In [36]:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

class BidDataImporter:
    def __init__(self):
        # Common construction bid columns we expect to see
        self.expected_columns = {
            'item_number': str,
            'description': str,
            'quantity': float,
            'unit': str,
            'unit_price': float,
            'total_price': float
        }

        # Common unit conversions in construction
        self.unit_mappings = {
            'cubic yards': ['CY', 'cy', 'Cu Yd', 'cubic yard'],
            'square feet': ['SF', 'sf', 'SqFt', 'square foot'],
            'linear feet': ['LF', 'lf', 'Lin Ft', 'linear foot'],
            'each': ['EA', 'ea', 'Each', 'unit'],
            'tons': ['TN', 'tn', 'Tons', 'ton']
        }

    def read_excel_bid(self, file_path, sheet_name=0):
        try:
            df = pd.read_excel(file_path, sheet_name=sheet_name)
            df = self._clean_column_names(df)
            df = self._convert_data_types(df)
            df = self._standardize_units(df)
            self._validate_data(df)
            return df
        except Exception as e:
            print(f"Error reading file {file_path}: {str(e)}")
            return None

    def _clean_column_names(self, df):
        df.columns = df.columns.str.lower().str.replace(' ', '_')
        column_mappings = {
            'item_no': 'item_number',
            'item': 'item_number',
            'desc': 'description',
            'qty': 'quantity',
            'unit_cost': 'unit_price',
            'total': 'total_price'
        }
        return df.rename(columns=column_mappings)

    def _convert_data_types(self, df):
        for col, dtype in self.expected_columns.items():
            if col in df.columns:
                try:
                    if dtype in [float, int]:
                        if df[col].dtype == object:
                            df[col] = df[col].replace('[\$,]', '', regex=True)
                        df[col] = pd.to_numeric(df[col], errors='coerce')
                    else:
                        df[col] = df[col].astype(dtype)
                except Exception as e:
                    print(f"Warning: Could not convert {col} to {dtype}: {str(e)}")
        return df

    def _standardize_units(self, df):
        if 'unit' in df.columns:
            for standard_unit, variations in self.unit_mappings.items():
                df['unit'] = df['unit'].replace(variations, standard_unit)
        return df

    def _validate_data(self, df):
        # Check for missing columns
        missing_cols = set(self.expected_columns.keys()) - set(df.columns)
        if missing_cols:
            print(f"Warning: Missing expected columns: {missing_cols}")

        # Check for null values
        null_counts = df.isnull().sum()
        if null_counts.any():
            print("Warning: Found null values:")
            print(null_counts[null_counts > 0])

        # Validate calculations
        if all(col in df.columns for col in ['quantity', 'unit_price', 'total_price'])
            calculated_total = (df['quantity'] * df['unit_price']).round(2)
            mismatches = (calculated_total != df['total_price'].round(2))
            if mismatches.any():
                print(f"Warning: Found {mismatches.sum()} rows with calculation mismat
```

In [37]:

```
data = {
    'Item No': ['1001', '1002', '1003'],
    'Description': [
        'Excavation Work',
        'Concrete Foundation',
        'Steel Beams'
    ],
    'Qty': [100, 50, 25],
    'Unit': ['CY', 'CY', 'TN'],
    'Unit Cost': ['$45.00', '$125.00', '$1,200.00'],
    'Total': [4500, 6250, 30000] }
# Create DataFrame and save to Excel
df_original = pd.DataFrame(data)
df_original.to_excel('simple_bid.xlsx', index=False)
print("Original data:")
display(df_original)
```

```
Original data:
```

	Item No	Description	Qty	Unit	Unit Cost	Total
0	1001	Excavation Work	100	CY	$45.00	4500
1	1002	Concrete Foundation	50	CY	$125.00	6250
2	1003	Steel Beams	25	TN	$1,200.00	30000

In [38]:

```
importer = BidDataImporter()
```

```
# Read and clean the bid data
cleaned_bid = importer.read_excel_bid('simple_bid.xlsx')

print("\nCleaned data:")
display(cleaned_bid)
```

```
Cleaned data:
```

	item_number	description	quantity	unit	unit_price	total_price
0	1001	Excavation Work	100	cubic yards	45.0	4500
1	1002	Concrete Foundation	50	cubic yards	125.0	6250
2	1003	Steel Beams	25	tons	1200.0	30000

In [39]:

```
import scipy.stats as stats
```

In [40]:

