Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = ""
COLLABORATORS = ""
```

Assignment: Using Machine Learning for Hedging

Welcome to the first assignment !

Problem description

We will solve a Regression task that is very common in Finance

- Given the return of "the market", predict the return of a particular stock

That is

- Given the return of a proxy for "the market" at time t, predict the return of, e.g., Apple at time t.

As we will explain, being able to predict the relationship between two financial instruments opens up possibilities

- Use one instrument to "hedge" or reduce the risk of holding the other
- Create strategies whose returns are independent of "the market"
    - Hopefully make a profit regardless of whether the market goes up or down

Goal

You will create models of increasing complexity in order to explain the return of Apple (ticker AAPL)

- The first model will have a single feature: return of the market proxy, ticker SPY
- Subsequent models will add the return of other tickers as additional features

Learning Objectives

- Learn how to solve a Regression task
- Become facile in the `sklearn` toolkit for Machine Learning

How to report your answers

We will mix explanation of the topic with tasks that you must complete.

Look for the string "**Question**" to find a task that you must perform.

Most of the tasks will require you to create some code at the location indicated by

```
# YOUR CODE HERE
raise NotImplementedError()
```

- Replace `raise NotImplementedError()` with your own code

Standard imports

In [2]:
```
# Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn

import os
import math

%matplotlib inline
```

In [3]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Reload all modules imported with %aimport
%load_ext autoreload
%autoreload 1

# Import nn_helper module
import helper
%aimport helper

helper = helper.HELPER()
```

Get The data

The first step in our Recipe is Get the Data.

The data are the daily prices of a number of individual equities and equity indices.

The prices are arranged in a series in ascending date order (a timeseries).

- There are many `.csv` files for equity or index in the directory `DATA_DIR`

API for students

We will define some utility routines to help you.

In this way, you can focus on the learning objectives rather than data manipulation.

This is not representative of the "real world"; you will need to complete data manipulation tasks in later assignments.

We provide a class `HELPER`

- Instantiated as

```
helper = helper.HELPER()
```

With methods

- `getData`:
    - Get examples for a list of equity tickers and an index ticker.
    - Called as

```
data = helper.getData( tickers,
index_ticker, attrs)
```

- *`tickers` is a list of tickers*
- *`index` is the ticker of the index*
- *`attrs` is a list of data attributes*

Question:

- Create code to
    - Get the adjusted close price of AAPL and SPY
    - Assign the result to variable `data`

Hint:

- Use the `getData` method from the helper class

In [4]:
```python
ticker = "AAPL"
index_ticker = "SPY"
dateAttr = "Dt"
priceAttr = "Adj Close"

# YOUR CODE HERE
raise NotImplementedError()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-4-228edb97c2dd> in <module>
      5 
      6 # YOUR CODE HERE
----> 7 raise NotImplementedError()

NotImplementedError: 
```

Have a look at the data

We will not go through all steps in the Recipe, nor in depth.

But here's a peek at the data you retrieved

In [ ]:
```python
data.head()
```

In [ ]:
```
# Print the Start time and End time
print("Start time: ", data.index.min())
print("End time: ", data.index.max())
```

Create DataFrame of price levels for the training examples

The training examples will be stored in a DataFrame.

- The DataFrame should have two columns: the price level for the ticker and for the index
- The minimum date in the DataFrame should be **the trading day before** `start_dt`
  - That is: the latest date for which there is data and which is less than `start_dt`
  - For example, if `start_dt` is a Monday, the "day before" would be Friday, not Sunday.
    - Similarly for the case where the day before `start_dt` is a holiday
- The maximum date in the DataFrame should be `end_dt`

The reason we are adding one day prior to `start_dt`

- We want to have returns (percent price changes) from `start_dt` onwards
- In order to compute a return for `start_dt`, we need the level from the prior day

Question:

- Complete the function `getRange()`
    - To return the subset of rows of our examples
    - Beginning on the **trading day before** date `start_dt`
    - Ending on date `end_dt`

In [ ]:

```python
start_dt = "2018-01-02"
end_dt = "2018-09-28"
train_data_price = None

# Set variable train_data_price to be a DataFrame with two columns
## AAPL_Adj_Close, SPY_Adj_Close
## with dates as the index
## Having minimum date equal to THE DAY BEFORE start_dt
## Having maximum date equal to end_dt

def getRange(df, start_dt, end_dt):
    '''
    Return the the subset of rows of DataFrame df
    restricted to dates between start_dt and end_dt

    Parameters
    ----------
    df: DataFrame
    - The data from which we will take a subset

    start_dt: String
    - Start date

    end_dt: String
    - End date
    '''
    # YOUR CODE HERE
    raise NotImplementedError()

train_data_price = getRange(data, start_dt, end_dt)
print(train_data_price.head())
```

In [ ]:

As you can see, each row has two attributes for one date

- Price (adjusted close) of ticker AAPL
- Price (adjusted close) of the market proxy SPY

Create test set

We just created a set of training examples as a subset of the rows of `data` .

We will do the same to create a set of test examples.

Question:

Set variable `test_data_price`

- To the subset of rows of our examples
- Beginning on the **trading day before** date `test_start_dt`
- Ending on date `test_end_dt`

Hint

- Use `getRange` with different arguments for the dates

In [ ]:
```
test_start_dt = '2018-10-01'
test_end_dt = '2018-12-31'

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```

```

Prepare the data

In Finance, it is very typical to work with *relative changes* (e.g., percent price change) rather than *absolute changes* (price change) or *levels* (prices).

Without going into too much detail

- Relative changes are more consistent over time than either absolute changes or levels
- The consistency can facilitate the use of data over a longer time period

For example, let's suppose that prices are given in units of USD (dollar)

- A price change of 1 USD is more likely for a stock with price level 100 than price level 10
    - A relative change of $1/100 = 1$ is more likely than a change of $1/10 = 10$
    - So relative changes are less dependent on price level than either price changes or price levels

To compute the *return* (percent change in prices) for ticker AAPL (Apple) on date t

$$r^{(t)}_\mathrm{AAPL} = \frac{p^{(t)}_\mathrm{AAPL}}{p^{(t-1)}_\mathrm{AAPL}} - 1$$

Transformations: transform the training data

Our first task is to transform the data from price levels (Adj Close) to Percent Price Changes.

Moreover, the date range for the training data is specified to be in the range from `start_dt` (start date) to `end_dt`, inclusive on both sides.

Note

We will need to apply **identical** transformations to both the training and test data examples.

In the cells that immediately follow, we will do this only for the **training data**

You will need to repeat these steps for the test data in a subsequent step.

You are well-advised to create subroutines or functions to accomplish these tasks !

- You will apply them first to transform training data
- You will apply them a second time to transform the test data

We will achieve this is several steps

Create DataFrame of returns for training examples

Create a new DataFrame with percent price changes of the columns, rather than the levels

Question:

- Complete function `getReturns()` to set variable `train_data_ret` to be a DataFrame with the same columns
    - But where the prices have been replaced by day over day percent changes
    - The column names of `train_data_ret` should be the same as the original columns names
    - We give you code to rename the columns to reflect the changed meaning of the data in the next step

Hint:

- look up the Pandas `pct_change()` method

In [ ]:
```
train_data_ret = None

def getReturns(df):
    '''
    Return the day over day percent changes of adjusted price

    Parameters
    ----------
    df: DataFrame
    '''

    # YOUR CODE HERE
    raise NotImplementedError()

train_data_ret = getReturns(train_data_price)
train_data_ret.head()
```

In [ ]:
```

```

Since the columns of `train_data_ret` are now returns, we will rename then for you.

Also, we will drop the earliest date

- There is now return for this date
- We included this row only so we could compute the return for the following

In [ ]:
```
## Rename the columns to indicate that they have been transformed from price (Adj_close) to Return
train_data_ret = helper.renamePriceToRet( train_data_ret )

## Drop the first date (the day before `start_dt`) since it has an undefined return
train_data_ret = train_data_ret[ start_dt:]
train_data_ret.head()
```

Remove the target

The only feature is the return of the market proxy SPY.

Predicting the target given the target as a feature would be cheating !

So we will create `X_train, y_train` from `train_data_ret`

- `X_train` has only features for the example
- `y_train` is the target for the example

In [ ]:
```
tickerAttr = ticker + "_Ret"

X_train, y_train =  train_data_ret.drop(columns=[tickerAttr]), train_data_ret[[tickerAttr ]]
```

Transformations: transform the test data

We have just performed some transformations of the training data.

Remember:

You need to perform *identical* transformations to the test data.

The test data will be returns from `test_start_dt` to `test_end_dt` inclusive.

We will apply identical transformations as we did to the training data, but with a different date range.

We obtained `X_train, y_train` via transformations to `train_data_price.`

We will now obtain `X_test, y_test` by identical transformations to `test_data_price`

Question:

Create the training data `X_test, y_test`

- Apply the same transformations to `test_data_price` as you did to `train_data_price`
- To create variable `test_data_ret`
- We will convert `test_data_ret` to `X_test, y_test` for you

Hints

Create `test_data_ret` in a manner analogous to the creation of `train_data_ret`

- Use `getReturns` to convert price levels to returns
- Use `helper.renamePriceToRet` to rename the columns to reflect the change

In [ ]:
```
test_data_price.head()
```

In [ ]:
```
test_data_ret = None
X_test = None
y_test = None

# YOUR CODE HERE
raise NotImplementedError()

X_test, y_test =  test_data_ret.drop(columns=[tickerAttr]), test_data_ret[[ tick
erAttr ]]

print("test data length", test_data_ret.shape[0])
print("X test length", X_test.shape[0])
print("y test length", y_test.shape[0])
test_data_ret.head()
```

```
In [ ]:
```

Train a model (Regression)

Use Linear Regression to predict the return of a ticker from the return of the market proxy SPY. For example, for ticker AAPL

$$r_\text{AAPL}^{(t)} = \beta_0 + \beta_{\text{AAPL},\text{SPY}} * r_\text{SPY}^{(t)} + \epsilon_\text{AAPL}^{(t)}$$

Each example corresponds to one day (time t)

- has features
  - constant 1, corresponding to the intercept parameter
  - return of the market proxy SPY

$$\mathbf{x}^{(t)} = \begin{pmatrix} 1 \\ r_\text{SPY}^{(t)} \end{pmatrix}$$

- has target
  - return of the ticker

$$\mathbf{y}^{(t)} = r_\text{AAPL}^{(t)}$$

You will use Linear Regression to solve for parameters β_0, $\beta_{\text{AAPL},\text{SPY}}$

- In the lectures we used the symbol Θ to denote the parameter vector; here we use β
- In Finance the symbol β is often used to denote the relationship between returns.
- Rather than explicitly creating a constant 1 feature
  - you may invoke the model object with the option including an intercept
  - if you do so, the feature vector you pass will be

$$\mathbf{x}^{(t)} = \left(r_{\text{SPY}}^{(t)} \right)$$

- Use the entire training set
- Do not use cross-validation

Question:

Train your model to estimate the parameters beta_0 and beta_SPY

- Complete the function `createModel()` to build your linear regression model. The detailed description is in the function below.
- Complete the function `regress()` to perform the regression and return two item: the intercept and coefficients. The detailed description is in the function below.
    - `beta_0` is the regression parameter for the constant;
    - `beta_SPY` is the regression parameter for the return of SPY.
    - We will test if the parameters of your regression are correct. We have initialized them to be 0.

Hints:

In [ ]:
```
from sklearn import datasets, linear_model

beta_0 = 0    # The regression parameter for the constant
beta_SPY = 0  # The regression parameter for the return of SPY
ticker = "AAPL"

def createModel():
    '''
    Build your linear regression model using sklearn

    Returns
    -------
    An sklearn model object implementing Linear Regression
    '''
    # YOUR CODE HERE
    raise NotImplementedError()

def regress(model, X, y):
    '''
    Do regression using returns of your ticker and index

    Parameters
    -----------
    model: model object implementing Linear Regression

    X: DataFrame
    - Index returns

    y: DataFrame
    - Ticker returns

    Returns
    -------
    Tuple (beta_0, beta_SPY)
    where,
```

```
        beta_0: Scalar number
        - Parameter for the constant

        beta_SPY: Scalar number
        - Parameter for the return of SPY


    '''
    # YOUR CODE HERE
    raise NotImplementedError()

# Assign to answer variables
regr = createModel()

beta_0, beta_SPY = regress(regr, X_train, y_train)


print("{t:s}: beta_0={b0:3.3f}, beta_SPY={b1:3.3f}".format(t=ticker, b0=beta_0,
b1=beta_SPY))
```

Your expected outputs should be:

beta_0	0.001
beta_SPY	1.071

In [ ]:

```

```

Train the model using Cross validation

Since we only have one test set, we want to use 5-fold cross validation to assess model performance.

Question:

- Complete the function `compute_cross_val_avg()` to compute the average score of 5-fold cross validation
    - Set `cross_val_avg` as your average score of k-fold results
    - Set `k = 5` as the number of folds

Hint:

- You can use the `cross_val_score` in `sklearn.model_selection`

In [ ]:
```
from sklearn.model_selection import cross_val_score

cross_val_avg = 0 # average score of cross validation
k = 5             # 5-fold cross validation

def compute_cross_val_avg(model, X, y, k):
    '''
    Compute the average score of k-fold cross validation

    Parameters
    -----------
    model: An sklearn model

    X: DataFrame
    - Index returns

    y: DataFrame
    - Ticker returns

    k: Scalar number
    - k-fold cross validation

    Returns
    --------
    The average, across the k iterations, of the score
    '''
    # YOUR CODE HERE
    raise NotImplementedError()


cross_val_avg = compute_cross_val_avg(regr, X_train, y_train, 5)
print("{t:s}: Avg cross val score = {sc:3.2f}".format(t=ticker, sc=cross_val_av
g) )
```

In [ ]:
```

```

Evaluate Loss (in sample RMSE) and Performance (Out of sample RMSE)

To see how well your model performs, we can check the in-sample loss and out-of-sample performance.

Question:

- Complete the function `computeRMSE()` to compute the Root of Mean Square Error (RMSE)
    - Set `rmse_in_sample` to be in-sample loss
    - Set `rmse_out_sample` to be out-of-sample performance