In [1]:
```python
import pandas as pd

# read dataset
df = pd.read_csv("dataset.csv", parse_dates=["date"])

# format index
df = df.set_index(["ticker", "date"])
```

In [2]:
```python

# if the price increases by more than x%, we label it as "True" or "Buy"
threshold = 0.05  # 5%

# calculate the return within the month
df["return_month"] = (df["adjClose"] / df["adjOpen"]) - 1

# create the target
df["target"] = df["return_month"] >= threshold

df["target"]
```

Out[2]:
```
ticker  date
AAPL    2000-01-31    False
        2000-02-29     True
        2000-03-31     True
        2000-04-30    False
        2000-05-31    False
                      ...
WMT     2020-08-31     True
        2020-09-30    False
        2020-10-31    False
        2020-11-30     True
        2020-12-31    False
Name: target, Length: 7160, dtype: bool
```

In [3]:
```python
# list of features
features = [
    "price_rate_of_change_1M",
    "price_rate_of_change_3M",
    "epsDil",
    "return_on_assets",
    "return_on_equity",
    "price_to_earnings_ratio",
    "debt_to_equity_ratio",
]

# shift the value of the features by one period (make sure to use groupby!)
df[features] = df.groupby("ticker")[features].shift(1)
```

In [4]:
```python
# remove the first row for each ticker to get rid of the NaN created after doing
df = df.loc[df.groupby("ticker").cumcount() > 0]
```

In [5]:
```
split_date = 2020

df_train = df.loc[df.index.get_level_values("date").year < split_date]
df_test = df.loc[df.index.get_level_values("date").year == split_date]
```

In [8]:
```
!pip install lightgbm
```

```
Collecting lightgbm
  Downloading lightgbm-3.3.2-py3-none-win_amd64.whl (1.0 MB)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\users\proma.gupta\ana
conda3\lib\site-packages (from lightgbm) (1.0.2)
Requirement already satisfied: scipy in c:\users\proma.gupta\anaconda3\lib\site
-packages (from lightgbm) (1.7.3)
Requirement already satisfied: wheel in c:\users\proma.gupta\anaconda3\lib\site
-packages (from lightgbm) (0.37.1)
Requirement already satisfied: numpy in c:\users\proma.gupta\anaconda3\lib\site
-packages (from lightgbm) (1.21.5)
Requirement already satisfied: joblib>=0.11 in c:\users\proma.gupta\anaconda3\l
ib\site-packages (from scikit-learn!=0.22.0->lightgbm) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\proma.gupta\ana
conda3\lib\site-packages (from scikit-learn!=0.22.0->lightgbm) (2.2.0)
Installing collected packages: lightgbm
Successfully installed lightgbm-3.3.2
```

In [ ]:

In [ ]:

In [9]:
```
from lightgbm import LGBMClassifier

# define classifier
estimator = LGBMClassifier(
    is_unbalance=True,
    max_depth=4,
    num_leaves=8,
    min_child_samples=400,
    n_estimators=50,
)

# fit classifier on training data
estimator.fit(df_train[features], df_train["target"])
```

Out[9]:
```
LGBMClassifier(is_unbalance=True, max_depth=4, min_child_samples=400,
               n_estimators=50, num_leaves=8)
```

In [10]:
```python
# make prediction using test data
df_test["buy"] = estimator.predict(df_test[features])
```

```
C:\Users\proma.gupta\AppData\Local\Temp\ipykernel_10272\2696836011.py:2: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  df_test["buy"] = estimator.predict(df_test[features])
```

In [11]:
```python
# select only the stocks that were picked by the model
df_buy = df_test.loc[df_test["buy"] == True][["return_month", "target", "buy"]]
```

In [12]:
```python
df_results = (
    df_buy.reset_index()
    .groupby("date")
    .agg({"ticker": "count", "return_month": "mean"})
)
```

In [13]:
```python
df_results.describe()
```

Out[13]:

	ticker	return_month
count	12.000000	12.000000
mean	11.666667	0.029608
std	7.227892	0.088410
min	3.000000	-0.122494
25%	7.750000	-0.039792
50%	9.000000	0.050519
75%	12.750000	0.086988
max	27.000000	0.152646

In [14]:
```python
import numpy as np

def sharpe(s_return: pd.Series, annualize: int, rf: float = 0) -> float:
    """
    Calculate sharpe ratio

    :param s_return: pd.Series with return
    :param annualize: int periods to use for annualization (252 daily, 12 monthly
    :param rf: float risk-free rate
    :return: float sharpe ratio
    """
    # (mean - rf) / std
    sharpe_ratio = (s_return.mean() - rf) / s_return.std()

    # annualize
    sharpe_ratio = sharpe_ratio * np.sqrt(annualize)

    return sharpe_ratio
```

In [15]:
```python
# by using the monthly return, we can calculate the cumulative return over the en
df_results["return_month_cumulative"] = (df_results["return_month"] + 1).cumprod(

df_results
```

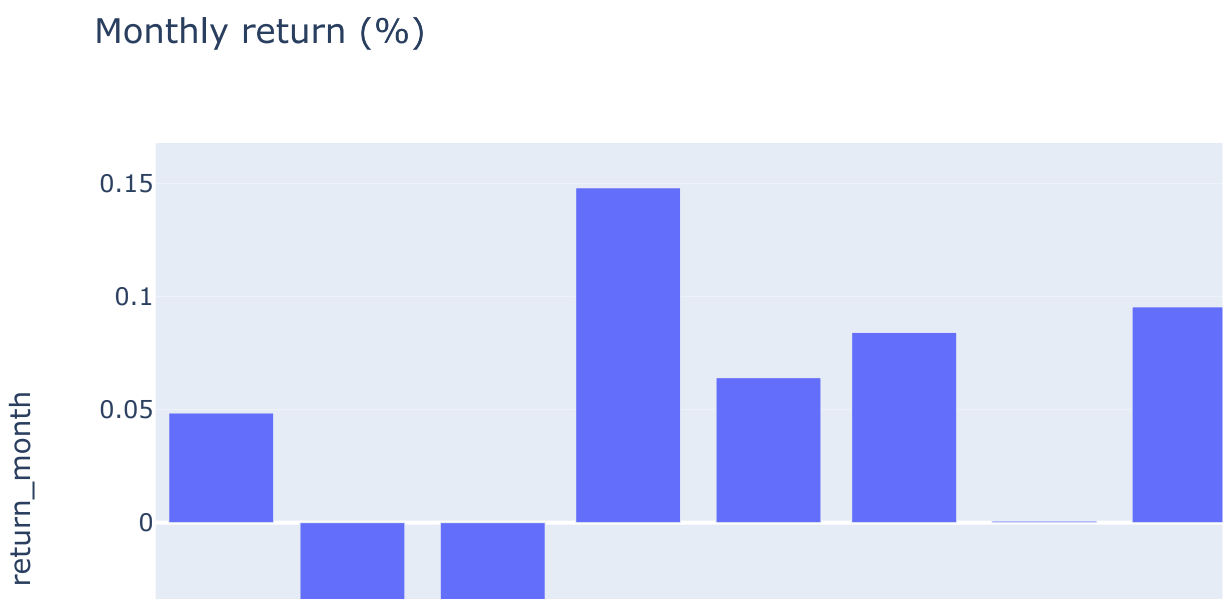
Out[15]:

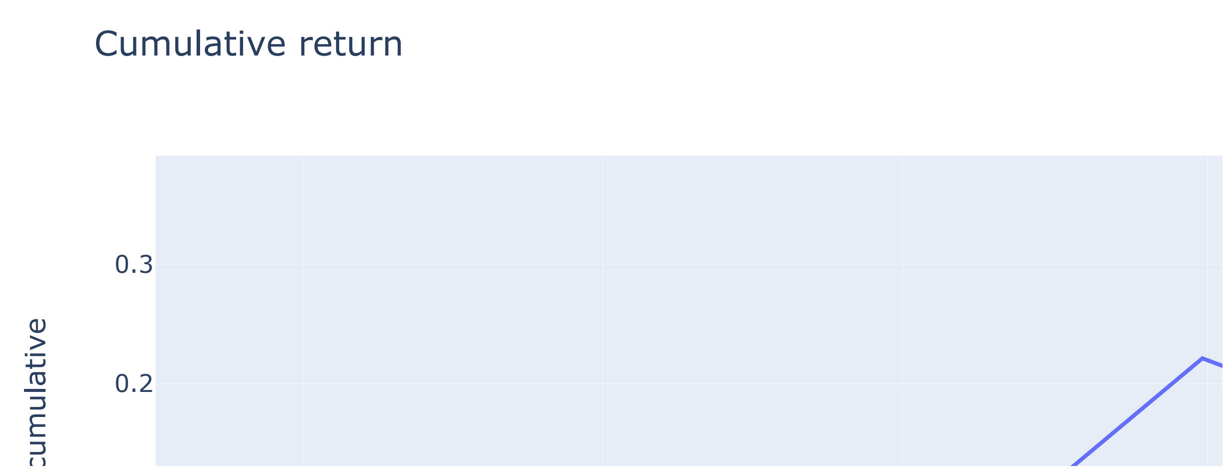
date	ticker	return_month	return_month_cumulative
2020-01-31	3	0.048522	0.048522
2020-02-29	9	-0.085939	-0.041586
2020-03-31	24	-0.122494	-0.158986
2020-04-30	27	0.148161	-0.034381
2020-05-31	12	0.064157	0.027570
2020-06-30	5	0.084158	0.114049
2020-07-31	9	0.000687	0.114814
2020-08-31	9	0.095477	0.221252
2020-09-30	8	-0.038285	0.174496
2020-10-31	15	-0.044311	0.122453
2020-11-30	12	0.152646	0.293791
2020-12-31	7	0.052516	0.361736

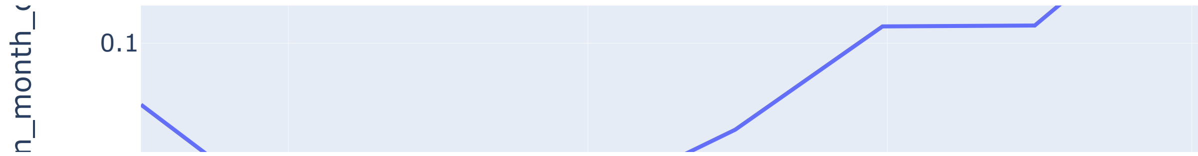
In [16]:
```
import plotly.express as px

# plot monthly return
fig = px.bar(df_results, y="return_month", title="Monthly return (%)")
fig.show()

# plot cumulative return
fig = px.line(df_results, y="return_month_cumulative", title="Cumulative return")
fig.show()
```







In [17]:
```
from sklearn.metrics import classification_report

print(classification_report(df_test["target"], df_test["buy"]))
```

```
              precision    recall  f1-score   support

       False       0.73      0.66      0.69       241
        True       0.42      0.50      0.46       119

    accuracy                           0.61       360
   macro avg       0.57      0.58      0.57       360
weighted avg       0.63      0.61      0.62       360
```

The overall accuracy is 61%. The model does quite a fairly good job at predicting the False class (73% precision, 66% recall) but is less good at predicting the True class (42% precision, 50% recall).

We should be careful when using accuracy in this case, given the high-class imbalance of the dataset.

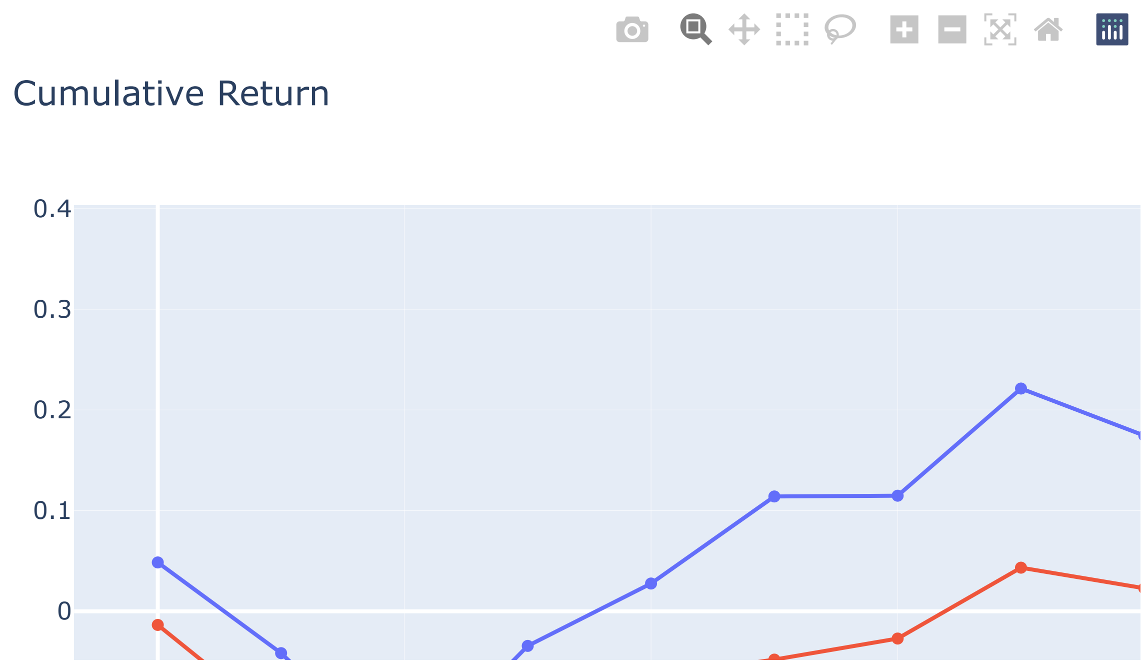
In [18]:
```
# Load the historical price DIA (benchmark strategy)
df_benchmark = pd.read_csv("prices_DIA.csv")
```

In [19]:
```
sharpe_ratio_benchmark = sharpe(df_benchmark["return_month"], annualize=12)
print(f"Sharpe ratio benchmark: {round(sharpe_ratio_benchmark, 2)}")
```

```
Sharpe ratio benchmark: 0.45
```

In [20]:
```python
import plotly.graph_objects as go

fig = go.Figure()
fig = fig.add_trace(
    go.Scatter(y=df_results["return_month_cumulative"], name="ML Model"),
)
fig = fig.add_trace(
    go.Scatter(y=df_benchmark["return_month_cumulative"], name="Benchmark")
)
fig.update_layout(
    title="Cumulative Return",
)
fig.show()
```



The ML model follows the same trajectory as the benchmark strategy, which makes sense given that the set of stocks is limited to only 30 tickers.

However, it does seem that the model was able to distinguish and pick highly performant stocks, leading to a 3x higher return than the benchmark and a boosted Sharpe ratio