## Python Data Cleaning, Exploration and Visualization

### Importing the suppliers sheet into the data frame.

The below code downloads a publicly accessible Google Sheet as an Excel file. It first extracts the file ID from the sheet's URL and then constructs a download URL for the Excel version. The `requests` library is used to fetch the file content, which is then saved locally as an Excel file. This step is likely a prerequisite for subsequent data analysis using pandas or other tools.

```
import pandas as pd
# requests is imported to handle HTTP requests, to allow download of files from
the internet
import requests

# URL of the publicly accessible Google Sheet
sheet_url = "https://docs.google.com/spreadsheets/d/1rBcidV-
_Bdx8q1ILMnTnSCDzwK0C4QAm/edit?usp=sharing"

# Extracting the file ID from the URL
file_id = sheet_url.split("/")[-2]
download_url =
f"https://docs.google.com/spreadsheets/d/{file_id}/export?format=xlsx"

# Downloading the Excel file
response = requests.get(download_url)

# Saving the file locally
with open('downloaded_file.xlsx', 'wb') as f:
        f.write(response.content)
```

The code below loads the downloaded Excel file into individual pandas DataFrames for each sheet. This step is crucial for organizing and managing the data from different sources within the file. By separating the data into DataFrames based on their respective sheets, we can analyze and process each part independently, making the data analysis process more efficient and manageable.

```
# Loading data form Excel into dataframes
sheet_names = ['Vendor', 'Plant', 'Defected Items', 'Material Type', 'Defects',
'Defect Type', 'Category']
dfs = pd.read_excel('downloaded_file.xlsx', sheet_name=sheet_names)
# Extracting dataframes
vendor_df, plant_df, defected_items_df, material_type_df, defects_df,
defect_type_df, category_df = dfs.values()
```

## Part 1: Data Cleaning and Preparation

### Step 1: Merging Defected Items with all necessary lookups

The code below merges the individual DataFrames into a single DataFrame named merged_df. This is done by successively merging each DataFrame with the merged_df based on common identifiers (foreign keys). This step combines information from different sources into a comprehensive dataset, enabling more effective analysis.

```
# Part 1: Data Cleaning and Preparation
# Step 1: Merging Defected Items with all necessary lookups
 merged_df = defected_items_df.merge(vendor_df, on='Vendor ID', how='left') \
                              .merge(plant_df, on='Plant ID', how='left') \
                              .merge(material_type_df, on='Material Type ID',
how='left') \
                              .merge(defects_df, on='Defect ID', how='left') \
                              .merge(defect_type_df, on='Defect Type ID',
how='left') \
                              .merge(category_df, on='Sub Category ID',
how='left')

print("Data merged successfully.")
```

### Step 2 and 3: Cleaning Text data and Removing duplicate values

The code below cleans and prepares the merged DataFrame by removing extra whitespace from text columns and eliminating duplicate rows. This ensures data quality and consistency, which is essential for accurate and reliable analysis.

```
import pandas as pd
# Step 2: Cleaning Text Data
# Trimming extra spaces
def clean_text(text):
    if isinstance(text, str):
        # Trim whitespace
        return text.strip()
    return text

# Applying clean_text function to all string columns in the DataFrame
string_columns = merged_df.select_dtypes(include=['object']).columns

for col in string_columns:
    merged_df[col] = merged_df[col].apply(clean_text)
# Step 3: Removing Duplicate Values
#Removing the duplicate "Category" column
merged_df = merged_df.drop(columns=['Category'])
```

```
#Removing duplicate rows based on all columns
merged_df_cleaned = merged_df.drop_duplicates()


# Preview of the cleaned DataFrame
print("Preview of the cleaned data:")
print(merged_df_cleaned.head())
```

### Step 4: Data Type Conversions

The code  below converts data types in the `merged_df_cleaned` DataFrame to ensure they are appropriate for subsequent analysis. It converts all columns to string, then converts specific columns like 'Downtime min' and 'Defect Qty' to numeric for calculations, and finally converts the 'Date' column to datetime for time-based analysis. This step is crucial for accurate and meaningful data analysis.

```
# Step 4: Data Type Conversions
# Changing all Data Types to String
merged_df_cleaned = merged_df_cleaned.astype('string')

# Converting Downtime and Defect Qty columns to whole numbers
merged_df_cleaned['Downtime min'] = pd.to_numeric(merged_df_cleaned['Downtime
min'], errors='coerce').fillna(0).astype(int)
merged_df_cleaned['Defect Qty'] = pd.to_numeric(merged_df_cleaned['Defect
Qty'], errors='coerce').fillna(0).astype(int)
# Converting Date column to Date
merged_df_cleaned['Date'] = pd.to_datetime(merged_df_cleaned['Date'],
errors='coerce')

# Printing data types to confirm
print("\nData types")
print(merged_df_cleaned.dtypes)
```

The code below splits the 'Plant' column into two new columns: 'Plant Name' and 'Plant State'. This is done by splitting the column based on the rightmost space and assigning the results to the new columns. The original 'Plant' column is then dropped. This step is useful when the 'Plant' column contains combined information that needs to be analyzed separately.

```
# Step 5: Splitting Column Values
# Splitting the 'Plant' column by the rightmost space delimiter
split_columns = merged_df_cleaned['Plant'].str.rsplit(' ', n=1, expand=True)

# Assigning the split results to new columns
merged_df_cleaned['Plant Name'] = split_columns[0]
merged_df_cleaned['Plant State'] = split_columns[1].fillna('')  # Fill NaN with
```

```
empty string if no state found

# Removing any commas from the end of 'Plant State'
merged_df_cleaned['Plant Name'] = merged_df_cleaned['Plant
Name'].str.rstrip(',')

# Dropping the original 'Plant' column
merged_df_cleaned = merged_df_cleaned.drop(columns=['Plant'])
```

**Step 5: Removing Null value Row:**

The code below removes rows from the DataFrame that contain more than four null values. This helps to clean the data by eliminating incomplete or inaccurate rows, improving the quality and reliability of subsequent analysis.

```
# Remove rows with more than 4 null or zero values
merged_df_cleaned = merged_df_cleaned.dropna(thresh=5)
```

```
!pip install pyspellchecker
```

**Step 6: Spell Check and Captilzation**

The code below addresses potential inconsistencies in the data by performing spell checking on the 'Defect' column and capitalizing the first letter of each word in specified text columns. This ensures data consistency and improves the overall quality of the dataset, leading to more accurate and reliable analysis.

```
from spellchecker import SpellChecker
# Step 6: Spell check and Capitalization
# Function to capitalize the first letter of each word in text columns
def capitalize_text_columns(merged_df_cleaned, text_columns):
    for col in text_columns:
        merged_df_cleaned[col] = merged_df_cleaned[col].apply(lambda x:
x.title() if isinstance(x, str) else x)
    return merged_df_cleaned

# Function to perform spell checking on the "Defect" column
def spell_check_defects(merged_df_cleaned):
    spell = SpellChecker()
    merged_df_cleaned['Defect'] = merged_df_cleaned['Defect'].apply(
        lambda x: ' '.join([spell.correction(word) or word for word in
x.split()]) if isinstance(x, str) else x
    ) # Added or word to return the original word if spell.correction returns
None
    return merged_df_cleaned
```

```
# List of text columns to apply capitalization
text_columns = ['Vendor', 'Plant Name', 'Plant State', 'Material Type', 'Defect
Type']

# Capitalize text columns
merged_df_cleaned = capitalize_text_columns(merged_df_cleaned, text_columns)

# Perform spell check on "Defects" column only
merged_df_cleaned = spell_check_defects(merged_df_cleaned)
```

```
# Preview Print
print("Preview of the DataFrame:")
display(merged_df_cleaned.head())
```

## Part 2: Data Analytics and Exploration

The code below calculates summary metrics from the cleaned DataFrame, including total rows, total downtime, and total defect quantity. It then creates a pandas DataFrame to store these metrics and exports the DataFrame to an Excel file for easy sharing and further analysis.

```
# Part 2: Data Analytics and Exploration
# Calculating main analytics
total_rows = merged_df_cleaned.shape[0]
total_downtime = merged_df_cleaned['Downtime min'].sum()
total_defected_qty = merged_df_cleaned['Defect Qty'].sum()

# Creating a summary DataFrame
analytics_summary = pd.DataFrame({
    'Metric': ['Total Rows', 'Total Downtime (min)', 'Total Defect Qty'],
    'Value': [total_rows, total_downtime, total_defected_qty]
})

# Exporting the analytics to an Excel file
analytics_summary.to_excel('analytics_summary.xlsx', index=False)

# Printing the summary
print("Analytics Summary:")
print(analytics_summary)
```

The code below calculates descriptive statistics for the 'Downtime min' and 'Defect Qty' columns in the DataFrame. This provides insights into the distribution, central tendency, and dispersion

of these numerical variables, which is valuable for understanding the data and identifying potential trends or patterns.

```
# Data Description
description = merged_df_cleaned[['Downtime min', 'Defect Qty']].describe()
print("Data Description for 'Downtime min' and 'Defect Qty':")
display(description)
```

