, S. & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. Proceedings of the 9th Python in Scientific Conference. Retrieved December 10, 2024, from https://www.statsmodels.org/stable/index.html.

Waskom, M. L. (2021). Seaborn: statistical data visualization. Retrieved December 10, 2024, from https://seaborn.pydata.org/index.html.

SOURCES

Geeks for Geeks (May 06, 2023). Introduction to Dimensionality Reduction. Retrieved December 27, 2024, from https://www.geeksforgeeks.org/dimensionality-reduction/.

Shillingsburg, S. (2024, October 30). Predictive Analytics for Network Optimization. Retrieved December 19, 2024, from https://sonar.software/blog/predictive-analytics-for-network-optimization.

Torkildson, A. (2017-2024). Why The Internet is so Important for Modern Business. Retrieved December 19, 2024, from https://www.thinkers360.com/tl/blog/members/why-the-internet-is-so-important-for-modern-business.

APPENDIX

The Python code in Jupyter Notebook

```
#import important libraries
import numpy as np
import pandas as pd
from pandas import DataFrame

#import statistics libraries
import statistics
from scipy import stats
from statsmodels.formula.api import ols

#import visualization libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings #to ignore warnings
warnings.simplefilter(action='ignore')
warnings.filterwarnings('ignore')
```

```
#import CSV file
df = pd.read_csv("~/Documents/WGU/D214 - Data Analytics Graduate Capstone/Data Analytics Capstone Project.csv")
```

```
df.info()
```

```
#Removing columns not needed for this exercise
df.drop(['ARBITRARY_ID', 'CITY', 'STATE', 'CASE_DURATION', 'CREATEDDATE', 'CLOSEDDATE',
        'CLOSING_FAULT', 'CLOSING_ISSUE', 'CLOSING_RESOLUTION', 'ACTION_CODE', 'CAUSE_CODE', 'CLASS_ITEM_CODE'],
               axis=1,
               inplace=True)
```

```
#detect Income outliers before treatment
plt.boxplot(df['CUSTOMER_COMMENTS'], vert=False)
plt.xlabel('CUSTOMER_COMMENTS')
plt.title('CUSTOMER_COMMENTS before treatment', fontsize=15, pad=10, color='blue')
plt.show()
```

```
#Treating outliers
df['CUSTOMER_COMMENTS'] = np.where(df['CUSTOMER_COMMENTS'] > 25, np.nan, df['CUSTOMER_COMMENTS'])
df['CUSTOMER_COMMENTS'].fillna(df['CUSTOMER_COMMENTS'].median(), inplace=True)
```

```
#detect Income outliers after treatment
plt.boxplot(df['CUSTOMER_COMMENTS'], vert=False)
plt.xlabel('CUSTOMER_COMMENTS')
plt.title('CUSTOMER_COMMENTS after treatment', fontsize=15, pad=10, color='blue')
plt.show()
```

```python
#Define a dictionary to rename ASSIGNED_QUEUE -- this will help to reduce dimensionality

queue_dict = {
    'SA Advanced OS Support': 'AdvOSSpt',
    'SA Automation Hold': 'Automation',
    'SA Customer Owned Automation Hold': 'Automation',
    'SA Customer Power Outage': 'Automation',
    'SA Customer Request Automation Hold': 'Automation',
    'SA Disco Automation Hold': 'Automation',
    'SA Order Related Automation Hold': 'Automation',
