**SURVEYMASTER PRO: DYNAMIC DATA ANALYSIS & VISUALIZATION HUB FOR IN-DEPTH SURVEY INSIGHT**

**Description of the Proposed System**

This Python-based statistical system automation is named ‘SURVEYMASTER PRO’. This proposed system addresses the challenge of making survey data analysis accessible and understandable to a broad audience, particularly students and researchers with limited statistical knowledge.

The development of this system arises from a compelling need to democratize survey data analysis, making it accessible and comprehensible to researchers, particularly students, who may lack expertise in statistics. This system aims to eliminate reliance on costly statisticians and empower individuals with limited statistical knowledge to conduct meaningful survey data analysis independently.

The proposed system is designed to give researchers an easier way to analyze their survey data with interpretations and graphs. However, this section outlines the boundaries and defines how much this system will provide support while identifying specific processes beyond its purview. Additionally, external factors that may influence the system's performance are discussed.

The system encompasses a range of critical functions and features tailored to assist researchers, particularly students, in navigating the intricacies of survey data analysis:

1. **Survey Data Processing**
2. **Descriptive Statistics (Mean, Frequencies, Percentages)**
3. **Graphical Visualization**
4. **Statistical Tests (T-TEST, P-TEST, ANOVA)**
5. **Demographic Analysis:**
6. **Category/Factor Analysis:**
7. **Significant Differences Assessment:**
8. **Correlational Analysis:**

**PROGRAM AUTOMAINTERFACE**

1. **Install Python and Libraries:**A screenshot of a computer

   Description automatically generated
2. **Data File Preparation:**

A screenshot of a computer

Description automatically generated

* A screenshot of a survey

  Description automatically generated**Run the Script**

A screenshot of a survey

Description automatically generated

\

1. **Selection of Analysis Type (PROGRAM CAPABILITIES):**

A screenshot of a computer

Description automatically generated

1. **NON-NUMERIC RESPONSE CONVERSION:**

**A screenshot of a computer

Description automatically generated**

1. **Profile of the Respondents:**

A screenshot of a graph

Description automatically generatedA screenshot of a data

Description automatically generated

1. **CATEGORY/FACTOR ANALYSIS**

**A white paper with black text

Description automatically generated**

**A screenshot of a document

Description automatically generated**

1. A chart with different colored squares

   Description automatically generated**Correlational Analysis*Implications:***

A screenshot of a graph

Description automatically generated

1. **Significant Differences Analysis**

A screenshot of a graph

Description automatically generated

1. A green squares on a white background

   Description automatically generated**Individual Question AnalysisVerbal Interpretation:**

**Source Code**

#Installing of Required Systems

pip install pandas numpy matplotlib seaborn --upgrade scipy

pip install dash dash-core-components dash-html-components dash-table plotly

pip install dash dash-renderer dash-html-components dash-core-components plotly

pip install dash dash-bootstrap-components dash-table

pip install dash plotly

pip install dash dash-bootstrap-components pandas

pip install tabulate

### import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from scipy.stats import ttest\_ind

import dash

from dash import dcc, html, dash\_table

from dash.dependencies import Input, Output

import plotly.express as px

from tabulate import tabulate

import plotly.graph\_objs as go

from dash import Dash, dcc, html

from IPython.display import display, HTML, Markdown

import warnings

# This function loads survey data from an Excel file.

# It takes a filename as input and returns a DataFrame.

def load\_survey\_data(filename):

try:

# Try to read the Excel file into a DataFrame with the first row as the header.

return pd.read\_excel(filename, header=0)

except FileNotFoundError:

# If the file is not found, handle the exception and return None.

return None

except ValueError as e:

# If a value error occurs during reading, print the error message and return None.

print(f"Error: {e}")

return None

except Exception as e:

# If any unexpected error occurs, print the error message and return None.

print(f"Unexpected error: {e}")

return None

# This function preprocesses data based on the specified choice and handles missing values.

# It takes data (a DataFrame), num\_personal\_info\_cols (number of personal info columns),

# choice (preprocessing choice), and categorical\_mappings (mapping for categorical data) as inputs.

def preprocess\_data(data, num\_personal\_info\_cols, choice, categorical\_mappings):

if num\_personal\_info\_cols >= data.shape[1]:

# Check if the number of personal info columns exceeds the total number of columns.

# If so, raise a ValueError.

raise ValueError("Number of personal info columns exceeds total columns.")

if choice == 1:

# If choice is 1, drop rows with missing values (NaN).

data = data.dropna()

elif choice == 2:

# If choice is 2, fill missing values based on data type (object or numeric).

for column in data.columns[num\_personal\_info\_cols:]:

if data[column].dtype == 'object':

data[column].fillna(data[column].mode()[0], inplace=True)

else:

data[column].fillna(data[column].mean(), inplace=True)

elif choice == 3:

# If choice is 3, prompt the user for a value to fill missing data with.

fill\_value = input("Enter the value to fill missing data with: ")

data = data.fillna(fill\_value)

# Convert non-numeric responses to their mapped values if categorical\_mappings is provided.

if categorical\_mappings:

for column in data.columns:

if data[column].dtype == 'object':

data[column] = data[column].map(categorical\_mappings).fillna(data[column])

return data

# Define a function to display usage guidelines

def display\_usage\_guidelines():

while True:

# Define the usage guidelines as an HTML-formatted string

guidelines = """

<h2 style='color:green; font-weight:bold;'>USAGE GUIDELINES:</h2>

<b>Kindly adhere to the following recommendations to minimize errors and achieve the utmost precision in your outcomes.</b>

<br><br>

1. It is recommended to run this code in Jupyter Notebook within the Anaconda environment for

optimal accuracy, as certain features may not be available in other programming environments or languages.<br>

2. File Format: Ensure your survey data is in Excel format (.xlsx).<br>

3. The format of your excel datasets should be presented as follows:<br>

"""

# Define sample data as a list of dictionaries

sample\_data = [

{"Age": 15, "Gender": "Male", "Location": "Manila", "RTU is a Good School": "Agree", "The Facility of RTU is promising": "Strongly Agree", "BS Satatistics is Easy": "Disagree"},

{"Age": 18, "Gender": "Female", "Location": "Cavite", "RTU is a Good School": "Disagree", "The Facility of RTU is promising": "Agree", "BS Satatistics is Easy": "Strongly Agree"}