## Data Exploration and Visualization

The code below creates **histograms** for the 'Defect Quantity' and 'Downtime min' columns in the DataFrame. These histograms visually represent the distribution of these numerical variables, allowing for a better understanding of their characteristics and potential trends.

```
import matplotlib.pyplot as plt
# Part 3:Data Visualization
# Creating and displaying Histograms of Defect Quantity and Downtime
# Create a figure with 1 row and 2 columns
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot the histogram for Defect Quantity
ax1.hist(merged_df_cleaned['Defect Qty'], bins=30, color='grey',
edgecolor='black')
ax1.set_title('Histogram of Defect Quantity')
ax1.set_xlabel('Defect Quantity')
ax1.set_ylabel('Frequency')
ax1.grid(False)

# Plot the histogram for Downtime
ax2.hist(merged_df_cleaned['Downtime min'], bins=30, color='#ae123a',
edgecolor='black')
ax2.set_title('Histogram of Downtime')
ax2.set_xlabel('Downtime (min)')
ax2.set_ylabel('Frequency')
ax2.grid(False)

# Adjust spacing for better readability
plt.tight_layout()

# Display the plot
plt.show()
```

The code creates a **scatter plot** to visualize the relationship between 'Downtime min' and 'Defect Qty' in the DataFrame. This allows us to explore potential correlations or patterns between these two variables, which can be helpful for understanding how changes in one variable might be associated with changes in the other.

```
import matplotlib.pyplot as plt
# Part 3:Data Visualization
# Creating and displaying Histograms of Defect Quantity and Downtime
# Create a figure with 1 row and 2 columns
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot the histogram for Defect Quantity
ax1.hist(merged_df_cleaned['Defect Qty'], bins=30, color='grey',
edgecolor='black')
ax1.set_title('Histogram of Defect Quantity')
ax1.set_xlabel('Defect Quantity')
ax1.set_ylabel('Frequency')
ax1.grid(False)

# Plot the histogram for Downtime
ax2.hist(merged_df_cleaned['Downtime min'], bins=30, color='#ae123a',
edgecolor='black')
ax2.set_title('Histogram of Downtime')
ax2.set_xlabel('Downtime (min)')
ax2.set_ylabel('Frequency')
ax2.grid(False)

# Adjust spacing for better readability
plt.tight_layout()

# Display the plot
plt.show()
```

The code below identifies the **top 10 vendors** with the highest number of defects by calculating the total defect quantity for each vendor. It then visualizes the results using a bar chart, providing a clear representation of the distribution of defects among the vendors. This information can be valuable for quality control efforts and identifying areas for improvement.

```
# Analyze defect quantities by vendor
top_10_vendors = (
    merged_df_cleaned.groupby('Vendor')['Defect Qty']
    .sum()  # Calculate the total defect quantity for each vendor
    .rename('Total_Defects')  # Rename the resulting Series for clarity
    .nlargest(10)  # Extract the top 10 vendors with the highest defect
```

```
quantity
)

# Create a bar chart to visualize the results
fig, ax = plt.subplots(figsize=(12, 6))  # Create a figure with an axis
ax.bar(top_10_vendors.index, top_10_vendors.values, color='#ae123a', width =
0.5)  # Plot the bar chart

# Set the title and labels using the axis object
ax.set_title('Top 10 Vendors by Total Defect Quantity')
ax.set_ylabel('Total Defect Quantity')
ax.set_xlabel('Vendor')
ax.tick_params(axis='x', rotation=45)  # Rotate x-axis labels

# Display the bar chart
plt.tight_layout()
plt.show()
```

The code below exports the cleaned DataFrame to an Excel file and then downloads the file to your local machine. This allows you to save and share the cleaned data for further analysis or use in other software tools.

```
from google.colab import files
# Part 4: Data Exportation
export_path = 'cleaned_data.xlsx'

# Exporting the cleaned DataFrame to Excel
merged_df_cleaned.to_excel(export_path, index=False)

# Downloading the file
files.download(export_path)
```

### More Data Exploration:

The code below creates **a heatmap to visualize the frequency of defects by plant and material** type. This allows you to identify patterns and correlations between these factors, potentially revealing areas where specific material types might be more prone to defects at certain plants.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Create the heatmap using seaborn
def create_heatmap(merged_df_cleaned):
  defect_frequency = pd.pivot_table(
```

```
        merged_df_cleaned,
        values='Defect Qty',
        index='Plant Name',
        columns='Material Type',
        aggfunc='sum',
        fill_value=0
    )

    # Create the heatmap with a dedicated figure
    fig, ax = plt.subplots(figsize=(12, 8))  # Use subplots for better control

    sns.heatmap(defect_frequency, annot=True, cmap='viridis', fmt='d', ax=ax)
    ax.set_title('Defect Frequency by Plant and Material Type')
    ax.set_xlabel('Material Type')
    ax.set_ylabel('Plant Name')
    plt.show()

# Assuming you have your merged_df_cleaned data
create_heatmap(merged_df_cleaned)
```

**Python Workbook Link**

https://colab.research.google.com/drive/1dzd7lGdtzl0uZaQSHIo6e_hsRtsnDc6S?usp=sharing#scrollTo=iv-2OdsI7a7G