WHAT’S THIS TALK ABOUT?

• A talk of two halves!

• In the first half we talk about quantitative trading and backtesting from a theoretical point of view.

• In the second half we show how to use modern Python tools to implement a backtesting environment for a simple trading strategy.
QUANTITATIVE TRADING

• Creates a set of rules for trade order generation and risk management of positions with minimal subsequent manager intervention.

• Attempts to identify statistically significant and repeatable market behaviour that can be exploited to generate profits.

• Low-frequency (weekly, daily) through to high-frequency (seconds, milliseconds...)

• Carried out both at the “retail” level and at the large quantitative hedge funds.
TAXONOMY OF TRADING STRATEGIES

- **Forecasting** methods attempt to predict the direction or value of an instrument in subsequent future time periods based on certain historical factors.

- **Mean Reversion** trades on the deviation of a *spread* between two or more instruments. Utilises *cointegration* tests to ascertain mean reverting behaviour.

- **Momentum** or “trend following”. Trades on the basis of the slow diffusion of information (in direct contrast to Efficient Market Hypothesis).

- **High Frequency Trading** or HFT. Specifically referring to exploitation of sub-millisecond market microstructure. FPGAs, Infiniband networks, lots of “dirty tricks”!
WHAT IS BACKTESTING?

• A simulation designed to test the performance of a set of trading and risk management rules on historical data.

• Provides quantified performance of a strategy that can be used for comparison with other strategies.

• Outlines likely capital requirements, trade frequency and risk to a portfolio.

• Arguably a significant improvement beyond guessing!
BACKTESTING PITFALLS

- Market regime shift - Regulatory change, macroeconomic events, “black swans”
- Transaction costs - Unrealistic handling of slippage, market impact and fees
- Liquidity constraints - Ban of short sales (e.g. finance stocks in 2008)
- Optimisation Bias - Over-fitting a model too closely to limited data
- Survivorship Bias - Only using instruments which still exist (incorrect sample)
- Lookahead Bias - Accidental introduction of future information into past data
- Interference - Ignoring strategy rules “just this once” because “I know better”
## Different Types of Backtester

<table>
<thead>
<tr>
<th>Research</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Rapid prototyping</td>
<td>• Extensive development and testing time.</td>
</tr>
<tr>
<td>• Many strategies/parameters can be tested quickly.</td>
<td>• Full Order Management System (OMS).</td>
</tr>
<tr>
<td>• Identifying statistical relationships</td>
<td>• Often event-driven or CEP.</td>
</tr>
<tr>
<td>• Vectorised (pandas, MatLab or R).</td>
<td>• Code-reuse between live implementation and backtesting.</td>
</tr>
<tr>
<td>• Often unrealistic (inflated) performance</td>
<td>• More realistic performance.</td>
</tr>
</tbody>
</table>
COMPONENTS OF A BACKTESTER

- **Data Handler** - An interface to a set of historic or live market data.
- **Strategy** - Encapsulates “signal” generation based on market data.
- **Portfolio** - Generates “orders” and manages of Profit & Loss “PnL”
- **Execution Handler** - Sends orders to broker and receives “fills”.
- **...and many more** depending upon complexity
PYTHON TOOLS FOR BACKTESTING

- **NumPy/SciPy** - Provide vectorised operations, optimisation and linear algebra routines all needed for certain trading strategies.

- **Pandas** - Provides the DataFrame, highly useful for “data wrangling” of time series data. Takes a lot of the work out of pre-processing financial data.

- **Scikit-Learn** - Machine Learning library useful for creating regression and classification models, that are used in forecasting strategies.

- **Statsmodels** - Statistical library (contains packages similar to R). Highly useful for time series analysis for mean-reversion/momentum detection.

- **IbPy** - Pythonic wrapper for Interactive Brokers proprietary market/order API.

MOVING AVERAGE CROSSOVER

• The “Hello World” of quantitative trading!

• A very basic momentum strategy, but useful for calibrating backtesters.

• Strategy Rules:
  - Create two separate simple moving averages (SMA) of a time series with differing lookback periods, e.g. 40 days and 100 days.
  - If the short moving average exceeds the long moving average then “go long”
  - If the long moving average exceeds the short moving average then “exit”
NOW PLEASE SHOW ME SOME PYTHON!
OBTAINING FREE FINANCIAL DATA

Use the Quandl data service (www.quandl.com):

$ pip install Quandl

Easy to obtain daily financial market data (returns a pandas DataFrame):

```python
>>> import datetime
>>> import pandas as pd
>>> import Quandl
>>> ibm = Quandl.get("GOOG/NYSE_IBM")  # Use Google Finance as data source
```