    'SA Partner Standard Automation Hold': 'Automation',
    'SA SDWAN Automation Hold': 'Automation',
    'SA Simplex Automation Hold': 'Automation',
    'SA Unresponsive Customer Automation Hold': 'Automation',
    'SA Wireless Automation Hold': 'Automation',
    'SA Transport HB': 'TransHB',
    'SA Complex NOC': 'Complex',
    'SA SMB Complex': 'Complex',
    'SA CSOC SASE': 'CSOC',
    'SA CSOC SIEM': 'CSOC',
    'SA CSOC Support': 'CSOC',
    'SA Outage Child': 'Outage',
    'SA Outage Lead': 'Outage',
    'SA Premier Managed Network Solutions': 'MNS',
    'SA Prime Managed Network Solutions': 'MNS',
    'SA Select Managed Network Solutions': 'MNS',
    'SA Proactive Alarm': 'Proactive',
    'SA Proactive Hold': 'Proactive',
    'SA Proactive Support': 'Proactive',
    'SA TSM': 'TSM',
    'SA WSL Event': 'WSL',
    'SA WSL Waves': 'WSL'
}

#rename values in the ASSIGNED_QUEUE column
df['ASSIGNED_QUEUE'] = df['ASSIGNED_QUEUE'].replace(queue_dict)
```

```python
#Define a dictionary to rename ORIGIN

origin_dict = {
    'Alarm - SD WAN': 'SDWAN',
    'Alarm - Customer Netcool': 'Cust_Netcool',
    'Alarm - Core Netcool': 'Core_Netcool',
    'Alarm - SIEM': 'SIEM'
}

#rename values in the ASSIGNED_QUEUE column
df['ORIGIN'] = df['ORIGIN'].replace(origin_dict)
```

```python
#Define a dictionary to rename PRODUCT

product_dict = {
    'LAN Services': 'LAN',
    'SD-WAN C': 'SDWAN_C',
    'SD-WAN F': 'SDWAN_F',
    'SD-WAN V': 'SDWAN_V',
    'Security-Firewall': 'FIREWALL',
    'Security-C': 'Security_C',
    'Security-S': 'SIEM'
}

#rename values in the ASSIGNED_QUEUE column
df['PRODUCT'] = df['PRODUCT'].replace(product_dict)
```

```python
#Define a dictionary to rename SERVICE_LEVEL

svclvl_dict = {
    'Mid Market': 'Mid'
}

#rename values in the ASSIGNED_QUEUE column
df['SERVICE_LEVEL'] = df['SERVICE_LEVEL'].replace(svclvl_dict)
```

```
#Define a dictionary to rename SERVICE_STATUS

svcst_dict = {
    'Out of Service': 'OOS',
    'In Service': 'IS'
}

#rename values in the ASSIGNED_QUEUE column
df['SERVICE_STATUS'] = df['SERVICE_STATUS'].replace(svcst_dict)
```

```
#Define a dictionary to rename ISSUE

issue_dict = {
    'Service Impairment': 'Impaired',
    'Service Down': 'Down'
}

#rename values in the ASSIGNED_QUEUE column
df['ISSUE'] = df['ISSUE'].replace(issue_dict)
```

```
#Define a dictionary to rename OPEN_TIER_1

ot1_dict = {
    'LAN Services': 'LAN',
    'SD-WAN - Data': 'SDWAN_D',
    'SD-WAN Data': 'SDWAN_D',
    'SD-WAN C': 'SDWAN_C',
    'SD-WAN F': 'SDWAN_F',
    'SD-WAN V': 'SDWAN_V',
    'Security-CSOC': 'Security_CSOC'
}

#rename values in the ASSIGNED_QUEUE column
df['OPEN_TIER_1'] = df['OPEN_TIER_1'].replace(ot1_dict)
```

```
#Define a dictionary to rename OPEN_TIER_2

ot2_dict = {
    'Ethernet Circuit': 'ETH',
    'High Bandwidth': 'HB',
    'LAN - Switch': 'SWITCH',
    'LAN - Access Point': 'AP'
}

#rename values in the ASSIGNED_QUEUE column