]

# Generate a formatted table from the sample data using the tabulate library

table = tabulate(sample\_data, headers="keys", tablefmt="html")

# Append the table to the guidelines

guidelines += table

# Add an explanation about the data columns

guidelines += """

<p>The first columns in your Excel file should contain demographic or personal information, like Age, Gender, Location, etc.. The remaining columns should contain survey questions</p>

<p>In the example data above, the first three (3) columns are reserved for demographic information (Age, Gender, Location). All columns following these initial three columns are dedicated to survey questions.\n</p>

"""

# Continue with the rest of the guidelines

guidelines += """

<p>4. Import Libraries: Make sure to install the required libraries at the beginning of your code:<br>

5. Categorical Mappings: If your survey contains non-numeric responses, consider creating mappings to convert them to numeric values.<br>

6. Interactive Dashboards: Some analysis options provide interactive dashboards. Use dropdown menus and graphs to explore data interactively.<br>

7. Run the code and follow the prompts and instructions.<br>

8. Enjoy Exploring Your Survey Data: This automation code simplifies survey data analysis, allowing you to gain valuable insights efficiently.<br><br>

<b>Have you read and taken into account the provided instructions? Please respond with 'yes' or 'no'<b>:

"""

# Display the guidelines in HTML format

display(HTML(guidelines))

# Prompt the user for a response

response = input().strip().lower()

# Check the user's response and return True or display error messages accordingly

if response == 'yes':

return True

elif response == 'no':

error\_message = """

<font color='red'><b>Please review and follow the provided instructions before proceeding.</b></font><br>

<font color='red'><b>If you are having trouble following the instructions or need assistance, you can contact the creators through the contact information provided in the user manual.</b></font>

"""

display(HTML(error\_message))

else:

error\_message = """

<font color='red'><b>Invalid response. Please respond with ' yes' or ' no'.</b></font>

"""

display(HTML(error\_message))

# This function displays a menu of program capabilities and allows the user to select analyses to perform.

def display\_menu():

# Define the menu with HTML formatting.

menu = """

<h2 style='color:green; font-weight:bold;'>PROGRAM CAPABILITIES</h2>

<p><b>1. DEMOGRAPHICS OF THE RESPONDENTS</b></p>

<p>- Create an interactive table displaying demographics data for respondents, including age, gender, and location. Show measures, frequency, and percentages.</p>

<p><b>2. CATEGORY/FACTOR ANALYSIS</b></p>

<p>- Analyze questions that measure a single factor or category. Provide the titles and questions under each factor/category as an example.</p>

<p><b>3. CORRELATIONAL ANALYSIS</b></p>

<p>- Investigate the presence of positive relationships between survey questions and between factors/categories. Generate a heatmap and highlight the top 3 strong positive relationships.</p>

<p><b>4. SIGNIFICANT DIFFERENCES BETWEEN DATAS</b></p>

<p>- Examine if there are significant differences between respondent demographics and their responses within each factor/category. <i>This analysis should only be conducted after completing the category/factor analysis (Capability No.2).</i></p>

<p><b>5. INDIVIDUAL ANALYSIS FOR ALL QUESTIONS</b></p>

<p>- Perform an individual analysis of all survey questions, presenting them with interactive charts for comprehensive insights.\n</p>

"""

# Display the menu

display(HTML(menu))

# Get user's selection

while True:

display(HTML("<font style='font-family:Arial;'><b>\nSelect the analysis you want to perform (e.g., '1' for Profile of the Respondents, '1,2,3' for multiple analyses, or 'all' for all analyses):</b></font> "))

selected\_analysis = input().lower()

try:

# Check if the input is valid (digits from 1 to 5, comma-separated, or 'all')

if selected\_analysis == 'all' or all(char.isdigit() and 1 <= int(char) <= 5 or char == ',' for char in selected\_analysis.split(',')):

# Check if '4' is selected without '2'

selected\_numbers = [int(char) for char in selected\_analysis.split(',') if char.isdigit()]

if 4 in selected\_numbers and 2 not in selected\_numbers:

raise ValueError("\n<font color='red'><b>Error:</b><b> You cannot select '4' without selecting '2'.</font></b>")

else:

return selected\_analysis

else:

raise ValueError("\n<font color='red'><b>Error:</b><b> Invalid input. Please enter a digit from 1 to 5, digits from 1 to 5 separated by a comma, or 'all'.\nExample: '1', '2,3', 'all'.</font></b>")

except ValueError as e:

display(Markdown(str(e)))

# This function checks for non-numeric responses in the data, provides instructions to convert them to numeric values,

# and allows the user to input the mappings.

# It takes 'data' (a DataFrame) and 'num\_personal\_info\_cols' (number of personal info columns) as inputs.

def display\_categorical\_mappings(data, num\_personal\_info\_cols):

non\_numeric\_responses = set()

# Check each column for non-numeric responses

for column in data.columns[num\_personal\_info\_cols:]:

if data[column].dtype == 'object':

non\_numeric\_responses.update(data[column].unique())

# Proceed only if there are non-numeric responses

if non\_numeric\_responses:

display(HTML("<h2 style='color:green; font-weight:bold;'>NON-NUMERIC RESPONSES CONVERSIONS:</h3>"))

non\_numeric\_responses\_str = ', '.join(f'<u><b>{response}</b></u>' for response in non\_numeric\_responses) # Format each response

# Display the instruction, non-numeric responses, and the example conversion table

instruction\_html = f"""

<b>\nINSTRUCTION:</b> Based on checking, your survey contains non-numeric responses: {non\_numeric\_responses\_str}<br>

<p>Please convert these non-numeric responses into numeric values so that the system can calculate them.</p>

<h4 style='color:green; font-weight:bold;'>Non-numeric Response Conversion Table Example:</h3>

"""

display(HTML(instruction\_html))

# Example data for non-numeric response conversion

example\_mapping\_data = [

['Very Satisfied', 4],

['Satisfied', 3],

['Dissatisfied', 2],

['Very Dissatisfied', 1]

]

# Generate an HTML table from the example data

html\_table = tabulate(example\_mapping\_data, headers=['Non-numeric', 'Numeric Equivalent'], tablefmt='html')

display(HTML(html\_table))

# Ask the user to input mappings

mapping\_data = []

for response in non\_numeric\_responses:

while True:

display(HTML(f"<p>Please enter the numeric value for <u><b>{response}</b></u>:</p>"))

try:

numeric\_value = int(input())

if numeric\_value > 0:

mapping\_data.append([response, numeric\_value])

break # Exit the loop if a valid input is provided

else:

raise ValueError("<font color='red'><b>Error:</b></font> Please enter a numeric value greater than 0.")

except ValueError as e:

display(Markdown(f"<font color='red'><b>Error:</b> Invalid input. Please enter a valid numeric value greater than 0.</font>"))

# Now you can create the dictionary from the mapping\_data

result\_dict = {response: value for response, value in mapping\_data}

return result\_dict

# If no non-numeric responses are found, an empty dictionary is returned

return {}

# This function analyzes personal information columns in the data and provides a profile analysis.

# It takes 'data' (a DataFrame) and 'num\_personal\_info\_cols' (number of personal info columns) as inputs.

def analyze\_personal\_info(data, num\_personal\_info\_cols):

# Initialize an empty dictionary to store the profile analysis results.

profile\_analysis = {}

# Iterate through the personal information columns.

for column in data.columns[:num\_personal\_info\_cols]:

# Calculate the counts of each unique value in the column.

counts = data[column].value\_counts()

# Calculate the total number of responses in the column.

total = len(data[column])

# Calculate the percentage of each unique value.

percentages = (counts / total \* 100).round(0).astype(int)

# Create a DataFrame to store the analysis results.

profile\_analysis[column] = pd.DataFrame({

'Measure': counts.index,

'Frequency': counts.values,

'Percentage': percentages.astype(str) + '%' # Format percentage values as strings.

})

# Return the profile analysis results as a dictionary.

return profile\_analysis

# This function interprets a verbal response (mean) based on a reverse mapping.

# It takes 'mean' (the mean value) and 'reverse\_mapping' (a mapping from mean to interpretation) as inputs.

def interpret\_verbal(mean, reverse\_mapping):

# Round the mean value to the nearest integer.

mean\_rounded = round(mean)

# Get the interpretation from the reverse mapping, or a default message if not found.

return reverse\_mapping.get(mean\_rounded, "No interpretation available")

# This function displays a Dash DataTable for profile analysis along with a pie chart and verbal interpretation.

# It takes 'profile\_analysis' (a dictionary of profile analysis data) and 'data' (the original data) as inputs.

def display\_profile\_table(profile\_analysis, data):

# Create a Dash web application

app = dash.Dash(\_\_name\_\_)

# Define the layout of the Dash application

app.layout = html.Div([

html.Div([

dcc.Dropdown(

id='profile-dropdown',

options=[{'label': column, 'value': column} for column in profile\_analysis.keys()],

value=list(profile\_analysis.keys())[0]

),

dash\_table.DataTable(

id='profile-table',

columns=[

{'name': 'Measure', 'id': 'Measure'},

{'name': 'Frequency', 'id': 'Frequency'},

{'name': 'Percentage', 'id': 'Percentage'}

],

style\_table={'overflowX': 'auto'},

style\_cell={'minWidth': 95},

),

html.Div([

dcc.Graph(id='profile-pie-chart', style={'height': '350px', 'width': '800px'})

], style={'display': 'flex', 'justify-content': 'center'}), # Center the pie chart

html.Div(id='verbal-interpretation', style={'color': 'black', 'text-align': 'center', 'font-weight': 'bold'})

]),

])

# Define a callback function to update the table, verbal interpretation, and pie chart based on user's selection

@app.callback(

[Output('profile-table', 'data'), Output('verbal-interpretation', 'children'), Output('profile-pie-chart', 'figure')],

[Input('profile-dropdown', 'value')]

)

def update\_table(selected\_column):

# Retrieve the DataFrame for the selected column

df = profile\_analysis[selected\_column]

# Find the response with the highest percentage

highest\_percentage\_row = df[df['Percentage'] == df['Percentage'].max()]

highest\_percentage\_response = highest\_percentage\_row.iloc[0]['Measure']

highest\_percentage = highest\_percentage\_row.iloc[0]['Percentage']

# Find the response with the lowest percentage

lowest\_percentage\_row = df[df['Percentage'] == df['Percentage'].min()]

lowest\_percentage\_response = lowest\_percentage\_row.iloc[0]['Measure']

lowest\_percentage = lowest\_percentage\_row.iloc[0]['Percentage']

# Create the verbal interpretation

verbal\_interpretation = html.Div([

html.H3(f"VERBAL INTERPRETATION [Demographic 1 : {selected\_column}]", style={'color': 'green', 'text-align': 'center', 'font-weight': 'bold'}),

html.P(f"\nAccording to the result of '{selected\_column}' demographics, '{highest\_percentage\_response}' has the highest number of respondents with {highest\_percentage} while the lowest is '{lowest\_percentage\_response}' with {lowest\_percentage}"),

])

# Create a pie chart

fig = px.pie(df, values='Frequency', names='Measure', title=f'{selected\_column} Distribution')

return df.to\_dict('records'), verbal\_interpretation, fig

# Run the Dash application on port 8050 without reloader

app.run\_server(port=8050, use\_reloader=False)

# This function analyzes categories of survey questions and provides category-wise and overall analysis.

# It takes 'data' (a DataFrame), 'categories' (a dictionary of categories and their associated questions),

# 'num\_personal\_info\_cols' (number of personal info columns), and 'reverse\_categorical\_mappings' (mapping for verbal responses) as inputs.

def analyze\_category(data, categories, num\_personal\_info\_cols, reverse\_categorical\_mappings):

# Initialize dictionaries to store category-wise and overall analysis.

category\_analysis = {}

overall\_analysis = []

overall\_n = []

# Iterate through the categories and their associated questions.

for category, questions in categories.items():

# Filter out invalid questions that are not present in the data columns.

valid\_questions = [q for q in questions if q in data.columns[num\_personal\_info\_cols:]]

# If no valid questions are found for the category, skip it with a warning.

if not valid\_questions:

print(f"Warning: No valid questions found for category '{category}'. Skipping this category.")

continue

# Extract data for the valid questions in the category.

category\_data = data[valid\_questions]

n = len(category\_data)

