FE 520 Final Report

By: Jordan Braycewski, Anith Joy, Angelina Rabbia and Vishal Sukumar

(1) What does the package do?

This python package was made to aid in calculating how much value either a cash flow in the future is worth today or how much a present cash flow will be worth at some time in the future. There are many different analysis tools in this package to be able to analyze many different cash flows which will allow our users to have a better understanding of any of the cash flows they are researching.

(2) What does each function do to serve the package?

pv\_one: This function aids in calculating cash flows by allowing the user to easily calculate the present value of a future single sum

fv\_one: This function aids in calculating cash flows by allowing the user to easily calculate the future value of a current single sum

pv\_ordinary\_annuity: This function aids in calculating cash flows by allowing the user to easily calculate the present value of a future ordinary annuity

fv\_ordinary\_annuity: This function aids in calculating cash flows by allowing the user to easily calculate the future value of a current ordinary annuity

pv\_annuity\_due: This function aids in calculating cash flows by allowing the user to easily calculate the present value of a future annuity due

fv\_annuity\_due: This function aids in calculating cash flows by allowing the user to easily calculate the future value of a current annuity due

growing\_perpetuity: Calculates the present value of a growing perpetuity based on the initial cash flow, discount rate, and growth rate.

annuity\_payment: Calculates the periodic payment for a loan or annuity given the principal amount, interest rate, and number of periods.

cash\_flow\_analysis: Performs cash flow analysis on potential investment projects by calculating the net present value (NPV) based on the initial investment, a list of future cash flows, and a discount rate.

calculate\_PayBackPeriod: Calculates the payback period for an investment based on the initial investment and annual cash flow.

calculate\_probabilityIndex: Calculates the probability index for an investment based on the initial investment, a list of cash inflows over time, and a discount rate.

calculate\_loanPayment: Calculates the periodic loan payment based on the principal amount, annual interest rate, and the number of payments.

calculate\_totalInterestPaid: Calculates the total interest paid over the life of a loan based on the periodic loan payment, total number of payments, and initial loan amount.

calculate\_totalPayments: Calculates the total payments made over the life of a loan based on the periodic loan payment and total number of payments.

(3) What's the connection between each function?

**pv\_one and fv\_one:**

* pv\_one calculates the present value of a future single sum, while fv\_one calculates the future value of a current single sum.
* Together, they provide a way to analyze the time value of money for a single cash flow.

**pv\_ordinary\_annuity and fv\_ordinary\_annuity:**

* pv\_ordinary\_annuity calculates the present value of a future ordinary annuity.
* fv\_ordinary\_annuity calculates the future value of a current ordinary annuity.
* These functions are useful for evaluating cash flows that occur at regular intervals.

**pv\_annuity\_due and fv\_annuity\_due:**

* pv\_annuity\_due calculates the present value of a future annuity due.
* fv\_annuity\_due calculates the future value of a current annuity due.
* An annuity due is a series of equal payments occurring at the beginning of each period.

**growing\_perpetuity:**

* Calculates the present value of a growing perpetuity based on the initial cash flow, discount rate, and growth rate.
* Useful for evaluating perpetuities with a constant growth rate.

**annuity\_payment:**

* Calculates the periodic payment for a loan or annuity given the principal amount, interest rate, and number of periods.
* Useful for determining regular payments needed to satisfy a loan or achieve a specific financial goal.

**cash\_flow\_analysis:**

* Performs cash flow analysis on potential investment projects.
* Calculates the net present value (NPV) based on the initial investment, a list of future cash flows, and a discount rate.

**calculate\_PayBackPeriod:**

* Calculates the payback period for an investment based on the initial investment and annual cash flow.
* Provides insight into the time it takes for the investment to recoup its initial cost.

**calculate\_probabilityIndex:**

* Calculates the probability index for an investment based on the initial investment, a list of cash inflows over time, and a discount rate.
* Offers a measure of the desirability of an investment project, considering both cash inflows and the time value of money.

**calculate\_loanPayment:**

* Calculates the periodic loan payment based on the principal amount, annual interest rate, and the number of payments.
* Useful for determining regular loan payments.

