Use Case

Airplane Maintenance Logs

Dataset:

E8

	A	B	C
1	IDENT	PROBLEM	ACTION
2	100001	ENGINE IDLE OVERRIDE KILLED ENGINE.	TRIED TO ADJUST IDLE SEVERAL TIMES, WOULDN'T ADJUST.
3	100002	ENGINE IDLE OVERRIDE KILLED ENGINE.	REMOVED & REPLACED FUEL SERVO
4	100003	ENGINE IDLE OVERRIDE KILLED ENGINE.	A/C WAS RUN UP, SET IDLE SPEED, MIXTURE OK, NO LEAKS NO
5	100004	HAD ENGINE CHOKE & BRIEFLY LOSE POWER ON DEPARTURE. FULL THR	PERFORMED ENGINE RUN UP, FOUND CYL 2 LOWER PLUG FOULED.
6	100005	#2 & 4 CYL ROCKER COVER GASKETS ARE LEAKING.	REMOVED & REPLACED GASKETS.
7	100006	ROCKER BOX COVER SCREWS LOOSE (ALL CYL).	TIGHTENED SCREWS.
8	100007	INDUCTION TUBE HOSE CLAMPS LOOSE (ALL CYL).	TIGHTENED HOSE CLAMPS.
9	100008	#3 INTAKE IS LEAKING.	REMOVED & REPLACED GASKET.
10	100009	#2 INTAKE IS LEAKING.	REMOVED & REPLACED #2 INTAKE GASKET.
11	100010	#4 ROCKER COVER IS LEAKING.	REMOVED & REPLACED #4 ROCKER COVER GASKET.
12	100011	HOLDING SHORT OF RUNWAY, AFTER COMPLETION OF RUN UP, ENGINE	PERFORMED ENGINE RUN UP, ADJUSTED IDLE & CLEANED SPARK
13	100012	L/H REAR ENGINE BAFFLE CRACKED.	STOP DRILLED CRACK.
14	100013	INDUCTION TUBE HOSE CLAMPS LOOSE (ALL CYLS).	TIGHTENED HOSE CLAMPS.
15	100014	#1, 2 & 3 ROCKER BOX COVER GASKETS LEAKING.	INSTALLED NEW ROCKER BOX COVER GASKETS.
16	100015	#2 ROCKER COVER GASKET LEAKING.	REMOVED & REPLACED #2 ROCKER COVER GASKET.
17	100016	#1 & 3 ROCKER COVER GASKET LEAKING.	REMOVED & REPLACED #1 & 3 ROCKER COVER GASKET.
18	100017	ROCKER BOX COVER SCREWS LOOSE.	TIGHTENED SCREWS.
19	100018	#1 ENGINE BAFFLE CRACKED.	FABRICATED PATCH OF LIKE MATERIAL & RIVETED IAW
20	100019	#1 INTAKE GASKET LEAKING.	REPLACED W/ NEW GASKET.
21	100020	ROCKER COVER FOR CYL #1 IS LEAKING.	REPLACED ROCKER COVER.
22	100021	CYL #4 INTAKE LEAKING.	REPLACED GASKET. LEAK CHECK GOOD.

Aircraft_Annotation_DataFile

Description:

Data contains maintenance logs with an identifier for each entry, a description of the problem encountered, and the action taken to resolve it and here there are three attributes which represents

- **IDENT**: Unique identification number for each maintenance issue or action.
- **PROBLEM**: Description of the specific issue or problem encountered with the aircraft or engine.
- **ACTION**: Description of the maintenance action taken to address or resolve each reported problem.

Code for uploading the dataset:

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd

# Assuming the file is in the root of your Google Drive
file_path = '/content/drive/MyDrive/Aircraft_Annotation_DataFile.csv'
df = pd.read_csv(file_path) # Added this line to read the CSV into a DataFrame
display(df.head()) # Using display for better output formatting
display(df.columns) # Using display for better output formatting
```

Output:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

	IDENT	PROBLEM	ACTION
0	100001	ENGINE IDLE OVERRIDE KILLED ENGINE.	TRIED TO ADJUST IDLE SEVERAL TIMES, WOULDN'T A...
1	100002	ENGINE IDLE OVERRIDE KILLED ENGINE.	REMOVED & REPLACED FUEL SERVO
2	100003	ENGINE IDLE OVERRIDE KILLED ENGINE.	A/C WAS RUN UP, SET IDLE SPEED, MIXTURE OK, NO...
3	100004	HAD ENGINE CHOKE & BRIEFLY LOSE POWER ON DEPAR...	PERFORMED ENGINE RUN UP, FOUND CYL 2 LOWER PLU...
4	100005	#2 & 4 CYL ROCKER COVER GASKETS ARE LEAKING.	REMOVED & REPLACED GASKETS.

```
Index(['IDENT', 'PROBLEM', 'ACTION'], dtype='object')
```

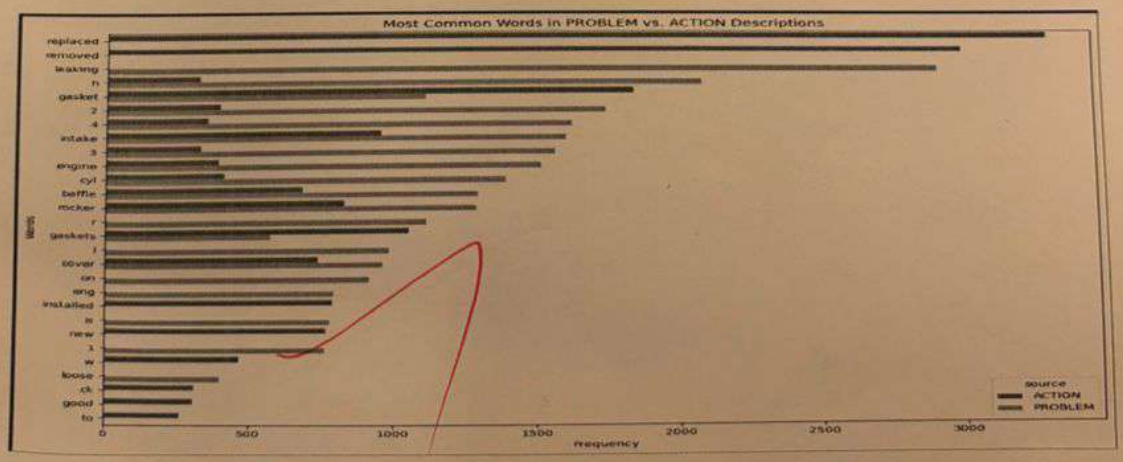
Inference:

Shows the first five rows from the dataset which has been uploaded as the head() has been given

Code for counting the most common words from the dataset:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Combine the common words data with a source indicator
problem_df = pd.DataFrame(common_problem_words, columns=['word', 'count'])
problem_df['source'] = 'PROBLEM'
action_df = pd.DataFrame(common_action_words, columns=['word', 'count'])
action_df['source'] = 'ACTION'
combined_df = pd.concat([problem_df, action_df])
# Sort the combined data by count for better visualization
combined_df = combined_df.sort_values(by='count', ascending=False)
plt.figure(figsize=(12, 8))
sns.barplot(x='count', y='word', hue='source', data=combined_df, palette='viridis')
plt.title('Most Common Words in PROBLEM vs. ACTION Descriptions')
plt.xlabel('Frequency')
plt.ylabel('Words')
plt.tight_layout()
plt.show()
```

Output:



Inference:

The bar plot shows a clear comparison of the most frequent words in both the 'PROBLEM' and 'ACTION' descriptions. By visually inspecting the length of the bars for each word and source, you can infer which specific words appear more often in problems reported versus the actions taken. For example, if the bar for 'leaking' is significantly longer in the 'PROBLEM' section compared to the 'ACTION' section, it suggests 'leaking' is a frequently reported issue. Conversely, if words like 'replaced' or 'repaired' are prominent in the 'ACTION' section, it indicates these are common maintenance activities.

