Automated EM Summary Report | Prototype Idea

Introduction & Purpose

As a part of DP Worlds Quality and Planning section in the Engineering Department, a crucial aspect of their work is to ensure that they have a summary of an Emergency Maintenance report for each month.

As a part of my training, I was tasked with ensuring that I understood how to create a summary given an EM excel report. In the report contained a long list of approximately 10,000 sets of data entailing details regarding the description of the problem, the asset type, the cause code, problem code, remedy, and a recount of the work log just to name a few. In addition to creating a summary of the excel file, I was tasked with providing a more technical summary, giving a summary of the most common problems that occur and what was done to amend the issue.

As a result of this, I thought of an idea to automate this process to ensure that time could be saved for those who are constantly going through these long lists of EM Reports monthly.

Idea

The idea for this Automated Summary Report would be to integrate this into DP Worlds current web page for their employees. As such, a following outline of the workings of the idea are as follows:

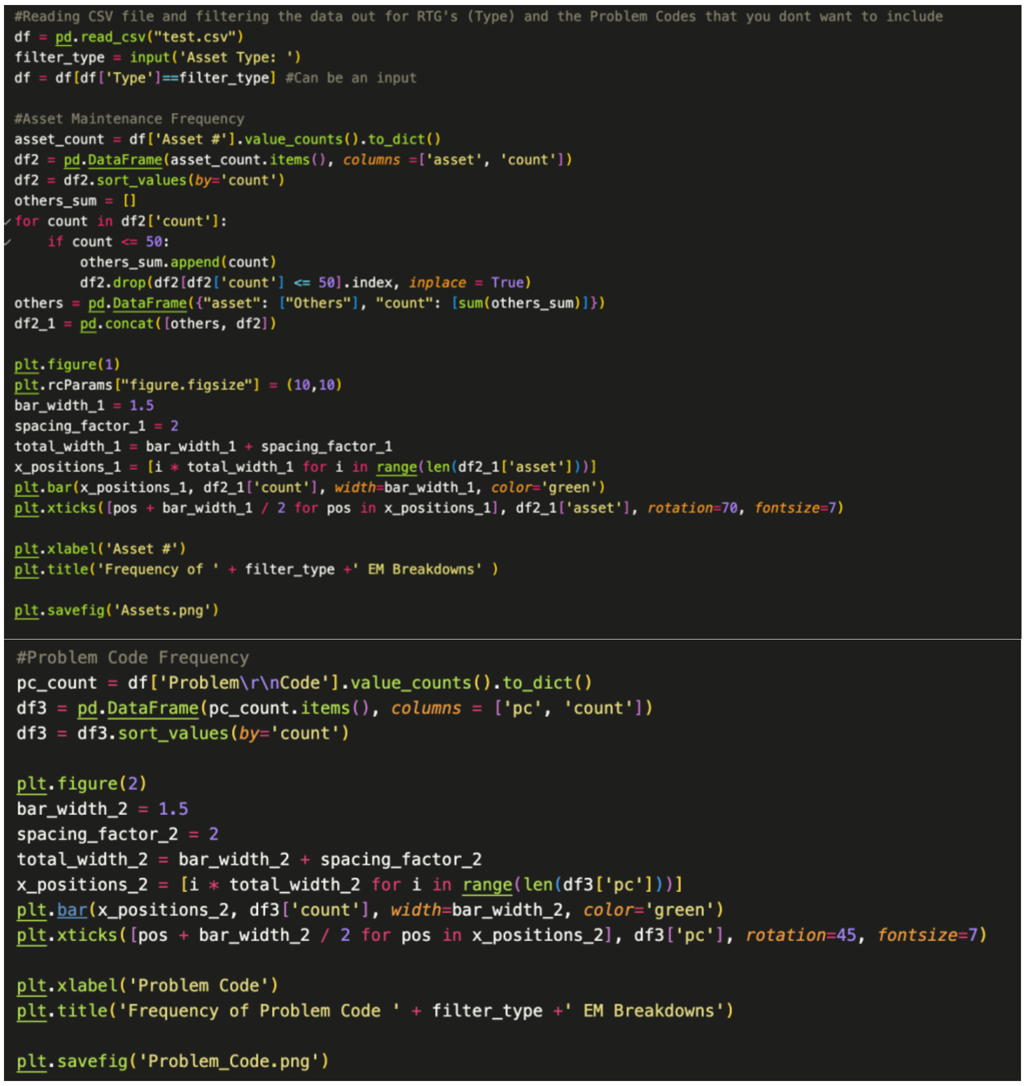
- An excel file is uploaded to the relevant file upload area.
- Drop downs are presented to allow the user to filter the data that they want to be summarized based on the asset type and the problem code.
- After having submitted the required filters, the outputs are provided on the website.
- This would include a graph of the frequency of breakdowns for each asset number, a graph of the frequency of problem codes that arise, and 3 graphs of the frequency of cause codes that occur for the three problem codes that were specified in the filter.
- Additionally, a text summary of the most common problems, causes and what was done to amend the issue are provided for each problem code.