```
class BidAnalysisTool:
    def __init__(self, bid_dataframes):
        """
        Initialize the analysis tool with multiple bid dataframes

        :param bid_dataframes: List of pandas DataFrames containing bid information
        """
        self.bids = bid_dataframes
        self.analysis_results = {}

    def calculate_price_variance(self, reference_bid=None):
        """
        Calculate price variances across different bids

        :param reference_bid: Optional reference bid for comparison (default: first bi
        :return: DataFrame with variance calculations
        """
        if reference_bid is None:
            reference_bid = self.bids[0]

        variance_results = []

        for bid in self.bids:
            # Merge bids on item descriptions
            merged_bid = pd.merge(
                reference_bid[['description', 'unit_price']],
                bid[['description', 'unit_price']],
                on='description',
                suffixes=('_reference', '_compared')
            )

            # Calculate percentage variance
            merged_bid['price_variance_pct'] = (
                (merged_bid['unit_price_compared'] - merged_bid['unit_price_reference'
                merged_bid['unit_price_reference'] * 100
            )

            variance_results.append(merged_bid)

        return variance_results
    def detect_price_outliers(self, method='zscore', threshold=2.5):
        """
        Detect price outliers using statistical methods

        :param method: Statistical method for outlier detection
        :param threshold: Significance threshold for outliers
        :return: Outliers in bid prices
        """
        outliers = {}

        for i, bid in enumerate(self.bids):
            if method == 'zscore':
                # Z-score method
                z_scores = np.abs(stats.zscore(bid['unit_price']))
                bid_outliers = bid[z_scores > threshold]
            elif method == 'iqr':
                # Interquartile Range method
                Q1 = bid['unit_price'].quantile(0.25)
                Q3 = bid['unit_price'].quantile(0.75)
                IQR = Q3 - Q1
                lower_bound = Q1 - 1.5 * IQR
                upper_bound = Q3 + 1.5 * IQR
                bid_outliers = bid[(bid['unit_price'] < lower_bound) | (bid['unit_pric

            outliers[f'Bid_{i+1}'] = bid_outliers

        return outliers
    def compare_total_bid_costs(self):
        """
        Compare total costs across different bids

        :return: Summary of total bid costs and comparisons
        """
        bid_cost_summary = []

        for i, bid in enumerate(self.bids):
            total_cost = bid['total_price'].sum()
            mean_unit_price = bid['unit_price'].mean()
            median_unit_price = bid['unit_price'].median()

            bid_summary = {
                'Bid_Number': i + 1,
                'Total_Cost': total_cost,
                'Mean_Unit_Price': mean_unit_price,
                'Median_Unit_Price': median_unit_price
            }

            bid_cost_summary.append(bid_summary)

        return pd.DataFrame(bid_cost_summary)

    def statistical_bid_analysis(self):
        """
        Comprehensive statistical analysis of bids

        :return: Detailed statistical insights
        """
        # Price variance analysis
        price_variances = self.calculate_price_variance()

        # Outlier detection
        price_outliers = self.detect_price_outliers()
```

```python
        # Total cost comparison
        cost_comparison = self.compare_total_bid_costs()

        return {
            'Price_Variances': price_variances,
            'Price_Outliers': price_outliers,
            'Cost_Comparison': cost_comparison
        }

# Example Usage and Demonstration
def main():
    # Sample bid data (simulated)
    bid1_data = {
        'description': ['Excavation', 'Concrete', 'Framing'],
        'quantity': [100, 50, 200],
        'unit_price': [45.50, 125.75, 75.25],
        'total_price': [4550, 6287.50, 15050]
    }

    bid2_data = {
        'description': ['Excavation', 'Concrete', 'Framing'],
        'quantity': [100, 50, 200],
        'unit_price': [48.25, 130.00, 72.90],
        'total_price': [4825, 6500, 14580]
    }

    # Create DataFrames
    bid1_df = pd.DataFrame(bid1_data)
    bid2_df = pd.DataFrame(bid2_data)

    # Initialize Analysis Tool
    bid_analysis = BidAnalysisTool([bid1_df, bid2_df])

    # Perform Analysis
    analysis_results = bid_analysis.statistical_bid_analysis()

    # Display Results
    print("Price Variances:")
    print(analysis_results['Price_Variances'])

    print("\nPrice Outliers:")
    print(analysis_results['Price_Outliers'])

    print("\nCost Comparison:")
    print(analysis_results['Cost_Comparison'])

if __name__ == "__main__":
    main()
```

```
Price Variances:
[  description  unit_price_reference  unit_price_compared  price_variance_pct
0  Excavation                  45.50                45.50                 0.0
1    Concrete                 125.75               125.75                 0.0
2     Framing                  75.25                75.25                 0.0,   descr
iption  unit_price_reference  unit_price_compared  price_variance_pct
0  Excavation                  45.50                48.25            6.043956
1    Concrete                 125.75               130.00            3.379722
2     Framing                  75.25                72.90           -3.122924]

