智慧金融量化分析
(Artificial Intelligence in Finance and Quantitative Analysis)

人工智慧優先金融
(AI-First Finance)

1101AIFQA08
MBA, IM, NTPU (M6132) (Fall 2021)
Tue 2, 3, 4 (9:10-12:00) (8F40)

戴敏育 副教授
Min-Yuh Day, Ph.D, Associate Professor
國立臺北大學 資訊管理研究所
Institute of Information Management, National Taipei University
https://web.ntpu.edu.tw/~myday

2021-11-30

課程大綱 (Syllabus)

國立臺北大學
National Taipei University

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
1	2021/09/28	智慧金融量化分析概論 (Introduction to Artificial Intelligence in Finance and Quantitative Analysis)
2	2021/10/05	AI 金融科技: 金融服務創新應用 (AI in FinTech: Financial Services Innovation and Application)
3	2021/10/12	投資心理學與行為財務學 (Investing Psychology and Behavioral Finance)
4	2021/10/19	財務金融事件研究法 (Event Studies in Finance)
5	2021/10/26	智慧金融量化分析個案研究 I (Case Study on AI in Finance and Quantitative Analysis I)
6	2021/11/02	財務金融理論 (Finance Theory)

課程大綱 (Syllabus)

國立臺北大學 National Taipei University

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
7	2021/11/09	數據驅動財務金融 (Data-Driven Finance)
8	2021/11/16	期中報告 (Midterm Project Report)
9	2021/11/23	金融計量經濟學 (Financial Econometrics)
10	2021/11/30	人工智慧優先金融 (AI-First Finance)
11	2021/12/07	智慧金融量化分析產業實務 (Industry Practices of AI in Finance and Quantitative Analysis)
12	2021/12/14	智慧金融量化分析個案研究 II (Case Study on AI in Finance and Quantitative Analysis II)

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
13	2021/12/21	財務金融深度學習(Deep Learning in Finance); 財務金融強化學習 (Reinforcement Learning in Finance)
14	2021/12/28	演算法交易 (Algorithmic Trading); 風險管理 (Risk Management); 交易機器人與基於事件的回測 (Trading Bot and Event-Based Backtesting)
15	2022/01/04	期末報告 I (Final Project Report I)
16	2022/01/11	期末報告 II (Final Project Report II)
17	2022/01/18	學生自主學習 (Self-learning)
18	2022/01/25	學生自主學習 (Self-learning)

AI-First Finance

AI-First Finance

- Efficient Markets
- Market Prediction Based on Returns Data
- Market Prediction with More Features
- Market Prediction Intraday

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Life 3.0:
Being human in the age of artificial intelligence
Max Tegmark (2017)

A computation takes information and transforms it, implementing what mathematicians call a function....

If you're in possession of a function that inputs all the world's financial data and outputs the best stocks to buy, you'll soon be extremely rich.

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Efficient Markets

- Efficient Market Hypothesis (EMH)
  - Random Walk Hypothesis (RWH)
- Weak form of EMH
  - The information set θ_t only encompasses the past price and return history of the market.
- Semi-strong form of EMH
  - The information set θ_t is taken to be all publicly available information, including not only the past price and return history but also financial reports, news articles, weather data, and so on.
- Strong form of EMH
  - The information set θ_t includes all information available to anyone (that is, even private information).

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
import numpy as np
import pandas as pd
from pylab import plt, mpl
plt.style.use('seaborn')
mpl.rcParams['savefig.dpi'] = 300
mpl.rcParams['font.family'] = 'serif'
pd.set_option('precision', 4)
np.set_printoptions(suppress=True, precision=4)

url = 'http://hilpisch.com/aiif_eikon_eod_data.csv'

data = pd.read_csv(url, index_col=0, parse_dates=True).dropna()

(data / data.iloc[0]).plot(figsize=(10, 6), cmap='coolwarm')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Normalized time series data (end-of-day)

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
lags = 7

def add_lags(data, ric, lags):
    cols = []
    df = pd.DataFrame(data[ric])
    for lag in range(1, lags + 1):
        col = 'lag_{}'.format(lag)
        df[col] = df[ric].shift(lag)
        cols.append(col)
    df.dropna(inplace=True)
    return df, cols

dfs = {}
for sym in data.columns:
    df, cols = add_lags(data, sym, lags)
    dfs[sym] = df
dfs[sym].head(7)
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

lagged prices

Date	GLD	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7
2010-01-13	111.54	110.49	112.85	111.37	110.82	111.51	109.70	109.80
2010-01-14	112.03	111.54	110.49	112.85	111.37	110.82	111.51	109.70
2010-01-15	110.86	112.03	111.54	110.49	112.85	111.37	110.82	111.51
2010-01-19	111.52	110.86	112.03	111.54	110.49	112.85	111.37	110.82
2010-01-20	108.94	111.52	110.86	112.03	111.54	110.49	112.85	111.37
2010-01-21	107.37	108.94	111.52	110.86	112.03	111.54	110.49	112.85
2010-01-22	107.17	107.37	108.94	111.52	110.86	112.03	111.54	110.49

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
regs = {}
for sym in data.columns:
df = dfs[sym]
reg = np.linalg.lstsq(df[cols], df[sym], rcond=-1)[0]
#Return the least-squares solution to a linear matrix equation
regs[sym] = reg

rega = np.stack(tuple(regs.values()))
regd = pd.DataFrame(rega, columns=cols, index=data.columns)
regd
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

regression analysis

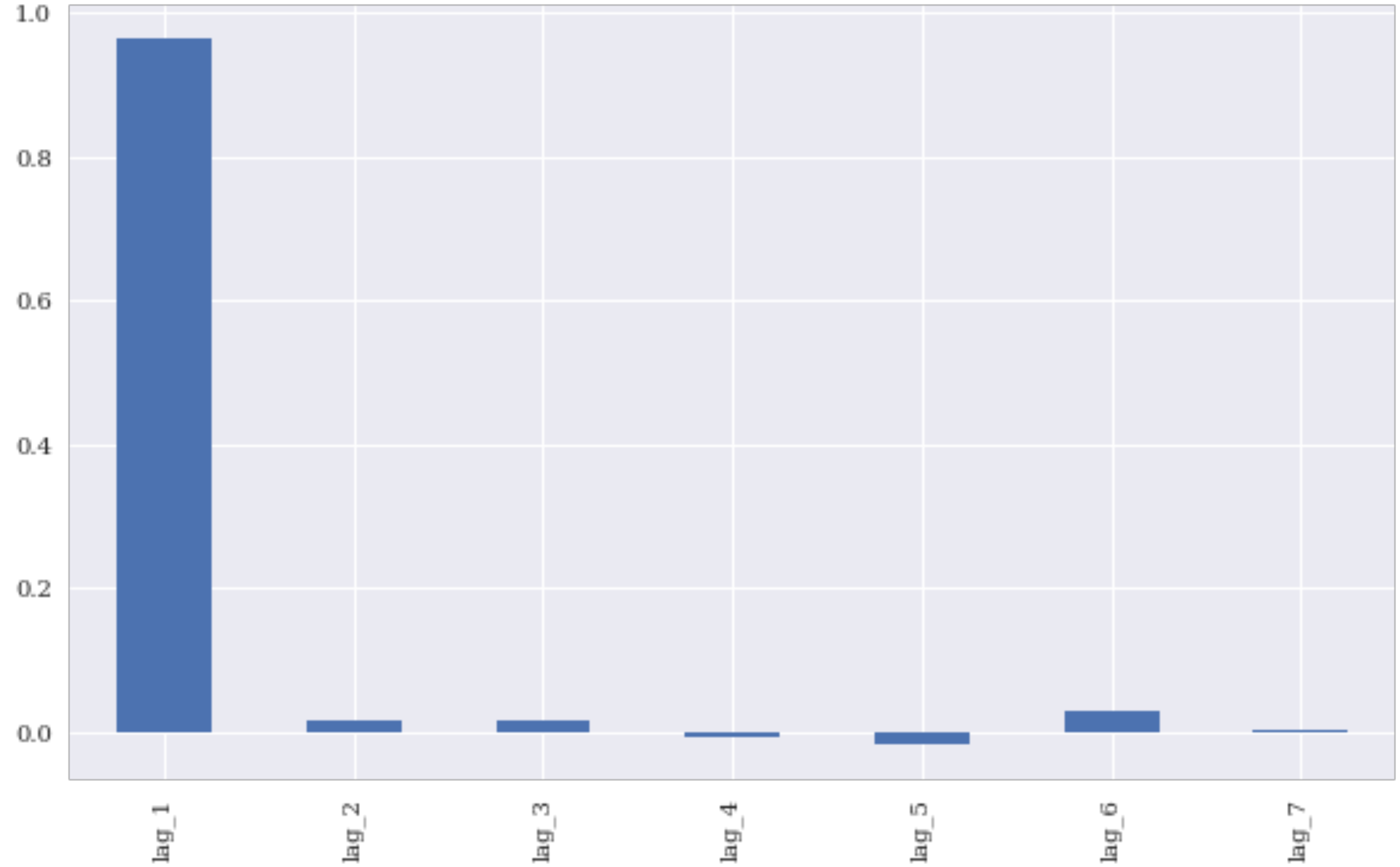
```
reg = np.linalg.lstsq(df[cols], df[sym], rcond=-1)[0]
```

	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7
AAPL.O	1.0106	-0.0592	0.0258	0.0535	-0.0172	0.0060	-0.0184
MSFT.O	0.8928	0.0112	0.1175	-0.0832	-0.0258	0.0567	0.0323
INTC.O	0.9519	0.0579	0.0490	-0.0772	-0.0373	0.0449	0.0112
AMZN.O	0.9799	-0.0134	0.0206	0.0007	0.0525	-0.0452	0.0056
GS.N	0.9806	0.0342	-0.0172	0.0042	-0.0387	0.0585	-0.0215
SPY	0.9692	0.0067	0.0228	-0.0244	-0.0237	0.0379	0.0121
.SPX	0.9672	0.0106	0.0219	-0.0252	-0.0318	0.0515	0.0063
.VIX	0.8823	0.0591	-0.0289	0.0284	-0.0256	0.0511	0.0306
EUR=	0.9859	0.0239	-0.0484	0.0508	-0.0217	0.0149	-0.0055
XAU=	0.9864	0.0069	0.0166	-0.0215	0.0044	0.0198	-0.0125
GDX	0.9765	0.0096	-0.0039	0.0223	-0.0364	0.0379	-0.0065
GLD	0.9766	0.0246	0.0060	-0.0142	-0.0047	0.0223	-0.0106

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Average optimal regression parameters for the lagged prices

```
regd.mean().plot(kind='bar', figsize=(10, 6))
```



Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Correlations between the lagged time series

```
dfs[sym].corr()
```

	GLD	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7
GLD	1.0000	0.9972	0.9946	0.9920	0.9893	0.9867	0.9841	0.9815
lag_1	0.9972	1.0000	0.9972	0.9946	0.9920	0.9893	0.9867	0.9842
lag_2	0.9946	0.9972	1.0000	0.9972	0.9946	0.9920	0.9893	0.9867
lag_3	0.9920	0.9946	0.9972	1.0000	0.9972	0.9946	0.9920	0.9893
lag_4	0.9893	0.9920	0.9946	0.9972	1.0000	0.9972	0.9946	0.9920
lag_5	0.9867	0.9893	0.9920	0.9946	0.9972	1.0000	0.9972	0.9946
lag_6	0.9841	0.9867	0.9893	0.9920	0.9946	0.9972	1.0000	0.9972
lag_7	0.9815	0.9842	0.9867	0.9893	0.9920	0.9946	0.9972	1.0000

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O’Reilly Media.

```python
from statsmodels.tsa.stattools import adfuller
#Tests for stationarity using the Augmented Dickey-Fuller (ADF) test

adfuller(data[sym].dropna())
```

```
(-1.9488969577009954,
 0.3094193074034718,
 0,
 2515,
 {'1%': -3.4329527780962255,
  '10%': -2.567382133955709,
  '5%': -2.8626898965523724},
 8446.683102944744)
```

Market Prediction Based on Returns Data

- **Statistical inefficiencies**
  - are given when a model is able to predict the direction of the future price movement with a certain edge (say, the prediction is correct in 55% or 60% of the cases)
- **Economic inefficiencies**
  - would only be given if the statistical inefficiencies can be exploited profitably through a trading strategy that takes into account, for example, transaction costs.

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Market Prediction Based on Returns Data

- Create data sets with lagged log returns data
- The normalized lagged log returns data is also tested for stationarity (given)
- The features are tested for correlation (not correlated)
- Time-series-related data
  - weak form market efficiency

```python
rets = np.log(data / data.shift(1))
rets.dropna(inplace=True)
rets
```

log returns

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-05	0.0017	0.0003	-0.0005	0.0059	0.0175	2.6436e-03	3.1108e-03	-0.0350	-2.9883e-03	-0.0012	0.0096	-0.0009
2010-01-06	-0.0160	-0.0062	-0.0034	-0.0183	-0.0107	7.0379e-04	5.4538e-04	-0.0099	3.0577e-03	0.0176	0.0240	0.0164
2010-01-07	-0.0019	-0.0104	-0.0097	-0.0172	0.0194	4.2124e-03	3.9933e-03	-0.0052	-6.5437e-03	-0.0058	-0.0049	-0.0062
2010-01-08	0.0066	0.0068	0.0111	0.0267	-0.0191	3.3223e-03	2.8775e-03	-0.0500	6.5437e-03	0.0037	0.0150	0.0050
2010-01-11	-0.0089	-0.0128	0.0057	-0.0244	-0.0159	1.3956e-03	1.7452e-03	-0.0325	6.9836e-03	0.0144	0.0066	0.0132
...	...	...	...	...	...	...	...	...	...	...	...	...
2019-12-24	0.0010	-0.0002	0.0030	-0.0021	0.0036	3.1131e-05	-1.9543e-04	0.0047	9.0200e-05	0.0091	0.0315	0.0094
2019-12-26	0.0196	0.0082	0.0069	0.0435	0.0056	5.3092e-03	5.1151e-03	-0.0016	8.1143e-04	0.0083	0.0145	0.0078
2019-12-27	-0.0004	0.0018	0.0043	0.0006	-0.0024	-2.4775e-04	3.3951e-05	0.0598	7.0945e-03	-0.0006	-0.0072	-0.0004
2019-12-30	0.0059	-0.0087	-0.0077	-0.0123	-0.0037	-5.5285e-03	-5.7976e-03	0.0985	1.9667e-03	0.0031	0.0212	0.0021
2019-12-31	0.0073	0.0007	0.0039	0.0005	0.0006	2.4264e-03	2.9417e-03	-0.0728	1.1604e-03	0.0012	-0.0071	0.0019

2515 rows × 12 columns

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
dfs = {}
for sym in data:
   df, cols = add_lags(rets, sym, lags)
   mu, std = df[cols].mean(), df[cols].std()
   df[cols] = (df[cols] - mu) / std
   dfs[sym] = df
dfs[sym].head()
```

Date	GLD	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7
2010-01-14	0.0044	0.9570	-2.1692	1.3386	0.4959	-0.6434	1.6613	-0.1028
2010-01-15	-0.0105	0.4379	0.9571	-2.1689	1.3388	0.4966	-0.6436	1.6614
2010-01-19	0.0059	-1.0842	0.4385	0.9562	-2.1690	1.3395	0.4958	-0.6435
2010-01-20	-0.0234	0.5967	-1.0823	0.4378	0.9564	-2.1686	1.3383	0.4958
2010-01-21	-0.0145	-2.4045	0.5971	-1.0825	0.4379	0.9571	-2.1680	1.3384

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Augmented Dickey-Fuller (ADF)
Tests for stationarity of the time series data

```
adfuller(dfs[sym]['lag_1'])
```

```
(-51.568251505825536,
 0.0,
 0,
 2507,
 {'1%': -3.4329610922579095,
  '10%': -2.567384088736619,
  '5%': -2.8626935681060375},
 7017.165474260225)
```

Shows the correlation data for the features

```
dfs[sym].corr()
```

	GLD	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7
GLD	1.0000	-0.0297	0.0003	1.2635e-02	-0.0026	-5.9392e-03	0.0099	-0.0013
lag_1	-0.0297	1.0000	-0.0305	8.1418e-04	0.0128	-2.8765e-03	-0.0053	0.0098
lag_2	0.0003	-0.0305	1.0000	-3.1617e-02	0.0003	1.3234e-02	-0.0043	-0.0052
lag_3	0.0126	0.0008	-0.0316	1.0000e+00	-0.0313	-6.8542e-06	0.0141	-0.0044
lag_4	-0.0026	0.0128	0.0003	-3.1329e-02	1.0000	-3.1761e-02	0.0002	0.0141
lag_5	-0.0059	-0.0029	0.0132	-6.8542e-06	-0.0318	1.0000e+00	-0.0323	0.0002
lag_6	0.0099	-0.0053	-0.0043	1.4115e-02	0.0002	-3.2289e-02	1.0000	-0.0324
lag_7	-0.0013	0.0098	-0.0052	-4.3869e-03	0.0141	2.1707e-04	-0.0324	1.0000

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

OLS Regression

```
from sklearn.metrics import accuracy_score
```

```
%%time
for sym in data:
    df = dfs[sym]
    reg = np.linalg.lstsq(df[cols], df[sym], rcond=-1)[0]
    pred = np.dot(df[cols], reg)
    acc = accuracy_score(np.sign(df[sym]), np.sign(pred))
    print(f'OLS | {sym:10s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

OLS Regression Accuracy

```
OLS | AAPL.O       | acc=0.5056
OLS | MSFT.O       | acc=0.5088
OLS | INTC.O       | acc=0.5040
OLS | AMZN.O       | acc=0.5048
OLS | GS.N         | acc=0.5080
OLS | SPY          | acc=0.5080
OLS | .SPX         | acc=0.5167
OLS | .VIX         | acc=0.5291
OLS | EUR=         | acc=0.4984
OLS | XAU=         | acc=0.5207
OLS | GDX          | acc=0.5307
OLS | GLD          | acc=0.5072
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```
from sklearn.neural_network import MLPRegressor
```

```
%%time
for sym in data.columns:
   df = dfs[sym]
   model = MLPRegressor(hidden_layer_sizes=[512],
                    random_state=100,
                    max_iter=1000,
                    early_stopping=True,
                    validation_fraction=0.15,
                    shuffle=False)
   model.fit(df[cols].values, df[sym].values)
   pred = model.predict(df[cols].values)
   acc = accuracy_score(np.sign(df[sym].values),
   np.sign(pred))
   print(f'MLP | {sym:10s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Scikit-learn MLPRegressor Accuracy

```
MLP | AAPL.O      | acc=0.6005
MLP | MSFT.O      | acc=0.5853
MLP | INTC.O      | acc=0.5766
MLP | AMZN.O      | acc=0.5510
MLP | GS.N        | acc=0.6527
MLP | SPY         | acc=0.5419
MLP | .SPX        | acc=0.5399
MLP | .VIX        | acc=0.6579
MLP | EUR=        | acc=0.5642
MLP | XAU=        | acc=0.5522
MLP | GDX         | acc=0.6029
MLP | GLD         | acc=0.5259
```

```python
import tensorflow as tf
from keras.layers import Dense
from keras.models import Sequential

np.random.seed(100)
tf.random.set_seed(100)

def create_model(problem='regression'):
    model = Sequential()
    model.add(Dense(512, input_dim=len(cols), activation='relu'))
    if problem == 'regression’:
        model.add(Dense(1, activation='linear’))
        model.compile(loss='mse', optimizer='adam')
    else:
        model.add(Dense(1, activation='sigmoid’))
        model.compile(loss='binary_crossentropy', optimizer='adam')
    return model
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O’Reilly Media.

```python
%%time
for sym in data.columns[:]:
    df = dfs[sym]
    model = create_model()
    model.fit(df[cols], df[sym], epochs=25, verbose=False)
    pred = model.predict(df[cols])
    acc = accuracy_score(np.sign(df[sym]), np.sign(pred))
    print(f'DNN | {sym:10s} | acc={acc:.4f}')
```

TF Keras DNN Accuracy

```
DNN | AAPL.O       | acc=0.6069
DNN | MSFT.O       | acc=0.6260
DNN | INTC.O       | acc=0.6344
DNN | AMZN.O       | acc=0.6316
DNN | GS.N         | acc=0.6045
DNN | SPY          | acc=0.5610
DNN | .SPX         | acc=0.5435
DNN | .VIX         | acc=0.6096
DNN | EUR=         | acc=0.5817
DNN | XAU=         | acc=0.6017
DNN | GDX          | acc=0.6164
DNN | GLD          | acc=0.5973
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Train Data (0.8): In-Sample
Test Data (0.2): Out-of-Sample

```
split = int(len(dfs[sym]) * 0.8)
```

```
%%time
for sym in data.columns:
    df = dfs[sym]
    train = df.iloc[:split]
    reg = np.linalg.lstsq(train[cols], train[sym], rcond=-1)[0]
    test = df.iloc[split:]
    pred = np.dot(test[cols], reg)
    acc = accuracy_score(np.sign(test[sym]), np.sign(pred))
    print(f'OLS | {sym:10s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

OLS Out-of-Sample Accuracy

```
OLS | AAPL.O      | acc=0.5219
OLS | MSFT.O      | acc=0.4960
OLS | INTC.O      | acc=0.5418
OLS | AMZN.O      | acc=0.4841
OLS | GS.N        | acc=0.4980
OLS | SPY         | acc=0.5020
OLS | .SPX        | acc=0.5120
OLS | .VIX        | acc=0.5458
OLS | EUR=        | acc=0.4482
OLS | XAU=        | acc=0.5299
OLS | GDX         | acc=0.5159
OLS | GLD         | acc=0.5100
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
%%time
for sym in data.columns:
    df = dfs[sym]
    train = df.iloc[:split]
    model = MLPRegressor(hidden_layer_sizes=[512],
                         random_state=100,
                         max_iter=1000,
                         early_stopping=True,
                         validation_fraction=0.15,
                         shuffle=False)
    model.fit(train[cols].values, train[sym].values)
    test = df.iloc[split:]
    pred = model.predict(test[cols].values)
    acc = accuracy_score(np.sign(test[sym].values), np.sign(pred))
    print(f'MLP | {sym:10s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

MLP Out-of-Sample Accuracy

```
MLP | AAPL.O       | acc=0.4920
MLP | MSFT.O       | acc=0.5279
MLP | INTC.O       | acc=0.5279
MLP | AMZN.O       | acc=0.4641
MLP | GS.N         | acc=0.5040
MLP | SPY          | acc=0.5259
MLP | .SPX         | acc=0.5478
MLP | .VIX         | acc=0.5279
MLP | EUR=         | acc=0.4980
MLP | XAU=         | acc=0.5239
MLP | GDX          | acc=0.4880
MLP | GLD          | acc=0.5000
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
%%time
for sym in data.columns:
   df = dfs[sym]
   train = df.iloc[:split]
   model = create_model()
   model.fit(train[cols], train[sym], epochs=50,
   verbose=False)
   test = df.iloc[split:]
   pred = model.predict(test[cols])
   acc = accuracy_score(np.sign(test[sym]), np.sign(pred))
   print(f'DNN | {sym:10s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

DNN Out-of-Sample Accuracy

```
DNN | AAPL.O     | acc=0.4701
DNN | MSFT.O     | acc=0.4960
DNN | INTC.O     | acc=0.5040
DNN | AMZN.O     | acc=0.4920
DNN | GS.N       | acc=0.5538
DNN | SPY        | acc=0.5299
DNN | .SPX       | acc=0.5458
DNN | .VIX       | acc=0.5020
DNN | EUR=       | acc=0.5100
DNN | XAU=       | acc=0.4940
DNN | GDX        | acc=0.4661
DNN | GLD        | acc=0.4880
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Market Prediction with More Features

- In trading, there is a long tradition of using technical indicators to generate, based on observed patterns, buy or sell signals.
- Such technical indicators, basically of any kind, can also be used as features for the training of neural networks.
- SMA, rolling minimum and maximum values, momentum, and rolling volatility as features

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
url = 'http://hilpisch.com/aiif_eikon_eod_data.csv'

data = pd.read_csv(url, index_col=0, parse_dates=True).dropna()
data
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-04	30.5728	30.950	20.88	133.90	173.08	113.33	1132.99	20.04	1.4411	1120.0000	47.71	109.80
2010-01-05	30.6257	30.960	20.87	134.69	176.14	113.63	1136.52	19.35	1.4368	1118.6500	48.17	109.70
2010-01-06	30.1385	30.770	20.80	132.25	174.26	113.71	1137.14	19.16	1.4412	1138.5000	49.34	111.51
2010-01-07	30.0828	30.452	20.60	130.00	177.67	114.19	1141.69	19.06	1.4318	1131.9000	49.10	110.82
2010-01-08	30.2828	30.660	20.83	133.52	174.31	114.57	1144.98	18.13	1.4412	1136.1000	49.84	111.37
...	...	...	...	...	...	...	...	...	...	...	...	...
2019-12-24	284.2700	157.380	59.41	1789.21	229.91	321.23	3223.38	12.67	1.1087	1498.8100	28.66	141.27
2019-12-26	289.9100	158.670	59.82	1868.77	231.21	322.94	3239.91	12.65	1.1096	1511.2979	29.08	142.38
2019-12-27	289.8000	158.960	60.08	1869.80	230.66	322.86	3240.02	13.43	1.1175	1510.4167	28.87	142.33
2019-12-30	291.5200	157.590	59.62	1846.89	229.80	321.08	3221.29	14.82	1.1197	1515.1230	29.49	142.63
2019-12-31	293.6500	157.700	59.85	1847.84	229.93	321.86	3230.78	13.78	1.1210	1517.0100	29.28	142.90

2516 rows × 12 columns

```python
def add_lags(data, ric, lags, window=50):
    cols = []
    df = pd.DataFrame(data[ric])
    df.dropna(inplace=True)
    df['r'] = np.log(df / df.shift())
    df['sma'] = df[ric].rolling(window).mean()
    df['min'] = df[ric].rolling(window).min()
    df['max'] = df[ric].rolling(window).max()
    df['mom'] = df['r'].rolling(window).mean()
    df['vol'] = df['r'].rolling(window).std()
    df.dropna(inplace=True)
    df['d'] = np.where(df['r'] > 0, 1, 0)
    features = [ric, 'r', 'd', 'sma', 'min', 'max', 'mom', 'vol']
    for f in features:
        for lag in range(1, lags + 1):
            col = f'{f}_lag_{lag}'
            df[col] = df[f].shift(lag)
            cols.append(col)
    df.dropna(inplace=True)
    return df, cols
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
lags = 5

dfs = {}
for ric in data:
   df, cols = add_lags(data, ric, lags)
   dfs[ric] = df.dropna(), cols

len(cols)
```

```
40
```

```python
from sklearn.neural_network import MLPClassifier
```

```python
%%time
for ric in data:
  model = MLPClassifier(hidden_layer_sizes=[512],
                        random_state=100,
                        max_iter=1000,
                        early_stopping=True,
                        validation_fraction=0.15,
                        shuffle=False)
  df, cols = dfs[ric]
  df[cols] = (df[cols] - df[cols].mean()) / df[cols].std()
  model.fit(df[cols].values, df['d'].values)
  pred = model.predict(df[cols].values)
  acc = accuracy_score(df['d'].values, pred)
  print(f'IN-SAMPLE | {ric:7s} | acc={acc:.4f}')
```

MLP In-Sample Accuracy

```
IN-SAMPLE | AAPL.O   | acc=0.5510
IN-SAMPLE | MSFT.O   | acc=0.5376
IN-SAMPLE | INTC.O   | acc=0.5607
IN-SAMPLE | AMZN.O   | acc=0.5559
IN-SAMPLE | GS.N     | acc=0.5794
IN-SAMPLE | SPY      | acc=0.5729
IN-SAMPLE | .SPX     | acc=0.5941
IN-SAMPLE | .VIX     | acc=0.6940
IN-SAMPLE | EUR=     | acc=0.5766
IN-SAMPLE | XAU=     | acc=0.5672
IN-SAMPLE | GDX      | acc=0.5847
IN-SAMPLE | GLD      | acc=0.5567
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
%%time
for ric in data:
    model = create_model('classification')
    df, cols = dfs[ric]
    df[cols] = (df[cols] - df[cols].mean()) / df[cols].std()
    model.fit(df[cols], df['d'], epochs=50, verbose=False)
    pred = np.where(model.predict(df[cols]) > 0.5, 1, 0)
    acc = accuracy_score(df['d'], pred)
    print(f'IN-SAMPLE | {ric:7s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O’Reilly Media.

TF Keras DNN In-Sample Accuracy

```
IN-SAMPLE | AAPL.O   | acc=0.7042
IN-SAMPLE | MSFT.O   | acc=0.6928
IN-SAMPLE | INTC.O   | acc=0.6969
IN-SAMPLE | AMZN.O   | acc=0.6713
IN-SAMPLE | GS.N     | acc=0.6924
IN-SAMPLE | SPY      | acc=0.6806
IN-SAMPLE | .SPX     | acc=0.6920
IN-SAMPLE | .VIX     | acc=0.7347
IN-SAMPLE | EUR=     | acc=0.6766
IN-SAMPLE | XAU=     | acc=0.7038
IN-SAMPLE | GDX      | acc=0.6806
IN-SAMPLE | GLD      | acc=0.6936
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
def train_test_model(model):
    for ric in data:
    df, cols = dfs[ric]
    split = int(len(df) * 0.85)
    train = df.iloc[:split].copy()
    mu, std = train[cols].mean(), train[cols].std()
    train[cols] = (train[cols] - mu) / std
    model.fit(train[cols].values, train['d'].values)
    test = df.iloc[split:].copy()
    test[cols] = (test[cols] - mu) / std
    pred = model.predict(test[cols].values)
    acc = accuracy_score(test['d'].values, pred)
    print(f'OUT-OF-SAMPLE | {ric:7s} | acc={acc:.4f}')
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
model_mlp = MLPClassifier(hidden_layer_sizes=[512],
                          random_state=100,
                          max_iter=1000,
                          early_stopping=True,
                          validation_fraction=0.15,
                          shuffle=False)
```

```python
train_test_model(model_mlp)
```

```
train_test_model(model_mlp)
```

```
OUT-OF-SAMPLE | AAPL.O   | acc=0.4432
OUT-OF-SAMPLE | MSFT.O   | acc=0.4595
OUT-OF-SAMPLE | INTC.O   | acc=0.5000
OUT-OF-SAMPLE | AMZN.O   | acc=0.5270
OUT-OF-SAMPLE | GS.N     | acc=0.4838
OUT-OF-SAMPLE | SPY      | acc=0.4811
OUT-OF-SAMPLE | .SPX     | acc=0.5027
OUT-OF-SAMPLE | .VIX     | acc=0.5676
OUT-OF-SAMPLE | EUR=     | acc=0.4649
OUT-OF-SAMPLE | XAU=     | acc=0.5514
OUT-OF-SAMPLE | GDX      | acc=0.5162
OUT-OF-SAMPLE | GLD      | acc=0.4946
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O’Reilly Media.

```python
from sklearn.ensemble import BaggingClassifier

base_estimator = MLPClassifier(hidden_layer_sizes=[256],
                      random_state=100,
                      max_iter=1000,
                      early_stopping=True,
                      validation_fraction=0.15,
                      shuffle=False)

model_bag = BaggingClassifier(base_estimator=base_estimator,
                      n_estimators=35,
                      max_samples=0.25,
                      max_features=0.5,
                      bootstrap=False,
                      bootstrap_features=True,
                      n_jobs=8,
                      random_state=100
                      )
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O’Reilly Media.

```
train_test_model(model_bag)
```

```
OUT-OF-SAMPLE | AAPL.O  | acc=0.5000
OUT-OF-SAMPLE | MSFT.O  | acc=0.5703
OUT-OF-SAMPLE | INTC.O  | acc=0.5054
OUT-OF-SAMPLE | AMZN.O  | acc=0.5270
OUT-OF-SAMPLE | GS.N    | acc=0.5135
OUT-OF-SAMPLE | SPY     | acc=0.5568
OUT-OF-SAMPLE | .SPX    | acc=0.5514
OUT-OF-SAMPLE | .VIX    | acc=0.5432
OUT-OF-SAMPLE | EUR=    | acc=0.5054
OUT-OF-SAMPLE | XAU=    | acc=0.5351
OUT-OF-SAMPLE | GDX     | acc=0.5054
OUT-OF-SAMPLE | GLD     | acc=0.5189
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Market Prediction Intraday

- Weakly efficient on an end-of-day basis
- Weakly inefficient intraday
  - Intraday Data
  - hourly data

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

Intraday Data

```
url = 'http://hilpisch.com/aiif_eikon_id_data.csv'
data = pd.read_csv(url, index_col=0, parse_dates=True) # .dropna()
data.tail()
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2019-12-31 20:00:00	292.36	157.2845	59.575	1845.22	228.92	320.94	3219.75	14.16	1.1215	1519.6451	29.40	143.12
2019-12-31 21:00:00	293.37	157.4900	59.820	1846.95	229.89	321.89	3230.56	13.92	1.1216	1517.3600	29.29	142.93
2019-12-31 22:00:00	293.82	157.9000	59.990	1850.20	229.93	322.39	3230.78	13.78	1.1210	1517.0100	29.30	142.90
2019-12-31 23:00:00	293.75	157.8300	59.910	1851.00	NaN	322.22	NaN	NaN	1.1211	1516.8900	29.40	142.88
2020-01-01 00:00:00	293.81	157.8800	59.870	1850.10	NaN	322.32	NaN	NaN	1.1211	NaN	29.34	143.00

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

```python
lags = 5

dfs = {}
for ric in data:
   df, cols = add_lags(data, ric, lags)
   dfs[ric] = df, cols
```

Source: Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media.

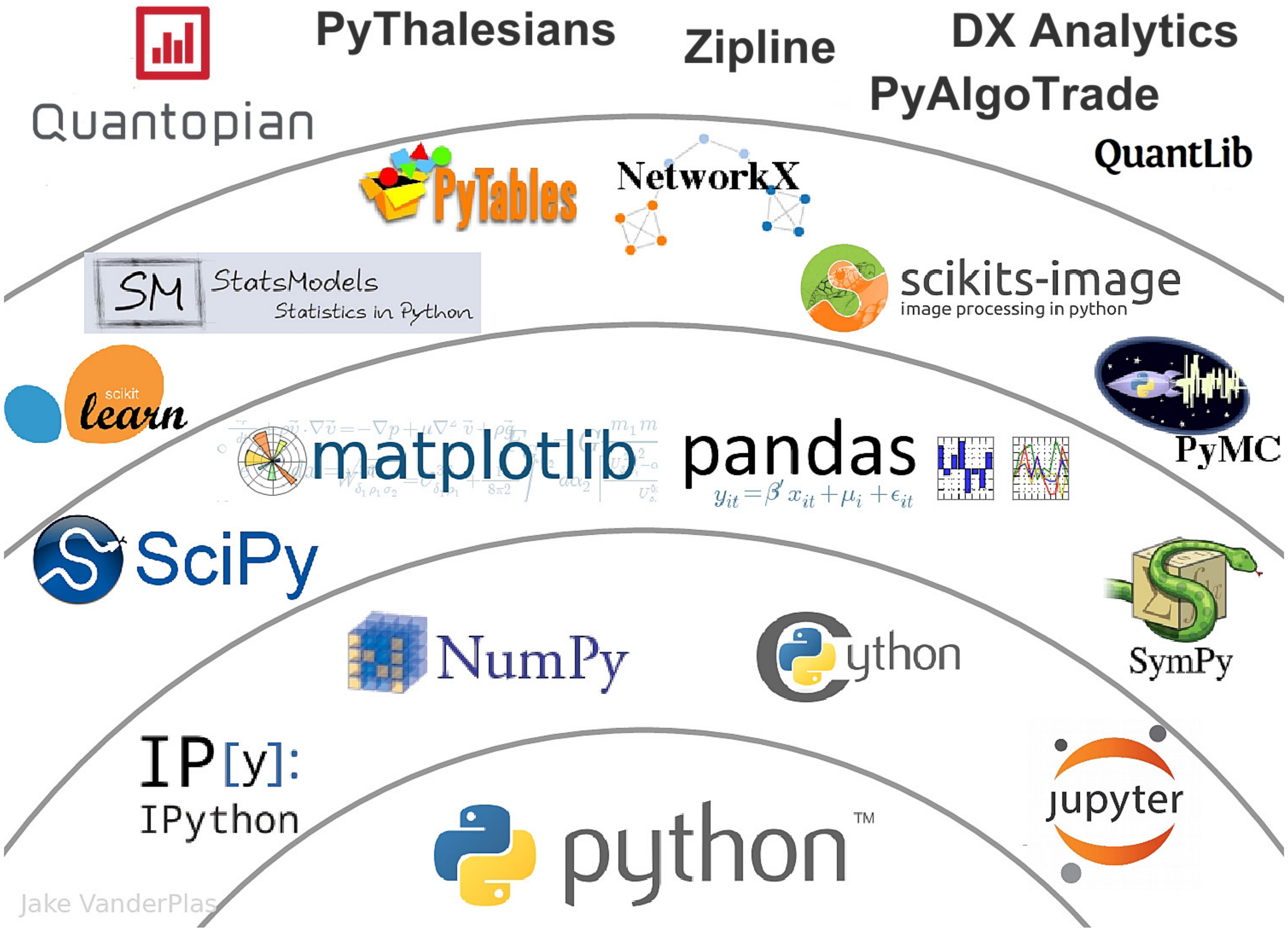
```
train_test_model(model_mlp)
```

```
OUT-OF-SAMPLE | AAPL.O   | acc=0.5420
OUT-OF-SAMPLE | MSFT.O   | acc=0.4930
OUT-OF-SAMPLE | INTC.O   | acc=0.5549
OUT-OF-SAMPLE | AMZN.O   | acc=0.4709
OUT-OF-SAMPLE | GS.N     | acc=0.5184
OUT-OF-SAMPLE | SPY      | acc=0.4860
OUT-OF-SAMPLE | .SPX     | acc=0.5019
OUT-OF-SAMPLE | .VIX     | acc=0.4885
OUT-OF-SAMPLE | EUR=     | acc=0.5130
OUT-OF-SAMPLE | XAU=     | acc=0.4824
OUT-OF-SAMPLE | GDX      | acc=0.4765
OUT-OF-SAMPLE | GLD      | acc=0.5455
```

```
train_test_model(model_bag)
```

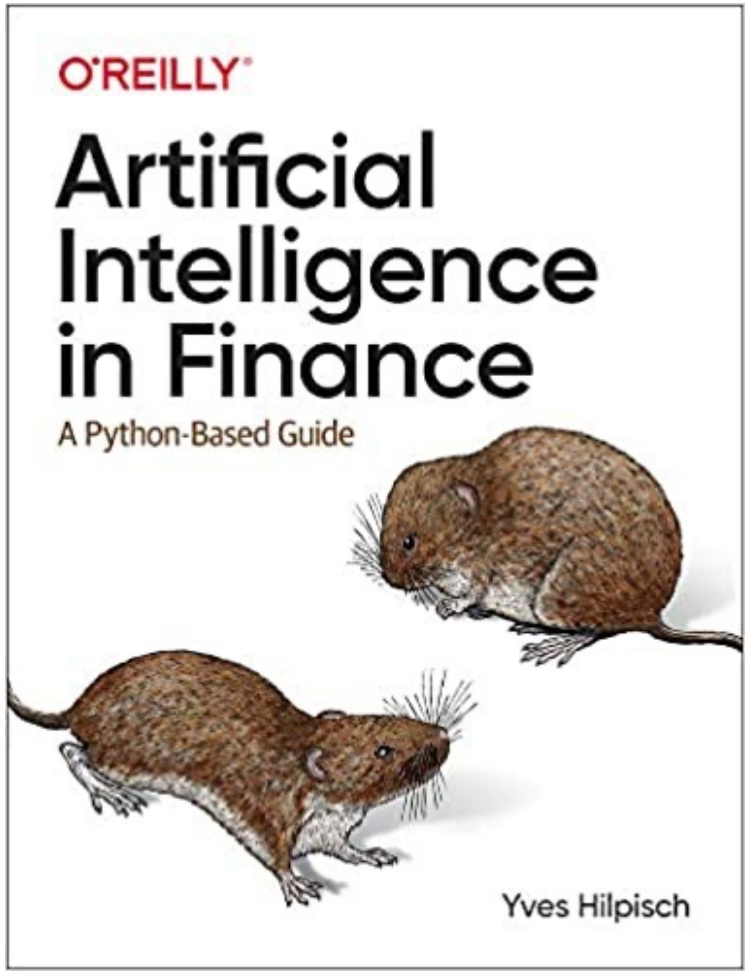
```
OUT-OF-SAMPLE | AAPL.O   | acc=0.5660
OUT-OF-SAMPLE | MSFT.O   | acc=0.5551
OUT-OF-SAMPLE | INTC.O   | acc=0.5072
OUT-OF-SAMPLE | AMZN.O   | acc=0.4830
OUT-OF-SAMPLE | GS.N     | acc=0.5020
OUT-OF-SAMPLE | SPY      | acc=0.4680
OUT-OF-SAMPLE | .SPX     | acc=0.4677
OUT-OF-SAMPLE | .VIX     | acc=0.5161
OUT-OF-SAMPLE | EUR=     | acc=0.5242
OUT-OF-SAMPLE | XAU=     | acc=0.5229
OUT-OF-SAMPLE | GDX      | acc=0.5107
OUT-OF-SAMPLE | GLD      | acc=0.5475
```

The Quant Finance PyData Stack



Source: http://nbviewer.jupyter.org/format/slides/github/quantopian/pyfolio/blob/master/pyfolio/examples/overview_slides.ipynb#/5

Yves Hilpisch (2020),
Artificial Intelligence in Finance:
A Python-Based Guide,
O'Reilly



Source: https://www.amazon.com/Artificial-Intelligence-Finance-Python-Based-Guide/dp/1492055433

Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly

yhilpisch / aiif Public

https://github.com/yhilpisch/aiif

Notifications | Star 98 | Fork 77

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights

main | 1 branch | 0 tags | Go to file | Code

yves Code updates for TF 2.3. e334251 on Dec 8, 2020 | 4 commits

code	Code updates for TF 2.3.	11 months ago
.gitignore	Code updates for TF 2.3.	11 months ago
LICENSE.txt	Code updates.	11 months ago
README.md	Code updates.	11 months ago

README.md

Artificial Intelligence in Finance

About this Repository

This repository provides Python code and Jupyter Notebooks accompanying the **Artificial Intelligence in Finance** book published by O'Reilly.

O'REILLY

About

Jupyter Notebooks and code for the book Artificial Intelligence in Finance (O'Reilly) by Yves Hilpisch.

home.tpq.io/books/aiif

Readme

View license

Releases

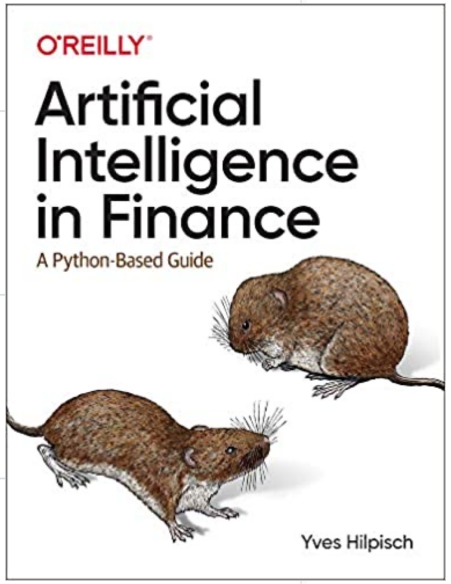
No releases published

Packages

No packages published

Languages

Jupyter Notebook 97.4% Python 2.6%



Source: https://github.com/yhilpisch/aiif

Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly

yhilpisch / aiif Public

Notifications | Star 98 | Fork 77

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights

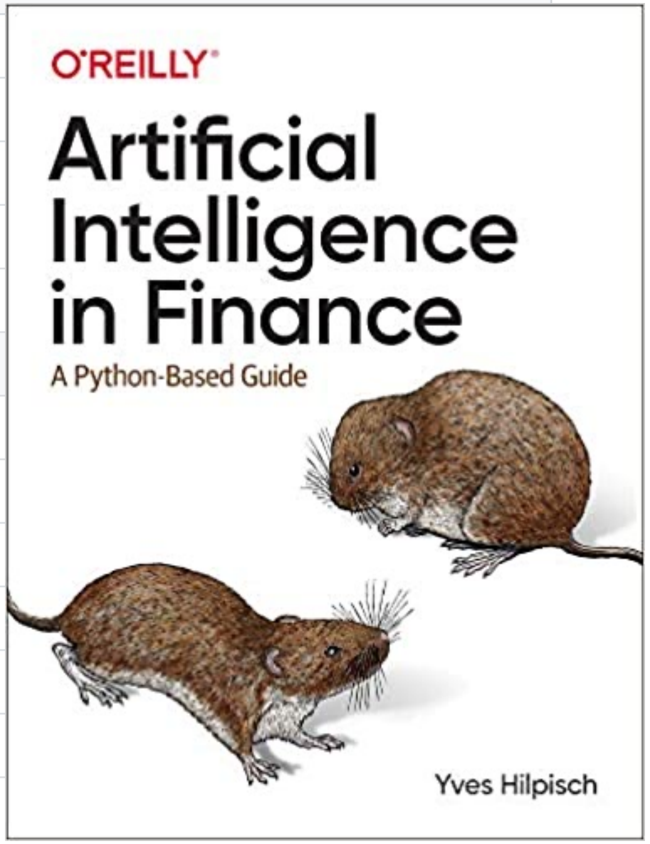
main | aiif / code /

https://github.com/yhilpisch/aiif/tree/main/code

Go to file

yves Code updates for TF 2.3. e334251 on Dec 8, 2020 History

..	
oanda	Code updates for TF 2.3.
01_artificial_intelligence.ipynb	Code updates for TF 2.3.
02_superintelligence.ipynb	Code updates for TF 2.3.
03_normative_finance.ipynb	Code updates for TF 2.3.
04_data_driven_finance_a.ipynb	Initial commit.
04_data_driven_finance_b.ipynb	Initial commit.
05_machine_learning.ipynb	Code updates for TF 2.3.
06_ai_first_finance.ipynb	Code updates for TF 2.3.
07_dense_networks.ipynb	Code updates for TF 2.3.
08_recurrent_networks.ipynb	Code updates for TF 2.3.
09_reinforcement_learning_a.ipynb	Code updates.
09_reinforcement_learning_b.ipynb	Code updates for TF 2.3.



Source: https://github.com/yhilpisch/aiif/tree/main/code

Python in Google Colab (Python101)

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT

python101.ipynb - Colaboratory

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT?authuser=2#scrollTo=wsh36fLxDKC3

python101.ipynb

File Edit View Insert Runtime Tools Help

COMMENT SHARE

CODE TEXT CELL CELL CONNECTED EDITING

```python
# Future Value
pv = 100
r = 0.1
n = 7
fv = pv * ((1 + (r)) ** n)
print(round(fv, 2))
```

```
194.87
```

[11]
```python
amount = 100
interest = 10 #10% = 0.01 * 10
years = 7

future_value = amount * ((1 + (0.01 * interest)) ** years)
print(round(future_value, 2))
```

```
194.87
```

[12]
```python
# Python Function def
def getfv(pv, r, n):
    fv = pv * ((1 + (r)) ** n)
    return fv
fv = getfv(100, 0.1, 7)
print(round(fv, 2))
```

```
194.87
```

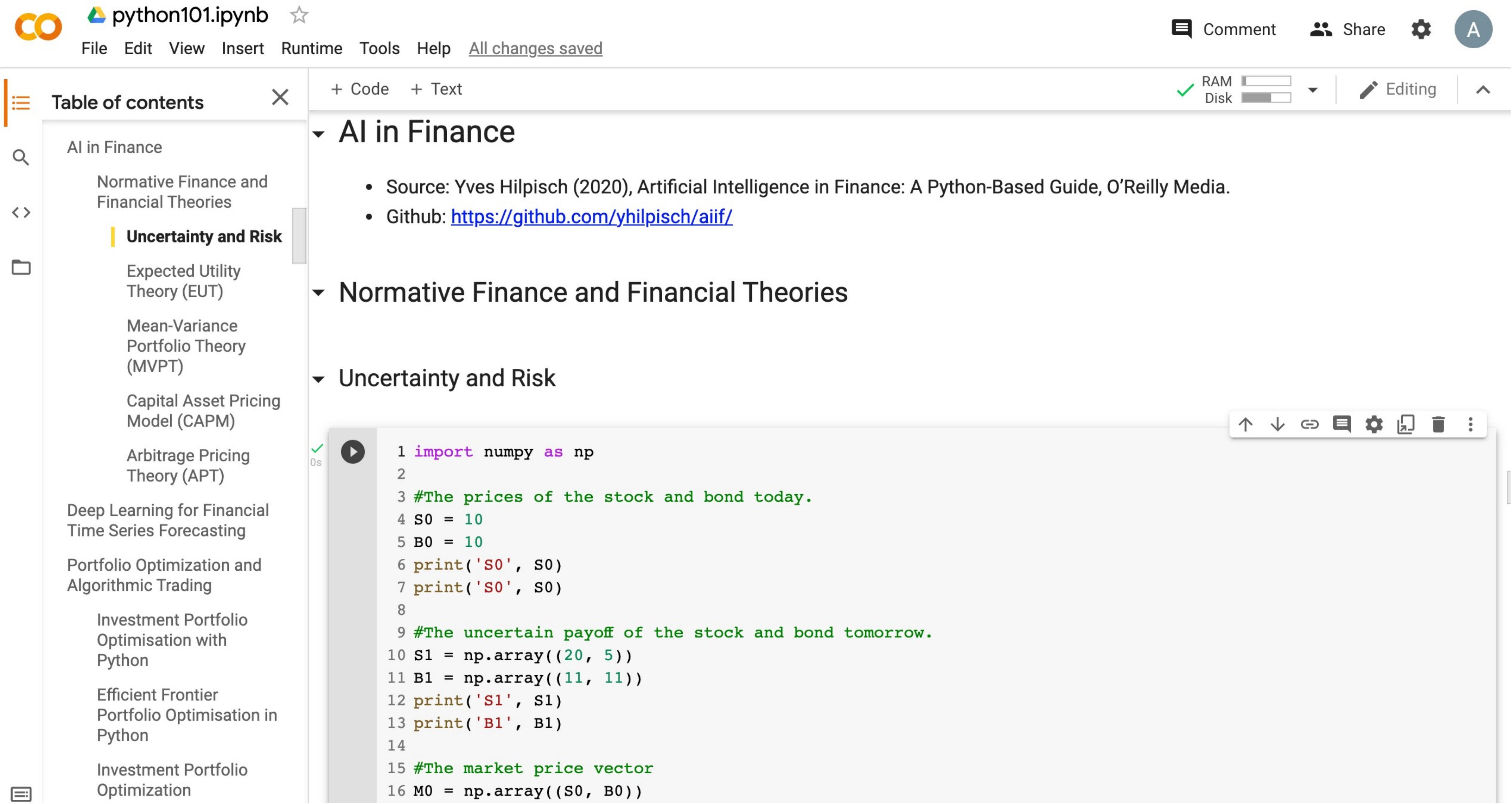
[13]
```python
# Python if else
score = 80
if score >=60 :
    print("Pass")
else:
    print("Fail")
```

```
Pass
```

https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT



https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)

python101.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text RAM Disk Editing

Table of contents

Data Driven Finance

Financial Econometrics and Regression

[18]

```python
import numpy as np

def f(x):
    return 2 + 1 / 2 * x

x = np.arange(-4, 5)
x
```

```
array([-4, -3, -2, -1,  0,  1,  2,  3,  4])
```

```python
y = f(x)
y
```

```
array([ 0.00,  0.50,  1.00,  1.50,  2.00,  2.50,  3.00,  3.50,  4.00])
```

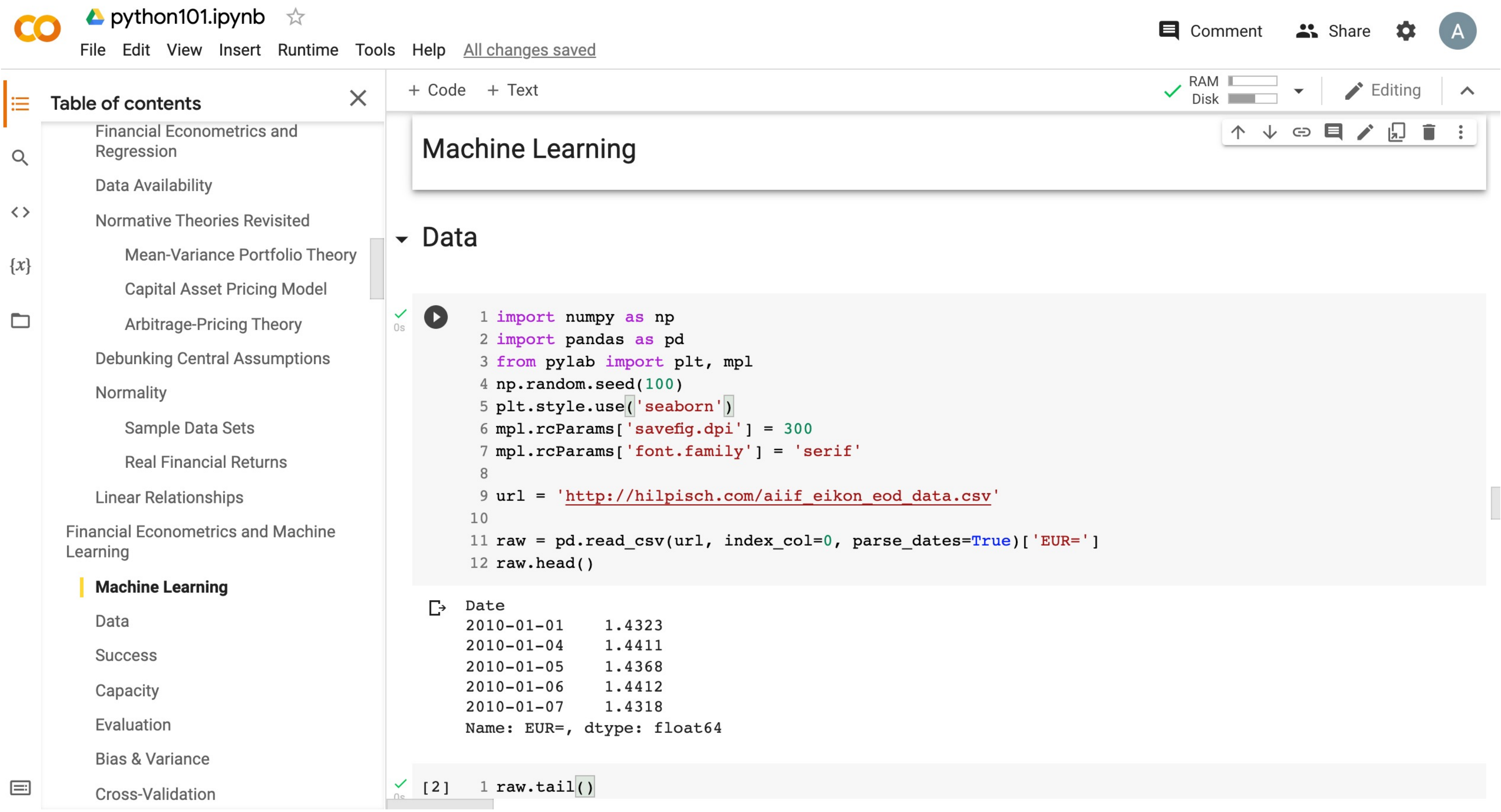
```python
print('x', x)

print('y', y)

beta = np.cov(x, y, ddof=0)[0, 1] / x.var()
print('beta', beta)
```

https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)



https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)

python101.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Code + Text RAM Disk Editing

Efficient Markets

```python
import numpy as np
import pandas as pd
from pylab import plt, mpl
plt.style.use('seaborn')
mpl.rcParams['savefig.dpi'] = 300
mpl.rcParams['font.family'] = 'serif'
pd.set_option('precision', 4)
np.set_printoptions(suppress=True, precision=4)

url = 'http://hilpisch.com/aiif_eikon_eod_data.csv'
data = pd.read_csv(url, index_col=0, parse_dates=True).dropna()
(data / data.iloc[0]).plot(figsize=(10, 6), cmap='coolwarm')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29f972f210>
```

https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)

python101.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

Table of contents

Market Prediction Based on Returns Data

```python
rets = np.log(data / data.shift(1))

rets.dropna(inplace=True)
rets
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-05	0.0017	0.0003	-0.0005	0.0059	0.0175	2.6436e-03	3.1108e-03	-0.0350	-2.9883e-03	-0.0012	0.0096	-0.0009
2010-01-06	-0.0160	-0.0062	-0.0034	-0.0183	-0.0107	7.0379e-04	5.4538e-04	-0.0099	3.0577e-03	0.0176	0.0240	0.0164
2010-01-07	-0.0019	-0.0104	-0.0097	-0.0172	0.0194	4.2124e-03	3.9933e-03	-0.0052	-6.5437e-03	-0.0058	-0.0049	-0.0062
2010-01-08	0.0066	0.0068	0.0111	0.0267	-0.0191	3.3223e-03	2.8775e-03	-0.0500	6.5437e-03	0.0037	0.0150	0.0050
2010-01-11	-0.0089	-0.0128	0.0057	-0.0244	-0.0159	1.3956e-03	1.7452e-03	-0.0325	6.9836e-03	0.0144	0.0066	0.0132
...	...	...	...	...	...	...	...	...	...	...	...	...
2019-12-24	0.0010	-0.0002	0.0030	-0.0021	0.0036	3.1131e-05	-1.9543e-04	0.0047	9.0200e-05	0.0091	0.0315	0.0094
2019-12-26	0.0196	0.0082	0.0069	0.0435	0.0056	5.3092e-03	5.1151e-03	-0.0016	8.1143e-04	0.0083	0.0145	0.0078
2019-12-27	-0.0004	0.0018	0.0043	0.0006	-0.0024	-2.4775e-04	3.3951e-05	0.0598	7.0945e-03	-0.0006	-0.0072	-0.0004

https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)

python101.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Code + Text RAM Disk Editing

Market Prediction With More Features

```
url = 'http://hilpisch.com/aiif_eikon_eod_data.csv'

data = pd.read_csv(url, index_col=0, parse_dates=True).dropna()
data
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2010-01-04	30.5728	30.950	20.88	133.90	173.08	113.33	1132.99	20.04	1.4411	1120.0000	47.71	109.80
2010-01-05	30.6257	30.960	20.87	134.69	176.14	113.63	1136.52	19.35	1.4368	1118.6500	48.17	109.70
2010-01-06	30.1385	30.770	20.80	132.25	174.26	113.71	1137.14	19.16	1.4412	1138.5000	49.34	111.51
2010-01-07	30.0828	30.452	20.60	130.00	177.67	114.19	1141.69	19.06	1.4318	1131.9000	49.10	110.82
2010-01-08	30.2828	30.660	20.83	133.52	174.31	114.57	1144.98	18.13	1.4412	1136.1000	49.84	111.37
...	...	...	...	...	...	...	...	...	...	...	...	...
2019-12-24	284.2700	157.380	59.41	1789.21	229.91	321.23	3223.38	12.67	1.1087	1498.8100	28.66	141.27
2019-12-26	289.9100	158.670	59.82	1868.77	231.21	322.94	3239.91	12.65	1.1096	1511.2979	29.08	142.38
2019-12-27	289.8000	158.960	60.08	1869.80	230.66	322.86	3240.02	13.43	1.1175	1510.4167	28.87	142.33
2019-12-30	291.5200	157.590	59.62	1846.89	229.80	321.08	3221.29	14.82	1.1197	1515.1230	29.49	142.63
2019-12-31	293.6500	157.700	59.85	1847.84	229.93	321.86	3230.78	13.78	1.1210	1517.0100	29.28	142.90

2516 rows × 12 columns

https://tinyurl.com/aintpupython101

Python in Google Colab (Python101)

python101.ipynb

File Edit View Insert Runtime Tools Help Saving...

Comment Share

Table of contents

+ Code + Text RAM Disk Editing

Market Prediction Intraday

```python
url = 'http://hilpisch.com/aiif_eikon_id_data.csv'
data = pd.read_csv(url, index_col=0, parse_dates=True) # .dropna()
data.tail()
```

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	GDX	GLD
2019-12-31 20:00:00	292.36	157.2845	59.575	1845.22	228.92	320.94	3219.75	14.16	1.1215	1519.6451	29.40	143.12
2019-12-31 21:00:00	293.37	157.4900	59.820	1846.95	229.89	321.89	3230.56	13.92	1.1216	1517.3600	29.29	142.93
2019-12-31 22:00:00	293.82	157.9000	59.990	1850.20	229.93	322.39	3230.78	13.78	1.1210	1517.0100	29.30	142.90
2019-12-31 23:00:00	293.75	157.8300	59.910	1851.00	NaN	322.22	NaN	NaN	1.1211	1516.8900	29.40	142.88
2020-01-01 00:00:00	293.81	157.8800	59.870	1850.10	NaN	322.32	NaN	NaN	1.1211	NaN	29.34	143.00

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5529 entries, 2019-03-01 00:00:00 to 2020-01-01 00:00:00
Data columns (total 12 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   AAPL.O   3384 non-null   float64
 1   MSFT.O   3378 non-null   float64
```

https://tinyurl.com/aintpupython101

Summary

- **Efficient Markets**
- **Market Prediction Based on Returns Data**
- **Market Prediction with More Features**
- **Market Prediction Intraday**

References

- Yves Hilpisch (2020), Artificial Intelligence in Finance: A Python-Based Guide, O'Reilly Media, https://github.com/yhilpisch/aiif .
- Max Tegmark (2017), Life 3.0: Being human in the age of artificial intelligence. Vintage.
- Ajay Agrawal, Joshua Gans, and Avi Goldfarb (2018). Prediction machines: the simple economics of artificial intelligence. Harvard Business Press.
- Eugene F. Fama (1995), "Random walks in stock market prices." Financial Analysts Journal 51, no. 1, 75-80.
- Ruey S. Tsay (2005), Analysis of financial time series, Wiley.
- Min-Yuh Day (2021), Python 101, https://tinyurl.com/aintpupython101