# Calculate the mean and standard deviation for the category's questions.

mean = category\_data.mean(numeric\_only=True)

std\_dev = category\_data.std(numeric\_only=True)

# Create a list to store category-specific analysis data.

analysis\_data = []

# For each question in the category.

for question in valid\_questions:

# Interpret the verbal response mean value.

interpretation = interpret\_verbal(mean[question], reverse\_categorical\_mappings)

analysis\_data.append([question, n, f"{mean[question]:.2f}", f"{std\_dev[question]:.2f}", interpretation])

# Calculate category averages.

total\_mean = mean.mean()

total\_std\_dev = std\_dev.mean()

total\_interpretation = interpret\_verbal(total\_mean, reverse\_categorical\_mappings)

analysis\_data.append([f"{category} OVERALL", n, f"{total\_mean:.2f}", f"{total\_std\_dev:.2f}", total\_interpretation])

category\_analysis[category] = analysis\_data

# Collect data for overall analysis.

overall\_n.append(n)

if analysis\_data:

overall\_analysis.append([category, n, f"{total\_mean:.2f}", f"{total\_std\_dev:.2f}", total\_interpretation])

# Add an overall row in overall analysis.

if overall\_analysis:

overall\_avg\_n = np.mean(overall\_n)

overall\_means = np.mean([float(row[2]) for row in overall\_analysis])

overall\_stdevs = np.mean([float(row[3]) for row in overall\_analysis])

overall\_interpretation = interpret\_verbal(overall\_means, reverse\_categorical\_mappings)

overall\_analysis.append(["OVERALL", overall\_avg\_n, f"{overall\_means:.2f}", f"{overall\_stdevs:.2f}", overall\_interpretation])

return category\_analysis, overall\_analysis

# This function displays category-wise and overall analysis using Dash.

# It takes 'category\_analysis' (a dictionary of category-wise analysis data),

# 'overall\_analysis' (a list of overall analysis data), and 'data' (the original data) as inputs.

def display\_category\_analysis(category\_analysis, overall\_analysis, data):

# Create a Dash web application

app = Dash(\_\_name\_\_)

# Define the layout of the Dash application

app.layout = html.Div([

html.Div([

dcc.Dropdown(

id='category-dropdown',

options=[{'label': category, 'value': category} for category in category\_analysis.keys()],

value=list(category\_analysis.keys())[0]

),

dash\_table.DataTable(

id='category-table',

columns=[

{'name': 'Question', 'id': 'Question'},

{'name': 'N', 'id': 'N'},

{'name': 'Mean', 'id': 'Mean'},

{'name': 'Std Dev', 'id': 'Std Dev'},

{'name': 'Interpretation', 'id': 'Interpretation'}

],

style\_table={'overflowX': 'auto'},

style\_cell={'minWidth': 95},

)

]),

html.Div(

id='verbal-interpretation',

style={

'margin': 'auto',

'text-align': 'center',

}

)

])

# Define a callback function to update the table and verbal interpretation based on the selected category

@app.callback(

[Output('category-table', 'data'), Output('verbal-interpretation', 'children')],

[Input('category-dropdown', 'value')]

)

def update\_category\_table(selected\_category):

# Retrieve the analysis data for the selected category

df = pd.DataFrame(category\_analysis[selected\_category], columns=['Question', 'N', 'Mean', 'Std Dev', 'Interpretation'])

# Find the highest and lowest mean scores for questions within the category

highest = df.iloc[df['Mean'].idxmax()]

lowest = df.iloc[df['Mean'].idxmin()]

# Calculate the overall mean and assessment for the category

overall\_mean = df['Mean'].astype(float).mean()

overall\_assessment = mean\_to\_category(overall\_mean)

# Prepare verbal interpretation content

interpretation\_content = html.Div([

html.H3("VERBAL INTERPRETATION", style={'color': 'green', 'text-align': 'center', 'font-weight': 'bold'}),

html.Center(html.P(f"In '{selected\_category}', the highest scoring question is '{highest['Question']}' with a mean score of {highest['Mean']} ({highest['Interpretation']}), while the lowest is '{lowest['Question']}' with a score of {lowest['Mean']} ({lowest['Interpretation']}),", style={'font-weight': 'bold'})),

html.Center(html.P(f"Overall, this category rates as {overall\_assessment} with an average score of {overall\_mean:.2f},", style={'font-weight': 'bold'}))

])

return df.to\_dict('records'), interpretation\_content

# Additional layout elements for overall analysis

if overall\_analysis:

overall\_df = pd.DataFrame(overall\_analysis, columns=['Category', 'N', 'Mean', 'Std Dev', 'Interpretation'])

# Modify the last row's category to "OVERALL"

if not overall\_df.empty:

overall\_df.at[len(overall\_df) - 1, 'Category'] = "OVERALL"

# Create an Overall Analysis Table

overall\_table = dash\_table.DataTable(

id='overall-table',

columns=[

{'name': 'Category', 'id': 'Category'},

{'name': 'N', 'id': 'N'},

{'name': 'Mean', 'id': 'Mean'},

{'name': 'Std Dev', 'id': 'Std Dev'},

{'name': 'Interpretation', 'id': 'Interpretation'}

],

data=overall\_df.to\_dict('records'),

style\_table={'overflowX': 'auto'},

style\_cell={'minWidth': 95},

)

# Find the highest and lowest mean scores across all categories in the overall analysis

highest = overall\_df.iloc[overall\_df['Mean'].astype(float).idxmax()]

lowest = overall\_df.iloc[overall\_df['Mean'].astype(float).idxmin()]

# Calculate the overall mean and assessment for the overall analysis

overall\_mean = overall\_df['Mean'].astype(float).mean()

overall\_assessment = mean\_to\_category(overall\_mean)

# Prepare verbal interpretation content for overall analysis

overall\_interpretation\_content = html.Div([

html.H3("OVERALL VERBAL INTERPRETATION", style={'color': 'green', 'text-align': 'center', 'font-weight': 'bold'}),

html.Center(html.P(f"In the overall analysis, the highest scoring category is '{highest['Category']}' with a mean score of {highest['Mean']} ({highest['Interpretation']}), while the lowest is '{lowest['Category']}' with a score of {lowest['Mean']} ({lowest['Interpretation']}),", style={'font-weight': 'bold'})),

html.Center(html.P(f"Overall, the analysis rates as {overall\_assessment} with an average score of {overall\_mean:.2f},", style={'font-weight': 'bold'}))