Price Outliers:
{'Bid_1': Empty DataFrame
Columns: [description, quantity, unit_price, total_price]
Index: [], 'Bid_2': Empty DataFrame
Columns: [description, quantity, unit_price, total_price]
Index: []}

Cost Comparison:
   Bid_Number  Total_Cost  Mean_Unit_Price  Median_Unit_Price
0           1     25887.5        82.166667              75.25
1           2     25905.0        83.716667              72.90
```

In [13]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import openpyxl
from openpyxl.styles import PatternFill, Font, Alignment, Border, Side
from openpyxl.formatting.rule import ColorScaleRule, CellIsRule
```

In [41]:

```python
class BidReporter:
    def __init__(self):
        # Set style for better-looking plots
        plt.style.use('seaborn')
        self.colors = sns.color_palette("husl", 8)

    def create_summary_report(self, df):
        """
        Create a summary report of the bid data
        """
        summary = pd.DataFrame()

        # Basic statistics
        summary['Total Items'] = [len(df)]
        summary['Total Cost'] = [df['total_price'].sum()]
        summary['Average Unit Price'] = [df['unit_price'].mean()]
        summary['Median Unit Price'] = [df['unit_price'].median()]
        summary['Total Quantity'] = [df['quantity'].sum()]

        # Price ranges
        summary['Lowest Unit Price'] = [df['unit_price'].min()]
        summary['Highest Unit Price'] = [df['unit_price'].max()]
        summary['Price Range'] = [df['unit_price'].max() - df['unit_price'].min()]

        # Unit type counts
        unit_counts = df['unit'].value_counts()
        for unit in unit_counts.index:
            summary[f'Count of {unit}'] = [unit_counts[unit]]

        # Format currency columns
        currency_cols = ['Total Cost', 'Average Unit Price', 'Median Unit Price',
                        'Lowest Unit Price', 'Highest Unit Price', 'Price Range']
        for col in currency_cols:
            summary[col] = summary[col].map('${:,.2f}'.format)

        return summary.T.rename(columns={0: 'Value'})

    def plot_cost_distribution(self, df, figsize=(15, 10)):
        """Create comprehensive cost analysis visualizations"""
        fig = plt.figure(figsize=figsize)

        # 1. Total Price Distribution
        plt.subplot(2, 2, 1)
        sns.barplot(x=df.index, y='total_price', data=df, color=self.colors[0])
        plt.title('Total Price by Item')
        plt.xticks(rotation=45)
        plt.ylabel('Total Price ($)')

        # 2. Unit Price Comparison
        plt.subplot(2, 2, 2)
        sns.barplot(x=df.index, y='unit_price', data=df, color=self.colors[1])
        plt.title('Unit Price Comparison')
        plt.xticks(rotation=45)
        plt.ylabel('Unit Price ($)')

        # 3. Cost Breakdown Pie Chart
        plt.subplot(2, 2, 3)
        plt.pie(df['total_price'], labels=df['description'], autopct='%1.1f%%',
                colors=self.colors)
        plt.title('Cost Breakdown by Item')

        # 4. Unit Type Distribution
        plt.subplot(2, 2, 4)
        unit_counts = df['unit'].value_counts()
        sns.barplot(x=unit_counts.index, y=unit_counts.values, color=self.colors[2])
        plt.title('Distribution of Unit Types')
        plt.xticks(rotation=45)

        plt.tight_layout()
        return fig

    def plot_detailed_analysis(self, df, figsize=(15, 12)):
        """Create detailed analysis visualizations"""
        fig = plt.figure(figsize=figsize)

        # 1. Quantity vs Total Price Scatter
        plt.subplot(2, 2, 1)
        sns.scatterplot(data=df, x='quantity', y='total_price', size='unit_price',
                       sizes=(100, 1000), color=self.colors[3])
        plt.title('Quantity vs Total Price')
        plt.xlabel('Quantity')
        plt.ylabel('Total Price ($)')

        # 2. Unit Price Range by Unit Type
        plt.subplot(2, 2, 2)
        sns.boxplot(data=df, x='unit', y='unit_price', color=self.colors[4])
        plt.title('Unit Price Range by Unit Type')
        plt.xticks(rotation=45)
        plt.ylabel('Unit Price ($)')

        # 3. Cumulative Cost Chart
        plt.subplot(2, 2, 3)
        cumulative_cost = df['total_price'].cumsum()
        plt.plot(range(len(cumulative_cost)), cumulative_cost,
                marker='o', color=self.colors[5])
        plt.title('Cumulative Cost')
