Student Name: Peace Ekundayo Bakare

Title: REPORT OF ASSIGNMENT 1

Libraries Used:

1. *Math* Library for mathematical calculations
2. *Matplotlib.pyplot as peaceplt* for visualizations and graph plotting
3. *Pandas as peacepd* for exploring and manipulating data in data sets
4. *Scipy.interpolate.interp1d as peacep1d* for interpolation functions
5. *Google.colab.drive* for file storage and retrieval
6. *Datetime* for manipulating dates and times

Question 1 - Number of folds

```
[1]  import math
     import matplotlib.pyplot as peaceplt
     import pandas as peacepd
     # import numpy as peacenp
     from scipy.interpolate import interp1d as peacep1d
     from google.colab import drive
     from datetime import datetime
```

Collaborators:

1. Stephen Oduh
2. Miracle James

Question 1.

A piece of paper is 1mm thick. Assuming you can fold it as many times as you want, how many folds would it take to exceed the height of Mount Everest at 8,848 m?

Solution

Each fold of the paper adds a new layer of thickness to the paper. The get how many times to fold the paper to exceed the height of Mount Everest at 8,848 m simply requires a repetitive action of folding and measuring the height gotten. I have translated this into code by using a simple while statement.

Steps Used to solving the problem:

1. Initialized the paper thickness in meters as *paper_thickness_in_meters*, the height of Mount Everest as *mount_everest*, and count as *count*.
2. Created a new variable, *new_thickness* and assigned the *paper_thickness_in_meters,* to use as loop invariant.
3. Using while loop, while the *new_thickness* value is less than *mount_everest,* multiply the new value of *new_thickness* by 2 and increase the count by 1.
4. Return the count value, which is **24.**

Insight: Quantities grow when they double repeatedly. This is a powerful concept of compounded increase over a period of time. Though starting with a very small thickness (1 mm), repeated doubling can lead to a surprisingly large number).

```
[ ]  #convert paper thickness of 1mm to meters for easy comparison with the height of Mount Everest
     paper_thickness_in_meters = 1 * (10 ** -3)

     #create a variable and assign the height of Mount Everest to it.
     mount_everest = 8848

     #this counts the number of folds to exceed Mount Everest's height
     count = 0
```

```
[ ]  #paper thickness doubles up each time folding is done i.e. 2 layers of the original layer
     new_thickness = paper_thickness_in_meters
     while(new_thickness < mount_everest):
       new_thickness *= 2
       count += 1

     print(count)
```

```
24
```

Solution 1: Code Screenshot and result displayed

Question 2

The volume of water in a reservoir decreases at an exponential rate, following

v(t) = v(0)exp(-at) with a = 0.1. How much time, t, does it take for the volume to decrease to less than one half of its initial volume, v(0)?

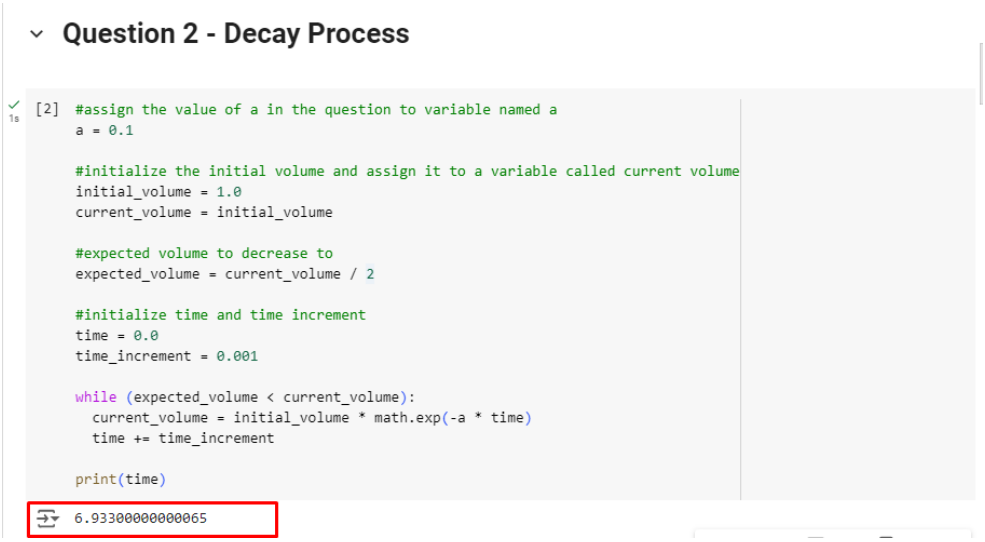
Solution

This Question is more like the previous question. The *current_volume* and time is iteratively updated until the desired condition is met. The use of small-time increments ensures a smooth and accurate simulation.

