Introduction to Programming with Scientific Applications
(Spring 2024)

Final project

Study ID	Name	% contributed
202107872	**Manuela Skov Thomasen**	**33%**
202104577	**Katrine Ott Frendorf**	**34%**
202105178	**Louie Mølgaard Hessellund**	**33%**

Briefly state the contributions of each of the group members to the project

We have mostly sat together working on the code and the report interchangeably, while providing editorial suggestions and corrections along the way. Katrine and Louie were responsible for correcting the code according to the PEP8 guidelines, while Manuela took care of the construction of the data frame, as well as conversion to a csv file. Manuela and Katrine worked together on writing the final report, while Louie set up a script for analysis of our data in R.

Note on plagiarism

Since the evaluation of the project report and code will be part of the final grade in the course, **plagiarism in your project handin will be considered cheating at the exam**. Whenever adopting code or text from elsewhere you must state this and give a reference/link to your source. It is perfectly fine to search information and adopt partial solutions from the internet – actually, this is encouraged – but always state your source in your handin. Also discussing your problems with your project with other students is perfectly fine, but remember each group should handin their own solution. If you are in doubt if you solution will be very similar to another group because you discussed the details, please put a remark that you have discussed your solution with other groups.

For more Aarhus University information on plagiarism, please visit

http://library.au.dk/en/students/plagiarism/

202104577 Katrine Ott Frendorf
202105178 Louie Mølgaard Hessellund
202107872 Manuela Skov Thomasen

Data and code availability

All referenced code and results are available in the following GitHub repository:

https://github.com/mthomasen/IPSA_project_handin.git

Introduction

Blackjack is a classic casino game where the player plays against the dealer, who has a strategy set by the house. The player's goal is to get a sum of cards above the dealers, but not above 21. The player can achieve this using different strategies. We wish to test which of various strategies is the most profitable in the long run.

We wish to set up a table class with a dealer and a player with a specific strategy. The strategy is defined as a function. The table class uses a game class, that plays one game which the table then uses to calculate the player's new wealth. At the table there can be played multiple games, and the table then sums up game results when a predetermined number of games are played.

To compare the strategies, we will visualize the accumulated wealth for each strategy throughout the games.

Reflection upon the implementation

Throughout our project we have made some choices for our implementation. For example, in card_deck we have chosen to use both syllables and numbers instead of giving it their own value. We chose this to give a realistic view of Black Jack and to visualize it for ourselves.
In strategies we defined 7 different strategies, where some are alike and some are completely different. We did this to see what kind of effect it would have. An example of two rather similar strategies are 3 and 6, wherein 3 will hit when the sum of it's cards are less than 17, and in 6 it hits when the sum of the cards are less than 18. Which means that strategy 6 will hit more often than the dealer, whereas strategy 3's strategy is identical to the dealer. This small detail can change the outcome of the game and thereby we thought it would be interesting to observe.

To get the end result we made a function called who_won, which you can see below.

```
def who_won(self):
  if self.game.game_ending == 1:
    self.player_won += 1
  elif self.game.game_ending == 1.5:
    self.player_blackjack += 1
  elif self.game.game_ending == -1:
    self.dealer_won += 1
  else:
    self.tied += 1
```

The outcome was defined after game ending and would add or remove points, and thereby wealth, to the overall outcome. The function print_table_stats provided a summary of the results (code\Table_class.py, line 148). We mainly chose to set it up like this, to make the results more easily readable and interpretable for us (Appendix 4).

Other implementations we have done are calculating wealth. The sum of each players wealth is determined by a set starting wealth, constant across strategies, which then updates after each game.

```
  def update_wealth(self):
    self.player_wealth += (self.game.player_bet * self.game.game_ending)
```

We sat up our function so the wealth would be updated after the end of each game. Here it would take the bet and multiply it with the result of the game and then add to the previous wealth. An example of the development of wealth can be seen in appendix 1 where the player's wealth is plotted across the number of games played.

Python modules used

We used some different python (Van Rossum & Drake, 2009) modules, including random (haofan, n.d.), datetime (Contributors, n.d.), pyts (Bishop, n.d.), seaborn (Waskom, 2021), matplotlip.pyplot (Hunter, 2007), os (KTHM, n.d.) and pandas (Team, n.d.).

Random is the module we use in the card_deck to randomize our players and dealers' cards, so the deck, and thus the results will be different for each round (code\Card_deck_class.py, line 12).

