# IAQF Code

## quantopian_rep_backtest.py

```
# This code is run in quantopian.com backtesting platform
# Dates: 05/04/2008 to 02/26/2020

import quantopian.algorithm as algo
import quantopian.optimize as opt


def initialize(context):

    # Reference to the AAPL security.
    context.GOOG = sid(46631)
    context.AMZN = sid(16841)
    context.AXP = sid(679)
    context.CVX = sid(23112)
    context.C = sid(1335)
    context.COP =sid(23998)
    context.FB = sid(42950)
    context.GILD = sid(3212)
    context.HON = sid(25090)
    context.MRO = sid(5035)
    context.MRK = sid(5029)
    context.PYPL = sid(49242)
    context.QCOM = sid(6295)
    context.CRM = sid(26401)
    context.V = sid(35920)

    context.democrat_dict = {context.GOOG : 1/15,
                             context.AMZN : 1/15,
                             context.AXP : 1/15,
                             context.CVX : 1/15,
                             context.C : 1/15,
                             context.COP : 1/15,
                             context.FB : 1/15,
                             context.GILD : 1/15,
```

```
                            context.HON : 1/15,
                            context.MRO : 1/15,
                            context.MRK : 1/15,
                            context.PYPL : 1/15,
                            context.QCOM : 1/15,
                            context.CRM : 1/15,
                            context.V : 1/15
                           }

    context.democrat_dict_pypl = {context.GOOG : 1/14,
                            context.AMZN : 1/14,
                            context.AXP : 1/14,
                            context.CVX : 1/14,
                            context.C : 1/14,
                            context.COP : 1/14,
                            context.FB : 1/14,
                            context.GILD : 1/14,
                            context.HON : 1/14,
                            context.MRO : 1/14,
                            context.MRK : 1/14,
                            context.QCOM : 1/14,
                            context.CRM : 1/14,
                            context.V : 1/14
                           }
    context.democrat_dict_face = {
                            #context.GOOG : 1/13,
                            context.AMZN : 1/13,
                            context.AXP : 1/13,
                            context.CVX : 1/13,
                            context.C : 1/13,
                            context.COP : 1/13,
                            context.GILD : 1/13,
                            context.HON : 1/13,
                            context.MRO : 1/13,
                            context.MRK : 1/13,
                            context.QCOM : 1/13,
                            context.CRM : 1/13,
                            context.V : 1/13
                           }
    # Rebalance every day, one hour and a half after market open.
    algo.schedule_function(
        rebalance,
        algo.date_rules.every_day(),
        algo.time_rules.market_open(hours=1, minutes=30)
```

```
    )

def rebalance(context, data):

    # Target a 100% long allocation of our portfolio in AAPL.
    #objective = opt.TargetWeights({context.aapl: 1.0})
    #objective = opt.TargetWeights({context.aapl: 0.5, context.APTV : 0.5})


    # The Optimize API allows you to define portfolio constraints, which can be
    # useful when you have a more complex objective. In this algorithm, we
    # don't have any constraints, so we pass an empty list.
    constraints = []

    # order_optimal_portfolio uses `objective` and `constraints` to find the
    # "best" portfolio weights (as defined by your objective) that meet all of
    # your constraints. Since our objective is just "target 100% in AAPL", and
    # we have no constraints, this will maintain 100% of our portfolio in AAPL.

    flag = 2
    if(flag == 1):
        objective = opt.TargetWeights(context.democrat_dict)
        algo.order_optimal_portfolio(objective, constraints)
    else:
        try:
            objective = opt.TargetWeights(context.democrat_dict)
            algo.order_optimal_portfolio(objective, constraints)
            flag = 1
        except:
            flag = 2
            try:
                objective = \
                opt.TargetWeights(context.context.democrat_dict_pypl)
                algo.order_optimal_portfolio(objective, constraints)
            except:
                objective = opt.TargetWeights(context.democrat_dict_face)
                algo.order_optimal_portfolio(objective, constraints)
```

## quantopian_dem_backtest.py

```
# This code is run in quantopian.com backtesting platform
# Dates: 05/04/2008 to 02/26/2020

import quantopian.algorithm as algo
import quantopian.optimize as opt


def initialize(context):

