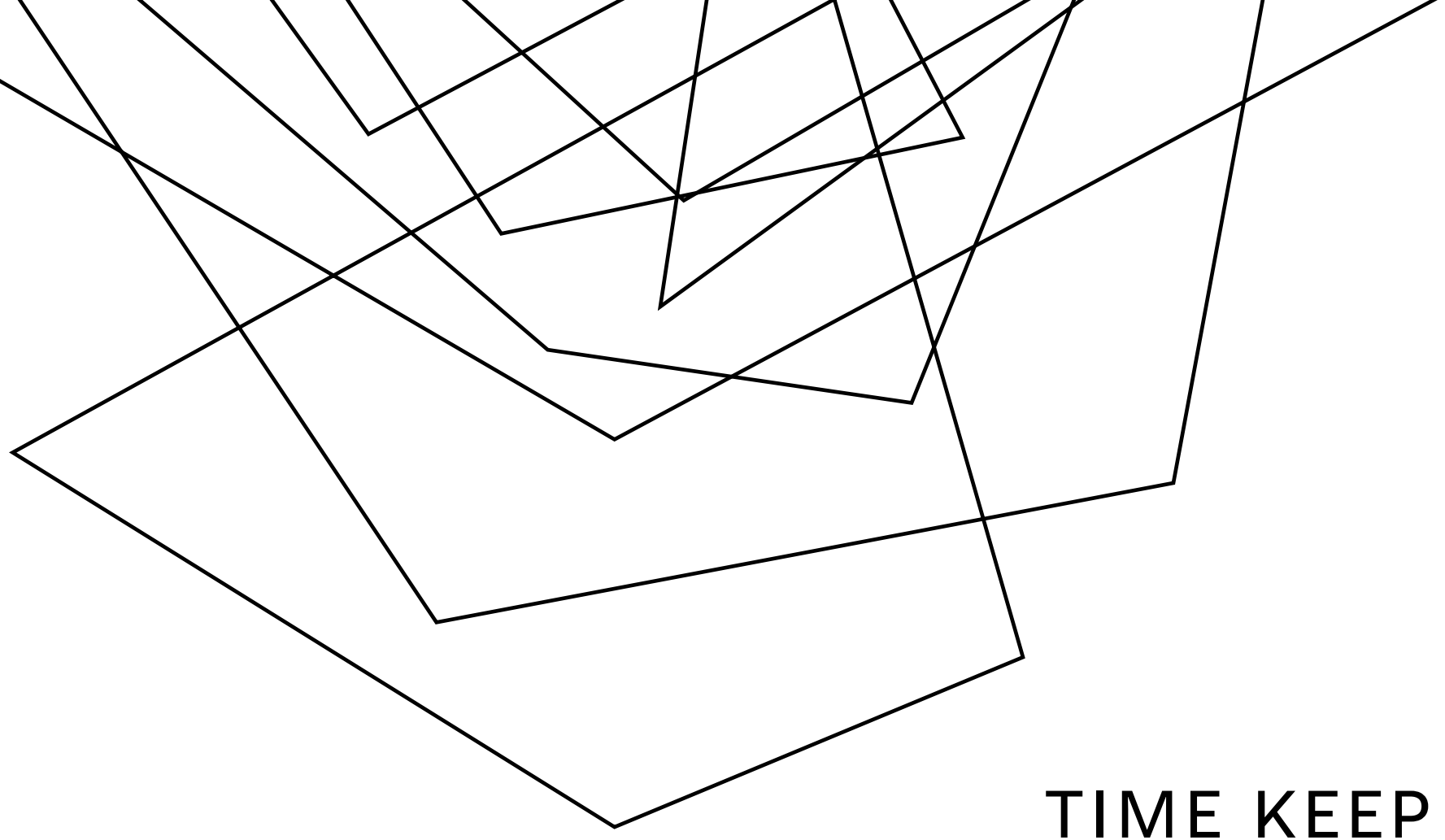


TIME KEEP CO. AND CHATTERBOT

Briana Metalia

AGENDA

Time Keep Co.

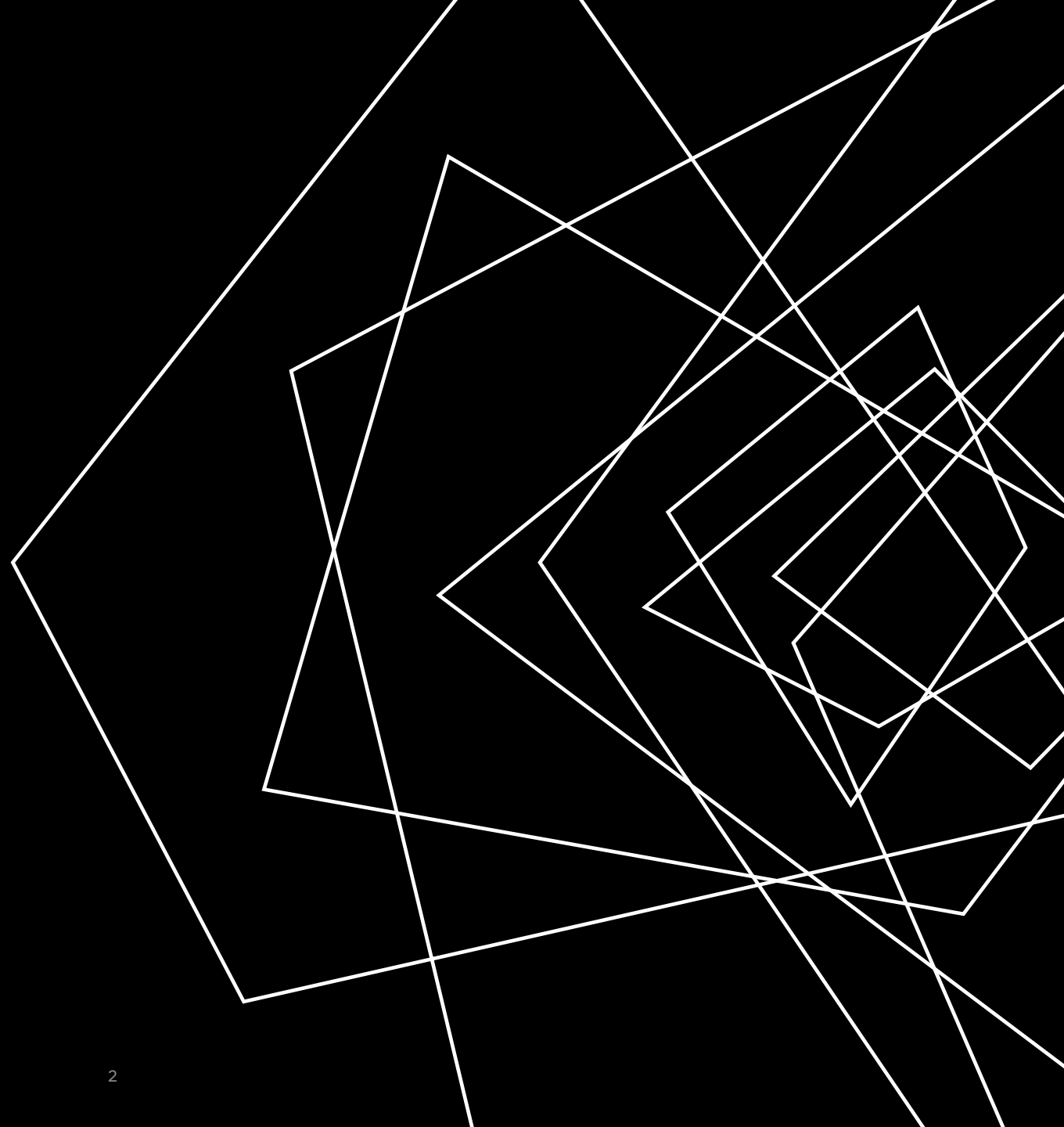
- Planning

- OOP Implementation

- Code Snippets

- Sample Use

-Reflection



PLANNING:

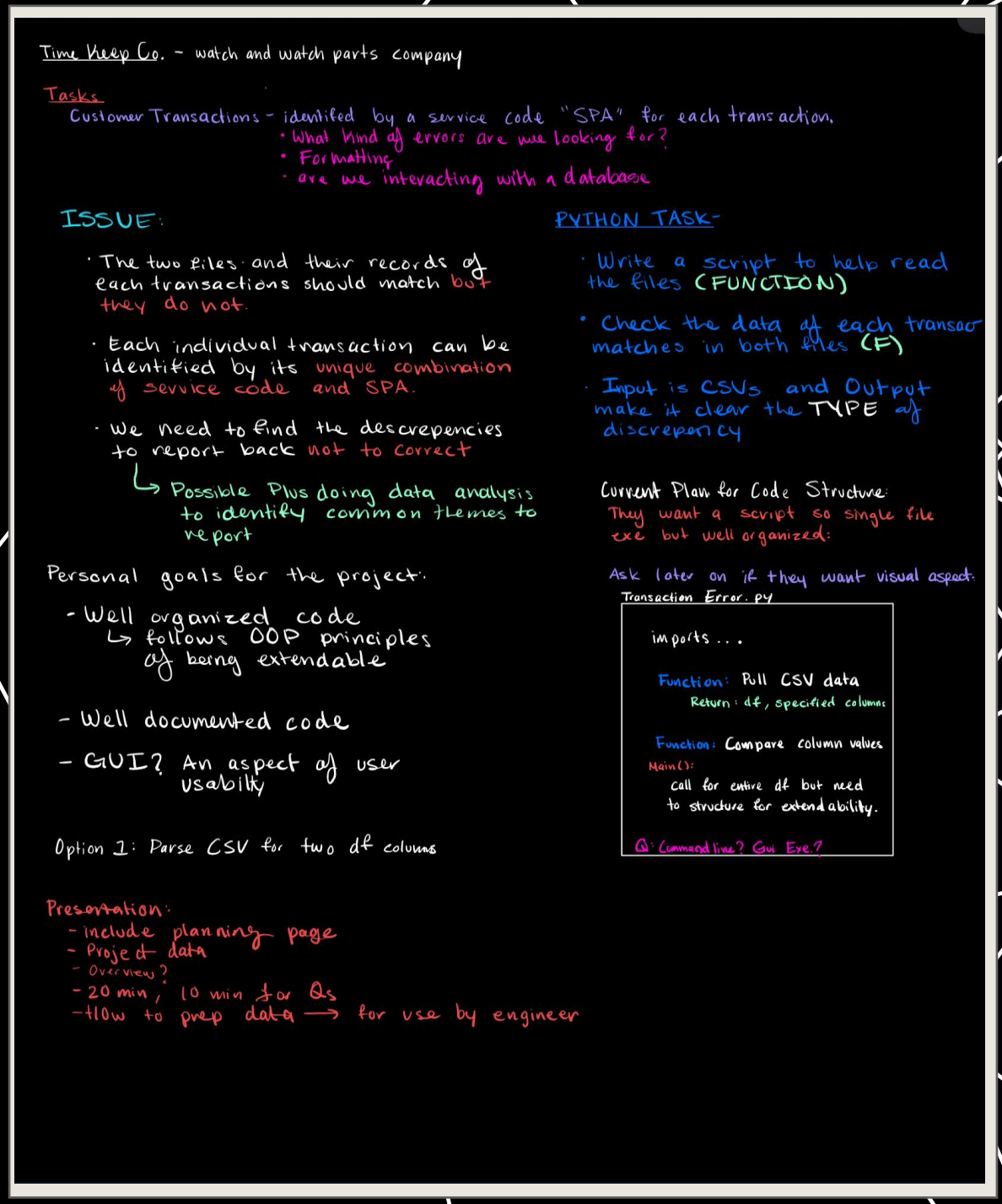
- First, need to break down the problem

- Who are we making this for, and what are the tasks and parameters of the prompt?

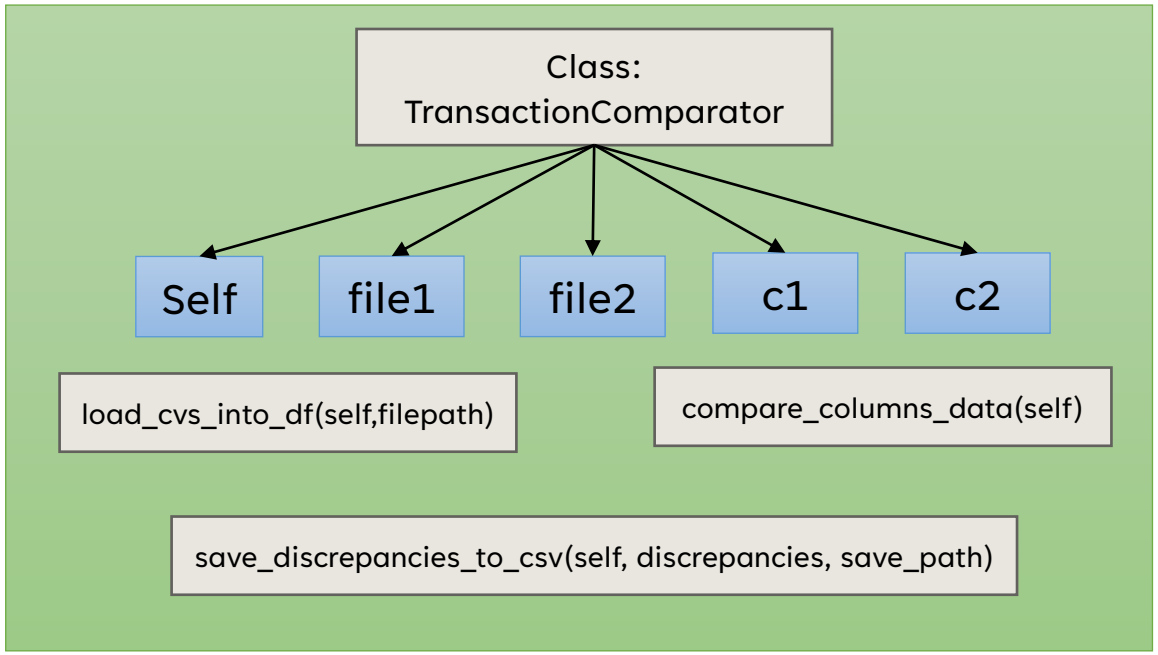
- What are my goals for the project?

- What are the individual things I want to code/do?

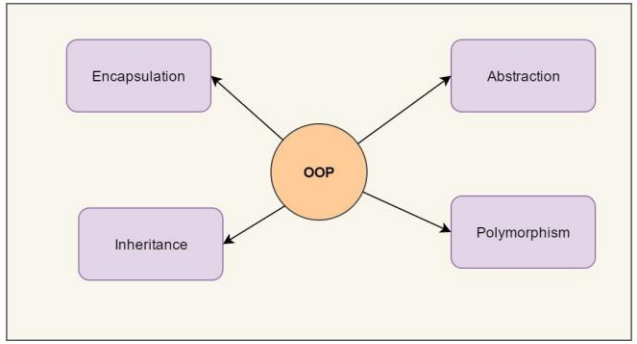
- Writing down questions for myself



OVERVIEW OF CODE STRUCTURE: OOP




```python
class TransactionComparator:
    def __init__(self, file1, file2, column1="SPA", column2="SPA"):
        self.df1 = self.load_csv_into_df(file1)
        self.df2 = self.load_csv_into_df(file2)
        self.column1 = column1
        self.column2 = column2

    def load_csv_into_df(self, csv_filepath):
        try:
            df = pd.read_csv(csv_filepath)
            return df
        except FileNotFoundError:
            raise FileNotFoundError(f"CSV file '{csv_filepath}' not found.")

    def compare_columns_data(self):
        discrepancies = {'missing_transactions': [], 'unmatched_transactions': []}

        for index, row in self.df1.iterrows():
            df1_spa_value = row[self.column1]
            df1_service_code_value = row[self.column1]

            matching_row = self.df2[(self.df2["SPA"] == df1_spa_value) & (self.df2["Service Code"] == df1_service_code_value)]

            if matching_row.empty:
                discrepancies['missing_transactions'].append(row)
            else:
                for col in self.df1.columns:
                    if row[col] != matching_row.iloc[0][col]:
                        discrepancies['unmatched_transactions'].append((row, matching_row.iloc[0]))
                        break

        for index, row in self.df2.iterrows():
            df2_spa_value = row[self.column2]
            df2_service_code_value = row[self.column2]
            matching_row = self.df1[(self.df1["SPA"] == df2_spa_value) & (self.df1["Service Code"] == df2_service_code_value)]
            if matching_row.empty:
                discrepancies['missing_transactions'].append(row)

        return discrepancies
```



Four Pillars of Object Oriented Programming

CLASS METHODS

Loading a CSV into DF

```
class TransactionComparator:
    def load_csv_into_df(self, csv_filepath):
            return df
        except FileNotFoundError:
            raise FileNotFoundError(f"CSV file '{csv_filepath}' not found.")
```

Comparing row data

```
#This compares the data of the dfs row by row and collects the missing data in a dictionary
def compare_columns_data(self):
    # The dictionary of missing transactions
    discrepancies = {'missing_transactions': []}

    # To make it easier to avoid indexing the worng column I used a set of tuples for each DataFrame based
    set_df1 = set(self.df1[[self.column1, self.column2]].itertuples(index=False, name=None))

    set_df2 = set(self.df2[[self.column1, self.column2]].itertuples(index=False, name=None))

    # Find missing transactions in df2 that are present in df1
    missing_in_df2 = set_df1 - set_df2
    # Find missing transactions in df1 that are present in df2
    missing_in_df1 = set_df2 - set_df1

    # Process missing transactions for the second file
    for missing in missing_in_df2:
        discrepancies['missing_transactions'].append(f'{self.file2}: {missing}')

    # Process missing transactions for the first file
    for missing in missing_in_df1:
        discrepancies['missing_transactions'].append(f'{self.file1}: {missing}')

    return discrepancies
```

Saving Data as CSV

```
#This saves the missing data as a CSV for the user
def save_discrepancies_to_csv(self, discrepancies, save_path):
    # Convert the discrepancies dictionary to a DataFrame
    missing_transactions = pd.DataFrame(discrepancies['missing_transactions'], columns=['Description'])
    # You may want to adjust the column name or structure based on how you want the output CSV to look

    # Save the DataFrame to a CSV file
    missing_transactions.to_csv(save_path, index=False)
```

MAIN CLASS: ARGSPARSE

```python
# This is the main method
def main():
    # We are using argsparse to handle arguements as well as present information on the program
    try:
        #Purpose
        parser = argparse.ArgumentParser(description="Compare two CSV files containing transaction records.")
        #File 1
        parser.add_argument("file1", help="Path to the first CSV file")
        #File 2
        parser.add_argument("file2", help="Path to the second CSV file")

        # This is just an option if other columns wanted to be looked at
        parser.add_argument("--column1", default="SPA", help="Name of the column to use as a key from the first CSV file (default: 'SPA')")
        parser.add_argument("--column2", default="Service Code", help="Name of the column to use as a key from the second CSV file (default: 'Service
        Code')")

        #This is how we state if we want to save the missing data as a csv
        parser.add_argument("-s", "--save", help="Path to save the discrepancies as a CSV file", nargs='?', const="discrepancies.csv", type=str)
        args = parser.parse_args()

        #We create an object from our class using user inputs
        comparator = TransactionComparator(args.file1, args.file2, args.column1, args.column2)
        #Then use the object to call on the method to compare df data and return a dictionary
        discrepancies = comparator.compare_columns_data()

        #Print statements to show data in console
        print("Discrepancies:")
        print("Missing Transactions:")
        for entry in discrepancies['missing_transactions']:
            print(entry)

        #Inform the user what the CSV is saved as if '-s' or '--save' is used
        if args.save:
            comparator.save_discrepancies_to_csv(discrepancies, args.save)
            print(f"Discrepancies saved to {args.save}")

    except FileNotFoundError as e:
        print(f"Error: {e}")
```

SAMPLE USE!

REFLECTION

- Very fun project

- Working with a lot of data makes it a lot easier to miss mistakes

- There are still bugs!

- Always room to grow