In [ ]:
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Step 2.1: Numerical Variables Analysis
This script analyzes the numerical variables in the female farmers dataset.
It produces summary statistics, distributions, and identifies patterns and outliers.

Author: [Your Name]
Date: March 31, 2025
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import os

# Set style for plots
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette('viridis')

# Create output directory for results
output_dir = 'results/numerical_analysis'
os.makedirs(output_dir, exist_ok=True)

def load_data():
    """Load the preprocessed dataset"""
    print("Loading the dataset...")
    data = pd.read_excel('fixed_female_farmers_data.xlsx')
    print(f"Dataset loaded with {data.shape[0]} rows and {data.shape[1]} columns")
    return data

def identify_numerical_variables(data):
    """Identify numerical variables in the dataset"""
    # Define expected numerical variables based on the codebook
    expected_numerical = [
        'Age', 'Nb enfants', 'Nb pers à charge', 'H travail / jour',
        'J travail / Sem', 'Ancienneté agricole', 'Poids', 'Taille',
```

```
        'TAS', 'TAD', 'GAD'
    ]

    # Filter to include only columns that exist in the dataset
    numerical_vars = [var for var in expected_numerical if var in data.columns]

    # Add any other numerical variables from the dataset
    for col in data.columns:
        if col not in numerical_vars and data[col].dtype in ['int64', 'float64']:
            numerical_vars.append(col)

    print(f"Identified {len(numerical_vars)} numerical variables: {', '.join(numerical_vars)}")
    return numerical_vars

def calculate_summary_statistics(data, numerical_vars):
    """
    Calculate and save summary statistics for numerical variables
    """
    print("Calculating summary statistics...")

    # Basic summary statistics
    summary = data[numerical_vars].describe().T

    # Add additional statistics
    summary['range'] = summary['max'] - summary['min']
    summary['cv'] = summary['std'] / summary['mean']  # Coefficient of variation
    summary['missing'] = data[numerical_vars].isnull().sum()
    summary['missing_pct'] = (data[numerical_vars].isnull().sum() / len(data)) * 100
    summary['skewness'] = data[numerical_vars].skew()
    summary['kurtosis'] = data[numerical_vars].kurtosis()

    # Round statistics for better readability
    summary = summary.round(2)

    # Save summary to CSV
    summary_path = os.path.join(output_dir, "numerical_summary_statistics.csv")
    summary.to_csv(summary_path)
    print(f"Summary statistics saved to {summary_path}")

    # Print a part of the summary for quick reference
    print("\nSummary statistics overview (partial):")
    print(summary[['count', 'mean', 'std', 'min', 'max', 'missing_pct']].to_string())
```

```
    return summary

def create_distribution_plots(data, numerical_vars):
    """
    Create distribution plots for each numerical variable
    """
    print("\nCreating distribution plots...")

    for var in numerical_vars:
        print(f"  Creating distribution plot for {var}")

        # Create figure with two subplots (histogram and boxplot)
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
        fig.suptitle(f'Distribution of {var}', fontsize=16)

        # Histogram with KDE
        sns.histplot(data[var].dropna(), kde=True, ax=ax1)
        ax1.set_title(f'Histogram of {var}')
        ax1.set_xlabel(var)
        ax1.set_ylabel('Frequency')

        # Add mean and median lines
        mean_val = data[var].mean()
        median_val = data[var].median()
        ax1.axvline(mean_val, color='red', linestyle='--', label=f'Mean: {mean_val:.2f}')
        ax1.axvline(median_val, color='green', linestyle='-.', label=f'Median: {median_val:.2f}')
        ax1.legend()

        # Boxplot
        sns.boxplot(y=data[var].dropna(), ax=ax2)
        ax2.set_title(f'Boxplot of {var}')
        ax2.set_ylabel(var)

        # Add annotations for outliers
        Q1 = data[var].quantile(0.25)
        Q3 = data[var].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        outliers = data[var][(data[var] < lower_bound) | (data[var] > upper_bound)]
```

```
        outlier_count = len(outliers)

        if outlier_count > 0:
            ax2.text(0.5, 0.01,
                    f"Outliers: {outlier_count} ({outlier_count/len(data[var].dropna())*100:.1f}%)",
                    transform=ax2.transAxes, ha='center', va='bottom',
                    bbox=dict(facecolor='white', alpha=0.8))

        # Add a text box with statistics
        stats_text = (
            f"Count: {data[var].count()}\n"
            f"Mean: {data[var].mean():.2f}\n"
            f"Std Dev: {data[var].std():.2f}\n"
            f"Min: {data[var].min():.2f}\n"
            f"25%: {data[var].quantile(0.25):.2f}\n"
            f"Median: {data[var].median():.2f}\n"
            f"75%: {data[var].quantile(0.75):.2f}\n"
            f"Max: {data[var].max():.2f}\n"
            f"Missing: {data[var].isnull().sum()} ({data[var].isnull().sum()/len(data)*100:.1f}%)"
        )

        # Add text box to the first subplot
        props = dict(boxstyle='round', facecolor='white', alpha=0.8)
        ax1.text(0.05, 0.95, stats_text, transform=ax1.transAxes, fontsize=10,
                verticalalignment='top', bbox=props)

        plt.tight_layout()

        # Save the figure
        fig_path = os.path.join(output_dir, f"{var}_distribution.png")
        plt.savefig(fig_path, dpi=300)
        plt.close()
        print(f"    Plot saved to {fig_path}")

def analyze_normality(data, numerical_vars):
    """
    Test for normality using Shapiro-Wilk test and QQ plots
    """
    print("\nAnalyzing normality of numerical variables...")

    # Create a DataFrame to store normality test results
    normality_results = pd.DataFrame(
```

```
        columns=['Variable', 'Shapiro_Stat', 'Shapiro_p', 'Skewness', 'Kurtosis', 'Normal_Distribution']
    )

    for var in numerical_vars:
        # Drop missing values
        values = data[var].dropna()

        # Skip if too few values or too many
        if len(values) < 3 or len(values) > 5000:  # Shapiro-Wilk works best for small to moderate samples
            print(f"  Skipping normality test for {var} due to sample size constraints")
            continue

        print(f"  Testing normality for {var}")

        # Create QQ plot
        fig, ax = plt.subplots(figsize=(10, 6))
        stats.probplot(values, dist="norm", plot=ax)
        plt.title(f'Q-Q Plot for {var}', fontsize=14)
        plt.grid(True, linestyle='--', alpha=0.7)

        # Save QQ plot
        qq_path = os.path.join(output_dir, f"{var}_qq_plot.png")
        plt.savefig(qq_path, dpi=300)
        plt.close()
        print(f"    QQ plot saved to {qq_path}")

        # Shapiro-Wilk test for normality
        try:
            stat, p = stats.shapiro(values)

            # Calculate skewness and kurtosis
            skewness = stats.skew(values)
            kurtosis = stats.kurtosis(values)

            # Determine if normally distributed (p > 0.05)
            is_normal = "Yes" if p > 0.05 else "No"

            # Add to results DataFrame
            result = pd.DataFrame({
                'Variable': [var],
                'Shapiro_Stat': [stat],
                'Shapiro_p': [p],
```

```
                'Skewness': [skewness],
                'Kurtosis': [kurtosis],
                'Normal_Distribution': [is_normal]
            })
            normality_results = pd.concat([normality_results, result], ignore_index=True)

            print(f"    Shapiro-Wilk test: stat={stat:.4f}, p={p:.4f}")
            print(f"    Skewness: {skewness:.4f}, Kurtosis: {kurtosis:.4f}")
            print(f"    Normal distribution: {is_normal}")

        except Exception as e:
            print(f"    Error testing normality for {var}: {e}")

    # Save normality test results
    if not normality_results.empty:
        normality_path = os.path.join(output_dir, "normality_test_results.csv")
        normality_results.to_csv(normality_path, index=False)
        print(f"Normality test results saved to {normality_path}")

    return normality_results

def create_correlation_matrix(data, numerical_vars):
    """
    Create and visualize correlation matrix for numerical variables
    """
    print("\nAnalyzing correlations between numerical variables...")

    # Calculate correlation matrix
    corr_matrix = data[numerical_vars].corr()

    # Save correlation matrix to CSV
    corr_path = os.path.join(output_dir, "correlation_matrix.csv")
    corr_matrix.to_csv(corr_path)
    print(f"Correlation matrix saved to {corr_path}")

    # Create correlation heatmap
    plt.figure(figsize=(14, 10))
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    cmap = sns.diverging_palette(230, 20, as_cmap=True)

    sns.heatmap(
        corr_matrix,
```

```
        annot=True,
        fmt=".2f",
        cmap=cmap,
        mask=mask,
        linewidths=0.5,
        vmin=-1,
        vmax=1,
        square=True
    )
    plt.title('Correlation Matrix: Numerical Variables', fontsize=16)
    plt.tight_layout()

    # Save correlation heatmap
    heatmap_path = os.path.join(output_dir, "correlation_heatmap.png")
    plt.savefig(heatmap_path, dpi=300)
    plt.close()
    print(f"Correlation heatmap saved to {heatmap_path}")

    # Identify and report strong correlations
    strong_correlations = []

    for i in range(len(numerical_vars)):
        for j in range(i+1, len(numerical_vars)):
            var1 = numerical_vars[i]
            var2 = numerical_vars[j]
            corr = corr_matrix.loc[var1, var2]

            if abs(corr) >= 0.5:  # Consider correlations >= 0.5 as strong
                strong_correlations.append((var1, var2, corr))

    if strong_correlations:
        print("\nStrong correlations (|r| ≥ 0.5):")
        for var1, var2, corr in sorted(strong_correlations, key=lambda x: abs(x[2]), reverse=True):
            print(f"  {var1} & {var2}: r = {corr:.2f}")

        # Save strong correlations to CSV
        strong_corr_df = pd.DataFrame(strong_correlations, columns=['Variable 1', 'Variable 2', 'Correlation'])
        strong_corr_path = os.path.join(output_dir, "strong_correlations.csv")
        strong_corr_df.to_csv(strong_corr_path, index=False)
        print(f"Strong correlations saved to {strong_corr_path}")
    else:
        print("No strong correlations (|r| ≥ 0.5) found among numerical variables.")
```

```python
def create_pairplots(data, numerical_vars):
    """
    Create pairplots for selected numerical variables to visualize relationships
    """
    print("\nCreating pairplots for key numerical variables...")

    # Select top numerical variables (to avoid too cluttered plots)
    # We'll choose a subset based on importance or interest
    if len(numerical_vars) > 6:
        key_vars = ['Age', 'Ancienneté agricole', 'H travail / jour', 'J travail / Sem', 'Poids', 'Taille']
        key_vars = [var for var in key_vars if var in numerical_vars][:5]  # Limit to 5 variables
        print(f"Selected key variables for pairplot: {key_vars}")
    else:
        key_vars = numerical_vars

    # Create pairplot
    plt.figure(figsize=(12, 10))

    try:
        pair_plot = sns.pairplot(
            data[key_vars],
            diag_kind='kde',
            plot_kws={'alpha': 0.6, 's': 50, 'edgecolor': 'k'},
            height=2.5
        )

        pair_plot.fig.suptitle('Relationships Between Key Numerical Variables', y=1.02, fontsize=16)

        # Save pairplot
        pairplot_path = os.path.join(output_dir, "key_variables_pairplot.png")
        pair_plot.savefig(pairplot_path, dpi=300)
        print(f"Pairplot saved to {pairplot_path}")

    except Exception as e:
        print(f"Error creating pairplot: {e}")

def create_age_analysis(data):
    """
    Create specific analysis for the Age variable, including age distribution and age groups
    """
    if 'Age' not in data.columns:
```

```
        print("Age variable not found in dataset. Skipping age analysis.")
        return

    print("\nPerforming detailed analysis of Age variable...")

    # Create age groups if not already present
    if 'Age_Group' not in data.columns:
        data['Age_Group'] = pd.cut(
            data['Age'],
            bins=[0, 30, 40, 50, 60, 100],
            labels=['<30', '30-40', '40-50', '50-60', '>60']
        )

    # Create figure for age distribution
    plt.figure(figsize=(14, 8))

    # First subplot: Age histogram with age group colors
    plt.subplot(1, 2, 1)

    # Get colors from a colormap
    cmap = plt.cm.viridis
    colors = cmap(np.linspace(0, 1, len(data['Age_Group'].cat.categories)))

    # Create a dictionary mapping categories to colors
    color_dict = {category: color for category, color in zip(data['Age_Group'].cat.categories, colors)}

    # Create the base histogram
    sns.histplot(data['Age'], bins=20, kde=True, alpha=0.3)

    # Add colored regions for each age group
    for i, (category, group_data) in enumerate(data.groupby('Age_Group')):
        if not group_data.empty:
            min_val = group_data['Age'].min()
            max_val = group_data['Age'].max()
            plt.axvspan(min_val, max_val, alpha=0.2, color=color_dict[category], label=category)

    plt.title('Age Distribution with Age Groups', fontsize=14)
    plt.xlabel('Age (years)', fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.legend(title='Age Group')
    plt.grid(True, linestyle='--', alpha=0.7)
```

```
# Second subplot: Age group counts
plt.subplot(1, 2, 2)

age_counts = data['Age_Group'].value_counts().sort_index()
bars = plt.bar(age_counts.index, age_counts.values, alpha=0.7, color=colors)

plt.title('Count by Age Group', fontsize=14)
plt.xlabel('Age Group', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.grid(True, axis='y', linestyle='--', alpha=0.7)

# Add count and percentage labels
total = len(data)
for i, (bar, count) in enumerate(zip(bars, age_counts)):
    percentage = count / total * 100
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.5,
        f'{count}\n({percentage:.1f}%)',
        ha='center',
        va='bottom',
        fontsize=10
    )

plt.tight_layout()

# Save the figure
age_analysis_path = os.path.join(output_dir, "age_analysis.png")
plt.savefig(age_analysis_path, dpi=300)
plt.close()
print(f"Age analysis saved to {age_analysis_path}")

# Analyze numerical variables by age group
print("\nAnalyzing key variables by age group...")

# Select key variables to analyze by age
key_vars_by_age = ['H travail / jour', 'J travail / Sem', 'Ancienneté agricole']
key_vars_by_age = [var for var in key_vars_by_age if var in data.columns]

if key_vars_by_age:
    # Create a figure
    fig, axes = plt.subplots(len(key_vars_by_age), 1, figsize=(12, 5 * len(key_vars_by_age)))
```

```
    # Handle the case of a single variable (axes would not be an array)
    if len(key_vars_by_age) == 1:
        axes = [axes]

    # Analyze each variable by age group
    for i, var in enumerate(key_vars_by_age):
        print(f"  Analyzing {var} by age group")

        # Calculate statistics by age group
        age_group_stats = data.groupby('Age_Group')[var].agg(['mean', 'median', 'std', 'count']).reset_index()

        # Create the boxplot
        sns.boxplot(x='Age_Group', y=var, data=data, ax=axes[i])

        # Add jittered points for better visualization
        sns.stripplot(
            x='Age_Group',
            y=var,
            data=data,
            color='black',
            alpha=0.4,
            size=4,
            jitter=True,
            ax=axes[i]
        )

        # Set titles and labels
        axes[i].set_title(f'{var} by Age Group', fontsize=14)
        axes[i].set_xlabel('Age Group', fontsize=12)
        axes[i].set_ylabel(var, fontsize=12)
        axes[i].grid(True, linestyle='--', alpha=0.7)

        # Add mean values as text
        for j, row in enumerate(age_group_stats.iterrows()):
            axes[i].text(
                j,
                row[1]['mean'],
                f"Mean: {row[1]['mean']:.1f}\nN: {row[1]['count']}",
                ha='center',
                va='bottom',
                fontsize=9,
```

```
                    bbox=dict(facecolor='white', alpha=0.8)
                )

        plt.tight_layout()

        # Save the figure
        vars_by_age_path = os.path.join(output_dir, "variables_by_age_group.png")
        plt.savefig(vars_by_age_path, dpi=300)
        plt.close()
        print(f"Analysis of variables by age group saved to {vars_by_age_path}")

        # Save statistics to CSV
        for var in key_vars_by_age:
            stats_by_age = data.groupby('Age_Group')[var].agg(['mean', 'median', 'min', 'max', 'std', 'count']).reset
            stats_by_age_path = os.path.join(output_dir, f"{var}_by_age_group.csv")
            stats_by_age.to_csv(stats_by_age_path, index=False)
            print(f"Statistics for {var} by age group saved to {stats_by_age_path}")

def create_bmi_analysis(data):
    """
    Create BMI analysis if weight and height data are available
    """
    if not all(var in data.columns for var in ['Poids', 'Taille']):
        print("Weight or height variables not found in dataset. Skipping BMI analysis.")
        return

    print("\nPerforming BMI analysis...")

    # Calculate BMI if not already calculated
    if 'BMI' not in data.columns:
        data['BMI'] = data['Poids'] / ((data['Taille']/100) ** 2)

    # Create BMI categories if not already present
    if 'BMI_Category' not in data.columns:
        data['BMI_Category'] = pd.cut(
            data['BMI'],
            bins=[0, 18.5, 25, 30, 100],
            labels=['Underweight', 'Normal', 'Overweight', 'Obese']
        )

    # Calculate BMI statistics
    bmi_stats = data['BMI'].describe()
```

```
print(f"BMI statistics:\n{bmi_stats}")

# Count BMI categories
bmi_category_counts = data['BMI_Category'].value_counts().sort_index()
bmi_category_percent = data['BMI_Category'].value_counts(normalize=True).sort_index() * 100

print("\nBMI category distribution:")
for category, count in bmi_category_counts.items():
    print(f"  {category}: {count} ({bmi_category_percent[category]:.1f}%)")

# Create BMI distribution visualization
plt.figure(figsize=(14, 8))

# First subplot: BMI histogram with categories
plt.subplot(1, 2, 1)

# Plot histogram
sns.histplot(data['BMI'].dropna(), bins=20, kde=True)

# Add category regions
categories = [(0, 18.5, 'Underweight', 'blue'),
              (18.5, 25, 'Normal', 'green'),
              (25, 30, 'Overweight', 'orange'),
              (30, 50, 'Obese', 'red')]

for start, end, label, color in categories:
    plt.axvspan(start, end, alpha=0.2, color=color, label=label)

# Add mean and median lines
plt.axvline(data['BMI'].mean(), color='black', linestyle='--', label=f'Mean: {data["BMI"].mean():.1f}')
plt.axvline(data['BMI'].median(), color='black', linestyle=':', label=f'Median: {data["BMI"].median():.1f}')

plt.title('BMI Distribution with Categories', fontsize=14)
plt.xlabel('BMI (kg/m²)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.legend(title='BMI Category')
plt.grid(True, linestyle='--', alpha=0.7)