df['OPEN_TIER_2'] = df['OPEN_TIER_2'].replace(ot2_dict)
```

```
#Define a dictionary to rename OPEN_TIER_3

ot3_dict = {
    'Circuit Jitter': 'JL_PL',
    'Circuit Latency': 'JL_PL',
    'Circuit Packet Loss': 'JL_PL',
    'Circuit Proactive Down': 'Pro_Ckt',
    'Circuit Down': 'Ckt_Down',
    'Customer Owned Circuit Degraded': 'Cust_Deg',
    'Customer Owned Circuit Down': 'Cust_Down',
    'Customer Power Outage': 'Cust_Power',
    'Customer-Ordered Circuit Degraded': 'Cust_Deg',
    'Customer-Ordered Circuit Down': 'Cust_Down',
    'Device Down': 'Dev_Down',
    'Impaired - w/ Redundant Link Down': 'SDWAN_HA',
    'Outage Data': 'Outage_Data',
    'Proactive 4g Wireless Circuit': 'Pro_4GW',
    'Proactive Virtual VCE Down': 'Pro_VCE',
    'Proactive Circuit Down': 'Pro_Ckt',
    'Proactive Down': 'Pro',
    'Proactive Down - HA': 'Pro_HA',
    'Proactive Down - Multiple Underlay': 'Pro_Multi',
    'Service Proactive Down': 'Pro_Svc',
    'Severity 3-Alarm': 'Sev3'
}

#rename values in the ASSIGNED_QUEUE column
df['OPEN_TIER_3'] = df['OPEN_TIER_3'].replace(ot3_dict)
```

```python
df.rename(columns={'OPEN_TIER_1': 'OT1',
                    'OPEN_TIER_2': 'OT2',
                    'OPEN_TIER_3': 'OT3',
                   'ASSIGNED_QUEUE': 'QUEUE',
                   'ALARM_STATUS': 'ALARM',
                   'SERVICE_LEVEL': 'SVCLVL',
                   'SERVICE_STATUS': 'SVCST'}, inplace=True)
```

```python
## Re-expression of Categorical Variables

## Ordinal Categorical

#ordinal encoding via dictionary
df['PRIORITY_NUMERIC'] = df['PRIORITY']
dict_priority =  {"PRIORITY_NUMERIC": {"Low": 1, "Medium": 2, "High": 3, "Critical": 4}}
df.replace(dict_priority, inplace=True)

#print before & after
print("PRIORITY_NUMERIC unique values are", df.PRIORITY_NUMERIC.unique())
print(dict_priority)
print("")

#ordinal encoding via dictionary
df['CUST_COND_NUMERIC'] = df['PRIMARY_CUSTOMER_CONDITION']
dict_cust_cond =  {"CUST_COND_NUMERIC": {"Lost": 1, "At Risk": 2, "Save Motion": 3, "Stable": 4, "Growth": 5}}
df.replace(dict_cust_cond, inplace=True)

#print before & after
print("CUST_COND_NUMERIC unique values are", df.CUST_COND_NUMERIC.unique())
print(dict_cust_cond)
print("")

#ordinal encoding via dictionary
df['CUST_IMP_NUMERIC'] = df['CUSTOMER_IMPACT']
dict_cust_imp =  {"CUST_IMP_NUMERIC": {"Sev 1": 1, "Sev 2": 2, "Sev 3": 3, "Sev 4": 4}}
df.replace(dict_cust_imp, inplace=True)

#print before & after
print("CUST_IMP_NUMERIC unique values are", df.CUST_IMP_NUMERIC.unique())
print(dict_cust_imp)
print("")

## Nominal Categorical

prefix_list = ['QUEUE', 'ORIGIN', 'PRODUCT', 'ALARM', 'SVCLVL',
               'OT1', 'OT2', 'OT3', 'ISSUE', 'SVCST']

# save original dataframe
original_df = df

# overwrite df dataframe with dummy variables
df = pd.get_dummies(df,
                    prefix=prefix_list,
                    prefix_sep='_',
                    dummy_na=False,