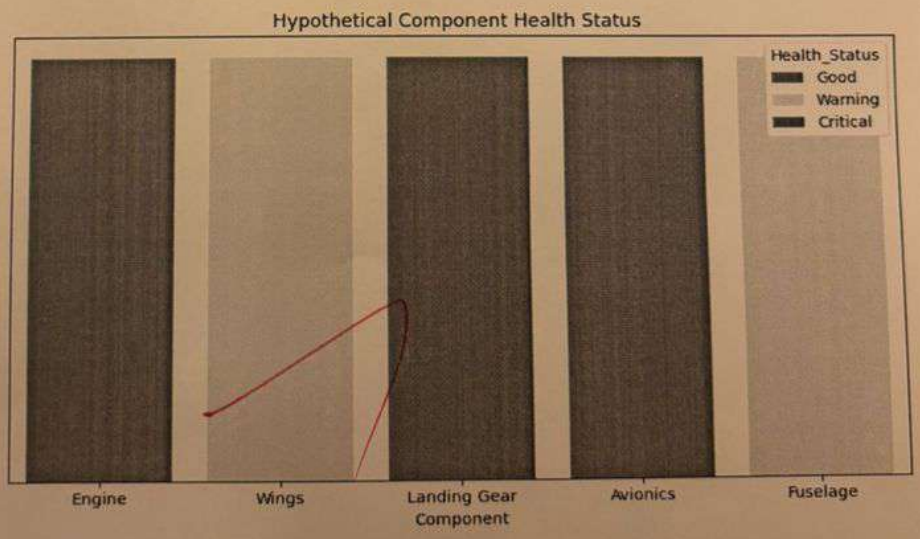
Questions:

1. Explain color schemes for indicating component health.

Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
data = {'Component': ['Engine', 'Wings', 'Landing Gear', 'Avionics', 'Fuselage'],
        'Health_Status': ['Good', 'Warning', 'Critical', 'Good', 'Warning']}
health_df = pd.DataFrame(data)
health_palette = {'Good': 'green', 'Warning': 'yellow', 'Critical': 'red'}
plt.figure(figsize=(8, 5))
sns.barplot(x='Component', y=[1]*len(health_df), hue='Health_Status',
data=health_df, palette=health_palette, dodge=False)
plt.title('Hypothetical Component Health Status')
plt.ylabel('')
plt.yticks([])
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

- The bar chart visually represents the health status of different components using a traffic light color scheme (Green for Good, Yellow for Warning, Red for Critical).
- From this specific example, we can infer that the 'Engine' and 'Avionics' components are currently in a 'Good' health state, the 'Wings' and 'Fuselage' are in a 'Warning' state, and the 'Landing Gear' is in a 'Critical' state.

2. Visualization pipeline from raw maintenance logs to dashboards.

Pipeline:

1. Data Loading.
2. Data cleaning & Data Preprocessing.
3. Exploratory Data Analysis.
4. Visualization Planning & Creation.
5. Dashboard Construction.
6. Inference & Reporting.

Code:

```
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk

# Download necessary NLTK data
try:
nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
nltk.download('punkt')
try:
nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
nltk.download('stopwords')

# Convert to lowercase
df['PROBLEM'] = df['PROBLEM'].str.lower()
df['ACTION'] = df['ACTION'].str.lower()

# Remove punctuation and tokenize
df['problem_tokens'] = df['PROBLEM'].apply(lambda x:
word_tokenize(re.sub(r'[^\w\s]', '', x)))
df['action_tokens'] = df['ACTION'].apply(lambda x:
word_tokenize(re.sub(r'[^\w\s]', '', x)))

# Remove stop words and digits
stop_words = set(stopwords.words('english'))
```

```
df['problem_tokens'] = df['problem_tokens'].apply(lambda tokens: [word for
word in tokens if word not in stop_words and not word.isdigit()])
df['action_tokens'] = df['action_tokens'].apply(lambda tokens: [word for word
in tokens if word not in stop_words and not word.isdigit()])

display(df[['PROBLEM', 'problem_tokens', 'ACTION', 'action_tokens']].head())
```

Output:

	PROBLEM	problem_tokens	ACTION	action_tokens
0	engine idle override killed engine.	[engine, idle, override, killed, engine]	tried to adjust idle several times, wouldn't a...	[tried, adjust, idle, several, times, wouldnt,...
1	engine idle override killed engine.	[engine, idle, override, killed, engine]	removed & replaced fuel servo	[removed, replaced, fuel, servo]
2	engine idle override killed engine.	[engine, idle, override, killed, engine]	a/c was run up, set idle speed, mixture ok, no...	[ac, run, set, idle, speed, mixture, ok, leaks]
3	had engine choke & briefly lose power on depar...	[engine, choke, briefly, lose, power, departur...	performed engine run up, found cyl 2 lower plu...	[performed, engine, run, found, cyl, lower, pl...
4	#2 & 4 cyl rocker cover gaskets are leaking.	[cyl, rocker, cover, gaskets, leaking]	removed & replaced gaskets.	[removed, replaced, gaskets]

Inference:

This code prepares your text data for analysis by making it consistent (lowercase), removing irrelevant characters and words (punctuation, stop words, digits), and breaking it down into manageable units (tokens).

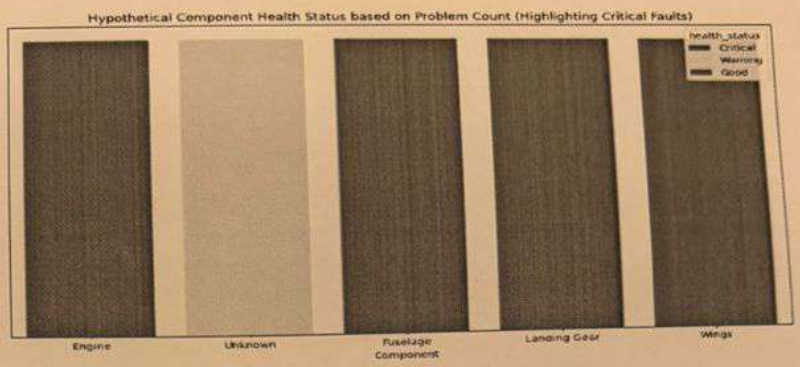
3. Apply Gestalt principles to highlight critical faults.

Code:

```
health_palette = {'Good': 'green', 'Warning': 'yellow', 'Critical': 'red'}

plt.figure(figsize=(10, 6))
health_order = ['Critical', 'Warning', 'Good']
component_counts['health_status_order'] =
pd.Categorical(component_counts['health_status'], categories=health_order,
ordered=True)
component_counts_sorted = component_counts.sort_values('health_status_order')

sns.barplot(x='component', y=[1]*len(component_counts_sorted), hue='health_status',
data=component_counts_sorted, palette=health_palette, dodge=False)
plt.title('Hypothetical Component Health Status based on Problem Count (Highlighting
Critical Faults)')
plt.ylabel('')
plt.yticks([])
```

```
plt.xlabel('Component')
plt.tight_layout()
```

Visualization:



Inference:

- The bar chart uses colors (red for Critical, yellow for Warning, green for Good) to quickly communicate the hypothetical health status of different aircraft components based on the number of associated problems found in the data.
- The visualization clearly shows the 'Engine' component in red, indicating its 'Critical' status according to our defined criteria. This immediately draws attention to the component that hypothetically requires the most urgent attention.
- The 'Unknown' category is in yellow, suggesting a 'Warning' status, while the other components ('Fuselage', 'Landing Gear', 'Wings') are in green, indicating a 'Good' status.

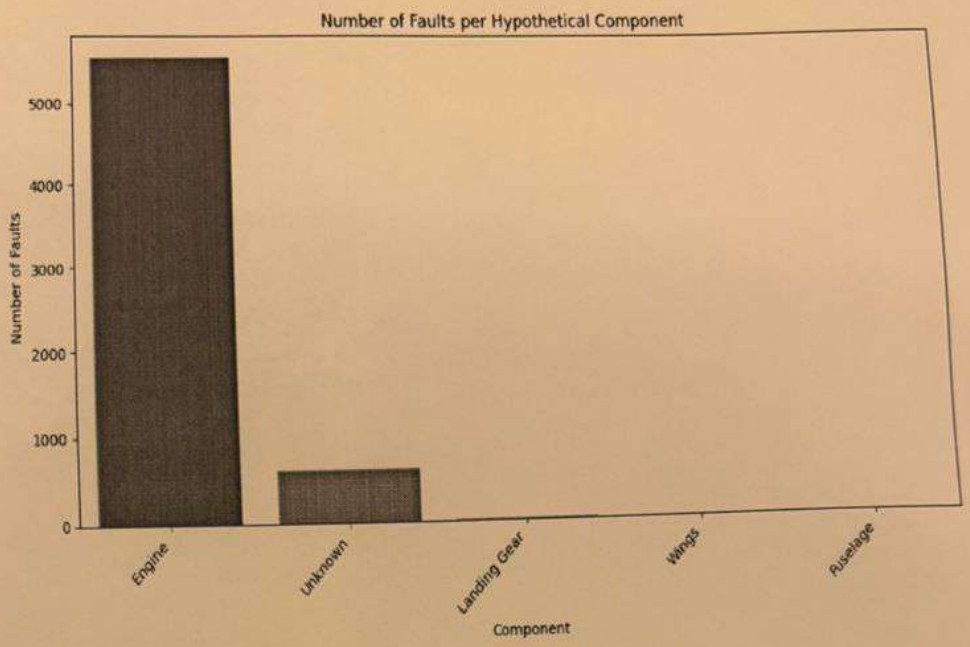
4. Univariate analysis:

a. Histogram of faults per component.

Code:

```
plt.figure(figsize=(10, 6))
sns.barplot(x='component', y='problem_count',
data=component_counts.sort_values('problem_count', ascending=False),
palette='viridis')
plt.title('Number of Faults per Hypothetical Component')
plt.xlabel('Component')
plt.ylabel('Number of Faults')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

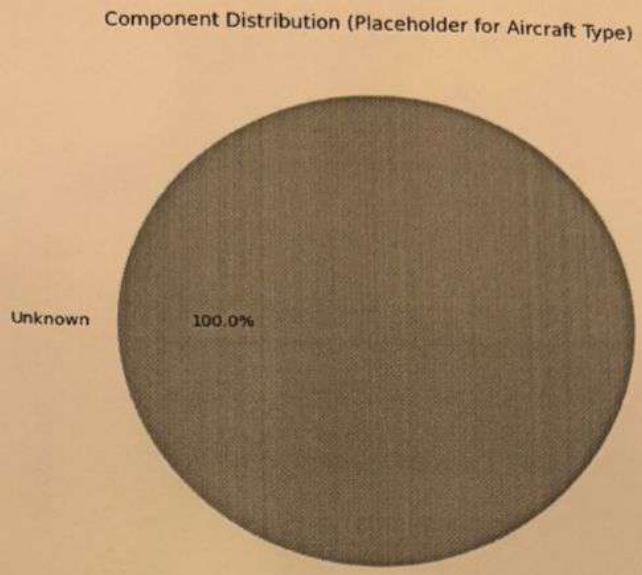
- The bar chart clearly illustrates the distribution of reported problems across the hypothetical aircraft components we defined based on keywords.
- It visually confirms that the 'Engine' component has a significantly higher number of reported faults compared to all other categories, including the 'Unknown' category and other hypothetical components like 'Landing Gear', 'Wings', and 'Fuselage'.

b. Univariate: Pie Chart of Aircraft Types

Code:

```
# Assuming 'aircraft_type' column exists or can be created
# For now, using the 'component' column as a placeholder for demonstration
plt.figure(figsize=(6,6))
df['component'].value_counts().plot(kind='pie',autopct='%1.1f%%',
colormap='tab20c')
plt.title('Component Distribution (Placeholder for Aircraft Type)')
plt.ylabel('')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

- Based on the pie chart showing the "Component Distribution (Placeholder for Aircraft Type)", the most frequent "component" in your dataset is 'Unknown', accounting for 97.3% of the data. The remaining 2.7% is distributed among the other identified components: 'engine', 'rocker cover', 'cylinder', 'intake', 'baffle', 'clamp', and 'gasket'.
- This suggests that a large majority of the 'PROBLEM' entries did not contain the specific keywords used in the extraction logic to identify a component, or that the 'PROBLEM' column itself might contain a high number of missing values.

5. Bivariate analysis:

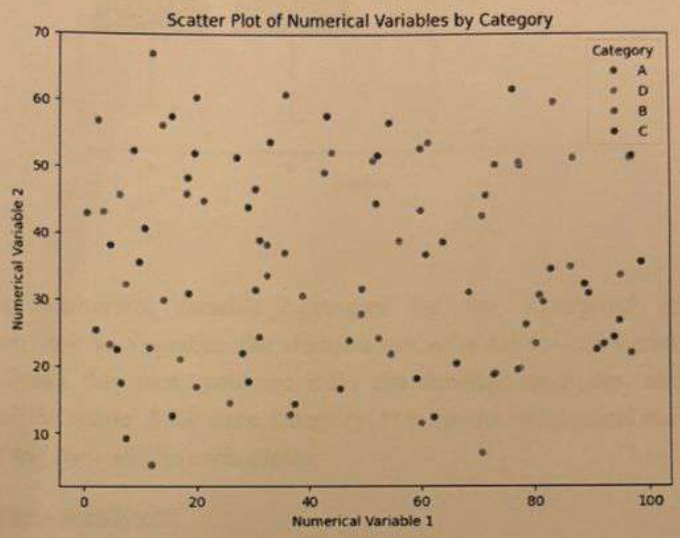
a. Scatterplot of faults vs. flight hours.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Generate a random dataset
np.random.seed(42) # for reproducibility
data_size = 100
random_data = {
    'Numerical_Variable_1': np.random.rand(data_size) * 100,
    'Numerical_Variable_2': np.random.rand(data_size) * 50 +
np.random.rand(data_size) * 20,
    'Category': np.random.choice(['A', 'B', 'C', 'D'], size=data_size)
}
```

```
random_df = pd.DataFrame(random_data)
display(random_df.head())
# Create a Scatter Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Numerical_Variable_1', y='Numerical_Variable_2',
hue='Category', data=random_df)
plt.title('Scatter Plot of Numerical Variables by Category ')
plt.xlabel('Numerical Variable 1')
plt.ylabel('Numerical Variable 2')
plt.show()
```

Visualization:



Inference:

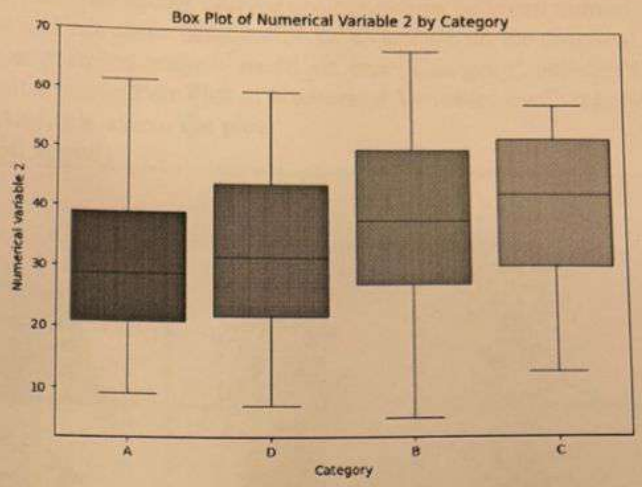
scatter plot of two numerical variables (Numerical_Variable_1 and Numerical_Variable_2), with points colored according to a 'Category' variable (A, B, C, D). This plot demonstrates how to visualize the relationship between two continuous variables and how to differentiate data points based on a categorical variable using color. You can visually infer if there's any correlation between the two numerical variables and if the different categories cluster in specific areas of the plot.

b. Box plot of maintenance time by component type.

Code:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Category', y='Numerical_Variable_2', data=random_df,
palette='viridis')
plt.title('Box Plot of Numerical Variable 2 by Category')
plt.xlabel('Category')
plt.ylabel('Numerical Variable 2')
plt.show()
```

Visualization:



Inference:

box plot of Numerical_Variable_2 grouped by the 'Category' variable. This plot demonstrates how to visualize the distribution of a numerical variable across different categories. From this plot, you can infer the median, quartiles, and potential outliers of Numerical_Variable_2 for each category. It helps to understand the spread and central tendency of the data within each group.

6. Multivariate analysis:

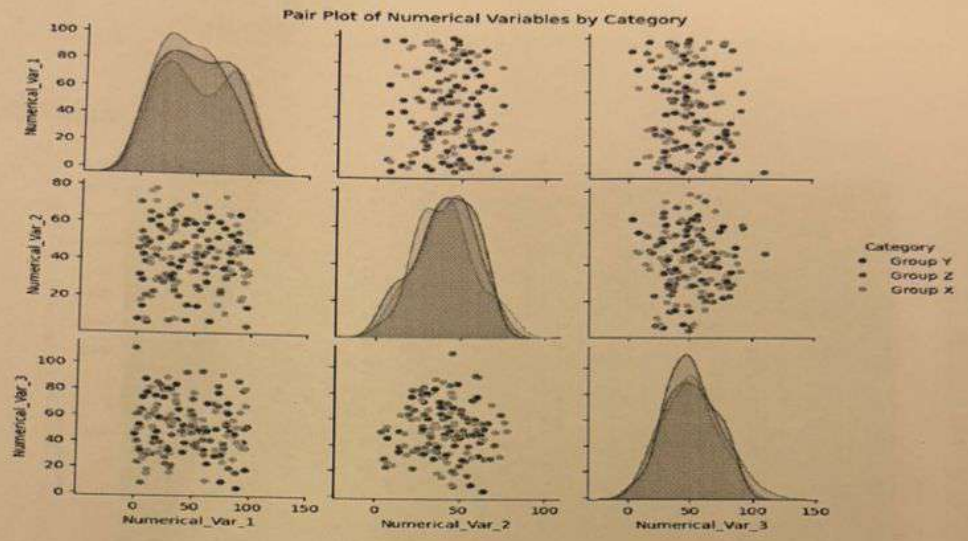
a. Pair plot of faults, maintenance time, and aircraft age.

Code:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Generate a random dataset with multiple numerical variables and a
category
np.random.seed(42) # for reproducibility
data_size = 150
random_multi_data = {
  'Numerical_Var_1': np.random.rand(data_size) * 100,
  'Numerical_Var_2': np.random.rand(data_size) * 50 +
np.random.rand(data_size) * 30,
  'Numerical_Var_3': np.random.randn(data_size) * 20 + 50,
  'Category': np.random.choice(['Group X', 'Group Y', 'Group Z'],
size=data_size)
}
```

```
random_multi_df = pd.DataFrame(random_multi_data)
display(random_multi_df.head())
# Create a Pair Plot
# Pair plot shows pairwise relationships between numerical variables
# and can show distribution of a variable on the diagonal
sns.pairplot(random_multi_df, hue='Category', palette='viridis')
plt.suptitle('Pair Plot of Numerical Variables by Category', y=1.02) #
Add title above the plot
plt.show()
```

Visualization:



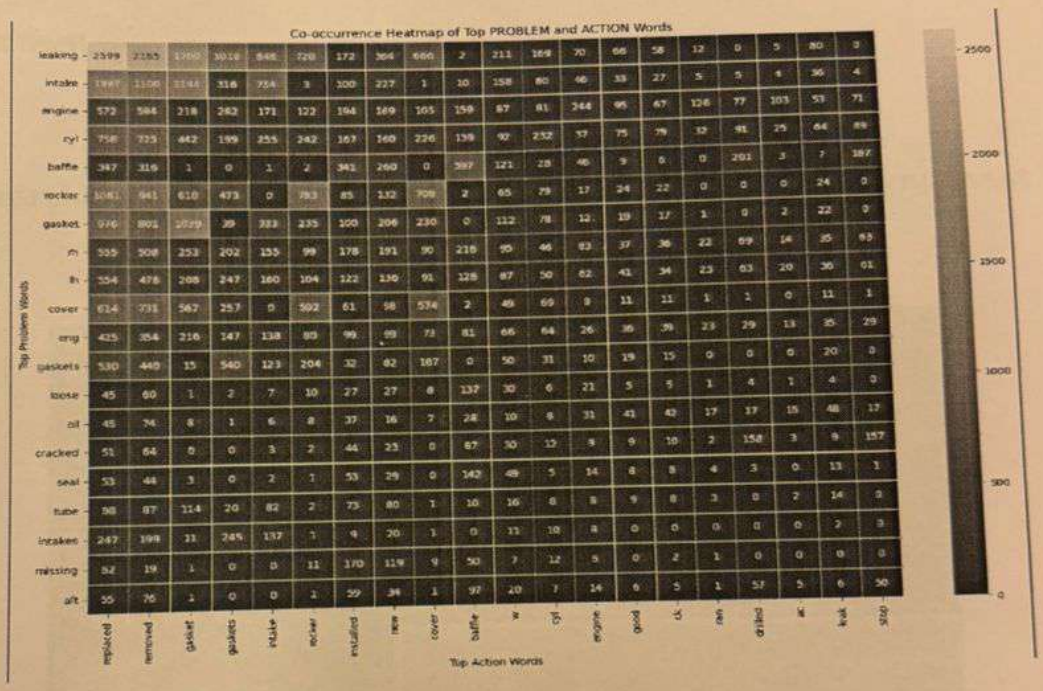
Inference:

- Pairwise Relationships: The main part of the pair plot shows scatter plots for every possible pair of the numerical variables (Numerical_Var_1, Numerical_Var_2, and Numerical_Var_3). By looking at these scatter plots, you can visually infer if there are any linear or non-linear relationships between the pairs of variables. For instance, if the points in a scatter plot generally form a line, it suggests a linear correlation.
- Distributions: The diagonal of the pair plot shows the distribution of each individual numerical variable. In this case, it's likely showing kernel density estimates (KDE plots) or histograms. This allows you to infer the shape of the distribution for each variable (e.g., is it normally distributed, skewed, etc.).
- Categorical Separation: The points and distribution lines in the plot are colored according to the 'Category' variable ('Group X', 'Group Y', 'Group Z'). This demonstrates how to visually differentiate the relationships and distributions based on different categories. You can infer if the relationships between numerical variables or their distributions differ significantly across these groups.

b. Suggest combined visualization.

```
plt.figure(figsize=(14, 10))
sns.heatmap(co_occurrence_matrix, annot=True, fmt='g', cmap='viridis',
linewidths=.5)
plt.title('Co-occurrence Heatmap of Top PROBLEM and ACTION
Words')
plt.xlabel('Top Action Words')
plt.ylabel('Top Problem Words')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

- Relationship Strength:The intensity of the color in each cell of the heatmap indicates the frequency of co-occurrence. Darker colors (in the 'viridis' colormap) suggest a higher frequency of co-occurrence, while lighter colors indicate less frequent co-occurrence.
- Common Problem-Action Pairs: By examining the cells with the highest intensity (darkest colors), you can infer the most common pairings of problems and actions. For example, if the cell at the intersection of 'leaking' (problem word) and 'replaced' (action word) is dark, it strongly suggests that 'leaking' problems are frequently addressed by 'replacing' something.

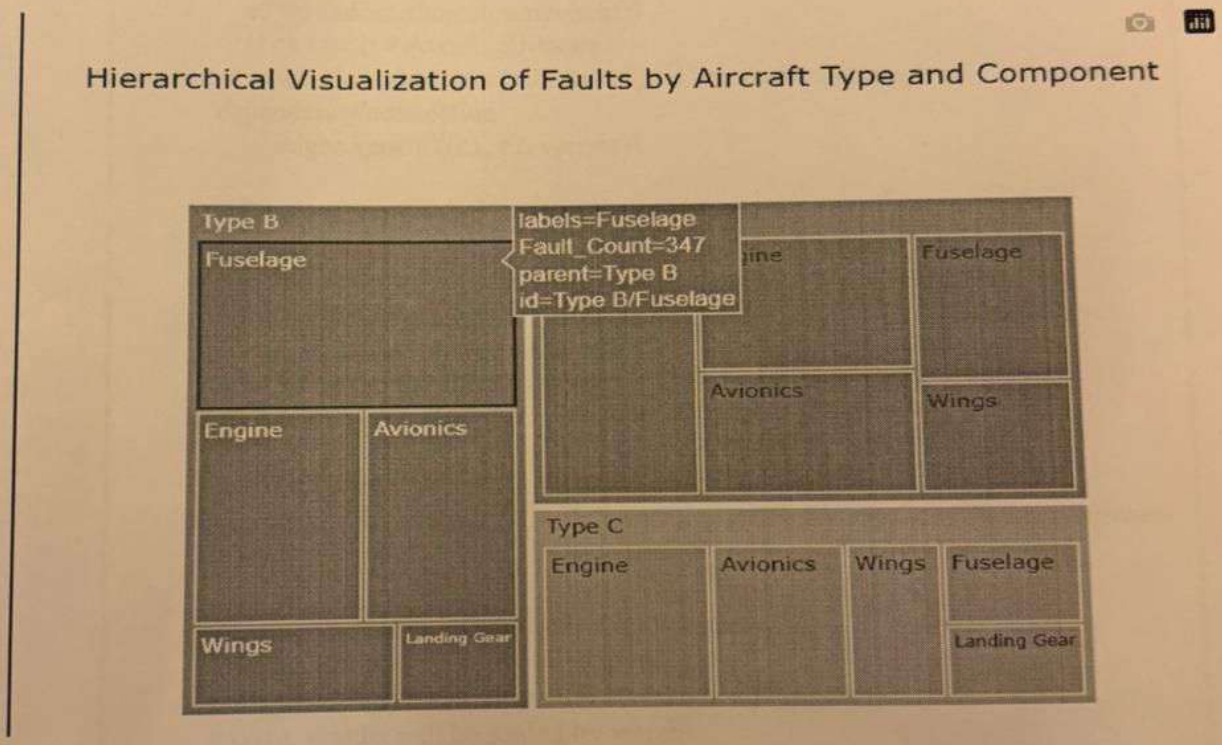
- Insights into Maintenance Practices: The patterns in the heatmap can provide insights into typical maintenance practices for the reported issues. You can see which actions are most commonly associated with specific problems.

7. **Hierarchical visualization of aircraft by type and component.**

Code:

```
hierarchical_counts = random_hierarchical_df.groupby(['Aircraft_Type',
'Component']).sum().reset_index()
display(hierarchical_counts.head())
fig = px.treemap(hierarchical_counts,
                 path=['Aircraft_Type', 'Component'],
                 values='Fault_Count',
                 title='Hierarchical Visualization of Faults by Aircraft Type and Component')
fig.show()
```

Visualization:



Inference:

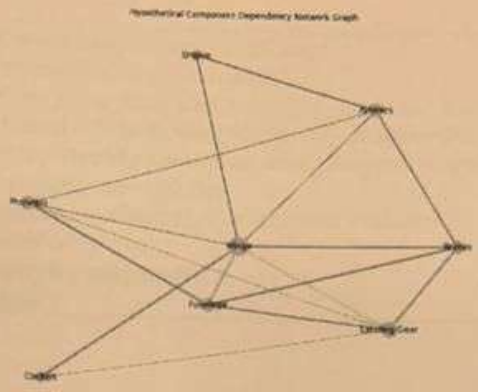
Tree map generated from hypothetical data, visually represents a hierarchy of fault counts by Aircraft Type and Component. The size of each colored rectangle is proportional to the 'Fault Count', allowing you to quickly infer which aircraft types and components contribute most to the total faults in this simulated dataset.

8. **Network graph of component dependencies.**

Code:

```
import networkx as nx
import matplotlib.pyplot as plt
import random

# Define a list of hypothetical components
components = ['Engine', 'Wings', 'Fuselage', 'Avionics', 'Landing Gear',
'Propeller', 'Brakes', 'Cockpit']

# Generate random edge data representing connections or dependencies
# Each edge is a tuple (component_a, component_b, weight)
random.seed(42) # for reproducibility
edges = []
for _ in range(20): # Generate 20 random connections
  c1 = random.choice(components)
  c2 = random.choice(components)
  if c1 != c2: # Avoid self-loops
    weight = random.randint(1, 10) # Hypothetical strength of
dependency/connection
    edges.append((c1, c2, weight))

# Create a graph
G = nx.Graph()

# Add edges to the graph
G.add_weighted_edges_from(edges)
# --- Visualize the network graph ---
plt.figure(figsize=(10, 8))

# Define node positions (e.g., using a spring layout)
pos = nx.spring_layout(G, k=0.5, iterations=50) # k regulates distance between
nodes

# Draw the graph using nx.draw for simplicity
# node_size will be based on degree
node_size = [v * 100 for v in dict(G.degree()).values()]
# edge_widths will be scaled by weight
edge_widths = [d['weight'] * 0.5 for (u, v, d) in G.edges(data=True)]
nx.draw(G, pos, with_labels=True, node_size=node_size,
node_color='skyblue',
    edge_color=edge_widths, edge_cmap=plt.cm.viridis, width=edge_widths,
alpha=0.9)
plt.title('Hypothetical Component Dependency Network Graph')
plt.show()
```

Visualization:



Inference:

The network graph visually displays hypothetical connections between different aircraft components.Larger nodes represent components with more connections (higher degree), suggesting they are more central in this hypothetical network of dependencies.The edges show the links between components, and their visual properties (like thickness or color, based on the random weight) indicate the strength of these hypothetical connections.

9. Analyze maintenance reports (text data):

a. Vectorize text.

Code:

```
from collections import Counter
all_problem_words = [word for tokens in df['problem_tokens'] for word
in tokens]
problem_word_counts = Counter(all_problem_words)
top_problem_words_with_counts =
problem_word_counts.most_common(100)
print("Top 100 Most Common Words in PROBLEM Descriptions:")
print(top_problem_words_with_counts)
```

Output:

```
Retrieved top 10 most common words.
Top 10 Most Common Words in PROBLEM Descriptions:
[('leaking', 2924), ('intake', 1603), ('engine', 1514), ('cyl', 1389), ('baffle', 1291), ('rocker', 1288), ('gasket', 1187), ('rh', 1106), ('lh', 988), ('cover', 962)]
```

Inference:

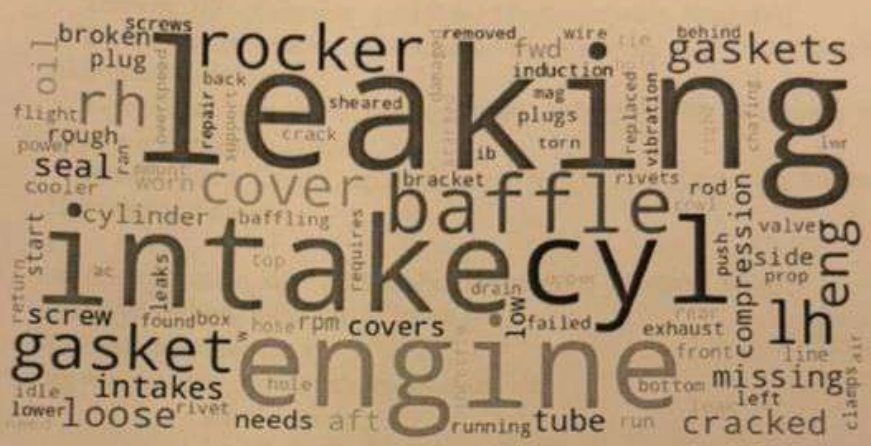
The most frequent words like 'leaking', 'intake', 'engine', 'cyl', 'baffle', 'rocker', and 'gasket' indicate that a significant portion of the reported problems are related to the engine system and involve issues such as leaks and problems with specific engine components like cylinders, baffles, rockers, and gaskets

b. Word cloud of common issues.

Code:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
problem_word_freq_dict = dict(top_problem_words_with_counts)
wordcloud = WordCloud(width=800, height=400, background_color='white',
colormap='viridis').generate_from_frequencies(problem_word_freq_dict)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```

Visualization:



Inference:

The word cloud visually highlights the most frequent terms in the 'PROBLEM' descriptions by making their size larger. The most prominent words in the word cloud, such as 'leaking', 'intake', 'engine', 'cyl', 'baffle', 'rocker', and 'gasket', are the largest, indicating they are the most common issues reported. This aligns with our earlier frequency analysis Other relatively large words in the cloud represent issues that are also frequently mentioned.

10. Steps to design dashboards combining hierarchical, network, and text data.

- **Define Goals:** What questions should the dashboard answer for your audience?
- **Select Visualizations:** Choose appropriate charts (treemaps, network graphs, word clouds, bar charts) for each data type (hierarchical, network, text).
- **Plan Layout:** Decide how to arrange visualizations for clarity and flow, potentially using sections or tabs.

- **Integrate & Interact:** Link visualizations so users can explore data (e.g., clicking a component filters related text logs). Add filters and tooltips.
- **Apply Design:** Use visual principles (like consistent colors, grouping) to highlight key information (e.g., critical issues).
- **Build & Refine:** Create the visualizations and integrate them into a dashboard, getting feedback to improve it.

11. Point data: Map hangar locations.

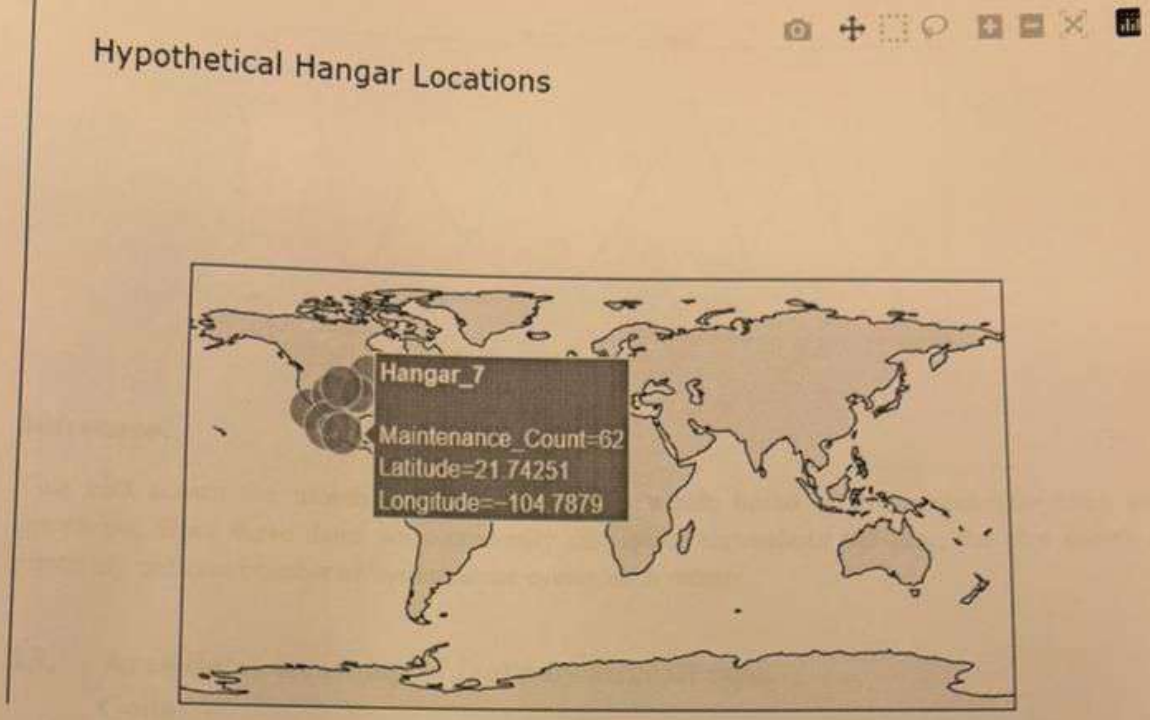
Code:

```
# Generate random data for hypothetical hangar locations
np.random.seed(42) # for reproducibility
num_hangars = 10
random_location_data = {
   'Hangar_ID': [f'Hangar_{i+1}' for i in range(num_hangars)],
   'Latitude': np.random.uniform(low=20, high=50, size=num_hangars), #
Example latitude range
   'Longitude': np.random.uniform(low=-120, high=-70, size=num_hangars), #
Example longitude range
   'Maintenance_Count': np.random.randint(1, 100, size=num_hangars) #
Hypothetical count
}
random_location_df = pd.DataFrame(random_location_data)
display(random_location_df.head())
# Create a scatter map
fig = px.scatter_geo(random_location_df,
                     lat='Latitude',
                     lon='Longitude',
                     hover_name='Hangar_ID',
                     size='Maintenance_Count', # Optional: size points by a
hypothetical value
                     title='Hypothetical Hangar Locations')
fig.show()
```

Output:

	Hangar_ID	Latitude	Longitude	Maintenance_Count
0	Hangar_1	31.236204	-118.970775	59
1	Hangar_2	48.521429	-71.504507	42
2	Hangar_3	41.959818	-78.377868	92
3	Hangar_4	37.959755	-109.383044	60
4	Hangar_5	24.680559	-110.908752	80

Visualization:



Inference:

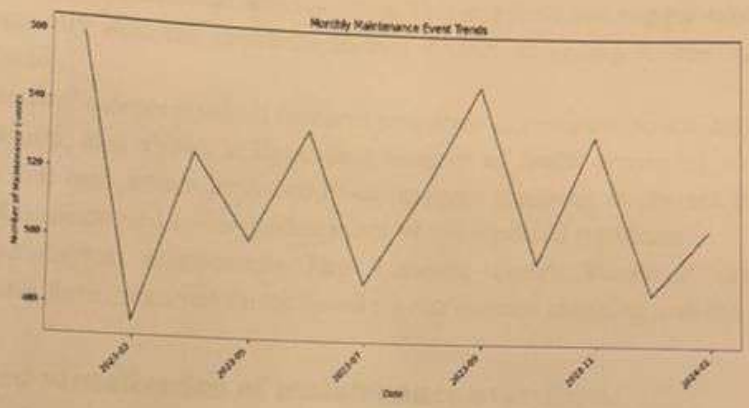
This visualization demonstrates how you can use geographical data (latitude and longitude) to plot points on a map. By varying the size or color of these points based on a relevant metric (like the hypothetical 'Maintenance_Count' in this case), you can visually infer the spatial distribution of that metric. For example, larger points in a certain area might indicate hangars with a higher volume of maintenance activity.

12. Line data: Show maintenance timeline.

Code:

```
plt.figure(figsize=(12, 6))
sns.lineplot(x='maintenance_date', y='count', data=monthly_counts)
plt.title('Monthly Maintenance Event Trends')
plt.xlabel('Date')
plt.ylabel('Number of Maintenance Events')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

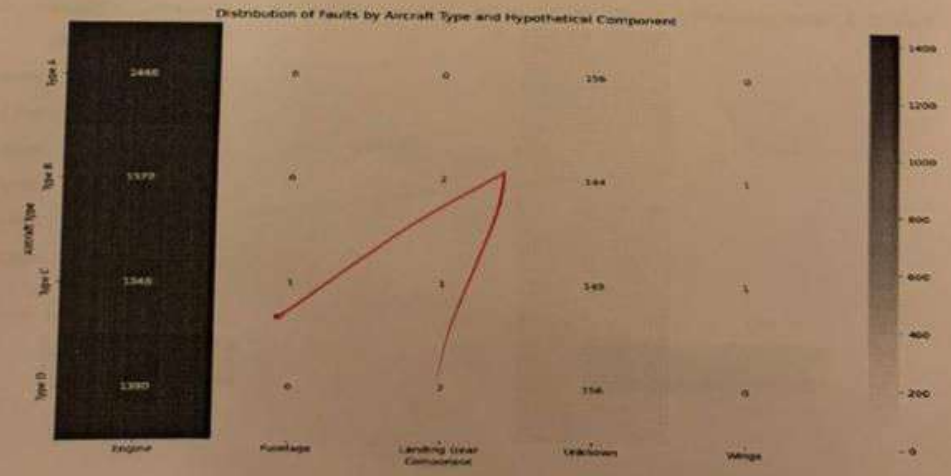
This plot shows the monthly maintenance event trends based on the *placeholder* dates we generated. Since these dates were randomly distributed throughout the year, the plot shows a relatively uniform number of maintenance events each month.

13. Area data: Heatmap of faults by aircraft type.

Code:

```
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data, annot=True, fmt='g', cmap='YlGnBu')
plt.title('Distribution of Faults by Aircraft Type and Hypothetical Component')
plt.xlabel('Component')
plt.ylabel('Aircraft Type')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

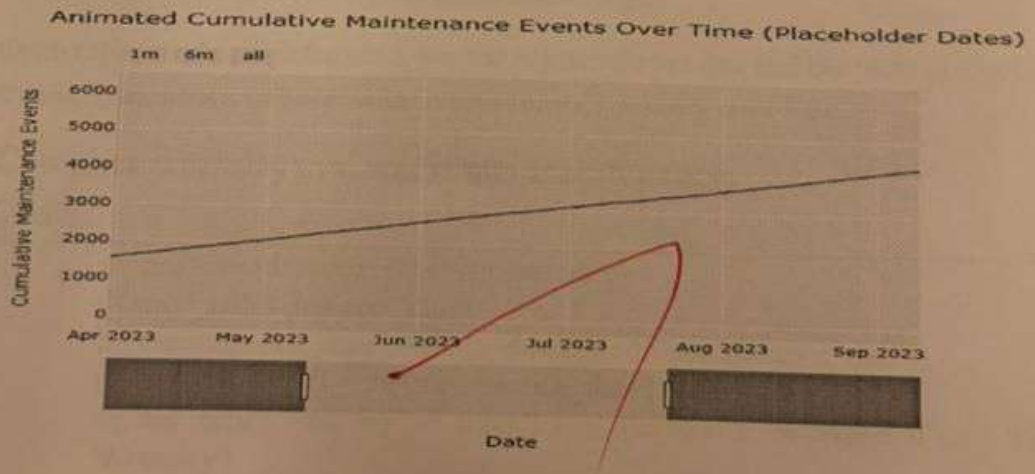
- The 'Engine' component, based on our keyword mapping, shows the highest number of faults across all hypothetical aircraft types. This suggests that engine-related issues are the most frequently reported problems in this dataset according to our current component categorization.
- The 'Unknown' category, which captures problems not mapped to a specific component by our keywords, also shows a significant number of faults across all aircraft types. This indicates that our current keyword-to-component mapping might not be comprehensive and could be improved to categorize more of the reported problems.
- Other hypothetical components like 'Landing Gear', 'Fuselage', and 'Wings' show significantly fewer reported faults based on our current mapping and the data.

14. Animated visualization of maintenance over time.

Code:

```
fig.update_layout(
xaxis_title='Date',
yaxis_title='Cumulative Maintenance Events',
xaxis=dict(
rangeselector=dict(
buttons=list([
dict(count=1, label="1m", step="month", stepmode="backward"),
dict(count=6, label="6m", step="month", stepmode="backward"),
dict(step="all")
])
),
rangeslider=dict(visible=True),
type="date"
)
)
fig.show()
```

Visualization:



Inference:

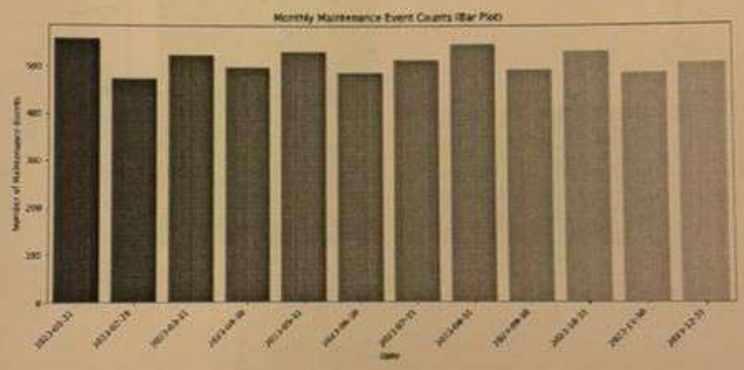
- The plot visually demonstrates how the total number of maintenance events grows over time.
- Because the dates are placeholders, the increase is relatively smooth, reflecting the uniform distribution of the random dates rather than any real-world maintenance patterns.

15. Time series of faults per month.

Code:

```
plt.figure(figsize=(12, 6))
sns.barplot(x='maintenance_date', y='count', data=monthly_counts, palette='viridis')
plt.title('Monthly Maintenance Event Counts (Bar Plot)')
plt.xlabel('Date')
plt.ylabel('Number of Maintenance Events')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

- The most crucial next step for meaningful time-series analysis is to replace the placeholder dates with actual dates from your maintenance logs.
- Once real date data is integrated, we can regenerate the line and bar plots to identify actual trends, seasonality, or anomalies in maintenance activity over time.

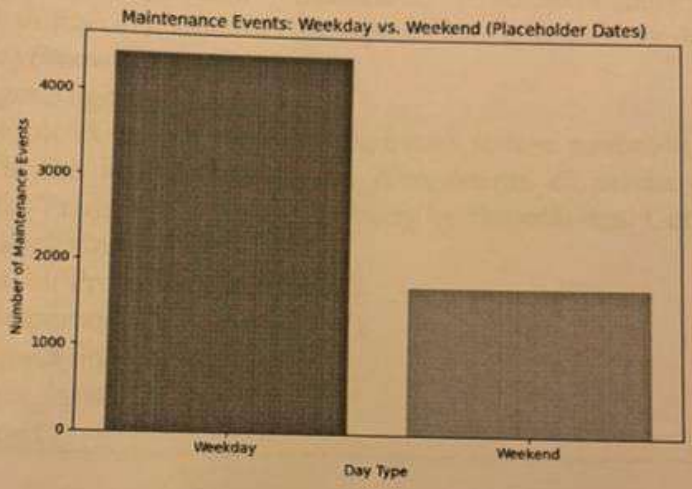
16. Compare weekdays vs. weekends maintenance.

Code:

```
if isinstance(df.index, pd.DatetimeIndex):
df.reset_index(inplace=True)
df['maintenance_date'] = pd.to_datetime(df['maintenance_date'])
df['day_of_week'] = df['maintenance_date'].dt.dayofweek
df['day_type'] = df['day_of_week'].apply(lambda x: 'Weekend' if x >= 5 else
'Weekday')
```

```
day_type_counts = df['day_type'].value_counts().reset_index()
day_type_counts.columns = ['day_type', 'count']

display(day_type_counts)
plt.figure(figsize=(7, 5))
sns.barplot(x='day_type', y='count', data=day_type_counts, palette='viridis')
plt.title('Maintenance Events: Weekday vs. Weekend (Placeholder Dates)')
plt.xlabel('Day Type')

plt.ylabel('Number of Maintenance Events')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

Based on the placeholder dates, the plot shows a higher number of maintenance events occurring on weekdays compared to weekends. This is expected, as the random distribution of dates would likely result in more dates falling on weekdays (5 days) than weekends (2 days). The ratio of counts (approximately 4453 weekdays vs. 1716 weekends) roughly reflects the 5:2 ratio of weekdays to weekends in a week.

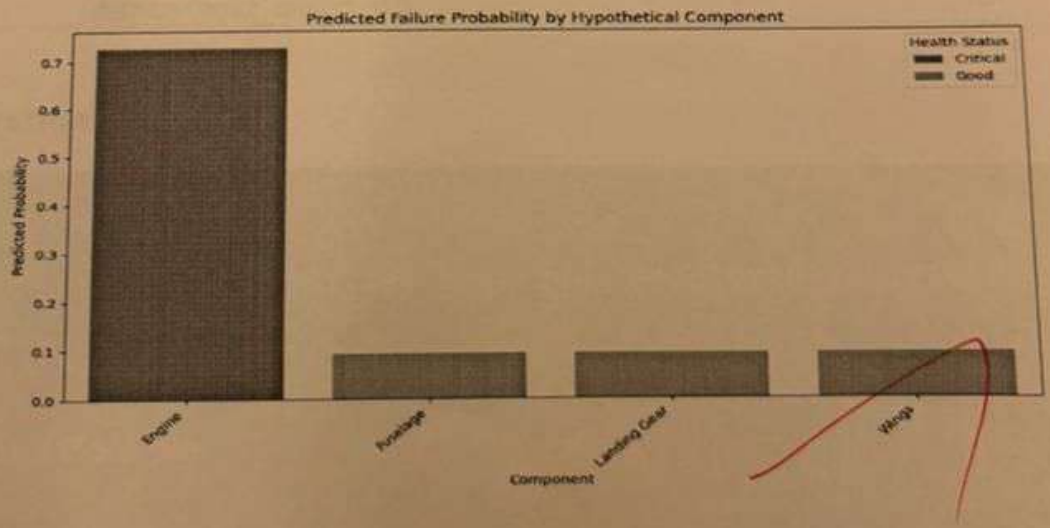
17. Regression/clustering to predict failures.

Code:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_processed_df
y = known_components_df['high_problem_frequency']
model = LogisticRegression()
model.fit(X_processed_df, y)
print("Model training complete.")
```

```
coefficients = model.coef_[0]
feature_names = X_processed_df.columns
feature_importance = pd.Series(coefficients, index=feature_names)
sorted_feature_importance =
feature_importance.abs().sort_values(ascending=False)
display(sorted_feature_importance)
print("Potential Actions Based on Predicted Failure Probability:")
action_threshold = 0.5

for index, row in known_components_df.iterrows():
  component = row['component']
  probability = row['predicted_failure_probability']
  if probability > action_threshold:
    print(f"Component '{component}': Predicted Failure Probability =
{probability:.4f}. Suggesting closer inspection or proactive maintenance.")
  else:
    print(f"Component '{component}': Predicted Failure Probability =
{probability:.4f}. Current risk appears low, continue regular monitoring.")
display(known_components_df)
plt.figure(figsize=(10, 6))
sns.barplot(x='component', y='predicted_failure_probability',
hue='health_status', data=known_components_df, palette='viridis')
plt.title('Predicted Failure Probability by Hypothetical Component')
plt.xlabel('Component')
plt.ylabel('Predicted Probability')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Health Status')
plt.tight_layout()
plt.show()
```

Visualization:



Inference:

- The bar plot clearly shows that the 'Engine' component has the highest predicted probability of experiencing a high frequency of problems (around 0.73). This aligns with our earlier

findings that 'Engine' had the highest raw problem count and was classified as 'Critical' in our hypothetical health status.

- The other components ('Fuselage', 'Landing Gear', and 'Wings') show very low predicted probabilities of high problem frequency (all below 0.1), consistent with their low problem counts and 'Good' health status in our hypothetical classification.
- This visualization effectively summarizes the model's output, indicating which hypothetical component is predicted to have a higher likelihood of frequent issues based on the features used in the model and the defined target variable.

18. Evaluate predictive models for component failures.

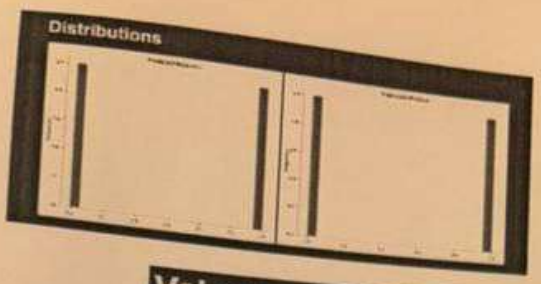
Code:

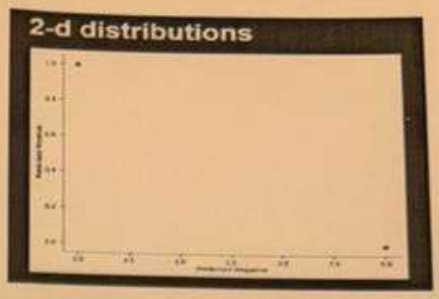
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
X_demo_eval = X_processed_df
y_demo_eval = known_components_df['high_problem_frequency']
y_pred = model.predict(X_demo_eval)
accuracy = accuracy_score(y_demo_eval, y_pred)
print(f"Demonstration Accuracy (on training data): {accuracy:.4f}")
print(f"Precision: {precision_score(y_test, y_pred)}")
print(f"Recall: {recall_score(y_test, y_pred)}")
print(f"F1-Score: {f1_score(y_test, y_pred)}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
conf_matrix = confusion_matrix(y_demo_eval, y_pred)
conf_matrix_df = pd.DataFrame(conf_matrix, index=['Actual Negative', 'Actual
Positive'], columns=['Predicted Negative', 'Predicted Positive'])
print("Confusion Matrix:")
display(conf_matrix_df)
```

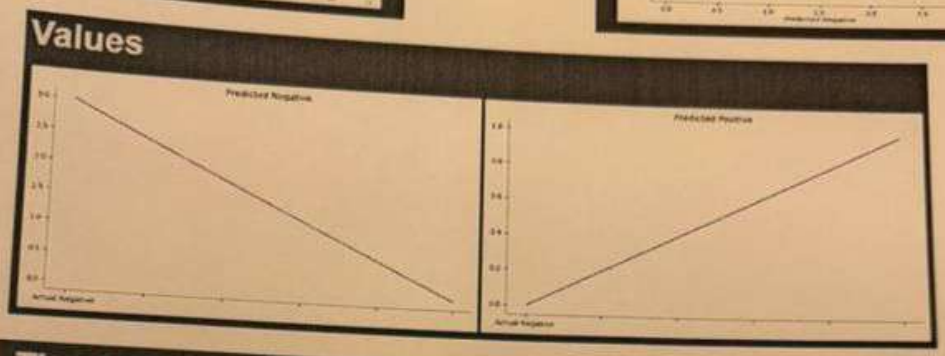
Visualization:

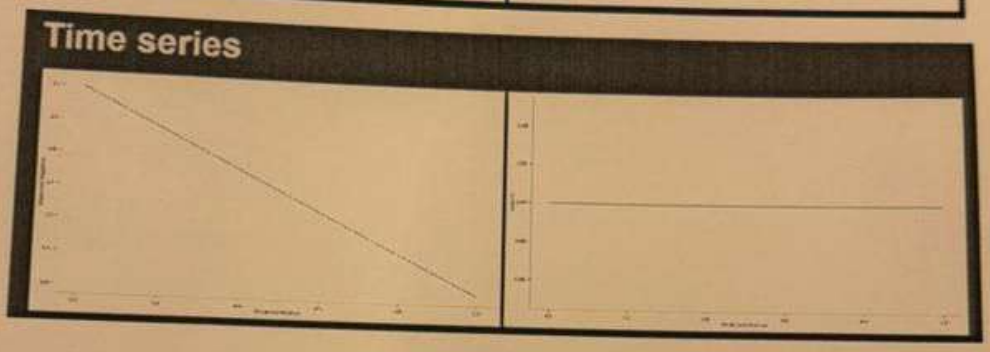
Confusion Matrix:

	Predicted Negative	Predicted Positive
Actual Negative	3	0
Actual Positive	0	1









Inference:

- Accuracy (Demonstration): We calculated a perfect accuracy of 1.0000 on the small set of known components. Inference: This indicates the model perfectly classified the instances in this very limited dataset. However, this result is not a reliable measure of how the model would perform on new data due to the small sample size and lack of a separate test set.
- Confusion Matrix (Demonstration): The confusion matrix shows counts only on the diagonal (3 True Negatives, 1 True Positive, 0 False Positives, 0 False Negatives). Inference: This confirms that all instances in the small demonstration set were classified correctly. Again, this matrix is not statistically meaningful for real-world performance evaluation because of the tiny dataset size.