Frontend – JavaScript React would most likely be used with HTML and CSS to integrate this into the current web page that DP world uses for their employees (Coding languages for frontend could be based on what DP World already uses)

Backend – Python can be used to process the data as it is able to provide visualizations for graphs. Additionally, with the use of AI and NLP (Natural Language Processing), a textual summary can also be provided for better understanding for the user.

Progress

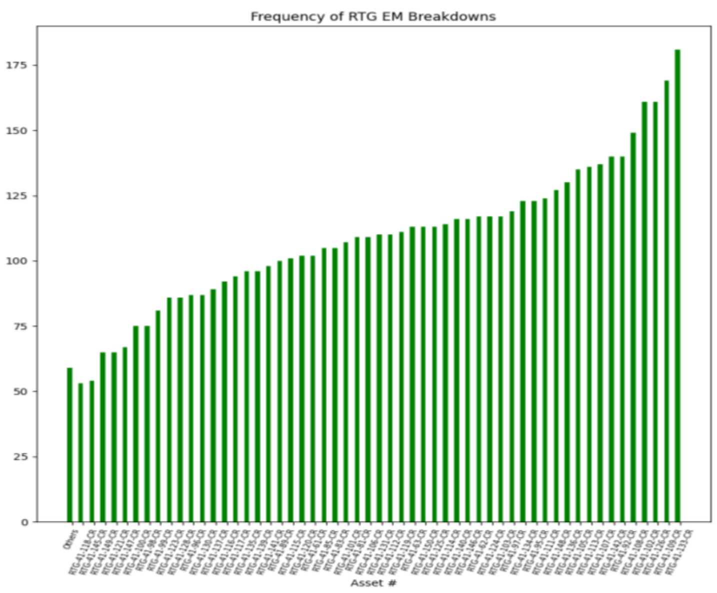
The current idea has only been progressed by doing the backend code using python. This involved using the data from the excel file. As such, this is still only a semi-working prototype that provides all the necessary output:



```
#Reading CSV file and filtering the data out for RTG's (Type) and the Problem Codes that you dont want to include
df = pd.read_csv("test.csv")
filter_type = input('Asset Type: ')
df = df[df['Type']==filter_type] #Can be an input

#Asset Maintenance Frequency
asset_count = df['Asset #'].value_counts().to_dict()
df2 = pd.DataFrame(asset_count.items(), columns =['asset', 'count'])
df2 = df2.sort_values(by='count')
others_sum = []
for count in df2['count']:
    if count <= 50:
        others_sum.append(count)
        df2.drop(df2[df2['count'] <= 50].index, inplace = True)
others = pd.DataFrame({"asset": ["Others"], "count": [sum(others_sum)]})
df2_1 = pd.concat([others, df2])

plt.figure(1)
plt.rcParams["figure.figsize"] = (10,10)
bar_width_1 = 1.5
spacing_factor_1 = 2
total_width_1 = bar_width_1 + spacing_factor_1
x_positions_1 = [i * total_width_1 for i in range(len(df2_1['asset']))]
plt.bar(x_positions_1, df2_1['count'], width=bar_width_1, color='green')
plt.xticks([pos + bar_width_1 / 2 for pos in x_positions_1], df2_1['asset'], rotation=70, fontsize=7)

plt.xlabel('Asset #')
plt.title('Frequency of ' + filter_type +' EM Breakdowns' )

plt.savefig('Assets.png')
```

```
#Problem Code Frequency
pc_count = df['Problem\r\nCode'].value_counts().to_dict()
df3 = pd.DataFrame(pc_count.items(), columns = ['pc', 'count'])
df3 = df3.sort_values(by='count')

plt.figure(2)
bar_width_2 = 1.5
spacing_factor_2 = 2
total_width_2 = bar_width_2 + spacing_factor_2
x_positions_2 = [i * total_width_2 for i in range(len(df3['pc']))]
plt.bar(x_positions_2, df3['count'], width=bar_width_2, color='green')
plt.xticks([pos + bar_width_2 / 2 for pos in x_positions_2], df3['pc'], rotation=45, fontsize=7)

plt.xlabel('Problem Code')
plt.title('Frequency of Problem Code ' + filter_type +' EM Breakdowns')

plt.savefig('Problem_Code.png')
```

