Contents

1) Pre-processing data for machine learning python programming for finance p9.

In the previous Project Parts, we learned how to:

- Get financial data.
- Manipulate financial data and
- Visualize financial data.

In the Visualization of financial data, we created my co-relation tables of data.

We can conclude that with pricing data, there are definite relationships between companies.

Sometimes, it is certain companies or a group of companies, but a co-relation definitely exists.

Groups of companies are likely to move together (either up or down).

Although, the companies will not move exactly with each other.

Example: Company A will move earlier. That is, it will be the First Mover.

Company B, unlike Company A, will not move first. It will lag.

Which is why and the question is, can we get a machine to recognize those relationships as we move forward?

While working with financial data, some people would look at just one company at a time period.

They would follow the pricing history of a single company and chart patterns of the specific company.

Sometimes, they use an indicator on just that company to predict weather that specific company will go anywhere in the future.

In this section, we are going to take into account the movement of all other companies as well as the current desired company.

All machine learning does is take "feature-sets" and attempts to map them to "labels." Whether we're doing K Nearest Neighbours or deep learning with neural networks, this remains the same. Thus, we need to:-

Convert our existing data to feature-sets and labels.

Our features can be other company's prices, but we're going to instead say the features are the pricing changes that day for all companies. Our label will be whether or not we actually want to buy a specific company. Let's say we're considering Exxon (XOM). What we'll do for feature-sets is take into account all company percent changes that day, and those will be our features. Our label will be whether or not Exxon (XOM) rose more than x% within the next x days, where we can pick whatever we want for x. To start, let's say a company is a buy if, within the next 7 days, its price goes up more than 2% and it is a sell if the price goes down more than 2% within those 7 days.

This is something we could also relatively easily make a strategy for. If the algorithm says buy, we can buy, place a 2% drop stop-loss (basically something that tells the exchange is price falls below this number / or goes above if you're shorting the company, then exit my position). Otherwise, sell the company once it has risen 2%, or you could be conservative and sell at 1% rise...etc. Regardless, you could relatively easily build a strategy from this classifier. In order to begin, we need the prices into the future for our training data.

CODE:

```
import numpy as np
import pandas as pd
import pickle


def process_data_for_labels(ticker):
    hm_days = 7
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
    tickers = df.columns.values.tolist()
    df.fillna(0, inplace = True)


    for i in range (1, hm_days+1):
        df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-i) -
df[ticker]) / df[ticker]



    df.fillna(0, inplace = True)
    return tickers, df


process_data_for_labels('XOM')
```


python-for-finance-9.1.py - C:/myProject/work/python-for-finance-9.1.py (3.6.8)
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd
import pickle
def process_data_for_labels(ticker):
hm_days = 7
df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
tickers = df.columns.values.tolist()
df.fillna(0, inplace = True)
for i in range (1, hm_days+1):
df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-i) - df[ticker]) / df[ticker]
df.fillna(0, inplace = True)
return tickers, df
process_data_for_labels('XOM')

OUTPUT:


========== RESTART: C:/myProject/work/python-for-finance-9.1.py
>>>

2) Creating machine learning target function python programming for finance p10.

In this section of the Project, we will create a new function.

Creating a new function.

```
def buy_sell_hold()
```

To achieve 33% more accuracy, would be our goal, when we test this function.

This function will traverse by row. We can pass columns of the financial data, as parameters.

We can pull the 'column-row' values and work with the same.

By using this function and `*args`, we will pass the values of the whole week's future percent changes.

