

# Streamlit

## Aktien Web-App

von

Marvin Kraus  
Matr.-Nr.: 30207837

Dozent:

Prof. Dr. Samanpour  
Business Intelligence 2

Sommersemester 2022

# Inhaltsverzeichnis

## Abbildungsverzeichnis

<b>1</b>	<b>Einstieg Streamlit</b>	<b>1</b>
1.1	Installation Python Bibliotheken . . . . .	1
<b>2</b>	<b>Einführung in Aktien Web-App</b>	<b>3</b>
<b>3</b>	<b>Implementierung des Aktien Dashboards</b>	<b>5</b>
3.1	Plotten der Daten . . . . .	6
<b>4</b>	<b>Tabelle für Unternehmen und Ticker</b>	<b>8</b>
<b>5</b>	<b>Vorhersage des Aktienkurses mithilfe von Fbprophet</b>	<b>11</b>
<b>6</b>	<b>Main Methode</b>	<b>13</b>
<b>7</b>	<b>Modifikation von Streamlit</b>	<b>14</b>
7.1	CSS und Markdown . . . . .	14
7.2	Weiterführende Links . . . . .	15

# Abbildungsverzeichnis

1.1	Installation der Bibliotheken über Python Packages . . . . .	1
1.2	Importieren der Bibliotheken . . . . .	2
2.1	Endergebnis Aktien Web-App . . . . .	3
2.2	Starten der Streamlit App über Terminal . . . . .	4
2.3	Menü von Streamlit . . . . .	4
2.4	Einstellungen Streamlit . . . . .	4
3.1	Beispiel eines Tickers am Unternehmen Apple . . . . .	5
3.2	Festlegen des Zeitraums . . . . .	5
3.3	Speichern der Yfinance Daten in Dataframe . . . . .	6
3.4	Deklariieren der globalen Variablen . . . . .	6
3.5	Methode zum Verarbeiten der Yfinance Daten . . . . .	7
3.6	Erstellen der Graphen . . . . .	7
3.7	Hinzufügen von Slider und Titel . . . . .	7
4.1	Auswahl der Seiten . . . . .	8
4.2	Top 100 Unternehmen und deren Ticker Kürzel . . . . .	8
4.3	Einlesen der CSV . . . . .	9
4.4	Methode zum Anzeigen der Tabelle . . . . .	9
4.5	Tabelle mit Unternehmen und deren Tickerkürzel . . . . .	10
5.1	Prognose des Aktienkurses mithilfe von Fbprophet . . . . .	12
5.2	Forecasting Aktienkurs von Apple . . . . .	12
6.1	Main Methode des Programms . . . . .	13
6.2	Logik hinter "Go"Button . . . . .	13
7.1	Personalisierung von Streamlit durch CSS . . . . .	14
7.2	Methode CSS Datei . . . . .	14
7.3	Code von Markdown Formatierung . . . . .	15
7.4	Ergebnis von Markdown Formatierung . . . . .	15

---

# 1 Einstieg Streamlit

Streamlit ist eine Bibliothek und ein Framework für Python. Streamlit bietet die Möglichkeit mit wenigen Zeilen Code eine Web-App zu entwickeln. In der Web-App selbst ist es möglich mithilfe von anderen Bibliotheken wie zum Beispiel Pandas oder Matplotlib Daten zu verarbeiten und ausgeben zu lassen. Streamlit dient der Datenvisualisierung und ist somit optimal für Data Scientists, die im Umfeld des Machine Learning tätig sind. Streamlit übernimmt einen großen Teil für den Entwickler, da das Framework sehr viel bereitstellt worüber sich der Entwickler nicht kümmern muss.

Es bietet auch die Möglichkeit erstellte Web-Apps für jeden zugänglich zu machen. Streamlit ist Open-Source und somit für jeden frei zugänglich. Aktuell läuft es unter der Apache-Lizenz 2.0. Das Framework wurde entwickelt um Data Scientists es so einfach wie möglich zu machen selbst eine App zu entwickeln, da in der Vergangenheit die Entwicklung solcher Apps zweigeteilt war. Die Kernlogik wurde von den Data Scientists implementiert aber für die die App selbst wurden dann Software Entwickler zur Hilfe gebeten. Streamlit setzt genau da an um Data Scientists ein schnelleres Arbeiten zu ermöglichen.

Für diese Anleitung habe ich mich entschieden ein direktes Beispiel zur Visualisierung von Streamlit zu nehmen. Anhand einer Aktien Web-App soll exemplarisch das Potenzial von Streamlit gezeigt werden. Hierbei werden jedoch noch andere Bibliotheken verwendet, auf die ich nur kurz bei der Verwendung eingehen werde. Dies zeigt aber das Streamlit mit eigentlichen allen Bibliotheken kompatibel ist. Streamlit ist jedoch die Bibliothek, mit der die Web-App erstellt und auch ausgegeben wird.