                    drop_first=True,
                    columns=prefix_list)

#show how many new columns/variables
print("Orig Rows: ", original_df.shape[0], "Columns: ", original_df.shape[1])
print("New Rows:  ", df.shape[0], "Columns: ", df.shape[1])
```

```python
# Set pandas options to show all columns
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.width', 1000)        # Set a wide enough width to avoid wrapping

# Display column names in list
column_names = df.columns
column_names_list = list(column_names)
print("All Column Names:")
print(column_names_list)
```

```python
# Calculate the correlation matrix
correlation_matrix = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'PRIORITY', 'PRIMARY_CUSTOMER_CONDITION',
                         'CUSTOMER_IMPACT', 'PRIORITY_NUMERIC', 'CUST_COND_NUMERIC', 'CUST_IMP_NUMERIC',
                         'QUEUE_Automation', 'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS', 'QUEUE_Outage',
                         'QUEUE_Proactive', 'QUEUE_TSM', 'QUEUE_TransHB', 'QUEUE_WSL', 'ORIGIN_Cust_Netcool',
                         'ORIGIN_SDWAN', 'ORIGIN_SIEM', 'PRODUCT_FIREWALL', 'PRODUCT_LAN', 'PRODUCT_SDWAN_C',
                         'PRODUCT_SDWAN_F', 'PRODUCT_SDWAN_V', 'PRODUCT_SIEM', 'PRODUCT_Security_C']].corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Matrix Heatmap #1")
plt.show()
```

```python
# Calculate the correlation matrix
correlation_matrix = df[['PRODUCT_LAN', 'PRODUCT_SDWAN_C', 'PRODUCT_SDWAN_F', 'PRODUCT_SDWAN_V', 'PRODUCT_SIEM',
                         'PRODUCT_Security_C', 'ALARM_Clear', 'SVCLVL_Diamond', 'SVCLVL_Elite', 'SVCLVL_Enterprise',
                         'SVCLVL_Mid', 'ISSUE_Impaired', 'SVCST_OOS']].corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Matrix Heatmap #2")
plt.show()
```

```python
# Calculate the correlation matrix
correlation_matrix = df[['OT1_Data', 'OT1_LAN', 'OT1_SDWAN_D', 'OT1_SDWAN_C', 'OT1_SDWAN_D',
                         'OT1_SDWAN_F', 'OT1_SDWAN_V', 'OT1_Security', 'OT1_Security_CSOC', 'OT2_CPE', 'OT2_Circuit',
                         'OT2_ETH', 'OT2_Firewall', 'OT2_HB', 'OT2_LAN', 'OT2_MASSCOM', 'OT2_SIEM', 'OT2_SWITCH',
                         'OT2_Service']].corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Matrix Heatmap #3")
plt.show()
```

```python
# Calculate the correlation matrix
correlation_matrix = df[['OT2_CPE', 'OT2_Circuit', 'OT2_ETH', 'OT2_Firewall', 'OT2_HB', 'OT2_LAN', 'OT2_MASSCOM',
                         'OT2_SIEM', 'OT2_SWITCH', 'OT2_Service', 'OT3_Ckt_Down', 'OT3_Cust_Deg', 'OT3_Cust_Down',
                         'OT3_Cust_Power', 'OT3_Dev_Down', 'OT3_Impairment', 'OT3_JL_PL', 'OT3_Outage_Data',
                         'OT3_Pro', 'OT3_Pro_4GW', 'OT3_Pro_Ckt', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_Svc',
                         'OT3_Pro_VCE', 'OT3_SDWAN_HA', 'OT3_Sev3']].corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Matrix Heatmap #4")