Steps Used to solving the problem:

1. Initialize the decay constant *a* as 0.1, as stated in the question. Also, initialize the volumes, *initial_volume* as 1.0, and *current_volume* as *initial_volume*.
2. Divide the *current_volume* by 2 and assign the value to a variable *expected_volume*. Assign 0.0 to variable *time* and 0.001 to variable *time_increment*. The time will be incremented by 0.001 in each loop iteration.
3. In simulation of the Decay Process, I used a while loop with a looping condition until *expected_volume* equals to or exceeds *current_volume*. *current_volume* is calculated with the mathematical formula in the question and time is incremented.
4. The time value is returned as **6.93300000000065.**

Insights: By using a small time increment of 0.001, a precise calculation of the time required for the volume to reach the expected value was gotten. Little increment helps to be precise in calculations.



Solution 2: Code Screenshot and result displayed

Question 3

If you deposit $100 in a bank account that offers an annualized interest rate of 5% (compounded annually), how much money will you have (round to the $) after one, two, three, four and five years?

Solution

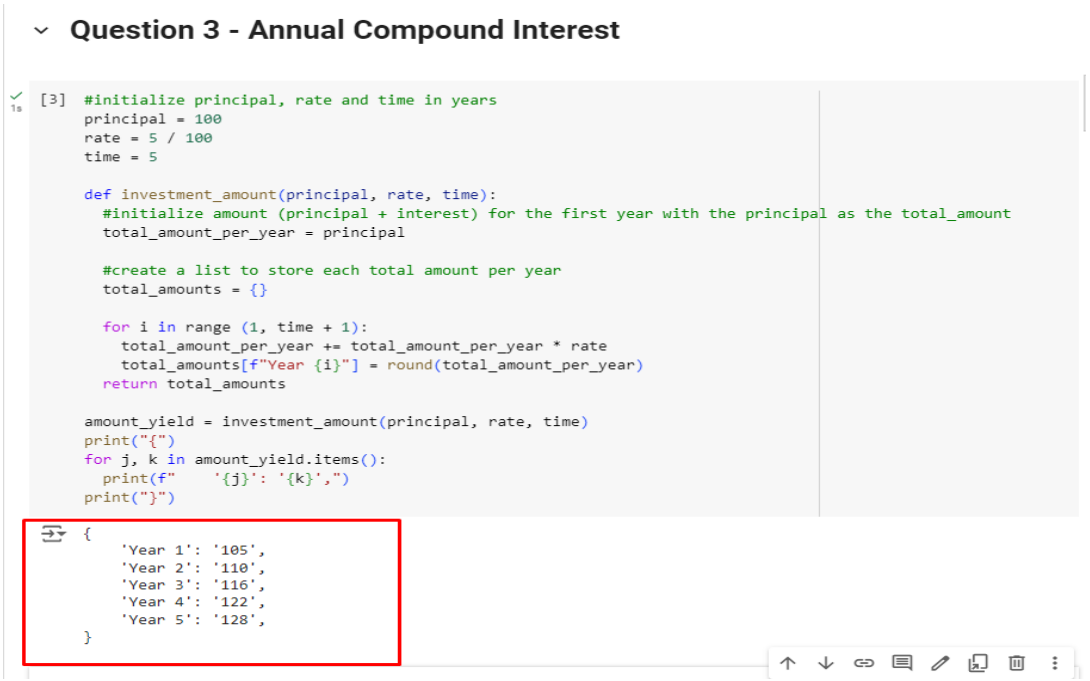
To calculate the investment growth over a period and a fixed interest rate, I have used a simple loop to solve this problem, applying the interest rate annually and a dictionary to store yearly totals to make it easier to access and print the results over the 5 years. I have defined a simple function, *investment_amount* for proper code structuring.

Steps Used to Solving the problem:

1. The initial investment *principal* is initialized to 100, the annual interest rate, *rate* is initialized to 5% (5 / 100), and the period *time* is initialized to 5. Also, I initialized the total amount for the first year, *total_amount_per_year* with the principal, and created a dictionary, *total_amounts* to store the total amount for each year.
2. I used a for-loop that runs from 1 to the number of years, *time*. In each iteration, the *total_amount* is updated by adding the interest for that year, and the result is stored in a dictionary.
3. In addition to printing the result i.e. the dictionary, I have formatted the print method to return a visually appealing result.

The image below shows the code and result.

Insights: The solution reveals the power of compounded interests over a number of years.



Question 4

Suppose you want to buy a car worth $20,000. A financial institution can provide a loan with a monthly interest rate of 1%. What is the monthly payment to pay off the debt in one, two and three years (rounded to the nearest $)?

Solution

My approach ensures that the loan balance is updated monthly, accounting for the interest accrual. By recalculating the monthly payment at the end of each year, I provided a more accurate reflection of the repayment schedule.