        plt.xlabel('Number of Items')
        plt.ylabel('Cumulative Cost ($)')

        # 4. Cost Proportion Analysis
        plt.subplot(2, 2, 4)
        cost_proportion = (df['total_price'] / df['total_price'].sum()) * 100
        sns.barplot(x=df.index, y=cost_proportion, color=self.colors[6])
        plt.title('Cost Proportion Analysis')
        plt.ylabel('Percentage of Total Cost')
        plt.xticks(rotation=45)

        plt.tight_layout()
        return fig

    def plot_unit_analysis(self, df, figsize=(15, 5)):
        """Create unit-specific analysis visualizations"""
        fig = plt.figure(figsize=figsize)

        # 1. Average Cost per Unit Type
        plt.subplot(1, 3, 1)
        unit_avg_price = df.groupby('unit')['unit_price'].mean()
        sns.barplot(x=unit_avg_price.index, y=unit_avg_price.values,
                   color=self.colors[7])
        plt.title('Average Cost per Unit Type')
        plt.xticks(rotation=45)
        plt.ylabel('Average Unit Price ($)')

        # 2. Quantity Distribution by Unit
        plt.subplot(1, 3, 2)
        unit_quantities = df.groupby('unit')['quantity'].sum()
        sns.barplot(x=unit_quantities.index, y=unit_quantities.values,
                   color=self.colors[0])
        plt.title('Quantity Distribution by Unit')
        plt.xticks(rotation=45)
        plt.ylabel('Total Quantity')

        # 3. Unit Price Density
        plt.subplot(1, 3, 3)
        sns.kdeplot(data=df, x='unit_price', fill=True, color=self.colors[1])
        plt.title('Unit Price Distribution Density')
        plt.xlabel('Unit Price ($)')

        plt.tight_layout()
        return fig

    def generate_excel_report(self, df, output_path):
        """Generate a comprehensive Excel report with multiple sheets"""
        with pd.ExcelWriter(output_path, engine='xlsxwriter') as writer:
            # Write raw data
            df.to_excel(writer, sheet_name='Raw Data', index=False)

            # Write summary statistics
            summary = self.create_summary_report(df)
            summary.to_excel(writer, sheet_name='Summary Statistics')
            # Create pivot tables
            unit_pivot = pd.pivot_table(df,
                                      values=['quantity', 'total_price'],
                                      index='unit',
                                      aggfunc={'quantity': 'sum',
                                              'total_price': 'sum'})
            unit_pivot.to_excel(writer, sheet_name='Unit Analysis')

            # Format the Excel file
            workbook = writer.book
            currency_format = workbook.add_format({'num_format': '$#,##0.00'})

            # Apply formatting to relevant sheets
            worksheet = writer.sheets['Raw Data']
            worksheet.set_column('D:E', 12, currency_format)  # Format price columns

            worksheet = writer.sheets['Unit Analysis']
            worksheet.set_column('B:B', 12, currency_format)  # Format total_price col

    def generate_all_visualizations(self, df):
        """Generate all visualizations at once"""
        visualizations = {
            'cost_distribution': self.plot_cost_distribution(df),
            'detailed_analysis': self.plot_detailed_analysis(df),
            'unit_analysis': self.plot_unit_analysis(df)
        }
        return visualizations
```

In [25]:

```
pip install xlsxwriter
```

```
Collecting xlsxwriter
  Obtaining dependency information for xlsxwriter from https://files.pythonhosted.or
g/packages/a7/ea/53d1fe468e63e092cf16e2c18d16f50c29851242f9dd12d6a66e0d7f0d02/XlsxWri
ter-3.2.0-py3-none-any.whl.metadata
  Downloading XlsxWriter-3.2.0-py3-none-any.whl.metadata (2.6 kB)
Downloading XlsxWriter-3.2.0-py3-none-any.whl (159 kB)
   ---------------------------------------- 0.0/159.9 kB ? eta -:--:--
   -- ------------------------------------- 10.2/159.9 kB ? eta -:--:--
   -- ------------------------------------- 10.2/159.9 kB ? eta -:--:--
   -------------- ------------------------- 61.4/159.9 kB 550.5 kB/s eta 0:00:01
   ------------------------ ------------- 102.4/159.9 kB 658.3 kB/s eta 0:00:01
   -------------------------------------- 159.9/159.9 kB 800.5 kB/s eta 0:00:00
Installing collected packages: xlsxwriter
Successfully installed xlsxwriter-3.2.0
Note: you may need to restart the kernel to use updated packages.
```