plt.show()
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['ORIGIN_SIEM', 'ISSUE_Impaired', 'OT2_Firewall', 'OT2_SIEM',
        'OT3_Pro_Svc', 'OT2_Service'],
               axis=1,
               inplace=True)
```

```python
## A bit of housecleaning on variables

#update dtype
df['CUSTOMER_COMMENTS'] = df['CUSTOMER_COMMENTS'].astype('int64')
df['QUEUE_Automation'] = df['QUEUE_Automation'].astype('int64')
df['QUEUE_CSOC'] = df['QUEUE_CSOC'].astype('int64')
df['QUEUE_Complex'] = df['QUEUE_Complex'].astype('int64')
df['QUEUE_MNS'] = df['QUEUE_MNS'].astype('int64')
df['QUEUE_Outage'] = df['QUEUE_Outage'].astype('int64')
df['QUEUE_Proactive'] = df['QUEUE_Proactive'].astype('int64')
df['QUEUE_TSM'] = df['QUEUE_TSM'].astype('int64')
df['QUEUE_TransHB'] = df['QUEUE_TransHB'].astype('int64')
df['QUEUE_WSL'] = df['QUEUE_WSL'].astype('int64')
df['ORIGIN_Cust_Netcool'] = df['ORIGIN_Cust_Netcool'].astype('int64')
df['ORIGIN_SDWAN'] = df['ORIGIN_SDWAN'].astype('int64')
df['PRODUCT_FIREWALL'] = df['PRODUCT_FIREWALL'].astype('int64')
df['PRODUCT_LAN'] = df['PRODUCT_LAN'].astype('int64')
df['PRODUCT_SDWAN_C'] = df['PRODUCT_SDWAN_C'].astype('int64')
df['PRODUCT_SDWAN_F'] = df['PRODUCT_SDWAN_F'].astype('int64')
df['PRODUCT_SDWAN_V'] = df['PRODUCT_SDWAN_V'].astype('int64')
df['PRODUCT_SIEM'] = df['PRODUCT_SIEM'].astype('int64')
df['PRODUCT_Security_C'] = df['PRODUCT_Security_C'].astype('int64')
df['ALARM_Clear'] = df['ALARM_Clear'].astype('int64')
df['SVCLVL_Diamond'] = df['SVCLVL_Diamond'].astype('int64')
df['SVCLVL_Elite'] = df['SVCLVL_Elite'].astype('int64')
df['SVCLVL_Enterprise'] = df['SVCLVL_Enterprise'].astype('int64')
df['SVCLVL_Mid'] = df['SVCLVL_Mid'].astype('int64')
df['SVCST_OOS'] = df['SVCST_OOS'].astype('int64')
df['OT1_Data'] = df['OT1_Data'].astype('int64')
df['OT1_LAN'] = df['OT1_LAN'].astype('int64')
df['OT1_SDWAN_C'] = df['OT1_SDWAN_C'].astype('int64')
df['OT1_SDWAN_D'] = df['OT1_SDWAN_D'].astype('int64')
df['OT1_SDWAN_F'] = df['OT1_SDWAN_F'].astype('int64')
df['OT1_SDWAN_V'] = df['OT1_SDWAN_V'].astype('int64')
df['OT1_Security'] = df['OT1_Security'].astype('int64')
df['OT1_Security_CSOC'] = df['OT1_Security_CSOC'].astype('int64')
df['OT2_CPE'] = df['OT2_CPE'].astype('int64')
df['OT2_Circuit'] = df['OT2_Circuit'].astype('int64')
df['OT2_ETH'] = df['OT2_ETH'].astype('int64')
df['OT2_HB'] = df['OT2_HB'].astype('int64')
df['OT2_LAN'] = df['OT2_LAN'].astype('int64')
df['OT2_MASSCOM'] = df['OT2_MASSCOM'].astype('int64')
df['OT2_SWITCH'] = df['OT2_SWITCH'].astype('int64')
df['OT3_Ckt_Down'] = df['OT3_Ckt_Down'].astype('int64')
df['OT3_Cust_Deg'] = df['OT3_Cust_Deg'].astype('int64')
df['OT3_Cust_Down'] = df['OT3_Cust_Down'].astype('int64')
df['OT3_Cust_Power'] = df['OT3_Cust_Power'].astype('int64')
df['OT3_Dev_Down'] = df['OT3_Dev_Down'].astype('int64')
df['OT3_Impairment'] = df['OT3_Impairment'].astype('int64')