])

# Add Overall Analysis section to the layout

app.layout.children.append(html.Div([html.H3('Overall Analysis'), overall\_table, overall\_interpretation\_content]))

# Run the Dash application on port 8051 without reloader

app.run\_server(port=8051, use\_reloader=False)

# This function converts categorical data in a DataFrame to numeric values based on provided mappings.

# It takes 'data' (a DataFrame) and 'categories' (a dictionary of category-to-mapping pairs) as inputs.

def convert\_categorical\_to\_numeric(data, categories):

# Create a copy of the original data to avoid modifying the input DataFrame.

converted\_data = data.copy()

# Iterate through the categories and their associated mappings.

for category, mappings in categories.items():

# Check if the category exists as a column in the DataFrame.

if category in converted\_data.columns:

# Use the 'map' method to replace categorical values with their numeric equivalents.

converted\_data[category] = converted\_data[category].map(mappings)

# Return the DataFrame with categorical values converted to numeric values.

return converted\_data

# This function interprets a mean value using a reverse mapping.

# It rounds the mean value to the nearest integer and looks up its interpretation in the reverse mapping.

# If the rounded mean exists as a key in the reverse\_mapping, it returns the corresponding interpretation; otherwise, it returns "No interpretation available."

def interpret\_verbal(mean, reverse\_mapping):

mean\_rounded = round(mean) # Round the mean value to the nearest integer

return reverse\_mapping.get(mean\_rounded, "No interpretation available") # Look up the interpretation in the reverse\_mapping

# This function categorizes a mean value into descriptive categories.

# It categorizes mean values based on the following thresholds:

# - If the mean\_value is greater than or equal to 4, it returns 'Very High'.

# - If the mean\_value is greater than or equal to 3, it returns 'High'.

# - If the mean\_value is greater than or equal to 2, it returns 'Medium'.

# - Otherwise, it returns 'Low'.

def mean\_to\_category(mean\_value):

if mean\_value >= 4:

return 'Very High'

elif mean\_value >= 3:

return 'High'

elif mean\_value >= 2:

return 'Medium'

else:

return 'Low'

# This function displays combined correlation heatmaps for individual survey questions and categories,

# along with verbal interpretations for both. It visualizes the correlations between questions and

# categories in the cleaned data using heatmaps and provides insights into relationships.

def display\_combined\_correlation\_heatmaps(cleaned\_data, num\_personal\_info\_cols, categories):

# Set font style with Times New Roman

font\_style = {'family': 'serif', 'weight': 'bold', 'fontname': 'Times New Roman'}

# Heatmap for Individual Questions

survey\_questions\_data = cleaned\_data.iloc[:, num\_personal\_info\_cols:]

corr\_matrix\_questions = survey\_questions\_data.corr()

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

# Use a green colormap for the heatmap

ax1 = sns.heatmap(corr\_matrix\_questions, annot=True, cmap='YlGnBu', fmt=".2f", cbar=False)

ax1.set\_title("Individual Question Correlation Heatmap", fontdict=font\_style)

# Numbering format for x-axis labels

x\_labels = [f'Q{i}' for i in range(1, len(corr\_matrix\_questions.columns) + 1)]

ax1.set\_xticklabels(x\_labels, rotation=90, va="center", position=(0, -0.05))

# Numbering format for y-axis labels

y\_labels = [f'Q{i}' for i in range(1, len(corr\_matrix\_questions.columns) + 1)]

ax1.set\_yticklabels(y\_labels, rotation=0)

ax1.xaxis.tick\_top() # Move x-axis labels to top

plt.show()

# Displaying verbal interpretation for Individual Questions

display\_top\_relationships(corr\_matrix\_questions)

# Heatmap for Categories

category\_data = pd.DataFrame()

for category, questions in categories.items():

category\_data[category] = cleaned\_data[questions].mean(axis=1)

corr\_matrix\_categories = category\_data.corr()

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

# Use a green colormap for the heatmap

ax2 = sns.heatmap(corr\_matrix\_categories, annot=True, cmap='YlGnBu', fmt=".2f", cbar=False)

ax2.set\_title("Category Correlation Heatmap", fontdict=font\_style)

ax2.xaxis.tick\_top() # Move x-axis labels to top

plt.show()

# Displaying verbal interpretation for Categories

display\_top\_relationships(corr\_matrix\_categories)

# This function displays the top 3 strong positive relationships in a correlation matrix

# and provides verbal interpretations for each relationship.

# It takes a correlation matrix 'corr\_matrix' as input and outputs the verbal interpretations in HTML format.

def display\_top\_relationships(corr\_matrix):

# Get all pairs of correlations from the correlation matrix

corr\_pairs = corr\_matrix.unstack()

# Sort the correlation pairs in descending order (strongest correlations first)

sorted\_pairs = corr\_pairs.sort\_values(kind="quicksort", ascending=False)

# Filter out correlations with a value of 1 (self-correlations) and keep only strong positive correlations

strong\_pairs = sorted\_pairs[sorted\_pairs != 1]

# Filter out repeated pairs to keep only unique top pairs

seen\_pairs = set()

unique\_top\_pairs = []

for pair, value in strong\_pairs.items():

# Check if the reverse pair (B, A) is not already seen and avoid self-correlations

if pair[::-1] not in seen\_pairs and pair[0] != pair[1]:

seen\_pairs.add(pair)

unique\_top\_pairs.append((pair, value))

if len(unique\_top\_pairs) == 3:

break # Stop after finding the top 3 unique pairs

# Generate HTML output for the top 3 strong positive relationships

html\_output = '<h2 style="color: green; text-align: center; font-weight: bold; font-family: Times New Roman;">VERBAL INTERPRETATION - Top 3 Strong Positive Relationships</h2>'

for pair, value in unique\_top\_pairs:

# Create a verbal interpretation for each strong positive relationship

relationship = f"The variables '{pair[0]}' and '{pair[1]}' show a strong positive correlation: {value:.2f}"

html\_output += f'<p style="text-align: center; font-weight: bold; font-family: Times New Roman;">{relationship}</p>'

# Display the HTML output

display(HTML(html\_output))

# This function computes significant differences in means between demographic groups and category means.

# It performs t-tests to assess whether there are significant differences in category responses

# based on different demographic groups.

# The function takes 'data' (cleaned survey data), 'num\_personal\_info\_cols' (number of demographic columns),

# and 'categories' (a dictionary of category names and their associated questions) as input.

def compute\_significant\_differences(data, num\_personal\_info\_cols, categories):

results = []

# Calculate category means

category\_means = {category: data[questions].mean(axis=1).mean() for category, questions in categories.items()}

# Loop through each demographic column

for i in range(num\_personal\_info\_cols):

demographic\_column = data.columns[i]

unique\_demographics = data[demographic\_column].unique()

# Loop through each category

for category, category\_mean in category\_means.items():

# Calculate mean for each demographic within the category

for demo in unique\_demographics:

demo\_data = data[data[demographic\_column] == demo][categories[category]].mean(axis=1)

demo\_data = demo\_data[~np.isnan(demo\_data)] # Exclude NaN values

# Check if any column is empty

if demo\_data.empty:

continue

# Perform the t-test against the overall category mean

t\_stat, p\_val = stats.ttest\_1samp(demo\_data, category\_mean)

# Append the results

results.append({

'Category': category,

'Demographic': demographic\_column,

'Group': demo,

'N': len(demo\_data),

'Mean': f"{demo\_data.mean():.2f}",

'Std Dev': f"{demo\_data.std(ddof=1):.4f}",

'T-Value': f"{t\_stat:.4f}",

'P-Value': f"{p\_val:.4f}",

'Significant Difference': 'Yes' if p\_val < 0.05 else 'No'

})

return pd.DataFrame(results)

# This function displays a dashboard for significant differences analysis based on demographic groups.

# It uses Dash for creating the interactive dashboard. The dashboard includes a dropdown to select

# a demographic column, a table to display significant differences results, and a verbal interpretation.

# The function takes 'data' (cleaned survey data), 'num\_personal\_info\_cols' (number of demographic columns),

# and 'categories' (a dictionary of category names and their associated questions) as input.

def display\_significant\_differences\_dashboard(data, num\_personal\_info\_cols, categories):

# Compute significant differences using the provided function

sig\_diffs\_df = compute\_significant\_differences(data, num\_personal\_info\_cols, categories)

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_, external\_stylesheets=['https://codepen.io/chriddyp/pen/bWLwgP.css'])

# Define the font style

font\_style = {'font-family': 'Times New Roman'}

# Define the layout of the dashboard

app.layout = html.Div([

dcc.Dropdown(

id='demographic-dropdown',

options=[{'label': col, 'value': col} for col in data.columns[:num\_personal\_info\_cols]],

value=data.columns[0]

),

dash\_table.DataTable(

id='sig-diffs-table',

columns=[{"name": i, "id": i} for i in sig\_diffs\_df.columns],

data=sig\_diffs\_df.to\_dict('records'),

style\_table={'overflowX': 'auto'},

),

html.Div(id='verbal-interpretation', style=font\_style)

])

@app.callback(

Output('sig-diffs-table', 'data'),

Output('verbal-interpretation', 'children'),

[Input('demographic-dropdown', 'value')]

)

def update\_dashboard(selected\_demographic):

# Filter the data based on the selected demographic

filtered\_data = sig\_diffs\_df[sig\_diffs\_df['Demographic'] == selected\_demographic]

# Filter significant differences within the selected demographic

significant\_diffs = filtered\_data[filtered\_data['Significant Difference'] == 'Yes']

if significant\_diffs.empty:

verbal\_summary = html.P(f"No significant differences found for the demographic: {selected\_demographic}.",

style={'text-align': 'center', 'font-weight': 'bold'})

else:

categories\_with\_diffs = ', '.join(significant\_diffs['Category'].unique())

verbal\_summary = html.P(f"Significant differences found for the demographic: {selected\_demographic} in categories: {categories\_with\_diffs}.",

style={'text-align': 'center', 'font-weight': 'bold'})

# Create the verbal interpretation section

return filtered\_data.to\_dict('records'), html.Div([

html.H3("VERBAL INTERPRETATION", style={'text-align': 'center', 'color': 'green', 'font-family': 'Times New Roman', 'font-weight': 'bold'}),

verbal\_summary

])

# Run the Dash app in debug mode

app.run\_server(debug=True)

# This function generates a verbal interpretation for a selected survey question.

# It calculates statistics such as the highest response option, its count, percentage of total responses,

# and the mean value for the selected question. It then creates an HTML interpretation section.

# The function takes 'selected\_question' (the question for which interpretation is generated)

# and 'data' (survey data containing the responses to the selected question) as input.

def generate\_verbal\_interpretation(selected\_question, data):

# Select the data for the chosen question

selected\_data = data[selected\_question]

# Calculate the option with the highest responses and its count

highest\_response\_option = selected\_data.value\_counts().idxmax()

highest\_response\_count = selected\_data.value\_counts().max()

# Calculate the percentage of total responses for the highest response option

total\_respondents = len(selected\_data)

percentage = (highest\_response\_count / total\_respondents) \* 100

# Calculate the mean value for the selected question

mean\_value = selected\_data.mean()

# Create a verbal interpretation section as an HTML Div

interpretation = html.Div([

html.H3(f"VERBAL INTERPRETATION: '{selected\_question}'", style={'color': 'green', 'text-align': 'center', 'font-weight': 'bold'}),

html.P(f"\nThe data of '{selected\_question}' shows that the option with the highest responses is '{highest\_response\_option}' with {highest\_response\_count} respondents, accounting for {percentage:.2f}% of the total. The calculated mean for '{selected\_question}' is {mean\_value:.2f}."),

])

return interpretation

def run\_interactive\_dashboard(data, num\_personal\_info\_cols):

app = dash.Dash(\_\_name\_\_)

# Filter out personal info columns from survey\_questions

survey\_questions = [q for q in data.columns if q not in data.columns[:num\_personal\_info\_cols]]

app.layout = html.Div([

dcc.Dropdown(

id='question-dropdown',

options=[{'label': q, 'value': q} for q in survey\_questions],

value=survey\_questions[0]

),

dcc.Graph(id='question-graph'),

html.Div(id='verbal-interpretation', style={'color': 'black', 'text-align': 'center', 'font-weight': 'bold'})

])

@app.callback(

[Output('question-graph', 'figure'), Output('verbal-interpretation', 'children')],

[Input('question-dropdown', 'value')]

)

def update\_graph(selected\_question):

filtered\_data = data[selected\_question].value\_counts()

fig = px.bar(

filtered\_data,

x=filtered\_data.index,

y=filtered\_data.values,

labels={'x': selected\_question, 'y': 'Counts'},

color\_discrete\_sequence=['green'] # Set the color to green

)

# Generate the verbal interpretation based on the selected question

interpretation = generate\_verbal\_interpretation(selected\_question, data)

return fig, interpretation

app.run\_server(port=8053, use\_reloader=False)

# This function generates and displays an HTML heading for analysis with the specified title.

def display\_analysis\_heading(title):

# Generate an HTML heading for analysis with green color and bold font weight

analysis\_heading = f"<h2 style='color: green; font-weight: bold;'>{title}</h2>"

# Display the HTML heading

display(HTML(analysis\_heading))

# This function displays the title of the program with a specified console width.

def display\_title(title):

# Determine the console width for centering the title

console\_width = 140

# Print the title centered within the console width

print(title.center(console\_width), "\n")

# Use a context manager to temporarily filter out the specific warning

with warnings.catch\_warnings():

warnings.filterwarnings("ignore", category=RuntimeWarning, message="Precision loss occurred in moment calculation due to catastrophic cancellation.")

def custom\_warning\_filter(message, category, filename, lineno, file=None, line=None):

if "Precision loss occurred in moment calculation due to catastrophic cancellation" in str(message):

return None

else:

return message, category, filename, lineno, line

# Apply the custom warning filter

warnings.showwarning = custom\_warning\_filter

# This is the main function of the SURVEYMASTER PRO system, which automates survey data analysis.

def main():

# Set the title for the web page

title = "<h1 style='color:white; background-color:green; font-size:24px; padding:10px; text-align:center;'>SURVEYMASTER PRO: SURVEY DATA ANALYSIS AUTOMATION SYSTEM</h1>"

display(HTML(title))

# Display usage guidelines and confirm if the user understands them

if not display\_usage\_guidelines():

return

# Add the H2 title for dataset collection

dataset\_collection\_title = "<h2 style='font-size:20px; color:green;'>COLLECTION AND CLEANING OF SURVEY DATASET</h2>"

display(HTML(dataset\_collection\_title))

# Display instructions for the user

instructions = """

<p style='font-family:Arial;'><b>INSTRUCTION:</b> Please ensure you have the directory path of your survey data file.</p>

<p style='font-family:Arial;'>Your file should be in Excel (.xlsx) format. For example: 'C:/Users/YourName/Documents/survey\_data.xlsx'</p>

<p style='font-family:Arial;'>For guidance on how to find your file's directory, visit: <a href='https://www.datanumen.com/blogs/3-effective-methods-to-get-the-path-of-the-current-excel-workbook/'>here</a>.</p>

"""

display(HTML(instructions))

# Loop to input and load survey data file

while True:

try:

display(HTML("<p><b>Enter the directory path of your survey data file:</b></p>"))

filename = input()

data = load\_survey\_data(filename)

if data is None:

raise ValueError("<font color='red'><b>Error:</b></font>")

break

except ValueError as e:

display(Markdown(f"\n<font color='red'><b>Error:</b><b> No data loaded. Please check your file path and ensure it's an Excel file. Please try again.</font></b>"))

# Check if data is loaded successfully

if data is not None:

# Input the number of columns with demographics

while True:

try:

display(HTML("<font style='font-family:Arial;'><b>Enter the number of columns with demographics of the respondents, for example, age, gender, and/or location (You may refer to USAGE GUIDELINES No.2 for further instruction.):</b></font> "))

num\_personal\_info\_cols = int(input())

if num\_personal\_info\_cols < 1 or not isinstance(num\_personal\_info\_cols, int):

raise ValueError("<font color='red'><b>Error:</b></font>")

break

except ValueError as e:

display(Markdown(f"\n<font color='red'><b>Error:</b><b> Number of personal info/demographics columns should be greater than 0 and less than the total columns. Please enter a valid number.</font></b>"))

# Choose how to handle missing data

while True:

try:

display(HTML("<font style='font-family:Arial;'><b>Choose how to handle missing data (1: Drop, 2: Fill with mean/mode, 3: Fill with a value):</font> "))

choice = int(input())

if choice not in [1, 2, 3]:

raise ValueError("<font color='red'><b>Error:</b></font>")

break

except ValueError as e:

display(Markdown(f"\n<font color='red'><b>Error:</b><b> Please enter a valid choice (1, 2, or 3).</font></b>"))

# Preprocess the data based on user's choice

cleaned\_data = preprocess\_data(data, num\_personal\_info\_cols, choice, {})

# Get user's analysis selection

selected\_analysis = display\_menu()

# Map non-numeric responses to numeric values

categorical\_mappings = display\_categorical\_mappings(data, num\_personal\_info\_cols)

cleaned\_data = preprocess\_data(data, num\_personal\_info\_cols, choice, categorical\_mappings)

# Profile of the Respondents Analysis

if '1' in selected\_analysis or 'all' in selected\_analysis:

analysis\_title = "I. PROFILE OF THE RESPONDENTS"

# Display the analysis heading

display\_analysis\_heading(analysis\_title)

# Display all tables for personal information or profile of the respondents

profile\_analysis = analyze\_personal\_info(cleaned\_data, num\_personal\_info\_cols)

display\_profile\_table(profile\_analysis, data)

# Factor/Category Analysis

if '2' in selected\_analysis or 'all' in selected\_analysis:

print() # Add a newline to separate sections

display\_analysis\_heading("II. CATEGORY/FACTOR ANALYSIS")

# Updated instruction with HTML formatting

instruction\_html = """

<font color='black'><b>DESCRIPTION:</b></font> This program capability enables you to analyze your survey questions grouped under a single category.

This feature is particularly useful when you want to assess questions that measure a specific factor or category.<br><br>

In your survey data file, each question is assigned a unique number for your convenience (Q1, Q2, Q3..).

This means you no longer need to copy and paste questions. Instead, you can simply input the corresponding question number."

"""

# Display the updated instruction using HTML

display(HTML(instruction\_html))

# Display survey questions with HTML formatting and numbering

questions\_html = ""

for i, question in enumerate(cleaned\_data.columns[num\_personal\_info\_cols:], start=1):

questions\_html += f"Q{i}. {question}<br>"

display(HTML(questions\_html))

# Ask for categories and questions under each category

categories = {}

instruction\_html = "<font color='black'><b>INSTRUCTION:</b></font> Please specify the titles for each of your categories and their respective questions by entering the corresponding question numbers from the list above. Once you have listed all of the categories, simply type 'done' to continue."

# Display the instruction with HTML formatting

display(HTML(instruction\_html))

# Display the example input message with HTML formatting

example\_inputs\_html = "<font color='black'><b> EXAMPLE:</b></font><br> Enter the 1st category title (or 'done' to finish): ENTERTAINMENT<br> Enter the question numbers for the ‘ENTERTAINMENT category, separated by commas. (Please refer to the numbered questions above): Q1, Q2, Q3, Q4"

display(HTML(example\_inputs\_html))

def ordinal(n):

if 10 <= n % 100 <= 20:

suffix = 'th'

else:

suffix = {1: 'st', 2: 'nd', 3: 'rd'}.get(n % 10, 'th')

return f"{n}{suffix}"

categories = {}

category\_number = 1

BOLD = "\033[1m"

BLACK = "\033[30m"

# Main loop to enter categories and questions

while True:

try:

# Prompt for the category title

category\_name = input(f"{BLACK}{BOLD}Enter the {ordinal(category\_number)} category title (or 'done' to finish): {BOLD}")

if category\_name.lower() == 'done':

break

# Prompt for question numbers

while True:

question\_nums = input(f"{BLACK}{BOLD}Enter the question numbers for the '{category\_name}' category, separated by commas. (Please refer to the numbered questions above. Enter as Q1, Q2, Q3, Q4): {BOLD} ")

try:

# Process question numbers and get corresponding questions

question\_indices = []

for num in question\_nums.split(','):

num = num.strip().upper() # Convert to uppercase and remove spaces

if not num.startswith("Q") or not num[1:].isdigit():

raise ValueError("<font color='red'><b>Error: Invalid input format. Please enter question numbers as Q1, Q2, Q3, etc. You may refer to the list provided above.</b></font>")

index = int(num[1:]) - 1

if index < 0 or index >= len(cleaned\_data.columns) - num\_personal\_info\_cols:

raise ValueError(f"<font color='red'><b>Error: Invalid question number '{num}'. Please enter a valid question number within the provided list.</b></font>")

question\_indices.append(index)

questions = [cleaned\_data.columns[num\_personal\_info\_cols + i] for i in question\_indices]

# Store the category and its questions

categories[category\_name] = questions

# Increment category number

category\_number += 1

break # Exit the loop if input is valid

except ValueError as e:

display(HTML(f"<font color='red'><b>{e}</b></font>"))

except KeyboardInterrupt:

print("\nProcess interrupted.")

except KeyboardInterrupt:

print("\nProcess interrupted.")

# Define reverse\_categorical\_mappings here if necessary

reverse\_categorical\_mappings = {

1: 'Low',

2: 'Medium',

3: 'High',

4: 'Very High'

}

if categories:

category\_analysis, overall\_analysis = analyze\_category(cleaned\_data, categories, num\_personal\_info\_cols,

reverse\_categorical\_mappings)

display\_category\_analysis(category\_analysis, overall\_analysis, data)

# Correlational Analysis

if '3' in selected\_analysis or 'all' in selected\_analysis:

display\_analysis\_heading("III. CORRELATIONAL ANALYSIS")

# Display the correlation heatmap

display\_combined\_correlation\_heatmaps(cleaned\_data, num\_personal\_info\_cols, categories)

# Significant Difference Analysis

if '4' in selected\_analysis or 'all' in selected\_analysis:

display\_analysis\_heading("IV. SIGNIFICANT DIFFERENCES BETWEEN PROFILE OF THE RESPONDENTS AND SURVEY RESPONSES")

display\_significant\_differences\_dashboard(cleaned\_data, num\_personal\_info\_cols, categories)

# Individual Analysis

if '5' in selected\_analysis or 'all' in selected\_analysis:

display\_analysis\_heading("V. INDIVIDUAL ANALYSIS FOR ALL QUESTIONS")

# Run interactive dashboard

run\_interactive\_dashboard(data, num\_personal\_info\_cols)

# Check if the script is being run as the main program

if \_\_name\_\_ == "\_\_main\_\_":

# If so, execute the main() function

main()

**Bibliography**

Bajaj, K. (2020). *Importance of Statistical Tools and Methods in Data Science*. https://www.academia.edu/43677731/Importance\_of\_statistical\_tools\_and\_methods\_in\_Data\_Science

Cambridge University Press. (2023). *Empathy*. In *Cambridge Dictionary*. Retrieved December 11, 2023, fromhttps://dictionary.cambridge.org/dictionary/english

IBM. (2023). *Data Analysis for Small Businesses: Overcoming Budget Constraints*. https://www.ibm.com/blog/how-small-and-mid-sized-businesses-can-perform-big-data-analytics/

Konecki, M., LaPierre, C., & Jervis, K. (2018). Accessible Data Visualization in Higher Education. *International Convention on Information and Communication Technology, Electronics and Microelectronics.* doi:10.23919/MIPRO.2018.8400136.

Rashid, C. A. (2021). The importance of statistical analysis in accounting research. Journal of Global Social Sciences, 2(7). <https://doi.org/10.31039/jgss.v2i7.26>

Smith, J., Jones, M., & Brown, T. (2023). Understanding the challenges faced by novice users of IBM SPSS Statistics: A qualitative inquiry. Computers in Human Behavior, p. 139, 107417.

**Rubrics**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Points** | **Score** |
| Functionality: Does the system meet all the requirements and accurately perform computations? | 25 |  |
| Code Quality: Is the code well-written, readable, well-structured, and efficient? | 20 |  |
| User Experience: Is the system easy to use and navigate for users with a basic understanding of automation? Does it provide helpful messages and error handling? | 25 |  |
| Documentation: Is the code and system documentation complete and clear, including explanations of concepts and procedures used? | 15 |  |
| Presentation: Is the presentation clear, concise, and visually appealing, including charts and graphs where applicable? | 15 |  |
| **TOTAL** | 100 |  |