    # Reference to the AAPL security.
    context.APTV = sid(42173)
    context.KO = sid(4283)
    context.STZ = sid(24873)
    context.CSX = sid(1937)
    context.EL = sid(13841)
    context.EXC = sid(22114)
    context.FSLR = sid(32902)
    context.F = sid(2673)
    context.HD = sid(3496)
    context.MCD = sid(4707)
    context.NEE = sid(2968)
    context.NSC = sid(5442)
    context.SPG = sid(10528)
    context.SPWR = sid(27817)
    context.WMT = sid(8229)

    context.democrat_dict = {context.APTV : 1/15,
                             context.KO : 1/15,
                             context.STZ : 1/15,
                             context.CSX : 1/15,
                             context.EL : 1/15,
                             context.EXC : 1/15,
                             context.FSLR : 1/15,
                             context.F : 1/15,
                             context.HD : 1/15,
                             context.MCD : 1/15,
                             context.NEE : 1/15,
                             context.NSC : 1/15,
                             context.SPG : 1/15,
                             context.SPWR : 1/15,
                             context.WMT : 1/15}
```

```
    context.democrat_dict_APTV = {
                               context.KO : 1/14,
                               context.STZ : 1/14,
                               context.CSX : 1/14,
                               context.EL : 1/14,
                               context.EXC : 1/14,
                               context.FSLR : 1/14,
                               context.F : 1/14,
                               context.HD : 1/14,
                               context.MCD : 1/14,
                               context.NEE : 1/14,
                               context.NSC : 1/14,
                               context.SPG : 1/14,
                               context.SPWR : 1/14,
                               context.WMT : 1/14}
    # Rebalance every day, one hour and a half after market open.
    algo.schedule_function(
        rebalance,
        algo.date_rules.every_day(),
        algo.time_rules.market_open(hours=1, minutes=30)
    )

   # if ifcantrade(sid(46631)):
    #    context.GOOG: 1/ 15

def rebalance(context, data):

    # Target a 100% long allocation of our portfolio in AAPL.
    #objective = opt.TargetWeights({context.aapl: 1.0})
    #objective = opt.TargetWeights({context.aapl: 0.5, context.APTV : 0.5})



    # The Optimize API allows you to define portfolio constraints, which can be
    # useful when you have a more complex objective. In this algorithm, we
    # don't have any constraints, so we pass an empty list.
    constraints = []

    # order_optimal_portfolio uses `objective` and `constraints` to find the
    # "best" portfolio weights (as defined by your objective) that meet all of
    # your constraints. Since our objective is just "target 100% in AAPL", and
    # we have no constraints, this will maintain 100% of our portfolio in AAPL.
```

```
    try:
        objective = opt.TargetWeights(context.democrat_dict)
        algo.order_optimal_portfolio(objective, constraints)
    except:
        objective = opt.TargetWeights(context.democrat_dict_APTV)
        algo.order_optimal_portfolio(objective, constraints)
```

## basic_comparison.py

```
import yfinance as yf
import lxml
import numpy as np
import pandas as pd
import scipy.stats as stats
import math
import pandas as pd
import matplotlib.pyplot as plt
demo=pd.read_pickle('democrat_data.pkl')
demo=demo[demo['Date']>=pd.Timestamp('2010-01-01')]
demo=demo.drop(columns=['Date']).reset_index()
#We need a benchmark for excess return, thus I chose SP500. Now let's grab
#historical data of sp500

tick = yf.Ticker('^GSPC')
data = pd.DataFrame(tick.history(period="max", rounding = False).loc[:,"Close"])
data['sp500'] = np.log(data['Close'] / data['Close'].shift(1))
data=data.reset_index()

data=data[data['Date']>=pd.Timestamp('2010-01-01')]
data=data.drop(columns=['Close'])
demo=demo.merge(data, how='left',on='Date')
demo['excess_ret']=demo['ret']-demo['sp500']


repu=pd.read_pickle('republican_data.pkl')
repu=repu[repu['Date']>=pd.Timestamp('2010-01-01')]
repu=repu.drop(columns=['Date']).reset_index()
repu=repu.merge(data, how='left',on='Date')
repu['excess_ret']=repu['ret']-repu['sp500']

np.correlate(repu.groupby('Date').mean()['excess_ret'], demo.groupby('Date').mean()['exc
#we may need to randomly select several event dates so as to test for
#distribution and anomalies
#dates should exclude year of election, this time maybe between 2012-01-01 &
#2016-12-31
k=np.random.randint(1,1826,14)

import random
random.seed(a=None, version=2)
event_date=list(map(lambda x: pd.tseries.offsets.BDay(x)+
pd.Timestamp('2012-01-01'),list(k)))
```

```
demo_code=set(demo['code'].tolist())

#what we need: mean return, std of excess return , t-stats, studendized range
#from 240, excess ret t-stats from 0
demo_250=''
#demo=demo.rename(columns={"ret": "original_ret"})
#demo=demo.rename(columns={"excess_ret": "ret"})
for i in range(len(event_date)):
    day0=event_date[i]
    day_240=day0-pd.tseries.offsets.BDay(240)
    day_30=day0+pd.tseries.offsets.BDay(30)
    df1=demo[demo['Date']>=day_240]
    df1=df1[df1['Date']<day0]
    frame=df1.groupby(['code']).mean()['ret'].reset_index().rename(columns=
    {'ret':'mean'})
    df1=demo[demo['Date']>=day0]
    df1=df1[df1['Date']<day_30]
    df1=df1.merge(frame, how='left',on='code')
    df1['excess_ret']=df1['ret']-df1['mean']
    frame['Date0']=[day0]*len(demo_code)
    frame=frame.merge(df1.groupby('code').std()['excess_ret'].reset_index().\
    rename(columns={'excess_ret':'excess_std'}), how='left',on='code')
    frame=frame.merge(df1.groupby('code').mean()['excess_ret'].reset_index().\
    rename(columns={'excess_ret':'excess_mean'}), how='left',on='code')

    if len(demo_250)==0:
        demo_250=frame
    else:
        demo_250=pd.concat([demo_250,frame])

demo_250['t_stats']=demo_250['excess_mean']*(31**0.5)/demo_250['excess_std']

import numpy as np
interval = np.linspace(0.001,0.005,9)
for i in interval:
    abnormal = round(i,5)
    str = ('t_stats_{}'.format(abnormal))
    #print(str)
    demo_250[str]=(demo_250['excess_mean']+
    abnormal)*(31**0.5)/demo_250['excess_std']
    total = len(demo_250[str])
    reject = len(demo_250[str][np.abs(demo_250[str]) > 1.64])
    percentage = reject/total
    print(i,percentage)
```

## partb_code.py

```
import yfinance as yf
import lxml
import numpy as np
import pandas as pd
import scipy.stats as stats
import math
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import t as t
import statsmodels.api as sm
lowess = sm.nonparametric.lowess

#=================== Part (a) ============================
## ============== Retrieve the data ====================
# =======================================================
'''
log returns for sp 500
data contains sp500
demo and repu contains stock data
repu_code, demo_code represents the tickers for the stock
'''

## 1. Retrive data for SP500
tick = yf.Ticker('^GSPC')
data = pd.DataFrame(tick.history(period="max", rounding = False).loc[:,"Close"])
data['sp500'] = np.log(data['Close'] / data['Close'].shift(1))
data=data.reset_index()
data=data[data['Date']>=pd.Timestamp('1990-01-01')]
data=data.drop(columns=['Close'])

## 2. Retrive data for democracy portfolio
demo=pd.read_pickle('democrat_data.pkl')
demo=demo[demo['Date']>=pd.Timestamp('1990-01-01')]
demo=demo.drop(columns=['Date']).reset_index()
demo=demo.merge(data, how='left',on='Date')
demo['excess_ret']=demo['ret']-demo['sp500']
## get all the ticker democracy portfolio
demo_code=set(demo['code'].tolist())

## 3. Retrive data for republican portfolio
repu=pd.read_pickle('republican_data.pkl')
repu=repu[repu['Date']>=pd.Timestamp('1990-01-01')]
```

```
repu=repu.drop(columns=['Date']).reset_index()
repu=repu.merge(data, how='left',on='Date')
repu['excess_ret']=repu['ret']-repu['sp500']
# get all the ticker republican portfolio
repu_code=set(repu['code'].tolist())