Figure 1: Takes in a excel/csv file and based on the input for the required asset type, it will create a graph for the Asset Maintenance Frequency and Problem Code Frequency



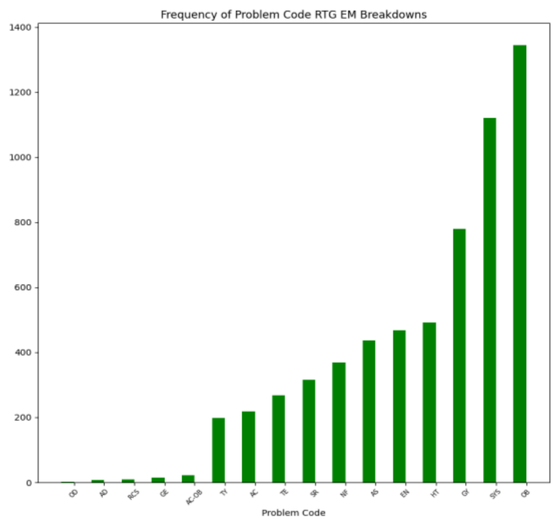
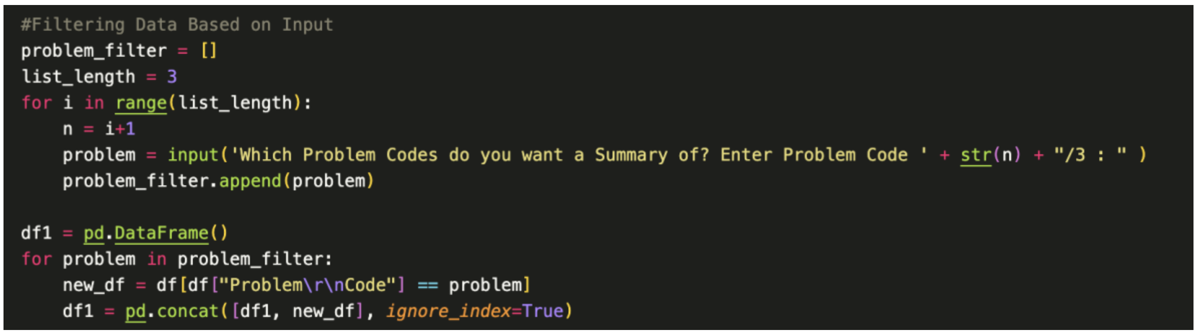
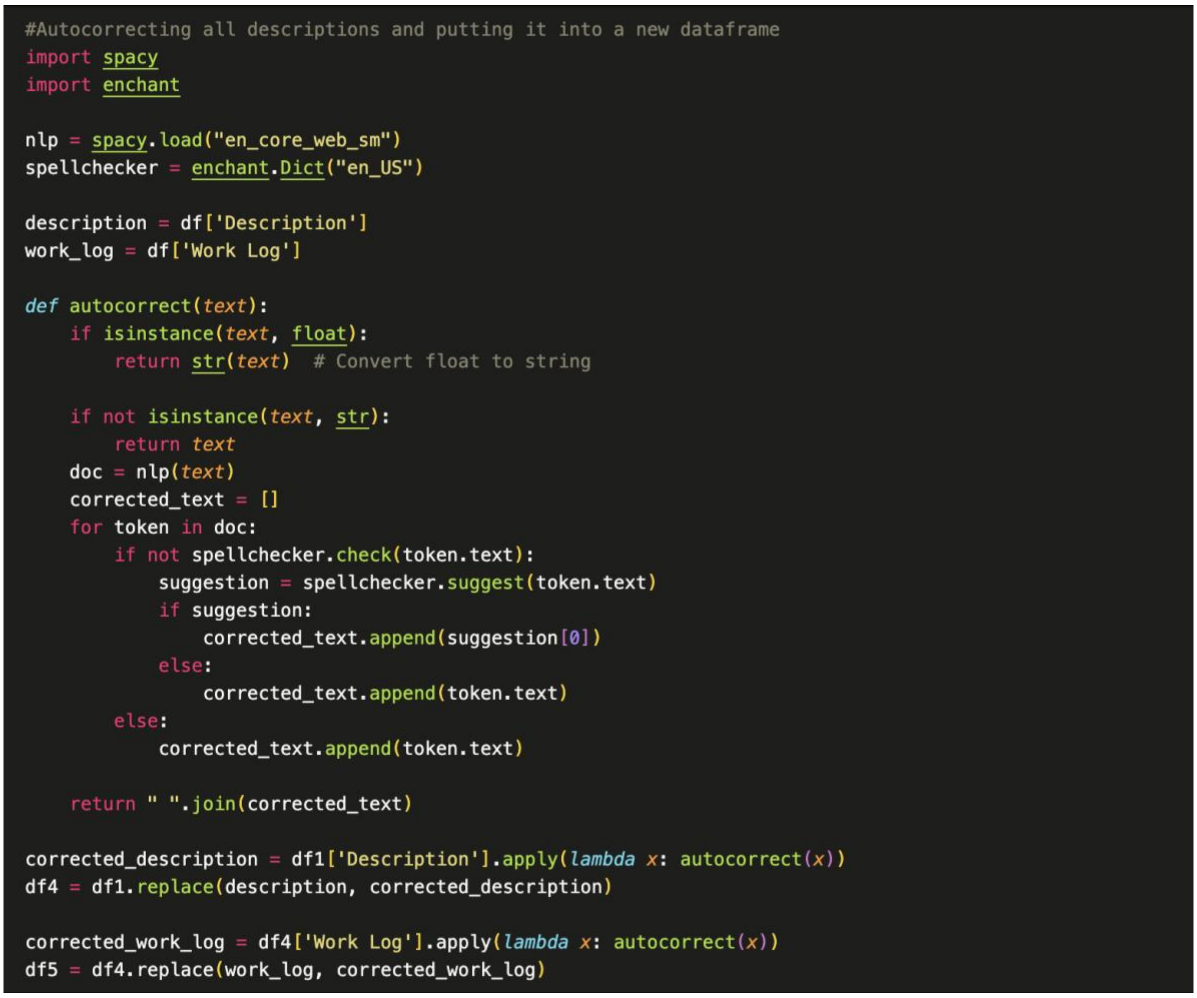