Or with Yahoo Finance:

```python
>>> start_date = datetime.datetime(2009,1,1)
>>> end_date = datetime.datetime(2014,1,1)
>>> amzn = pd.io.data.DataReader("AMZN", "yahoo", start_date, end_date)
```
CLASS HIERARCHIES

• Create Strategy and Portfolio class hierarchies
• Abstract base classes enforce interface for subclasses
• Strategies and Portfolios can be “swapped out” easily and are loosely coupled to data and execution modules.
• Example Strategy abstract base class:

```python
from abc import ABCMeta, abstractmethod

class Strategy(object):
    __metaclass__ = ABCMeta

    @abstractmethod
    def generate_signals(self):
        raise NotImplementedError("Should implement generate_signals()!")
```
class MovingAverageCrossStrategy(Strategy):
    ..

    def generate_signals(self):
        # Create DataFrame and initialise signal series to zero
        signals = pd.DataFrame(index=self.bars.index)
        signals['signal'] = 0

        # Create the short/long simple moving averages
        signals['short_mavg'] = pd.rolling_mean(bars['Adj Close'], self.short_window, min_periods=1)
        signals['long_mavg'] = pd.rolling_mean(bars['Adj Close'], self.long_window, min_periods=1)

        # When the short SMA exceeds the long SMA, set the 'signals' Series to 1 (else 0)
        signals['signal'][self.short_window:] = np.where(signals['short_mavg'][self.short_window:] >
                signals['long_mavg'][self.short_window:], 1, 0)

        # Take the difference of the signals in order to generate actual trading orders
        signals['positions'] = signals['signal'].diff()
        return signals

generate_signals creates a signals DataFrame used by the Portfolio
class MarketOnClosePortfolio(Portfolio):
    ..

    def generate_positions(self):
        # Generate a pandas DataFrame to store quantity held at any “bar” timeframe
        positions = pd.DataFrame(index=signals.index).fillna(0.0)
        positions[self.symbol] = 100 * signals['signal']  # Transact 100 shares on a signal
        return positions

    def backtest_portfolio(self):
        # Create a new DataFrame ‘portfolio’ to store the market value of an open position
        portfolio = self.positions * self.bars['Adj Close']
        pos_diff = self.positions.diff()

        # Create a ‘holdings’ Series that totals all open position market values
        # and a ‘cash’ column that stores remaining cash in account
        portfolio['holdings'] = (self.positions*self.bars['Adj Close']).sum(axis=1)
        portfolio['cash'] = self.initial_capital - (pos_diff*self.bars['Adj Close']).sum(axis=1).cumsum()

        # Sum up the cash and holdings to create full account ‘equity’, then create the percentage returns
        portfolio['total'] = portfolio['cash'] + portfolio['holdings']
        portfolio['returns'] = portfolio['total'].pct_change()
        return portfolio
if __name__ == "__main__":
    # Obtain daily bars of Amazon from Yahoo Finance
    # for the period 1st Jan 2009 to 1st Jan 2014
    symbol = 'AMZN'
bars = DataReader(symbol, "yahoo", datetime.datetime(2009,1,1), datetime.datetime(2014,1,1))

    # Create a Moving Average Cross Strategy instance
    # with short and long moving average windows
    mac = MovingAverageCrossStrategy(symbol, bars, short_window=40, long_window=100)
signals = mac.generate_signals()

    # Create a portfolio of AMZN, with $100,000 initial capital
    portfolio = MarketOnClosePortfolio(symbol, bars, signals, initial_capital=100000.0)
    returns = portfolio.backtest_portfolio()

    # Plot the performance with Matplotlib
    ..
PERFORMANCE

• What next?
  - Calculate a Sharpe Ratio
  - Calculate a Maximum Drawdown
  - Many other metrics, e.g.
    - CAGR
    - Risk/Reward Ratios
    - Distribution of returns
    - Trade-level metrics
  - All very straightforward with pandas
IMPROVEMENTS?

• Multi-symbol portfolios, by adding more columns to a pandas DataFrame.
• Risk management framework (much more important than signal generation!)
• True event-driven backtesting helps mitigate lookahead bias
• Realistic handling of transaction costs - fees, slippage and possible market impact
• Optimisation routines to find best parameters (be careful of curve-fitting!)
• GUI via PyQT or other libraries
WHERE CAN I FIND OUT MORE?

• Visit QuantStart to find the complete code and the slides from the talk:

• Make sure to investigate these fantastic free tools:
  - Pandas - http://pandas.pydata.org/
  - Scikit-Learn - http://scikit-learn.org/
  - Statsmodels - http://statsmodels.sourceforge.net/
  - ZipLine - https://github.com/quantopian/zipline
  - Quandl - http://www.quandl.com/

• Email: mike@quantstart.com, Twitter: @mhallsmoore
THANK YOU!