# Practical Data Science

## *in finance

Puneet Batra

# Goals

- Introduce relevant Python libraries
  - numpy
  - pandas
  - scikit-learn
- Use these libraries to
  - Extract Data
  - Align/clean data
  - Do something useful
- Why use these tools?

# numpy

- Python package used for scientific/numerical computing
- Lots of useful functions
  - ```
    >> np.exp(1)
    >> 2.718281828459045l
    ```
  - ```
    >> np.log([1,2,3])
    >> array([ 0.,  0.69314718,  1.09861229])
    ```
  - ```
    >> np.linalg.eig(foo)
         (array([ 2.53349352,  0.41821668,  0.55359127,  0.81049877,  0.68419976]),
          array([[ 0.43048555,  0.2161367 ,  0.53748643, -0.42905633,  0.54312626],
                 [ 0.5006138 ,  0.62269993, -0.58715677, -0.06323348, -0.11348603],
                 [ 0.50952931, -0.74753099, -0.37272446, -0.13964699,  0.15215944],
                 [ 0.43037763, -0.07736151,  0.44100365, -0.04520822, -0.78247191],
                 [ 0.34528682,  0.02724608,  0.18151453,  0.88902594,  0.23815963]]))
    ```
    ```

# pandas

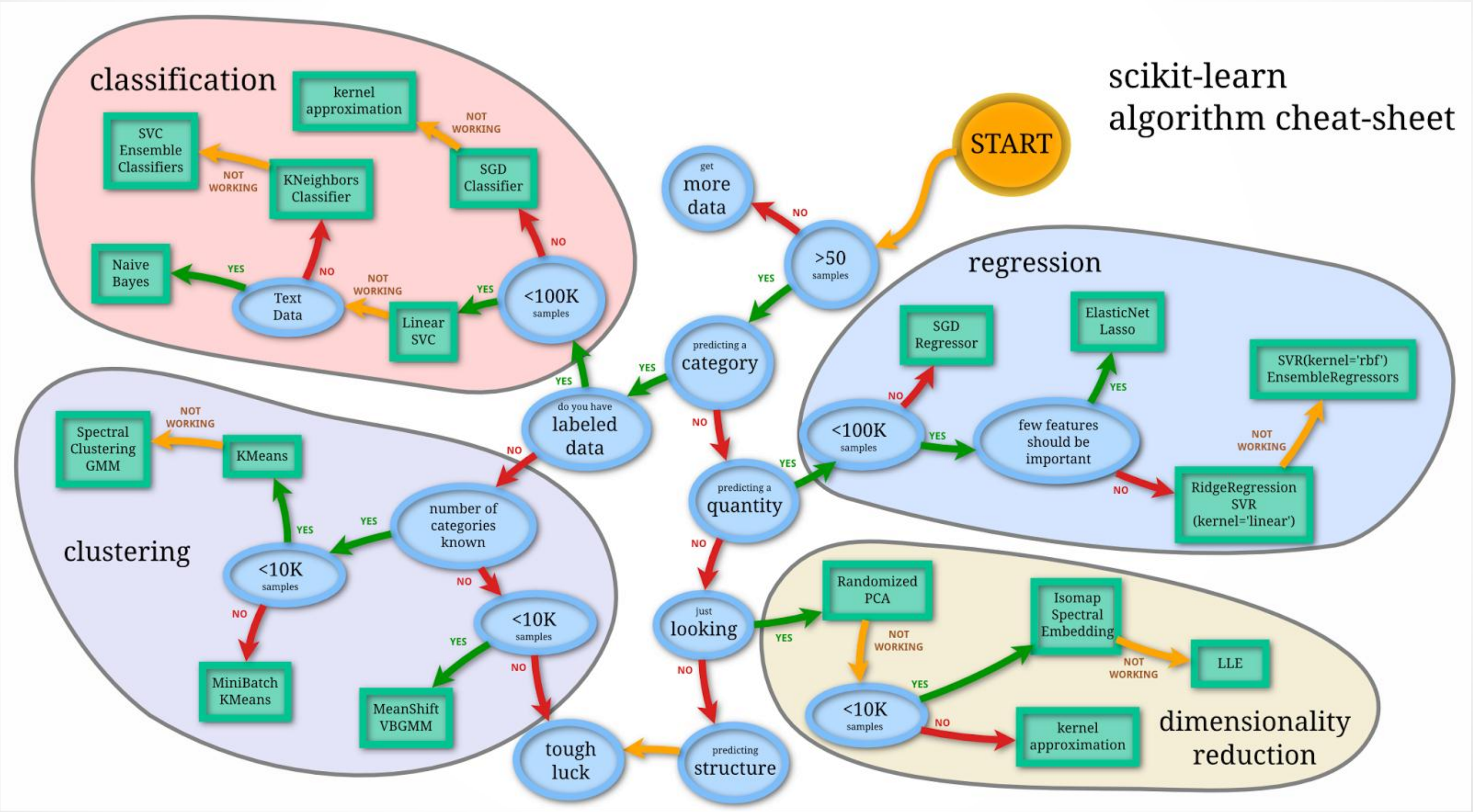
- Utility functions to read/write data
  - read_csv: Read a comma-delimited file and maintain column headers
- Friendly Data structures
  - Capable of holding a variety of data types e.g. strings, int, float, etc.
    - Series
      - one dimensional labeled array
    - DataFrame
      - Two dimensional object
      - Each column can be different types e.g. ticker, date, price
      - Can refer to columns by column names instead of numbers
        - E.g. stockReturns['JPM']
- Example (using numpy and pandas)