Datetime is the module we use to extract a time stamp that then is used for creating the plots and logfile names (code\main.py, line 33).

Pyts is a module used within daytime that determines what timezone we are in (code\main.py, line 33).

Seaborn and matplotlib.pyplot are used to visualize our data. Seaborn provides an interface of visualization through statisticals graphics. We visualize the wealth accumulation over games played individually for each strategy using matplotlib.pyplot as seen in appendix 1 and to visualize this in a collective plot we use seaborn, as can be seen in appendix 2 and 3.
Os is used to check if there already exists a folder for the logfile and if it doesn't exist then create a new one (code\Table_class.py, line 176).
Pandas is used to create our dataframe with the data from all of the strategies and games, that is then later saved as a cvs file (code\Table_class.py, line 209).

Design choices and discussion

To get a structure of our code, we started by setting up the card deck as well as a game that we could interact with. We defined the dealer's strategy, to hit whenever the dealer's card sum is below 17.
In the beginning we had to make a lot of decisions about which rules our game would be played by. To create a limitation for our game, we decided that our players would only have the choice to stand or hit and would not be able to double down or split. We set up our card deck so we could choose how many decks we would use and so that we could change it if we wished (code\Card_deck_class.py, line 7).

Then we defined the table so that multiple games could be played after each other. In the beginning our design choice was very simple and just printed some of the results of each game after it had been played.

When the table was set, we defined a strategy that was set to interact with the table instead of us (code\Strategies.py, line 1). After succeeding we defined multiple strategies that used the players cards and dealers first card to decide when to hit or stand. Afterwards we started further developing the table so instead of printing after each game it gave a summary of how many times the dealer and player won as well as an update of the wealth (Appendix 4).
When this was done we made a for loop so each strategy was set to play an amount of times.
After debugging we got it all to work smoothly and then began to work on the task of visualizing the data.

We set it up so that a plot visualized the games played for each strategy with the development in wealth (Appendix 1). The plot was constructed after the table was done playing with all strategies, and then saved in individual folders for that day and timestamp. We realized it would be more informative to visualize the development in wealth over all the strategies (Appendix 2 & Appendix 3). To set this up we found it necessary to extract the data to a data frame and make a merged data frame containing all the data from all the strategies. Then afterwards we defined a function to set up and save this main plot, so it was given a data frame, predictor and outcome variable as well as a folder to add it to which you can see below.

```
def save_main_plot(data, x_axis, y_axis, filter, folder):
  sns.set(style='whitegrid')
  plt.figure(figsize=(10, 6))
  sns.lineplot(data=data, x=x_axis, y=y_axis, hue=filter)
  plt.title('{} Over {} by {}'.format(y_axis, x_axis, filter))
  plt.xlabel(f'{x_axis}')
  plt.ylabel(f'{y_axis}')
  plt.legend(title=f'{filter}')

  plot_name = '{}/main_plot_{}_{}_{}.png'.format(folder, time_stamp.hour,
                                                 time_stamp.minute,
                                                 time_stamp.second)

  plt.savefig(plot_name, dpi=300, bbox_inches='tight')

  plt.close()
```

You can see a difference in the development of wealth over games played in appendix 2 and 3, since the two plots show the results of tables with different numbers of games. There is a distinct color for each strategy, and it shows the development over games played (Appendix 2).

We wanted to see if there was a difference in wealth development over games played for the seven strategies. All strategies led to the player losing money and we could see on the final wealth that there was a difference for each strategy. For us to be able to test this we saw it necessary to extract the data from the constructed data frames to csv files. The files were then imported into an R (R core team, 2019) script (result\IPSA-BlackJack_analysis.pdf) where a linear model was set up with the following syntax:

```
wealth ~ games_played * strategy
```

The model testing told us that the perceived differences in performance were not significant, meaning that playing with different strategies did not influence the development in wealth.

Limitations

In appendix 2 it looks like there is a more significant difference when we played fewer games compared to appendix 3 where we worked with 1.000.000 games per strategy. This could indicate that it would make sense to play multiple tables with only a thousand games instead of playing all the games at one table.
When designing the game and the strategies, we limited ourselves to only the most basic game actions (hit or stand) and excluded actions such as split and double down. This limits the amount of depth we had to design strategies, while also limiting the direct comparability to real life casino games.