## 1.1 Installation Python Bibliotheken

Die Entwicklungsumgebung, die in dieser Ausarbeitung verwendet wird, ist PyCharm von JetBrains. Zuerst erstellt man sich ein leeres Python Projekt mit einer .py Datei. Dies sollte von JetBrains automatisch erstellt werden. Im nächsten Schritt müssen nun die Bibliotheken eingebunden werden. Um die Bibliotheken einzubinden gibt es verschiedene Wege. Hier habe ich mich für die Variante, die in PyCharm angeboten wird, entschieden.

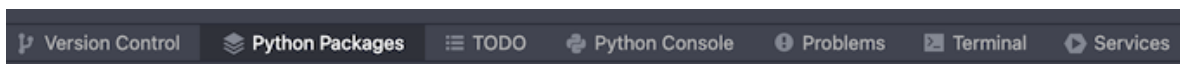


Abbildung 1.1: Installation der Bibliotheken über Python Packages

Die Bibliotheken, in der Liste auf der nächsten Seite, werden für das Entwickeln der Aktien Web-App benötigt. Streamlit wird benötigt um das Framework für die Web-App zu bekommen. Yfinance lädt die

```
import yfinance as yf
import streamlit as st

from datetime import date
from fbprophet import Prophet
from fbprophet.plot import plot_plotly
from plotly import graph_objs as go # for interactive graphs
import pandas as pd
import csv
```

Abbildung 1.2: Importieren der Bibliotheken

Daten zu einer Aktie per API in die Web-App. Yfinance wird von Yahoo bereitgestellt und bietet eine Vielzahl an Daten über Aktienunternehmen, die vom einfachen Aktienwert bis hin zur Bereitstellung der vierteljährlichen Geschäftszahlen einiges bietet. Fbprophet ist eine Bibliothek von Facebook, die es ermöglicht den Kurs einer Aktie vorherzusagen. Plotly dient der Darstellung der Graphen. Hier habe ich mich für Plotly und nicht Matplotlib entschieden, da die Graphen, die mit Plotly erstellt werden eine bessere Interaktivität bieten. Zuletzt müssen noch Pandas und Datetime heruntergeladen und einbezogen werden. Pandas dient der allgemeinen Verarbeitung der Daten, die wir mit Yfinance bekommen. Datetime wandelt das Datum von Yfinance um, da es sich um ein amerikanisches Format handelt. Über Python Packages können die Bibliotheken runtergeladen und danach eingebunden werden. Für das Projekt werden folgende Bibliotheken benötigt:

- Streamlit
- Yfinance
- Fbprophet
- Plotly
- Pandas
- Datetime

Es muss jedoch beachtet werden, dass über diesen Weg die Bibliotheken nur für das erstellte Projekt heruntergeladen werden. Will man die Bibliotheken dauerhaft in Python haben, muss dies über die Path Variable getan werden. Nachfolgend drei Links, die diesen Vorgang erläutern.

- Möglichkeiten zur Installation von Python Packages
- Path Variable auf MacOS hinzufügen
- Path Variable auf Windows hinzufügen

## 2 Einführung in Aktien Web-App

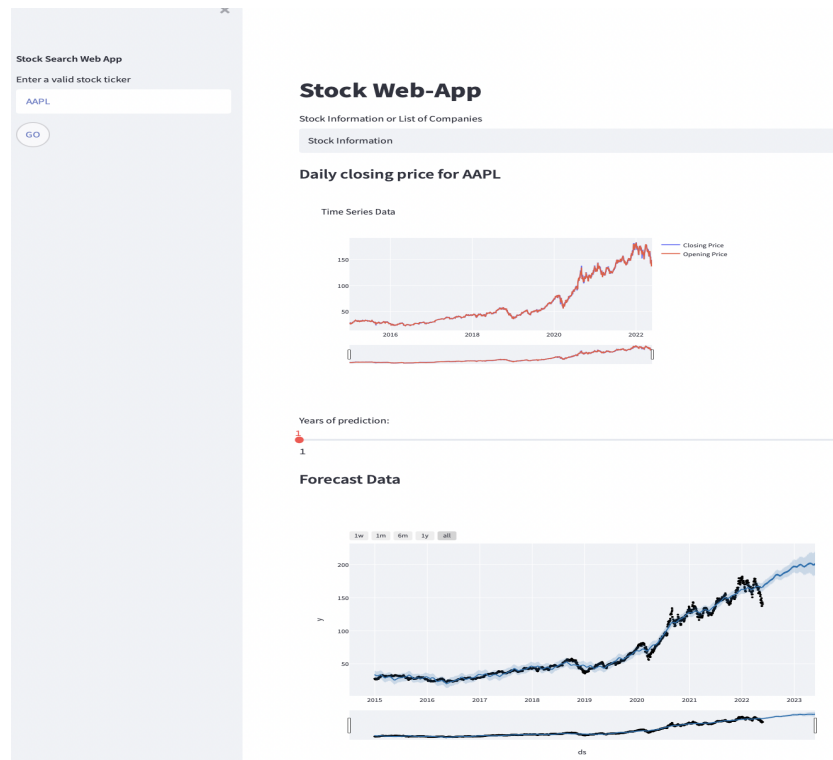


Abbildung 2.1: Endergebnis Aktien Web-App

Bevor es an die Erstellung der Web-App geht soll mit Abbildung 2.1 gezeigt werden wie das Resultat aussehen kann. Links soll der Anwender die Ticker Kürzel eingeben können. Wenn das Kürzel eingegeben worden ist sollen rechts die Informationen zu der Aktie gezeigt werden. Der Anwender soll die Möglichkeit haben durch sogenannte Slider interaktiv die Graphen zu benutzen. Des Weiteren soll mithilfe von Fbprophet eine Prognose zu der gewünschten Aktie angezeigt werden. Da der Anwender meist nicht alle Ticker Kürzel der Unternehmen auswendig kann, diese jedoch für die Benutzung der Web-App essentiell sind, soll eine Liste mit diesen Kürzeln auf einer weiteren Seite Abhilfe schaffen.

Nachdem die Bibliotheken heruntergeladen und wie in Abbildung 1.2 eingebunden sind kann getestet werden ob die App an sich schon funktioniert. Mit dem Befehl `streamlit run [name der Datei.py]` kann die App aus dem Terminal heraus gestartet werden. Im Terminal sollte nun folgendes ausgegeben werden (siehe Abbildung 2.2):

Die App kann jedoch auch über den klassischen Weg über den grünen Pfeil gestartet werden. Die App sollte sich in einem Browser Fenster öffnen. Die Web-App läuft über Localhost und ist mit

```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.178.136:8501
```

Abbildung 2.2: Starten der Streamlit App über Terminal

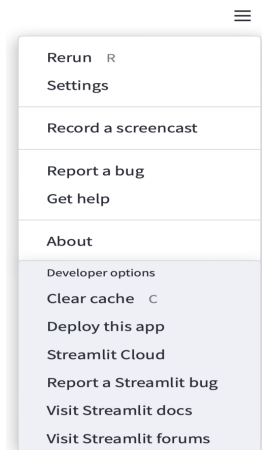


Abbildung 2.3: Menü von Streamlit

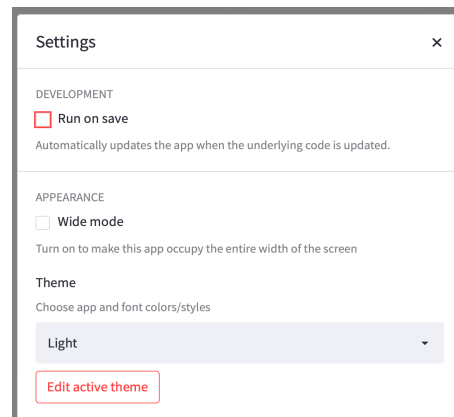


Abbildung 2.4: Einstellungen Streamlit

eigentlich jedem Browser erreichbar. Nachdem die App gestartet worden ist lässt sich wie erwartet noch kein Inhalt selbst auf der Seite erkennen, da bisher nur die Funktionalitäten des Programmes getestet worden sind. Auch wenn noch kein Inhalt auf der Seite implementiert wurde bietet Streamlit die Möglichkeit Einstellungen vorzunehmen.

Einerseits lässt sich die App über den "Rerun" Button oder mit dem Tastaturkürzel "R" neu laden. Der Deploy Button ist interessant für Entwickler, die ihre Web-App für jeden zugänglich machen wollen. Über die Settings ist es auch möglich zur Streamlit Dokumentation zu gelangen. Diese wird im Kapitel "Weiterführende Links" auch verlinkt.