df['OT3_JL_PL'] = df['OT3_JL_PL'].astype('int64')
df['OT3_Outage_Data'] = df['OT3_Outage_Data'].astype('int64')
df['OT3_Pro'] = df['OT3_Pro'].astype('int64')
df['OT3_Pro_Ckt'] = df['OT3_Pro_Ckt'].astype('int64')
df['OT3_Pro_4GW'] = df['OT3_Pro_4GW'].astype('int64')
df['OT3_Pro_HA'] = df['OT3_Pro_HA'].astype('int64')
df['OT3_Pro_Multi'] = df['OT3_Pro_Multi'].astype('int64')
df['OT3_Pro_VCE'] = df['OT3_Pro_VCE'].astype('int64')
df['OT3_SDWAN_HA'] = df['OT3_SDWAN_HA'].astype('int64')
df['OT3_Sev3'] = df['OT3_Sev3'].astype('int64')
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'PRIORITY_NUMERIC', 'CUST_COND_NUMERIC', 'CUST_IMP_NUMERIC',
        'QUEUE_Automation', 'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS', 'QUEUE_Outage', 'QUEUE_Proactive', 'QUEUE_TSM',
        'QUEUE_TransHB', 'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'ORIGIN_SDWAN', 'PRODUCT_FIREWALL', 'PRODUCT_LAN',
        'PRODUCT_SDWAN_C', 'PRODUCT_SDWAN_F', 'PRODUCT_SDWAN_V', 'PRODUCT_SIEM', 'PRODUCT_Security_C', 'ALARM_Clear',
        'SVCLVL_Diamond', 'SVCLVL_Elite', 'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT1_Data', 'OT1_LAN', 'OT1_SDWAN_C',
        'OT1_SDWAN_D', 'OT1_SDWAN_F', 'OT1_SDWAN_V', 'OT1_Security', 'OT1_Security_CSOC', 'OT2_CPE', 'OT2_Circuit',
        'OT2_ETH', 'OT2_HB', 'OT2_LAN', 'OT2_MASSCOM', 'OT2_SWITCH', 'OT3_Ckt_Down', 'OT3_Cust_Deg', 'OT3_Cust_Down',
        'OT3_Cust_Power', 'OT3_Dev_Down', 'OT3_Impairment', 'OT3_JL_PL', 'OT3_Outage_Data', 'OT3_Pro', 'OT3_Pro_4GW',
        'OT3_Pro_Ckt', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE', 'OT3_SDWAN_HA', 'OT3_Sev3', 'SVCST_OOS']]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['PRODUCT_Security_C', 'OT3_Ckt_Down', 'OT1_Data',
         'CUST_IMP_NUMERIC','OT2_MASSCOM'], axis=1, inplace=True)
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'PRIORITY_NUMERIC', 'CUST_COND_NUMERIC', 'QUEUE_Automation',
        'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS', 'QUEUE_Outage', 'QUEUE_Proactive', 'QUEUE_TSM', 'QUEUE_TransHB',
        'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'ORIGIN_SDWAN', 'PRODUCT_FIREWALL', 'PRODUCT_LAN', 'PRODUCT_SDWAN_C',
        'PRODUCT_SDWAN_F', 'PRODUCT_SDWAN_V', 'PRODUCT_SIEM', 'ALARM_Clear', 'SVCLVL_Diamond', 'SVCLVL_Elite',
        'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT1_LAN', 'OT1_SDWAN_C', 'OT1_SDWAN_D', 'OT1_SDWAN_F', 'OT1_SDWAN_V',
        'OT1_Security', 'OT1_Security_CSOC', 'OT2_CPE', 'OT2_Circuit', 'OT2_ETH', 'OT2_HB', 'OT2_LAN', 'OT2_SWITCH',
        'OT3_Cust_Deg', 'OT3_Cust_Down', 'OT3_Cust_Power', 'OT3_Dev_Down', 'OT3_Impairment', 'OT3_JL_PL',
        'OT3_Outage_Data', 'OT3_Pro', 'OT3_Pro_4GW', 'OT3_Pro_Ckt', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE',
        'OT3_SDWAN_HA', 'OT3_Sev3', 'SVCST_OOS']]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['PRODUCT_SDWAN_V', 'PRODUCT_FIREWALL', 'OT1_SDWAN_D',
        'OT1_SDWAN_F', 'PRODUCT_SDWAN_F', 'OT1_SDWAN_V', 'OT1_Security'], axis=1, inplace=True)
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'PRIORITY_NUMERIC', 'CUST_COND_NUMERIC', 'QUEUE_Automation',
        'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS', 'QUEUE_Outage', 'QUEUE_Proactive', 'QUEUE_TSM', 'QUEUE_TransHB',
        'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'ORIGIN_SDWAN', 'PRODUCT_LAN', 'PRODUCT_SDWAN_C', 'PRODUCT_SIEM',
        'ALARM_Clear', 'SVCLVL_Diamond', 'SVCLVL_Elite', 'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT1_LAN', 'OT1_SDWAN_C',
        'OT1_Security_CSOC', 'OT2_CPE', 'OT2_Circuit', 'OT2_ETH', 'OT2_HB', 'OT2_LAN', 'OT2_SWITCH', 'OT3_Cust_Deg',
        'OT3_Cust_Down', 'OT3_Cust_Power', 'OT3_Dev_Down', 'OT3_Impairment', 'OT3_JL_PL', 'OT3_Outage_Data', 'OT3_Pro',
        'OT3_Pro_4GW', 'OT3_Pro_Ckt', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE', 'OT3_SDWAN_HA', 'OT3_Sev3',
        'SVCST_OOS']]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['OT3_Sev3', 'OT1_Security_CSOC', 'OT3_Dev_Down',
        'OT1_LAN', 'OT3_Impairment', 'PRODUCT_SIEM'], axis=1, inplace=True)
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'PRIORITY_NUMERIC', 'CUST_COND_NUMERIC', 'QUEUE_Automation',
        'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS', 'QUEUE_Outage', 'QUEUE_Proactive', 'QUEUE_TSM', 'QUEUE_TransHB',
        'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'ORIGIN_SDWAN', 'PRODUCT_LAN', 'PRODUCT_SDWAN_C', 'ALARM_Clear',
        'SVCLVL_Diamond', 'SVCLVL_Elite', 'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT1_SDWAN_C', 'OT2_CPE', 'OT2_Circuit',
        'OT2_ETH', 'OT2_HB', 'OT2_LAN', 'OT2_SWITCH', 'OT3_Cust_Deg', 'OT3_Cust_Down', 'OT3_Cust_Power', 'OT3_JL_PL',
        'OT3_Outage_Data', 'OT3_Pro', 'OT3_Pro_4GW', 'OT3_Pro_Ckt', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE',
        'OT3_SDWAN_HA', 'SVCST_OOS']]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['PRIORITY_NUMERIC', 'QUEUE_Automation', 'ORIGIN_SDWAN', 'OT1_SDWAN_C',
        'PRODUCT_SDWAN_C', 'QUEUE_Proactive'], axis=1, inplace=True)
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'CUST_COND_NUMERIC', 'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS',
        'QUEUE_Outage', 'QUEUE_TSM', 'QUEUE_TransHB', 'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'PRODUCT_LAN', 'ALARM_Clear',
        'SVCLVL_Diamond', 'SVCLVL_Elite', 'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT2_CPE', 'OT2_Circuit', 'OT2_ETH', 'OT2_HB',
        'OT2_LAN', 'OT2_SWITCH', 'OT3_Cust_Deg', 'OT3_Cust_Down', 'OT3_Cust_Power', 'OT3_JL_PL', 'OT3_Outage_Data',
        'OT3_Pro', 'OT3_Pro_4GW', 'OT3_Pro_Ckt', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE', 'OT3_SDWAN_HA', 'SVCST_OOS'
       ]]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['OT3_Pro_Ckt', 'OT2_Circuit', 'SVCLVL_Elite',