```
>> numpy.sum(stockReturns['JPM'])
   0.7285
```

# scikit-learn

- Open source machine learning library for Python
  - Classification
    - Decision Trees
    - Logistic Regression
    - etc.
  - Regression
    - Decision Trees
    - Support Vector Machines
    - etc.
  - Clustering
    - K Means, etc.
  - Preprocessing
    - e.g. binary encoding a.k.a. One Hot Encoding
  - Dimensionality Reduction
    - e.g. Principal Component Analysis

# scikit-learn



# Data – always step #1

## Comma separated file containing components of S&P 500 Index

| Ticker | Name | Sector | SubIndustry | Address |
|---|---|---|---|---|
| MMM | 3M Company | Industrials | Industrial Conglomerates | St Paul, Minnesota |
| ABT | Abbott Laboratories | Health Care | Health Care Equipment & Services | North Chicago, Illinois |
| ABBV | AbbVie | Health Care | Pharmaceuticals | North Chicago, Illinois |
| ANF | Abercrombie & Fitch Company A | Consumer Discretionary | Apparel, Accessories & Luxury Goods | New Albany, Ohio |
| ACE | ACE Limited | Financials | Property & Casualty Insurance | Zurich, Switzerland |
| ACN | Accenture plc | Information Technology | IT Consulting & Services | Dublin, Ireland |
| ACT | Actavis plc | Health Care | Pharmaceuticals | Parsippany, New Jersey |
| ADBE | Adobe Systems Inc | Information Technology | Application Software | San Jose, California |
| ADT | ADT Corp | Industrials | Diversified Commercial Services | Boca Raton, Florida |
| AMD | Advanced Micro Devices | Information Technology | Semiconductors | Sunnyvale, California |
| AES | AES Corp | Utilities | Electric Utilities | Arlington, Virginia |
| AET | Aetna Inc | Health Care | Managed Health Care | Hartford, Connecticut |
| AFL | AFLAC Inc | Financials | Life & Health Insurance | Columbus, Georgia |
| A | Agilent Technologies Inc | Health Care | Health Care Equipment & Services | Santa Clara, California |
| GAS | AGL Resources Inc. | Utilities | Gas Utilities | Atlanta, Georgia |
| APD | Air Products & Chemicals Inc | Materials | Industrial Gases | Allentown, Pennsylvania |
| ARG | Airgas Inc | Materials | Industrial Gases | Radnor, Pennsylvania |
| AKAM | Akamai Technologies Inc | Information Technology | Internet Software & Services | Cambridge, Massachusetts |
| AA | Alcoa Inc | Materials | Aluminum | New York, New York |
| ALXN | Alexion Pharmaceuticals | Health Care | Biotechnology | Cheshire, Connecticut |
| ATI | Allegheny Technologies Inc | Materials | Diversified Metals & Mining | Pittsburgh, Pennsylvania |
| AGN | Allergan Inc | Health Care | Pharmaceuticals | Irvine, California |
| ALL | Allstate Corp | Financials | Property & Casualty Insurance | Northfield Township, Illinois |
| ALTR | Altera Corp | Information Technology | Semiconductors | San Jose, California |
| MO | Altria Group Inc | Consumer Staples | Tobacco | Richmond, Virginia |
| AMZN | Amazon.com Inc | Consumer Discretionary | Internet Retail | Seattle, Washington |
| AEE | Ameren Corp | Utilities | Multi-Utilities & Unregulated Power | St. Louis, Missouri |
| AEP | American Electric Power | Utilities | Electric Utilities | Columbus, Ohio |

# Reading from a file

```
>> import pandas as pd
>> sp500components =
pd.read_csv("c:/dev/sp500_components_20131030.csv")

>>> sp500components
     <class 'pandas.core.frame.DataFrame'>
     Index: 500 entries, MMM to ZTS
     Data columns (total 5 columns):
     ticker          500  non-null values
     name            500  non-null values
     Sector          500  non-null values
     SubIndustry     497  non-null values
     Address         500  non-null values
     dtypes: object(5)
```

# Accessing the data

```
>>> sp500components['ticker'][0:5]
ticker
MMM         MMM
ABT         ABT
ABBV        ABBV
ANF         ANF
ACE         ACE
Name: ticker, dtype: object
```

**Why does the ticker appear twice??**

# Accessing the data

```
>>> sp500components[['ticker', 'Sector']][0:5]
       ticker                  Sector
ticker
MMM       MMM             Industrials
ABT       ABT             Health Care
ABBV     ABBV             Health Care
ANF       ANF  Consumer Discretionary
ACE       ACE              Financials
```

# Accessing the data

```
>>> sp500components[['ticker', 'Sector']][0:5]
       ticker                  Sector
ticker
MMM       MMM             Industrials
ABT       ABT             Health Care
ABBV     ABBV             Health Care
ANF       ANF  Consumer Discretionary
ACE       ACE              Financials
```

- The “Index” can be anything – very useful

```
>>> sp500components[['ticker', 'Sector']][0:5]

  ticker                  Sector
0    MMM             Industrials
1    ABT             Health Care
2   ABBV             Health Care
3    ANF  Consumer Discretionary
4    ACE              Financials
```

# Pulling data from the web

- Stock prices from Yahoo! Finance

```
import pandas.io.data as web
startDate = datetime.datetime(2002,1,1)
endDate = datetime.datetime(2013,10,29)
thisData = web.DataReader("AAPL", "yahoo", startDate, endDate)
```

```
>>> thisData
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2978 entries, 2002-01-02 00:00:00 to 2013-10-29 00:00:00
Data columns (total 6 columns):
Open         2978  non-null values
High         2978  non-null values
Low          2978  non-null values
Close        2978  non-null values
Volume       2978  non-null values
Adj Close    2978  non-null values
dtypes: float64(5), int64(1)
```

**What's the index here??**

# Processing the data

- Dates need to be aligned
  - merge is a very useful function in the pandas package
  - Can be used to merge two data-frames, on a specified index

```
>>> aapl = web.DataReader("AAPL", "yahoo", startDate, endDate)
>>> fslr = web.DataReader("FSLR", "yahoo", startDate, endDate)
>>> min(aapl.index)
    Timestamp('2002-01-02 00:00:00', tz=None)
>>> min(fslr.index)
    Timestamp('2006-11-17 00:00:00', tz=None)
>>> newData = pd.merge(aapl, fslr, how='outer', left_index=True, right_index=True)

>>> newData
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2978 entries, 2002-01-02 00:00:00 to 2013-10-29 00:00:00
Data columns (total 12 columns):
Open_x         2978  non-null values
High_x         2978  non-null values
Low_x          2978  non-null values
Close_x        2978  non-null values
Volume_x       2978  non-null values
Adj Close_x    2978  non-null values
Open_y         1748  non-null values
High_y         1748  non-null values
Low_y          1748  non-null values
Close_y        1748  non-null values
Volume_y       1748  non-null values
Adj Close_y    1748  non-null values
dtypes: float64(11), int64(1)
```

# Processing the data

```
>>> newData = pd.merge(aapl, fslr, how='outer',
left_index=True, right_index=True, suffixes=["AAPL","FSLR"])

>>> newData
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2978 entries, 2002-01-02 00:00:00 to 2013-10-29 00:00:00
Data columns (total 12 columns):
OpenAAPL         2978  non-null values
HighAAPL         2978  non-null values
LowAAPL          2978  non-null values
CloseAAPL        2978  non-null values
VolumeAAPL       2978  non-null values
Adj CloseAAPL    2978  non-null values
OpenFSLR         1748  non-null values
HighFSLR         1748  non-null values
LowFSLR          1748  non-null values
CloseFSLR        1748  non-null values
VolumeFSLR       1748  non-null values
Adj CloseFSLR    1748  non-null values
dtypes: float64(11), int64(1)
```

**Why is this useful??**

# Processing the data

```
>>> newData[["Adj CloseAAPL","Adj CloseFSLR"]][0:5]
Adj CloseAAPL  Adj CloseFSLR
Date
2002-01-02          11.33            NaN
2002-01-03          11.47            NaN
2002-01-04          11.52            NaN
2002-01-07          11.14            NaN
2002-01-08          10.99            NaN

>>> newData[["Adj CloseAAPL","Adj CloseFSLR"]][2973:2978]
Adj CloseAAPL  Adj CloseFSLR
Date
2013-10-23         524.96          53.08
2013-10-24         531.91          54.22
2013-10-25         525.96          52.80
2013-10-28         529.88          51.48
2013-10-29         516.68          52.37
```

**Dates are aligned!**

# Processing the data

- Prices are not very useful – need returns
  - Return can be calculated using 2 methods:
    - Method 1, more common: $(P_t – P_{t-1})/P_{t-1}$
    - Method 2, log of difference: $\log(P_t) – \log(P_{t-1}) = \log(Pt / Pt-1)$
  - Method 2 is easier to calculate for large data sets
  - It also yields symmetrical returns
    - Method 1:
      - If price goes from 100 to 101: 1.000%
      - If price goes from 101 to 100: -0.990%
    - Method 2:
      - If price goes from 100 to 101: 0.995%
      - If price goes from 101 to 100: -0.995%
  - The two methods are 99.9% correlated
  - In the following examples, I use method 2, for ease of calculation

# Processing the data
## Calculating returns

```
>> np.diff(np.log(aapl["Adj Close"].tolist()))
array([ 0.01228086, 0.00434972, -0.03354242, ..., -0.01124914, 0.0074254, -0.02522684])
```

**np.diff() takes a list and gives you lagged differences, e.g.:**

```
>> np.diff([1, 2, 4, 5, 6, 10, 11, 20])
        array([1, 2, 1, 1,  4,  1,  9])
```

We get one less item than in the original list!

```
>> len(aapl["Adj Close"])
   2978

>> len(np.diff(np.log(aapl["Adj Close"].tolist())))
   2977
```

We append a 0 to the front of the list

```
>> len(np.insert(np.diff(np.log(aapl["Adj Close"].tolist())), 0, [0]))
   2978
```

**As we saw before, log differences are equivalent to returns.**

```
aaplRets = np.insert(np.diff(np.log(aapl["Adj Close"].tolist())), 0, [0])
```

# Merged data. What next?

```
>> mergedRetData[['MMM','DRI','AAPL','JPM','HD']][0:9]
```

```
                 MMM       DRI      AAPL       JPM        HD
Date
2002-01-02  0.000000  0.000000  0.000000  0.000000  0.000000
2002-01-03 -0.003401  0.030013  0.012281  0.025986 -0.006897
2002-01-04  0.003175  0.056953  0.004350  0.044185  0.017785
2002-01-07 -0.012072 -0.025254 -0.033542 -0.002567 -0.007076
2002-01-08 -0.005745  0.000000 -0.013556 -0.007742  0.010093
2002-01-09 -0.003463  0.002554 -0.042757  0.002956 -0.016709
2002-01-10 -0.012567  0.003057 -0.020145  0.011009  0.001021
2002-01-11  0.005139  0.003047 -0.007782 -0.020650  0.003056
2002-01-14 -0.018103 -0.009682  0.003899 -0.027192 -0.018476
```

# Analyze the data

```
>>> cormat = mergedRetData.corr()
>>> thisKMeans = cluster.KMeans(10)
>>> thisKMeans.fit(cormat)
KMeans(copy_x=True, init='k-means++', man_jobs=1, precompute_distances=True,
verbose=0)
>>> cormat['AAPL']
A        0.404374
AA       0.390887
AAPL     1.000000
ABC      0.206329
...
YUM      0.300696
ZION     0.299398
ZMH      0.248488
Name: AAPL, Length: 436, dtype: float64
>>> cormat.shape
(436, 436)
```

# Save your output

```
clusterDistances = thisKMeans.transform(cormat)
clusterDistances[0,:]
array([ 2.24614501,  1.45824868,  1.38389929,  2.99676636,  2.27274464,
        2.19303793,  1.35246865,  1.90428341,  2.16030916,  1.67363007,
        2.29423109,  2.81273354,  1.33286153,  1.99858223,  4.03701107,
        2.35519057,  1.21179857,  2.098009  ,  2.05794025,  1.10006343,
        3.66823202,  1.76523283,  1.11351949,  1.81320414,  2.75810668,
        2.58122509,  2.11497663,  1.77503711,  0.76635101,  3.05929254,
        2.43152803,  2.15090248,  2.45870967,  2.03636479,  1.48396469,
        2.16925974,  5.35974798,  1.53908383,  3.6498919 ,  2.61885338,
        1.48165862,  1.63719572,  3.5198943 ,  1.56530971,  2.91279676,
        1.4984982 ,  1.46975875,  1.14341005,  1.23312652,  4.87159054])

stockClusters = pd.DataFrame(thisKMeans.predict(cormat), index=stockList)
stockClusters.columns=['cluster']
stockClusters = pd.merge(stockClusters, sp500components, how='left',
left_index=True, right_index=True)
stockClusters.sort("cluster")

outData = stockClusters.sort("cluster")
outData.to_csv("c:/dev/stocks_clustered.50.output.csv")
```

# K Means Clustering

- Sci-kit learn provides a KMeans class
- We compute a correlation matrix and run K Means

```
>> cormat = mergedRetData.corr()

>> thisKMeans = cluster.KMeans(10)
>> thisKMeans.fit(cormat)

>>> thisKMeans.predict(cormat)
array([6, 1, 9, 8, 8, 5, 9, 8, 6, 6, 2, 5, 6, 0, 0, 3, 8, 7, 2, 8, 7, 9, 7,
       6, 3, 6, 9, 8, 3, 9, 5, 2, 8, 4, 4, 1, 1, 5, 5, 7, 8, 1, 1, 2, 5, 7,
       3, 5, 7, 2, 8, 8, 5, 1, 8, 4, 3, 1, 7, 5, 1, 8, 9, 2, 8, 4, 7, 7, 6,
       8, 8, 4, 1, 7, 8, 9, 1, 8, 3, 4, 5, 8, 1, 8, 4, 8, 7, 5, 1, 8, 3, 4,
       7, 4, 5, 5, 4, 2, 8, 2, 6, 1, 5, 2, 6, 9, 9, 2, 4, 0, 1, 1, 6, 8, 5,
       1, 1, 3, 2, 4, 4, 1, 1, 2, 0, 0, 3, 4, 9, 6, 1, 0, 1, 0, 2, 9, 1, 1,
       4, 7, 4, 8, 4, 2, 1, 0, 3, 0, 5, 2, 5, 4, 3, 5, 0, 9, 2, 5, 7, 9, 1,
       5, 1, 2, 1, 8, 4, 8, 0, 5, 5, 1, 8, 8, 9, 1, 2, 9, 1, 5, 1, 4, 2, 2,
       7, 7, 7, 7, 5, 4, 2, 1, 1, 1, 4, 6, 2, 3, 6, 7, 8, 3, 6, 5, 5, 6, 9,
       1, 5, 1, 2, 3, 1, 1, 6, 1, 2, 9, 1, 8, 9, 4, 7, 5, 8, 7, 6, 2, 2, 8,
       8, 2, 5, 1, 5, 7, 3, 8, 2, 6, 2, 1, 8, 7, 5, 6, 6, 5, 1, 2, 5, 7, 1,
       5, 2, 8, 6, 8, 5, 8, 8, 7, 5, 8, 2, 1, 3, 3, 1, 2, 8, 4, 1, 6, 9, 7,
       6, 4, 1, 8, 4, 4, 4, 0, 3, 4, 0, 5, 2, 4, 5, 6, 1, 0, 4, 9, 5, 2, 4,
       5, 6, 2, 4, 5, 5, 5, 1, 8, 1, 3, 5, 2, 8, 2, 2, 7, 2, 7, 1, 5, 9, 7,
       5, 7, 5, 0, 0, 1, 0, 3, 7, 7, 5, 9, 1, 4, 6, 5, 3, 4, 3, 7, 5, 9, 5,
       1, 1, 2, 4, 2, 2, 5, 0, 1, 2, 2, 1, 8, 4, 2, 1, 9, 0, 7, 5, 8, 0, 7,
       8, 7, 3, 1, 4, 8, 2, 9, 2, 5, 3, 0, 0, 6, 5, 3, 5, 2, 7, 5, 1, 7, 8,
       9, 2, 5, 6, 5, 3, 8, 7, 1, 1, 2, 7, 1, 8, 5, 4, 5, 7, 9, 7, 2, 2, 8,
       9, 0, 7, 2, 5, 8, 5, 3, 2, 2, 1, 4, 3, 7, 6, 4, 5, 2, 9, 2, 7, 8])
```

# K Means Clustering

| ticker | cluster | name | Sector | SubIndustry |
|---|---|---|---|---|
| LEN | 1 | Lennar Corp. | Consumer Discretionary | Homebuilding |
| PHM | 1 | Pulte Homes Inc. | Consumer Discretionary | Homebuilding |
| AKAM | 2 | Akamai Technologies Inc | Information Technology | Internet Software & Services |
| BAC | 3 | Bank of America Corp | Financials | Banks |
| JPM | 3 | JPMorgan Chase & Co. | Financials | Banks |
| PRU | 3 | Prudential Financial | Financials | Diversified Financial Services |
| ALL | 4 | Allstate Corp | Financials | Property & Casualty Insurance |
| DIS | 4 | The Walt Disney Company | Consumer Discretionary | Broadcasting & Cable TV |
| DRI | 4 | Darden Restaurants | Consumer Discretionary | Restaurants |
| FDX | 4 | FedEx Corporation | Industrials | Air Freight & Logistics |
| GPS | 4 | Gap (The) | Consumer Discretionary | Apparel Retail |
| HD | 4 | Home Depot | Consumer Discretionary | Home Improvement Retail |
| LOW | 4 | Lowe's Cos. | Consumer Discretionary | Home Improvement Retail |
| SBUX | 4 | Starbucks Corp. | Consumer Discretionary | Restaurants |
| TGT | 4 | Target Corp. | Consumer Discretionary | General Merchandise Stores |
| TRV | 4 | The Travelers Companies Inc. | Financials | Property & Casualty Insurance |
| URBN | 4 | Urban Outfitters | Consumer Discretionary | Apparel Retail |
| AFL | 5 | AFLAC Inc | Financials | Life & Health Insurance |
| AIV | 5 | Apartment Investment & Mgmt | Financials | REITs |
| AVB | 5 | AvalonBay Communities, Inc. | Financials | REITs |
| AXP | 5 | American Express Co | Financials | Consumer Finance |
| F | 5 | Ford Motor | Consumer Discretionary | Automobile Manufacturers |
| M | 5 | Macy's Inc. | Consumer Discretionary | Department Stores |
| SPG | 5 | Simon Property Group Inc | Financials | REITs |
| USB | 5 | U.S. Bancorp | Financials | Banks |
| VNO | 5 | Vornado Realty Trust | Financials | REITs |
| JNJ | 6 | Johnson & Johnson | Health Care | Health Care Equipment & Services |
| KO | 6 | The Coca Cola Company | Consumer Staples | Soft Drinks |
| MCD | 6 | McDonald's Corp. | Consumer Discretionary | Restaurants |
| MO | 6 | Altria Group Inc | Consumer Staples | Tobacco |
| PEP | 6 | PepsiCo Inc. | Consumer Staples | Soft Drinks |
| COP | 7 | ConocoPhillips | Energy | Integrated Oil & Gas |
| HES | 7 | Hess Corporation | Energy | Integrated Oil & Gas |
| OXY | 7 | Occidental Petroleum | Energy | Oil & Gas Exploration & Production |
| SLB | 7 | Schlumberger Ltd. | Energy | Oil & Gas Equipment & Services |
| AMGN | 8 | Amgen Inc | Health Care | Biotechnology |
| CVX | 8 | Chevron Corp. | Energy | Integrated Oil & Gas |
| LLY | 8 | Lilly (Eli) & Co. | Health Care | Pharmaceuticals |
| MMM | 8 | 3M Company | Industrials | Industrial Conglomerates |
| MRK | 8 | Merck & Co. | Health Care | Pharmaceuticals |
| PFE | 8 | Pfizer Inc. | Health Care | Pharmaceuticals |
| SHW | 8 | Sherwin-Williams | Materials | |
| UPS | 8 | United Parcel Service | Industrials | Air Freight & Logistics |
| XOM | 8 | Exxon Mobil Corp. | Energy | Integrated Oil & Gas |
| JDSU | 9 | JDS Uniphase Corp. | Information Technology | Telecommunications Equipment |
| NVDA | 9 | Nvidia Corporation | Information Technology | Semiconductors |
| SNDK | 9 | SanDisk Corporation | Information Technology | Computer Storage & Peripherals |
| AMZN | 10 | Amazon.com Inc | Consumer Discretionary | Internet Retail |
| CSCO | 10 | Cisco Systems | Information Technology | Networking Equipment |
| ORCL | 10 | Oracle Corp. | Information Technology | Application Software |
| QCOM | 10 | QUALCOMM Inc. | Information Technology | Semiconductors |
| SYMC | 10 | Symantec Corp. | Information Technology | Application Software |
| YHOO | 10 | Yahoo Inc. | Information Technology | Internet Software & Services |

# Why use a data-driven approach?

- Eliminate human bias
- We're not ignoring the fundamentals
- Fundamentals are usually reflected in price any way!
- Find patterns before they become humanly noticeable
- Find patterns that may not be noticeable at all
- Requires you to think less!

# Other examples – data

| ACTION | RESOURCE | MGR_ID | ROLE_ROLLUP_1 | ROLE_ROLLUP_2 | ROLE_DEPTNAME | ROLE_TITLE | ROLE_FAMILY_DESC | ROLE_FAMILY | ROLE_CODE |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 77175 | 3889 | 117961 | 118386 | 121668 | 117905 | 240983 | 290919 | 117908 |
| 0 | 16461 | 5919 | 117961 | 118446 | 16232 | 121594 | 302830 | 4673 | 121596 |
| 0 | 44724 | 50056 | 117893 | 117894 | 117878 | 130479 | 226503 | 119784 | 130481 |
| 1 | 73815 | 92887 | 118315 | 118316 | 140453 | 129229 | 182258 | 119788 | 129231 |
| 1 | 4675 | 3023 | 117961 | 118343 | 122012 | 118321 | 117906 | 290919 | 118322 |
| 1 | 87802 | 4193 | 117961 | 118343 | 118514 | 119849 | 133986 | 118638 | 119851 |
| 1 | 41264 | 7551 | 117961 | 118052 | 118867 | 117905 | 117906 | 290919 | 117908 |
| 1 | 42085 | 9959 | 118742 | 118743 | 117920 | 117899 | 117897 | 19721 | 117900 |
| 1 | 4675 | 25557 | 117961 | 118300 | 121951 | 118321 | 117906 | 290919 | 118322 |
| 1 | 40069 | 5730 | 117961 | 118446 | 118684 | 124305 | 132003 | 118762 | 124307 |
| 1 | 79092 | 1331 | 117961 | 118225 | 118403 | 118784 | 117906 | 290919 | 118786 |
| 0 | 29304 | 70479 | 118953 | 118954 | 117941 | 118568 | 135898 | 19721 | 118570 |
| 1 | 20097 | 1755 | 117961 | 117962 | 119223 | 125793 | 146749 | 118643 | 125795 |
| 1 | 5764 | 14811 | 117961 | 118343 | 118344 | 117905 | 117906 | 290919 | 117908 |
| 1 | 4675 | 7454 | 117961 | 118413 | 122007 | 118321 | 117906 | 290919 | 118322 |
| 1 | 75078 | 18686 | 117961 | 118386 | 121883 | 118321 | 117906 | 290919 | 118322 |
| 0 | 20897 | 23345 | 117961 | 118327 | 123757 | 119997 | 278014 | 118131 | 119998 |
| 1 | 6977 | 2612 | 117961 | 118327 | 123901 | 118321 | 117906 | 290919 | 118322 |
| 1 | 17308 | 17598 | 117961 | 118300 | 118631 | 119928 | 219829 | 118331 | 119929 |
| 1 | 13878 | 17881 | 117961 | 118300 | 121030 | 118801 | 311498 | 19793 | 118803 |
| 1 | 29108 | 111936 | 117961 | 118300 | 118783 | 117905 | 117906 | 290919 | 117908 |
| 1 | 35068 | 1923 | 117902 | 117903 | 118783 | 117905 | 117906 | 290919 | 117908 |
| 0 | 31183 | 46663 | 118256 | 118257 | 118623 | 259173 | 193644 | 292795 | 118943 |
| 1 | 34924 | 15385 | 117902 | 118041 | 117945 | 259173 | 130913 | 292795 | 118943 |
| 1 | 88481 | 1483 | 117961 | 117962 | 118840 | 118841 | 118842 | 118643 | 118843 |
| 1 | 51987 | 110249 | 117961 | 117962 | 117904 | 179731 | 131272 | 117887 | 117973 |
| 0 | 18072 | 15484 | 118256 | 118257 | 118623 | 259173 | 193644 | 292795 | 118943 |
| 1 | 28360 | 36051 | 117961 | 118386 | 118635 | 118685 | 262400 | 308574 | 118687 |

# Other examples – data

| ACTION | RESOURCE=16461 | RESOURCE=44724 | RESOURCE=73815 | RESOURCE=4675 | MGR_ID=3889 | MGR_ID=5919 | MGR_ID=50056 | MGR_ID=92887 | MGR_ID=3023 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

# Classification – steps

- Read the data: pandas.read_csv()
- One Hot Encoding: sklearn.preprocessing.OneHotEncoder()
- Pick a technique, e.g. Logistic Regression
- Fit a model: sklearn.linear_model.LogisticRegression()
- Cross Validation: sklearn.cross_validation.ShuffleSplit()
  - Train on part of the data
  - Test on the remaining data  i.e. data you haven't "seen"
- If performance is acceptable, you're good to go!

# Classification – ROC curve