In Abbildung 2.4 sehen wir die Möglichkeit, dass die App durch jegliche Code Änderung sich im Hintergrund aktualisiert wenn der Button "Run on Save" aktiviert wird. Dies kann gerade bei kleinen Änderungen an der App hilfreich sein, da nur die Änderung neugeladen wird und der Rest aus dem Cache geladen wird. Des Weiteren lässt sich Streamlit im Dark Mode anschauen. Die Hintergrundfarbe lässt sich in Python mithilfe von CSS konfigurieren. Dazu später mehr.

### 3 Implementierung des Aktien Dashboards

Ziel des Aktien-Dashboard soll sein, dass der Anwender einen Ticker eingibt und zu diesem Unternehmen die dazugehörigen Daten bekommt. Ein Ticker ist ein Kürzel, welches einmalig ist und somit wie eine ID funktioniert. Am Beispiel Apple lautet der Ticker 'AAPL'. Über diesen Ticker soll der Anwender die Informationen zu der Aktie ausgegeben bekommen und zwar von einem gesetzten Zeitpunkt in der Vergangenheit bis zum aktuellen Tag. Außerdem soll mithilfe der Fbprophet Bibliothek eine Prognose des Aktienkurses zu dem Unternehmen ausgegeben werden.

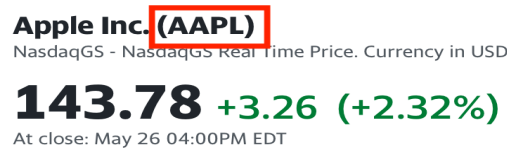


Abbildung 3.1: Beispiel eines Tickers am Unternehmen Apple

Im ersten Schritt legen wir ein Start und Ende Zeitraum fest in dem die Aktien abgerufen werden sollen. Hierfür werden zwei Variablen folgendermaßen deklariert:

```
START = '2015-01-01'  
TODAY = date.today().strftime("%Y-%m-%d")
```

Abbildung 3.2: Festlegen des Zeitraums

Für den Start Zeitpunkt wird der 01.01.2015 festgelegt. Dies kann aber nach Belieben gewählt werden. Für die Variable Today soll der heute aktuelle Aktienwert ermittelt werden. Hierfür kann die Klasse Date, die das Attribut today besitzt, benutzt werden. Das Datum muss anschließend noch formatiert werden, sodass es mit dem Datum von Yfinance übereinstimmt. Im nächsten Schritt werden die Daten von Yfinance benötigt. Auch hier wird dies in eine Methode ausgelagert um den Code lesbarer und übersichtlicher zu machen.

In der Methode LoadData werden die Yfinance Daten in der Variable Data gespeichert. Für den Download der Daten stellt Yfinance eine Möglichkeit zur Verfügung. Mit Download (Ticker, Start, Today) können die Daten des Tickers vom vorher festgelegten Zeitraum heruntergeladen werden.



```
def load_data(ticker):  
    data = yf.download(ticker, START, TODAY) # returns a panda dataframe  
    data.reset_index(inplace=True)  
    return data
```

Abbildung 3.3: Speichern der Yfinance Daten in Dataframe

Im nächsten Schritt werden einige Variablen deklariert die global benötigt werden. Zum einem bekommt die Web-App einen Titel. Dies kann über den Ausdruck `st.title("Stock Web-App")` gemacht werden.

```
# Global variables  
st.title("Stock Web-App")  
page = st.selectbox("Stock Information or List of Companies", ["Stock Information", "List of Companies"])  
st.sidebar.subheader("***Stock Search Web App***")  
selected_stock = st.sidebar.text_input("Enter a valid stock ticker")  
button_clicked = st.sidebar.button("GO")  
data = load_data(selected_stock)
```

Abbildung 3.4: Deklarieren der globalen Variablen

Außerdem wird für die 2-Seiten Funktionalität eine Selectbox benötigt mit der der Anwender zwischen beiden Seiten wechseln kann. Für diesen Schritt wird eine Variable namens Page deklariert. Eine Selectbox wird über `st.selectbox` erstellt. In runden Klammern dahinter bekommt die Selectbox einen Text, den der Anwender sieht. Im Fall der Aktien Web-App steht, dass sicher der Anwender zwischen "Stock Information oder List of Companies" entscheiden kann. Zusätzlich bekommt die Sidebar einen Titel. Auch hier wird über `st.sidebar.subheader` und der gewünschte Titel in Klammern gearbeitet.

Als nächstes wird der Ticker, den der Anwender eingibt benötigt, sodass dieser verarbeitet werden kann. Der Input des Anwenders wird in "selected\_stock " gespeichert. Hierfür wird über `st.sidebar.text_input` der eingegebene Ticker Name gespeichert. Der "GO" Button funktioniert wie die Enter Taste und soll am Ende das Programm starten. Der Anwender soll den Ticker über "GO" bestätigen. Der Button wird über `st.sidebar.button` erstellt. Im letzten Schritt der globalen Variablen wird die Methode "load\_data" aufgerufen und dieser wird der eingegebene Ticker Name übergeben. Das ganze wird dann in "data" gespeichert.

### 3.1 Plotten der Daten

Nun da die Daten von Yfinance heruntergeladen und schon gespeichert werden, geht es jetzt darum diese Daten in vernünftige Graphen anzeigen zu lassen. In Abbildung 3.5 ist dies innerhalb einer Methode gemacht worden.

```
def plot_raw_data():
    st.subheader("""Daily **closing price** for "" + selected_stock)
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=data['Date'], y=data['Open'], name='Closing Price'))
    fig.add_trace(go.Scatter(x=data['Date'], y=data['Close'], name='Opening Price'))
    fig.layout.update(title_text="Time Series Data", xaxis_rangeslider_visible=True)
    st.plotly_chart(fig)
```

Abbildung 3.5: Methode zum Verarbeiten der Yfinance Daten

Im ersten Schritt wird ein Subheader erstellt, der dem Anwender auf einen Blick zeigen soll, um welche Aktie es sich handelt. Die Graphen werden mit Plotly erstellt und diese Bibliothek wurde wie folgt implementiert: *"from plotly import graph\_objs as go"*. Ziel ist es zwei Graphen zu haben, ein Graph der den Opening Price und ein Graph der den Closing Preis einer Aktie zeigt. Dies ist möglich wenn über "go" eine Figure erstellt wird. Eine Figure ist ein Graph. In den folgenden beiden Zeilen werden die Figuren erstellt. Hierfür wird die Methode "Figure", die Plotly und speziell graph\_objs bereitstellt, aufgerufen.

Der nächste Schritt befasst sich mit den Achsen des Graphens. Hierfür wird über die erstellte Variable "fig" die Methode add\_trace aufgerufen. Über "go.scatter" wird der eigentliche Graph erstellt. Anschließend wird der X-Achse und Y-Achse aus den Daten, die von Yfinance bereitgestellt werden, das Datum und der jeweilige Opening und Closing Price zu diesem Datum zugewiesen. Dies geschieht über den Spaltenname aus den Yfinance Daten, diese werden in den eckigen Klammern angegeben. Nun werden den beiden Graphen sprechende Namen gegeben, sodass diese auch in der Legende, die neben des Graphens sein wird, stehen werden. (siehe Abbildung 3.6) Sind die Graphen erstellt muss das Layout geupdated werden.

```
fig.add_trace(go.Scatter(x=data['Date'], y=data['Open'], name='Closing Price'))
fig.add_trace(go.Scatter(x=data['Date'], y=data['Close'], name='Opening Price'))
```

Abbildung 3.6: Erstellen der Graphen

Außerdem bekommt der Graph einen sprechenden Namen und einen Slider, der es möglich macht sehr intuitiv den Graphen zu bewegen, sodass der Anwender rein und rauszoomen kann. Zuletzt muss noch Streamlit mitgeteilt werden, dass ein Graph gezeichnet werden soll.

Mit st.plotly\_chart(fig) wird ein Plotly Chart erstellt dem die zuvor angelegte "fig" übergeben wird.

```
fig.layout.update(title_text="Time Series Data", xaxis_rangeslider_visible=True)
st.plotly_chart(fig)
```

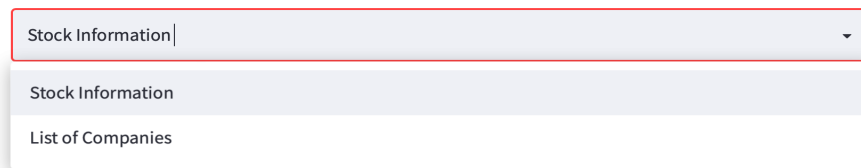
Abbildung 3.7: Hinzufügen von Slider und Titel

## 4 Tabelle für Unternehmen und Ticker

Um wie angesprochen es dem Anwender einfacher zu machen nach Ticks zu suchen wird eine Tabelle implementiert, die diese Ticker Kürzel ausgibt. Die Funktionalität mit den zwei verschiedenen Seiten ist schon im vorherigen Kapitel implementiert worden.

### Stock Web-App

Stock Information or List of Companies



Stock Information |

Stock Information

List of Companies

Abbildung 4.1: Auswahl der Seiten

Um an die Ticker Daten zu gelangen wird eine CSV verwendet, die wie folgt aussieht:

```
Ticker,Company,Sector
AAPL,Apple,Information Technology
ABBV,AbbVie,Health Care
ABT,Abbott,Health Care
ACN,Accenture,Information Technology
ADBE,Adobe,Information Technology
AIG,AIG,Financials
AMGN,Amgen,Health Care
AMT,American Tower,Real Estate
AMZN,Amazon,Consumer Discretionary
AVGO,Broadcom,Information Technology
```

Abbildung 4.2: Top 100 Unternehmen und deren Ticker Kürzel

Die ersten beiden Spalten sind für die folgende Schritte interessant. Dort sind in der ersten Spalte das Ticker Kürzel und in der zweiten Spalte das dazugehörige Unternehmen zu finden. Diese CSV wird, wie in der anschließenden Grafik zu sehen, eingelesen und die beiden genannten Spalten in der Variable "dict\_tickers" gespeichert.

Um die Tabelle ausgeben zu können wird ein Pandas Data Frame benötigt. In Abbildung 4.4 werden deswegen die Inhalte, die in "dict\_tickers" gespeichert sind, also der Ticker und das Unternehmen dazu in "data\_items" gespeichert. Die Variable "data\_items" wird dann in eine Liste gecastet, sodass am Ende die Liste als Pandas Data Frame zur Verfügung steht. Anschließend bekommen beide Spalten sprechende Bezeichnungen und die Tabelle selbst wird mit "st.table(df)" erstellt.

Hier es ist wichtig, dass der Data Frame erstellt wird weil Streamlit dies zur Verarbeitung benötigt. Das Endergebnis lässt sich an Abbildung 4.5 erkennen.

```
file = open("Top100_Company_Tickers.csv")
csvreader = csv.reader(file)
print(csvreader)
header = []
header = next(csvreader)
name = []
dict_tickers = {}

for rows in csvreader:
    dict_tickers.update({rows[0]: rows[1]})
file.close()
```

Abbildung 4.3: Einlesen der CSV

```
def Table_Ticker():
    data_items = dict_tickers.items()
    print(list(data_items))
    data_list = list(data_items)
    df = pd.DataFrame(data_list)
    df.columns = ['Ticker Name', 'Company Name']
    st.table(df)
```

Abbildung 4.4: Methode zum Anzeigen der Tabelle

# Stock Web-App

Stock Information or List of Companies

List of Companies	
Ticker Name	Company Name
AAPL	Apple
ABBV	AbbVie
ABT	Abbott
ACN	Accenture
ADBE	Adobe
AIG	AIG
AMGN	Amgen
AMT	American Tower
AMZN	Amazon
AVGO	Broadcom
AXP	American Express
BA	Boeing
BAC	Bank of America
BK	BNY Mellon
BKNG	Booking Holdings
BLK	BlackRock

Abbildung 4.5: Tabelle mit Unternehmen und deren Tickerkürzel

## 5 Vorhersage des Aktienkurses mithilfe von Fbprophet

Als letztes Feature soll innerhalb der Aktien Web-App eine Prognose des Aktienkurses zu einem Unternehmen angezeigt werden. Um dies machen zu können wird die Bibliothek "Fbprophet" von Facebook benötigt. Auch hier soll es dem Anwender möglich gemacht werden per Slider den Zeitraum beliebig anzupassen, sodass sich der Graph anpasst. Außerdem soll über einen weiteren Slider der Zeitraum der Prognose eingestellt werden können.

Zuallererst wird der Slider für das Einstellen des Prognosezeitraum implementiert. Dieser Slider unterscheidet sich vom Slider der beim ersten Graph implementiert wurde, der die Opening und Closing Price ausgeben soll davon, dass hier nur eine Linie zu sehen sein wird. Beim ersten Slider ist der Graph in klein im Slider selbst zu erkennen. Die Implementierung ist aber die gleiche. Um den Slider erstellen zu können muss folgendes geschrieben werden: `"st.slider("Years of Prediction:", 1,4)"`. Die Zahlen 1 und 4 repräsentieren den Anfang und Endzeitpunkt der Prognose. Somit kann bis zu vier Jahren in der Zukunft der Aktienkurs prognostiziert werden. Für Fbprophet wird nun ein Zeitraum (Period) benötigt. Die Variable "period" nimmt das Jahr welches der Anwender mit dem Slider einstellt und multipliziert dieses mit 365.

Im nächsten Schritt geht es darum das eigentliche Forecasting zu betreiben. Zuerst wird ein Data Frame erstellt dem die Spalten "Date" und "Close" übergeben wird. Anhand der Aktiendaten zu einem Unternehmen wird die Variable (Data Frame) "df\_train" "trainiert". Der nächste Schritt ist essentiell, da Fbprophet ein bestimmtes Format benötigt um die Prognose erstellen zu können. Um das Format zu bekommen wird folgendes geschrieben:

```
df_train = df_train.rename(columns={'Date': "ds", "Close": "y"})
```

Die Spaltennamen, die uns Yfinance liefert, müssen umbenannt werden. Die Umbenennung wird über ein Dictionary, also über ein Key-Value Paar, gelöst. Dies kann sehr einfach über "rename" erfolgen. Date bekommt "ds" und Close "y" zugewiesen. Für nachfolgende Informationen wird auf die Dokumentation von Fbprophet verwiesen. (Fbprophet Dokumentation)

Im nächsten Schritt wird nun die Methode Prophet aufgerufen. Dies wird in "m" für "Model" gespeichert. Noch ist aber kein Inhalt vorhanden, der eine Prognose liefern kann. Nun kann das "trainieren" des Models beginnen, indem mit `m.fit(df_train)` der erstellte Data Frame übergeben wird. Für die eigentliche Prognose wird aber ein weiterer Data Frame benötigt. Hierfür wird mit `future = m.make_future_dataframe(periods=period)` der Data Frame erstellt. In Klammern muss der Zeitraum übergeben, sodass der Data Frame weiß bis wann die Prognose gemacht werden soll. Die zuvor erstellte Variable "period" wird nun "periods" zugewiesen.

Der letzte Schritt, um die Prognose machen zu können ist, dass dem Model der "future Data Frame" übergeben wird. Über die Methode `.predict(future)` wird nun die Prognose erstellt. Diese Daten können auch schon ausgegeben werden, jedoch soll der Anwender dies auch in einem Graphen zu sehen bekommen. Um die Web-App übersichtlich zu halten wird auch hier ein Subheader implementiert. Um den Graphen zu erstellen wird wie beim ersten Graphen vorgegangen. Über `plot_plotly` wird der Graph erstellt. `Plot_plotly` erwartet für die Prognose zwei Sachen. Einerseits das Model und andererseits den Data Frame, in unserem Fall "future". Dies wird in der Variable "fig1" gespeichert.

```
def forecasting():
    n_year = st.slider("Years of prediction:", 1, 4)
    period = n_year * 365
    df_train = data[['Date', 'Close']]
    df_train = df_train.rename(columns={'Date': "ds", "Close": "y"})
    m = Prophet()
    m.fit(df_train)
    future = m.make_future_dataframe(periods=period)
    forecast = m.predict(future)
    st.subheader('Forecast Data')
    fig1 = plot_plotly(m, forecast)
    st.plotly_chart(fig1)
```

Abbildung 5.1: Prognose des Aktienkurses mithilfe von Fbprophet

Abschließend muss der Graph aber auch ausgegeben werden, um dies zu tun muss über `st.plotly_chart(fig1)` die Variable "fig1" Streamlit übergeben werden. Das Endergebnis lässt sich an Abbildung 5.2 am Beispiel Apple erkennen:

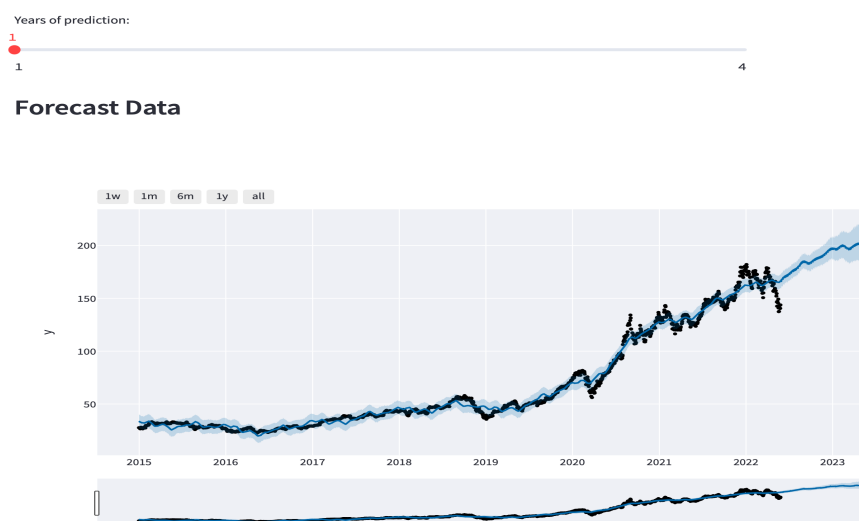


Abbildung 5.2: Forecasting Aktienkurs von Apple

## 6 Main Methode

Zum Schluss der eigentlichen Anleitung für das Erstellen der Aktien Web-App wird die Main Methode gezeigt. An Abbildung 6.1 lässt sich erkennen, dass durch eine einfache If-Abfrage abgefragt wird, welche "Page" der Anwender durch die Selectbox ausgewählt hat. Aufgrund der Entscheidung wird dann die entsprechende Methode aufgerufen.

```
def main():  
    if page == "Stock Information":  
        plot_raw_data()  
        forecasting()  
  
    if page == "List of Companies":  
        Table_Ticker()
```

Abbildung 6.1: Main Methode des Programms

Außerdem muss für die Logik hinter dem "Go" Button noch die Abfrage implementiert werden, dass die Main Methode aufgerufen wird, wenn der "Go" Button gedrückt wird.

```
def main():  
    pass  
  
if button_clicked == "GO":  
    main()
```

Abbildung 6.2: Logik hinter "Go"Button



## 7 Modifikation von Streamlit

### 7.1 CSS und Markdown

Abschließend soll aufgezeigt werden welche Möglichkeiten bestehen eine Web-App zu modifizieren. In Streamlit ist es möglich durch CSS und Markdown seine Web-App zu gestalten. Am Beispiel der Aktien Web-App wurde eine CSS Datei angelegt, die Primärfarbe und die Hintergrundfarbe ändert. An Abbildung 7.1 lässt sich die Implementierung erkennen. Unter anderem kann die Farbe des Inputs sowie auch die des "Go" Buttons geändert werden. Streamlit bietet somit eine große Auswahl an Modifikation. Für dieses Tutorial soll hier jedoch nicht näher auf die Formatierungsmöglichkeiten mit CSS eingegangen werden. Für weitere Informationen über CSS in Streamlit kann folgender Link helfen: [CSS Wiki](#)

Es muss jedoch erwähnt werden wie die CSS Datei in Streamlit selbst aufgerufen wird.



```
body {
  color: #ff0000;
  background-color: rgb(255, 255, 255);
}

.stButton>button {
  color: #ff0000;
  border-radius: 50%;
  height: 3em;
  width: 3em;
}

.stTextInput>div>div>input {
  color: #ff0000;
}
```

Abbildung 7.1: Personalisierung von Streamlit durch CSS



```
def local_css(file_name):
    with open(file_name) as f:
        st.markdown(hide_dataframe_row_index, unsafe_allow_html=True)
        st.markdown(f'<style>{f.read()}</style>', unsafe_allow_html=True)
```

Abbildung 7.2: Methode CSS Datei

In der Abbildung 7.2 lässt sich erkennen, dass dies über Markdown gelöst ist. Mit Markdown lassen sich einige weitere Dinge in der Streamlit App modifizieren. Ein Beispiel für die Modifizierung mit Markdown wäre folgendes:

Hallo ich bin ein Text

```
st.write("Hallo ich bin ein Text")  
  
st.write("***Hallo ich bin ein Text***")  
  
st.write("""**Hallo ich bin ein Text**""")
```

Abbildung 7.3: Code von Markdown  
Formatierung

*Hallo ich bin ein Text*

**Hallo ich bin ein Text**

Abbildung 7.4: Ergebnis von Markdown  
Formatierung

Wie zu erkennen ist können Überschriften damit vergrößert bzw. kursiv gemacht werden. Zudem konnte über Markdown das Problem gelöst werden, dass bei der Tabelle der Tickerkürzel eine durchgehende Nummerierung von Streamlit implementiert wurde.

## 7.2 Weiterführende Links

Die folgende Links in der Liste können beim Entwickeln einer eigener Web-App helfen. Die Punkte führen jeweils durch einen Klick auf die jeweilige Seite. Vor allem das Cheat Sheet, also die Streamlit Dokumentation, ist sehr ausführlich. Mein Quellcode ist unter Github einzusehen, falls noch etwas unklar ist. Auch verlinkt sind die weiteren Darstellungsmöglichkeiten, die Streamlit anbietet.

- [Streamlit Cheat Sheet](#)
- [Video Tutorial](#)
- [Quellcode GitHub](#)
- [Weitere Darstellungsmöglichkeiten in Streamlit](#)

Unter folgendem Link können noch andere Projekte gefunden werden. Die Seiten können im eigenen Browser ausgetestet werden. [Link zur Seite von Streamlit](#)

Eine Auswahl von Projekten, die angeklickt werden können, sind in folgender Liste zu sehen:

- [Covid-19 Dashboard](#)
- [Spracherkennung in Streamlit](#)
- [AI Piano](#)