# Second subplot: BMI category counts
plt.subplot(1, 2, 2)

# Plot bar chart
```

```
colors = ['blue', 'green', 'orange', 'red']
bars = plt.bar(bmi_category_counts.index, bmi_category_counts.values, alpha=0.7, color=colors)

plt.title('Count by BMI Category', fontsize=14)
plt.xlabel('BMI Category', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.grid(True, axis='y', linestyle='--', alpha=0.7)

# Add count and percentage labels
total = len(data.dropna(subset=['BMI']))
for i, (bar, count) in enumerate(zip(bars, bmi_category_counts)):
    percentage = count / total * 100
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.5,
        f'{count}\n({percentage:.1f}%)',
        ha='center',
        va='bottom',
        fontsize=10
    )

plt.tight_layout()

# Save the figure
bmi_analysis_path = os.path.join(output_dir, "bmi_analysis.png")
plt.savefig(bmi_analysis_path, dpi=300)
plt.close()
print(f"BMI analysis saved to {bmi_analysis_path}")

# Save BMI statistics to CSV
bmi_stats_df = pd.DataFrame({
    'Statistic': bmi_stats.index,
    'Value': bmi_stats.values
})
bmi_stats_path = os.path.join(output_dir, "bmi_statistics.csv")
bmi_stats_df.to_csv(bmi_stats_path, index=False)
print(f"BMI statistics saved to {bmi_stats_path}")

# Save BMI category counts to CSV
bmi_category_df = pd.DataFrame({
    'BMI_Category': bmi_category_counts.index,
    'Count': bmi_category_counts.values,
```

```
        'Percentage': bmi_category_percent.values
    })
    bmi_category_path = os.path.join(output_dir, "bmi_category_counts.csv")
    bmi_category_df.to_csv(bmi_category_path, index=False)
    print(f"BMI category counts saved to {bmi_category_path}")

def analyze_work_patterns(data):
    """
    Analyze work hours, patterns, and experience
    Includes analysis of daily hours, weekly days, total weekly hours,
    and years of agricultural experience
    """
    work_vars = ['H travail / jour', 'J travail / Sem', 'Ancienneté agricole']
    available_work_vars = [var for var in work_vars if var in data.columns]

    if not available_work_vars:
        print("Work-related variables not found in dataset. Skipping work pattern analysis.")
        return

    print("\nAnalyzing work patterns...")

    # Calculate work volume (hours per week) if both variables are available
    if all(var in data.columns for var in ['H travail / jour', 'J travail / Sem']):
        data['Hours_Per_Week'] = data['H travail / jour'] * data['J travail / Sem']

        print(f"Created Hours_Per_Week variable. Summary:")
        hours_per_week_stats = data['Hours_Per_Week'].describe()
        print(hours_per_week_stats)

        # Save work volume statistics
        hours_per_week_stats_df = pd.DataFrame({
            'Statistic': hours_per_week_stats.index,
            'Value': hours_per_week_stats.values
        })
        hours_path = os.path.join(output_dir, "hours_per_week_statistics.csv")
        hours_per_week_stats_df.to_csv(hours_path, index=False)
        print(f"Hours per week statistics saved to {hours_path}")

        # Create visualization for hours per week
        plt.figure(figsize=(10, 6))

        sns.histplot(data['Hours_Per_Week'].dropna(), bins=20, kde=True)
```

```
    # Add reference lines for standard work weeks
    plt.axvline(40, color='red', linestyle='--', label='40 hours (standard work week)')
    plt.axvline(data['Hours_Per_Week'].mean(), color='blue', linestyle='-', label=f'Mean: {data["Hours_Per_Week"]

    plt.title('Distribution of Weekly Work Hours', fontsize=14)
    plt.xlabel('Hours per Week', fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)

    # Add text box with statistics
    stats_text = (
        f"Mean: {data['Hours_Per_Week'].mean():.1f} hours\n"
        f"Median: {data['Hours_Per_Week'].median():.1f} hours\n"
        f"Min: {data['Hours_Per_Week'].min():.1f} hours\n"
        f"Max: {data['Hours_Per_Week'].max():.1f} hours\n"
        f"Std Dev: {data['Hours_Per_Week'].std():.1f} hours\n"
        f"% Working >40h: {(data['Hours_Per_Week'] > 40).mean()*100:.1f}%\n"
        f"% Working >50h: {(data['Hours_Per_Week'] > 50).mean()*100:.1f}%"
    )

    props = dict(boxstyle='round', facecolor='white', alpha=0.8)
    plt.text(0.05, 0.95, stats_text, transform=plt.gca().transAxes, fontsize=10,
           verticalalignment='top', bbox=props)

    plt.tight_layout()

    # Save the figure
    hours_per_week_path = os.path.join(output_dir, "hours_per_week_distribution.png")
    plt.savefig(hours_per_week_path, dpi=300)
    plt.close()
    print(f"Hours per week distribution saved to {hours_per_week_path}")

# Analyze work experience (Ancienneté agricole)
if 'Ancienneté agricole' in data.columns:
    print("\nAnalyzing agricultural work experience...")

    # Create experience categories if not already present
    if 'Experience_Category' not in data.columns:
        data['Experience_Category'] = pd.cut(
            data['Ancienneté agricole'],
```

```
            bins=[0, 5, 10, 20, 30, 100],
            labels=['<5 years', '5-10 years', '10-20 years', '20-30 years', '>30 years']
        )

    # Calculate experience statistics
    exp_stats = data['Ancienneté agricole'].describe()
    print(f"Experience statistics:\n{exp_stats}")

    # Count experience categories
    exp_category_counts = data['Experience_Category'].value_counts().sort_index()
    exp_category_percent = data['Experience_Category'].value_counts(normalize=True).sort_index() * 100

    print("\nExperience category distribution:")
    for category, count in exp_category_counts.items():
        print(f"  {category}: {count} ({exp_category_percent[category]:.1f}%)")

    # Create experience distribution visualization
    plt.figure(figsize=(14, 6))

    # Plot histogram
    sns.histplot(data['Ancienneté agricole'].dropna(), bins=20, kde=True)

    # Add mean and median lines
    plt.axvline(data['Ancienneté agricole'].mean(), color='red', linestyle='--',
                label=f'Mean: {data["Ancienneté agricole"].mean():.1f} years')
    plt.axvline(data['Ancienneté agricole'].median(), color='green', linestyle='-.',
                label=f'Median: {data["Ancienneté agricole"].median():.1f} years')

    plt.title('Distribution of Agricultural Work Experience', fontsize=14)
    plt.xlabel('Years of Experience', fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)

    # Add text box with statistics
    stats_text = (
        f"Mean: {data['Ancienneté agricole'].mean():.1f} years\n"
        f"Median: {data['Ancienneté agricole'].median():.1f} years\n"
        f"Min: {data['Ancienneté agricole'].min():.1f} years\n"
        f"Max: {data['Ancienneté agricole'].max():.1f} years\n"
        f"Std Dev: {data['Ancienneté agricole'].std():.1f} years\n"
        f"% with >10 years: {(data['Ancienneté agricole'] > 10).mean()*100:.1f}%\n"
```

```
        f"% with >20 years: {(data['Ancienneté agricole'] > 20).mean()*100:.1f}%"
    )

    props = dict(boxstyle='round', facecolor='white', alpha=0.8)
    plt.text(0.05, 0.95, stats_text, transform=plt.gca().transAxes, fontsize=10,
           verticalalignment='top', bbox=props)

    plt.tight_layout()

    # Save the figure
    exp_path = os.path.join(output_dir, "experience_distribution.png")
    plt.savefig(exp_path, dpi=300)
    plt.close()
    print(f"Experience distribution saved to {exp_path}")

    # Save experience statistics to CSV
    exp_stats_df = pd.DataFrame({
        'Statistic': exp_stats.index,
        'Value': exp_stats.values
    })
    exp_stats_path = os.path.join(output_dir, "experience_statistics.csv")
    exp_stats_df.to_csv(exp_stats_path, index=False)
    print(f"Experience statistics saved to {exp_stats_path}")

    # Plot experience categories
    plt.figure(figsize=(10, 6))

    # Create a colormap
    cmap = plt.cm.viridis
    colors = cmap(np.linspace(0, 1, len(exp_category_counts)))

    # Plot bar chart
    bars = plt.bar(exp_category_counts.index, exp_category_counts.values, alpha=0.7, color=colors)

    plt.title('Distribution by Years of Agricultural Experience', fontsize=14)
    plt.xlabel('Experience Category', fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.grid(True, axis='y', linestyle='--', alpha=0.7)

    # Add count and percentage labels
    total = len(data.dropna(subset=['Ancienneté agricole']))
    for i, (bar, count) in enumerate(zip(bars, exp_category_counts)):
```

```
            percentage = count / total * 100
            plt.text(
                bar.get_x() + bar.get_width() / 2,
                bar.get_height() + 0.5,
                f'{count}\n({percentage:.1f}%)',
                ha='center',
                va='bottom',
                fontsize=10
            )

        plt.tight_layout()

        # Save the figure
        exp_cat_path = os.path.join(output_dir, "experience_categories.png")
        plt.savefig(exp_cat_path, dpi=300)
        plt.close()
        print(f"Experience categories chart saved to {exp_cat_path}")

        # Analyze relationship between Age and Experience
        if 'Age' in data.columns:
            print("\nAnalyzing relationship between Age and Work Experience...")

            plt.figure(figsize=(10, 6))

            # Create scatter plot with regression line
            sns.regplot(
                x='Age',
                y='Ancienneté agricole',
                data=data,
                scatter_kws={'alpha': 0.6, 's': 50, 'edgecolor': 'k'},
                line_kws={'color': 'red'}
            )

            # Calculate Pearson correlation
            corr, p_value = stats.pearsonr(
                data['Age'].dropna(),
                data['Ancienneté agricole'].dropna()
            )

            # Add correlation text
            plt.annotate(
                f"Correlation: {corr:.2f}\np-value: {p_value:.4f}",
```

```
                xy=(0.05, 0.95),
                xycoords='axes fraction',
                backgroundcolor='white',
                fontsize=10,
                va='top'
            )

            plt.title('Relationship between Age and Agricultural Experience', fontsize=14)
            plt.xlabel('Age (years)', fontsize=12)
            plt.ylabel('Agricultural Experience (years)', fontsize=12)
            plt.grid(True, linestyle='--', alpha=0.7)

            # Add reference line for 1:1 relationship (experience = age - 18, assuming start at 18)
            x = np.array([data['Age'].min(), data['Age'].max()])
            y = x - 18
            plt.plot(x, y, 'k--', alpha=0.3, label='If started at age 18')
            plt.legend()

            plt.tight_layout()

            # Save the figure
            age_exp_path = os.path.join(output_dir, "age_vs_experience.png")
            plt.savefig(age_exp_path, dpi=300)
            plt.close()
            print(f"Age vs Experience relationship saved to {age_exp_path}")
```

In [ ]:

```
def main():
    # Load data
    data = load_data()

    # Identify numerical variables
    numerical_vars = identify_numerical_variables(data)

    # Calculate summary statistics
    summary = calculate_summary_statistics(data, numerical_vars)

    # Create distribution plots
    create_distribution_plots(data, numerical_vars)

    # Analyze normality
    normality_results = analyze_normality(data, numerical_vars)
```

```
    # Create correlation matrix
    create_correlation_matrix(data, numerical_vars)

    # Create pairplots
    create_pairplots(data, numerical_vars)

    # Age analysis
    create_age_analysis(data)

    # BMI analysis
    create_bmi_analysis(data)

    # Work patterns analysis
    analyze_work_patterns(data)

    print(f"\nAll numerical analysis results saved to {output_dir}")

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

In [ ]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Step 2.2: Categorical Variables Analysis
This script analyzes the categorical variables in the female farmers dataset.
It produces frequency distributions, visualizations, and cross-tabulations
to understand patterns and relationships in categorical data.

Author: [Your Name]
Date: April 1, 2025
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import os
from collections import Counter
import matplotlib.ticker as mtick
import logging
```

```
# Set up logging
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# Set style for plots
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette('viridis')

# Create output directory for results
output_dir = 'results/categorical_analysis'
os.makedirs(output_dir, exist_ok=True)

def load_data():
    """Load the preprocessed dataset"""
    logger.info("Loading the dataset...")
    data = pd.read_excel('fixed_female_farmers_data.xlsx')
    logger.info(f"Dataset loaded with {data.shape[0]} rows and {data.shape[1]} columns")

    # Log columns and data types for debugging
    logger.info(f"Columns: {list(data.columns)}")
    logger.info(f"Data types:\n {data.dtypes}")

    return data

def identify_categorical_variables(data):
    """Identify categorical variables in the dataset"""
    # Define expected categorical variables based on the codebook
    expected_categorical = [
        'Situation maritale', 'Domicile', 'Niveau socio-économique', 'Niveau scolaire',
        'Statut', 'Tabagisme', 'Neffa', 'Fumées de Tabouna', 'Ménopause',
        'Masque pour pesticides', 'Bottes', 'Gants', 'Casquette/Mdhalla',
        'Manteau imperméable', 'Moyen de transport'
    ]

    # Filter to include only columns that exist in the dataset
    categorical_vars = [var for var in expected_categorical if var in data.columns]

    # Add columns that are object type but not in expected list
    for col in data.columns:
        if col not in categorical_vars and data[col].dtype == 'object':
```

```
            # Skip some free-text columns that aren't suitable for categorical analysis
            skip_columns = ['Nom', 'Prénom', 'N° du téléphone', 'Examen cardiovasculaire et pulmonaire',
                            'Examen des membres supérieurs', 'Examen du rachis', 'Examen visuel',
                            'Spirométrie', 'Interprétation Spiro']
            if col not in skip_columns:
                categorical_vars.append(col)

    # Also add any columns that have been converted to category type
    for col in data.columns:
        if col not in categorical_vars and pd.api.types.is_categorical_dtype(data[col]):
            categorical_vars.append(col)

    logger.info(f"Identified {len(categorical_vars)} categorical variables: {', '.join(categorical_vars)}")
    return categorical_vars

def calculate_frequency_distributions(data, categorical_vars):
    """
    Calculate and save frequency distributions for categorical variables
    """
    logger.info("Calculating frequency distributions...")

    # Dictionary to store all frequency distributions
    all_frequencies = {}

    # Create a summary dataframe for all categorical variables
    summary_data = []

    for var in categorical_vars:
        logger.info(f"  Analyzing {var}")

        # Get value counts and percentages
        value_counts = data[var].value_counts()
        value_percentages = data[var].value_counts(normalize=True) * 100

        # Combine counts and percentages
        freq_df = pd.DataFrame({
            'Count': value_counts,
            'Percentage': value_percentages
        })

        # Add missing count
        missing_count = data[var].isnull().sum()
```

```
        missing_percent = (missing_count / len(data)) * 100

        # Add to summary
        summary_data.append({
            'Variable': var,
            'Unique_Values': len(value_counts),
            'Most_Common': value_counts.index[0] if not value_counts.empty else 'N/A',
            'Most_Common_Count': value_counts.iloc[0] if not value_counts.empty else 0,
            'Most_Common_Pct': value_percentages.iloc[0] if not value_counts.empty else 0,
            'Missing_Count': missing_count,
            'Missing_Percentage': missing_percent
        })

        # Store frequencies
        all_frequencies[var] = freq_df

        # Save individual frequency distribution to CSV
        freq_path = os.path.join(output_dir, f"{var.replace('/', '_')}_frequencies.csv")
        freq_df.to_csv(freq_path)
        logger.info(f"    Saved frequency distribution to {freq_path}")

    # Create and save summary dataframe
    summary_df = pd.DataFrame(summary_data)
    summary_path = os.path.join(output_dir, "categorical_summary.csv")
    summary_df.to_csv(summary_path, index=False)
    logger.info(f"Categorical variable summary saved to {summary_path}")

    return all_frequencies, summary_df

def create_bar_charts(data, categorical_vars, frequencies):
    """
    Create bar charts for categorical variables
    """
    logger.info("Creating bar charts for categorical variables...")

    for var in categorical_vars:
        logger.info(f"  Creating bar chart for {var}")

        # Get frequency data for this variable
        freq_df = frequencies[var]

        # Sort by count (optional)
```

```
        freq_df = freq_df.sort_values('Count', ascending=False)

        # Create figure
        plt.figure(figsize=(12, 7))

        # Create bar plot
        ax = sns.barplot(x=freq_df.index, y='Count', data=freq_df)

        # Add percentage text on top of bars
        for i, p in enumerate(ax.patches):
            percentage = freq_df['Percentage'].iloc[i] if i < len(freq_df) else 0
            ax.annotate(f'{percentage:.1f}%',
                        (p.get_x() + p.get_width() / 2., p.get_height()),
                        ha = 'center', va = 'bottom',
                        xytext = (0, 5), textcoords = 'offset points')

        # Rotate x-axis labels if there are many categories
        if len(freq_df) > 3:
            plt.xticks(rotation=45, ha='right')

        # Set title and labels
        plt.title(f'Distribution of {var}', fontsize=14)
        plt.xlabel(var, fontsize=12)
        plt.ylabel('Count', fontsize=12)

        # Add text box with statistics
        stats_text = (
            f"Total: {freq_df['Count'].sum()}\n"
            f"Unique values: {len(freq_df)}\n"
            f"Most common: {freq_df.index[0]} ({freq_df['Percentage'].iloc[0]:.1f}%)\n"
            f"Missing: {data[var].isnull().sum()} ({data[var].isnull().sum()/len(data)*100:.1f}%)"
        )

        props = dict(boxstyle='round', facecolor='white', alpha=0.8)
        plt.text(0.05, 0.95, stats_text, transform=plt.gca().transAxes, fontsize=10,
                verticalalignment='top', bbox=props)

        plt.tight_layout()

        # Save the figure
        bar_path = os.path.join(output_dir, f"{var.replace('/', '_')}_barchart.png")
        plt.savefig(bar_path, dpi=300)
```

```
        plt.close()
        logger.info(f"    Bar chart saved to {bar_path}")

def create_pie_charts(data, categorical_vars, frequencies):
    """
    Create pie charts for categorical variables with fewer categories
    """
    logger.info("Creating pie charts for categorical variables with fewer categories...")

    for var in categorical_vars:
        # Skip variables with too many categories
        if frequencies[var].shape[0] > 7:
            logger.info(f"  Skipping pie chart for {var} (too many categories: {frequencies[var].shape[0]})")
            continue

        logger.info(f"  Creating pie chart for {var}")

        # Get frequency data for this variable
        freq_df = frequencies[var]

        # Create figure
        plt.figure(figsize=(10, 8))

        # Create pie chart
        plt.pie(freq_df['Count'],
                labels=freq_df.index,
                autopct='%1.1f%%',
                startangle=90,
                shadow=False,
                explode=[0.05] * len(freq_df),  # Slight separation for all slices
                textprops={'fontsize': 12})

        # Equal aspect ratio ensures that pie is drawn as a circle
        plt.axis('equal')

        # Set title
        plt.title(f'Distribution of {var}', fontsize=16)

        # Add legend if there are more than 5 categories
        if len(freq_df) > 5:
            plt.legend(title=var, loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))
```

```
        plt.tight_layout()

        # Save the figure
        pie_path = os.path.join(output_dir, f"{var.replace('/', '_')}_piechart.png")
        plt.savefig(pie_path, dpi=300)
        plt.close()
        logger.info(f"    Pie chart saved to {pie_path}")

def analyze_protection_equipment(data):
    """
    Analyze protection equipment usage patterns
    """
    protection_vars = ['Masque pour pesticides', 'Bottes', 'Gants', 'Casquette/Mdhalla', 'Manteau imperméable']

    # Check if protection variables exist
    available_protection_vars = [var for var in protection_vars if var in data.columns]

    if not available_protection_vars:
        logger.info("Protection equipment variables not found. Skipping protection equipment analysis.")
        return

    logger.info("Analyzing protection equipment usage patterns...")

    # Create figure for combined visualization
    plt.figure(figsize=(14, 10))

    # Dictionary to store frequency data
    protection_data = {}
    categories = []

    # Process each protection variable
    for i, var in enumerate(available_protection_vars):
        logger.info(f"  Processing {var}")

        # Get value counts and percentages
        value_counts = data[var].value_counts().sort_index()
        categories = list(value_counts.index)
        protection_data[var] = value_counts

    # Create DataFrame for plotting
    if categories:
        plot_data = pd.DataFrame({var: protection_data[var] for var in available_protection_vars})
```

```
        plot_data = plot_data.fillna(0)

        # Calculate percentage use
        plot_data_pct = plot_data / plot_data.sum() * 100

        # Create stacked bar chart
        ax = plot_data_pct.plot(kind='bar', stacked=False, figsize=(14, 8), width=0.7)

        # Add value labels on the bars
        for c in ax.containers:
            labels = [f'{v:.1f}%' if v > 0 else '' for v in c.datavalues]
            ax.bar_label(c, labels=labels, label_type='center')

        # Set title and labels
        plt.title('Protection Equipment Usage Patterns', fontsize=16)
        plt.xlabel('Usage Frequency', fontsize=14)
        plt.ylabel('Percentage of Workers', fontsize=14)
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.legend(title='Protection Type')

        # Format y-axis as percentage
        ax.yaxis.set_major_formatter(mtick.PercentFormatter())

        plt.tight_layout()

        # Save the figure
        protection_path = os.path.join(output_dir, "protection_equipment_usage.png")
        plt.savefig(protection_path, dpi=300)
        plt.close()
        logger.info(f"Protection equipment usage chart saved to {protection_path}")

        # Calculate and save overall protection score
        protection_score = {}

        # Define scoring system (example: jamais=0, parfois=1, souvent=2, toujours=3)
        score_mapping = {'jamais': 0, 'parfois': 1, 'souvent': 2, 'toujours': 3}

        # Calculate score for each worker
        data['protection_score'] = 0
        for var in available_protection_vars:
            # Convert categories to scores
            data[f'{var}_score'] = data[var].map(score_mapping)
```

```
            data['protection_score'] += data[f'{var}_score']

        # Normalize score (0-100%)
        max_possible_score = len(available_protection_vars) * 3  # 3 is the max score for 'toujours'
        data['protection_score_pct'] = (data['protection_score'] / max_possible_score) * 100

        # Create protection score categories
        data['protection_level'] = pd.cut(
            data['protection_score_pct'],
            bins=[0, 25, 50, 75, 100],
            labels=['Poor', 'Basic', 'Good', 'Excellent']
        )

        # Plot protection score distribution
        plt.figure(figsize=(12, 8))

        # Histogram of protection scores
        sns.histplot(data['protection_score_pct'], bins=20, kde=True)

        # Add mean and median lines
        plt.axvline(data['protection_score_pct'].mean(), color='red', linestyle='--',
                    label=f'Mean: {data["protection_score_pct"].mean():.1f}%')
        plt.axvline(data['protection_score_pct'].median(), color='green', linestyle='-.',
                    label=f'Median: {data["protection_score_pct"].median():.1f}%')

        plt.title('Distribution of Protection Equipment Usage Scores', fontsize=14)
        plt.xlabel('Protection Score (%)', fontsize=12)
        plt.ylabel('Frequency', fontsize=12)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.legend()

        # Add text box with statistics
        stats_text = (
            f"Mean: {data['protection_score_pct'].mean():.1f}%\n"
            f"Median: {data['protection_score_pct'].median():.1f}%\n"
            f"Min: {data['protection_score_pct'].min():.1f}%\n"
            f"Max: {data['protection_score_pct'].max():.1f}%\n"
            f"Std Dev: {data['protection_score_pct'].std():.1f}%"
        )

        props = dict(boxstyle='round', facecolor='white', alpha=0.8)
        plt.text(0.05, 0.95, stats_text, transform=plt.gca().transAxes, fontsize=10,
```

```
            verticalalignment='top', bbox=props)

    plt.tight_layout()

    # Save the figure
    score_path = os.path.join(output_dir, "protection_score_distribution.png")
    plt.savefig(score_path, dpi=300)
    plt.close()
    logger.info(f"Protection score distribution saved to {score_path}")

    # Create bar chart of protection levels
    plt.figure(figsize=(10, 6))

    level_counts = data['protection_level'].value_counts().sort_index()
    level_percentages = data['protection_level'].value_counts(normalize=True).sort_index() * 100

    # Get a colormap based on protection level (poor to excellent)
    cmap = plt.cm.RdYlGn  # Red-Yellow-Green colormap
    colors = cmap(np.linspace(0.15, 0.85, len(level_counts)))

    bars = plt.bar(level_counts.index, level_counts.values, color=colors, alpha=0.7)

    plt.title('Protection Equipment Usage Levels', fontsize=14)
    plt.xlabel('Protection Level', fontsize=12)
    plt.ylabel('Number of Workers', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Add count and percentage labels
    for i, (bar, count) in enumerate(zip(bars, level_counts)):
        percentage = level_percentages.iloc[i]
        plt.text(
            bar.get_x() + bar.get_width() / 2,
            bar.get_height() + 0.5,
            f'{count}\n({percentage:.1f}%)',
            ha='center',
            va='bottom',
            fontsize=10
        )

    plt.tight_layout()

    # Save the figure
```

```
        level_path = os.path.join(output_dir, "protection_levels.png")
        plt.savefig(level_path, dpi=300)
        plt.close()
        logger.info(f"Protection levels chart saved to {level_path}")

        # Save protection score statistics
        protection_stats = data[['protection_score', 'protection_score_pct', 'protection_level']].describe()
        stats_path = os.path.join(output_dir, "protection_score_statistics.csv")
        protection_stats.to_csv(stats_path)
        logger.info(f"Protection score statistics saved to {stats_path}")

        # Save individual protection scores
        score_data = data[['protection_score', 'protection_score_pct', 'protection_level'] + available_protection_var
        score_path = os.path.join(output_dir, "individual_protection_scores.csv")
        score_data.to_csv(score_path, index=False)
        logger.info(f"Individual protection scores saved to {score_path}")

def analyze_health_complaints(data):
    """
    Analyze health complaints recorded in the dataset
    """
    # Define health-related variables based on the codebook
    health_vars = [
        'Troubles cardio-respiratoires', 'Troubles cognitifs',
        'Troubles neurologiques', 'Troubles cutanés/phanères', 'Autres plaintes'
    ]

    # Check if health variables exist
    available_health_vars = [var for var in health_vars if var in data.columns]

    if not available_health_vars:
        logger.info("Health complaint variables not found. Skipping health complaints analysis.")
        return

    logger.info("Analyzing health complaints...")