The hardest parts

We struggled a lot with our doctests in general, it required a lot of debugging and going back and forth. The problem wasn't with setting it up and getting it to test, but with defining the input and expected output correctly. We spent a lot of time debugging and even though it was a struggle it made us understand our code better and let us catch some mistakes that we previously didn’t notice.
In the beginning of our project, we tried to make a clear plan of what each class had to contain, but as we made the code, we found out that we missed some functions and had to go back to create them. Looking back, setting up the data frame took a lot of time and probably wasn't necessary, though we thought it was cool.
We are a group with little experience with Black Jack and therefore we spent a lot of time reading up on the rules, such as when to hit or stand, how the dealer had to move and what you could and couldn't do within the game.
Even though we spent a lot of time on the game and setting up the table, we had a lot of fun setting it up and developing all the functions and classes.

References

Bishop, S. (n.d.). *pytz: World timezone definitions, modern and historical* (2024.1) [Python; OS Independent]. Retrieved May 20, 2024, from http://pythonhosted.org/pytz

Contributors, Z. F. and. (n.d.). *DateTime: This package provides a DateTime data type, as known from Zope. Unless you need to communicate with Zope APIs, you're probably better off using Python's built-in datetime module.* (5.5) [Python; OS Independent]. Retrieved May 20, 2024, from https://github.com/zopefoundation/DateTime

haofan. (n.d.). *random11: Str int* (0.0.1) [Computer software].

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

KTHM. (n.d.). *os11: Os11* (0.0.3) [Python; MacOS :: MacOS X, Microsoft :: Windows, Unix].

R core team. (2019). *R: a language and environment for statistical computing* (3.0.2) [Computer software]. R Foundation for Statistical Computing.

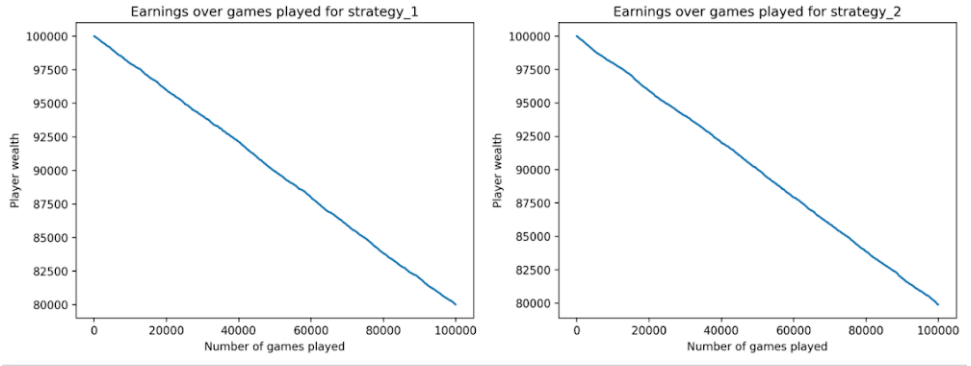
Team, T. P. D. (n.d.). *pandas: Powerful data structures for data analysis, time series, and statistics* (2.2.2) [Cython, Python; OS Independent]. Retrieved May 20, 2024, from https://pandas.pydata.org

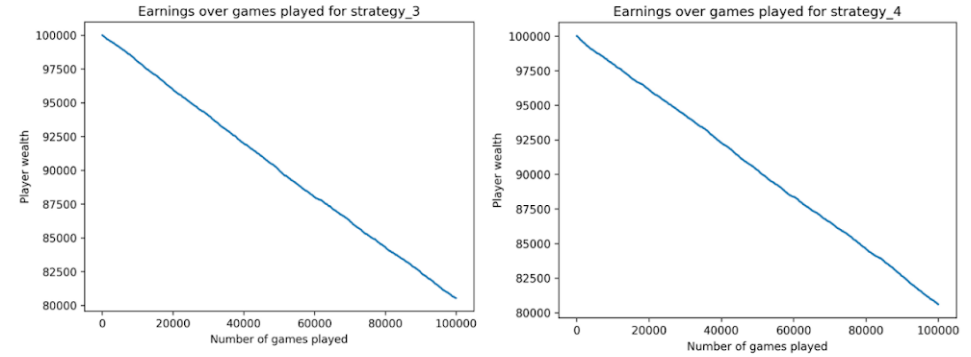
Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

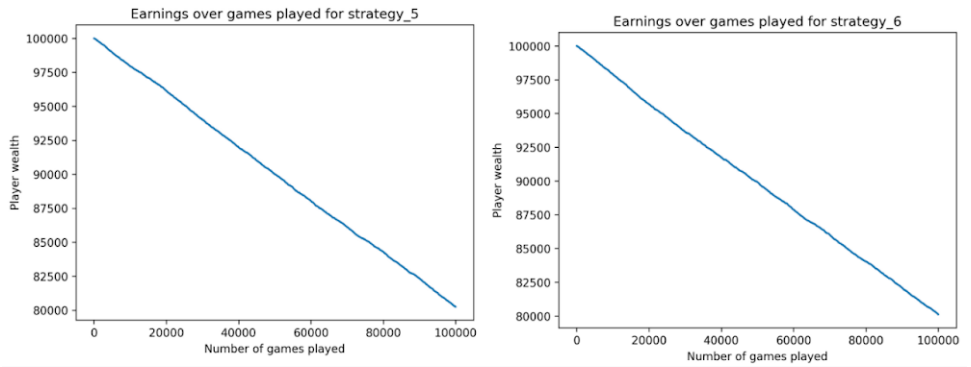
Waskom, M. (2021). seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

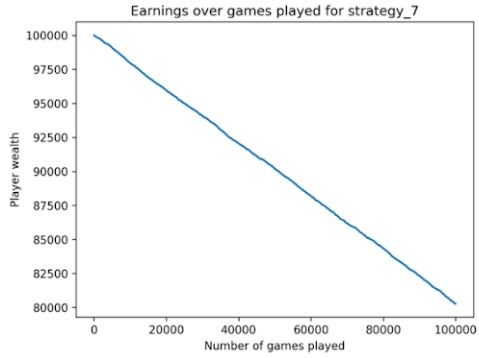
Appendix

Appendix 1:

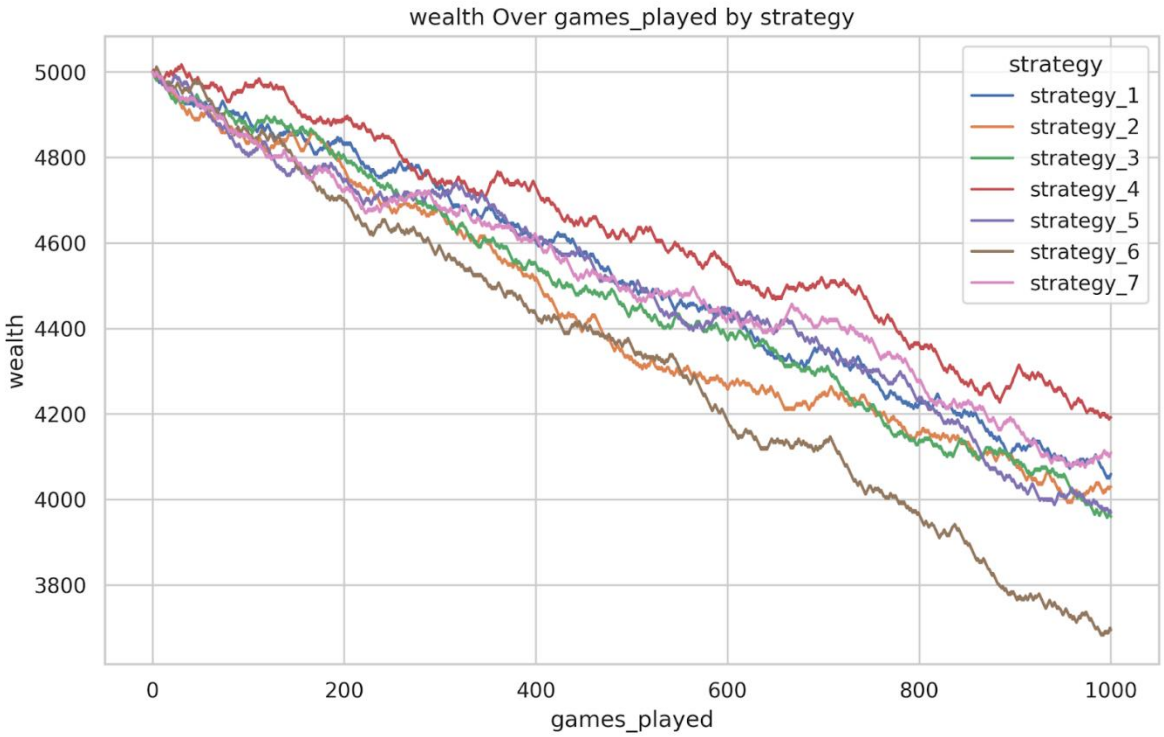




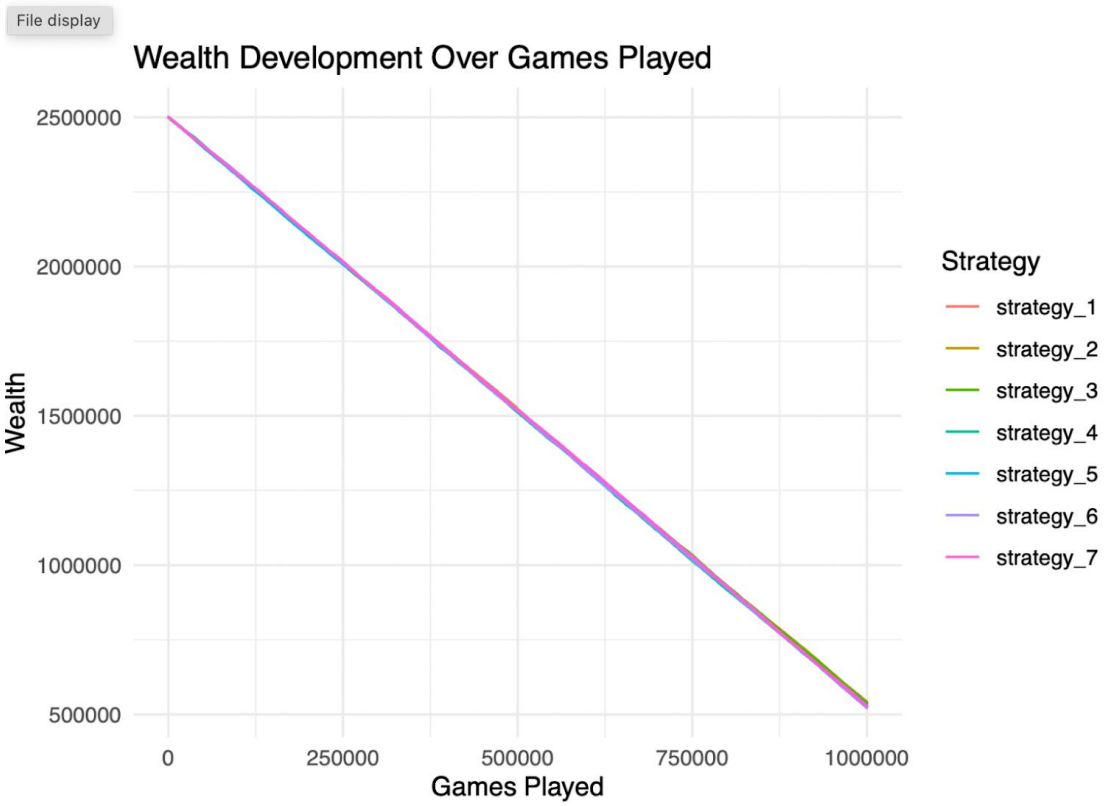




Appendix 2:



Appendix 3:



Appendix 4:

```
---------------------------
strategy_1
You got black Jack 45200 times
The player won 319936 times
You tied 50039 times
The dealer won 584825 times
The Players wealth is now 529110.0
d:\AU_Cog_Sci\matematik_tilvalg\intro
nt entries before the concat operatio
  main_dataframe = pd.concat([main_da
---------------------------
strategy_2
You got black Jack 45343 times
The player won 320219 times
You tied 50382 times
The dealer won 584056 times
The Players wealth is now 541775.0
---------------------------
strategy_3
You got black Jack 45393 times
The player won 320073 times
You tied 50118 times
The dealer won 584416 times
The Players wealth is now 537465.0
---------------------------
strategy_4
You got black Jack 45086 times
The player won 320149 times
You tied 49742 times
The dealer won 585023 times
The Players wealth is now 527550.0
---------------------------
strategy_5
You got black Jack 45217 times
The player won 319827 times
You tied 50244 times
The dealer won 584712 times
The Players wealth is now 529405.0
---------------------------
strategy_6
You got black Jack 45139 times
The player won 319747 times
You tied 49859 times
The dealer won 585255 times
The Players wealth is now 522005.0
---------------------------
strategy_7
You got black Jack 45327 times
The player won 319668 times
You tied 50215 times
The dealer won 584790 times
The Players wealth is now 528685.0
```