Figure 2a & 2b: Output graphs based on the filter, showing most common problem codes and most common asset breakdowns



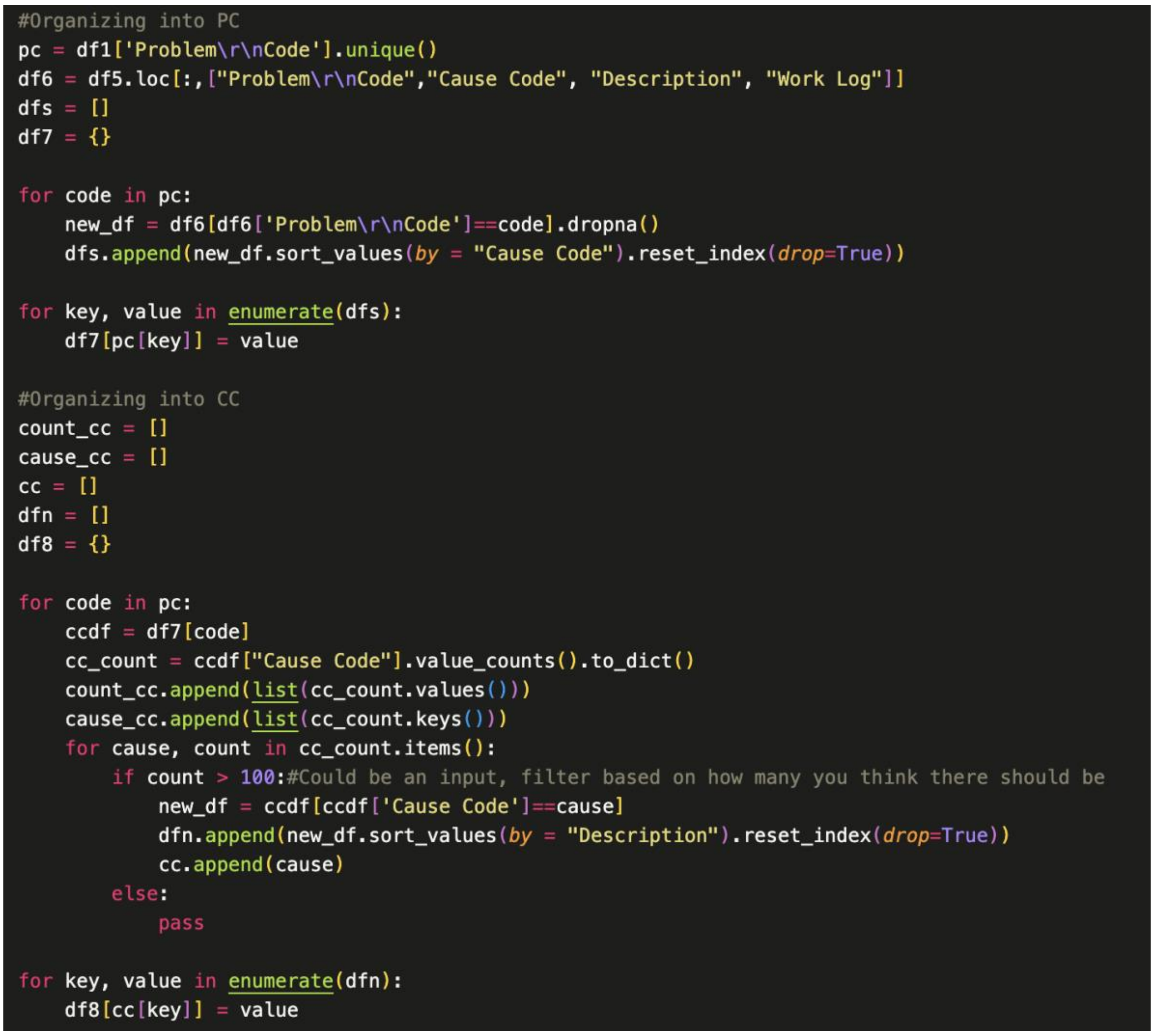
```
#Filtering Data Based on Input
problem_filter = []
list_length = 3
for i in range(list_length):
    n = i+1
    problem = input('Which Problem Codes do you want a Summary of? Enter Problem Code ' + str(n) + "/3 : " )
    problem_filter.append(problem)

df1 = pd.DataFrame()
for problem in problem_filter:
    new_df = df[df["Problem\r\nCode"] == problem]
    df1 = pd.concat([df1, new_df], ignore_index=True)
```

