PA1

```
#%% MODULE BEGINS
module_name = 'module_tmp'

'''
Version: v0.2

Description:
    A module to handle data processing, querying, visualization, and logging tasks, using CSV
and Pickle files.

Authors:
   Madison DeHart, Natalie Tallant, Shakurah

Date Created     :  11/11/2024
Date Last Updated:  11/13/2024


#%% IMPORTS                  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
import os
from copy import deepcopy as dpcpy
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
import pickle
import logging
import numpy as np
from typing import List, Tuple

#%% CONFIGURATION              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CONFIG = {
    "input_path": "./input",
    "output_path": "./output",
    "log_file": "./log.txt",
```

```
    "fig_size": (10, 6),
}

logging.basicConfig(filename=CONFIG['log_file'], level=logging.INFO,
                    format='%(asctime)s:%(levelname)s:%(message)s')

#%% CLASS DEFINITIONS

class DataProcessor:
    """ Parent class for data processing and visualization. """

    def __init__(self):
        self.config = CONFIG
        logging.info("DataProcessor instance created.")

    def read_csv(self, csv_file: str) -> pd.DataFrame:
        """ Reads data from a CSV file into a DataFrame. """
        try:
            df = pd.read_csv(os.path.join(self.config["input_path"], csv_file))
            logging.info(f"CSV file '{csv_file}' read successfully.")
            return df
        except FileNotFoundError as e:
            logging.error(f"File '{csv_file}' not found: {e}")
            return pd.DataFrame()

    def read_pickle(self, pickle_file: str) -> pd.DataFrame:
        """ Reads data from a Pickle file into a DataFrame. """
        try:
            with open(os.path.join(self.config["input_path"], pickle_file), 'rb') as file:
                data = pickle.load(file)
            logging.info(f"Pickle file '{pickle_file}' read successfully.")
            return pd.DataFrame(data)
        except FileNotFoundError as e:
            logging.error(f"File '{pickle_file}' not found: {e}")
            return pd.DataFrame()

    def export_csv(self, df: pd.DataFrame, filename: str) -> None:
        """ Exports DataFrame to a CSV file. """
        export_path = os.path.join(self.config["output_path"], filename)
        df.to_csv(export_path, index=False)
```

```
        logging.info(f"Data exported to CSV file '{filename}'.")

    def export_pickle(self, df: pd.DataFrame, filename: str) -> None:
        """ Exports DataFrame to a Pickle file. """
        export_path = os.path.join(self.config["output_path"], filename)
        with open(export_path, 'wb') as file:
            pickle.dump(df, file)
        logging.info(f"Data exported to Pickle file '{filename}'.")

    def visualize_distribution(self, df: pd.DataFrame, column: str) -> None:
        """ Visualizes the distribution of a column using a histogram and a violin plot. """
        plt.figure(figsize=self.config['fig_size'])
        sns.histplot(df[column], kde=True)
        plt.title(f"Histogram of {column}")
        plt.savefig(os.path.join(self.config["output_path"], f"hist_{column}.png"))
        plt.close()

        plt.figure(figsize=self.config['fig_size'])
        sns.violinplot(data=df, x=column)
        plt.title(f"Violin Plot of {column}")
        plt.savefig(os.path.join(self.config["output_path"], f"violin_{column}.png"))
        plt.close()
        logging.info(f"Visualizations for column '{column}' saved.")

    def calculate_statistics(self, df: pd.DataFrame, column: str) -> dict:
        """ Calculates basic statistics for a given column in the DataFrame. """
        stats = {
            "mean": df[column].mean(),
            "median": df[column].median(),
            "std": df[column].std(),
            "min": df[column].min(),
            "max": df[column].max()
        }
        logging.info(f"Statistics for column '{column}': {stats}")
        return stats


class DataAnalyzer(DataProcessor):
    """ Child class extending DataProcessor to add querying and probability calculations. """
```

```
    def query_data(self, df: pd.DataFrame, condition: str) -> pd.DataFrame:
        """ Queries the DataFrame based on a given condition. """
        result = df.query(condition)
        logging.info(f"Data queried with condition '{condition}'.")
        return result

    def calculate_probabilities(self, df: pd.DataFrame, column: str) -> dict:
        """ Calculates probability distributions (mean, median, std, etc.). """
        mean = df[column].mean()
        std_dev = df[column].std()
        probabilities = {
            "mean": mean,
            "std_dev": std_dev,
            "prob_gt_mean": norm.sf(mean, mean, std_dev)
        }
        logging.info(f"Probabilities calculated for column '{column}': {probabilities}")
        return probabilities

    def calculate_vector_metrics(self, vector_a: np.ndarray, vector_b: np.ndarray) -> dict:
        """ Calculates vector metrics including dot product and angle between vectors. """
        dot_product = np.dot(vector_a, vector_b)
        angle = np.arccos(dot_product / (np.linalg.norm(vector_a) * np.linalg.norm(vector_b)))
        orthogonal = np.isclose(dot_product, 0)
        metrics = {
            "dot_product": dot_product,
            "angle_radians": angle,
            "orthogonal": orthogonal
        }
        logging.info(f"Vector metrics calculated: {metrics}")
        return metrics


#%% MAIN FUNCTION
def main():
    processor = DataAnalyzer()
    df_csv = processor.read_csv("data.csv")
    if not df_csv.empty:
        processor.visualize_distribution(df_csv, 'some_numeric_column')
        stats = processor.calculate_statistics(df_csv, 'some_numeric_column')
        probabilities = processor.calculate_probabilities(df_csv, 'some_numeric_column')
```

```
        processor.export_csv(df_csv, "processed_data.csv")
    else:
        logging.error("No data to process from CSV file.")

#%% SELF-RUN
if __name__ == "__main__":
    main()
```