project_1_starter

July 10, 2020

1 Project 1: Trading with Momentum

1.1 Instructions

Each problem consists of a function to implement and instructions on how to implement the function. The parts of the function that need to be implemented are marked with a `# TODO` comment. After implementing the function, run the cell to test it against the unit tests we've provided. For each problem, we provide one or more unit tests from our `project_tests` package. These unit tests won't tell you if your answer is correct, but will warn you of any major errors. Your code will be checked for the correct solution when you submit it to Udacity.

1.2 Packages

When you implement the functions, you'll only need to you use the packages you've used in the classroom, like Pandas and Numpy. These packages will be imported for you. We recommend you don't add any import statements, otherwise the grader might not be able to run your code.

The other packages that we're importing are `helper`, `project_helper`, and `project_tests`. These are custom packages built to help you solve the problems. The `helper` and `project_helper` module contains utility functions and graph functions. The `project_tests` contains the unit tests for all the problems.

1.2.1 Install Packages

```
In [27]: import sys
         !{sys.executable} -m pip install -r requirements.txt
```

```
Requirement already satisfied: colour==0.1.5 in /opt/conda/lib/python3.6/site-packages (from -r
Requirement already satisfied: cvxpy==1.0.3 in /opt/conda/lib/python3.6/site-packages (from -r r
Requirement already satisfied: cycler==0.10.0 in /opt/conda/lib/python3.6/site-packages/cycler-0
Requirement already satisfied: numpy==1.13.3 in /opt/conda/lib/python3.6/site-packages (from -r
Requirement already satisfied: pandas==0.21.1 in /opt/conda/lib/python3.6/site-packages (from -r
Requirement already satisfied: plotly==2.2.3 in /opt/conda/lib/python3.6/site-packages (from -r
Requirement already satisfied: pyparsing==2.2.0 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: python-dateutil==2.6.1 in /opt/conda/lib/python3.6/site-packages
Requirement already satisfied: pytz==2017.3 in /opt/conda/lib/python3.6/site-packages (from -r r
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: scipy==1.0.0 in /opt/conda/lib/python3.6/site-packages (from -r r
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/python3.6/site-packages (f
```

```
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/site-packages (from -r re
Requirement already satisfied: tqdm==4.19.5 in /opt/conda/lib/python3.6/site-packages (from -r r
Requirement already satisfied: fastcache in /opt/conda/lib/python3.6/site-packages (from cvxpy==
Requirement already satisfied: toolz in /opt/conda/lib/python3.6/site-packages (from cvxpy==1.0.
Requirement already satisfied: multiprocess in /opt/conda/lib/python3.6/site-packages (from cvxp
Requirement already satisfied: ecos>=2 in /opt/conda/lib/python3.6/site-packages (from cvxpy==1.
Requirement already satisfied: scs>=1.1.3 in /opt/conda/lib/python3.6/site-packages (from cvxpy=
Requirement already satisfied: osqp in /opt/conda/lib/python3.6/site-packages (from cvxpy==1.0.3
Requirement already satisfied: nbformat>=4.2 in /opt/conda/lib/python3.6/site-packages (from plo
Requirement already satisfied: decorator>=4.0.6 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/python3.6/site-packages (
Requirement already satisfied: idna<2.7,>=2.5 in /opt/conda/lib/python3.6/site-packages (from re
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/lib/python3.6/site-packages (
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: dill>=0.3.2 in /opt/conda/lib/python3.6/site-packages (from multi
Requirement already satisfied: future in /opt/conda/lib/python3.6/site-packages (from osqp->cvxp
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.6/site-packages (from nbfo
Requirement already satisfied: traitlets>=4.1 in /opt/conda/lib/python3.6/site-packages (from nb
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python3.6/site-packages
```

1.2.2 Load Packages

```
In [28]: import pandas as pd
         import numpy as np
         import helper
         import project_helper
         import project_tests
```

1.3 Market Data

1.3.1 Load Data

The data we use for most of the projects is end of day data. This contains data for many stocks, but we'll be looking at stocks in the S&P 500. We also made things a little easier to run by narrowing down our range of time period instead of using all of the data.

```
In [29]: df = pd.read_csv('../../data/project_1/eod-quotemedia.csv', parse_dates=['date'], index

         close = df.reset_index().pivot(index='date', columns='ticker', values='adj_close')


         print(close)
```

```
ticker                A         AAL          AAP         AAPL        ABBV  \
date
2013-07-01 29.99418563 16.17609308  81.13821681  53.10917319 34.92447839
2013-07-02 29.65013670 15.81983388  80.72207258  54.31224742 35.42807578
```

2013-07-03	29.70518453	16.12794994	81.23729877	54.61204262	35.44486235
2013-07-05	30.43456826	16.21460758	81.82188233	54.17338125	35.85613355
2013-07-08	30.52402098	16.31089385	82.95141667	53.86579916	36.66188936
2013-07-09	30.68916447	16.71529618	82.43619048	54.81320389	36.35973093
2013-07-10	31.17771395	16.53235227	81.99032166	54.60295791	36.85493502
2013-07-11	31.45983407	16.72492481	82.00022986	55.45406479	37.08155384
2013-07-12	31.48047700	16.90786872	81.91105609	55.35309481	38.15724076
2013-07-15	31.72819223	17.10044125	82.61453801	55.47379158	37.79303181
2013-07-16	31.59057266	17.28338516	81.62371841	55.83133953	37.10696377
2013-07-17	31.38414330	17.76481650	80.74188897	55.84626440	37.23401341
2013-07-18	31.58369168	17.73593062	81.74261676	56.03418797	37.53893253
2013-07-19	31.79012104	17.55298671	81.45527908	55.15063572	37.70833205
2013-07-22	32.20297975	17.47595770	81.99032166	55.32713852	38.08948096
2013-07-23	31.97590746	17.37967143	81.94078068	54.37713815	37.53046256
2013-07-24	32.17545584	17.81295964	80.78152175	57.17003539	36.96297418
2013-07-25	32.10664605	18.13070432	81.46518728	56.90917464	37.47117273
2013-07-26	31.37726233	18.38104862	81.88133151	57.23233050	37.93702140
2013-07-29	31.19835688	18.51584940	81.57417743	58.11484449	38.16571074
2013-07-30	30.86118893	18.48696352	81.43546269	58.83253602	37.86079161
2013-07-31	30.77861719	18.63139292	81.73270857	58.73000866	38.52144972
2013-08-01	31.68002538	18.66027880	82.66407899	59.26808264	38.32664028
2013-08-02	31.91397865	18.21736196	82.70371177	60.02912118	38.38593011
2013-08-05	31.61121560	18.45807764	82.64426260	60.92591114	37.86926159
2013-08-06	31.70754930	18.21736196	82.41637409	60.38082896	38.01325118
2013-08-07	31.84516887	18.16921883	81.53454465	60.34578796	37.75068193
2013-08-08	31.54928679	18.27513373	80.90042011	60.22638901	38.16571074
2013-08-09	31.80388300	17.90924591	82.40646589	59.36939001	37.86926159
2013-08-12	31.96214550	18.12107570	81.69307578	61.05595360	38.14877079
...	...	...	...	...	...
2017-05-19	55.50327007	44.83282860	151.06072036	150.70113045	63.42995100
2017-05-22	55.45382835	45.81435227	146.97179877	151.61679784	63.29454092
2017-05-23	58.00502088	46.26049939	140.27992953	151.42972601	63.68142686
2017-05-24	58.56865644	46.36955757	132.66057320	150.97681525	63.76847620
2017-05-25	58.63787484	47.60885514	131.61341035	151.49864721	64.14569000
2017-05-26	58.84553005	48.32269053	133.79749286	151.24265417	63.89421413
2017-05-30	59.69592756	47.54936885	132.63065426	151.30172949	63.85552554
2017-05-31	59.66626253	47.99551598	133.26892494	150.40575387	63.85552554
2017-06-01	60.05190791	48.63003633	136.72954883	150.81928108	64.52290380
2017-06-02	60.13101466	49.09601221	137.42765739	153.05429719	65.04519983
2017-06-05	59.72559259	49.31412858	135.20368297	151.55772252	65.29667569
2017-06-06	59.42894229	49.31412858	130.94522072	152.06970859	65.64487304
2017-06-07	59.95302448	50.42453920	130.26705812	152.97553010	66.49602213
2017-06-08	59.47838401	50.98965889	125.57975776	152.60138644	66.50569428
2017-06-09	58.54887976	49.83959075	128.01316475	146.68400898	67.38585980
2017-06-12	58.33133621	49.05635469	130.59616644	143.17887358	67.25044972
2017-06-13	58.61809816	49.02661155	131.27432905	144.33084223	67.38585980
2017-06-14	58.71698159	48.96712526	130.23713918	142.92288054	68.20799244
2017-06-15	58.54887976	48.68952261	130.79562603	142.06628846	68.28536963

2017-06-16	58.84553005	48.37226243	129.80830106	140.07741950	68.72061632
2017-06-19	59.87391774	49.23481354	129.24981421	144.08469508	69.00110863
2017-06-20	59.65637419	47.61876952	123.24608056	142.77519225	68.88504285
2017-06-21	59.12240366	48.01534474	119.84529455	143.62193844	69.00110863
2017-06-22	59.93324780	48.55072128	120.43399427	143.38563718	70.78078399
2017-06-23	59.10262697	48.21363235	119.47610998	144.02561977	70.25848796
2017-06-26	58.57854478	48.36234805	121.52159207	143.57270901	70.35520945
2017-06-27	58.22256443	48.08474540	121.69121741	141.51491885	70.01668424
2017-06-28	58.73675827	48.82832394	116.45278767	143.58255490	70.52930812
2017-06-29	58.27398382	49.19515602	115.79424221	141.46568942	70.10373358
2017-06-30	58.77942143	49.88916265	116.33305213	141.80044954	70.13275003

ticker	ABC	ABT	ACN	ADBE	ADI	\
date						
2013-07-01	50.86319750	31.42538772	64.69409505	46.23500000	39.91336014	
2013-07-02	50.69676639	31.27288084	64.71204071	46.03000000	39.86057632	
2013-07-03	50.93716689	30.72565028	65.21451912	46.42000000	40.18607651	
2013-07-05	51.37173702	31.32670680	66.07591068	47.00000000	40.65233352	
2013-07-08	52.03746147	31.76628544	66.82065546	46.62500000	40.25645492	
2013-07-09	51.69535307	31.16522893	66.48866080	47.26000000	40.69632003	
2013-07-10	52.28710814	31.16522893	66.71298151	47.25000000	41.10979324	
2013-07-11	53.72026495	31.85599537	67.47567196	47.99000000	42.22705062	
2013-07-12	53.98840397	31.81096287	67.76280247	48.39000000	42.53495620	
2013-07-15	53.84971137	31.95506689	68.41781897	48.12000000	42.57894271	
2013-07-16	53.88669607	32.15320992	67.55642741	47.48500000	42.68451033	
2013-07-17	54.06237335	32.26128793	67.43978064	48.04000000	42.80767257	
2013-07-18	53.91443458	32.15320992	67.69101984	48.19000000	42.52615889	
2013-07-19	54.37674323	32.30632044	67.49361761	48.07000000	42.20945601	
2013-07-22	54.54317435	32.24327493	67.29621538	48.28000000	42.17426681	
2013-07-23	53.28569482	33.03584705	66.62325323	48.07000000	42.56134810	
2013-07-24	52.49052395	32.82869752	66.14769330	47.80000000	42.42938857	
2013-07-25	53.26720248	32.94578204	65.62726924	47.79000000	42.88684829	
2013-07-26	54.06237335	33.12591207	65.60932358	47.64000000	42.71969954	
2013-07-29	53.98840397	33.08988606	64.93636143	47.17000000	42.66691573	
2013-07-30	53.84046520	33.21597708	66.16563896	47.36000000	43.04519972	
2013-07-31	53.87744989	32.99081455	66.22844876	47.28000000	43.44107832	
2013-08-01	54.36749706	33.17995107	67.16162295	47.70000000	43.93372725	
2013-08-02	54.07161953	33.09889256	66.92832940	47.45000000	43.87214613	
2013-08-05	54.58015904	32.82869752	66.60530757	47.63000000	43.61702437	
2013-08-06	54.23805064	32.51346997	65.72597036	47.39000000	43.47626753	
2013-08-07	54.31202002	32.36035945	65.50164964	47.10000000	43.23874037	
2013-08-08	55.11643707	32.35135295	65.48370398	47.51000000	43.19475386	
2013-08-09	55.00548300	32.32433344	65.97720956	47.18000000	43.10678084	
2013-08-12	54.24729681	32.33333995	65.31322023	47.20000000	43.46747023	
...	...	...	...	...	...	
2017-05-19	87.59994036	42.31486674	118.75601310	136.43000000	79.14250576	
2017-05-22	87.91434122	42.87370534	120.45421034	138.86000000	79.97056004	
2017-05-23	87.62941544	42.82468441	119.90450488	139.52000000	79.84391644	

```
2017-05-24 88.39576754 42.67762162 119.70818149 141.12000000 79.99978549
2017-05-25 89.51582062 43.08939743 120.83704093 142.85000000 80.21410542
2017-05-26 89.40774532 43.83451556 120.63090138 141.89000000 80.67197073
2017-05-30 89.43722040 44.11883696 121.49472426 142.41000000 82.61059193
2017-05-31 90.16427240 44.76591323 122.18185609 141.86000000 83.54580618
2017-06-01 91.51030109 45.19729742 122.98678196 141.38000000 80.08746182
2017-06-02 91.94260228 45.58946486 123.42850956 143.48000000 78.82102586
2017-06-05 91.68715157 45.70711509 124.25306776 143.59000000 76.71679380
2017-06-06 90.08567218 45.45220625 124.00766354 143.03000000 78.03193884
2017-06-07 90.32147283 45.64828997 124.22361925 143.62000000 79.18147302
2017-06-08 89.96777186 45.80515695 123.85060483 142.63000000 80.75863709
2017-06-09 90.48849829 46.36399555 123.50703891 138.05000000 76.99695384
2017-06-12 90.74394899 46.24634532 123.97821503 137.25000000 78.11370356
2017-06-13 91.37275071 46.53066671 124.73406004 139.09000000 79.57331502
2017-06-14 92.25700314 46.70714206 124.91075109 138.25000000 79.28922957
2017-06-15 92.77772957 47.17774299 124.69479537 137.52000000 78.12349961
2017-06-16 90.91097444 47.26598066 125.21505233 137.84000000 78.40758506
2017-06-19 92.10962773 47.93266531 125.42119188 140.35000000 78.73085471
2017-06-20 91.61837639 47.81501508 124.20398692 140.91000000 77.58471685
2017-06-21 93.94690777 47.61893136 124.77332472 144.24000000 78.34880876
2017-06-22 94.68378480 48.30522438 119.83579169 143.69000000 79.66147947
2017-06-23 94.14340831 48.11894484 120.48365885 145.41000000 79.88678863
2017-06-26 94.31043377 47.95227368 120.09101209 144.96000000 78.92677572
2017-06-27 93.85848253 47.71697322 119.94376955 142.54000000 76.54633554
2017-06-28 94.69360982 47.53069368 121.46527575 143.81000000 77.58471685
2017-06-29 94.08445815 47.77579833 120.72906307 141.24000000 76.15449354
2017-06-30 92.87597984 47.65814810 121.40637874 141.44000000 76.21326984

ticker          ...              XL        XLNX         XOM        XRAY  \
date            ...
2013-07-01      ...      27.66879066 35.28892781 76.32080247 40.02387348
2013-07-02      ...      27.54228410 35.05903252 76.60816761 39.96552964
2013-07-03      ...      27.33445191 35.28008569 76.65042719 40.00442554
2013-07-05      ...      27.69589920 35.80177117 77.39419581 40.67537968
2013-07-08      ...      27.98505704 35.20050655 77.96892611 40.64620776
2013-07-09      ...      28.31939579 35.50113886 78.89018496 40.80179133
2013-07-10      ...      27.95794850 36.39419366 78.45068533 40.71427558
2013-07-11      ...      28.50011944 37.00430040 78.83102155 41.01571874
2013-07-12      ...      28.92482002 38.00346072 78.94089646 40.83096325
2013-07-15      ...      29.27723113 38.17146113 78.81411772 40.84068723
2013-07-16      ...      29.04229039 38.27314559 78.85637730 40.86013517
2013-07-17      ...      29.18686931 38.48977769 78.99160796 40.93792696
2013-07-18      ...      29.55735279 40.52346684 79.76918424 41.22964615
2013-07-19      ...      29.71096789 40.54999322 80.43688561 41.24909410
2013-07-22      ...      29.84651063 40.59420386 80.14952046 41.49219343
2013-07-23      ...      29.13265221 40.52346684 80.46224136 41.32688588
2013-07-24      ...      28.73506019 40.24051879 80.28475112 41.15185437
2013-07-25      ...      29.02421802 41.05399445 80.26784729 40.91847901
```

```
2013-07-26     ...     29.11457985 40.83294128 80.11571280 40.98654682
2013-07-29     ...     28.92482002 40.38199282 79.47336717 40.93792696
2013-07-30     ...     28.38264907 40.88599404 79.28742502 41.08378656
2013-07-31     ...     28.32843198 41.28388974 79.23671352 41.69639686
2013-08-01     ...     28.97000093 41.69062757 78.37461808 41.71584481
2013-08-02     ...     28.87060292 41.12473146 77.71536862 41.78391262
2013-08-05     ...     28.60855363 41.00978381 77.41109964 41.52136535
2013-08-06     ...     28.34650434 40.50305117 77.30967665 41.40467767
2013-08-07     ...     28.19288924 40.52972131 77.19980174 41.22964615
2013-08-08     ...     28.02120177 40.52083127 77.57168605 41.57970919
2013-08-09     ...     27.86758667 40.27190997 77.20825366 41.39495370
2013-08-12     ...     27.76818866 40.35192038 76.50187303 41.53108932
...            ...             ...         ...         ...         ...
2017-05-19     ...     40.43133035 65.31985198 78.78898294 61.13598414
2017-05-22     ...     40.86051697 66.15263846 79.13518133 62.09836493
2017-05-23     ...     41.31896631 62.67453024 79.41406336 62.65396622
2017-05-24     ...     41.61159355 63.13501218 79.13518133 62.23726525
2017-05-25     ...     42.29439045 64.10496348 78.61588375 62.57459460
2017-05-26     ...     42.23586500 64.51645797 78.42355131 62.22734380
2017-05-30     ...     42.33340741 64.38909063 77.99080333 62.12812929
2017-05-31     ...     42.61628041 65.35904193 77.41380602 63.02105992
2017-06-01     ...     42.54800072 65.30025701 77.60613845 63.55681830
2017-06-02     ...     42.21635651 65.52559923 76.45214383 63.94375491
2017-06-05     ...     41.72864445 65.79013140 77.04837438 63.39807508
2017-06-06     ...     41.48478841 66.24081585 78.09658617 63.05082428
2017-06-07     ...     41.45552569 66.49555053 77.80808751 63.10043153
2017-06-08     ...     41.18240693 66.69150029 77.52920548 62.95160976
2017-06-09     ...     41.52380538 64.07557102 78.98131538 62.84247379
2017-06-12     ...     41.13363573 62.92926493 79.75064513 62.25710816
2017-06-13     ...     41.54331386 63.46812677 79.77949499 62.78294509
2017-06-14     ...     41.97472897 63.56610164 78.92361565 63.22941040
2017-06-15     ...     42.63165652 63.54650667 79.10633146 62.81270944
2017-06-16     ...     43.18073029 63.42893681 80.28917595 63.19964605
2017-06-19     ...     43.23955963 64.56544541 79.58716256 63.47744669
2017-06-20     ...     43.33760851 63.93840619 79.15441457 62.85239525
2017-06-21     ...     43.15131563 64.78099015 78.31776847 63.26909621
2017-06-22     ...     42.42575385 65.23167459 77.97157008 63.32862492
2017-06-23     ...     42.56302230 66.16243594 78.48125104 63.33854637
2017-06-26     ...     42.76892496 65.99587865 78.12543603 63.56673975
2017-06-27     ...     43.14151074 63.78164638 78.00041995 63.92391201
2017-06-28     ...     43.30819385 64.67321778 78.40431807 64.82428373
2017-06-29     ...     43.27877918 62.88027749 77.60613845 64.10898129
2017-06-30     ...     42.94541296 63.01744232 77.63498832 64.41695873

ticker                 XRX         XYL         YUM          ZBH        ZION  \
date
2013-07-01 22.10666494 25.75338607 45.48038323  71.89882693 27.85858718
2013-07-02 22.08273998 25.61367511 45.40266113  72.93417195 28.03893238
```

2013-07-03	22.20236479	25.73475794	46.06329899	72.30145844	28.18131017
2013-07-05	22.58516418	26.06075017	46.41304845	73.16424628	29.39626730
2013-07-08	22.48946433	26.22840332	46.95062632	73.89282298	29.57661249
2013-07-09	22.48946433	26.58233774	47.28094525	73.70108798	28.91218282
2013-07-10	22.96796358	26.98284247	47.08340158	74.00785631	28.32368796
2013-07-11	23.23113816	27.03872686	46.54333492	74.93774876	27.84909533
2013-07-12	23.49431274	27.08529718	45.96422730	75.68549560	28.44708204
2013-07-15	23.54216266	27.06666905	46.69299195	76.27027369	28.77929688
2013-07-16	23.27898808	26.61959399	46.56936223	76.81670381	28.06740794
2013-07-17	23.18328823	26.66616431	46.45874617	78.30261578	28.06740794
2013-07-18	23.49431274	26.94558622	46.97929234	78.81069986	28.77929688
2013-07-19	23.20721320	26.81518933	46.90121042	81.16898043	28.99760949
2013-07-22	23.47038778	26.88970184	46.50429396	81.02518181	29.27287321
2013-07-23	23.42253785	26.74067682	45.82758393	81.00601167	28.38063907
2013-07-24	23.51823770	26.62890805	46.49128030	80.56503316	28.53250871
2013-07-25	23.44646282	26.85244558	46.91422407	79.47217195	28.19080202
2013-07-26	23.18328823	26.70342056	48.15052124	80.98684153	28.04842424
2013-07-29	23.08758839	26.50782523	47.83819353	80.16240164	27.71620940
2013-07-30	23.06366342	23.78811863	47.53237266	79.55844670	27.77316051
2013-07-31	23.20721320	23.21996075	47.44778390	80.02819050	28.13385091
2013-08-01	23.70963740	23.46212640	48.08545297	80.97725310	28.61793539
2013-08-02	23.92496206	23.53663891	48.40428750	80.52668616	28.61793539
2013-08-05	24.09243679	23.54595298	48.68408107	80.46916901	28.39962278
2013-08-06	23.85318717	23.68566393	48.15052124	79.56803513	27.88706274
2013-08-07	23.61393755	23.19201856	48.07243931	79.17498438	27.57383161
2013-08-08	23.87711213	23.26653107	48.21558951	79.84604489	28.01994868
2013-08-09	23.99673694	23.26653107	48.41079433	79.15581519	27.99147312
2013-08-12	24.28383649	23.12682011	48.45634212	78.90656401	28.10537535
...	...	...	...	...	...
2017-05-19	26.81497802	51.24320268	68.90686651	116.46112494	39.54646594
2017-05-22	26.73836380	51.49936943	69.83126290	116.57989212	39.65494752
2017-05-23	26.62344247	52.19890171	69.74275686	116.59968666	40.76934918
2017-05-24	26.89159225	52.01106475	70.75565928	117.87643393	40.13818363
2017-05-25	26.77667091	51.53652928	70.93267135	118.07437924	40.31569894
2017-05-26	26.81497802	50.82472608	70.89333534	118.00509838	39.89163460
2017-05-30	27.12143491	51.20039999	71.20802347	117.21331713	39.31964082
2017-05-31	27.08312780	51.54641544	71.43420556	117.98530385	39.51688006
2017-06-01	27.19804914	51.84300011	72.60445204	121.40975776	39.99025421
2017-06-02	27.12143491	52.39662482	72.76179611	122.66671050	39.54646594
2017-06-05	26.73836380	52.59434793	72.96831019	122.65681324	39.90149656
2017-06-06	26.81497802	52.09015400	73.08631824	122.89434761	39.70425733
2017-06-07	26.96820647	52.85138798	73.02731422	123.07249839	39.90149656
2017-06-08	26.81497802	52.97002185	72.74212810	123.88407418	40.81865898
2017-06-09	26.58513535	53.35558192	71.88656975	123.72571793	41.75554533
2017-06-12	27.04482069	53.40501269	70.71632326	123.71582066	42.33740107
2017-06-13	26.85328513	53.17763111	71.46370757	124.18099215	42.53464030
2017-06-14	26.54682824	53.04911109	71.82756572	124.30965660	42.63325991
2017-06-15	26.61386569	53.34569576	71.40470355	124.54719098	42.27822930

```
2017-06-16 27.29381692 53.43467116 71.57188162 124.62636910 42.49519245
2017-06-19 27.57154347 53.55330503 72.70279208 125.78434918 42.76146542
2017-06-20 27.13101169 53.26660652 72.68312408 125.91301364 42.36698695
2017-06-21 26.70005669 52.93047722 73.16499028 127.43719255 41.74568337
2017-06-22 26.78624769 53.23694805 73.30266633 128.29986262 41.71609749
2017-06-23 27.23635625 53.71148352 73.57801845 128.00239018 41.35120491
2017-06-26 27.95461459 54.05749897 73.49934641 127.97264293 41.75554533
2017-06-27 27.75350225 53.87954816 72.74212810 127.16946735 41.95278457
2017-06-28 28.28980181 54.34419748 72.91914017 127.42727680 42.37684891
2017-06-29 28.12560699 54.27499439 72.23075989 126.81250043 43.38276899
2017-06-30 27.74892476 54.79896064 72.53561401 127.31820357 43.30387330
```

```
ticker             ZTS
date
2013-07-01 29.44789315
2013-07-02 28.57244125
2013-07-03 28.16838652
2013-07-05 29.02459772
2013-07-08 29.76536472
2013-07-09 29.80384612
2013-07-10 29.86156823
2013-07-11 29.74612402
2013-07-12 30.15979909
2013-07-15 30.38106716
2013-07-16 29.97701243
2013-07-17 29.81346647
2013-07-18 29.64992051
2013-07-19 29.09194018
2013-07-22 29.12080123
2013-07-23 28.91877387
2013-07-24 28.76484826
2013-07-25 29.36130999
2013-07-26 29.27472684
2013-07-29 28.94763492
2013-07-30 28.96206545
2013-07-31 28.74031861
2013-08-01 29.07775945
2013-08-02 29.82977047
2013-08-05 30.12864664
2013-08-06 30.01295264
2013-08-07 30.11900548
2013-08-08 30.11900548
2013-08-09 29.80084697
2013-08-12 29.24165929
...                ...
2017-05-19 59.92967369
2017-05-22 59.92967369
2017-05-23 61.04261076
```

```
2017-05-24 61.90712437
2017-05-25 62.18535864
2017-05-26 62.21516946
2017-05-30 61.86737662
2017-05-31 61.88725050
2017-06-01 62.24498027
2017-06-02 62.10586314
2017-06-05 62.27479108
2017-06-06 62.58283617
2017-06-07 62.86107043
2017-06-08 62.18535864
2017-06-09 62.19529558
2017-06-12 61.45996216
2017-06-13 61.67360633
2017-06-14 61.94235833
2017-06-15 62.10161877
2017-06-16 62.26087921
2017-06-19 62.73866054
2017-06-20 62.70879921
2017-06-21 62.70879921
2017-06-22 63.21644187
2017-06-23 62.48981610
2017-06-26 62.43009343
2017-06-27 62.46990854
2017-06-28 62.65903032
2017-06-29 62.21111032
2017-06-30 62.09166499

[1009 rows x 495 columns]
```

1.3.2 View Data

Run the cell below to see what the data looks like for `close`.

```
In [30]: project_helper.print_dataframe(close)
```

1.3.3 Stock Example

Let's see what a single stock looks like from the closing prices. For this example and future display examples in this project, we'll use Apple's stock (AAPL). If we tried to graph all the stocks, it would be too much information.

```
In [31]: apple_ticker = 'AAPL'
         project_helper.plot_stock(close[apple_ticker], '{} Stock'.format(apple_ticker))
```

1.4 Resample Adjusted Prices

The trading signal you'll develop in this project does not need to be based on daily prices, for instance, you can use month-end prices to perform trading once a month. To do this, you must first resample the daily adjusted closing prices into monthly buckets, and select the last observation of each month.

Implement the `resample_prices` to resample `close_prices` at the sampling frequency of `freq`.

```
In [32]: def resample_prices(close_prices, freq='M'):
             """
             Resample close prices for each ticker at specified frequency.

             Parameters
             ----------
             close_prices : DataFrame
                 Close prices for each ticker and date
             freq : str
                 What frequency to sample at
                 For valid freq choices, see http://pandas.pydata.org/pandas-docs/stable/timeser

             Returns
             -------
             prices_resampled : DataFrame
                 Resampled prices for each ticker and date
             """
             # TODO: Implement Function

             return close_prices.resample(freq).last()

         project_tests.test_resample_prices(resample_prices)
```

```
Tests Passed
```

1.4.1 View Data

Let's apply this function to `close` and view the results.

```
In [33]: monthly_close = resample_prices(close)
         project_helper.plot_resampled_prices(
             monthly_close.loc[:, apple_ticker],
             close.loc[:, apple_ticker],
             '{} Stock - Close Vs Monthly Close'.format(apple_ticker))
```

1.5 Compute Log Returns

Compute log returns (R_t) from prices (P_t) as your primary momentum indicator:

$$R_t = log_e(P_t) - log_e(P_{t-1})$$

Implement the `compute_log_returns` function below, such that it accepts a dataframe (like one returned by `resample_prices`), and produces a similar dataframe of log returns. Use Numpy's log function to help you calculate the log returns.

```
In [34]: def compute_log_returns(prices):
             """
             Compute log returns for each ticker.

             Parameters
             ----------
             prices : DataFrame
                 Prices for each ticker and date

             Returns
             -------
             log_returns : DataFrame
                 Log returns for each ticker and date
             """
             # TODO: Implement Function
             result = np.log(prices) - np.log(prices.shift(1))
             return result

         project_tests.test_compute_log_returns(compute_log_returns)
```

```
Tests Passed
```

1.5.1 View Data

Using the same data returned from `resample_prices`, we'll generate the log returns.

```
In [35]: monthly_close_returns = compute_log_returns(monthly_close)
         project_helper.plot_returns(
             monthly_close_returns.loc[:, apple_ticker],
             'Log Returns of {} Stock (Monthly)'.format(apple_ticker))
```

1.6 Shift Returns

Implement the `shift_returns` function to shift the log returns to the previous or future returns in the time series. For example, the parameter `shift_n` is 2 and `returns` is the following:

```
                        Returns
              A         B         C         D
2013-07-08    0.015     0.082     0.096     0.020     ...
2013-07-09    0.037     0.095     0.027     0.063     ...
2013-07-10    0.094     0.001     0.093     0.019     ...
2013-07-11    0.092     0.057     0.069     0.087     ...
...           ...       ...       ...       ...
```

the output of the `shift_returns` function would be:

```
                         Shift Returns
                A          B          C          D
2013-07-08      NaN        NaN        NaN        NaN        ...
2013-07-09      NaN        NaN        NaN        NaN        ...
2013-07-10      0.015      0.082      0.096      0.020      ...
2013-07-11      0.037      0.095      0.027      0.063      ...
...             ...        ...        ...        ...
```

Using the same `returns` data as above, the `shift_returns` function should generate the following with `shift_n` as -2:

```
                         Shift Returns
                A          B          C          D
2013-07-08      0.094      0.001      0.093      0.019      ...
2013-07-09      0.092      0.057      0.069      0.087      ...
...             ...        ...        ...        ...        ...
...             ...        ...        ...        ...        ...
...             NaN        NaN        NaN        NaN        ...
...             NaN        NaN        NaN        NaN        ...
```

Note: The "..." represents data points we're not showing.

```
In [36]: def shift_returns(returns, shift_n):
             """
             Generate shifted returns

             Parameters
             ----------
             returns : DataFrame
                 Returns for each ticker and date
             shift_n : int
                 Number of periods to move, can be positive or negative

             Returns
             -------
             shifted_returns : DataFrame
                 Shifted returns for each ticker and date
             """
             # TODO: Implement Function

             return returns.shift(shift_n)

         project_tests.test_shift_returns(shift_returns)
```

```
Tests Passed
```

1.6.1 View Data

Let's get the previous month's and next month's returns.

```
In [37]: prev_returns = shift_returns(monthly_close_returns, 1)
         lookahead_returns = shift_returns(monthly_close_returns, -1)

         project_helper.plot_shifted_returns(
             prev_returns.loc[:, apple_ticker],
             monthly_close_returns.loc[:, apple_ticker],
             'Previous Returns of {} Stock'.format(apple_ticker))
         project_helper.plot_shifted_returns(
             lookahead_returns.loc[:, apple_ticker],
             monthly_close_returns.loc[:, apple_ticker],
             'Lookahead Returns of {} Stock'.format(apple_ticker))
```

1.7 Generate Trading Signal

A trading signal is a sequence of trading actions, or results that can be used to take trading actions. A common form is to produce a "long" and "short" portfolio of stocks on each date (e.g. end of each month, or whatever frequency you desire to trade at). This signal can be interpreted as rebalancing your portfolio on each of those dates, entering long ("buy") and short ("sell") positions as indicated.

Here's a strategy that we will try: > For each month-end observation period, rank the stocks by *previous* returns, from the highest to the lowest. Select the top performing stocks for the long portfolio, and the bottom performing stocks for the short portfolio.

Implement the `get_top_n` function to get the top performing stock for each month. Get the top performing stocks from `prev_returns` by assigning them a value of 1. For all other stocks, give them a value of 0. For example, using the following `prev_returns`:

```
                                     Previous Returns
               A         B         C         D         E         F         G
2013-07-08     0.015     0.082     0.096     0.020     0.075     0.043     0.074
2013-07-09     0.037     0.095     0.027     0.063     0.024     0.086     0.025
...            ...       ...       ...       ...       ...       ...       ...
```

The function `get_top_n` with `top_n` set to 3 should return the following:

```
                                     Previous Returns
               A         B         C         D         E         F         G
2013-07-08     0         1         1         0         1         0         0
2013-07-09     0         1         0         1         0         1         0
...            ...       ...       ...       ...       ...       ...       ...
```

Note: You may have to use Panda's `DataFrame.iterrows` with `Series.nlargest` in order to implement the function. This is one of those cases where creating a vecorization solution is too difficult.

1.7.1 View Data

We want to get the best performing and worst performing stocks. To get the best performing stocks, we'll use the `get_top_n` function. To get the worst performing stocks, we'll also use the `get_top_n` function. However, we pass in `-1*prev_returns` instead of just `prev_returns`. Multiplying by negative one will flip all the positive returns to negative and negative returns to positive. Thus, it will return the worst performing stocks.

```
In [38]: def get_top_n(prev_returns, top_n):
             """
             Select the top performing stocks

             Parameters
             ----------
             prev_returns : DataFrame
                 Previous shifted returns for each ticker and date
             top_n : int
                 The number of top performing stocks to get

             Returns
             -------
             top_stocks : DataFrame
                 Top stocks for each ticker and date marked with a 1
             """
             # TODO: Implement Function
             # .apply(lambda dfg: dfg.nlargest(3))

             copy_prev_returns = prev_returns.copy()
             for i, row in prev_returns.iterrows():
                 t = row.nlargest(top_n)
                 copy_prev_returns.loc[i] = 0.0

                 copy_prev_returns.loc[i, t.index] = 1.0
                 copy_prev_returns = copy_prev_returns.fillna(0).astype(np.int)


             return copy_prev_returns

         project_tests.test_get_top_n(get_top_n)
```

```
Tests Passed
```

```
In [41]: top_bottom_n = 50
         df_long = get_top_n(prev_returns, top_bottom_n)
         df_short = get_top_n(-1*prev_returns, top_bottom_n)
         project_helper.print_top(df_long, 'Longed Stocks')
         project_helper.print_top(df_short, 'Shorted Stocks')
```

```
10 Most Longed Stocks:
INCY, AMD, AVGO, NFX, SWKS, NFLX, ILMN, UAL, NVDA, MU
10 Most Shorted Stocks:
RRC, FCX, CHK, MRO, GPS, WYNN, DVN, FTI, SPLS, TRIP
```

1.8 Projected Returns

It's now time to check if your trading signal has the potential to become profitable!

We'll start by computing the net returns this portfolio would return. For simplicity, we'll assume every stock gets an equal dollar amount of investment. This makes it easier to compute a portfolio's returns as the simple arithmetic average of the individual stock returns.

Implement the `portfolio_returns` function to compute the expected portfolio returns. Using `df_long` to indicate which stocks to long and `df_short` to indicate which stocks to short, calculate the returns using `lookahead_returns`. To help with calculation, we've provided you with `n_stocks` as the number of stocks we're investing in a single period.

```
In [42]: def portfolio_returns(df_long, df_short, lookahead_returns, n_stocks):
             """
             Compute expected returns for the portfolio, assuming equal investment in each long/

             Parameters
             ----------
             df_long : DataFrame
                 Top stocks for each ticker and date marked with a 1
             df_short : DataFrame
                 Bottom stocks for each ticker and date marked with a 1
             lookahead_returns : DataFrame
                 Lookahead returns for each ticker and date
             n_stocks: int
                 The number number of stocks chosen for each month

             Returns
             -------
             portfolio_returns : DataFrame
                 Expected portfolio returns for each ticker and date
             """
             # TODO: Implement Function
             portfolio_returns =  lookahead_returns*(df_long-df_short)/n_stocks


             return portfolio_returns

         project_tests.test_portfolio_returns(portfolio_returns)
```

```
Tests Passed
```

1.8.1 View Data

Time to see how the portfolio did.

```
In [43]: expected_portfolio_returns = portfolio_returns(df_long, df_short, lookahead_returns, 2*
         project_helper.plot_returns(expected_portfolio_returns.T.sum(), 'Portfolio Returns')
```

1.9 Statistical Tests

1.9.1 Annualized Rate of Return

```
In [44]: expected_portfolio_returns_by_date = expected_portfolio_returns.T.sum().dropna()
         portfolio_ret_mean = expected_portfolio_returns_by_date.mean()
         portfolio_ret_ste = expected_portfolio_returns_by_date.sem()
         portfolio_ret_annual_rate = (np.exp(portfolio_ret_mean * 12) - 1) * 100

         print("""
         Mean:                        {:.6f}
         Standard Error:              {:.6f}
         Annualized Rate of Return:   {:.2f}%
         """.format(portfolio_ret_mean, portfolio_ret_ste, portfolio_ret_annual_rate))
```

```
Mean:                        0.003253
Standard Error:              0.002203
Annualized Rate of Return:   3.98%
```

The annualized rate of return allows you to compare the rate of return from this strategy to other quoted rates of return, which are usually quoted on an annual basis.

1.9.2 T-Test

Our null hypothesis (H_0) is that the actual mean return from the signal is zero. We'll perform a one-sample, one-sided t-test on the observed mean return, to see if we can reject H_0.

We'll need to first compute the t-statistic, and then find its corresponding p-value. The p-value will indicate the probability of observing a t-statistic equally or more extreme than the one we observed if the null hypothesis were true. A small p-value means that the chance of observing the t-statistic we observed under the null hypothesis is small, and thus casts doubt on the null hypothesis. It's good practice to set a desired level of significance or alpha (α) *before* computing the p-value, and then reject the null hypothesis if $p < \alpha$.

For this project, we'll use $\alpha = 0.05$, since it's a common value to use.

Implement the `analyze_alpha` function to perform a t-test on the sample of portfolio returns. We've imported the `scipy.stats` module for you to perform the t-test.

Note: `scipy.stats.ttest_1samp` performs a two-sided test, so divide the p-value by 2 to get 1-sided p-value

```
In [45]: from scipy import stats
```

```
def analyze_alpha(expected_portfolio_returns_by_date):
    """
    Perform a t-test with the null hypothesis being that the expected mean return is ze

    Parameters
    ----------
    expected_portfolio_returns_by_date : Pandas Series
        Expected portfolio returns for each date

    Returns
    -------
    t_value
        T-statistic from t-test
    p_value
        Corresponding p-value
    """
    net_returns = pd.core.series.Series(expected_portfolio_returns_by_date)
    null_hypothesis = 0
    t_value, p_value = stats.ttest_1samp(expected_portfolio_returns_by_date,null_hypoth

    return t_value, p_value/2

project_tests.test_analyze_alpha(analyze_alpha)
```

```
Tests Passed
```

1.9.3 View Data

Let's see what values we get with our portfolio. After you run this, make sure to answer the question below.

```
In [46]: t_value, p_value = analyze_alpha(expected_portfolio_returns_by_date)
         print("""
         Alpha analysis:
          t-value:        {:.3f}
          p-value:        {:.6f}
         """.format(t_value, p_value))
```

```
Alpha analysis:
 t-value:        1.476
 p-value:        0.073359
```

1.9.4 Question: What p-value did you observe? And what does that indicate about your signal?

p-value > 0 it's a signal to buy

1.10 Submission

Now that you're done with the project, it's time to submit it. Click the submit button in the bottom right. One of our reviewers will give you feedback on your project with a pass or not passed grade. You can continue to the next section while you wait for feedback.