Figure 3: Asked for the user to input the three problem codes that they want a summary of



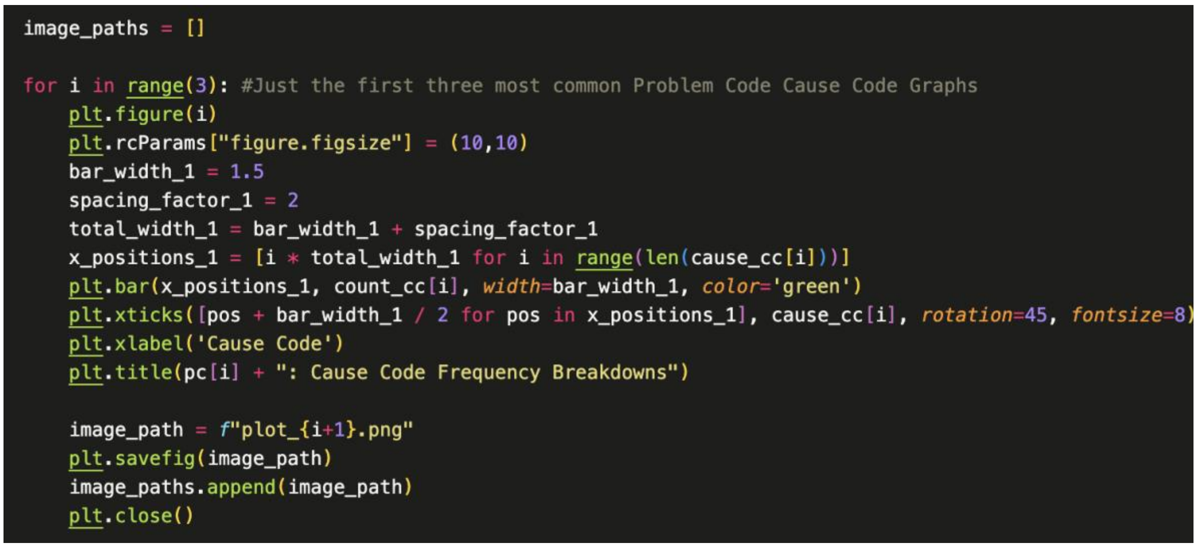
```
#Autocorrecting all descriptions and putting it into a new dataframe
import spacy
import enchant

nlp = spacy.load("en_core_web_sm")
spellchecker = enchant.Dict("en_US")

description = df['Description']
work_log = df['Work Log']

def autocorrect(text):
    if isinstance(text, float):
        return str(text)  # Convert float to string

    if not isinstance(text, str):
        return text
    doc = nlp(text)
    corrected_text = []
    for token in doc:
        if not spellchecker.check(token.text):
            suggestion = spellchecker.suggest(token.text)
            if suggestion:
                corrected_text.append(suggestion[0])
            else:
                corrected_text.append(token.text)
        else:
            corrected_text.append(token.text)

    return " ".join(corrected_text)

corrected_description = df1['Description'].apply(lambda x: autocorrect(x))
df4 = df1.replace(description, corrected_description)

corrected_work_log = df4['Work Log'].apply(lambda x: autocorrect(x))
df5 = df4.replace(work_log, corrected_work_log)
```