**calculate\_totalInterestPaid:**

* Calculates the total interest paid over the life of a loan based on the periodic loan payment, total number of payments, and initial loan amount.
* Provides insights into the cost of borrowing over the loan term.

**calculate\_totalPayments:**

* Calculates the total payments made over the life of a loan based on the periodic loan payment and total number of payments.
* Offers a comprehensive view of the overall financial commitment of the loan.

(4) What are the potential errors the user could create, how to handle these errors?

Some of the potential errors that the user could create when using our python package are improper data entry or misunderstanding of the functions calculations. Firstly, if there is a user that enters an invalid number, for example, if a percentage value is not a decimal value less than or equal to one and greater than or equal to zero there is a pair of try and except statements to ensure that the user understands why the function did not run and then prompt the user to enter the values again. This would also be done for any monetary values or percentages that are negative, a prompt would pop up letting the user know that their values did not work in the function, and then it gives them the reason why their values did not work so they can fix their code for the next time they use the function.

(5) Implement on a dataset (real or simulated data)

**Function Definitions:**

**Present Value (PV) of a Single Sum:**

* + Method: pv\_one(fv, rate, periods)
  + Takes in: fv (future value, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: present value in float
* Implementation:
* result = finpy.pv\_one(10000, 0.1, 5)
* output: 6209.21
* **Future Value (FV) of a Single Sum:**
  + Method: fv\_one(pv, rate, periods)
  + Takes in: pv (present value, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: future value in float

Implementation:

result1 = finpy.fv\_one(1000, 0.1, 2)

Output: 1210.0

* **Present Value (PV) of an Ordinary Annuity:**
  + Method: pv\_ordinary\_annuity(pmt, rate, periods)
  + Takes in: pmt (annuity payment, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: present value in float
* **Future Value (FV) of an Ordinary Annuity:**
  + Method: fv\_ordinary\_annuity(pmt, rate, periods)
  + Takes in: pmt (annuity payment, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: future value in float

Implementation:

result2 = finpy.pv\_ordinary\_annuity(1, 0.08, 30)

output: 11.26

result3 = finpy.fv\_ordinary\_annuity(10000, 0.1, 30)

output: 1644940.23

* **Present Value (PV) of an Annuity Due:**
  + Method: pv\_annuity\_due(pmt, rate, periods)
  + Takes in: pmt (annuity payment, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: present value in float
* **Future Value (FV) of an Annuity Due:**
  + Method: fv\_annuity\_due(pmt, rate, periods)
  + Takes in: pmt (annuity payment, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: future value in float

Implementation:

result5 = finpy.pv\_annuity\_due(1000, 0.05, 10)

output: 8107.82

result6 = finpy.fv\_annuity\_due(1200, 0.08, 6)

output: 9507.36

* **Present Value (PV) of a Growing Annuity:**
  + Method: growing\_annuity(initial\_cash\_flow, rate, growth\_rate, periods)
  + Takes in: initial\_cash\_flow (initial cash flow, float), rate (discount rate, float between 0 and 1), growth\_rate (growth rate, float between 0 and 1), periods (number of periods, int)
  + Returns: present value in float

Implementation:

result7 = finpy.growing\_annuity(12000, 0.08, 0.03, 45)

output: 211567.35

* **Present Value (PV) of a Perpetuity:**
  + Method: perpetuity(cash\_flow, rate)
  + Takes in: cash\_flow (annual cash flow, float), rate (discount rate, float)
  + Returns: present value in float
* **Present Value (PV) of a Growing Perpetuity:**
  + Method: growing\_perpetuity(initial\_cash\_flow, rate, growth\_rate)
  + Takes in: initial\_cash\_flow (initial cash flow, float), rate (discount rate, float), growth\_rate (growth rate, float)
  + Returns: present value in float

Implementation:

result8 = finpy.perpetuity(30000, 0.08)

output: 375000.0

result9 = finpy.growing\_perpetuity(200000, 0.07, 0.02)

output: 4000000.0

* **Net Present Value (NPV) of a Cash Flow Analysis:**
  + Method: cash\_flow\_analysis(initial\_investment, cash\_flows, discount\_rate)
  + Takes in: initial\_investment (initial investment, float), cash\_flows (list of cash flows, each element as float), discount\_rate (discount rate, float)
  + Returns: net present value in float

Implementation:

result10 = finpy.cash\_flow\_analysis(1000, [500, 500, 500], 0.1)

output: 243.43

* **Payback Period:**
  + Method: payback\_period(initial\_investment, annual\_cash\_flow)
  + Takes in: initial\_investment (initial investment, int/float), annual\_cash\_flow (annual cash flow, int/float)
  + Returns: payback period in float

Implementation:

result11 = finpy.payback\_period(10000, 4000)

output: 2.5

* **Probability Index:**
  + Method: probability\_index(initial\_investment, cash\_flow, discount\_rate)
  + Takes in: initial\_investment (initial investment, int/float), cash\_flow (list of cash flows, each element as float), discount\_rate (discount rate, float)
  + Returns: probability index in float

Implementation:

result12 = finpy.probability\_index(200, [200, 800, -800], 0.1)

output: 1.21

* **Annuity Payment:**
  + Method: annuity\_payment(principal, rate, periods)
  + Takes in: principal (loan amount, float), rate (interest rate, float between 0 and 1), periods (number of periods, int)
  + Returns: periodic payment in float
* **Total Interest Paid:**
  + Method: total\_interest\_paid(loan\_payment, num\_payments, principal)
  + Takes in: loan\_payment (periodic loan payment, float), num\_payments (number of payments, int/float), principal (loan amount, int/float)
  + Returns: total interest paid in float
* **Total Payments:**
  + Method: total\_payments(loan\_payment, num\_payments)
  + Takes in: loan\_payment (periodic loan payment, float), num\_payments (number of payments, int/float)
  + Returns: total payments in float

Implementation:

result13 = finpy.annuity\_payment(60000, 0.03/12, 60)

output: 1078.12

result14 = finpy.total\_interest\_paid(10000, 6, 36)

output: 59964

result15 = finpy.total\_payments(10000, 6)

output: 60000

**Error Handling:**

**Input Validation:**

The input arguments must be valid in order for the function to return a valid output so we have implemented an input validation to check whether the input values for financial calculations are valid. It checks if the provided values are positive and within the valid range, such as the interest rate being between 0 and 1 (expressed as a decimal).

Code Snippets for Input Validation:

if pv < 0 or rate < 0 or periods < 0:

raise ValueError("Please enter positive values for pv, rate, and periods.")

if not 0 <= rate <= 1:

raise ValueError("Rate should be expressed as a decimal between 0 and 1.")

**Exception Handling:**

We have implemented try-except blocks to catch potential errors during calculations. If an error occurs, it raises a ValueError with a specific error message.

Code Snippet for Exception Handling:

try:

# code that might raise an exception

except ValueError as ve:

return f"Error: {ve}"

**Type Checking:**

The type-checking code checks the data types of input parameters in some functions. For example, it ensures that the initial investment is either an integer or a float.

Code Snippet for Type Checking:

if not (isinstance(initial\_investment, int) or isinstance(initial\_investment, float)):

raise TypeError("Initial investment must be in type Int or Float")

**Specific Error Messages:**

This code snippet provides specific error messages to help users understand what went wrong. For instance, it specifies that the discount rate should be expressed as a decimal between 0 and 1.

Code Snippet for Specific Error Messages:

raise ValueError("Discount rate should be expressed as a decimal between 0 and 1. Eg- 0.10 for 10%")

(6) Contributions for each team member (This will be the important factor for individual final grades).

Jordan Braycewski: created pv\_one, pv\_ordinary\_annuity, pv\_annuity\_due, fv\_one, fv\_ordinary\_annuity, fv\_annuity\_due functions for the project, as well as helped complete part of the presentation and the report.

Vishal Sukumar: Created perpetuity, growing\_perpetuity, annuity\_payment and cash\_flow\_analysis functions for the project and worked on completing the report .

Anith Joy: Created calculate\_PayBackPeriod, calculate\_probabilityIndex, calculate\_total\_interestPaid, calculate\_total\_payments functions for the project, and helped to complete some parts of the presentation.

Angelina Rabbia: created annuity\_payment function, compiled functions into package, wrote documentation for the project