In [42]:

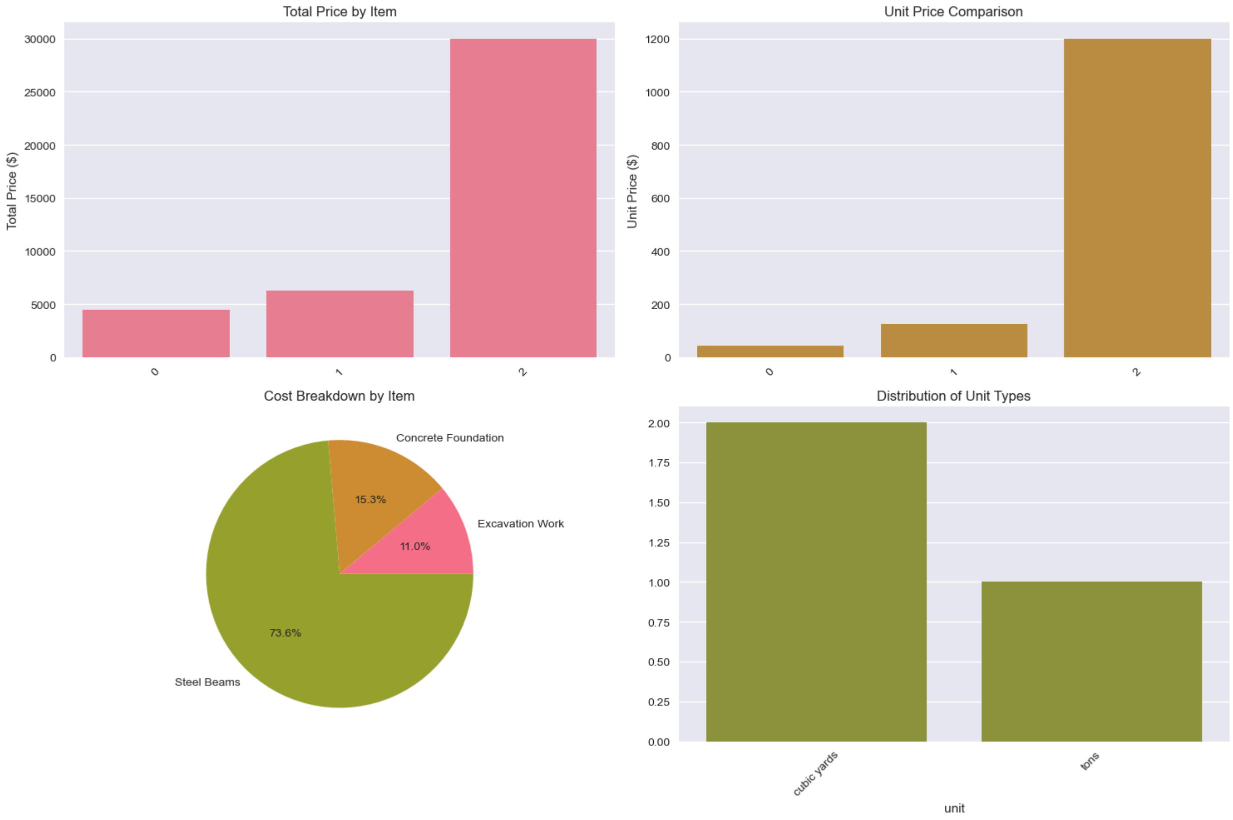
```
importer = BidDataImporter()  # From our previous module
cleaned_bid = importer.read_excel_bid('simple_bid.xlsx')

# Create reporter instance
reporter = BidReporter()

# Generate summary
print("Summary Statistics:")
display(reporter.create_summary_report(cleaned_bid))

# Create visualizations
print("\nGenerating visualizations...")
reporter.plot_cost_distribution(cleaned_bid)
plt.show()
```

```
# Generate Excel report
reporter.generate_excel_report(cleaned_bid, 'bid_analysis_report.xlsx')
```

```
Summary Statistics:
```