Figure 4: An autocorrect function required for the work logs and description by using NLP to ensure that the summarizer has an easier time in summarizing the text



```python
#Organizing into PC
pc = df1['Problem\r\nCode'].unique()
df6 = df5.loc[:,["Problem\r\nCode","Cause Code", "Description", "Work Log"]]
dfs = []
df7 = {}

for code in pc:
    new_df = df6[df6['Problem\r\nCode']==code].dropna()
    dfs.append(new_df.sort_values(by = "Cause Code").reset_index(drop=True))

for key, value in enumerate(dfs):
    df7[pc[key]] = value

#Organizing into CC
count_cc = []
cause_cc = []
cc = []
dfn = []
df8 = {}

for code in pc:
    ccdf = df7[code]
    cc_count = ccdf["Cause Code"].value_counts().to_dict()
    count_cc.append(list(cc_count.values()))
    cause_cc.append(list(cc_count.keys()))
    for cause, count in cc_count.items():
        if count > 100:#Could be an input, filter based on how many you think there should be
            new_df = ccdf[ccdf['Cause Code']==cause]
            dfn.append(new_df.sort_values(by = "Description").reset_index(drop=True))
            cc.append(cause)
        else:
            pass

for key, value in enumerate(dfn):
    df8[cc[key]] = value
```

Figure 5: Organizing the dataset into their respective problem codes, and then from that subsection of cause codes, organizing it by cause codes.



```python
image_paths = []

for i in range(3): #Just the first three most common Problem Code Cause Code Graphs
    plt.figure(i)
    plt.rcParams["figure.figsize"] = (10,10)
    bar_width_1 = 1.5
    spacing_factor_1 = 2
    total_width_1 = bar_width_1 + spacing_factor_1
    x_positions_1 = [i * total_width_1 for i in range(len(cause_cc[i]))]
    plt.bar(x_positions_1, count_cc[i], width=bar_width_1, color='green')
    plt.xticks([pos + bar_width_1 / 2 for pos in x_positions_1], cause_cc[i], rotation=45, fontsize=8)
    plt.xlabel('Cause Code')
    plt.title(pc[i] + ": Cause Code Frequency Breakdowns")

    image_path = f"plot_{i+1}.png"
    plt.savefig(image_path)
    image_paths.append(image_path)
    plt.close()
```

Figure 6: Creates a plot for problem code specified with the frequency of cause codes