    # Function to process free-text health complaints
    def extract_complaints(text):
        if pd.isna(text) or text.strip() == '':
            return []

        # Split text by common separators
```

```
        separators = [',', ';', '-', '/']
        complaint_list = [text]

        for sep in separators:
            new_list = []
            for item in complaint_list:
                new_list.extend(item.split(sep))
            complaint_list = new_list

        # Clean and return non-empty complaints
        return [complaint.strip() for complaint in complaint_list if complaint.strip()]

    # Dictionary to store all complaints by category
    all_complaints = {}
    complaint_counts = {}
    total_women_with_complaints = 0

    # Process each health variable
    for var in available_health_vars:
        logger.info(f"  Processing {var}")

        # Count women with any complaint in this category
        women_with_complaint = data[var].notna().sum()
        women_with_complaint_pct = (women_with_complaint / len(data)) * 100
        logger.info(f"    {women_with_complaint} women ({women_with_complaint_pct:.1f}%) reported {var}")

        # Extract individual complaints
        all_complaints[var] = []
        for text in data[var].dropna():
            complaints = extract_complaints(text)
            all_complaints[var].extend(complaints)

        # Count frequencies
        complaint_counts[var] = Counter(all_complaints[var])
        logger.info(f"    Extracted {len(all_complaints[var])} individual complaints, {len(complaint_counts[var])} ur

        # Save complaint frequencies to CSV
        if complaint_counts[var]:
            complaint_df = pd.DataFrame.from_dict(
                complaint_counts[var], orient='index', columns=['Count']
            ).sort_values('Count', ascending=False)
```

```
            complaint_df['Percentage'] = (complaint_df['Count'] / sum(complaint_counts[var].values())) * 100

            complaint_path = os.path.join(output_dir, f"{var.replace('/', '_')}_complaints.csv")
            complaint_df.to_csv(complaint_path)
            logger.info(f"    Complaint frequencies saved to {complaint_path}")

    # Create combined visualization of complaint categories
    plt.figure(figsize=(12, 8))

    # Prepare data for plotting
    categories = []
    counts = []
    for var in available_health_vars:
        categories.append(var)
        counts.append(data[var].notna().sum())

    # Sort by count
    sort_idx = np.argsort(counts)[::-1]  # Descending order
    categories = [categories[i] for i in sort_idx]
    counts = [counts[i] for i in sort_idx]

    # Create bar chart
    bars = plt.bar(categories, counts, color=sns.color_palette('viridis', len(categories)))

    plt.title('Distribution of Health Complaint Categories', fontsize=14)
    plt.xlabel('Complaint Category', fontsize=12)
    plt.ylabel('Number of Women Reporting', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Add count and percentage labels
    for i, (bar, count) in enumerate(zip(bars, counts)):
        percentage = (count / len(data)) * 100
        plt.text(
            bar.get_x() + bar.get_width() / 2,
            bar.get_height() + 0.5,
            f'{count}\n({percentage:.1f}%)',
            ha='center',
            va='bottom',
            fontsize=10
        )

    plt.xticks(rotation=45, ha='right')
```

```
    plt.tight_layout()

    # Save the figure
    category_path = os.path.join(output_dir, "health_complaint_categories.png")
    plt.savefig(category_path, dpi=300)
    plt.close()
    logger.info(f"Health complaint categories chart saved to {category_path}")

    # Create visualizations for top complaints in each category
    for var in available_health_vars:
        if not complaint_counts[var]:
            continue

        # Get top 10 complaints
        top_complaints = dict(sorted(complaint_counts[var].items(), key=lambda x: x[1], reverse=True)[:10])

        if not top_complaints:
            continue

        plt.figure(figsize=(12, 8))

        # Create horizontal bar chart for better readability with long complaint names
        bars = plt.barh(list(top_complaints.keys())[::-1], list(top_complaints.values())[::-1])

        plt.title(f'Top Complaints: {var}', fontsize=14)
        plt.xlabel('Number of Reports', fontsize=12)
        plt.ylabel('Complaint', fontsize=12)
        plt.grid(axis='x', linestyle='--', alpha=0.7)

        # Add count labels
        for bar in bars:
            width = bar.get_width()
            plt.text(
                width + 0.3,
                bar.get_y() + bar.get_height() / 2,
                f'{width}',
                ha='left',
                va='center',
                fontsize=10
            )

        plt.tight_layout()
```

```
    # Save the figure
    top_path = os.path.join(output_dir, f"{var.replace('/', '_')}_top_complaints.png")
    plt.savefig(top_path, dpi=300)
    plt.close()
    logger.info(f"Top complaints chart for {var} saved to {top_path}")

# Count how many women have multiple categories of complaints
complaint_presence = pd.DataFrame({
    var: data[var].notna().astype(int) for var in available_health_vars
})

complaint_presence['total_categories'] = complaint_presence.sum(axis=1)

# Create distribution of number of complaint categories
plt.figure(figsize=(10, 6))

value_counts = complaint_presence['total_categories'].value_counts().sort_index()

bars = plt.bar(value_counts.index, value_counts.values)

plt.title('Distribution of Number of Health Complaint Categories per Woman', fontsize=14)
plt.xlabel('Number of Complaint Categories', fontsize=12)
plt.ylabel('Number of Women', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(range(len(available_health_vars) + 1))

# Add count and percentage labels
for i, (bar, count) in enumerate(zip(bars, value_counts)):
    percentage = (count / len(data)) * 100
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.5,
        f'{count}\n({percentage:.1f}%)',
        ha='center',
        va='bottom',
        fontsize=10
    )

plt.tight_layout()

# Save the figure
```

```
    count_path = os.path.join(output_dir, "complaint_category_counts.png")
    plt.savefig(count_path, dpi=300)
    plt.close()
    logger.info(f"Complaint category counts chart saved to {count_path}")

    # Save the distribution to CSV
    count_df = pd.DataFrame({
        'Number_of_Categories': value_counts.index,
        'Count': value_counts.values,
        'Percentage': (value_counts.values / len(data)) * 100
    })

    count_path = os.path.join(output_dir, "complaint_category_counts.csv")
    count_df.to_csv(count_path, index=False)
    logger.info(f"Complaint category counts saved to {count_path}")

def create_cross_tabulations(data, categorical_vars):
    """
    Create cross-tabulations between key categorical variables
    """
    logger.info("Creating cross-tabulations between key categorical variables...")

    # Define key demographic variables to cross-tabulate against
    key_demographic_vars = [
        'Situation maritale', 'Niveau socio-économique', 'Niveau scolaire', 'Statut'
    ]

    # Filter to include only existing variables
    demo_vars = [var for var in key_demographic_vars if var in categorical_vars]

    if not demo_vars:
        logger.info("No key demographic variables found for cross-tabulation.")
        return

    # Create directory for cross-tabulations
    crosstab_dir = os.path.join(output_dir, "crosstabs")
    os.makedirs(crosstab_dir, exist_ok=True)

    # Process health habits
    health_habit_vars = ['Tabagisme', 'Neffa', 'Fumées de Tabouna']
    health_habit_vars = [var for var in health_habit_vars if var in categorical_vars]
```

```
    # Process protection variables
    protection_vars = ['Masque pour pesticides', 'Bottes', 'Gants', 'Casquette/Mdhalla', 'Manteau imperméable']
    protection_vars = [var for var in protection_vars if var in categorical_vars]

    # For each demographic variable, cross-tabulate with other key variables
    for demo_var in demo_vars:
        logger.info(f"  Creating cross-tabulations for {demo_var}")

        # Cross-tabulate with health habits
        for habit_var in health_habit_vars:
            logger.info(f"    Cross-tabulating {demo_var} with {habit_var}")

            # Create cross-tabulation
            cross_tab = pd.crosstab(
                data[demo_var],
                data[habit_var],
                normalize='index'
            ) * 100  # Convert to percentages

            # Save cross-tabulation to CSV
            cross_path = os.path.join(crosstab_dir, f"{demo_var.replace('/', '_')}_{habit_var.replace('/', '_')}_cros
            cross_tab.to_csv(cross_path)

            # Create stacked bar chart
            plt.figure(figsize=(12, 8))

            cross_tab.plot(kind='bar', stacked=True, colormap='viridis')

            plt.title(f'{habit_var} by {demo_var}', fontsize=14)
            plt.xlabel(demo_var, fontsize=12)
            plt.ylabel('Percentage', fontsize=12)
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title=habit_var)
            plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

            # If x-axis has many categories, rotate labels
            if len(cross_tab) > 3:
                plt.xticks(rotation=45, ha='right')

            plt.tight_layout()

            # Save the figure
```

```
            fig_path = os.path.join(crosstab_dir, f"{demo_var.replace('/', '_')}_{habit_var.replace('/', '_')}_chart
            plt.savefig(fig_path, dpi=300)
            plt.close()
            logger.info(f"      Saved to {fig_path}")

        # Cross-tabulate with protection equipment (just one for simplicity)
        if protection_vars:
            # Use protection level if available, otherwise use first protection variable
            if 'protection_level' in data.columns:
                protection_var = 'protection_level'
            else:
                protection_var = protection_vars[0]

            logger.info(f"    Cross-tabulating {demo_var} with {protection_var}")

            # Create cross-tabulation
            cross_tab = pd.crosstab(
                data[demo_var],
                data[protection_var],
                normalize='index'
            ) * 100  # Convert to percentages

            # Save cross-tabulation to CSV
            cross_path = os.path.join(crosstab_dir, f"{demo_var.replace('/', '_')}_{protection_var.replace('/', '_')}
            cross_tab.to_csv(cross_path)

            # Create stacked bar chart
            plt.figure(figsize=(12, 8))

            cross_tab.plot(kind='bar', stacked=True, colormap='viridis')

            plt.title(f'{protection_var} by {demo_var}', fontsize=14)
            plt.xlabel(demo_var, fontsize=12)
            plt.ylabel('Percentage', fontsize=12)
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.legend(title=protection_var)
            plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

            # If x-axis has many categories, rotate labels
            if len(cross_tab) > 3:
                plt.xticks(rotation=45, ha='right')
```

```
            plt.tight_layout()

            # Save the figure
            fig_path = os.path.join(crosstab_dir, f"{demo_var.replace('/', '_')}_{protection_var.replace('/', '_')}_
            plt.savefig(fig_path, dpi=300)
            plt.close()
            logger.info(f"      Saved to {fig_path}")

def analyze_educational_socioeconomic_effects(data):
    """
    Analyze effects of education level and socioeconomic status on various outcomes
    """
    # Check if the necessary variables exist
    if 'Niveau scolaire' not in data.columns or 'Niveau socio-économique' not in data.columns:
        logger.info("Education or socioeconomic status variables not found. Skipping analysis.")
        return

    logger.info("Analyzing effects of education level and socioeconomic status...")

    # Create directory for education/socioeconomic analysis
    edu_dir = os.path.join(output_dir, "education_socioeconomic")
    os.makedirs(edu_dir, exist_ok=True)

    # Analyze relationship between education and socioeconomic status
    logger.info("  Analyzing relationship between education and socioeconomic status")

    # Create cross-tabulation
    edu_socio_cross = pd.crosstab(
        data['Niveau scolaire'],
        data['Niveau socio-économique'],
        normalize='index'
    ) * 100  # Convert to percentages

    # Save cross-tabulation to CSV
    cross_path = os.path.join(edu_dir, "education_socioeconomic_crosstab.csv")
    edu_socio_cross.to_csv(cross_path)
    logger.info(f"    Cross-tabulation saved to {cross_path}")

            # Create bar chart
            plt.figure(figsize=(12, 8))

            # Only plot the "True" column (presence of health complaints)
```

```
            if True in health_marital_cross.columns:
                bars = plt.bar(
                    health_marital_cross.index,
                    health_marital_cross[True],
                    color=sns.color_palette('viridis', len(health_marital_cross))
                )

                plt.title('Percentage with Health Complaints by Marital Status', fontsize=14)
                plt.xlabel('Marital Status', fontsize=12)
                plt.ylabel('Percentage with Health Complaints', fontsize=12)
                plt.grid(axis='y', linestyle='--', alpha=0.7)
                plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

                # Add percentage labels
                for bar in bars:
                    height = bar.get_height()
                    plt.text(
                        bar.get_x() + bar.get_width() / 2,
                        height + 1,
                        f'{height:.1f}%',
                        ha='center',
                        fontsize=10
                    )

                plt.tight_layout()

                # Save the figure
                bar_path = os.path.join(family_dir, "health_complaints_by_marital_status.png")
                plt.savefig(bar_path, dpi=300)
                plt.close()
                logger.info(f"    Bar chart saved to {bar_path}")

def analyze_employment_status(data):
    """
    Analyze differences based on employment status (permanent vs. seasonal)
    """
    if 'Statut' not in data.columns:
        logger.info("Employment status variable not found. Skipping analysis.")
        return

    logger.info("Analyzing differences based on employment status...")
```

```
    # Create directory for employment status analysis
    status_dir = os.path.join(output_dir, "employment_status")
    os.makedirs(status_dir, exist_ok=True)

    # Analyze work hours by employment status
    if 'H travail / jour' in data.columns:
        logger.info("  Analyzing work hours by employment status")

        # Create boxplot
        plt.figure(figsize=(12, 8))

        sns.boxplot(x='Statut', y='H travail / jour', data=data)

        plt.title('Daily Work Hours by Employment Status', fontsize=14)
        plt.xlabel('Employment Status', fontsize=12)
        plt.ylabel('Hours per Day', fontsize=12)
        plt.grid(axis='y', linestyle='--', alpha=0.7)

        # Add mean values as text
        for i, status in enumerate(sorted(data['Statut'].unique())):
            if pd.isna(status):
                continue
            mean_val = data[data['Statut'] == status]['H travail / jour'].mean()
            count = len(data[data['Statut'] == status])
            plt.text(
                i,
                mean_val + 0.2,
                f"Mean: {mean_val:.1f}\nN: {count}",
                ha='center',
                fontsize=9,
                bbox=dict(facecolor='white', alpha=0.8)
            )

        plt.tight_layout()

        # Save the figure
        box_path = os.path.join(status_dir, "work_hours_by_status.png")
        plt.savefig(box_path, dpi=300)
        plt.close()
        logger.info(f"    Boxplot saved to {box_path}")

    # Analyze protection equipment usage by employment status
```

```
if 'protection_score_pct' in data.columns:
    logger.info("  Analyzing protection equipment usage by employment status")

    # Create boxplot
    plt.figure(figsize=(12, 8))

    sns.boxplot(x='Statut', y='protection_score_pct', data=data)

    plt.title('Protection Equipment Usage by Employment Status', fontsize=14)
    plt.xlabel('Employment Status', fontsize=12)
    plt.ylabel('Protection Score (%)', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Add mean values as text
    for i, status in enumerate(sorted(data['Statut'].unique())):
        if pd.isna(status):
            continue
        mean_val = data[data['Statut'] == status]['protection_score_pct'].mean()
        count = len(data[data['Statut'] == status])
        plt.text(
            i,
            mean_val + 2,
            f"Mean: {mean_val:.1f}%\nN: {count}",
            ha='center',
            fontsize=9,
            bbox=dict(facecolor='white', alpha=0.8)
        )

    plt.tight_layout()

    # Save the figure
    box_path = os.path.join(status_dir, "protection_by_status.png")
    plt.savefig(box_path, dpi=300)
    plt.close()
    logger.info(f"    Boxplot saved to {box_path}")

# Analyze health complaints by employment status
health_var_present = False
for var in ['Troubles cardio-respiratoires', 'Troubles cognitifs', 'Troubles neurologiques', 'Troubles cutanés/ph
    if var in data.columns:
        health_var_present = True
        break
```

```
if health_var_present:
    logger.info("  Analyzing health complaints by employment status")

    # Create a variable indicating if any health complaint is present if not already created
    if 'any_health_complaint' not in data.columns:
        data['any_health_complaint'] = False
        for var in ['Troubles cardio-respiratoires', 'Troubles cognitifs', 'Troubles neurologiques', 'Troubles cu
            if var in data.columns:
                data['any_health_complaint'] = data['any_health_complaint'] | data[var].notna()

    # Create cross-tabulation
    health_status_cross = pd.crosstab(
        data['Statut'],
        data['any_health_complaint'],
        normalize='index'
    ) * 100  # Convert to percentages

    # Save cross-tabulation to CSV
    cross_path = os.path.join(status_dir, "health_status_crosstab.csv")
    health_status_cross.to_csv(cross_path)
    logger.info(f"    Cross-tabulation saved to {cross_path}")

    # Create bar chart
    plt.figure(figsize=(12, 8))

    # Only plot the "True" column (presence of health complaints)
    if True in health_status_cross.columns:
        bars = plt.bar(
            health_status_cross.index,
            health_status_cross[True],
            color=sns.color_palette('viridis', len(health_status_cross))
        )

        plt.title('Percentage with Health Complaints by Employment Status', fontsize=14)
        plt.xlabel('Employment Status', fontsize=12)
        plt.ylabel('Percentage with Health Complaints', fontsize=12)
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

        # Add percentage labels
        for bar in bars:
```

```
                height = bar.get_height()
                plt.text(
                    bar.get_x() + bar.get_width() / 2,
                    height + 1,
                    f'{height:.1f}%',
                    ha='center',
                    fontsize=10
                )

            plt.tight_layout()

            # Save the figure
            bar_path = os.path.join(status_dir, "health_complaints_by_status.png")
            plt.savefig(bar_path, dpi=300)
            plt.close()
            logger.info(f"    Bar chart saved to {bar_path}")

    # Analyze age distribution by employment status
    if 'Age' in data.columns:
        logger.info("  Analyzing age distribution by employment status")

        # Create boxplot
        plt.figure(figsize=(12, 8))

        sns.boxplot(x='Statut', y='Age', data=data)

        plt.title('Age Distribution by Employment Status', fontsize=14)
        plt.xlabel('Employment Status', fontsize=12)
        plt.ylabel('Age (years)', fontsize=12)
        plt.grid(axis='y', linestyle='--', alpha=0.7)

        # Add mean values as text
        for i, status in enumerate(sorted(data['Statut'].unique())):
            if pd.isna(status):
                continue
            mean_val = data[data['Statut'] == status]['Age'].mean()
            count = len(data[data['Statut'] == status])
            plt.text(
                i,
                mean_val + 1,
                f"Mean: {mean_val:.1f}\nN: {count}",
                ha='center',
```

```
                fontsize=9,
                bbox=dict(facecolor='white', alpha=0.8)
            )

        plt.tight_layout()

        # Save the figure
        box_path = os.path.join(status_dir, "age_by_status.png")
        plt.savefig(box_path, dpi=300)
        plt.close()
        logger.info(f"    Boxplot saved to {box_path}")

        # Save summary statistics
        status_stats = data.groupby('Statut')['Age'].agg(['mean', 'std', 'min', 'max', 'count'])
        stats_path = os.path.join(status_dir, "age_by_status_stats.csv")
        status_stats.to_csv(stats_path)
        logger.info(f"    Statistics saved to {stats_path}")

def main():
    """
    Main function to execute all categorical analysis steps
    """
    try:
        # Load data
        data = load_data()

        # Identify categorical variables
        categorical_vars = identify_categorical_variables(data)

        # Calculate frequency distributions
        frequencies, summary = calculate_frequency_distributions(data, categorical_vars)

        # Create bar charts
        create_bar_charts(data, categorical_vars, frequencies)

        # Create pie charts for variables with fewer categories
        create_pie_charts(data, categorical_vars, frequencies)

        # Create cross-tabulations
        create_cross_tabulations(data, categorical_vars)

        # Analyze protection equipment usage
```

```
        analyze_protection_equipment(data)