## 4. Retrive the event data
# event_date is the timestamp of the event
# event_party contains the timestamp and the change of party.
event_date = pd.read_excel("election_dates.xlsx")
event_party = event_date.drop(columns=['Year','day']).rename(columns=
{'Date ':'Date'})
event_party['not_change']=event_party['Party']==event_party['Party'].shift(1)
# get the timestamep for the event
event_date=event_date.iloc[:,1].tolist()

#======================== Part (b) ============================
## ============== plot the moving average ====================
# ============================================================
'''
day0: the selection date
day_240: 240 days before the selection day
day_30: 30 days after the selection day
df_predict: the data between day_240 and day_30
df1: 20 days averages price for the period between day_240 and day_30
ma20_port_demo: 20 days averages price for democracy for 7 selection years
ma20_port_repu: 20 days averages price for republican for 7 selection years
'''

## moving average for democracy
ma20_port_demo=[]

for i in range(len(event_date[3:])):
    day0=event_date[i+3]
    day_240=day0-pd.tseries.offsets.BDay(239)
    day_30=day0+pd.tseries.offsets.BDay(60)
    df_predict=demo[demo['Date']>=day_240]
    df_predict=df_predict[df_predict['Date']<=day_30]
    df_predict=df_predict.loc[:,['ret','Date']].groupby('Date').mean()
    df1=df_predict.sort_values(by='Date').rolling(window=20).mean()
    ma20_port_demo.append(df1)

for i in ma20_port_demo:
    i.plot(legend=True)
```

```
    plt.axhline(0)
    plt.show()

## moving average for republican
ma20_port_repu=[]

for i in range(len(event_date[3:])):
    day0=event_date[i+3]
    day_240=day0-pd.tseries.offsets.BDay(239)
    day_30=day0+pd.tseries.offsets.BDay(60)
    df_predict=repu[repu['Date']>=day_240]
    df_predict=df_predict[df_predict['Date']<=day_30]
    df_predict=df_predict.loc[:,['ret','Date']].groupby('Date').mean()
    df1=df_predict.sort_values(by='Date').rolling(window=20).mean()
    ma20_port_repu.append(df1)

for i in ma20_port_demo:
    i.plot(legend=True)
    plt.axhline(0)
    plt.show()

#======================== Part (c) ============================
## ============== Test on the regular day ====================
# ===========================================================

## function to compute the t - test for democracy for normal day
def change_date_port_demo(before, test_before, test_after,after=90):
    '''
    before:
    test_befoore:
    test_after:
    after:
    '''

    frame=pd.DataFrame(columns=['Election_date','239_diff','30_diff','std'])
    demo_port_p=[]

    for i in range(len(event_date))[3:]:
        '''
        day0: the selection date
        day_240: 240 days before the selection day
        day_30: 30 days after the selection day
        df_predict: the data between day_240 and day_30
        df1: 20 days averages price for the period between day_240 and day_30
```

```
        ma20_port_demo: 20 days averages price for democracy for 7 selection
        years
        ma20_port_repu: 20 days averages price for republican for 7 selection
        years
        '''

        day0=event_date[i]
        day_240=day0-pd.tseries.offsets.BDay(before-1)
        day_test_b=day0-pd.tseries.offsets.BDay(test_before)
        day_test_a=day0+pd.tseries.offsets.BDay(test_after-1)
        day_30=day0+pd.tseries.offsets.BDay(after)
        df_predict=demo[demo['Date']>=day_240]
        df_predict=df_predict[df_predict['Date']<=day_30]

        ##use the portfolio ret and predict
        df=df_predict.loc[:,['Date','ret']].groupby('Date').mean().reset_index()
        x=df['Date'].to_numpy()
        y=df['ret'].to_numpy()
        # lowess function
        # Use the lowess function to predict
        # Which can prevent the influence from the seasons(frac = 0.25)
        predict=lowess(y,x, return_sorted=False, frac=0.25)
        df['predict']=predict
        df['diff']=df['ret']-df['predict']
        dic={}
        dic['Election_date']=[day0]
        b=df[df['Date']<day0]
        b=b[b['Date']>=day_test_b]
        a=df[df['Date']>=day0]
        a=a[a['Date']<=day_test_a]
        dic['239_diff']=b.mean()['diff']
        dic['30_diff']=a.mean()['diff']
        dic['std']=a.std()['diff']
        demo_port_p.append(df)
        a=pd.DataFrame(dic)
        frame=frame.append(a)
        # Compute the t-test
        frame['t_stats']=(frame['30_diff']-frame['239_diff'])*
        ((test_after)**0.5)/frame['std']

t_90 = t.ppf(0.9,test_after-1)
demo_sig=frame
demo_sig['positive_sig']=demo_sig['t_stats'] >= t_90
demo_sig['positive_insig']=(demo_sig['t_stats'] < t_90) & \
```

```
    (demo_sig['t_stats'] >= 0)
    demo_sig['negative_sig']=demo_sig['t_stats'] <= -t_90
    demo_sig['negative_insig']=(demo_sig['t_stats'] > -t_90) & \
    (demo_sig['t_stats'] < 0)
    demo_sig=demo_sig.iloc[:,[5,6,7,8]]

    return (demo_port_p, demo_sig)




## function to compute the t - test for republican for normal day
## the same structure like the democracy
def change_date_port_repu(before, test_before, test_after,after=90):
    frame=pd.DataFrame(columns=['Election_date','239_diff','30_diff','std'])
    demo_port_p=[]

    for i in range(len(event_date))[3:]:

        day0=event_date[i]
        day_240=day0-pd.tseries.offsets.BDay(before-1)
        day_test_b=day0-pd.tseries.offsets.BDay(test_before)
        day_test_a=day0+pd.tseries.offsets.BDay(test_after-1)
        day_30=day0+pd.tseries.offsets.BDay(after)
        df_predict=repu[repu['Date']>=day_240]
        df_predict=df_predict[df_predict['Date']<=day_30]

        ##use the portfolio ret and predict

        df=df_predict.loc[:,['Date','ret']].groupby('Date').mean().reset_index()
        x=df['Date'].to_numpy()
        y=df['ret'].to_numpy()
        predict=lowess(y,x, return_sorted=False, frac=0.25)
        df['predict']=predict
        df['diff']=df['ret']-df['predict']
        dic={}
        dic['Election_date']=[day0]
        b=df[df['Date']<day0]
        b=b[b['Date']>=day_test_b]
        a=df[df['Date']>=day0]
        a=a[a['Date']<=day_test_a]
        dic['239_diff']=b.mean()['diff']
        dic['30_diff']=a.mean()['diff']
        dic['std']=a.std()['diff']
```

```
        demo_port_p.append(df)
        a=pd.DataFrame(dic)
        frame=frame.append(a)
        frame['t_stats']=(frame['30_diff']-frame['239_diff'])*
        ((test_after)**0.5)/frame['std']

    t_90 = t.ppf(0.9,test_after-1)
    demo_sig=frame
    demo_sig['positive_sig']=demo_sig['t_stats'] >= t_90
    demo_sig['positive_insig']=(demo_sig['t_stats'] < t_90) & \
    (demo_sig['t_stats'] >= 0)
    demo_sig['negative_sig']=demo_sig['t_stats'] <= -t_90
    demo_sig['negative_insig']=(demo_sig['t_stats'] > -t_90) & \
    (demo_sig['t_stats'] < 0)
    demo_sig=demo_sig.iloc[:,[5,6,7,8]]

    return (demo_port_p, demo_sig)

#==================== Part (d) ============================
## ============== Run the function ======================
# ======= plot the returns and prediction ===============
# ========================================================

demo_port,demo_port_sig=change_date_port_demo(240,60,30)
for i in demo_port:
    plt.figure()
    i.set_index('Date').plot(legend=True)

repu_port,repu_port_sig=change_date_port_repu(240,60,30)
for i in repu_port:
    plt.figure()
    i.set_index('Date').plot(legend=True)


#====================== Part (e) ============================
## ============== Test on the event day ====================
# ==========================================================

