ARCADA

ARCADA UNIVERSITY OF APPLIED SCIENCES

Introduction to Analytics

Magnus.Westerlund @arcada.fi

Researcher & Programme Director

01.04.2017

Intro to Analytics - Course Schedule

- Week 1
  - 6.9: Intro to Analytics, Machine Learning, and AI
  - 7.9: Feature engineering, Pandas
- Week 2
  - 20.9: Time series processing, linear modeling and setting targets/labels
  - 21.9: Time series data visualization and regression
- Week 3
  - 4.10: External Presentation, understanding model output, and going from output to decision
  - 5.10: Open discussion, creating decisions, finalizing project

Todays Agenda

- Assignment 1 - Walkthrough
- Brief repetition
  - Constructing software
  - A few notes on features
- Course project
- Coding features
- Rolling calculations in dataframes

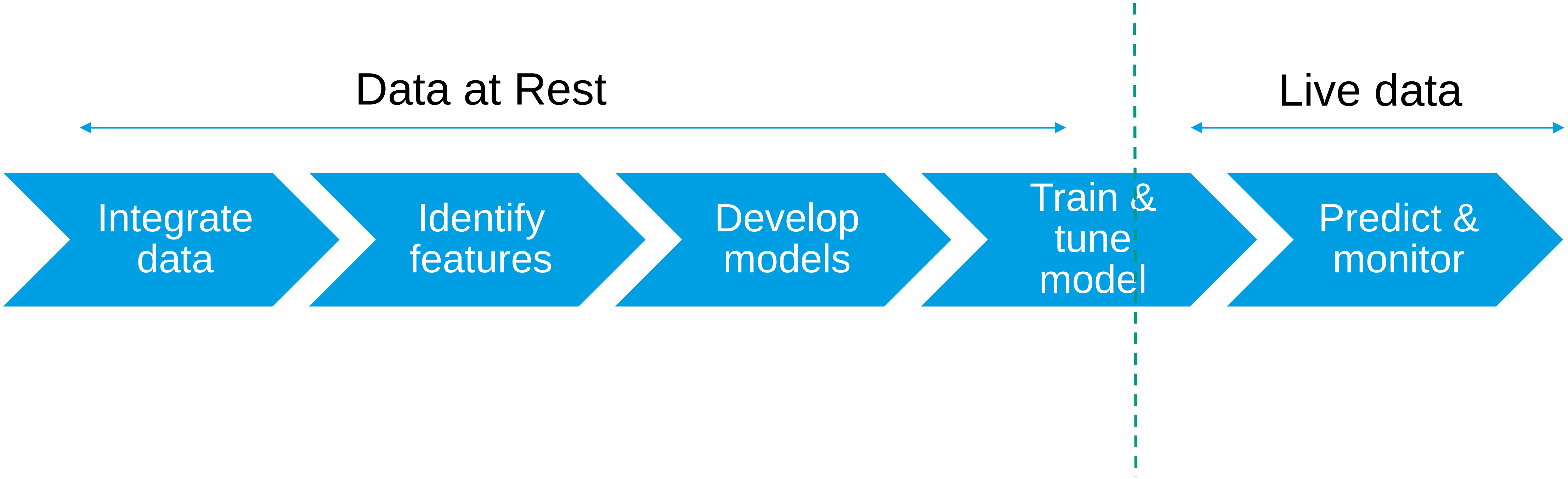
Assignment

Assignment 1

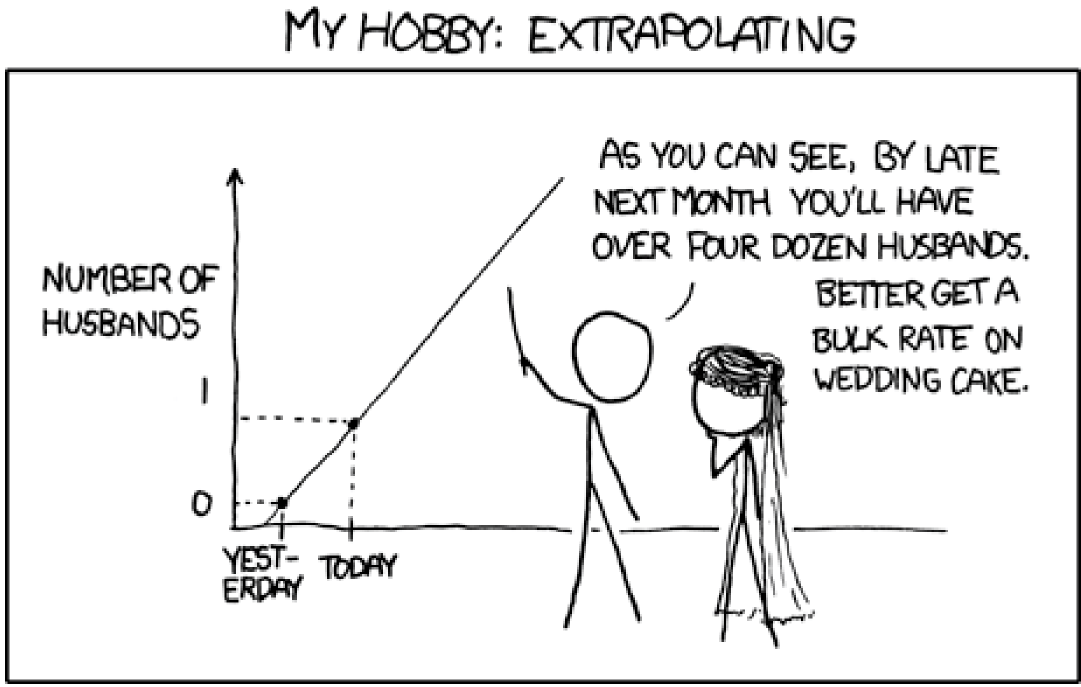
- In this exercise you will load the dataset provided below into a Python Pandas dataframe calculate some basic statistics. You should calculate the median and standard deviation for each place of measurement. You need to do this by implementing your own algorithm. Then you should visualize each time series for respective places in the same figure. The x-axis should have the correct time index indicated in the figure. The exercise will be 10% of the of the final grade.
- Delivery: a .ipynb file. It would be helpful if you describe the problems you encountered during the work with the assignment directly in the end of the file.
- You can find the data set you should use here:
- https://data.melbourne.vic.gov.au/Environment/Sensor-readings-with-temperature-light-humidity-ev/ez6b-syvw

Brief reflection on last week

The analytics process, for prediction

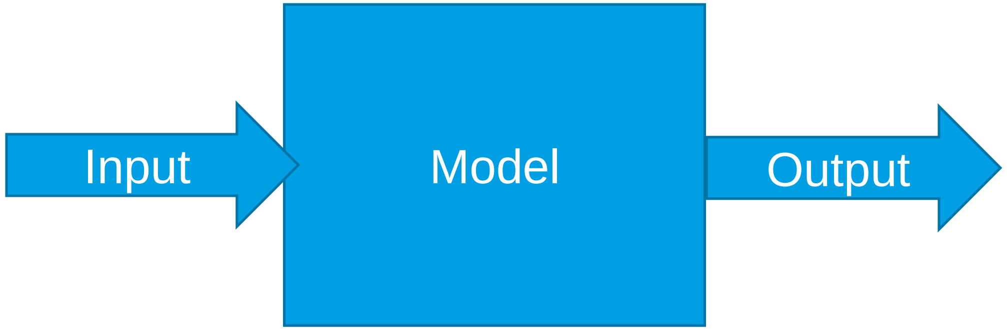


General idea of modeling



- Assume a distribution p(X,Y).
  - X : input
  - Y : output

- Given multiple features
  - X_i : one input feature
  - $X_{i,t}$: one input feature, at a time or index
- Given multiple outputs
  - Y_i : one output type
  - $Y_{i,t}$: one output, at a time or index



Feature Engineering

- Human concepts to model inputs
  - image → pixels, contours, textures, etc.
  - signal → samples, spectrograms, etc.
  - time series → ticks, trends, reversals, etc.
  - biological data → dna, marker sequences, genes, etc.
  - text data → words, grammatical classes and relations, etc.

Feature Relevance

- Some features hold more information than others, how to determine which to use.
  - Strongly relevant feature
    - Feature Xi brings information that no other feature contains.
  - Weakly relevant feature
    - Feature Xi brings information that also exists in other features.
    - Feature Xi brings information in conjunction with other features.
  - Irrelevant feature
    - Feature Xi is neither strongly relevant nor weakly relevant.

Feature Selection

- The selection is often a heuristic process (a discovery that employs a practical method).
- Can be a difficult and time consuming process, as you may have to test many combinations and variations.
- Feature relevance may in semi-chaotic processes change over time, e.g. price movements on a financial market.
- Feature selection should always be part of a validation and verification process, i.e. once the system is in use, you may need to maintain a certain measure for continued feature relevance.

Some techniques for Feature Selection

- Forward selection
  - Start with empty set of features.
  - Incrementally add features Xt.
  - Will find all strongly relevant features. May not find some weakly relevant features.
- Backward selection
  - Start with full set of features .
  - Incrementally remove features Xi.
  - Will keep all strongly relevant features. May eliminate some weakly relevant features (e.g. redundant).
- You may perform an exhaustive search through all the subsets of features, but finding all relevant features is NP-hard.

Why we only want relevant features as inputs

- A model will find it difficult to learn from noisy and/or irrelevant data.
- The more features we use, it will also make the learning process computationally more complex.
- We consider each input as its own dimension.

- However, we can also try to reduce dimensions.
  – This works for linear problems, but not really for non-linear problems.

Beneficial reasons

- **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.
- **Reduces Training Time**: Less data means that algorithms train faster

Course project

Project – Stock forecasting

- We will replicate parts of the following paper: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4873195/
- We will focus on features and determining a decision, by forecasting 1 step ahead
- Analyze data for a single stock, choose the MU symbol
- You get data from IEX in OHLC + Vol format
- Take five years of data and if you use a learning model, segment it as 80% training and 20% testing
- Implement some of the input features from the paper

Project – Stock forecasting (cont.)

- Grading (1=pass; 5=best): (grades 3-5 can be done in any order)
  - 1) Implement 2 features and visualize the price and the features in the same graph.
  - 2) Do a regression based on the features, forecast 1-day ahead.
  - 3) Plot (as a line) the regression and expected output, make the plot zoomable.
  - 4) Add 2 more features from the “Type 2” category of features presented in the paper.
  - 5) Design a decision for when to invest and when to sell based on your regression. The model can be naïve, meaning you can create a rule (if .. X .. then .. Y).
- Submission **deadline: 27.9, presentation 27.9 !!**

ARCADA

Constructing Analytics Software

Coding Features

Getting stock market data

```
import datetime as dt
import numpy as np
import pandas as pd
# https://stackoverflow.com/a/50970152
pd.core.common.is_list_like = pd.api.types.is_list_like
from pandas_datareader.data import DataReader
```

https://anaconda.org/anaconda/pandas-datareader

```
# Define timeframe of stocks we retreive
end = dt.datetime.now()
start = end - dt.timedelta(days=5*365)
```

```
# Use DataReader to get Apples stock data from IEX https://iextrading.com/developer/
# df = DataReader('AAPL','iex', start, end)
df = DataReader('AAPL','iex', start, end)
df.head(10)
```

5y

	open	high	low	close	volume
date					
2013-09-13	61.3916	61.7172	60.7847	60.8108	74578903
2013-09-16	60.3007	60.3805	58.4982	58.8776	136823442
2013-09-17	58.5950	60.1320	58.5349	59.5577	99756489
2013-09-18	60.5859	61.0005	60.2562	60.7821	113743049

Adding on Col level

```
# With pandas you can directly do basic operations on DataFrames, it will go
# on a row by row basis
df["E"] = df["A"] + df["B"]
```

```
df.tail() # .tail() to show the n last columns, 5 by default
```

	A	B	C	D	E
2013-01-02	0.000278	0.623113	0.137503	-0.460380	0.623391
2013-01-03	1.616602	-1.216541	-2.172673	-0.665277	0.400060
2013-01-04	0.238536	2.386232	-1.739818	0.133458	2.624768
2013-01-05	-1.000255	-1.203721	-1.335710	0.304096	-2.203976
2013-01-06	-0.661563	-0.536688	-0.076919	-0.702685	-1.198251

Working with a data frame

```
# Delete the e column with .drop()
df = df.drop("e", axis=1)
# instead of reassigning our DataFrame with df = we could just as well use
# inplace=True within the parenthesis
```

```
# Here we take the DataFrame, but only the rows where the a column is larger than 0.5
df[ df["a"] > 0.5 ]
```

	a	b	c	d
2013-01-03	1.616602	-1.216541	-2.172673	-0.665277

Filter data in columns

```
# We can "overwrite" these operations by assigning it to df again,
# or we can save it to another variable. You can also use multiple operations
df_filtered = df[ (df["a"] > 0) & (df["c"] < 1.2) ]

df_filtered
```

	a	b	c	d
2013-01-02	0.000278	0.623113	0.137503	-0.460380
2013-01-03	1.616602	-1.216541	-2.172673	-0.665277
2013-01-04	0.238536	2.386232	-1.739818	0.133458

Filter on column and create new

- Note that we here use numpy's where function.

```
# Create a new column called df.elderly where the value is yes
# if df.age is greater than 50 and no if not
df['elderly'] = np.where(df['age']>=50, 'yes', 'no')
```

```
# View the dataframe
df
```

	name	age	preTestScore	postTestScore	elderly
0	Jason	42	4	25	no
1	Molly	52	24	94	yes
2	Tina	36	31	57	no
3	Jake	24	2	62	no
4	Amy	73	3	70	yes

Working with values

```
# You can set a certain cell in a DataFrame to any value using .set_value()
df.set_value("2013-01-02", "d", None)
# Note, here we do not have to reassign the df for the operation to save
```

	a	b	c	d
2013-01-01	-0.167610	2.406043	-1.589572	0.445650
2013-01-02	0.000278	0.623113	0.137503	NaN
2013-01-03	1.616602	-1.216541	-2.172673	-0.665277
2013-01-04	0.238536	2.386232	-1.739818	0.133458
2013-01-05	-1.000255	-1.203721	-1.335710	0.304096
2013-01-06	-0.661563	-0.536688	-0.076919	-0.702685

```
# Now we have NaN data in our DataFrame, we can choose to use dropna() which
# deletes all rows with NaN / None data, or use fillna() to fill our data with
# a value we choose
df.dropna(0, inplace=True)
```

Date to week day

```
# Using .weekday_name on the Date values, we can map a column to the actual date names
df["Day"] = df.index.weekday_name
```

```
df.head()
```

	a	b	c	d	Day
2013-01-01	-0.167610	2.406043	-1.589572	0.445650	Tuesday
2013-01-03	1.616602	-1.216541	-2.172673	-0.665277	Thursday
2013-01-04	0.238536	2.386232	-1.739818	0.133458	Friday
2013-01-05	-1.000255	-1.203721	-1.335710	0.304096	Saturday
2013-01-06	-0.661563	-0.536688	-0.076919	-0.702685	Sunday

Week day as a feature

```
# Convert categorical variable into dummy/indicator variables using .get_dummies()
df = pd.get_dummies(df)
```

```
df.head()
```

	a	b	c	d	Day_Friday	Day_Saturday	Day_Sunday	Day_Thursday	Day_Tuesday
2013-01-01	-0.167610	2.406043	-1.589572	0.445650	0	0	0	0	1
2013-01-03	1.616602	-1.216541	-2.172673	-0.665277	0	0	0	1	0
2013-01-04	0.238536	2.386232	-1.739818	0.133458	1	0	0	0	0
2013-01-05	-1.000255	-1.203721	-1.335710	0.304096	0	1	0	0	0
2013-01-06	-0.661563	-0.536688	-0.076919	-0.702685	0	0	1	0	0

Ploting series

```
import matplotlib
import matplotlib.pyplot as plt
```

```
df["Adj Close"].plot(figsize=(12,8))  # Select a column using df["Column name"] or df.column_name,
                                      # .plot() automatically creates a plot of the data
plt.show()                            # plt.show() is used to actually show the plot
```

Resetting index

```
df.reset_index(inplace=True) # using reset_index() we make the Dates a column, instead of using them as an index
```

```
df.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close
0	2012-03-26	599.790016	607.150024	595.259979	606.979980	148935500	78.640054
1	2012-03-27	606.180016	616.280006	606.060013	614.480019	151782400	79.611755
2	2012-03-28	618.379974	621.450005	610.309990	617.620010	163865100	80.018571
3	2012-03-29	612.780006	616.560013	607.230026	609.859993	152059600	79.013187
4	2012-03-30	608.769981	610.559982	597.939987	599.550011	182759500	77.677430

Calculate change between days

- Simple return, up/down changes are different

$$(p_t - p_{t-1})/p_{t-1},$$

- Log return, up/down remains same

$$\log(p_t/p_{t-1})$$

- Note shift function, learn using shift!

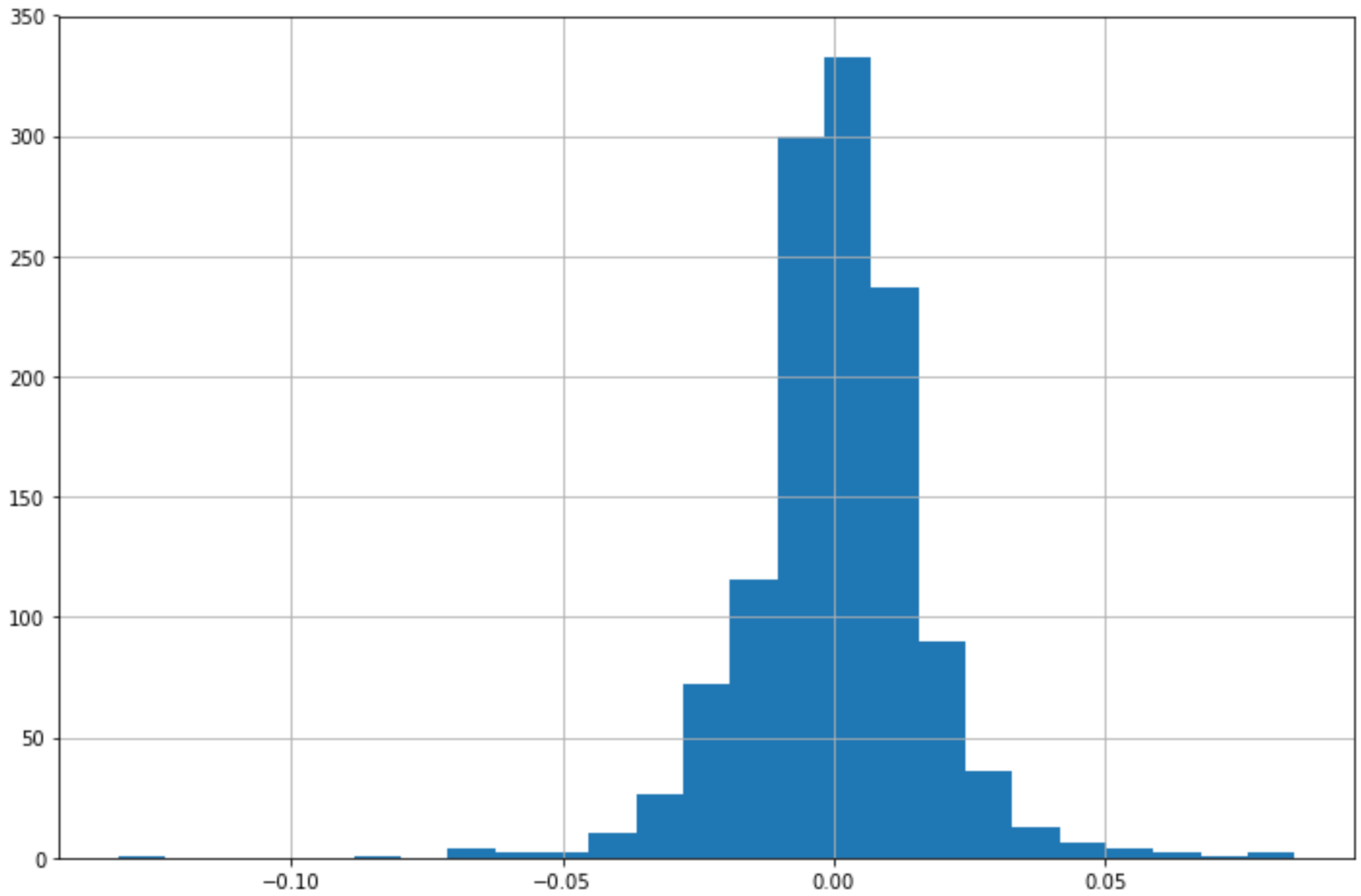
1	
3	0.4771
5	0.2218
7	0.1461
5	-0.1461
3	-0.2218
1	-0.4771
1	0

```
import numpy as np
#df["DPC"] = np.log(df["Adj Close"].iloc[1:] / df["Adj Close"].iloc[:-1].values)
df['Log_Ret'] = np.log(df["Adj Close"] / df["Adj Close"].shift(1))
```

Distribution of returns, histogram

```
print("Max value:", df["Log_Ret"].max())
print("Min value:", df["Log_Ret"].min())
df["Log_Ret"].hist(bins=25, figsize=(12,8))
plt.show()
```

```
Max value: 0.0850223244157
Min value: -0.13188468781
```



Describing statistics

```
# Using the describe() function we can get various data from our panadas data structures
df["Adj Close"].describe()
```

```
count    1258.000000
mean       91.868239
std        22.117804
min        51.343714
25%        72.446593
50%        93.131426
75%       110.185164
max       141.460007
Name: Adj Close, dtype: float64
```

These are quantiles

Exercise – Create features based on quantiles

- Implement the log-return for the close prices
  - Calculate the quantiles for the log-return column
  - Filter each category of quantiles and set 1/0 in respective column

0	25	50	75	100
	Crash	Down	Up	Jump
22	1	0	0	0
70	0	0	1	0

Normalizing column values

We can create a list with the normalized values within the DataDrame

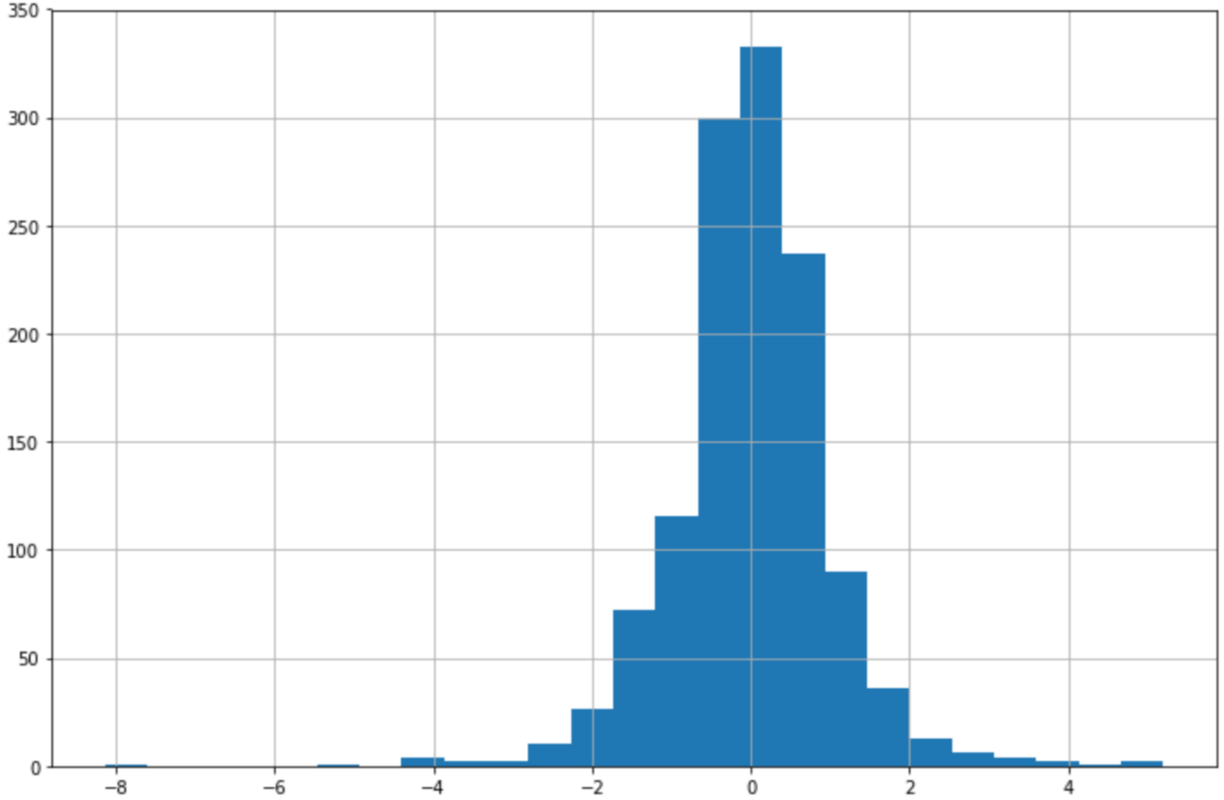
```
norm = (df["Log_Ret"] - df["Log_Ret"].mean()) / (df["Log_Ret"].std())
```

and plot that

```
print("Max value:", norm.max())
print("Min value:", norm.min())
```

```
Max value: 5.19943500646
Min value: -8.13773297324
```

```
norm.hist(bins=25, figsize=(12,8))
plt.show()
```



Resampling based on date index

```
# Now that the Dates are our indexes, we can use resample
examples.resample("M").mean()
# Here we make each row the mean values of the data that month
# Other options than M can be found here: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2012-03-31	140.822000	141.788003	139.813999	140.739999	26141720	140.739999
2012-04-30	138.594500	139.209999	138.028500	138.796001	22696940	138.796001
2012-05-31	127.136364	128.069999	126.835454	127.646817	32023631	127.280119
2012-06-30	117.192856	118.009524	116.818571	117.562857	25863685	117.055354
2012-07-31	111.124761	112.342857	110.635715	111.673810	31770852	111.191729