        # Analyze health complaints
        analyze_health_complaints(data)

        # Analyze education and socioeconomic effects
        analyze_educational_socioeconomic_effects(data)

        # Analyze marital status and children effects
        analyze_marital_status_and_children(data)

        # Analyze employment status
        analyze_employment_status(data)

        logger.info(f"All categorical analysis results saved to {output_dir}")

    except Exception as e:
        logger.error(f"Error in categorical analysis: {str(e)}")
        logger.exception("Detailed error information:")

if __name__ == "__main__":
    main()

    # Create heatmap
    plt.figure(figsize=(12, 8))

    sns.heatmap(
        edu_socio_cross,
        annot=True,
        fmt='.1f',
        cmap='viridis',
        cbar_kws={'label': 'Percentage'}
    )

    plt.title('Socioeconomic Status by Education Level', fontsize=14)
    plt.tight_layout()

    # Save the figure
    heatmap_path = os.path.join(edu_dir, "education_socioeconomic_heatmap.png")
    plt.savefig(heatmap_path, dpi=300)
    plt.close()
    logger.info(f"    Heatmap saved to {heatmap_path}")
```

```python
# Analyze protection equipment usage by education level
if 'protection_score_pct' in data.columns:
    logger.info("  Analyzing protection equipment usage by education level")

    # Create boxplot
    plt.figure(figsize=(12, 8))

    # Sort education levels if possible (assumes standardized order)
    education_order = None
    try:
        # Common education level ordering - modify based on your actual data
        education_order = [
            'analphabète', 'primaire', 'collège', 'secondaire', 'supérieur'
        ]
        # Filter to only existing categories
        education_order = [level for level in education_order if level in data['Niveau scolaire'].unique()]
    except:
        # If custom ordering fails, use data as is
        pass

    # Create boxplot
    sns.boxplot(
        x='Niveau scolaire',
        y='protection_score_pct',
        data=data,
        order=education_order
    )

    plt.title('Protection Equipment Usage by Education Level', fontsize=14)
    plt.xlabel('Education Level', fontsize=12)
    plt.ylabel('Protection Score (%)', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Add mean values as text
    for i, edu_level in enumerate(education_order if education_order else sorted(data['Niveau scolaire'].unique()
        mean_val = data[data['Niveau scolaire'] == edu_level]['protection_score_pct'].mean()
        count = len(data[data['Niveau scolaire'] == edu_level])
        plt.text(
            i,
            mean_val + 2,
            f"Mean: {mean_val:.1f}%\nN: {count}",
```

```
                ha='center',
                fontsize=9,
                bbox=dict(facecolor='white', alpha=0.8)
            )

        plt.tight_layout()

        # Save the figure
        box_path = os.path.join(edu_dir, "protection_by_education.png")
        plt.savefig(box_path, dpi=300)
        plt.close()
        logger.info(f"    Boxplot saved to {box_path}")

    # Analyze health complaints by socioeconomic status
    health_var_present = False
    for var in ['Troubles cardio-respiratoires', 'Troubles cognitifs', 'Troubles neurologiques', 'Troubles cutanés/ph
        if var in data.columns:
            health_var_present = True
            break

    if health_var_present:
        logger.info("  Analyzing health complaints by socioeconomic status")

        # Create a variable indicating if any health complaint is present
        data['any_health_complaint'] = False
        for var in ['Troubles cardio-respiratoires', 'Troubles cognitifs', 'Troubles neurologiques', 'Troubles cutane
            if var in data.columns:
                data['any_health_complaint'] = data['any_health_complaint'] | data[var].notna()

        # Create cross-tabulation
        health_socio_cross = pd.crosstab(
            data['Niveau socio-économique'],
            data['any_health_complaint'],
            normalize='index'
        ) * 100  # Convert to percentages

        # Save cross-tabulation to CSV
        cross_path = os.path.join(edu_dir, "health_socioeconomic_crosstab.csv")
        health_socio_cross.to_csv(cross_path)
        logger.info(f"    Cross-tabulation saved to {cross_path}")

        # Create bar chart
```

```
        plt.figure(figsize=(12, 8))

        # Only plot the "True" column (presence of health complaints)
        if True in health_socio_cross.columns:
            bars = plt.bar(
                health_socio_cross.index,
                health_socio_cross[True],
                color=sns.color_palette('viridis', len(health_socio_cross))
            )

            plt.title('Percentage with Health Complaints by Socioeconomic Status', fontsize=14)
            plt.xlabel('Socioeconomic Status', fontsize=12)
            plt.ylabel('Percentage with Health Complaints', fontsize=12)
            plt.grid(axis='y', linestyle='--', alpha=0.7)
            plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

            # Add percentage labels
            for bar in bars:
                height = bar.get_height()
                plt.text(
                    bar.get_x() + bar.get_width() / 2,
                    height + 1,
                    f'{height:.1f}%',
                    ha='center',
                    fontsize=10
                )

            plt.tight_layout()

            # Save the figure
            bar_path = os.path.join(edu_dir, "health_complaints_by_socioeconomic.png")
            plt.savefig(bar_path, dpi=300)
            plt.close()
            logger.info(f"    Bar chart saved to {bar_path}")

def analyze_marital_status_and_children(data):
    """
    Analyze the effects of marital status and number of children
    """
    # Check if necessary variables exist
    marital_var_exists = 'Situation maritale' in data.columns
    children_var_exists = 'Nb enfants' in data.columns
```

```
if not (marital_var_exists or children_var_exists):
    logger.info("Marital status and children variables not found. Skipping analysis.")
    return

logger.info("Analyzing effects of marital status and number of children...")

# Create directory for family analysis
family_dir = os.path.join(output_dir, "family_analysis")
os.makedirs(family_dir, exist_ok=True)

# Analyze working patterns by marital status
if marital_var_exists and 'H travail / jour' in data.columns:
    logger.info("  Analyzing work hours by marital status")

    # Create boxplot
    plt.figure(figsize=(12, 8))

    sns.boxplot(x='Situation maritale', y='H travail / jour', data=data)

    plt.title('Daily Work Hours by Marital Status', fontsize=14)
    plt.xlabel('Marital Status', fontsize=12)
    plt.ylabel('Hours per Day', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Add mean values as text
    for i, status in enumerate(sorted(data['Situation maritale'].unique())):
        mean_val = data[data['Situation maritale'] == status]['H travail / jour'].mean()
        count = len(data[data['Situation maritale'] == status])
        plt.text(
            i,
            mean_val + 0.2,
            f"Mean: {mean_val:.1f}\nN: {count}",
            ha='center',
            fontsize=9,
            bbox=dict(facecolor='white', alpha=0.8)
        )

    plt.tight_layout()

    # Save the figure
    box_path = os.path.join(family_dir, "work_hours_by_marital_status.png")
```

```
        plt.savefig(box_path, dpi=300)
        plt.close()
        logger.info(f"    Boxplot saved to {box_path}")

    # Analyze protection equipment usage by number of children (categorical)
    if children_var_exists and 'protection_score_pct' in data.columns:
        logger.info("  Analyzing protection equipment usage by number of children")

        # Create children categories
        data['children_category'] = pd.cut(
            data['Nb enfants'],
            bins=[-1, 0, 2, 4, float('inf')],
            labels=['None', '1-2', '3-4', '5+']
        )

        # Create boxplot
        plt.figure(figsize=(12, 8))

        sns.boxplot(x='children_category', y='protection_score_pct', data=data)

        plt.title('Protection Equipment Usage by Number of Children', fontsize=14)
        plt.xlabel('Number of Children', fontsize=12)
        plt.ylabel('Protection Score (%)', fontsize=12)
        plt.grid(axis='y', linestyle='--', alpha=0.7)

        # Add mean values as text
        for i, category in enumerate(sorted(data['children_category'].unique())):
            if pd.isna(category):
                continue
            mean_val = data[data['children_category'] == category]['protection_score_pct'].mean()
            count = len(data[data['children_category'] == category])
            plt.text(
                i,
                mean_val + 2,
                f"Mean: {mean_val:.1f}%\nN: {count}",
                ha='center',
                fontsize=9,
                bbox=dict(facecolor='white', alpha=0.8)
            )

        plt.tight_layout()
```

```
        # Save the figure
        box_path = os.path.join(family_dir, "protection_by_children.png")
        plt.savefig(box_path, dpi=300)
        plt.close()
        logger.info(f"    Boxplot saved to {box_path}")

    # Analyze health complaints by marital status
    if marital_var_exists:
        health_var_present = False
        for var in ['Troubles cardio-respiratoires', 'Troubles cognitifs', 'Troubles neurologiques', 'Troubles cutane
            if var in data.columns:
                health_var_present = True
                break

        if health_var_present:
            logger.info("  Analyzing health complaints by marital status")

            # Create a variable indicating if any health complaint is present if not already created
            if 'any_health_complaint' not in data.columns:
                data['any_health_complaint'] = False
                for var in ['Troubles cardio-respiratoires', 'Troubles cognitifs', 'Troubles neurologiques', 'Trouble
                    if var in data.columns:
                        data['any_health_complaint'] = data['any_health_complaint'] | data[var].notna()

            # Create cross-tabulation
            health_marital_cross = pd.crosstab(
                data['Situation maritale'],
                data['any_health_complaint'],
                normalize='index'
            ) * 100  # Convert to percentages

            # Save cross-tabulation to CSV
            cross_path = os.path.join(family_dir, "health_marital_crosstab.csv")
            health_marital_cross.to_csv(cross_path)
            logger.info(f"    Cross-tabulation saved to {cross_path}")
```