	Value
Total Items	3
Total Cost	$40,750.00
Average Unit Price	$456.67
Median Unit Price	$125.00
Total Quantity	175
Lowest Unit Price	$45.00
Highest Unit Price	$1,200.00
Price Range	$1,155.00
Count of cubic yards	2
Count of tons	1

```
Generating visualizations...
```



```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
Cell In[42], line 17
     14 plt.show()
     16 # Generate Excel report
---> 17 reporter.generate_excel_report(cleaned_bid, 'bid_analysis_report.xlsx')

Cell In[41], line 144, in BidReporter.generate_excel_report(self, df, output_path)
    142 def generate_excel_report(self, df, output_path):
    143     """Generate a comprehensive Excel report with multiple sheets"""
--> 144     with pd.ExcelWriter(output_path, engine='xlsxwriter') as writer:
    145         # Write raw data
    146         df.to_excel(writer, sheet_name='Raw Data', index=False)
    148         # Write summary statistics

File ~\anaconda3\Lib\site-packages\pandas\io\excel\_xlsxwriter.py:199, in XlsxWriter.
__init__(self, path, engine, date_format, datetime_format, mode, storage_options, if_
sheet_exists, engine_kwargs, **kwargs)
    196 if mode == "a":
    197     raise ValueError("Append mode is not supported with xlsxwriter!")
--> 199 super().__init__(
    200     path,
    201     engine=engine,
    202     date_format=date_format,
    203     datetime_format=datetime_format,
    204     mode=mode,
    205     storage_options=storage_options,
    206     if_sheet_exists=if_sheet_exists,
    207     engine_kwargs=engine_kwargs,
    208 )
    210 self._book = Workbook(self._handles.handle, **engine_kwargs)

File ~\anaconda3\Lib\site-packages\pandas\io\excel\_base.py:1219, in ExcelWriter.__in
it__(self, path, engine, date_format, datetime_format, mode, storage_options, if_shee
t_exists, engine_kwargs)
   1215 self._handles = IOHandles(
   1216     cast(IO[bytes], path), compression={"compression": None}
   1217 )
   1218 if not isinstance(path, ExcelWriter):
-> 1219     self._handles = get_handle(
   1220         path, mode, storage_options=storage_options, is_text=False
   1221     )
   1222 self._cur_sheet = None
   1224 if date_format is None:

File ~\anaconda3\Lib\site-packages\pandas\io\common.py:868, in get_handle(path_or_bu
f, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    859         handle = open(
    860             handle,
    861             ioargs.mode,
   (...)
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
--> 868         handle = open(handle, ioargs.mode)
    869     handles.append(handle)
    871 # Convert BytesIO or file objects passed with an encoding

PermissionError: [Errno 13] Permission denied: 'bid_analysis_report.xlsx'
```