Steps Used to Solving the problem:

1. Initializing the principal amount, *principal* to $20,000, annual interest rate, *rate* to 1% (1 / 100), and loan duration, *time* to 3.
2. A function, *loan_repayment* to calculate the yearly repayment amounts. A loop runs from 1 to the number of years, *time* and for each year, the number of months, *month_per_year* is calculated.
3. The initial monthly payment is calculated by dividing the principal, *principal* by the number of months in the year. For each month, the interest is calculated and added to the balance, while the monthly payment is subtracted from the balance.

4. After updating the balance for all months, the final monthly payment is recalculated and stored in the *yearly_repayment* dictionary, rounded to the nearest integer.
5. The function returns the *yearly_repayment* dictionary in a formatted manner.

Insights: The solution provides a clear and systematic way to calculate loan repayments over time.

Question 4 - Loan monthly repayment

```
principal = 20000
rate = 1 / 100
time_in_years = 3

def loan_repayment(principal, rate, time):
  yearly_repayment = {}

  for i in range (1, time + 1):
    month_per_year = i * 12
    balance_from_loan = principal
    monthly_payment = 0

    monthly_payment = principal / month_per_year
    for a in range (month_per_year):
      interest = balance_from_loan * rate
      balance_from_loan += interest - monthly_payment

    monthly_payment = (principal + balance_from_loan) / month_per_year

    yearly_repayment[f"Year {i}"] = round(monthly_payment)

  return yearly_repayment

loan_payment = loan_repayment(principal, rate, time_in_years)
print("{")
for j, k in loan_payment.items():
  print(f"    '{j}': '{k}',")
print("}")
```

```
{
    'Year 1': '1783',
    'Year 2': '955',
    'Year 3': '686',
}
```

Question 5

You are about to set up a new business and will invest $100,000. On day one you expect to have 100 customers and the number of customers will grow at a rate of 1% per day. If each customer provides profits of $10, how many days will it take to repay your initial investment based on cumulated profits? Plot cumulated profits per day, show initial investment and mark breakeven day.

Solution

The solution simulates profit growth and identifies the breakeven point, making it easy to understand how customer growth impacts profitability. The graph provides a clear visual representation of the business's financial progress.

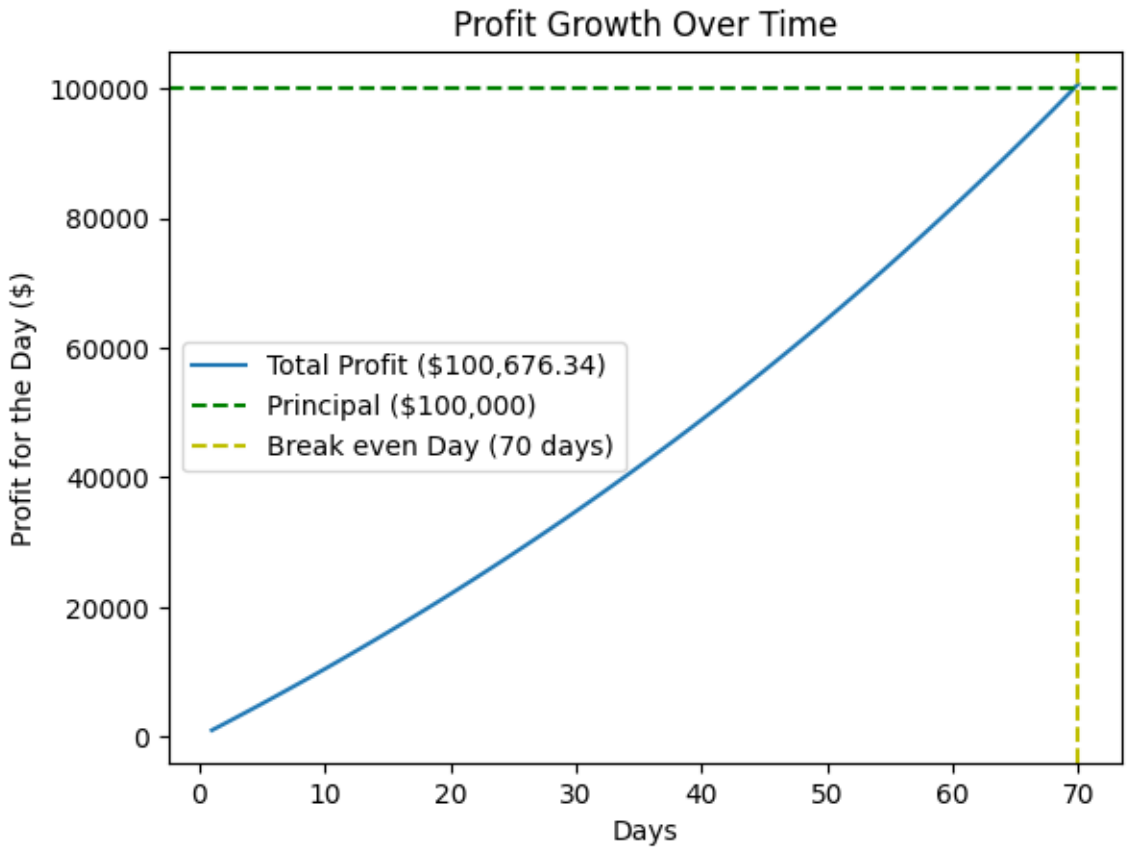
Steps Used to solving the problem:

1. Initialization of variables, *principal* as $100,000, *starting_customers* as 100, *customer_growth_rate_per_day* as 1% (1 / 100), *profit_from_each_customer* as 10, *breakeven_day* as None, *profit* as 0, *number_of_customers* with *starting_customers*. Also, I created 2 lists, *days* and *profits,* which are values plotted against each other in the graph.
2. Starting with zero profit, I loop through days, each day calculating the profit from the current number of customers. I track the progress of profit by storing daily profit and day number in lists. I also checked if the profit meets or exceeds $100,000 record, the breakeven day while growing the number of customers by 1% each day.
3. I have plotted a graph using the matplotlib library. The Graph has a title of "Profit Growth over Time", uses a legend, shows the increase in profit as the days increased. Appropriate colors have been used to enhance graph readings.

Question 5 - Business Investment

```
principal = 100000
starting_customers = 100
customer_growth_rate_per_day = 1 / 100
profit_from_each_customer = 10
breakeven_day = None

profit = 0
number_of_customers = starting_customers
day = 1
last_profit = 0
days = []
profits = []

peaceplt.ion()

while profit < principal:
  profit += number_of_customers * profit_from_each_customer

  last_profit = profit
  profits.append(profit)
  days.append(day)

  if(profit >= principal and breakeven_day is None):
    breakeven_day = day

  number_of_customers += number_of_customers * customer_growth_rate_per_day
  day += 1

#Plot Graph
peaceplt.plot(days, profits, label=f"Total Profit (${last_profit:,.2f})")
peaceplt.axhline(y=principal, color='g', linestyle='--', label="Principal ($100,000)")
peaceplt.axvline(x=breakeven_day, color='y', linestyle='--', label=f"Break even Day ({breakeven_day} days)")

peaceplt.xlabel("Days")
peaceplt.ylabel("Profit for the Day ($)")
peaceplt.title("Profit Growth Over Time")
peaceplt.legend()

peaceplt.draw()
peaceplt.pause(0.5)

peaceplt.show()

print(days)
```

Insight: The solution highlights how even a small daily growth rate of 1% in customer numbers can significantly impact overall profit over time. This demonstrates the power of compounding growth in a business context. We must understand to leverage customer growth to achieve profitability.



Question 6

Using data from http://bit.ly/1JJyf29 and linear interpolation, estimate the dates when the number of cases and deaths due to Ebola exceeded 100, 500, 1000, 2000 and 5000. Graph the cases and deaths (observations and interpolations) and mark the dates when thresholds were exceeded with a circle.

Solution

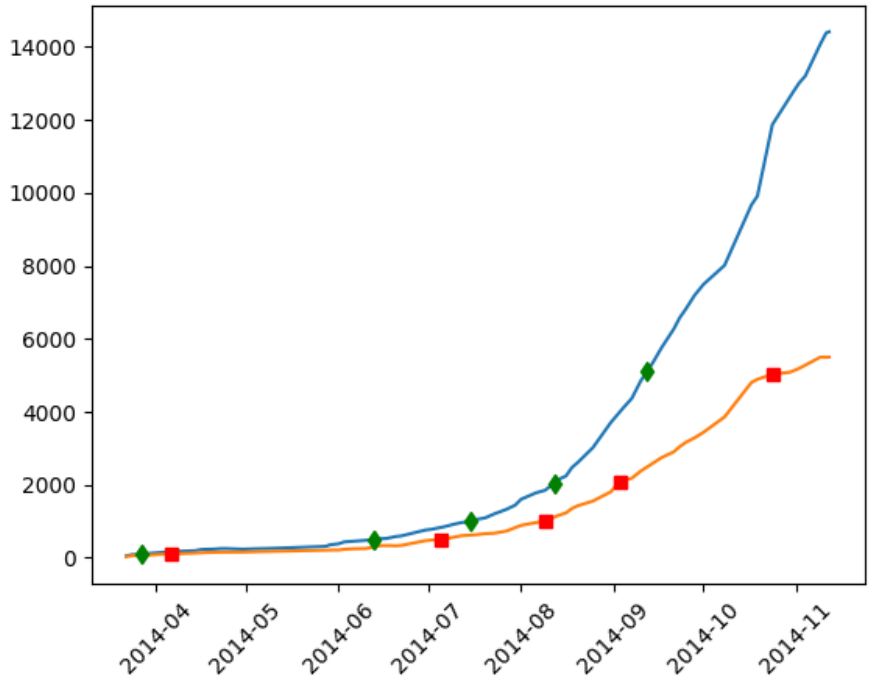
I have employed a thorough analysis of the Ebola data, making it easier to visualize and understand the trends and key events for completeness and easy visualization. I have also output the Threshold dates as required in the question.