CODE:

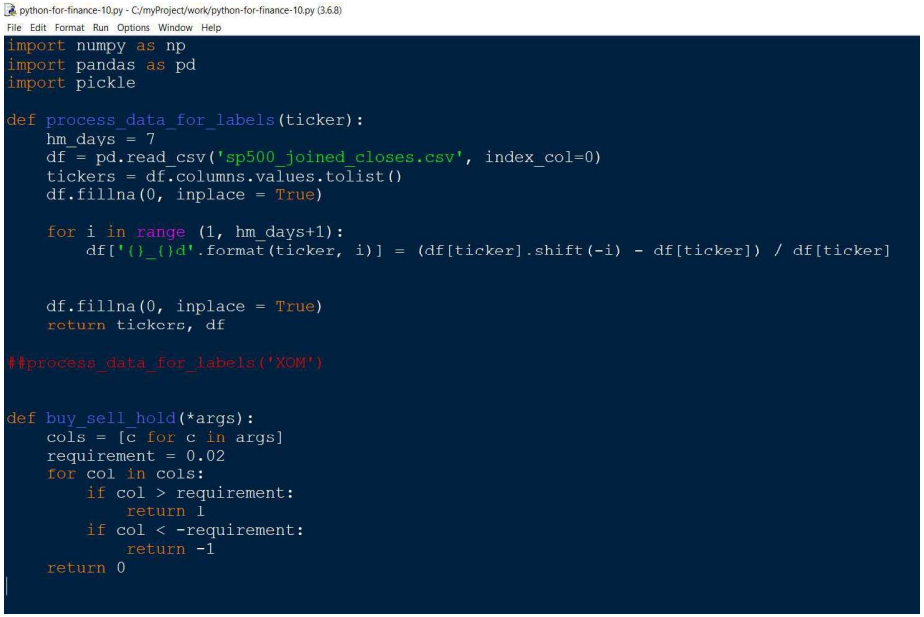
```
import numpy as np
import pandas as pd
import pickle


def process_data_for_labels(ticker):
    hm_days = 7
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
    tickers = df.columns.values.tolist()
    df.fillna(0, inplace = True)


    for i in range (1, hm_days+1):
        df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-i)
- df[ticker]) / df[ticker]


    df.fillna(0, inplace = True)
    return tickers, df


def buy_sell_hold(*args):
    cols = [c for c in args]
    requirement = 0.02
    for col in cols:
        if col > requirement:
            return 1
        if col < -requirement:
            return -1
    return 0
```


python-for-finance-10.py - C:/myProject/work/python-for-finance-10.py (3.6.8)
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd
import pickle
def process_data_for_labels(ticker):
hm_days = 7
df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
tickers = df.columns.values.tolist()
df.fillna(0, inplace = True)
for i in range (1, hm_days+1):
df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-i) - df[ticker]) / df[ticker]
df.fillna(0, inplace = True)
return tickers, df
##process_data_for_labels('XOM')
def buy_sell_hold(*args):
cols = [c for c in args]
requirement = 0.02
for col in cols:
if col > requirement:
return 1
if col < -requirement:
return -1
return 0

3) Creating labels for machine learning python programming for finance p11.

Creating the function that creates our labels.

Now we're going to create the function that creates our label. We have a lot of choices here. You might want to have something that dictates `buy`, `sell`, or `hold`, or maybe just `buy` or `sell`. I am going to have us do the former. Basically, if the price rises more than 2% in the next 7 days, we're going to say that's a buy. If it drops more than 2% in the next 7 days, that's a sell. If it doesn't do either of those, then it's not moving enough, and we're going to just `hold` whatever our position is. If we have shares in that company, we do nothing, we keep our position. If we don't have shares in that company, we do nothing, we just wait. Our function to do this:

```
def buy_sell_hold(*args):
    cols = [c for c in args]
    requirement = 0.02
    for col in cols:
        if col > requirement:
            return 1
        if col < -requirement:
            return -1
    return 0
```

We're using args here so we can take any number of columns here that we want. The idea here is that we're going to map this function to a Pandas DataFrame column, and that column will be our "label." A -1 is a sell, 0 is hold, and 1 is a buy. The `*args` will be those future price change columns, and we're interested if we see movement that exceeds 2% in either direction. Do note, this isn't a totally perfect function. For example, price might go up 2%, then fall 2%, and we might not be prepared for that, but it will do for now.

Let's actually make our features and labels! For this function, we're going to add the following import:

```
from collections import Counter
```

This will let us see the distributions of classes both in our dataset and in our algorithm's predictions. We don't want to feed highly imbalanced datasets to machine learning classifiers, and we also want to see if our classifier is predicting only one class. Our next function:

```
def extract_featuresets(ticker):
    tickers, df = process_data_for_labels(ticker)

    df['{}_target'.format(ticker)] = list(map(
buy_sell_hold,
df['{}_1d'.format(ticker)],
df['{}_2d'.format(ticker)],
df['{}_3d'.format(ticker)],
df['{}_4d'.format(ticker)],
df['{}_5d'.format(ticker)],
df['{}_6d'.format(ticker)],
df['{}_7d'.format(ticker)] ))
```

This function will take any ticker, create the needed dataset, and create our "target" column, which is our `label`. The target column will have either a -1, 0, or 1 for each row, based on our function and the columns we feed through. Now, we can get the distribution:

```
    vals =
df['{}_target'.format(ticker)].values.tolist()
    str_vals = [str(i) for i in vals]
    print('Data spread:',Counter(str_vals))
```

Clean up our data:

```
    df.fillna(0, inplace=True)
    df = df.replace([np.inf, -np.inf], np.nan)
    df.dropna(inplace=True)
```

Again, being careful about infinite numbers, and then filling any other missing data, and, now, finally, we are ready to create our features and labels:

```
    X = df_vals.values
    y = df['{}_target'.format(ticker)].values

    return X,y,df
```

The capital `X` contains our feature-sets (daily % changes for every company in the S&P 500). The lowercase `y` is our "target" or our "label." Basically, what we're trying to map our feature-sets.

CODE:

```
from collections import Counter
import numpy as np
import pandas as pd
import pickle



def process_data_for_labels(ticker):
    hm_days = 7
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
    tickers = df.columns.values.tolist()
    df.fillna(0, inplace = True)


    for i in range (1, hm_days+1):
        df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-
i) - df[ticker]) / df[ticker]



    df.fillna(0, inplace = True)
    return tickers, df
```

```
def buy_sell_hold(*args):
    cols = [c for c in args]
    requirement = 0.02
    for col in cols:
        if col > requirement:
            return 1
        if col < -requirement:
            return -1
    return 0
def extract_featuresets(ticker):
    tickers, df = process_data_for_labels(ticker)
    df['{}_target'.format(ticker)] = list(map(buy_sell_hold,
                                    df['{}_1d'.format(ticker)],
                                    df['{}_2d'.format(ticker)],
                                    df['{}_3d'.format(ticker)],
                                    df['{}_4d'.format(ticker)],
                                    df['{}_5d'.format(ticker)],
                                    df['{}_6d'.format(ticker)],
                                    df['{}_7d'.format(ticker)],
                                                ))
```

```
    vals = df['{}_target'.format(ticker)].values.tolist()
    str_vals = [str(i) for i in vals]
    print('Data spread: ', Counter(str_vals))
    df.fillna(0, inplace = True)

    df = df.replace([np.inf, -np.inf], np.nan)
    df.dropna(inplace = True)

    df_vals = df[[ticker for ticker in
tickers]].pct_change()
    df_vals = df_vals.replace([np.inf, -np.inf], 0)
    df_vals.fillna(0, inplace = True)

    x = df_vals.values
    y = df['{}_target'.format(ticker)].values

    return x, y, df

extract_featuresets('XOM')
```

python-for-finance-11.py - C:/myProject/work/python-for-finance-11.py (3.6.8)

File Edit Format Run Options Window Help

```
from collections import Counter
import numpy as np
import pandas as pd
import pickle


def process_data_for_labels(ticker):
    hm_days = 7
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
    tickers = df.columns.values.tolist()
    df.fillna(0, inplace = True)

    for i in range (1, hm_days+1):
        df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-i) - df[ticker]) / df[ticker]


    df.fillna(0, inplace = True)
    return tickers, df


def buy_sell_hold(*args):
    cols = [c for c in args]
    requirement = 0.02
    for col in cols:
        if col > requirement:
            return 1
        if col < -requirement:
            return -1
    return 0
```

```
def extract_featuresets(ticker):
    tickers, df = process_data_for_labels(ticker)
    df['{}_target'.format(ticker)] = list(map(buy_sell_hold,
                                              df['{}_1d'.format(ticker)],
                                              df['{}_2d'.format(ticker)],
                                              df['{}_3d'.format(ticker)],
                                              df['{}_4d'.format(ticker)],
                                              df['{}_5d'.format(ticker)],
                                              df['{}_6d'.format(ticker)],
                                              df['{}_7d'.format(ticker)],
                                              ))
    vals = df['{}_target'.format(ticker)].values.tolist()
    str_vals = [str(i) for i in vals]
    print('Data spread: ', Counter(str_vals))
    df.fillna(0, inplace = True)

    df = df.replace([np.inf, -np.inf], np.nan)
    df.dropna(inplace = True)

    df_vals = df[[ticker for ticker in tickers]].pct_change()
    df_vals = df_vals.replace([np.inf, -np.inf], 0)
    df_vals.fillna(0, inplace = True)

    x = df_vals.values
    y = df['{}_target'.format(ticker)].values

    return x, y, df

extract_featuresets('XOM')
```

OUTPUT:

```
========== RESTART: C:/myProject/work/python-for-finance-11.py
Data spread:  Counter({'1': 2038, '-1': 1815, '0': 1404})
>>>
>>>
```

4) Machine learning python programming for finance p12.

Importing some required libraries.

```
from sklearn import svm, cross_validation, neighbors


from sklearn.ensemble import VotingClassifier,
RandomForestClassifier
```

```
from collections import Counter
import numpy as np
import pandas as pd
import pickle
from sklearn import svm, cross_validation, neighbors
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
```

Sklearn is a machine learning framework.

The svm import is for a Support Vector Machine, cross_validation will let us easily create shuffled training and testing samples.

Neighbors is for K Nearest Neighbors.

Then, we're bringing in the VotingClassifier and RandomForestClassifier.

The voting classifier is just what it sounds like.

Basically, it's a classifier that will let us combine many classifiers, and allow them to each get a "vote" on what they think the class of the feature-sets is.

The random forest classifier is just another classifier.

We're going to use three classifiers in our voting classifier.

We're ready to do some machine learning now, so let's start our function:

```
def do_ml(ticker):
    X, y, df = extract_featuresets(ticker)
```

We've got our featuresets and labels, now we want to shuffle them up, train, and then test:

```
    X_train, X_test, y_train, y_test =
train_test_split(X,
                 y,
                 test_size=0.25)
```

What this does for us is shuffle our data (so it's not in any specific order any more), and then create training and testing samples for us. We don't want to "test" this algorithm on the same data we trained against. If we did that, chances are we'd do a lot better than we would in reality. We want to test the algorithm on data that it's never seen before to see if we've actually got a model that works.

Now we can choose from any of the classifiers we want, for now, let's do one for K Nearest Neighbors:

```
    clf = neighbors.KNeighborsClassifier()
```

Now we can fit(train) the classifier on our data:

```
    clf.fit(X_train, y_train)
```

This line will take our `X` data, and fit to our `y` data, for each of the pairs of X's and y's that we have. Once that's done, we can test it:

```
    confidence = clf.score(X_test, y_test)
```

This will take some featuresets, `X_test`, make a prediction, and see if it matches our labels, `y_test`. It will return to us the percentage accuracy in decimal form, where 1.0 is 100%, and 0.1 is 10% accurate. Now we can output some further useful information:

```
    print('accuracy:',confidence)
    predictions = clf.predict(X_test)
    print('predicted class counts:',
Counter(predictions))
    print()
    print()
```

This will tell us what the accuracy was, then we can get the precitions of the `X_test`data, and then output the distribution (using `Counter`), so we can see if our model is only classifying one class, which is something that can easily happen.

CODE:

```
import bs4 as bs
import datetime as dt
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np
import os
import pandas as pd
import pandas_datareader.data as web
import pickle
import requests
from collections import Counter
from sklearn import svm, neighbors
from sklearn.ensemble import VotingClassifier,
RandomForestClassifier
from sklearn.model_selection import train_test_split
style.use('ggplot')
def save_sp500_tickers():
    resp =
requests.get('http://en.wikipedia.org/wiki/List_of_S%26P_500
_companies')
    soup = bs.BeautifulSoup(resp.text, 'lxml')
    table = soup.find('table', {'class': 'wikitable
sortable'})
    tickers = []
    for row in table.findAll('tr')[1:]:
        ticker = row.findAll('td')[0].text
        tickers.append(ticker)
    with open("sp500tickers.pickle", "wb") as f:
        pickle.dump(tickers, f)
    return tickers
```

```
def get_data_from_yahoo(reload_sp500=False):
    if reload_sp500:
        tickers = save_sp500_tickers()
    else:
        with open("sp500tickers.pickle", "rb") as f:
            tickers = pickle.load(f)
    if not os.path.exists('stock_dfs'):
        os.makedirs('stock_dfs')


    start = dt.datetime(2010, 1, 1)
    end = dt.datetime.now()
    for ticker in tickers:


        if not os.path.exists
('stock_dfs/{}.csv'.format(ticker)):
            df = web.DataReader(ticker, 'morningstar',
start, end)
            df.reset_index(inplace=True)
            df.set_index("Date", inplace=True)
            df = df.drop("Symbol", axis=1)
            df.to_csv('stock_dfs/{}.csv'.format(ticker))
        else:
            print('Already have {}'.format(ticker))
```

```
def compile_data():
    with open("sp500tickers.pickle", "rb") as f:
        tickers = pickle.load(f)


    main_df = pd.DataFrame()


    for count, ticker in enumerate(tickers):
        df = pd.read_csv('stock_dfs/{}.csv'.format(ticker))
        df.set_index('Date', inplace=True)


        df.rename(columns={'Adj Close': ticker},
inplace=True)
        df.drop(['Open', 'High', 'Low', 'Close', 'Volume'],
1, inplace=True)


        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df, how='outer')


        if count % 10 == 0:
            print(count)
    print(main_df.head())
    main_df.to_csv('sp500_joined_closes.csv')
```

```
def visualize_data():
    df = pd.read_csv('sp500_joined_closes.csv')
    df_corr = df.corr()
    print(df_corr.head())
    df_corr.to_csv('sp500corr.csv')
    data1 = df_corr.values
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111)


    heatmap1 = ax1.pcolor(data1, cmap=plt.cm.RdYlGn)
    fig1.colorbar(heatmap1)


    ax1.set_xticks(np.arange(data1.shape[1]) + 0.5,
minor=False)
    ax1.set_yticks(np.arange(data1.shape[0]) + 0.5,
minor=False)
    ax1.invert_yaxis()
    ax1.xaxis.tick_top()
    column_labels = df_corr.columns
    row_labels = df_corr.index
    ax1.set_xticklabels(column_labels)
    ax1.set_yticklabels(row_labels)
    plt.xticks(rotation=90)
    heatmap1.set_clim(-1, 1)
    plt.tight_layout()
    plt.show()
```

```
def process_data_for_labels(ticker):
    hm_days = 7
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
    tickers = df.columns.values.tolist()
    df.fillna(0, inplace=True)
    for i in range(1, hm_days+1):
        df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-
i) - df[ticker]) / df[ticker]
    df.fillna(0, inplace=True)
    return tickers, df



def buy_sell_hold(*args):
    cols = [c for c in args]
    requirement = 0.02
    for col in cols:
        if col > requirement:
            return 1
        if col < -requirement:
            return -1
    return 0
```

```
def extract_featuresets(ticker):
    tickers, df = process_data_for_labels(ticker)
    df['{}_target'.format(ticker)] = list(map(
buy_sell_hold,
df['{}_1d'.format(ticker)],
```

```
def do_ml(ticker):
    X, y, df = extract_featuresets(ticker)


    X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.25)


    clf = neighbors.KNeighborsClassifier()


    clf.fit(X_train, y_train)
    confidence = clf.score(X_test, y_test)
    print('accuracy:', confidence)
    predictions = clf.predict(X_test)
    print('predicted class counts:', Counter(predictions))


    return confidence


do_ml('BAC')
```

python-for-finance-12.py - C:/myProject/work/python-for-finance-12.py (3.6.8)

File Edit Format Run Options Window Help

```
import bs4 as bs
import datetime as dt
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np
import os
import pandas as pd
import pandas_datareader.data as web
import pickle
import requests
from collections import Counter
from sklearn import svm, neighbors
from sklearn.ensemble import VotingClassifier, RandomForestClassifier


from sklearn.model_selection import train_test_split

style.use('ggplot')


def save_sp500_tickers():
    resp = requests.get('http://en.wikipedia.org/wiki/List_of_S%26P_500_companies')
    soup = bs.BeautifulSoup(resp.text, 'lxml')
    table = soup.find('table', {'class': 'wikitable sortable'})
    tickers = []
    for row in table.findAll('tr')[1:]:
        ticker = row.findAll('td')[0].text
        tickers.append(ticker)
    with open("sp500tickers.pickle", "wb") as f:
        pickle.dump(tickers, f)
    return tickers
```

```
def get_data_from_yahoo(reload_sp500=False):
    if reload_sp500:
        tickers = save_sp500_tickers()
    else:
        with open("sp500tickers.pickle", "rb") as f:
            tickers = pickle.load(f)
    if not os.path.exists('stock_dfs'):
        os.makedirs('stock_dfs')

    start = dt.datetime(2010, 1, 1)
    end = dt.datetime.now()
    for ticker in tickers:

        if not os.path.exists('stock_dfs/{}.csv'.format(ticker)):
            df = web.DataReader(ticker, 'morningstar', start, end)
            df.reset_index(inplace=True)
            df.set_index("Date", inplace=True)
            df = df.drop("Symbol", axis=1)
            df.to_csv('stock_dfs/{}.csv'.format(ticker))
        else:
            print('Already have {}'.format(ticker))


def compile_data():
    with open("sp500tickers.pickle", "rb") as f:
        tickers = pickle.load(f)

    main_df = pd.DataFrame()

    for count, ticker in enumerate(tickers):
        df = pd.read_csv('stock_dfs/{}.csv'.format(ticker))
```

```
        df.set_index('Date', inplace=True)

        df.rename(columns={'Adj Close': ticker}, inplace=True)
        df.drop(['Open', 'High', 'Low', 'Close', 'Volume'], 1, inplace=True)

        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df, how='outer')

        if count % 10 == 0:
            print(count)
    print(main_df.head())
    main_df.to_csv('sp500_joined_closes.csv')


def visualize_data():
    df = pd.read_csv('sp500_joined_closes.csv')
    df_corr = df.corr()
    print(df_corr.head())
    df_corr.to_csv('sp500corr.csv')
    data1 = df_corr.values
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111)

    heatmap1 = ax1.pcolor(data1, cmap=plt.cm.RdYlGn)
    fig1.colorbar(heatmap1)

    ax1.set_xticks(np.arange(data1.shape[1]) + 0.5, minor=False)
    ax1.set_yticks(np.arange(data1.shape[0]) + 0.5, minor=False)
    ax1.invert_yaxis()
```

```
    ax1.invert_yaxis()
    ax1.xaxis.tick_top()
    column_labels = df_corr.columns
    row_labels = df_corr.index
    ax1.set_xticklabels(column_labels)
    ax1.set_yticklabels(row_labels)
    plt.xticks(rotation=90)
    heatmap1.set_clim(-1, 1)
    plt.tight_layout()
    plt.show()


def process_data_for_labels(ticker):
    hm_days = 7
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)
    tickers = df.columns.values.tolist()
    df.fillna(0, inplace=True)
    for i in range(1, hm_days+1):
        df['{}_{}d'.format(ticker, i)] = (df[ticker].shift(-i) - df[ticker]) / df[ticker]
    df.fillna(0, inplace=True)
    return tickers, df


def buy_sell_hold(*args):
    cols = [c for c in args]
    requirement = 0.02
    for col in cols:
        if col > requirement:
            return 1
        if col < -requirement:
            return -1
```

```
def extract_featuresets(ticker):
    tickers, df = process_data_for_labels(ticker)

    df['{}_target'.format(ticker)] = list(map( buy_sell_hold,
                                               df['{}_1d'.format(ticker)],
                                               df['{}_2d'.format(ticker)],
                                               df['{}_3d'.format(ticker)],
                                               df['{}_4d'.format(ticker)],
                                               df['{}_5d'.format(ticker)],
                                               df['{}_6d'.format(ticker)],
                                               df['{}_7d'.format(ticker)]))

    vals = df['{}_target'.format(ticker)].values.tolist()
    str_vals = [str(i) for i in vals]
    print('Data spread:', Counter(str_vals))

    df.fillna(0, inplace=True)
    df = df.replace([np.inf, -np.inf], np.nan)
    df.dropna(inplace=True)

    df_vals = df[[ticker for ticker in tickers]].pct_change()
    df_vals = df_vals.replace([np.inf, -np.inf], 0)
    df_vals.fillna(0, inplace=True)

    X = df_vals.values
    y = df['{}_target'.format(ticker)].values
    return X, y, df
```

```
def do_ml(ticker):
    X, y, df = extract_featuresets(ticker)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

    clf = neighbors.KNeighborsClassifier()

    clf.fit(X_train, y_train)
    confidence = clf.score(X_test, y_test)
    print('accuracy:', confidence)
    predictions = clf.predict(X_test)
    print('predicted class counts:', Counter(predictions))

    return confidence

do_ml('BAC')
```

OUTPUT:

```
========== RESTART: C:/myProject/work/python-for-finance-12.py
Data spread: Counter({'1': 2243, '-1': 2034, '0': 980})
accuracy: 0.3307926829268293
predicted class counts: Counter({0: 609, -1: 481, 1: 222})
>>>
```

What is our goal? Well, something that picks randomly should be about 33% accurate, since we have three total choices in theory, but actually it isn't likely that our model will be truly balanced. Let's see some examples, and just run:

```
do_ml('XOM')
do_ml('AAPL')
do_ml('ABT')
```

1) For Exxon Mobil Corporation.

Exxon Mobil Corporation
NYSE: XOM

35.23 USD −1.24 (3.40%)
Closed: 12 Nov, 7:59 pm GMT-5 · Disclaimer
After hours 35.02 −0.21 (0.60%)

```
do_ml('XOM')
```

OUTPUT:

```
========== RESTART: C:/myProject/work/python-for-finance-12.py
Data spread: Counter({'1': 2038, '-1': 1815, '0': 1404})
accuracy: 0.31402439024390244
predicted class counts: Counter({0: 725, -1: 448, 1: 139})
>>>
```

2) For Apple INC.


```
do_ml('AAPL')
```

OUTPUT:

```
========== RESTART: C:/myProject/work/python-for-finance-12.py
Data spread: Counter({'1': 2622, '-1': 2132, '0': 503})
accuracy: 0.39939024390243905
predicted class counts: Counter({1: 663, -1: 526, 0: 123})
>>>
```

3) For Abbott Laboratories.

Abbott Laboratories
NYSE: ABT

111.95 USD −0.69 (0.61%)

Closed: 12 Nov, 6:21 pm GMT-5 · Disclaimer
After hours 111.07 −0.88 (0.79%)

```
do_ml('ABT')
```

References