In [43]:

```
import logging
import json
import os
from datetime import datetime

class BidDataExporter:
    def __init__(self, logger=None):
        """
        Initialize the data exporter with optional logging

        :param logger: Optional logger object, creates a default logger if not provide
        """
        self.logger = logger or self._setup_logger()

    def _setup_logger(self):
        """
        Set up a default logger for tracking export and documentation processes

        :return: Configured logger object
        """
        logger = logging.getLogger('BidDataExporter')
        logger.setLevel(logging.INFO)

        # Console handler
        console_handler = logging.StreamHandler()
        console_handler.setLevel(logging.INFO)
        formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(mess
        console_handler.setFormatter(formatter)

        logger.addHandler(console_handler)
        return logger

    def _convert_numpy_types(self, obj):
        """
        Convert NumPy types to standard Python types

        :param obj: Object to convert
        :return: Converted object
        """
        if isinstance(obj, (np.integer, np.int64)):
            return int(obj)
        elif isinstance(obj, (np.float64, np.float32)):
            return float(obj)
        elif isinstance(obj, np.ndarray):
            return obj.tolist()
        return obj

    def export_to_json(self, dataframe, output_path, indent=4):
        """
        Export DataFrame to JSON with detailed metadata

        :param dataframe: Pandas DataFrame to export
        :param output_path: Path to save the JSON file
        :param indent: Indentation for JSON formatting
        :return: Path to the exported JSON file
        """
        try:
            # Prepare comprehensive export with metadata
            export_data = {
                'metadata': {
                    'columns': list(dataframe.columns),
                    'row_count': len(dataframe),
                    'total_cost': self._convert_numpy_types(dataframe['total_price'].s
                    'export_timestamp': datetime.now().isoformat()
                },
                'data': [
                    {k: self._convert_numpy_types(v) for k, v in row.items()}
                    for row in dataframe.to_dict(orient='records')
                ]
            }

            with open(output_path, 'w') as f:
                json.dump(export_data, f, indent=indent)

            self.logger.info(f"Successfully exported data to {output_path}")
            return output_path
        except Exception as e:
            self.logger.error(f"Error exporting to JSON: {e}")
            raise

    def create_comprehensive_excel_report(self, dataframe, output_path):
        """
        Create a comprehensive Excel report with multiple sheets and advanced formatti

        :param dataframe: Pandas DataFrame to export
        :param output_path: Path to save the Excel file
        :return: Path to the created Excel report
        """
        try:
            with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
                # 1. Raw Data Sheet
                dataframe.to_excel(writer, sheet_name='Raw Data', index=False)

                # 2. Summary Statistics Sheet
                summary_stats = dataframe.describe().T
                summary_stats.to_excel(writer, sheet_name='Summary')

                # 3. Pivot Tables Sheet
                pivot_table = pd.pivot_table(
                    dataframe,
                    values=['quantity', 'total_price'],
                    index='unit',
                    aggfunc=['sum', 'mean']
                )
                pivot_table.to_excel(writer, sheet_name='Pivot Analysis')

            # Post-processing with openpyxl for advanced formatting
            wb = openpyxl.load_workbook(output_path)

            # Apply formatting to Raw Data sheet
            raw_data_sheet = wb['Raw Data']
            header_fill = PatternFill(start_color='DDDDDD', end_color='DDDDDD', fill_t

            for cell in raw_data_sheet[1]:
                cell.font = Font(bold=True)
                cell.fill = header_fill
                cell.alignment = Alignment(horizontal='center')

            # Color scale for total_price column
            if 'F' in raw_data_sheet.column_dimensions:
                price_column = [cell for cell in raw_data_sheet['F'] if cell.value is
                if len(price_column) > 1:
                    min_val = min(cell.value for cell in price_column[1:])
                    max_val = max(cell.value for cell in price_column[1:])
                    color_scale_rule = ColorScaleRule(
                        start_type='min', start_color='green',
                        end_type='max', end_color='red'
                    )
                    raw_data_sheet.conditional_formatting.add('F2:F'+str(len(price_col

            wb.save(output_path)

            self.logger.info(f"Comprehensive report created at {output_path}")
            return output_path

        except Exception as e:
            self.logger.error(f"Error creating comprehensive report: {e}")
            raise

    def generate_data_validation_report(self, dataframe):
        """
        Generate a comprehensive data validation report

        :param dataframe: Pandas DataFrame to validate
        :return: Dictionary containing validation results
        """
        validation_results = {
            'total_rows': len(dataframe),
            'total_columns': len(dataframe.columns),
            'missing_values': dataframe.isnull().sum().to_dict(),
            'unique_values_per_column': {col: dataframe[col].nunique() for col in data
            'data_types': {col: str(dtype) for col, dtype in dataframe.dtypes.items()}
            'price_validation': {
                'total_cost': self._convert_numpy_types(dataframe['total_price'].sum()
                'avg_unit_price': self._convert_numpy_types(dataframe['unit_price'].me
                'price_consistency': bool(np.allclose(
                    dataframe['total_price'],
                    dataframe['quantity'] * dataframe['unit_price'],
                    rtol=1e-05,
                    atol=1e-08
                ))
            }
        }

        return validation_results

    def save_validation_report(self, validation_report, output_path='data_validation_r
        """
        Save data validation report to a JSON file

        :param validation_report: Validation report dictionary
        :param output_path: Path to save the validation report
        :return: Path to the saved validation report
        """
        try:
            # Convert any remaining numpy types
            def convert_numpy_recursive(obj):
                if isinstance(obj, dict):
                    return {k: convert_numpy_recursive(v) for k, v in obj.items()}
                elif isinstance(obj, list):
                    return [convert_numpy_recursive(v) for v in obj]
                else:
                    return self._convert_numpy_types(obj)

            converted_report = convert_numpy_recursive(validation_report)

            with open(output_path, 'w') as f:
                json.dump(converted_report, f, indent=4)

            self.logger.info(f"Validation report saved to {output_path}")
            return output_path
        except Exception as e:
            self.logger.error(f"Error saving validation report: {e}")
            raise

# Example Usage
def main():

    # Initialize importer and exporter
    importer = BidDataImporter()
    exporter = BidDataExporter()

    # Read bid data
    bid_data = importer.read_excel_bid('simple_bid.xlsx')

    # Export to JSON
    exporter.export_to_json(bid_data, 'bid_data_export.json')

    # Create comprehensive Excel report
    exporter.create_comprehensive_excel_report(bid_data, 'comprehensive_bid_report.xls

    # Generate data validation report
    validation_report = exporter.generate_data_validation_report(bid_data)
    exporter.save_validation_report(validation_report)

if __name__ == "__main__":
    main()
```

```
2024-12-08 11:06:18,990 - BidDataExporter - INFO - Successfully exported data to bid_
data_export.json
2024-12-08 11:06:18,990 - BidDataExporter - INFO - Successfully exported data to bid_
data_export.json
2024-12-08 11:06:18,990 - BidDataExporter - INFO - Successfully exported data to bid_
data_export.json
2024-12-08 11:06:18,992 - BidDataExporter - ERROR - Error creating comprehensive repo
rt: [Errno 13] Permission denied: 'comprehensive_bid_report.xlsx'
2024-12-08 11:06:18,992 - BidDataExporter - ERROR - Error creating comprehensive repo
rt: [Errno 13] Permission denied: 'comprehensive_bid_report.xlsx'
2024-12-08 11:06:18,992 - BidDataExporter - ERROR - Error creating comprehensive repo
rt: [Errno 13] Permission denied: 'comprehensive_bid_report.xlsx'
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
Cell In[43], line 217
    214     exporter.save_validation_report(validation_report)
    216 if __name__ == "__main__":
--> 217     main()

Cell In[43], line 210, in main()
    207 exporter.export_to_json(bid_data, 'bid_data_export.json')
    209 # Create comprehensive Excel report
--> 210 exporter.create_comprehensive_excel_report(bid_data, 'comprehensive_bid_repor
t.xlsx')
    212 # Generate data validation report
    213 validation_report = exporter.generate_data_validation_report(bid_data)

Cell In[43], line 90, in BidDataExporter.create_comprehensive_excel_report(self, data
frame, output_path)
     82 """
     83 Create a comprehensive Excel report with multiple sheets and advanced formatt
ing
     84 
   (...)
     87 :return: Path to the created Excel report
     88 """
     89 try:
---> 90     with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
     91         # 1. Raw Data Sheet
     92         dataframe.to_excel(writer, sheet_name='Raw Data', index=False)
     94         # 2. Summary Statistics Sheet

File ~\anaconda3\Lib\site-packages\pandas\io\excel\_openpyxl.py:60, in OpenpyxlWrite
r.__init__(self, path, engine, date_format, datetime_format, mode, storage_options, i
f_sheet_exists, engine_kwargs, **kwargs)
     56 from openpyxl.workbook import Workbook
     58 engine_kwargs = combine_kwargs(engine_kwargs, kwargs)
---> 60 super().__init__(
     61     path,
     62     mode=mode,
     63     storage_options=storage_options,
     64     if_sheet_exists=if_sheet_exists,
     65     engine_kwargs=engine_kwargs,
     66 )
     68 # ExcelWriter replaced "a" by "r+" to allow us to first read the excel file f
rom
     69 # the file and later write to it
     70 if "r+" in self._mode:  # Load from existing workbook

File ~\anaconda3\Lib\site-packages\pandas\io\excel\_base.py:1219, in ExcelWriter.__in
it__(self, path, engine, date_format, datetime_format, mode, storage_options, if_shee
t_exists, engine_kwargs)
   1215 self._handles = IOHandles(
   1216     cast(IO[bytes], path), compression={"compression": None}
   1217 )
   1218 if not isinstance(path, ExcelWriter):
-> 1219     self._handles = get_handle(
   1220         path, mode, storage_options=storage_options, is_text=False
   1221     )
   1222 self._cur_sheet = None
   1224 if date_format is None:
```

```
File ~\anaconda3\Lib\site-packages\pandas\io\common.py:868, in get_handle(path_or_bu
f, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    859         handle = open(
    860             handle,
    861             ioargs.mode,
   (...)
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
--> 868         handle = open(handle, ioargs.mode)
    869     handles.append(handle)
    871 # Convert BytesIO or file objects passed with an encoding

PermissionError: [Errno 13] Permission denied: 'comprehensive_bid_report.xlsx'
```

In [ ]: