S&P500-App: (source code: https://www.freecodecamp.org/)

SP500-App is a data driven web application in python for web scraping sp 500 stock prices(the data is taken from Wikipedia: url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies', and we are going to scrape the first table: S&P 500 component stocks) so it retrieves the list of the S&P 500 and its corresponding stock closing price (year-to-date)!

But what is S&P500?

S&P500: The Standard and Poor's 500, or simply the S&P 500,[5] is a stock market index tracking the stock performance of 500 large companies listed on stock exchanges in the United States. It is one of the most commonly followed equity indices.[WIKIPEDIA]

The Code:

```
import streamlit as st
import pandas as pd
import base64
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import yfinance as yf

st.title(' Standard and Poor\'s 500')

st.sidebar.header('User Input Features')

# Web scraping of S&P 500 data
#
@st.cache
def load_data():
    url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
    html = pd.read_html(url, header = 0)
    df = html[0]
    return df

df = load_data()
sector = df.groupby('GICS Sector')

# Sidebar - Sector selection
sorted_sector_unique = sorted( df['GICS Sector'].unique() )
selected_sector = st.sidebar.multiselect('Sector', sorted_sector_unique,
sorted_sector_unique)

# Filtering data
df_selected_sector = df[ (df['GICS Sector'].isin(selected_sector)) ]

st.header('Display Companies in Selected Sector')
st.write('Data Dimension: ' + str(df_selected_sector.shape[0]) + ' rows and
' + str(df_selected_sector.shape[1]) + ' columns.')
st.dataframe(df_selected_sector)

# Download S&P500 data
# https://discuss.streamlit.io/t/how-to-download-file-in-streamlit/1806
def filedownload(df):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()  # strings <-> bytes
conversions
```

```
    href = f'<a href="data:file/csv;base64,{b64}"
download="SP500.csv">Download CSV File</a>'
    return href

st.markdown(filedownload(df_selected_sector), unsafe_allow_html=True)

# https://pypi.org/project/yfinance/

data = yf.download(
        tickers = list(df_selected_sector[:10].Symbol),
        period = "ytd",
        interval = "1d",
        group_by = 'ticker',
        auto_adjust = True,
        prepost = True,
        threads = True,
        proxy = None
    )

# Plot Closing Price of Query Symbol
def price_plot(symbol):
  df = pd.DataFrame(data[symbol].Close)
  df['Date'] = df.index
  plt.fill_between(df.Date, df.Close, color='skyblue', alpha=0.3)
  plt.plot(df.Date, df.Close, color='skyblue', alpha=0.8)
  plt.xticks(rotation=90)
  plt.title(symbol, fontweight='bold')
  plt.xlabel('Date', fontweight='bold')
  plt.ylabel('Closing Price', fontweight='bold')
  return st.pyplot()

num_company = st.sidebar.slider('Number of Companies', 1, 5)

if st.button('Show Plots'):
    st.header('Stock Closing Price')
    for i in list(df_selected_sector.Symbol)[:num_company]:
        price_plot(i)
```

Code's Description:

First of all the libraries that we will use:

```
import streamlit as st
import pandas as pd
import base64
import matplotlib.pyplot as plt
import yfinance as yf
```

We use **streamlit** because the web application build by streamlit.

We use **pandas** because we need the data frame to show the data.

We use **base64** because it will allow us to encode the data so we are able to provide it as a csv file.

We use **matplotlib** because we will make a plot.

We use **yfinance** because we need financial data for our application.

```
st.title(' Standard and Poor\'s 500')
```
The title of our web application.

```
st.sidebar.header('User Input Features')
```
we create a header with header and put it inside the sidebar (st.sidebar.header).

```
@st.cache
```
With this line we catch the data. If it has already been run for the first time so that the second time or subsequent time it wouldn't need to redownload the data again and so it only needs to do that only once for the first time

---

```
def load_data():
    url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
    html = pd.read_html(url, header = 0)
    df = html[0]
    return df
```
This function allows us to scrape the data from Wikipedia. The first thing is we're going to put in the url of the s p 500. We're going to take the first table so we will using **pandas.read_html**-function (the data must be in a table so if data is in a paragraph it wouldn't work). Because we want the first table, we're going to specify this to be zero **df = html[0]** and then the data will go into the **df** which we will return.

```
df = load_data()
```
we're assigning the web script data into the df data frame and then we're going to group by the sector name

---

```
sector = df.groupby('GICS Sector')
```
here we are going to aggregate the data by grouping them according to the specific sector name which is comprised of 11 different sectors (GICS Sector: The Global Industry Classification Standard is an industry taxonomy developed in 1999 by MSCI and Standard & Poor's (S&P) for use by the global financial community. The GICS structure consists of 11 sectors [Wikipedia])

```
# Sidebar - Sector selection
sorted_sector_unique = sorted( df['GICS Sector'].unique() )
selected_sector = st.sidebar.multiselect('Sector', sorted_sector_unique,
sorted_sector_unique)
```
We select the Sector column(the fourth sector: df['GICS Sector']) and the function unique() will count all the possible sectors. So the variable sorted_sector_unique contains all possible names of sectors. And selected sector is a multiselect-Button that allow us to choose one or more option of the list sorted_sector_unique and then we put it in the sidebar (st.sidebar.multiselect).

```
df_selected_sector = df[ (df['GICS Sector'].isin(selected_sector)) ]
```
this line is going to filter out from the entire data frame the sectors that you have selected in the **selected sector**
isin-function is used to see which are coming from the selected sectors in the sidebar.

```
def filedownload(df):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()
    href = f'<a href="data:file/csv;base64,{b64}"
download="SP500.csv">Download CSV File</a>'
    return href
```

This code allows us to decode and encode the data, and make the csv file of the data available for download.

```
data = yf.download(
        tickers = list(df_selected_sector[:10].Symbol),
        period = "ytd",
        interval = "1d",
        group_by = 'ticker',
        auto_adjust = True,
        prepost = True,
        threads = True,
        proxy = None
    )
```

This part of code is the finance data(yfinance was explained in a different code/file.). It will download the data of the stock price directly from y finance. Tickers are limited to 10 companies.

```
def price_plot(symbol):
  df = pd.DataFrame(data[symbol].Close)
  df['Date'] = df.index
  plt.fill_between(df.Date, df.Close, color='skyblue', alpha=0.3)
  plt.plot(df.Date, df.Close, color='skyblue', alpha=0.8)
  plt.xticks(rotation=90)
  plt.title(symbol, fontweight='bold')
  plt.xlabel('Date', fontweight='bold')
  plt.ylabel('Closing Price', fontweight='bold')
  return st.pyplot()
```

This function to create the plot

```
num_company = st.sidebar.slider('Number of Companies', 1, 5)
```

It is the slider for you to select the number of Companies(limited to 5 and must be < tickers in data-function)

```
if st.button('Show Plots'):
    st.header('Stock Closing Price')
    for i in list(df_selected_sector.Symbol)[:num_company]:
        price_plot(i)
```

If we press the button show Plots, it will show us the plots.

P.S.: I took the source code from freeCodeCamp: https://www.freecodecamp.org.