         'OT2_CPE', 'OT3_Pro', 'OT3_Cust_Power'], axis=1, inplace=True)
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'CUST_COND_NUMERIC', 'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS',
        'QUEUE_Outage', 'QUEUE_TSM', 'QUEUE_TransHB', 'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'PRODUCT_LAN', 'ALARM_Clear',
        'SVCLVL_Diamond', 'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT2_ETH', 'OT2_HB', 'OT2_LAN', 'OT2_SWITCH', 'OT3_Cust_Deg',
        'OT3_Cust_Down', 'OT3_JL_PL', 'OT3_Outage_Data', 'OT3_Pro_4GW', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE',
        'OT3_SDWAN_HA', 'SVCST_OOS']]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
#Removing multicollinear columns not needed for this exercise
df.drop(['PRODUCT_LAN'], axis=1, inplace=True)
```

```python
## Checking for Multicollinearity

# Import functions
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term
X = df[['SERVICE_POD_NUMBER', 'CUSTOMER_COMMENTS', 'CUST_COND_NUMERIC', 'QUEUE_CSOC', 'QUEUE_Complex', 'QUEUE_MNS',
        'QUEUE_Outage', 'QUEUE_TSM', 'QUEUE_TransHB', 'QUEUE_WSL', 'ORIGIN_Cust_Netcool', 'ALARM_Clear',
        'SVCLVL_Diamond', 'SVCLVL_Enterprise', 'SVCLVL_Mid', 'OT2_ETH', 'OT2_HB', 'OT2_LAN', 'OT2_SWITCH', 'OT3_Cust_Deg',
        'OT3_Cust_Down', 'OT3_JL_PL', 'OT3_Outage_Data', 'OT3_Pro_4GW', 'OT3_Pro_HA', 'OT3_Pro_Multi', 'OT3_Pro_VCE',
        'OT3_SDWAN_HA', 'SVCST_OOS']]

# Compute and view VIF
vif = pd.DataFrame()
vif["variables"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Sort VIF DataFrame by VIF values in descending order
vif_sorted = vif.sort_values(by="VIF", ascending=False)

# View sorted results
print(vif_sorted)
```

```python
from statsmodels.formula.api import ols

model_reduced = ols(
    'ALARM_Clear ~ CUST_COND_NUMERIC + OT2_HB + SERVICE_POD_NUMBER + '
    'QUEUE_TransHB + SVCST_OOS + ORIGIN_Cust_Netcool + SVCLVL_Mid + '
    'OT2_LAN + OT3_Pro_Multi + SVCLVL_Enterprise + OT3_Outage_Data + '
    'OT3_Pro_HA + CUSTOMER_COMMENTS + SVCLVL_Diamond + OT3_Cust_Down + '
    'OT3_Pro_4GW + OT2_ETH + OT2_SWITCH + QUEUE_WSL + QUEUE_Outage + '
    'OT3_JL_PL + OT3_SDWAN_HA + QUEUE_CSOC + QUEUE_Complex + '
    'OT3_Cust_Deg + QUEUE_MNS + OT3_Pro_VCE + QUEUE_TSM',
    data=df
).fit()

print(model_reduced.summary())
```

```python
from statsmodels.formula.api import ols

model_final = ols(
    'ALARM_Clear ~ CUST_COND_NUMERIC + SERVICE_POD_NUMBER + SVCST_OOS + '
    'ORIGIN_Cust_Netcool + OT2_LAN + OT3_Pro_Multi + SVCLVL_Enterprise + '
    'OT3_Outage_Data + OT3_Pro_HA + CUSTOMER_COMMENTS + OT3_Cust_Down + '
    'OT3_Pro_4GW + OT2_ETH + OT2_SWITCH + QUEUE_Outage + OT3_JL_PL + '
    'OT3_SDWAN_HA + QUEUE_CSOC + OT3_Cust_Deg',
    data=df
).fit()

print(model_final.summary())
```

```python
plt.hist(model_final.resid, rwidth=0.8, bins=5)
plt.title('Final Model Residuals', fontsize=20, pad=10, color='blue')
plt.show()
```