ETF Dashboard

The purpose of this project is to create a basic dashboard that can allow beginner retail investors access essential information on the ETF and compare them with potential benchmarks of their interest. The value added is to reduce the large amount of data that is found in websites like yahoofinance and JustETF and focus on the essential factors that can allow a non-professional investor to choose between different alternatives.

Download/Installation

Environment preparation:

- To display the following dashboard you should have an environment able to run a python code.
- The project was developed on Virtual Code Studio, which can be downloaded from here. After installing VSC you need to set up the environment for Python.

Packages download:

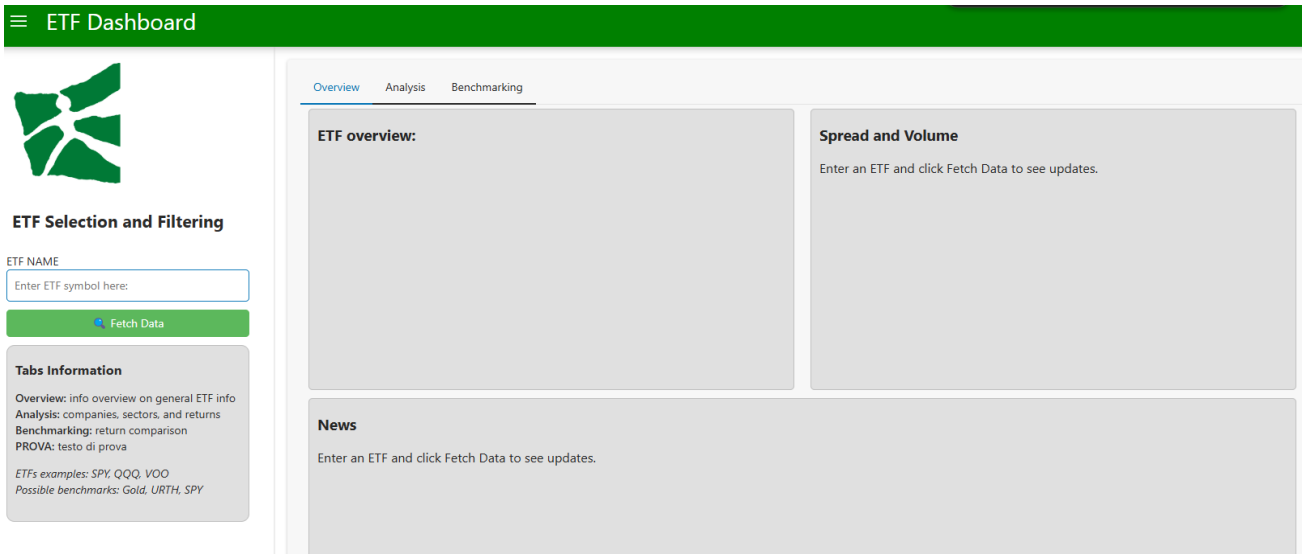
Once the environment is ready to run Python code we can proceed with downloading the different packages that will be used in this context:

- yfinance (Version 0.2.48): this package is used to fetch data from Yahoo Finance
- pandas (Version 2.2.1): this package is used to manipulate the data fetched and connect them to the different panels of the dashboard
- Hvplot (Version 0.11.1): this package allows us to create interactive plots where integrated with some widgets allowing the user to adjust the graph to its needs
- panel (Version 1.5.3): this is the overall package that supports the creation of the whole dashboard using the other packages to extract data, manipulate them and display them in different formats
- param (Version 2.1.1): is the library that allows us to create classes and objects with a series of parameters that are more easily updatable. Reducing our code and making the parameters more dynamic every time there is a change.

Usage

Here you can find the video tutorial: https://youtu.be/BW03_v-YDlU

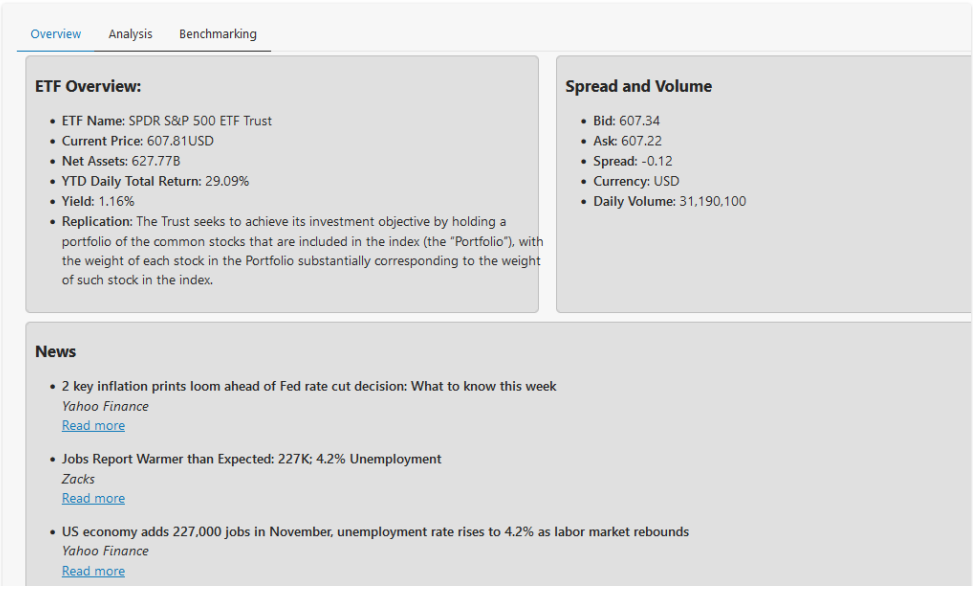
The dashboard is composed of two main areas:



- **Sidebar:** where you can fill the ETF Yahoo Finance ticker of your interest. Here it is crucial that you are first checking the ticker on Yahoo Finance as that is the queried dataset on the back end.
- **The Main Dashboard:** where you can look at some basic statistics and graphs about the specific ETF you have chosen

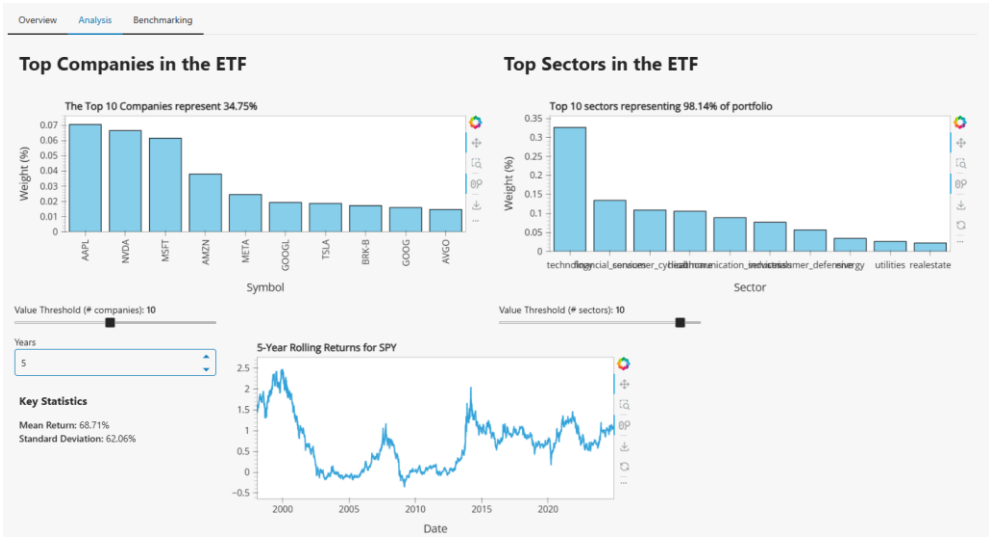
In the Main dashboard, we have three tabs that include:

- **Overview**



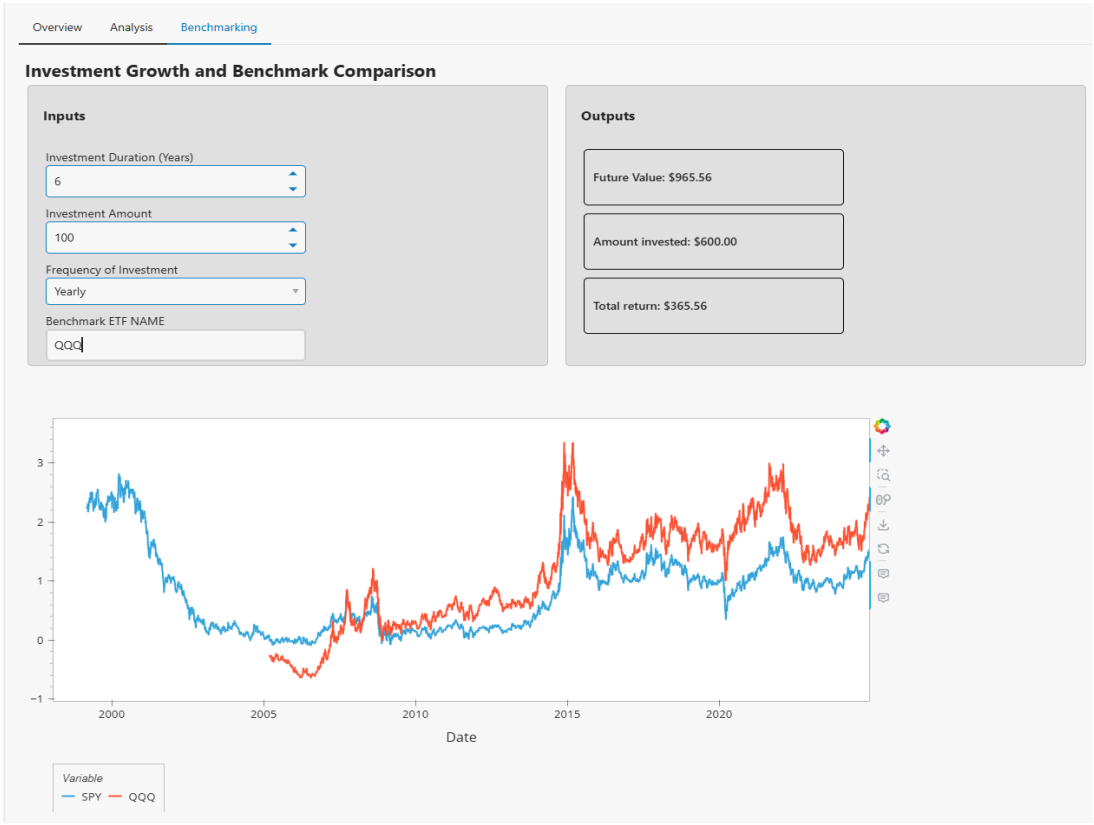
    - Here they can look at:
        - ETF Overview
            - *ETF Name*: The full name of the ETF, in case they want to search for it also in other platforms
            - *Current Price*: The current closing price that can give a quick glimpse of what potentially could be the price at which they are paying.
            - *Net Asset*: Approximate size of the fund calculated "as the value of an entity's assets minus its liabilities divided by outstanding shares"
            - *Replication*: Here we have the way the ETF has been constructed and therefore if it is a full, synthetic or sample replication (more details here).
        - *Spread and Volume:*
            - *Spread:* As the closing price is not indicative of the actual price at which they would be buying or selling, we are giving them on the side another section where they can look at the Bid (max buyer price) and Ask (min seller price) with the respective spread.
            - *Volume:* this variable indicates the most recent volume of the stock giving an idea of the liquidity of the market
        - *News*
            - This is the news section where the investor can quickly check if there are any relevant info that could impact the value of the ETF in question and therefore the decision of whether to buy it or not.

- **Analysis**



  - Here the user can dig deeper into critical components of the ETF looking at:
    - To which companies this ETF is exposed
    - How much this ETF is diversified
  - Then they can look at an approximate return they would get from their investment considering the historical prices and the years of investments (here we are not considering the ETFs that distribute the dividends and not reinvest them)
    - The formula calculates the “x” years percentage change for each single point in time, therefore displaying at each point what would have been the return if you invest “x” years before

- **Benchmarking**



- In the first section we are doing a simple simulation of a saving deposit.
  - Here the user defines the Inputs:
    - Years of the investment X
    - Amount to be invested Y
    - Frequency of the investment P
    - Benchmark against which to compare the X years performance
  - The output is calculated through
    - Future value Annuity formula
    - $FV = Y * (\frac{(1+R)^X - 1}{R})$
    - R is the annualised return in the period considered:
    - $R_t = \left(\frac{P_t}{P_{t-Y}}\right)^{\frac{Y}{365}} - 1$
- In addition to the simulation of a saving account, we are also giving the opportunity to compare the X years returns of the current ETF to a general benchmark that can be inserted freely by the investor. Here as for the previous section, for every point in time we are calculating the change in price with the value X years before.

Explanation of the code structure

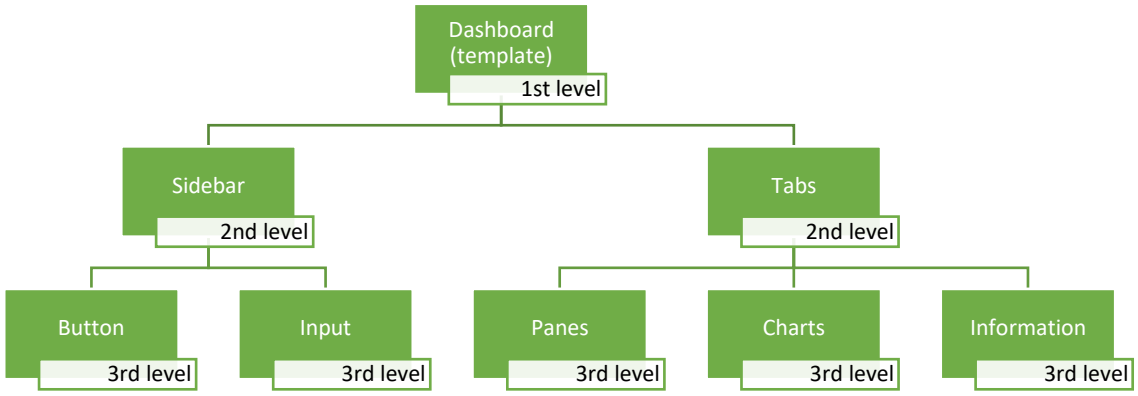
The code fundamentally utilizes 2 main tools to deliver the expected result: **panel widgets** and **functions**. On a broad level, the user interacts with the dashboard through widgets (e.g. sliders, text inputs) and the app manages these inputs through functions to return the desired information in a simple visual way. These two are intuitively connected through events (e.g. when a button is clicked by the user a function linked to it is called). When visualizing the *.py* file it is possible to clearly distinguish between the 3 "macro-parts" of the code:

1. Initialization of variables and creation of widgets
2. Definition of functions
3. Dashboard design and construction

Here's a description of the main features of the code:

- **Global Variables**
  We are using only one global of vital importance: "ticker_symbol". The value of this variable is a string that is utilized to fetch all the relevant information from Yahoo Finance. Whenever the user decides to launch a new search, they can just insert a new value for this variable and press again the "Fetch Data" button to get information for the new ETF. This is most likely the most fundamental part of the code: every time the user clicks on the button in the sidebar all the information will be updated according to the new string (new ETF).
- **Button Widget ("Fetch Data")**
  As mentioned already this button is key in the App. The user uses this button to launch the first and subsequent searches. When triggered, the button will update the ticker object and refresh all the information and charts in all the tabs.
- **Main Functions:**
  - "update_panes": this one is the most important function within the code, the one that is linked to the "Fetch Data" widget. It gets called when the button is pressed and what it does is:

        - (1) updating the ticker value,
        - (2) updating the 3 panes in the first tab (general info, news, volume and spread)
    - "update_plots": similarly this function serves to update the content of the second tab. It starts by updating the ticker value and then it proceeds to return the bar chart for the companies in the ETF, a bar chart for sector weights, the line chart with historical returns, and mean and standard deviation of the ETF.
    - "calculate_future_value": this function computes the future value, total invested amount, and total return of an ETF investment using historical data, annualized returns, and user-defined parameters like investment period and frequency. It displays the results in the up-right box in the 3rd tab of the dashboard.
    - "compare_benchmarks": it compares the performance of an ETF and a benchmark over a specified number of years by calculating percentage price changes. It merges their historical price data and visualizes the comparison as an interactive line plot. The user decides the benchmark by using an input text widget.
- **Design of the dashboard**:
  As mentioned, the dashboard is built with *panel*. The structure follows the simple bottom-up rule where "inner" objects (such as widgets and charts) are contained within medium objects (like rows and columns), that ultimately are contained in big objects (like panes, tabs, or the dashboard itself). More specifically in the dashboard, we have the following structure:



Best practices:

- **Error handling:** the code is designed to handle errors gracefully, ensuring that any issues during execution are clearly communicated to the user. This is achieved using Python's try-except structure to catch and address various errors. Additionally, the code anticipates potential user mistakes, such as incorrect formatting of the ETF ticker input. For instance, it automatically handles cases where the user enters "spy" or " SPY " instead of "SPY," ensuring the application functions correctly despite these imperfections.
- **Breakdown of code into small pieces:** since we collaborated on the design and structure of the code, we focused on ensuring it was clear and easy to understand. To achieve this, we created numerous small, reusable functions and objects, making the code more flexible for potential redesigns or seamless integration with each other's contributions.

Further development:

In the following section we have reported a list of tools we are currently working to further expand the usefulness and effectiveness of our dashboard:

- **Volumes plot:** Creating a graph representing the volumes of trading for a specific ETF that can allow investors to analyse the overall trend over time in the past
- **4th Dashboard:** Following the mean variance asset allocation we are currently working in the Creation of the efficient frontier that can allow the investors to create optimal portfolios and decide the most appropriate allocation of risk-free and risky assets.

Screenshot sections of code (in the collapsed version)

Part 1: Initialization of panes and widgets:

```
#####-----INITIALIZATION OF VARIABLES AND WIDGETS CREATION-----##########################################################

# enable the panel extension--> allows to work with widgets
pn.extension('tabulator') # for interactive data tables

# global variables
ticker_symbol:str = ''

# ETF input and fetch button
etf_input = pn.widgets.TextInput(name="ETF NAME", placeholder="Enter ETF symbol here:")
fetch_data_button = pn.widgets.Button(name="🔍 Fetch Data", button_type="success", width=300)

# Display panes (1st tab)
spread_volume_pane = pn.pane.Markdown("### Spread and Volume\n\nEnter an ETF and click Fetch Data to see updates.", ···

replication_pane = pn.pane.Markdown("### ETF overview:", ···

news_pane = pn.pane.Markdown("### News\n\nEnter an ETF and click Fetch Data to see updates.", ···

# Widgets for interactive plots (2nd tab)
wd_plot1_Top = pn.widgets.IntSlider(name='Value Threshold (# companies)', start=1, end=20, value=10)
wd_plot2_Top = pn.widgets.IntSlider(name='Value Threshold (# sectors)', start=1, end=11, value=10)
years_input = pn.widgets.IntInput(name='Years', value=5, step=1, start=1)

# Placeholder panes for interactive plots (2nd tab)--> used to update and clear when inserting new ETF
p1_interactive = pn.Column()
p2_interactive = pn.Column()
linked_data = pn.Column()
linked_data_2 = pn.Column()
```

Part 2: Definition of functions:

```
#####-----FUNCTIONS-----#####################################################################################################################################

# function to fetch news (Tab 1)
def get_news(ticker_symbol): ···

# Function to fetch spread and volume (Tab 1)
def get_spread_and_volume(ticker_symbol): ···

# Callback to update panes and plots (Tab 1)
def update_panes(event): ···

# Function to update plots in (Tab 2)
def update_plots(): ···

# link the fetch data button to the update function (Tab 2)
fetch_data_button.on_click(update_panes)

# widgets for benchmarking tab (Tab 3)
investment_years = pn.widgets.IntInput(name='Investment Duration (Years)', value=5, step=1, start=1)
investment_amount = pn.widgets.IntInput(name='Investment Amount', value=100, start=0)
investment_period = pn.widgets.Select(name='Frequency of Investment', options=["Yearly", "Monthly", "Quarterly"])
benchmark_select = pn.widgets.TextInput(name="Benchmark ETF NAME", placeholder="Enter the ETF symbol that will be used as benchmark:")

# Using param for ticker symbol (Tab 3)
class TickerParam(param.Parameterized): ···

ticker_param = TickerParam()

# Function to calculate future value (Tab 3)
def calculate_future_value(years, amount, period, ticker_param): ···

# Function to compare benchmarks (Tab 3)
def compare_benchmarks(years, benchmark, ticker_param): ···

# Create linked outputs (Tab 3)
investment_output = pn.bind(calculate_future_value, years=investment_years, amount=investment_amount, period=investment_period, ticker_param=ticker_param)
benchmark_comparison = pn.bind(compare_benchmarks, years=investment_years, benchmark=benchmark_select, ticker_param=ticker_param)
```

Part 3: Dashboard design and structuring:

```
#####-----DASHBOARD DESIGN-----#####################################

# Instruction text
side_text = pn.pane.Markdown(···

# Sidebar layout
sidebar = pn.Column(···

# Tab 1 content (Overview)
top_row = pn.Row(replication_pane, spread_volume_pane)
bottom_row = pn.Row(news_pane)
tab1_content = pn.Column(···

# Tab 2 content (Analysis)
plot1 = pn.Column(···
plot2 = pn.Column(···
stats_and_plot = pn.Row(···
tab2_content = pn.Column(···

#Tab 3 content (benchmarking)
middle_section = pn.Row(···
tab3_content = pn.Column(···

# Tabs
tabs = pn.Tabs(···

# Template design
template = pn.template.FastListTemplate(···

# Changing the color for the header bar in the template
template.header_background = "green"

# Defining main to display dashboard
def main():···

# If condition to avoid double calls
if __name__ == '__main__':···
```