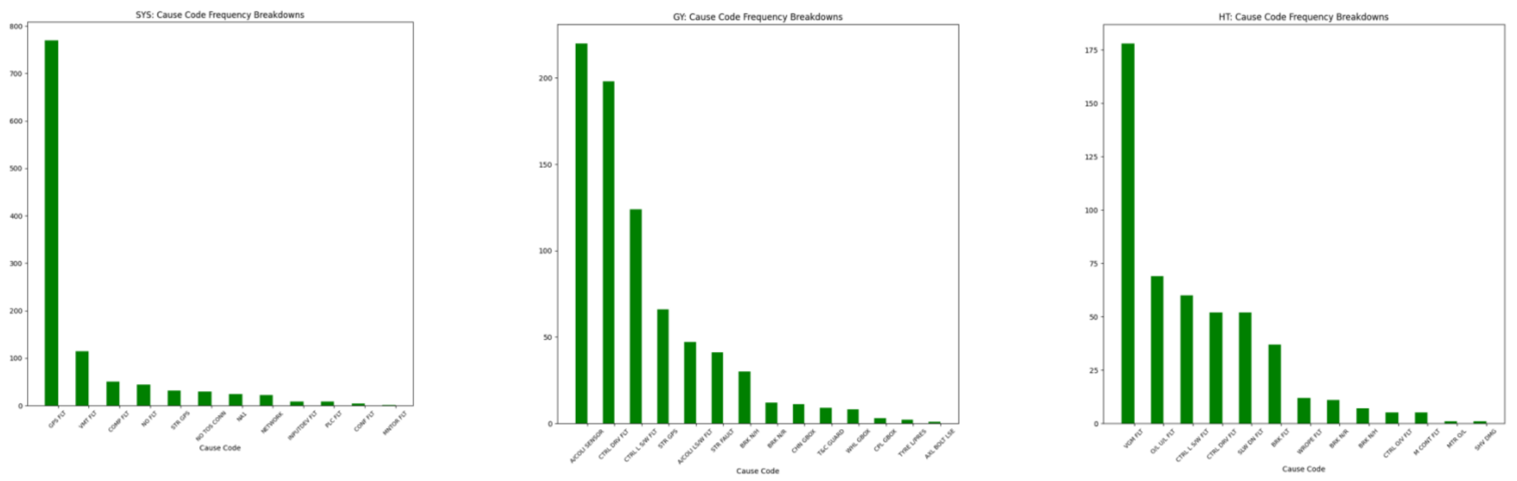
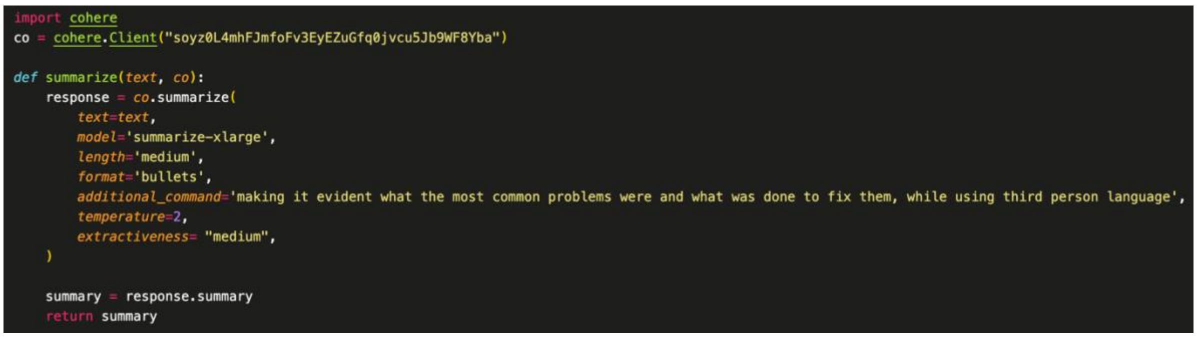
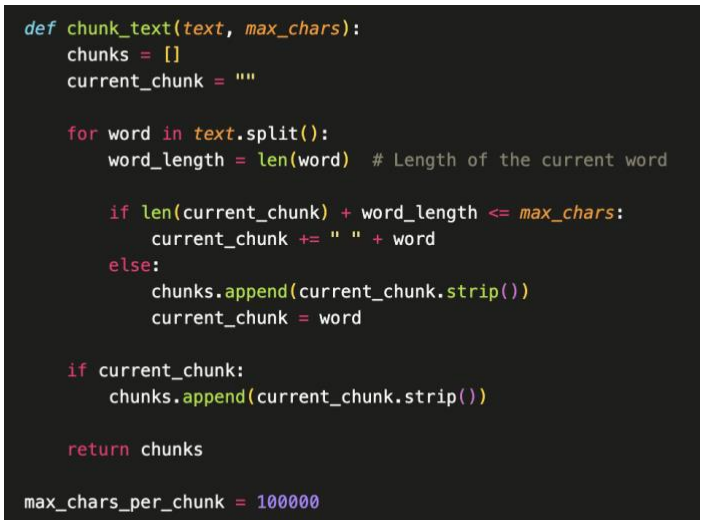


Figure 7a, 7b, & 7C: Output graphs based on the specified problem codes, showcasing the most common problem codes for thos
problem codes



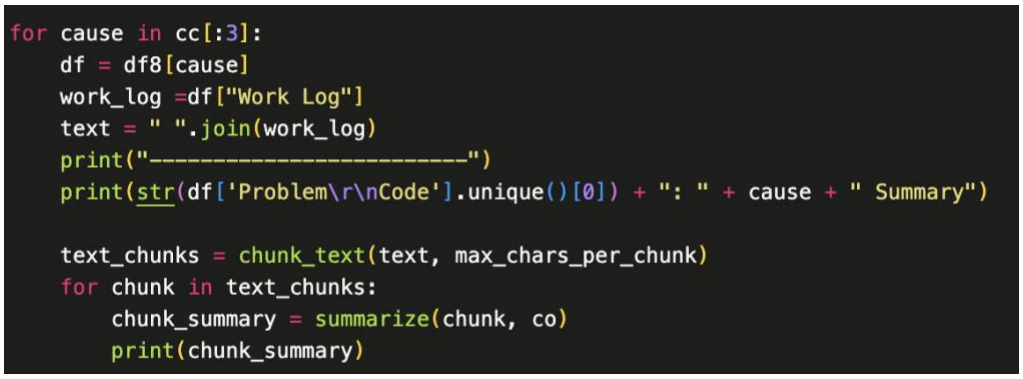
```
import cohere
co = cohere.Client("soyz0L4mhFJmfoFv3EyEZuGfq0jvcu5Jb9WF8Yba")

def summarize(text, co):
    response = co.summarize(
        text=text,
        model='summarize-xlarge',
        length='medium',
        format='bullets',
        additional_command='making it evident what the most common problems were and what was done to fix them, while using third person language',
        temperature=2,
        extractiveness= "medium",
    )

    summary = response.summary
    return summary
```

Figure 8: Using a cohere AI API to implement into the code. This function is what processes the text and provides a summary.



```
def chunk_text(text, max_chars):
    chunks = []
    current_chunk = ""

    for word in text.split():
        word_length = len(word)  # Length of the current word

        if len(current_chunk) + word_length <= max_chars:
            current_chunk += " " + word
        else:
            chunks.append(current_chunk.strip())
            current_chunk = word

    if current_chunk:
        chunks.append(current_chunk.strip())

    return chunks

max_chars_per_chunk = 100000
```

Figure 9: Function used to process all the text in the work logs as a single text rather than a list of different texts. Since a free version of the API is used, it can only take in 10,000 characters per summary, so the relevant feature was included in the function.



```
for cause in cc[:3]:
    df = df8[cause]
    work_log =df["Work Log"]
    text = " ".join(work_log)
    print("------------------------")
    print(str(df['Problem\r\nCode'].unique()[0]) + ": " + cause + " Summary")

    text_chunks = chunk_text(text, max_chars_per_chunk)
    for chunk in text_chunks:
        chunk_summary = summarize(chunk, co)
        print(chunk_summary)
```

Figure 10: Using the functions in ***Figure 8*** and ***9*** to create a summary for each cause code for the specified problem codes (Function differs to the idea due to technical difficulties. Will be discussed in **Possible Improvements**)

The following text is the output of the summary provided based on the work logs:

SYS: GPS FLT Summary
- Problems with the GPS system included: loss of signal, no power, high deviation, gantry going out, and the system freezing.
- To fix these problems, power was reset, and the system was monitored.
- In some cases, the GPS was reset, and the system was tested again.
- Auto-steering and auto-update were not working, but after power reset and testing, the system was functioning properly.
- Some problems, such as a faulty antenna, were not able to be fixed.

SYS: VMT FLT Summary
- One of the primary roles of a system technician is to resolve any technical issues that arise during the day-to-day operation of the VMT system.
- System technicians need to be familiar with VMT and have good knowledge of networks, and communication protocols to help with the diagnosis of the problem and find the solution.
- Some of the common issues faced by system technicians are: VMT not connecting to the network, keyboard not working, VMT not powering on, VMT auto-update not working, and so on.

GY: A/COLI SENSOR Summary
- Gantry problems such as anticollision sensors being activated, faulty wiring, and more caused the gantry to not work properly.
- To fix these issues, the team performed maintenance tasks such as checking and cleaning sensors, repairing faulty wiring, and adjusting the radar sensor angle position.
- After performing these tasks, the gantry operations were monitored to ensure that they were working properly.

Possible Improvements

Although this is still just a prototype idea for what could be an automated process, the code still provides useful technical information regarding the most common problem codes, cause codes and which assets require the most maintenance. Additionally, the text summary provided gives a better technical understanding of exactly what are the most common issues and how they were amended. There are however some drawbacks of the prototype.

The first drawback is that instead of accepting an excel file, the code accepts a csv file instead. It is possible to change the code so that it accepts an excel file, yet I was unable to make it work although the code is very similar. This poses a small issue regarding ease of use as one would most likely have to change the file into a csv file to summarize it, yet this feature can always be changed in the future.

The second drawback is the cohere AI API that is being used. As of now, the current version that I have used is the free version which only allows 5 entries per minute and only allows a maximum input of 10,000 characters. This means that although the work logs for each section may have more than 10,000 characters, they must be split up into different 10,000-character chunks, making the summaries longer than required and taking up more of the 5 entries per minute input for the API. This is the reason why a summary of the most common cause codes amongst the three problem codes were provided instead of an overall summary of the each of the three specified problem codes. If a paid version was used, this could be a possibility.

Another issue is with the quality of the summary provided. The first problem is that the summary is quite inconsistent as it will provide a different summary each time even with the same filter inputs. Additionally, the quality of the summaries itself are relatively low quality and don't provide as much information as could be used. If this project would be developed more, a different AI API such as the chatgpt API would be used instead due to its higher quality of summarization. In this, it also allows for open class classification which would take in the work logs and create a summary for the most common types of cause codes for each problem code, organizing it automatically.