```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import logging
from typing import List
import openpyxl
from matplotlib.ticker import MaxNLocator

# Configure logging and styling
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Initialize Seaborn theme and palette
sns.set_theme(style="whitegrid", palette="husl")

class ExcelDataAnalyzer:

    def __init__(self, file_path: str):
        self.file_path = file_path
        self.df = self._load_excel()
        self.visualization_paths = []

    def _load_excel(self) -> pd.DataFrame:
        try:
            return pd.read_excel(self.file_path, engine='openpyxl')
        except Exception as e:
            logger.error(f"Failed to load {self.file_path}: {str(e)}")
            raise

    def _save_visualization(self, fig, name: str):
        path = f"{self.__class__.__name__}_{name}.png"
        fig.savefig(path, bbox_inches='tight', dpi=300)
        plt.close(fig)
        self.visualization_paths.append(path)
        logger.info(f"Saved visualization: {path}")

    def get_visualizations(self) -> List[str]:
        return self.visualization_paths

class Task1Analyzer(ExcelDataAnalyzer):

    def analyze(self):
        self._clean_data()
        self._add_taxonomy_tags()
        self._generate_visualizations()

    def _clean_data(self):
        # Date handling
        self.df['Order Date'] = pd.to_datetime(self.df['Order Date'],
errors='coerce')
```

```
        # Text cleaning
        text_cols = ['Complaint', 'Cause', 'Correction']
        self.df[text_cols] = self.df[text_cols].apply(lambda x: x.str.strip())

    def _add_taxonomy_tags(self):
        # Define the detailed taxonomy mapping
        taxonomy_mapping = {
            'Not Tightened': ('Loose', 'Cab P Clip', 'Retightened', 'Cab P Clip'),
            'Not Installed': ("Won't stay open", 'Fuel Door', 'Installed', 'Gas Strut'),
            'Not Mentioned': ('Crushed', 'Compressor Pressure Line', 'Replaced', 'Braided Steel'),
            'Loosened': ('Oil Running', 'Not Mentioned', 'Topped Off', 'O-Ring'),
            'Not Included': ('Missing', 'Vector', 'Not Mentioned', 'Vector'),
            'Out of Fitting': ('Oil Dripping', 'Coupler', 'Cleaned Out', 'Coupler'),
            'Blown': ('Oil Leak', 'Mount SVM Sign', 'Reseted', 'Brackets'),
            'Poor Material': ('Broke', 'Harness', 'Repaired', 'Hydraulic'),
            'Leaking': ('Leak', 'Rinse Tank', 'Tightened', 'Not Mentioned'),
            'Failed Sending': ('Open', 'Fuel Sender', '', 'NCV Harness'),
            'No Oring': ('Hydraulic Leak', 'Boom', '', 'Tube'),
            'Not Tighten': ('Fold Uneven', 'Auto Boom', '', 'Oring'),
            'Out of Range': ('Getting Fault Code', 'Condenser', '', 'Sensor'),
            'Lubricant Drip Drown': ('Not Working', 'Left-Air Duct', '', 'Counter'),
            'Fault': ('Error Codes', 'Bulkhead Connector', '', 'Threads'),
            'Internal Issue': ('Product Leak', 'Braided Steel', '', 'Left Air Duct'),
            'Screwed in a Thread': ('Does not Light', 'Intrip Unlocks', '', 'Compressor Line'),
            'Faulty': ('', 'Sensor', '', 'Intrip Unlocks'),
        }

        # First basic Root Cause
        self.df['Root Cause'] = self.df['Cause'].apply(
            lambda x: next((root for root in taxonomy_mapping if root.lower() in str(x).lower()), 'Other')
        )
        # Fill Symptom Condition and Components
        self.df['Symptom Condition 1'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', 'Other', '', ''))
[0])
        self.df['Symptom Condition 2'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', 'Other', ''))
[1])
        self.df['Symptom Condition 3'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', 'Other'))
[2])

        self.df['Symptom Component 1'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', 'Other', '', ''))
[1])
        self.df['Symptom Component 2'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', 'Other', ''))
[2])
        self.df['Symptom Component 3'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', 'Other'))
[3])

        # Fix Conditions and Components
        self.df['Fix Condition 1'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', 'Other'))
[0])
        self.df['Fix Condition 2'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', ''))[1])
        self.df['Fix Condition 3'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', ''))[2])

        self.df['Fix Component 1'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', 'Other'))
[3])
        self.df['Fix Component 2'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', ''))[3])
        self.df['Fix Component 3'] = self.df['Root
Cause'].apply(lambda x: taxonomy_mapping.get(x, ('', '', '', ''))[3])

    def _generate_visualizations(self):
        # Visualization 1: Root Cause Distribution
        fig1, ax1 = plt.subplots(figsize=(10, 6))
        self.df['Root Cause'].value_counts().plot(kind='bar', ax=ax1)
        ax1.set_title('Root Cause Distribution')
        ax1.set_ylabel('Count')
        self._save_visualization(fig1, 'root_cause_distribution')

        # Visualization 2: Symptom Frequency
        fig2, ax2 = plt.subplots(figsize=(10, 6))
        self.df['Symptom Component 1'].value_counts().plot(kind='pie',
autopct='%1.1f%%', ax=ax2)
        ax2.set_title('Symptom Component Distribution')
        ax2.set_ylabel('')
        self._save_visualization(fig2,
'symptom_component_distribution')

        # Visualization 3: Fix Condition Distribution
        fig3, ax3 = plt.subplots(figsize=(10, 6))
        self.df['Fix Condition 1'].value_counts().plot(kind='pie',
autopct='%1.1f%%', ax=ax3)
        ax3.set_title('Fix Condition Distribution')
        ax3.set_ylabel('')
        self._save_visualization(fig3, 'fix_condition_distribution')

class Task2Analyzer(ExcelDataAnalyzer):

    def analyze(self):
        self._clean_data()
        self._add_engineering_tags()
        self._generate_visualizations()

    def _clean_data(self):
        num_cols = ['TOTALCOST', 'KM', 'REPAIR_AGE']
        self.df[num_cols] = self.df[num_cols].apply(
            lambda x: pd.to_numeric(x.astype(str).str.replace('[^\
d.]', '', regex=True), errors='coerce'))
        self.df['CUSTOMER_VERBATIM'] =
self.df['CUSTOMER_VERBATIM'].str[:500]

    def _add_engineering_tags(self):
        components = ['steering', 'sensor', 'module', 'harness',
'strut']
        self.df['Failure Component'] =
self.df['CUSTOMER_VERBATIM'].apply(
            lambda x: next((c for c in components if c in
str(x).lower()), 'Other'))
        self.df['Cost Category'] = pd.cut(self.df['TOTALCOST'],
                                         bins=[0, 100, 500, 1000,
float('inf')],
                                         labels=['<100', '100-500',
'500-1000', '>1000'])

    def _generate_visualizations(self):
        fig1, ax1 = plt.subplots(figsize=(10, 6))
        sns.histplot(self.df['TOTALCOST'], bins=30, kde=True, ax=ax1)
        ax1.set_title('Repair Cost Distribution')
        ax1.set_xlabel('Total Cost (USD)')
        self._save_visualization(fig1, 'repair_cost_distribution')

        fig2, ax2 = plt.subplots(figsize=(10, 6))
        self.df['Failure Component'].value_counts().plot(kind='bar', 
ax=ax2)
        ax2.set_title('Component Failure Frequency')
        ax2.set_ylabel('Count')
        self._save_visualization(fig2, 'component_failures')

if __name__ == "__main__":
    try:
        task1 = Task1Analyzer('task1.xlsx')
        task1.analyze()
        task1.df.to_excel('Tulika task1_analyzed.xlsx', index=False)
        logger.info(f"Tulika's Task 1 visualizations: 
{task1.get_visualizations()}")

        task2 = Task2Analyzer('task2.xlsx')
        task2.analyze()
        task2.df.to_excel('Tulika task2_analyzed.xlsx', index=False)
        logger.info(f"Tulika's Task 2 visualizations: 
{task2.get_visualizations()}")

    except Exception as e:
        logger.error(f"Analysis failed: {str(e)}")
        raise
```

```
INFO:__main__:Saved visualization: 
Task1Analyzer_root_cause_distribution.png
INFO:__main__:Saved visualization: 
Task1Analyzer_symptom_component_distribution.png
INFO:__main__:Saved visualization: 
Task1Analyzer_fix_condition_distribution.png
INFO:__main__:Tulika's Task 1 visualizations: 
['Task1Analyzer_root_cause_distribution.png', 
'Task1Analyzer_symptom_component_distribution.png', 
'Task1Analyzer_fix_condition_distribution.png']
INFO:__main__:Saved visualization: 
Task2Analyzer_repair_cost_distribution.png
INFO:__main__:Saved visualization: 
Task2Analyzer_component_failures.png
INFO:__main__:Tulika's Task 2 visualizations: 
['Task2Analyzer_repair_cost_distribution.png', 
'Task2Analyzer_component_failures.png']
```