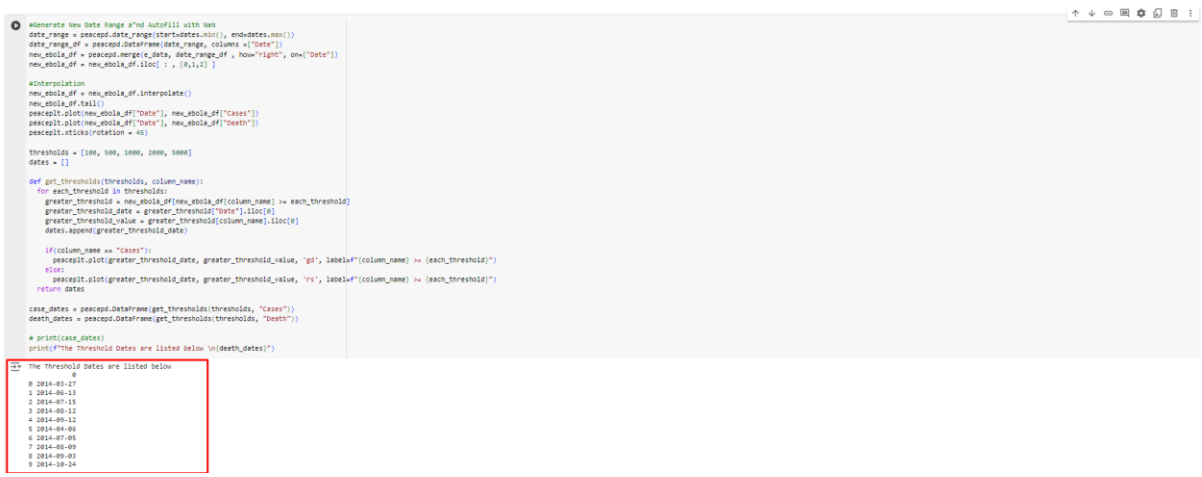
Steps Used in solving the problem

1. I read the data using the *read_excel()* function from the pandas library and extracted the first 5 rows of the Date, Death and Cases column using the *head()* function of the pandas library.

2. I generated a complete date range, creating a new date range from the minimum to the maximum data in the dataset. This ensures that all dates within the range are accounted for, even if some dates are missing in the original data.
3. I merged DataFrames. The new date range is then merged with the original data *e_data*. This helps to fill in any missing dates with NaN values, ensuring a continuous timeline.
4. I used the *interpolate()* method in the pandas library, to fill in all the missing values, NaNs in the dataset. This helps to create a smooth and continuous dataset for analysis.
5. Plotted the number of Cases and Death over time, with the axis labelled for easy and better readability.
6. The *get_thresholds()* method helps to identify the dates when the number of Cases and Deaths exceed the specified thresholds. These are then plotted on the graph with distinct markers.

Insight: The graph will be valuable for public health officials, researchers and policymakers to manage future outbreaks of pandemics as seen in the Ebola case.





Question 7

Using data from 2014, downloaded in the previous question, what is the average growth rate per day, as a percentage, in the number of Ebola cases and deaths?

Solution

The section is a continuation of Question 6. It provides a clear and concise way to analyze the rate of change in Ebola Cases and Deaths, offering valuable insights into the outbreak dynamics.

Steps Used to solving the problem

1. Calculated the percentage chage for the number of Cases and Death using the *pct_change()* method in the pandas library. This helps to understand the rate at which Cases and Deaths are increasing or decreasing over time.
2. I utilized the *head()* function in the pandas library to display the first few rows of the percentage change DataFrame *pct_case_death*. This gives a quick look at the initial changes in Cases and Deaths.
3. An empty dictionary *average* was created to store the mean percentage changes. The mean percentage change for Cases and Deaths is calculated and multiplied by 100 to convert it to a percentage and thereby stored in the dictionary.

The Two numbers required as results are displayed in the image attached below.

Insights: Comparing the average percentage change between Cases and Deaths revealed important patterns. For example, lower death than that of cases suggest improvements in treatment or healthcare response.

Question 7 - Average Growth Rate per day

+ Code + Text

```
[ ] pct_cases_death = new_ebola_df[["Cases", "Death"]].pct_change()
    pct_cases_death.head()

    average = {}

    average['Cases'] = pct_cases_death.mean()['Cases'] * 100
    average['Death'] = (pct_cases_death.mean()['Death'] * 100)

    print(average)
```

```
{'Cases': 2.5065218916499954, 'Death': 2.3306080833679683}
```

Question 8

Using the same date, plot the number of deaths versus the number of cases and estimate the average ratio of Ebola deaths to cases.

Solution

This solution provides a visual representation of the number of cases and deaths helping to better understand the problem.

Steps Used to solving the problem

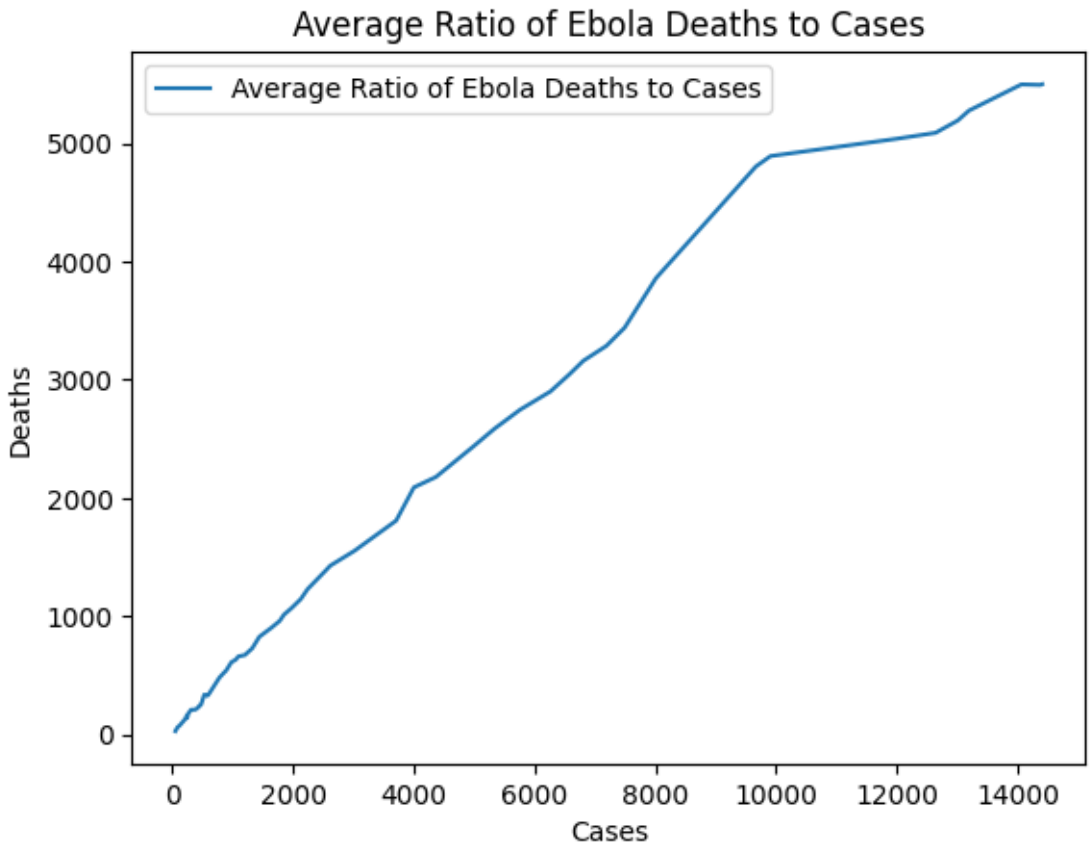
1. Created a new DataFrame *death_cases* that includes only the columns for Cases and Death from the original dataset. This focuses the analysis on the most relevant data.
2. A new column, "*Death Ratio"* is added to the DataFrame. This column calculates the ratio of deaths to Cases for each entry, providing a measure of the outbreak.
3. The mean of the "Death Ratio" column is calculated and stored in the variable *death_ratio*. This gives an overall average ratio of deaths to cases across the dataset.
4. Plotted the number of cases against the number of deaths, with a label indicating the average ratio of Ebola deaths to cases. This visual representation helps in understanding the relationship between cases and deaths. In addition, labels for the x-axis (Cases) and y-axis (Deaths) are added, along with a title for the plot. A legend is also included to explain the plotted data.

Insights: The graph will be valuable for public health officials, researchers and policymakers to manage future outbreaks of pandemics as seen in the Ebola case.

Question 8 - Deaths versus Cases

```
[ ]  death_cases = new_ebola_df[["Cases", "Death"]]

     death_cases['Death Ratio'] = (death_cases['Death'] / death_cases['Cases'])
     death_ratio = death_cases['Death Ratio'].mean()
     print(death_ratio)

     peaceplt.plot(death_cases['Cases'], death_cases['Death'], label=f"Average Ratio of Ebola Deaths to Cases")

     peaceplt.xlabel("Cases")
     peaceplt.ylabel("Deaths")
     peaceplt.title("Average Ratio of Ebola Deaths to Cases")
     peaceplt.legend()

     peaceplt.draw()
     peaceplt.pause(0.5)

     peaceplt.show()
```

```
0.5577992908998353
```



Question 9

Obtain daily prices for two ETFs called SPY and TLT which track the S&P500 index and long-term Treasury Bond. Select the adjusted closing prices. Plot the two time series during 12/31/2013 – 08/31/2015 and make them comparable by starting from prices of $100 on the first day in 12/31/2013 – 08/31/2015.

Solution

Steps Used to Solving the problem:

1. Read the 2 CSV files containing financial data for the S&P 500 index (SPY) and Long-Term Treasury Bonds (TLT). This data is loaded into DataFrames *spy_data* and *tlt_data*.
2. Only the 'Date' and 'Adj Close' columns are selected. This focuses the analysis on the adjusted closing prices, which account for dividends and stock splits. Then a function *adjust_close_to_hundred()* is defined to normalize the adjusted closing prices. This function scales the prices so that the first day's price is set to 100. This makes it easier to compare the performance of the two assets over time.
3. The normalization function is applied to both the SPY and TLT DataFrames, creating new DataFrames *spy_adj_close_df* and *tlt_adj_close_df* with the normalized prices.
4. Plotted the normalized prices of SPY and TLT over time. This visual comparison helps in understanding how the two assets have performed relative to each other. Labels for the x-axis (Date) and y-axis (Prices) are added, along with a title for the plot. A legend is also included to distinguish between the two assets. The x-axis labels are rotated for better readability.

Insight: Observing the fluctuations in the normalized prices can give you an idea of the volatility of each asset. Typically, stocks (SPY) are more volatile compared to bonds (TLT), which are generally considered safer investments.

Question 9

```
# Read data
spy_data = pacecpd.read_csv('/content/drive/My Drive/Colab Notebooks/SPY.csv')
tlt_data = pacecpd.read_csv('/content/drive/My Drive/Colab Notebooks/TLT.csv')

# spy_data.head()
adj_close_df = spy_data[['Date', 'Adj Close']]

def adjust_close_to_hundred(df):
  df['adjusted'] = df['Adj Close'] / df['Adj Close'].iloc[0] * 100
  return df

spy_adj_close_df = adjust_close_to_hundred(adj_close_df)
spy_adj_close_df.head()
```

```
<ipython-input-16-036583106204>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['adjusted'] = df['Adj Close'] / df['Adj Close'].iloc[0] * 100
```

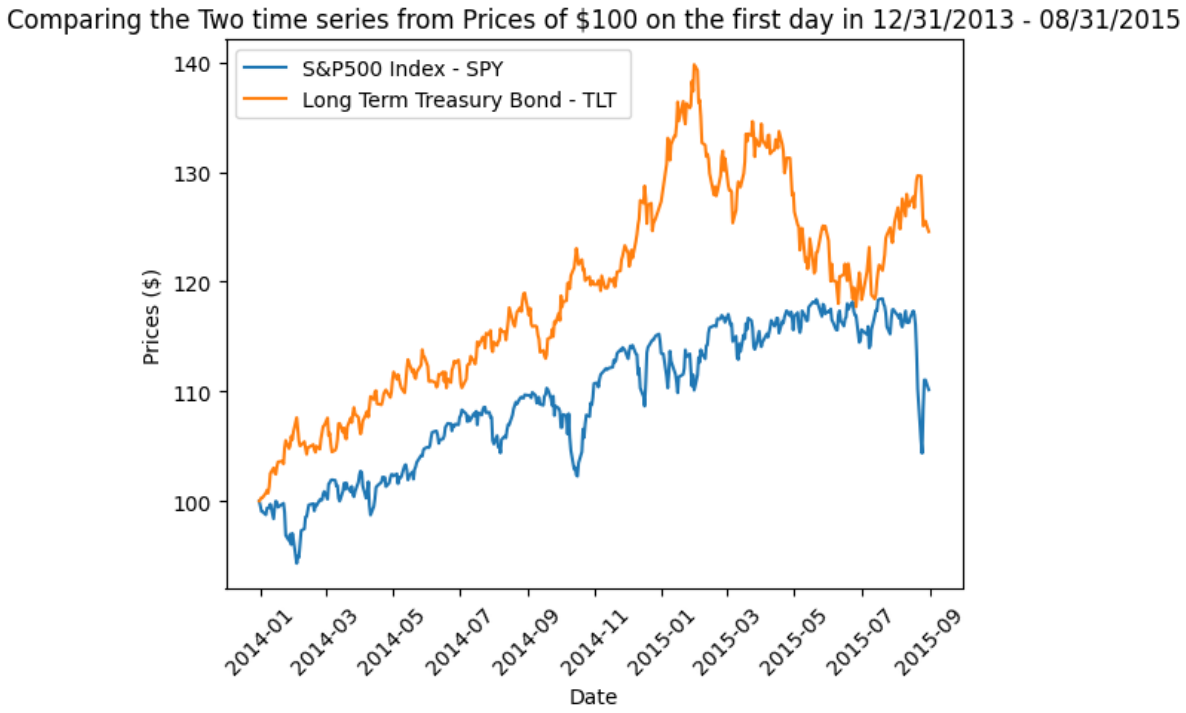
	Date	Adj Close	adjusted
0	2013-12-31	157.853638	100.000000
1	2014-01-02	156.340836	99.041643
2	2014-01-03	156.315201	99.025403
3	2014-01-06	155.862137	98.738388
4	2014-01-07	156.819412	99.344820

Next steps: Generate code with spy_adj_close_df | View recommended plots | New interactive sheet

```
# tlt_data.head()
tlt_adj_close_df = tlt_data[['Date', 'Adj Close']]

tlt_adj_close_df = adjust_close_to_hundred(tlt_adj_close_df)
```

```
<ipython-input-16-036583106204>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['adjusted'] = df['Adj Close'] / df['Adj Close'].iloc[0] * 100
```

```
# Plot the graph
pacecplt.plot(pacecpd.to_datetime(spy_adj_close_df['Date']), spy_adj_close_df['adjusted'], label="S&P500 Index - SPY")
pacecplt.plot(pacecpd.to_datetime(tlt_adj_close_df['Date']), tlt_adj_close_df['adjusted'], label="Long Term Treasury Bond - TLT ")
```



Question 10

For the ETFs on the previous question, calculate daily returns, r(t) = p(t)/p(t-1)-1, for each trading day in the same time period as above. Calculate the average, min and max daily return for each of the two ETFs during the time period and express these as percentages.

Solution

Steps Used to solving the problem

1. Calculated the daily percentage change in the adjusted closing prices for both the S&P 500 Index (SPY) and Long-Term Treasury Bonds (TLT). This helps in understanding the daily returns of these assets.
2. The *describe*() method is used to generate descriptive statistics for the percentage changes. This includes metrics like mean, standard deviation, minimum, maximum, and quartiles.
3. The descriptive statistics for both SPY and TLT are printed out, providing a summary of the daily returns for each asset.

Insight: By comparing the descriptive statistics of SPY and TLT, you can assess which asset has higher average returns and which one is more volatile. Typically, stocks (SPY) might have higher returns but also higher volatility compared to bonds (TLT).

Question 10

```
spy_data_stats = spy_data['Adj Close'].pct_change()
spy_data_description = spy_data_stats.describe()


tlt_data_stats = tlt_data['Adj Close'].pct_change()
tlt_data_description = tlt_data_stats.describe()

print(f"Average, Min, Max and general description of S&P500 Data \n {spy_data_description}")
print("\n")
print(f"Average, Min, Max and general description of Long-Term Treasury Bond Data \n {tlt_data_description}")
```

```
Average, Min, Max and general description of S&P500 Data
 count    419.000000
mean       0.000263
std        0.008007
min       -0.042107
25%       -0.003915
50%        0.000546
75%        0.004764
max        0.038394
Name: Adj Close, dtype: float64


Average, Min, Max and general description of Long-Term Treasury Bond Data
 count    419.000000
mean       0.000560
std        0.008419
min       -0.024325
25%       -0.004949
50%        0.001003
75%        0.006242
max        0.026469
Name: Adj Close, dtype: float64
```

References

[1] Matplotlib, “Matplotlib: Python plotting — Matplotlib 3.1.1 documentation,” Matplotlib.org, 2012. Available: https://matplotlib.org/

[2] Pandas, “Python Data Analysis Library,” Pydata.org, 2018. Available: https://pandas.pydata.org/

[3] CueMath, “Compound Interest | Formulas, Derivation & Solved Examples,” Cuemath. Available: https://www.cuemath.com/commercial-math/compound-interest/

[4] Finance Formulas, “Loan Payment Formula (with Calculator),” financeformulas.net. Available: https://financeformulas.net/Loan_Payment_Formula.html