Forecasting stock prices with machine learning

March 5, 2025

1 Forecasting stock data using different ML techniques

1.0.1 Igor Domaradzki

igor.domaradzki@unine.ch

In this project I want to: 1) Predict future stock values. 2) Use that information to make money!

For forecasting I used several techinques. After some initial testing I have decided to implement three: Sklearn's MLPRegressor, Meta's library called prophet and Logistic Regression.

In the first part I'll present how each model works. In the second, I'll compare their results by simulating real-life trading.

AI I tried to use as little AI as possible. I used Chat GPT 4o, mostly for explaining bugs, and to correct spelling mistakes in the comments. I will mark cells where AI was used. A copy of my entire conversation regarding this project can be found here https://chatgpt.com/share/67645b82-2df8-8006-8a30-ee6d809ce0e4

This doesn't mean all my code is original. I have based it a lot on some youtube videos on the topic. They are linked in their appropriate section.

Notice! yfinance, prophet are not included in noto. Please run the cell below with the bash kernel.

[625]:
```
my_venvs_create programming
my_venvs_activate programming
pip install yfinance
pip install xgboost
pip install prophet
my_venvs_deactivate
```

```
  Cell In[625], line 1
    my_venvs_create programming
                    ^
SyntaxError: invalid syntax
```

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

import numpy as np
import seaborn as sns
import statsmodels.api as sm

import warnings
warnings.filterwarnings("ignore")

plt.style.use("fivethirtyeight")
```

1.1 MLPRegressor

Multi-layer Perceptron regressor is a form of neural networks from sklearn.

https://scikit-learn.org/1.5/modules/generated/sklearn.neural_network.MLPRegressor.html

Let's see how it works.

[ ]:
```
# Downloading the data
data = yf.download("AAPL", start="2020-01-01", end="2024-01-01")
data.head()
```

[3]:
```
# Fixing occasional issues with how data from yfinance is downloaded.
try:
    data.columns = data.columns.droplevel('Ticker')
except:
    print(" ")
data.head()
```

[3]:
```
Price                       Adj Close      Close       High        Low  \
Date
2020-01-02 00:00:00+00:00  72.796005  75.087502  75.150002  73.797501
2020-01-03 00:00:00+00:00  72.088287  74.357498  75.144997  74.125000
2020-01-06 00:00:00+00:00  72.662720  74.949997  74.989998  73.187500
2020-01-07 00:00:00+00:00  72.320969  74.597504  75.224998  74.370003
2020-01-08 00:00:00+00:00  73.484367  75.797501  76.110001  74.290001

Price                            Open     Volume
Date
2020-01-02 00:00:00+00:00  74.059998  135480400
2020-01-03 00:00:00+00:00  74.287498  146322800
```

```
2020-01-06 00:00:00+00:00  73.447502  118387200
2020-01-07 00:00:00+00:00  74.959999  108872000
2020-01-08 00:00:00+00:00  74.290001  132079200
```

[4]:
```
# We want to predict the adjusted close price.
data = data[['Adj Close']]
data
```

[4]:
```
Price                       Adj Close
Date
2020-01-02 00:00:00+00:00   72.796005
2020-01-03 00:00:00+00:00   72.088287
2020-01-06 00:00:00+00:00   72.662720
2020-01-07 00:00:00+00:00   72.320969
2020-01-08 00:00:00+00:00   73.484367
…                                 …
2023-12-22 00:00:00+00:00  192.656174
2023-12-26 00:00:00+00:00  192.108871
2023-12-27 00:00:00+00:00  192.208359
2023-12-28 00:00:00+00:00  192.636276
2023-12-29 00:00:00+00:00  191.591385

[1006 rows x 1 columns]
```

First idea I had is to use lagged observations to predict the next one. Then to use the predicted observation to predict the next one, and so on.

[ ]:
```
# Adding lagged regressosrs.
for i in range(1,100):
    data[f"lag{i}"] = data['Adj Close'].shift(i)
#data.dropna(inplace=True)
data
```

[ ]:
```
# Removing rows with NAs
data.dropna(inplace=True)
data
```

[7]:
```
# The current stock price is our target
y = data["Adj Close"]
y
```

[7]:
```
Date
2020-05-26 00:00:00+00:00     77.156555
2020-05-27 00:00:00+00:00     77.492706
2020-05-28 00:00:00+00:00     77.526802
2020-05-29 00:00:00+00:00     77.451279
2020-06-01 00:00:00+00:00     78.403778
```

```
                                   …
2023-12-22 00:00:00+00:00    192.656174
2023-12-26 00:00:00+00:00    192.108871
2023-12-27 00:00:00+00:00    192.208359
2023-12-28 00:00:00+00:00    192.636276
2023-12-29 00:00:00+00:00    191.591385
Name: Adj Close, Length: 907, dtype: float64
```

[ ]:
```
# The lagged values are the features.
X = data.loc[:, data.columns != 'Adj Close']
X
```

[9]:
```
# Spliting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=144) # 144 is my favourite number! :)
```

[10]:
```
# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[ ]:
```
X_train.head(5)
```

[12]:
```
# Initializing and training the MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=[10,10], activation='relu',␣
↪solver='adam', max_iter=1000, random_state=144)
mlp.fit(X_train_scaled, y_train)
```

[12]:
```
MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000, random_state=144)
```

[13]:
```
# Using the model to predict
y_pred = mlp.predict(X_test_scaled)
```

[ ]:
```
y_pred
```

[15]:
```
# Evaluate the model
mape = mean_absolute_percentage_error(y_test, y_pred)
print(f'Mean Absolute Percentage Error: {(mape*100):.2f}%')
```

```
Mean Absolute Percentage Error: 1.87%
```

It seems the model maybe isn't that bad. Let's use it to perform a forecast

[16]:
```
# Creating a new df for the forecast
new_df = pd.DataFrame(columns = data.columns)
new_df = new_df.loc[:, new_df.columns != 'Adj Close']
new_df
```

[16]:
```
Empty DataFrame
Columns: [lag1, lag2, lag3, lag4, lag5, lag6, lag7, lag8, lag9, lag10, lag11,
lag12, lag13, lag14, lag15, lag16, lag17, lag18, lag19, lag20, lag21, lag22,
lag23, lag24, lag25, lag26, lag27, lag28, lag29, lag30, lag31, lag32, lag33,
lag34, lag35, lag36, lag37, lag38, lag39, lag40, lag41, lag42, lag43, lag44,
lag45, lag46, lag47, lag48, lag49, lag50, lag51, lag52, lag53, lag54, lag55,
lag56, lag57, lag58, lag59, lag60, lag61, lag62, lag63, lag64, lag65, lag66,
lag67, lag68, lag69, lag70, lag71, lag72, lag73, lag74, lag75, lag76, lag77,
lag78, lag79, lag80, lag81, lag82, lag83, lag84, lag85, lag86, lag87, lag88,
lag89, lag90, lag91, lag92, lag93, lag94, lag95, lag96, lag97, lag98, lag99]
Index: []

[0 rows x 99 columns]
```

[17]:
```python
# 99 last observations, used as input for the first prediction
list_last_obs = list(data["Adj Close"].tail(99)[::-1])
list_last_obs[:5]
```

[17]:
```
[191.5913848876953,
 192.6362762451172,
 192.20835876464844,
 192.10887145996094,
 192.6561737060547]
```

[18]:
```python
new_df.loc[len(new_df)] = list_last_obs
new_df
```

[18]:
```
Price         lag1        lag2        lag3        lag4        lag5        lag6  \
0       191.591385  192.636276  192.208359  192.108871  192.656174  193.730896

Price         lag7        lag8        lag9       lag10  …       lag90  \
0       193.880188  195.979889  194.935013  196.606827  …  179.999863

Price        lag91       lag92       lag93       lag94       lag95       lag96  \
0       176.133926  174.752533  173.410889  172.923904  175.478012  176.352585

Price        lag97       lag98       lag99
0       178.350143  176.690475  176.630844

[1 rows x 99 columns]
```

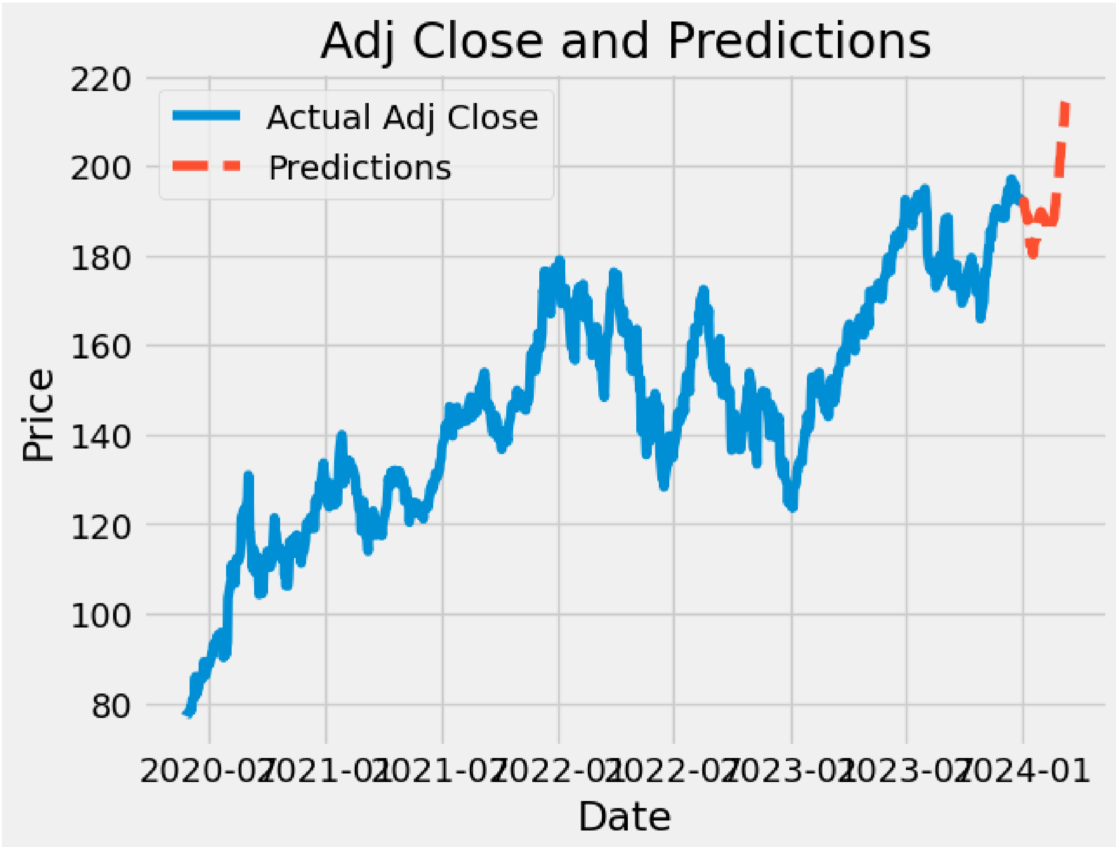
[19]:
```python
# Scaling the data to use with the model
new_df_scale = scaler.transform(new_df)
# Predicting!
pred = mlp.predict(new_df_scale)
float(pred)
```

[19]: 190.95221247658827

[ ]:
```python
# Now I'm making a loop, where I add the predition I made to the begining of␣
↪the features to predict the next observation
n_predictions = 50
list_curr_obs = list_last_obs.copy()
for i in range(n_predictions):
    new_df = pd.DataFrame(columns = data.columns)
    new_df = new_df.loc[:, new_df.columns != 'Adj Close']
    new_df.loc[len(new_df)] = list_curr_obs
    new_df_scale = scaler.transform(new_df)
    pred = mlp.predict(new_df_scale)
    list_curr_obs = [float(pred)] + list_curr_obs[:-1]
# The list of the 50 predictions and 50 last historical values
print(list_curr_obs)
```

Let's visualize the results!

[21]:
```python
import matplotlib.pyplot as plt
import pandas as pd
#n = 10
# Create a DataFrame from your list of predictions, with the same index as the␣
↪original 'data'
pred_dates = pd.date_range(start=data.index[-1] + pd.Timedelta(days=1),␣
↪periods=n_predictions, freq='C')  # 50 days after the last date in 'data'
pred_df = pd.DataFrame(list_curr_obs[:n_predictions][::-1], index=pred_dates,␣
↪columns=['Predicted'])

# Plot the original 'Adj Close' and the predictions
plt.plot(data['Adj Close'], label='Actual Adj Close')  # Actual data
plt.plot(pred_df.index, pred_df['Predicted'], label='Predictions',␣
↪linestyle='--')  # Predictions

# Customize the plot
plt.title('Adj Close and Predictions')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()

# Show the plot
plt.show()
```



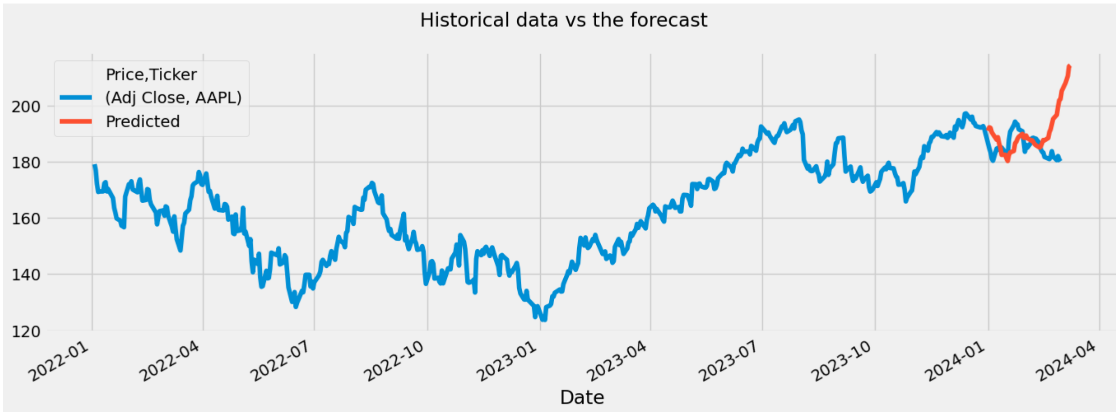
Let's compare it to the actual results.

[22]:
```
validation = yf.download("AAPL", start="2022-01-01", end="2024-03-01")
validation = validation[["Adj Close"]]
```

```
[*********************100%***********************]  1 of 1 completed
```

[23]:
```
# Plotting the actual data, and the forecst
fig,ax = plt.subplots(figsize=(15,5))
fig.suptitle("Historical data vs the forecast")
validation.plot(ax=ax)
pred_df.plot(ax=ax)
```

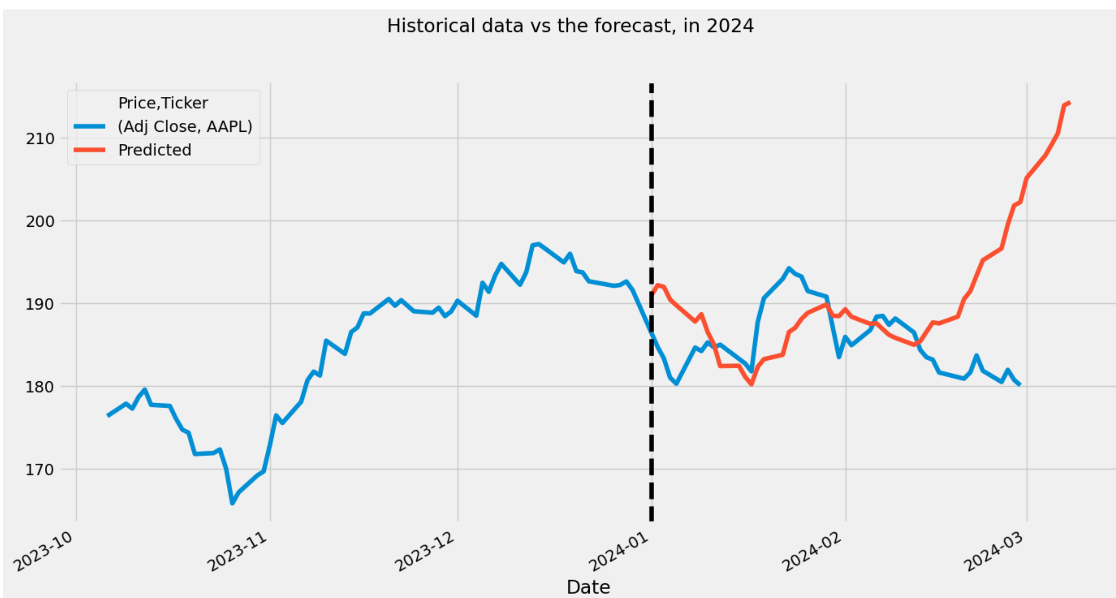
[23]: <Axes: xlabel='Date'>



Let's examine it more closely.

[24]:
```
fig,ax = plt.subplots(figsize=(15,8))
validation[(len(validation)-100):].plot(ax=ax, lw = 4)
pred_df.plot(ax=ax, lw = 4)
ax.axvline("01-01-2024", color = "black", ls = "--", lw = 4)
fig.suptitle("Historical data vs the forecast, in 2024")
```

[24]: Text(0.5, 0.98, 'Historical data vs the forecast, in 2024')



Turns out that at the beginning the predictions aren't that bad, but the further form the split point we go, the worse it gets. Which makes sense as with each prediction, the potential error increases.

This, I have to say, is a particularly good result. If the first couple of predictions predict growth, the effect will accumulate, and the predictions will grow exponentially.

1.2 Using Meta's Prophet

Made with help of a video "Forecasting with the FB Prophet Model" - https://youtu.be/j0eioK5edqg?si=w9LGEZZrcm6su-oy

Prophet is a forecasting model library from Meta based on econometrics.

[25]:
```
from prophet import Prophet
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪mean_absolute_percentage_error
```

[ ]:
```
# Let's predict the gold price
Gold = yf.download("GC=F", start="2000-01-01", end="2024-01-01")
Gold
```

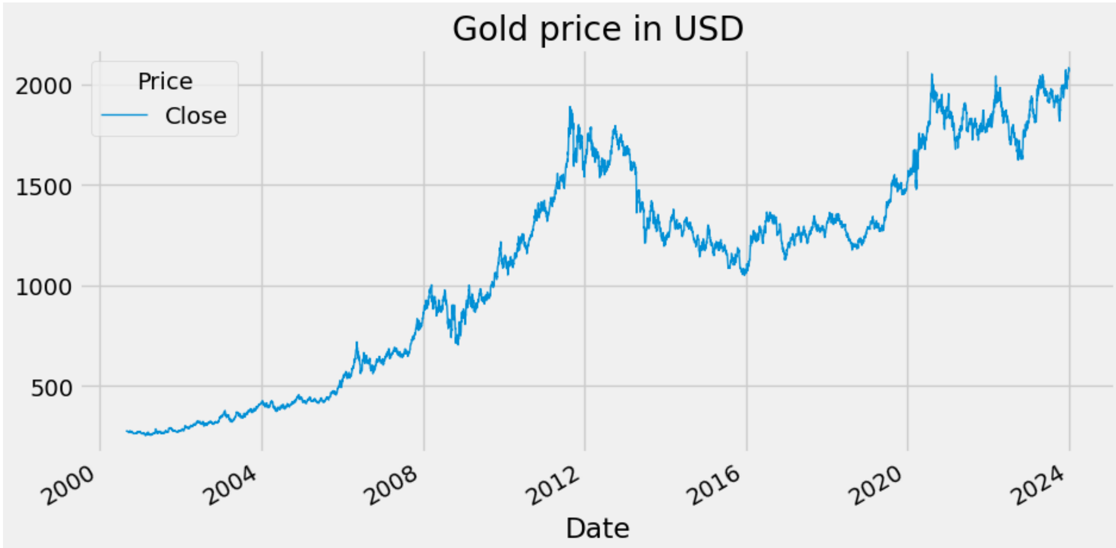
[27]:
```
try:
    Gold.columns = Gold.columns.droplevel('Ticker')
except:
    print(" ")
```

[28]:
```
Gold = Gold[["Close"]]
Gold
```

[28]:
```
Price                              Close
Date
2000-08-30 00:00:00+00:00   273.899994
2000-08-31 00:00:00+00:00   278.299988
2000-09-01 00:00:00+00:00   277.000000
2000-09-05 00:00:00+00:00   275.799988
2000-09-06 00:00:00+00:00   274.200012
…                                  …
2023-12-22 00:00:00+00:00  2057.100098
2023-12-26 00:00:00+00:00  2058.199951
2023-12-27 00:00:00+00:00  2081.899902
2023-12-28 00:00:00+00:00  2073.899902
2023-12-29 00:00:00+00:00  2062.399902

[5854 rows x 1 columns]
```

[29]:
```
#Let's visualize it
Gold.plot(figsize=(10,5), lw = 1, title = "Gold price in USD")
```

[29]:
```
<Axes: title={'center': 'Gold price in USD'}, xlabel='Date'>
```



[30]:
```
# Train / Test split

split_date = "01-01-2021"
Gold.index = Gold.index.tz_localize(None)

Gold_train = Gold.loc[Gold.index <= split_date].copy()
Gold_test = Gold.loc[Gold.index > split_date].copy()
```

[31]:
```
# The Prophet library asks for data with specific column names. ds for the␣
↪date, y for the value
Gold_train_prophet = Gold_train.reset_index() \
    .rename(columns = {"Date" : "ds", "Close" : "y"})
Gold_train_prophet
```

[31]:

Price	ds	y
0	2000-08-30	273.899994
1	2000-08-31	278.299988
2	2000-09-01	277.000000
3	2000-09-05	275.799988
4	2000-09-06	274.200012
…	…	…
5096	2020-12-24	1879.900024
5097	2020-12-28	1877.199951
5098	2020-12-29	1879.699951
5099	2020-12-30	1891.000000
5100	2020-12-31	1893.099976

[5101 rows x 2 columns]

[32]:

```
# Initializing
model = Prophet()
```

[33]:

```
model.fit(Gold_train_prophet)
```

```
18:08:51 - cmdstanpy - INFO - Chain [1] start processing
18:08:54 - cmdstanpy - INFO - Chain [1] done processing
```

[33]: <prophet.forecaster.Prophet at 0x7fe594c000d0>

[34]:

```
# Let's see the model's precition
Gold_test_prophet = Gold_test.reset_index() \
    .rename(columns = {"Date" : "ds", "Close" : "y"})
Gold_test_fcst = model.predict(Gold_test_prophet)
```

[35]:

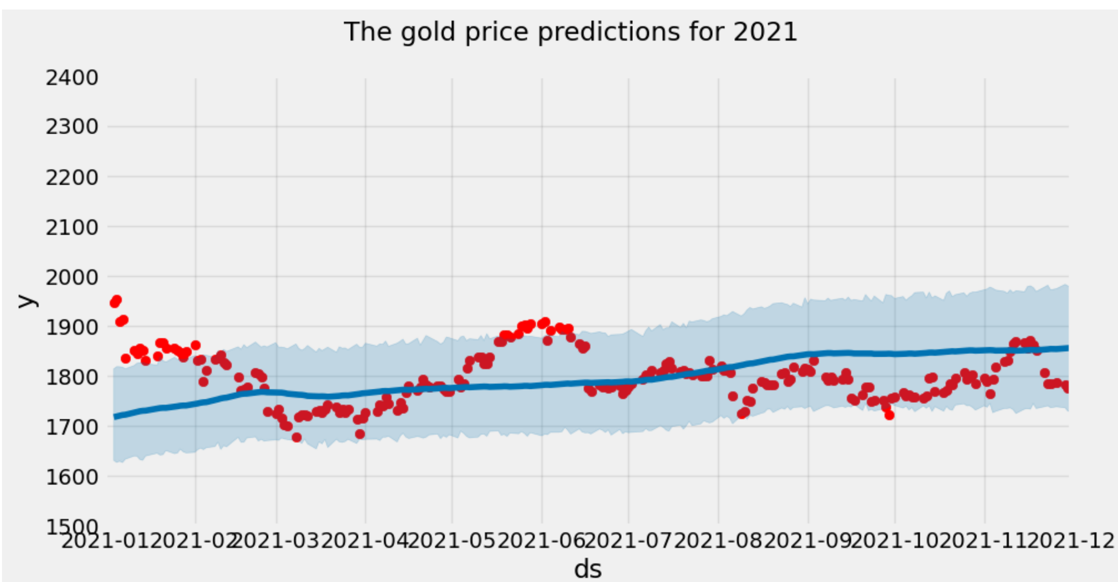
```
fig, ax = plt.subplots(figsize = (10,5))
fig.suptitle("The gold price, and the forecast for 2024")
fig = model.plot(Gold_test_fcst, ax =ax,)
```

[36]:

```
# And compare it with the results
f, ax = plt.subplots(figsize = (15,5))
ax.scatter(Gold_test.index, Gold_test["Close"], color = "r")
fig = model.plot(Gold_test_fcst, ax=ax)
```

[37]:
```python
from datetime import datetime
# Let's check it out more closely
fig,ax = plt.subplots(figsize = (10,5))
ax.scatter(Gold_test.index, Gold_test["Close"], color = "r")
fig = model.plot(Gold_test_fcst, ax = ax)
fig.suptitle("The gold price predictions for 2021")
lower_bound = datetime.strptime("2021-01-01", "%Y-%m-%d")
upper_bound = datetime.strptime("2021-12-01", "%Y-%m-%d")

# Set the x-axis bounds
ax.set_xbound(lower=lower_bound, upper=upper_bound)
ax.set_ylim(1500,2400)
```

[37]: (1500.0, 2400.0)



[38]:
```
print(f"Mean absolute percentage error: {mean_absolute_percentage_error(y_true␣
↪= Gold_test, y_pred = Gold_test_fcst['yhat'])*100:.2f}%") #MAPE
```

```
Mean absolute percentage error: 6.63%
```

The prophet model seems to be predicting quite well, as most of the values for 2021 were in the model's 95% confidence interval. As it's a econometric model, it focuses more on the long time trends and will not catch quick drops and rises in the stock prise

1.3 Logistic regression

In this part my goal is not to predict the exact value of the stock, but whether it will go up or down.

This part is based on Algovibes' youtube video "Logistic Regression in Python. (…)" https://youtu.be/X9jjyh0p7x8?si=zLgaFWTu5VZe8TTe

[39]:
```
import statsmodels.api as sm
```

[ ]:
```
data = yf.download("SPY", start="2019-01-01", end="2024-01-01")
data.head()
```

[41]:
```
#Fixing the issue with how sometimes data from yf is downloaded.
try:
    data.columns = data.columns.droplevel('Ticker')
except:
    print(" ")
```

[42]:
```
# Let's create a new dataframe, with the day-to-day percantege change data
df = data["Adj Close"].pct_change() * 100
df.head()
```

[42]:
```
Date
2019-01-02 00:00:00+00:00         NaN
2019-01-03 00:00:00+00:00   -2.386279
2019-01-04 00:00:00+00:00    3.349568
2019-01-07 00:00:00+00:00    0.788464
2019-01-08 00:00:00+00:00    0.939530
Name: Adj Close, dtype: float64
```

[43]:
```
# Renaming for clarity
df = df.rename("Today")
df = df.reset_index()
df.head()
```

[43]:
```
                       Date     Today
0 2019-01-02 00:00:00+00:00       NaN
1 2019-01-03 00:00:00+00:00 -2.386279
2 2019-01-04 00:00:00+00:00  3.349568
```

```
3 2019-01-07 00:00:00+00:00  0.788464
4 2019-01-08 00:00:00+00:00  0.939530
```

[ ]:
```
# In this model, we will predict based on lag values.
# Let's create lag values features

for i in range(1,6):
    df[f'Lag {i}'] = df["Today"].shift(i)
df.head()
```

[ ]:
```
#I'm shifting the Volume data, because for the day we predict, we will only now␣
↪the volume values from the past,
# so this actually is the Volume for the Lag 1 day
df['Volume'] = data.Volume.shift(1).values
df.head()
```

[ ]:
```
# Let's divide the volume to have more manageable values
df['Volume'] = data.Volume.values/1_000_000 # In milions
df.head()
```

[ ]:
```
# This is the final dataset
df = df.dropna()
df.head()
```

[ ]:
```
# If Today's percentage change from yesterday is positive the direction is 1␣
↪(Up), else 0 (Down)
df["Direction"] = [1 if i > 0 else 0 for i in df["Today"]]
df
```

[49]:
```
# To perform the logistic regression using the statsmodels library, we need to␣
↪add a constant
df = sm.add_constant(df)
df
```

[49]:
```
      const                      Date     Today     Lag 1     Lag 2     Lag 3  \
6       1.0 2019-01-10 00:00:00+00:00  0.352730  0.467358  0.939530  0.788464
7       1.0 2019-01-11 00:00:00+00:00  0.038640  0.352730  0.467358  0.939530
8       1.0 2019-01-14 00:00:00+00:00 -0.610067  0.038640  0.352730  0.467358
9       1.0 2019-01-15 00:00:00+00:00  1.146056 -0.610067  0.038640  0.352730
10      1.0 2019-01-16 00:00:00+00:00  0.241996  1.146056 -0.610067  0.038640
…       …                         …         …         …         …
1253    1.0 2023-12-22 00:00:00+00:00  0.200968  0.948199 -1.385735  0.608085
1254    1.0 2023-12-26 00:00:00+00:00  0.422248  0.200968  0.948199 -1.385735
1255    1.0 2023-12-27 00:00:00+00:00  0.180810  0.422248  0.200968  0.948199
1256    1.0 2023-12-28 00:00:00+00:00  0.037774  0.180810  0.422248  0.200968
1257    1.0 2023-12-29 00:00:00+00:00 -0.289494  0.037774  0.180810  0.422248
```

```
        Lag 4     Lag 5    Volume  Direction
6     3.349568 -2.386279   96.8239          1
7     0.788464  3.349568   73.8581          1
8     0.939530  0.788464   70.9082          0
9     0.467358  0.939530   85.2083          1
10    0.352730  0.467358   77.6367          1
…          …         …        …          …
1253  0.562515 -0.164661   67.1266          1
1254  0.608085  0.562515   55.3870          1
1255 -1.385735  0.608085   68.0003          1
1256  0.948199 -1.385735   77.1581          1
1257  0.200968  0.948199  122.2341          0

[1252 rows x 10 columns]
```

[50]:
```python
# Predictors
X = df[['const', 'Lag 1', 'Lag 2', 'Lag 3', 'Lag 4', 'Lag 5',
       'Volume']]
```

[51]:
```python
# Predictied variable
y = df.Direction
```

[52]:
```python
#Training the model
model = sm.Logit(y,X)
result = model.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.665339
         Iterations 5
```

[53]:
```python
# We can check the summary
result.summary()
```

[53]:

Dep. Variable:	Direction	**No. Observations:**	1252
Model:	Logit	**Df Residuals:**	1245
Method:	MLE	**Df Model:**	6
Date:	Thu, 19 Dec 2024	**Pseudo R-squ.:**	0.03371
Time:	18:08:57	**Log-Likelihood:**	-833.00
converged:	True	**LL-Null:**	-862.06
Covariance Type:	nonrobust	**LLR p-value:**	1.087e-10

	coef	std err	z	P> |z|	[0.025	0.975]
const	1.2275	0.162	7.580	0.000	0.910	1.545
Lag 1	-0.1484	0.050	-2.970	0.003	-0.246	-0.050
Lag 2	-0.0334	0.049	-0.679	0.497	-0.130	0.063
Lag 3	-0.0758	0.050	-1.528	0.126	-0.173	0.021
Lag 4	-0.1330	0.049	-2.702	0.007	-0.229	-0.037
Lag 5	-0.0576	0.049	-1.180	0.238	-0.153	0.038
Volume	-0.0120	0.002	-6.719	0.000	-0.016	-0.009

[54]:
```
#This is how we are going to analyse the results, using the confusion matrix

def confusion_matrix(act,pred):
    predicted_transformation = ["Up" if i > 0.5 else "Down" for i in pred]
    actuals_transformation = ["Up" if i > 0 else "Down" for i in act]
    confusion_matrix = pd.crosstab(pd.Series(actuals_transformation),
                                   pd.Series(predicted_transformation),␣
 ↪rownames=["Actual Values"], colnames=["Predicted Values"])
    return confusion_matrix
```

[55]:
```
prediction = result.predict(X)
confusion_matrix(y,prediction)
```

[55]:
```
Predicted Values  Down   Up
Actual Values
Down               189  377
Up                 122  564
```

[56]:
```
# Let's see how often we predict the correct answer
(confusion_matrix(y,prediction)["Down"]["Down"] +␣
 ↪confusion_matrix(y,prediction)["Up"]["Up"])/len(df) # Better than random␣
 ↪guessing
```

[56]:
```
0.6014376996805112
```

It's better than random guessing, but it's hard to compare this value with our other models, as here we predict something else.

1.4 Now, let's see these models in practice

In the first part, we saw how each model works.

In the second part I will train them on historical data of several stocks, and I will use the same strategy for each of them, with decisions made on the basis of the model's forecast.

I will compare them with each other, and also to a control group where I'm just buying the market and holding to it.

The models will be trained on the data before 2024, and tested on the 2024 data

[57]:
```
# Stocks chosen for the test
# Ford F
# Apple AAPL
# Xerox XRX
# JP Morgan JPM
# SPY500 - market
Tickers = ["F", "AAPL", "XRX", "JPM", "SPY"]
```

1.5 Strategy for MLPRegressor and Prophet

Loop: 1. Forecast the values for the next day. 2. Put the money into the stock we forecast to have the biggest rise. (If none rise, then hold) 3. Sell at the end of a day 4. Go to point 1

1.6 Strategy for Logistic regression

Loop: 1. Forecast which stocks will go up tommorow 2. Split the money equally between those stocks (if none rise, then hold) 3. Sell at the end of day 4. Go to point 1

1.7 Implementing MLPRegressor

[ ]:
```
#Downloading data
data = yf.download(Tickers, start="2020-01-01", end="2023-12-31")
data.head()
```

[59]:
```
#Test data is 2024
test_data = yf.download(Tickers, start="2024-01-01", end = "2024-12-01")
```

```
[*********************100%***********************]  5 of 5 completed
```

[60]:
```
# Removing unimportant data
data = data["Adj Close"]
test_data = test_data["Adj Close"]
data_to_copy = data.copy()
test_data_to_copy = test_data.copy()
```

[61]:
```
# Seperating the data per ticker
APPL_train = data["AAPL"]
F_train = data["F"]
JPM_train = data["JPM"]
SPY_train = data["SPY"]
XRX_train = data["XRX"]
```

[62]:
```
APPL_test = test_data["AAPL"]
F_test = test_data["F"]
JPM_test = test_data["JPM"]
SPY_test = test_data["SPY"]
XRX_test = test_data["XRX"]
```

```
[63]: # preparing the data for the MLPRegressor
MLP_APPL_train = APPL_train.copy()
MLP_F_train = F_train.copy()
MLP_JPM_train = JPM_train.copy()
MLP_SPY_train = SPY_train.copy()
MLP_XRX_train = XRX_train.copy()
```

```
[64]: MLP_vars = [MLP_XRX_train, MLP_SPY_train, MLP_JPM_train, MLP_F_train,␣
 ↪MLP_APPL_train]
```

```
[65]: # The above data is saved as a pd.Series. Let's transofrm it to df
MLP_vars_dfs = []
for var in MLP_vars:
    out = var.reset_index()
    MLP_vars_dfs.append(out)
MLP_vars_dfs[:2]
```

```
[65]: [                          Date        XRX
 0    2020-01-02 00:00:00+00:00  28.183081
 1    2020-01-03 00:00:00+00:00  27.816757
 2    2020-01-06 00:00:00+00:00  27.397024
 3    2020-01-07 00:00:00+00:00  27.435183
 4    2020-01-08 00:00:00+00:00  27.419922
 …                          …        …
 1001 2023-12-22 00:00:00+00:00  17.401197
 1002 2023-12-26 00:00:00+00:00  17.503557
 1003 2023-12-27 00:00:00+00:00  17.624529
 1004 2023-12-28 00:00:00+00:00  17.520798
 1005 2023-12-29 00:00:00+00:00  17.285049

 [1006 rows x 2 columns],
                           Date         SPY
 0    2020-01-02 00:00:00+00:00  302.208710
 1    2020-01-03 00:00:00+00:00  299.920227
 2    2020-01-06 00:00:00+00:00  301.064423
 3    2020-01-07 00:00:00+00:00  300.217926
 4    2020-01-08 00:00:00+00:00  301.817932
 …                          …          …
 1001 2023-12-22 00:00:00+00:00  469.225250
 1002 2023-12-26 00:00:00+00:00  471.206573
 1003 2023-12-27 00:00:00+00:00  472.058563
 1004 2023-12-28 00:00:00+00:00  472.236877
 1005 2023-12-29 00:00:00+00:00  470.869751

 [1006 rows x 2 columns]]
```

```
[ ]: # Adding lagged regressosrs.
resu = []
for var_df in MLP_vars_dfs:

    # Get stock name, because the name of the column we want to shift is the␣
 ↪stock ticker
    var_name = var_df.columns[1]
    # Adding 100 lags
    for i in range(1,100):
        var_df[f"lag{i}"] = var_df[var_name].shift(i)

    # Dropping NA rows
    var_df.dropna(inplace=True)
    var_df = var_df.drop("Date", axis = 1)
    resu.append(var_df)
MLP_vars_dfs = resu
resu[:2]
```

```
[67]: # AI was used here to correct bugs
# Here I'm calculating the models I will later use. Each stock has its own␣
 ↪model.
models = {}
predictions = {}

for var_df in MLP_vars_dfs:
    # Get the stock ticker
    var_name = var_df.columns[0]

    # The value we predict
    y = var_df[var_name]

    # The lagged values are the features.
    X = var_df.loc[:, var_df.columns != var_name]

    #scaling the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X)

    # Training our models, each stock has its own model
    mlp = MLPRegressor(hidden_layer_sizes=[10,10], activation='relu',␣
 ↪solver='adam', max_iter=1000, random_state=144)
    mlp.fit(X, y)

    # Now let's predict the next value
    new_df = pd.DataFrame(columns = var_df.columns)
    new_df = new_df.loc[:, new_df.columns != var_name]
```

```
    # 99 last observations, used as input for the first prediction
    list_last_obs = list(var_df[var_name].tail(99)[::-1])
    new_df.loc[len(new_df)] = list_last_obs

    # Predicting!
    pred = mlp.predict(new_df)
    models[var_name] = mlp
    predictions[var_name] = pred
```

[68]:
```
models
```

[68]:
```
{'XRX': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'SPY': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'JPM': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'F': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'AAPL': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144)}
```

[69]:
```
predictions # for the first observation after the training set
```

[69]:
```
{'XRX': array([17.25760797]),
 'SPY': array([469.40715171]),
 'JPM': array([163.67130129]),
 'F': array([11.53044852]),
 'AAPL': array([195.15648504])}
```

[70]:
```
# Trying to find which stock is predicted to have the biggest growth in the␣
 ↪next periodt
best_growth_pct = -np.Inf
best_growth_name = ""
hold_money = False

for var_df in MLP_vars_dfs:
    # Get ticker name
    var_name = var_df.columns[0]
    # get the last observation
    last_obs = list(var_df[var_name])[-1]
    # Compute the growth comparing it to the forecast
    predicted_growth = (float(predictions[var_name])-last_obs)/(last_obs)

    print(f"Analysing {var_name}, the model predicts growth of␣
 ↪{predicted_growth*100:.2f}%")
    # If the best one, keep it
```

```
    if predicted_growth > best_growth_pct:
        best_growth_pct = predicted_growth
        best_growth_name = var_name
# If none of the forecasts are positive, we will no buy anything
if best_growth_pct <= 0:
    hold_money = True
print(f"Best growth predicted today: {best_growth_name}, {best_growth_pct*100:.
 ↪2f}%")
```

```
Analysing XRX, the model predicts growth of -0.16%
Analysing SPY, the model predicts growth of -0.31%
Analysing JPM, the model predicts growth of -1.48%
Analysing F, the model predicts growth of 1.21%
Analysing AAPL, the model predicts growth of 1.86%
Best growth predicted today: AAPL, 1.86%
```

[71]:
```
# We will always start with the value of 100
wallet = 100
```

[72]:
```
# This is how i will update the data, to our train data
# I'll add values for the current day we want to predict. Then I will use them␣
 ↪as the model's input to forecast the next value.
# We only forecast one value at a time.
updated_df = pd.concat([data, test_data.head(5)])
updated_df
```

[72]:
```
Ticker                           AAPL          F         JPM         SPY  \
Date
2020-01-02 00:00:00+00:00   72.796036   7.548671  122.104614  302.208710
2020-01-03 00:00:00+00:00   72.088295   7.380390  120.493263  299.920227
2020-01-06 00:00:00+00:00   72.662704   7.340322  120.397461  301.064423
2020-01-07 00:00:00+00:00   72.320984   7.412442  118.350632  300.217926
2020-01-08 00:00:00+00:00   73.484352   7.412442  119.273895  301.817932
…                                 …          …           …           …
2024-01-02 00:00:00+00:00  184.734970  11.364589  168.066681  468.234619
2024-01-03 00:00:00+00:00  183.351761  10.944025  167.334152  464.410675
2024-01-04 00:00:00+00:00  181.023163  10.915987  168.444626  462.914764
2024-01-05 00:00:00+00:00  180.296707  11.074866  169.289749  463.548798
2024-01-08 00:00:00+00:00  184.655365  11.187016  169.044067  470.166382

Ticker                           XRX
Date
2020-01-02 00:00:00+00:00  28.183081
2020-01-03 00:00:00+00:00  27.816757
2020-01-06 00:00:00+00:00  27.397024
2020-01-07 00:00:00+00:00  27.435183
2020-01-08 00:00:00+00:00  27.419922
```

```
…                                      …
2024-01-02 00:00:00+00:00  17.002153
2024-01-03 00:00:00+00:00  14.936999
2024-01-04 00:00:00+00:00  15.625382
2024-01-05 00:00:00+00:00  15.389636
2024-01-08 00:00:00+00:00  15.644244

[1011 rows x 5 columns]
```

[73]:
```
models
```

[73]:
```
{'XRX': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'SPY': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'JPM': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'F': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144),
 'AAPL': MLPRegressor(hidden_layer_sizes=[10, 10], max_iter=1000,
random_state=144)}
```

[74]:
```
wallet = 100
wallet_history = [100]
days_of_testing=100
updated_df = data.copy()
for day in range(1, days_of_testing+1):
    data = updated_df
    #print(data.tail(3))
    APPL_train = data["AAPL"]
    F_train = data["F"]
    JPM_train = data["JPM"]
    SPY_train = data["SPY"]
    XRX_train = data["XRX"]

    # Training the MLPRegressor
    MLP_APPL_train = APPL_train.copy()
    MLP_F_train = F_train.copy()
    MLP_JPM_train = JPM_train.copy()
    MLP_SPY_train = SPY_train.copy()
    MLP_XRX_train = XRX_train.copy()

    MLP_vars = [MLP_XRX_train, MLP_SPY_train, MLP_JPM_train, MLP_F_train,
 ↪MLP_APPL_train]

    # The above data is saved as a pd.Series. Let's transofrm it to df
    MLP_vars_dfs = []
```

```
    for var in MLP_vars:
        out = var.reset_index()
        MLP_vars_dfs.append(out)

    # Adding lagged regressosrs.
    resu = []
    for var_df in MLP_vars_dfs:
        # Get stock name, because the name of the column we want to shift is␣
↪the stock ticker
        var_name = var_df.columns[1]
        # Adding 100 lags
        for i in range(1,100):
            var_df[f"lag{i}"] = var_df[var_name].shift(i)
        # Dropping NA rows
        var_df.dropna(inplace=True)
        var_df = var_df.drop("Date", axis = 1)
        resu.append(var_df)
    MLP_vars_dfs = resu
    #print(MLP_vars_dfs[0].tail(3))

    #models = {}
    predictions = {}

    # Selling the wallet
    if day > 1:
        if hold_money == False:
            for var_df in MLP_vars_dfs:
                var_name = var_df.columns[0]
                if var_name == stock_bought_name:
                    stock_curr_price = list(var_df[stock_bought_name])[-1]
                    wallet = stock_held*stock_curr_price
        else:
            wallet = wallet
    #print(wallet, stock_curr_price, stock_bought_name)


    for var_df in MLP_vars_dfs:
        var_name = var_df.columns[0]
        # The value we predict
        y = var_df[var_name]
        # The lagged values are the features.
        X = var_df.loc[:, var_df.columns != var_name]
        # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2, random_state=144)
        # We don't do the train test split, as we train on the entire data␣
↪before 2024
        scaler = StandardScaler()
```

```
        X_train_scaled = scaler.fit_transform(X)
        # That's out model
        mlp = models[var_name]
        # Now let's predict the next value
        new_df = pd.DataFrame(columns = var_df.columns)
        new_df = new_df.loc[:, new_df.columns != var_name]
        # 99 last observations, used as input for the first prediction
        list_last_obs = list(var_df[var_name].tail(99)[::-1])
        new_df.loc[len(new_df)] = list_last_obs
        # Predicting!
        pred = mlp.predict(new_df)
        predictions[var_name] = pred


    best_growth_pct = -np.Inf
    best_growth_name = ""
    hold_money = False

    for var_df in MLP_vars_dfs:
        var_name = var_df.columns[0]
        last_obs = list(var_df[var_name])[-1]
        predicted_growth = (round(float(predictions[var_name]))-last_obs)/
↪(last_obs)
        if predicted_growth > best_growth_pct:
            best_growth_pct = predicted_growth
            best_growth_name = var_name
            best_growth_price = last_obs

    if best_growth_pct <= 0:
        hold_money = True
    print(f"Iter {day}: wallet: {wallet:.2f} ")
    print(f"Best growth predicted today: {best_growth_name},␣
↪{best_growth_pct*100:.2f}%")

    if hold_money == False:
        stock_held = wallet/best_growth_price
        stock_buy_price = best_growth_price
        stock_bought_name = best_growth_name
    else:
        stock_held = wallet
        stock_buy_price = 1
        stock_bought_name = "Markets predicted to not grow"

    #print(stock_held, stock_buy_price,stock_bought_name)

    updated_df = pd.concat([data, test_data.head(day)])
    wallet_history.append(wallet)
```

```
Iter 1: wallet: 100.00
Best growth predicted today: F, 5.33%
Iter 2: wallet: 99.75
Best growth predicted today: F, 5.59%
Iter 3: wallet: 96.06
Best growth predicted today: XRX, 7.12%
Iter 4: wallet: 100.49
Best growth predicted today: AAPL, 3.30%
Iter 5: wallet: 100.09
Best growth predicted today: SPY, 1.82%
Iter 6: wallet: 101.52
Best growth predicted today: XRX, 2.27%
Iter 7: wallet: 99.43
Best growth predicted today: JPM, 2.56%
Iter 8: wallet: 99.64
Best growth predicted today: JPM, 1.15%
Iter 9: wallet: 99.22
Best growth predicted today: XRX, 5.58%
Iter 10: wallet: 99.16
Best growth predicted today: F, 2.70%
Iter 11: wallet: 99.16
Best growth predicted today: F, 2.70%
Iter 12: wallet: 97.52
Best growth predicted today: F, 4.44%
Iter 13: wallet: 95.10
Best growth predicted today: F, 7.10%
Iter 14: wallet: 96.91
Best growth predicted today: F, 5.09%
Iter 15: wallet: 96.91
Best growth predicted today: F, 5.09%
Iter 16: wallet: 98.38
Best growth predicted today: JPM, -0.04%
Iter 17: wallet: 98.38
Best growth predicted today: JPM, -0.33%
Iter 18: wallet: 98.38
Best growth predicted today: F, 3.79%
Iter 19: wallet: 98.82
Best growth predicted today: F, 3.34%
Iter 20: wallet: 100.21
Best growth predicted today: F, 1.90%
Iter 21: wallet: 102.20
Best growth predicted today: AAPL, 2.07%
Iter 22: wallet: 100.22
```

```
Best growth predicted today: AAPL, 3.54%
Iter 23: wallet: 101.56
Best growth predicted today: AAPL, 1.64%
Iter 24: wallet: 101.01
Best growth predicted today: AAPL, 3.82%
Iter 25: wallet: 102.01
Best growth predicted today: XRX, 7.30%
Iter 26: wallet: 105.96
Best growth predicted today: XRX, 3.29%
Iter 27: wallet: 103.10
Best growth predicted today: XRX, 0.27%
Iter 28: wallet: 105.96
Best growth predicted today: AAPL, 0.32%
Iter 29: wallet: 106.40
Best growth predicted today: XRX, 3.29%
Iter 30: wallet: 107.95
Best growth predicted today: AAPL, 0.82%
Iter 31: wallet: 106.73
Best growth predicted today: AAPL, 4.68%
Iter 32: wallet: 106.22
Best growth predicted today: AAPL, 4.10%
Iter 33: wallet: 106.05
Best growth predicted today: AAPL, 2.62%
Iter 34: wallet: 105.16
Best growth predicted today: XRX, 2.68%
Iter 35: wallet: 106.40
Best growth predicted today: F, 2.06%
Iter 36: wallet: 105.45
Best growth predicted today: F, 2.99%
Iter 37: wallet: 105.27
Best growth predicted today: F, 3.16%
Iter 38: wallet: 105.45
Best growth predicted today: F, 2.99%
Iter 39: wallet: 103.80
Best growth predicted today: F, 4.62%
Iter 40: wallet: 104.23
Best growth predicted today: F, 4.19%
Iter 41: wallet: 106.84
Best growth predicted today: XRX, 1.75%
Iter 42: wallet: 106.21
Best growth predicted today: F, 0.50%
Iter 43: wallet: 106.30
Best growth predicted today: AAPL, 1.11%
Iter 44: wallet: 103.60
Best growth predicted today: AAPL, 5.46%
Iter 45: wallet: 100.65
Best growth predicted today: AAPL, 9.14%
Iter 46: wallet: 100.06
```

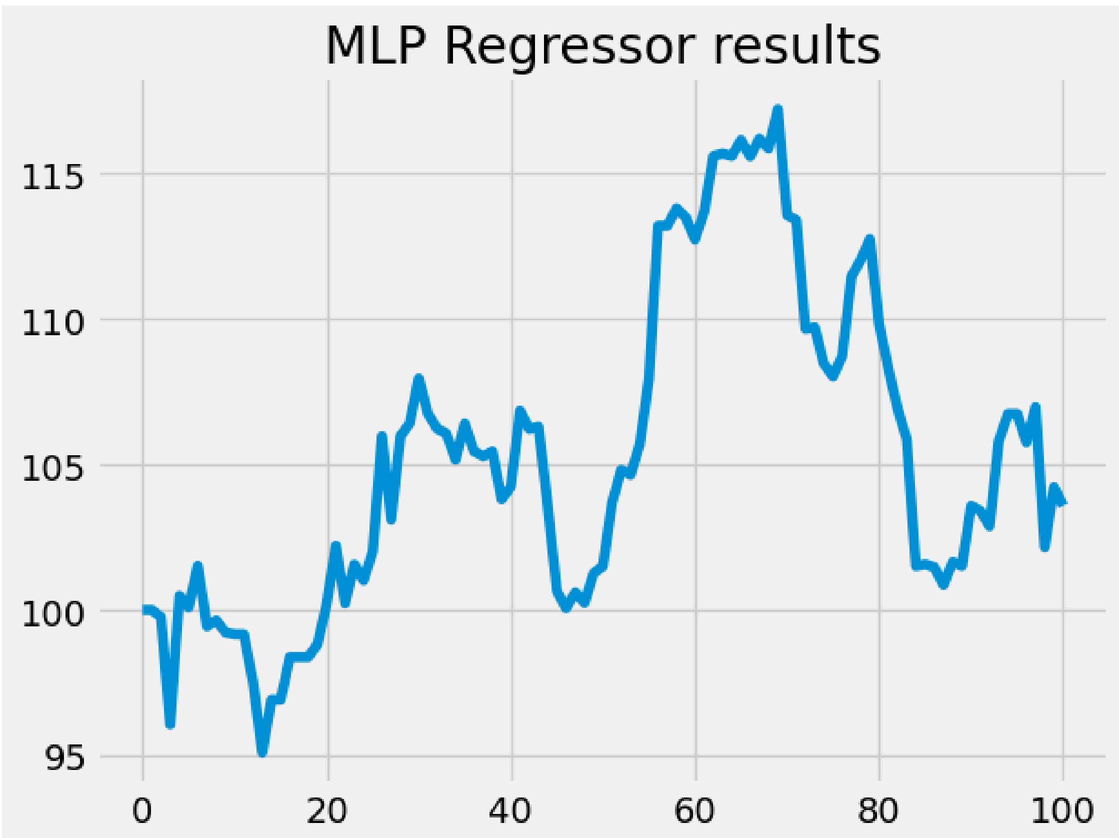
```
Best growth predicted today: XRX, 8.14%
Iter 47: wallet: 100.60
Best growth predicted today: XRX, 7.56%
Iter 48: wallet: 100.24
Best growth predicted today: XRX, 7.95%
Iter 49: wallet: 101.26
Best growth predicted today: XRX, 6.86%
Iter 50: wallet: 101.50
Best growth predicted today: F, 3.24%
Iter 51: wallet: 103.68
Best growth predicted today: AAPL, 2.04%
Iter 52: wallet: 104.81
Best growth predicted today: F, 3.50%
Iter 53: wallet: 104.64
Best growth predicted today: F, 3.67%
Iter 54: wallet: 105.68
Best growth predicted today: XRX, 3.02%
Iter 55: wallet: 107.93
Best growth predicted today: F, 1.65%
Iter 56: wallet: 113.19
Best growth predicted today: SPY, -0.24%
Iter 57: wallet: 113.19
Best growth predicted today: AAPL, 1.32%
Iter 58: wallet: 113.79
Best growth predicted today: SPY, 1.75%
Iter 59: wallet: 113.48
Best growth predicted today: AAPL, 1.04%
Iter 60: wallet: 112.72
Best growth predicted today: SPY, 1.64%
Iter 61: wallet: 113.67
Best growth predicted today: F, 3.71%
Iter 62: wallet: 115.58
Best growth predicted today: F, 1.99%
Iter 63: wallet: 115.67
Best growth predicted today: F, 1.91%
Iter 64: wallet: 115.58
Best growth predicted today: AAPL, 1.65%
Iter 65: wallet: 116.14
Best growth predicted today: AAPL, 1.16%
Iter 66: wallet: 115.57
Best growth predicted today: F, 2.53%
Iter 67: wallet: 116.18
Best growth predicted today: XRX, 2.85%
Iter 68: wallet: 115.85
Best growth predicted today: XRX, 3.15%
Iter 69: wallet: 117.19
Best growth predicted today: XRX, 1.97%
Iter 70: wallet: 113.56
```

```
Best growth predicted today: F, 3.71%
Iter 71: wallet: 113.39
Best growth predicted today: F, 3.87%
Iter 72: wallet: 109.65
Best growth predicted today: JPM, 9.59%
Iter 73: wallet: 109.71
Best growth predicted today: F, 10.75%
Iter 74: wallet: 108.45
Best growth predicted today: JPM, 5.20%
Iter 75: wallet: 108.02
Best growth predicted today: JPM, 6.18%
Iter 76: wallet: 108.72
Best growth predicted today: JPM, 4.94%
Iter 77: wallet: 111.45
Best growth predicted today: AAPL, 4.01%
Iter 78: wallet: 112.02
Best growth predicted today: AAPL, 2.27%
Iter 79: wallet: 112.73
Best growth predicted today: XRX, 6.24%
Iter 80: wallet: 109.83
Best growth predicted today: XRX, 9.05%
Iter 81: wallet: 108.30
Best growth predicted today: XRX, 3.22%
Iter 82: wallet: 106.93
Best growth predicted today: XRX, 12.01%
Iter 83: wallet: 105.86
Best growth predicted today: XRX, 5.60%
Iter 84: wallet: 101.51
Best growth predicted today: JPM, 3.42%
Iter 85: wallet: 101.57
Best growth predicted today: JPM, 4.41%
Iter 86: wallet: 101.46
Best growth predicted today: JPM, 2.93%
Iter 87: wallet: 100.85
Best growth predicted today: JPM, 1.96%
Iter 88: wallet: 101.64
Best growth predicted today: JPM, 1.17%
Iter 89: wallet: 101.51
Best growth predicted today: JPM, 1.83%
Iter 90: wallet: 103.58
Best growth predicted today: F, 1.67%
Iter 91: wallet: 103.41
Best growth predicted today: XRX, 7.54%
Iter 92: wallet: 102.87
Best growth predicted today: F, 3.02%
Iter 93: wallet: 105.79
Best growth predicted today: F, 0.18%
Iter 94: wallet: 106.73
```

```
Best growth predicted today: JPM, -0.09%
Iter 95: wallet: 106.73
Best growth predicted today: XRX, 3.44%
Iter 96: wallet: 105.75
Best growth predicted today: JPM, 0.94%
Iter 97: wallet: 106.97
Best growth predicted today: JPM, 0.78%
Iter 98: wallet: 102.16
Best growth predicted today: JPM, 3.46%
Iter 99: wallet: 104.21
Best growth predicted today: JPM, 1.92%
Iter 100: wallet: 103.58
Best growth predicted today: F, 2.68%
```

[75]:
```
#Let's visualize how we did!
plt.plot(wallet_history)
plt.title("MLP Regressor results")
```

[75]:
```
Text(0.5, 1.0, 'MLP Regressor results')
```



[76]:
```
print(f"Final value: {wallet_history[-1]:.2f}")
MLP_wallet_history = wallet_history.copy()
```

```
Final value: 103.58
```

To summarize MLP regressor's performance - It didn't make much money, and the gueses sometime can be very wrong. As we will see latet, it is worse than just following the market.

1.8 Prophet

Now I'll use Meta's Prophet. Here I will make one model for each ticker at the beginning. Then I will make one forecast for the entire period. Then I'll just compere the day-on-day percentage growth in the same way I did with the MLP.

It's important to notice here, that I will not be updating the models with historical data each day, because that would be too computationally intensive (especially for Noto).

[77]:
```
#Downloading data
data = yf.download(Tickers, start="2020-01-01", end="2023-12-31")
test_data = yf.download(Tickers, start="2024-01-01", end = "2024-12-01")
data = data["Adj Close"]
test_data = test_data["Adj Close"]
```

```
[*********************100%***********************]  5 of 5 completed
[*********************100%***********************]  5 of 5 completed
```

[78]:
```
# Seperating the data per ticker
APPL_train = data["AAPL"]
F_train = data["F"]
JPM_train = data["JPM"]
SPY_train = data["SPY"]
XRX_train = data["XRX"]
APPL_test = test_data["AAPL"]
F_test = test_data["F"]
JPM_test = test_data["JPM"]
SPY_test = test_data["SPY"]
XRX_test = test_data["XRX"]
```

[79]:
```
# preparing the data for the Prophet
Prophet_APPL_train = APPL_train.copy()
Prophet_F_train = F_train.copy()
Prophet_JPM_train = JPM_train.copy()
Prophet_SPY_train = SPY_train.copy()
Prophet_XRX_train = XRX_train.copy()
```

[80]:
```
Prophet_vars = [Prophet_APPL_train, Prophet_F_train, Prophet_JPM_train,␣
↪Prophet_SPY_train, Prophet_XRX_train]
```

[81]:
```python
# The library asks for variables to be named a certain way
Prophet_vars_dfs = {}
for var in Prophet_vars:
    var_name = var.name
    var = var.rename("Close")
    var = var.reset_index().rename(columns = {"Date" : "ds", "Close" : "y"})
    var["ds"] = var["ds"].dt.tz_localize(None)
    Prophet_vars_dfs[var_name] = var
```

[82]:
```python
# Dictionary to store the results
models_prophet = {}

# Creating the models for each ticker
for var_name in Prophet_vars_dfs.keys():
    print(var_name)
    var_df = Prophet_vars_dfs[var_name]
    model = Prophet()
    model.fit(var_df)
    models_prophet[var_name] = model
```

```
18:09:39 - cmdstanpy - INFO - Chain [1] start processing
```

```
AAPL
```

```
18:09:39 - cmdstanpy - INFO - Chain [1] done processing
```

```
F
```

```
18:09:39 - cmdstanpy - INFO - Chain [1] start processing
18:09:39 - cmdstanpy - INFO - Chain [1] done processing
18:09:40 - cmdstanpy - INFO - Chain [1] start processing
```

```
JPM
```

```
18:09:40 - cmdstanpy - INFO - Chain [1] done processing
18:09:40 - cmdstanpy - INFO - Chain [1] start processing
```

```
SPY
```

```
18:09:41 - cmdstanpy - INFO - Chain [1] done processing
```

```
XRX
```

```
18:09:41 - cmdstanpy - INFO - Chain [1] start processing
18:09:41 - cmdstanpy - INFO - Chain [1] done processing
```

[83]:
```python
# here I'm using the models to predict the values and to store them as a Data␣
↪Frame.

# Dictionary to store the results
results = {}
```

```python
results_df = pd.DataFrame()

for var_name in Prophet_vars_dfs.keys():

    model = models_prophet[var_name]

    future = model.make_future_dataframe(periods=101, freq = "C",␣
↪include_history=False)
    # Predicting !
    forecast = model.predict(future)

    results[var_name] = forecast
    results_df[var_name] = forecast["yhat"]
    #results_df[f"{var_name}_pct_change"] = forecast["yhat"].pct_change()

# These are our forcasted values for each day, for the next 100 days. Based on␣
↪the data availible on the 0th day
results_df
```

[83]:
```
           AAPL          F         JPM         SPY        XRX
0    193.831202  11.703568  162.315491  459.483513  15.808759
1    193.614098  11.790007  162.845349  459.488226  15.836372
2    193.615755  11.872878  163.194897  459.577747  15.865324
3    193.348028  11.962227  163.594140  459.274211  15.820706
4    193.215069  12.026085  164.047488  459.284999  15.820045
..          …          …           …           …          …
96   208.750043  10.694943  164.067826  479.737135  14.531313
97   208.865462  10.727571  164.324764  480.301646  14.588176
98   208.727142  10.775655  164.721593  480.584688  14.578615
99   208.735682  10.807985  165.266036  481.282591  14.619604
100  209.296750  10.981742  166.642272  483.275450  14.759475

[101 rows x 5 columns]
```

[84]:
```python
# These are actual values for 2024.
test_data
```

[84]:
```
Ticker                           AAPL          F         JPM         SPY  \
Date
2024-01-02 00:00:00+00:00  184.734970  11.364589  168.066681  468.234619
2024-01-03 00:00:00+00:00  183.351761  10.944025  167.334152  464.410675
2024-01-04 00:00:00+00:00  181.023163  10.915987  168.444626  462.914764
2024-01-05 00:00:00+00:00  180.296707  11.074866  169.289749  463.548798
2024-01-08 00:00:00+00:00  184.655365  11.187016  169.044067  470.166382
…                                   …          …           …           …
2024-11-22 00:00:00+00:00  229.869995  11.180000  248.550003  595.510010
2024-11-25 00:00:00+00:00  232.869995  11.400000  250.289993  597.530029
```

```
2024-11-26 00:00:00+00:00  235.059998  11.100000  249.970001  600.650024
2024-11-27 00:00:00+00:00  234.929993  11.100000  249.789993  598.830017
2024-11-29 00:00:00+00:00  237.330002  11.130000  249.720001  602.549988

Ticker                           XRX
Date
2024-01-02 00:00:00+00:00  17.002153
2024-01-03 00:00:00+00:00  14.936999
2024-01-04 00:00:00+00:00  15.625382
2024-01-05 00:00:00+00:00  15.389636
2024-01-08 00:00:00+00:00  15.644244
…                                …
2024-11-22 00:00:00+00:00   9.040000
2024-11-25 00:00:00+00:00   9.160000
2024-11-26 00:00:00+00:00   9.080000
2024-11-27 00:00:00+00:00   9.060000
2024-11-29 00:00:00+00:00   9.140000

[231 rows x 5 columns]
```

[85]:
```
# To calculate the first predicted pct change i need to add the last historical value to the data frame
historical_data_df = pd.DataFrame(columns = results_df.columns)

# A df for with just the last historical values for each ticker
historical_data_df.loc[len(historical_data_df)] = [df[-1] for df in Prophet_vars] # This line was suggested by AI
historical_data_df
```

[85]:
```
       AAPL          F         JPM         SPY        XRX
0  191.5914  11.392626  166.132858  470.869751  17.285049
```

[86]:
```
results_df_with_hist = pd.concat([historical_data_df, results_df])
```

[87]:
```
results_df_with_hist
```

[87]:
```
          AAPL          F         JPM         SPY        XRX
0   191.591400  11.392626  166.132858  470.869751  17.285049
0   193.831202  11.703568  162.315491  459.483513  15.808759
1   193.614098  11.790007  162.845349  459.488226  15.836372
2   193.615755  11.872878  163.194897  459.577747  15.865324
3   193.348028  11.962227  163.594140  459.274211  15.820706
..         …          …          …          …          …
96  208.750043  10.694943  164.067826  479.737135  14.531313
97  208.865462  10.727571  164.324764  480.301646  14.588176
98  208.727142  10.775655  164.721593  480.584688  14.578615
99  208.735682  10.807985  165.266036  481.282591  14.619604
```

```
100  209.296750  10.981742  166.642272  483.275450  14.759475

[102 rows x 5 columns]
```

[88]:
```python
# Calculating the day-on-day pct change
pct_change_df = results_df.pct_change().dropna()
pct_change_df.head(10)
```

[88]:

	AAPL	F	JPM	SPY	XRX
1	-0.001120	0.007386	0.003264	0.000010	0.001747
2	0.000009	0.007029	0.002147	0.000195	0.001828
3	-0.001383	0.007526	0.002446	-0.000660	-0.002812
4	-0.000688	0.005338	0.002771	0.000023	-0.000042
5	0.000551	0.017030	0.003309	-0.001156	-0.001132
6	-0.000856	0.004848	0.000759	-0.000501	0.000200
7	0.000418	0.004052	-0.000469	-0.000215	0.000455
8	-0.000830	0.004130	-0.000240	-0.000956	-0.004006
9	0.000013	0.001685	0.000078	-0.000142	-0.001021
10	0.003482	0.004550	-0.004157	-0.000804	-0.002670

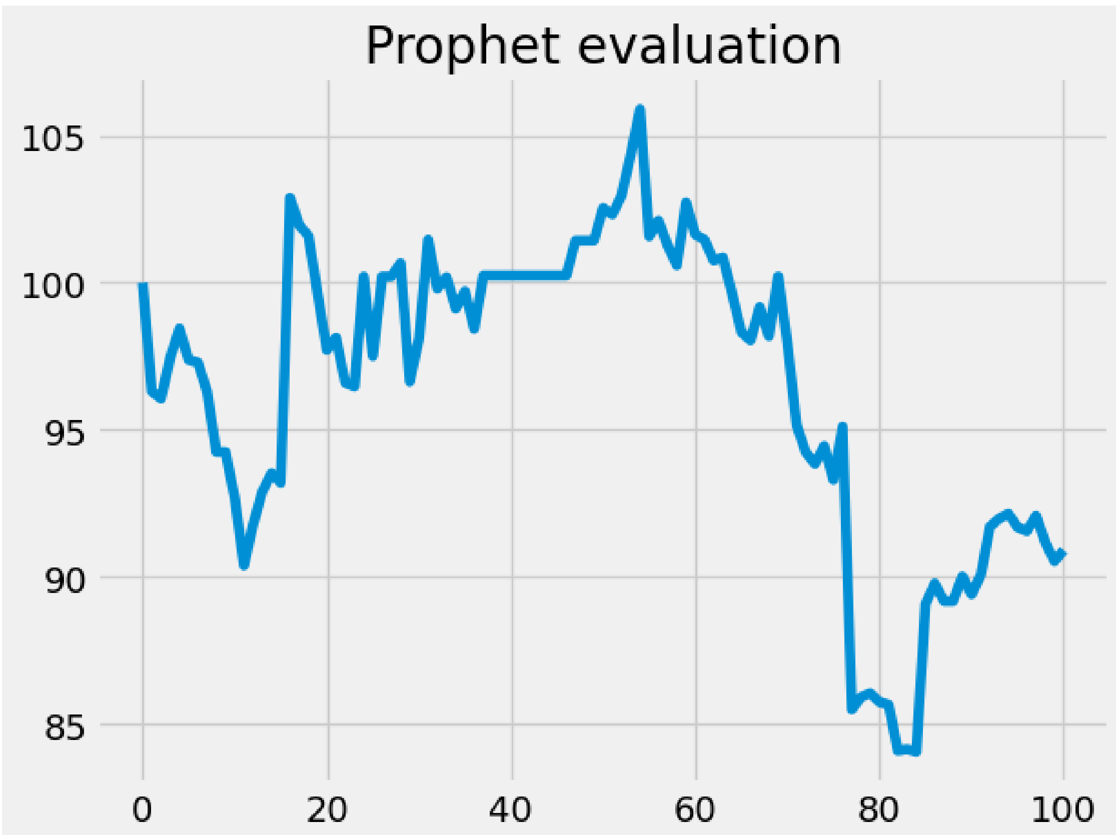
[89]:
```python
# Evaluating the model
wallet = 100
wallet_history = [100]

for i in range(len(pct_change_df)):

    analysed_row = pct_change_df.loc[i+1]
    # Searching for the stock with the highest predicted pct change
    best_stock = analysed_row.idxmax() # The use of .idxmax was suggested by AI
    # Checking its current stock price
    curr_price_of_best_stock = float(test_data[best_stock].iloc[i])
    # if the pct change is bigger than zero
    if analysed_row.max() > 0:
        # How many shares can i buy
        n_shares_bought = wallet/curr_price_of_best_stock
        # And how much will it be worth the next day
        wallet = n_shares_bought*float(test_data[best_stock].iloc[i+1])

    wallet_history.append(wallet)
    # Here I print the current result
    if i % 10 == 0:
        print(f"Day {i}, wallet: {wallet:.2f}")
```

```
Day 0, wallet: 96.30
Day 10, wallet: 90.38
```

```
Day 20, wallet: 98.13
Day 30, wallet: 101.46
Day 40, wallet: 100.24
Day 50, wallet: 102.31
Day 60, wallet: 101.46
Day 70, wallet: 95.16
Day 80, wallet: 85.65
Day 90, wallet: 90.05
```

[90]:
```python
plt.plot(wallet_history)
plt.title("Prophet evaluation")
```

[90]:
```
Text(0.5, 1.0, 'Prophet evaluation')
```



[91]:
```python
print(f"The wallet value at the end of testing {wallet_history[-1]:.2f}")
Prophet_wallet_history = wallet_history.copy()
```

```
The wallet value at the end of testing 90.90
```

Well..., The results are even worse. This may be on part based on the fact that we have not updated the model with current data in any way. Prophet doesn't work like a ML model with an input and output. Rather it can only forecast on the time series it modeled on.

1.9 Logistic regression

[92]:
```python
#Downloading data
data = yf.download(Tickers, start="2020-01-01", end="2023-12-31")
test_data = yf.download(Tickers, start="2023-12-29", end = "2024-12-01")
data = data["Adj Close"]
test_data = test_data["Adj Close"]
```

```
[*********************100%***********************]  5 of 5 completed
[*********************100%***********************]  5 of 5 completed
```

[93]:
```python
# Seperating the data per ticker
APPL_train = data["AAPL"]
F_train = data["F"]
JPM_train = data["JPM"]
SPY_train = data["SPY"]
XRX_train = data["XRX"]
APPL_test = test_data["AAPL"]
F_test = test_data["F"]
JPM_test = test_data["JPM"]
SPY_test = test_data["SPY"]
XRX_test = test_data["XRX"]
```

[94]:
```python
Logistic_data_dfs = [APPL_train, F_train, JPM_train, SPY_train, XRX_train]
```

[95]:
```python
# Training the models for each ticker
models_logistic = {}
for var_df in Logistic_data_dfs:
    var_name = var_df.name
    # Getting the pct change
    df = var_df.pct_change()
    df = df.rename("Today")
    # Changing as pd.Series to a pd.DataFrame
    df = df.reset_index()
    # Adding lags
    for i in range(1,6):
        df[f'Lag {i}'] = df["Today"].shift(i)

    df = df.dropna()
    # If today's pct change is bigger than 0 then the direction 1 (Up) else 0␣
 ↪(Down)
    df["Direction"] = [1 if i > 0 else 0 for i in df["Today"]]
    # We ahve to add a constant for the Logistic model
    df = sm.add_constant(df)
    X = df[['const', 'Lag 1', 'Lag 2', 'Lag 3', 'Lag 4', 'Lag 5']]
    y = df.Direction
    # Fitting the model
```

```
    model = sm.Logit(y,X).fit()
    models_logistic[var_name] = model
    #print(df)
```

```
Optimization terminated successfully.
         Current function value: 0.688876
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.688585
         Iterations 4
Optimization terminated successfully.
         Current function value: 0.689860
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.687626
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.690472
         Iterations 4
```

[96]:
```
models_logistic
```

[96]:
```
{'AAPL': <statsmodels.discrete.discrete_model.BinaryResultsWrapper at
0x7fe594ab4610>,
 'F': <statsmodels.discrete.discrete_model.BinaryResultsWrapper at
0x7fe594350110>,
 'JPM': <statsmodels.discrete.discrete_model.BinaryResultsWrapper at
0x7fe5948b29d0>,
 'SPY': <statsmodels.discrete.discrete_model.BinaryResultsWrapper at
0x7fe587dd5210>,
 'XRX': <statsmodels.discrete.discrete_model.BinaryResultsWrapper at
0x7fe587d657d0>}
```

[97]:
```
Logistic_data_dfs = [APPL_train, F_train, JPM_train, SPY_train, XRX_train]
Logistic_data_test_dfs = [APPL_test, F_test, JPM_test, SPY_test, XRX_test]
```

[98]:
```
# Preparing the test, creating a df with the same foremat
test_dfs_logistic = {}
for i in range(len(Logistic_data_test_dfs)):
    var_df = pd.concat([Logistic_data_dfs[i].tail(5),␣
 ↪Logistic_data_test_dfs[i]]) # This line was created by AI
    var_name = var_df.name
    df = var_df.pct_change()
    df = df.rename("Today")
    df = df.reset_index()
    for i in range(1,6):
        df[f'Lag {i}'] = df["Today"].shift(i)
```

```
    df = df.dropna()
    #df["Direction"] = [1 if i > 0 else 0 for i in df["Today"]]
    df = sm.add_constant(df)
    test_dfs_logistic[var_name] = df
```

[99]:
```
# An example
test_dfs_logistic["SPY"]
```

[99]:
```
     const                      Date     Today     Lag 1     Lag 2     Lag 3  \
6      1.0 2024-01-02 00:00:00+00:00 -0.005596  0.000000 -0.002895  0.000378
7      1.0 2024-01-03 00:00:00+00:00 -0.008167 -0.005596  0.000000 -0.002895
8      1.0 2024-01-04 00:00:00+00:00 -0.003221 -0.008167 -0.005596  0.000000
9      1.0 2024-01-05 00:00:00+00:00  0.001370 -0.003221 -0.008167 -0.005596
10     1.0 2024-01-08 00:00:00+00:00  0.014276  0.001370 -0.003221 -0.008167
..     …                         …       …       …       …       …
232    1.0 2024-11-22 00:00:00+00:00  0.003099  0.005368  0.000339  0.003655
233    1.0 2024-11-25 00:00:00+00:00  0.003392  0.003099  0.005368  0.000339
234    1.0 2024-11-26 00:00:00+00:00  0.005221  0.003392  0.003099  0.005368
235    1.0 2024-11-27 00:00:00+00:00 -0.003030  0.005221  0.003392  0.003099
236    1.0 2024-11-29 00:00:00+00:00  0.006212 -0.003030  0.005221  0.003392

        Lag 4     Lag 5
6    0.001808  0.004223
7    0.000378  0.001808
8   -0.002895  0.000378
9    0.000000 -0.002895
10  -0.005596  0.000000
..        …       …
232  0.004097 -0.012809
233  0.003655  0.004097
234  0.000339  0.003655
235  0.005368  0.000339
236  0.003099  0.005368

[231 rows x 8 columns]
```

[100]:
```
# Predicting the values and saving it as a df
results_df_all_stocks = pd.DataFrame()

# For every stock
for var_name in test_dfs_logistic.keys():

    # Copy the test df
    results_df = test_dfs_logistic[var_name].copy()
    X = test_dfs_logistic[var_name].drop(["Date", "Today"], axis = 1)
```

```
    model = models_logistic[var_name]
    #Predicting
    predictions = model.predict(X)

    # If prediction is bigger than 0.5, we predict Up, else Down
    results_df["Prediction"] = (predictions>0.5).astype(int)

    results_df = results_df[["Date", "Prediction"]]
    results_df.Date = results_df.Date.dt.tz_localize(None) # This line was␣
↪added by AI to fix a bug
    #Saving all results as a df
    if results_df_all_stocks.empty: #use of .empty was suggested by AI
        results_df_all_stocks = results_df.copy()
        results_df_all_stocks["Prediction"].rename({"Prediction":var_name},␣
↪inplace = True)
    else:
        results_df_all_stocks[var_name] = results_df["Prediction"]
```

[101]:
```
results_df_all_stocks.rename(columns={"Prediction": "AAPL"}, inplace=True)

results_df_all_stocks
```

[101]:
```
           Date  AAPL  F  JPM  SPY  XRX
6    2024-01-02     1  0    0    1    1
7    2024-01-03     1  0    1    1    1
8    2024-01-04     1  0    1    1    1
9    2024-01-05     1  1    0    1    1
10   2024-01-08     1  0    1    1    1
..          ...   ... ..  ...  ...  ...
232  2024-11-22     1  1    0    1    0
233  2024-11-25     1  0    1    1    0
234  2024-11-26     1  0    1    1    0
235  2024-11-27     1  0    0    1    1
236  2024-11-29     1  1    0    1    1

[231 rows x 6 columns]
```

If we predict some stocks to go up, we split the money between them

[102]:
```
APPL_test = test_data["AAPL"]
F_test = test_data["F"]
JPM_test = test_data["JPM"]
SPY_test = test_data["SPY"]
XRX_test = test_data["XRX"]
```

[103]:
```
combined_df = pd.concat([APPL_test, F_test, JPM_test, SPY_test, XRX_test], axis␣
↪= 1) # use of pd.concact(axis = 1) was suggested by AI
```

[104]:
```
# Getting the actual prices for each day in the test sample
combined_df
```

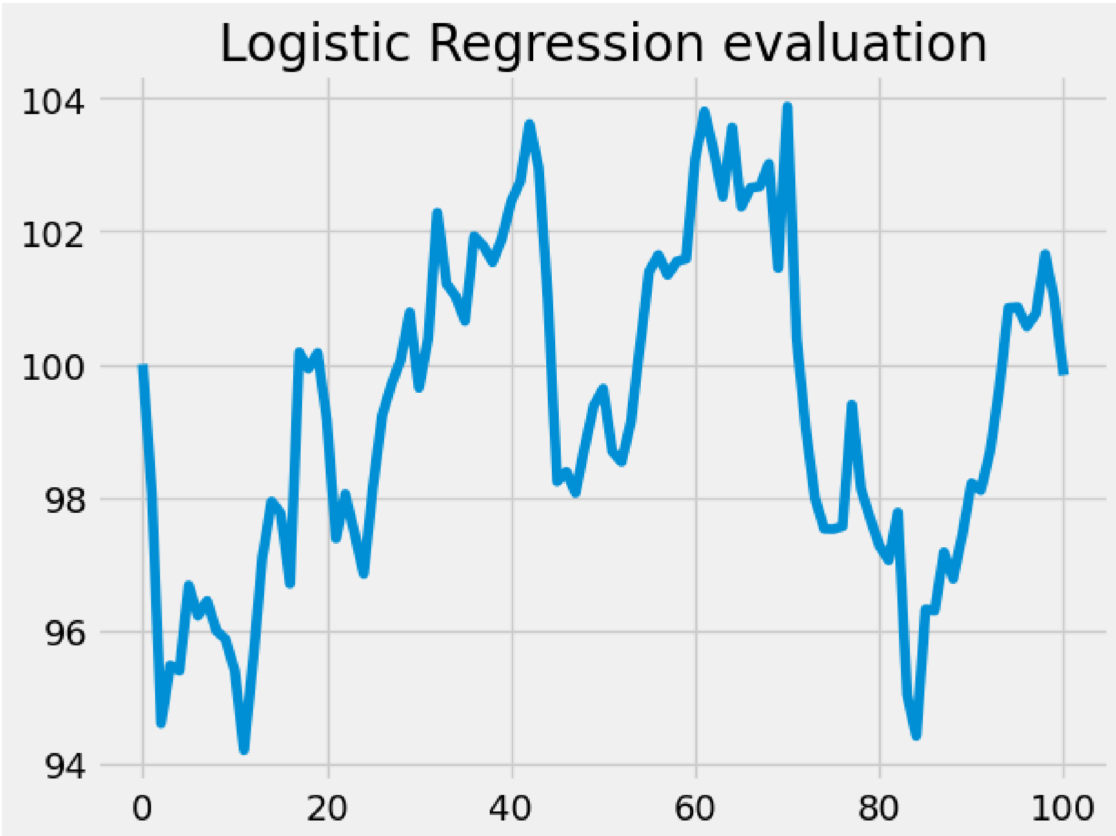
[104]:
```
                                 AAPL          F         JPM         SPY  \
Date
2023-12-29 00:00:00+00:00  191.591385  11.392626  166.132843  470.869751
2024-01-02 00:00:00+00:00  184.734985  11.364588  168.066666  468.234589
2024-01-03 00:00:00+00:00  183.351761  10.944024  167.334167  464.410675
2024-01-04 00:00:00+00:00  181.023178  10.915987  168.444626  462.914764
2024-01-05 00:00:00+00:00  180.296707  11.074867  169.289749  463.548798
…                                 …          …           …           …
2024-11-22 00:00:00+00:00  229.869995  11.180000  248.550003  595.510010
2024-11-25 00:00:00+00:00  232.869995  11.400000  250.289993  597.530029
2024-11-26 00:00:00+00:00  235.059998  11.100000  249.970001  600.650024
2024-11-27 00:00:00+00:00  234.929993  11.100000  249.789993  598.830017
2024-11-29 00:00:00+00:00  237.330002  11.130000  249.720001  602.549988

                                 XRX
Date
2023-12-29 00:00:00+00:00  17.285051
2024-01-02 00:00:00+00:00  17.002155
2024-01-03 00:00:00+00:00  14.937000
2024-01-04 00:00:00+00:00  15.625383
2024-01-05 00:00:00+00:00  15.389636
…                                …
2024-11-22 00:00:00+00:00   9.040000
2024-11-25 00:00:00+00:00   9.160000
2024-11-26 00:00:00+00:00   9.080000
2024-11-27 00:00:00+00:00   9.060000
2024-11-29 00:00:00+00:00   9.140000

[232 rows x 5 columns]
```

[105]:
```
days_to_check=100
wallet = 100
wallet_history = [wallet]

for i in range(days_to_check):
    # Our predictions
    our_preds = results_df_all_stocks

    # if any values are predicted to go up, we'll invest
    if((our_preds.iloc[i] == 1).any()): #This line was suggested by AI. Prompt:
 ↪"if any(our_preds.iloc[I] == 0 ) then continue"
        invest_money = True
    else:
        invest_money = False
```

```
    if invest_money == True:
        # Money in our wallet divided by the number of stock we will invest in
        how_much_to_invest_per_stock = wallet/((our_preds.iloc[i] == 1).sum())

        # Stocks we will invest in
        stocks_to_invest = our_preds.iloc[i][our_preds.iloc[i] == 1].index.
↪tolist() # This line was suggested by AI. Prompt: "get me var_names with 1"

        # The current and next prices
        current_prices = combined_df.iloc[i]
        next_prices = combined_df.iloc[i+1]

        # The value of our porfolio in the next period
        value_next_period = 0

        for stock_to_buy in stocks_to_invest:
            # The stock's current price
            curr_price = current_prices[stock_to_buy]
            # How many shares we can buy
            stocks_bought = how_much_to_invest_per_stock/curr_price
            # The value of the shares the next day
            value_next_period += stocks_bought*next_prices[stock_to_buy]
        wallet = value_next_period
    else:
        # We do not invest
        wallet = wallet # This is of course trivial, but spelled out for clarity
    wallet_history.append(wallet)
```

[106]:
```
plt.plot(wallet_history)
plt.title("Logistic Regression evaluation")
```

[106]: Text(0.5, 1.0, 'Logistic Regression evaluation')



[107]:
```
print(f"Final value for the Logistic Regression wallet {wallet_history[-1]:.
↪2f}")
Logistic_wallet_history = wallet_history.copy()
```

```
Final value for the Logistic Regression wallet 99.84
```

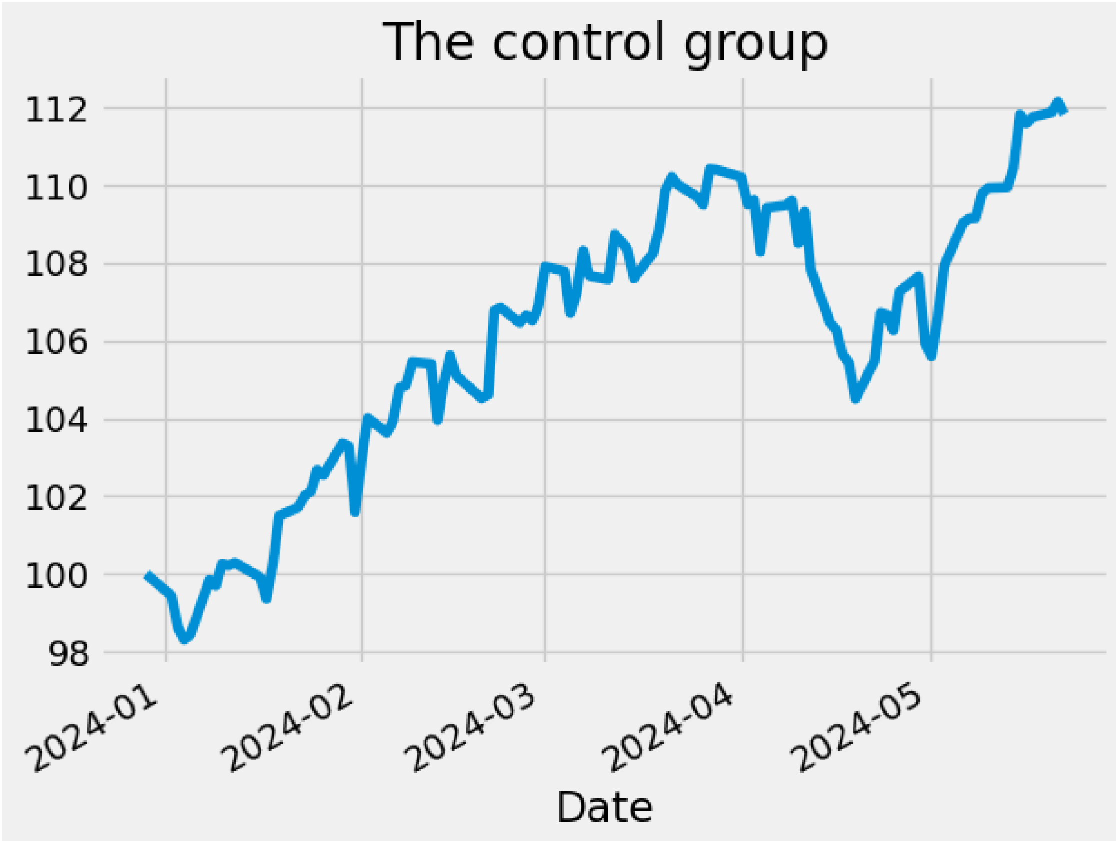
Turns out we left the same amount of money we started with :)

1.10 Just holding SPY

This is our control group. How much money would we make if bought the market, and kept it the entire time

[108]:
```
(SPY_test/(SPY_test[0]/100))[:100].plot(title = "The control group")
```

[108]:
```
<Axes: title={'center': 'The control group'}, xlabel='Date'>
```



[109]:
```
print(f"The wallet's value at the end: {(SPY_test/(SPY_test[0]/100))[99]:.2f}")
```

```
The wallet's value at the end: 111.82
```

Let's plot the comparison

[110]:
```
plot_df = (SPY_test/(SPY_test[0]/100))[:101]
plot_df = plot_df.reset_index()
plot_df["MLPRegressor"] = MLP_wallet_history
plot_df["Prophet"] = Prophet_wallet_history
plot_df["Logistic Regression"] = Logistic_wallet_history
plot_df
```

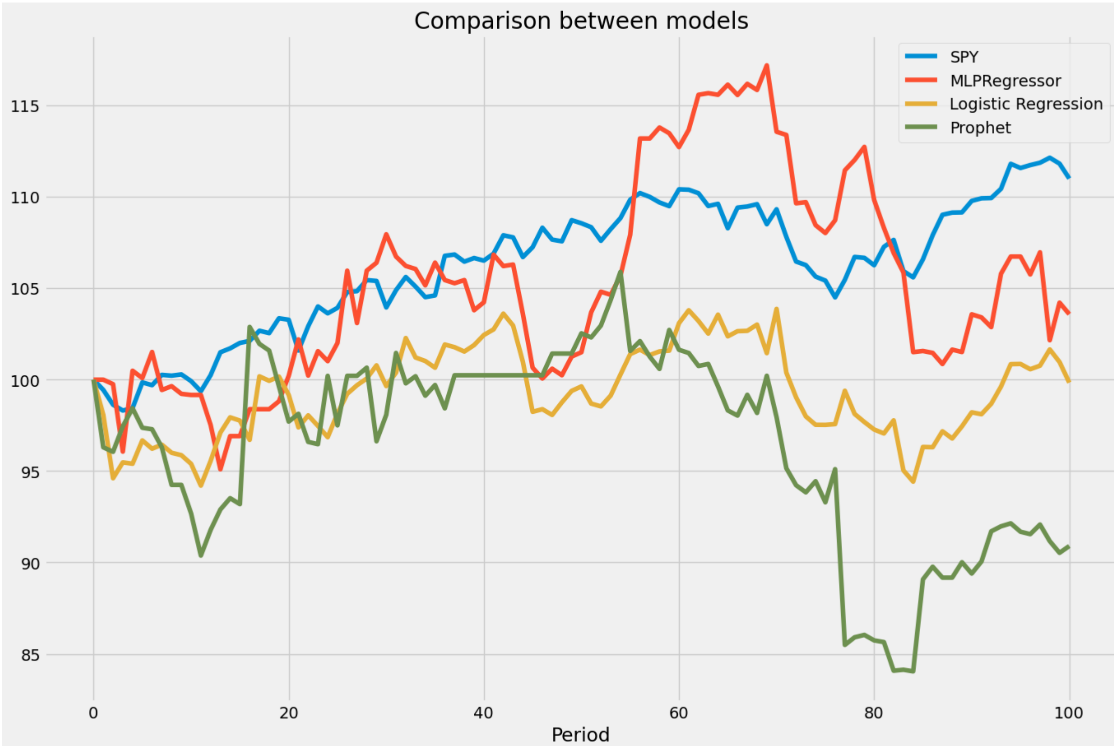
[110]:

	Date	SPY	MLPRegressor	Prophet	\
0	2023-12-29 00:00:00+00:00	100.000000	100.000000	100.000000	
1	2024-01-02 00:00:00+00:00	99.440363	100.000000	96.299350	
2	2024-01-03 00:00:00+00:00	98.628267	99.753902	96.052636	
3	2024-01-04 00:00:00+00:00	98.310576	96.062358	97.450656	
4	2024-01-05 00:00:00+00:00	98.445228	100.489466	98.437495	
..	…	…	…	…	
96	2024-05-17 00:00:00+00:00	111.736561	105.753959	91.550012	

```
97   2024-05-20 00:00:00+00:00  111.865289    106.965734   92.075783
98   2024-05-21 00:00:00+00:00  112.139634    102.155178   91.166395
99   2024-05-22 00:00:00+00:00  111.816758    104.213124   90.527384
100  2024-05-23 00:00:00+00:00  111.000022    103.581112   90.901158

     Logistic Regression
0             100.000000
1              98.075018
2              94.606179
3              95.476574
4              95.400763
..                   …
96            100.569457
97            100.766593
98            101.653288
99            100.965629
100            99.837237

[101 rows x 5 columns]
```

[111]:

```
fig, ax = plt.subplots(figsize=(15,10))
plot_df["SPY"].plot(label = "SPY", title = "Comparison between models", ax =␣
 ↪ax, legend = True)
plot_df["MLPRegressor"].plot(label = "MLPRegressor", ax = ax , legend = True)
plot_df["Logistic Regression"].plot(label = "Logistic Regression", ax = ax,␣
 ↪legend = True)
plot_df["Prophet"].plot(label = "Prophet", ax = ax, legend = True)
#plt.plot(Logistic_wallet_history, ax = ax)
ax.set_xlabel("Period") # set_xlabel suggested by AI

plt.show()
```


Comparison between models
SPY
MLPRegressor
Logistic Regression
Prophet
115
110
105
100
95
90
85
0
20
40
60
80
100
Period

[ ]: