## Pattern 1: Template Method

## Location: Filehandler.py – lines 9 – 158

## Why:

Template method is used when an algorithm in a system differs slightly in their process but achieve the same goal. The algorithm differs in some steps of the process but has similarities or shared code in others. The structure of the algorithm remains the same. E.g. to open a file and retrieve the data we need to:

1. Open the file
2. Read the file contents
3. Close the file

We close the file on most file types as not closing it can cause problems with clogging up RAM with unnecessary data, or cause issues opening this file again or saving changes made under another connection possibly.

In this system we have 3 different file types to be read, these include .txt, .csv and .xlsx. These files differ in a couple different ways. First of all an xlsx file differs in two ways, the file does not need to be closed after reading, and the file can contain multiple work sheets which would be the equivalent of opening multiple .txt files, as the data can be separated into multiple sheets. The .xlsx reader also needs a certain package to open the file.

The second file type is .csv. This file type is another Microsoft Excel file type but interpreted differently to .xlsx. This file type, like .txt needs to be opened and also closed in order to avoid issues. The CSV file type is also read with a certain file reading package for that particular file type.

The third file type is .txt which like .csv needs to be opened and closed to avoid issues. The .csv and .txt files differ by reading the data differently and also using different functions to open the connect to the file.

The reason template method is applicable in this scenario is all 3 file types open a file, read that file, and in certain cases close it afterwards using different functions. However the structure of the file reading process, or algorithm, is exactly the same.

It would be possible to write default functionality in the open method which is inherited by both .csv and .txt as they are quite similar in a sense and the method could be written to decide what method to use based on the file extension. Then .xlsx could overwrite the default functionality to it’s specific needs.

All three subclasses would be required to overwrite their own read functionality as this is where file types most greatly differ from one another.

As for the close function as it is no needed for .xlsx there is no close function body written in this subclass method, this is because it is not required but instantiating a subclass without references to all methods in the base class is not following coding conventions and the language conventions.

If this system was larger in size and more functionality was written to open different file types it would make it easier to demonstrate the benefits of applying this pattern in this scenario. There may be some file types that could share the .txt reader open method functionality so code duplication would be negated in the future. The read methods would likely all be different in many ways as it is the most unique part of the algorithm per different instance. The close method is sometimes not even necessary to use but needs to be referenced in the subclasses regardless of whether it is used. It can be instantiated with no body or functionality at all.

## Pattern 2: Factory Method

## Location: chart.py – lines 7 – 143

## Why:

In the previous system if a graph is created then it is first created without specifying the nature of graph it is going to be generating. Because this graph can be created without any dependency if we create it and then give it a value to determine the subclass to create for itself, then give the graph object an invalid graph type value, this could cause issues. The graph will remain created without a specific graph type to generate. To avoid this we can get the controller to determine what sort of product it wants to produce, e.g. a pie, scatter or bar graph. If the controller is given an invalid value then no new object is created. If the value is valid then the controller fines the concrete product it wants in a dictionary. This way there is no empty object created if there was an error in the value specified.

The Controller creates an instance of a concrete graph type. These graph types are now subclasses of the Graph ABC. This concrete creator acts as an interface for the product, which in this case is the type of graph produced. The methods in the ConcreteGraph objects are used to configure what information will be displayed in the product.