# now we use loess to estimate the trend with each of the stock at selected
# election date,
## and test whether there's significant difference between the mean diff of
# before and after
### election date.
## The function is almost like the one in the previous
```

```
## for democracy
def change_date_demo(before, after,test_before,test_after, frac=0.25):
    frame=pd.DataFrame(columns=['Election_date','code','239_diff','30_diff',
    'std'])

    for i in range(len(event_date)):

        day0=event_date[i]
        day_240=day0-pd.tseries.offsets.BDay(before-1)
        day_30=day0+pd.tseries.offsets.BDay(after-1)
        day_test_b=day0-pd.tseries.offsets.BDay(test_before)
        day_test_a=day0+pd.tseries.offsets.BDay(test_after-1)
        df_predict=demo[demo['Date']>=day_240]
        df_predict=df_predict[df_predict['Date']<=day_30]

        #then we substract each stock's timeseries
        code_lst=list(set(df_predict['code'].tolist()))

        #our loop begins
        for stock in code_lst:
            df=df_predict
            df=df[df['code']==stock]
            x=df['Date'].to_numpy()
            y=df['ret'].to_numpy()
            predict=lowess(y,x, return_sorted=False, frac=frac)
            df['predict']=predict
            df['diff']=df['ret']-df['predict']
            dic={}
            dic['Election_date']=[day0]
            dic['code']=[stock]


            b=df[df['Date']<day0]
            b=b[b['Date']>=day_test_b]
            a=df[df['Date']>=day0]
            a=a[a['Date']<=day_test_a]
            dic['239_diff']=b.mean()['diff']
            dic['30_diff']=a.mean()['diff']
            dic['std']=a.std()['diff']

            df2=pd.DataFrame(dic)
            frame=frame.append(df2, ignore_index=True)
    demo_before=frame
    demo_before['t_stats']=(demo_before['30_diff']-demo_before['239_diff'])*
```

```
    (test_after**0.5)/demo_before['std']
    demo_before=demo_before.merge(event_party,left_on='Election_date',
    right_on='Date', how='left')
    demo_before= demo_before.drop(columns = "Date")


    t_90 = t.ppf(0.9,test_after-1)
    demo_sig=demo_before
    demo_sig['positive_sig']=demo_sig['t_stats'] >= t_90
    demo_sig['positive_insig']=(demo_sig['t_stats'] < t_90) & \
    (demo_sig['t_stats'] >= 0)
    demo_sig['negative_sig']=demo_sig['t_stats'] <= -t_90
    demo_sig['negative_insig']=(demo_sig['t_stats'] > -t_90) & \
    (demo_sig['t_stats'] < 0)
    demo_sig=demo_sig.iloc[:,[6,7,8,9,10,11]]
    demo_sig=demo_sig.groupby(by=['Party','not_change']).mean()

    return (demo_before, demo_sig)


## for republican
def change_date_repu(before, after,test_before,test_after,frac=0.25):
    frame=pd.DataFrame(columns=['Election_date','code','239_diff','30_diff',
    'std'])

    for i in range(len(event_date)):

        day0=event_date[i]
        day_240=day0-pd.tseries.offsets.BDay(before-1)
        day_30=day0+pd.tseries.offsets.BDay(after-1)
        day_test_b=day0-pd.tseries.offsets.BDay(test_before)
        day_test_a=day0+pd.tseries.offsets.BDay(test_after-1)
        df_predict=repu[repu['Date']>=day_240]
        df_predict=df_predict[df_predict['Date']<=day_30]

        #then we substract each stock's timeseries
        code_lst=list(set(df_predict['code'].tolist()))

        #our loop begins
        for stock in code_lst:
            df=df_predict
            df=df[df['code']==stock]
            x=df['Date'].to_numpy()
            y=df['ret'].to_numpy()
```

```
            predict=lowess(y,x, return_sorted=False, frac=frac)
            df['predict']=predict
            df['diff']=df['ret']-df['predict']
            dic={}
            dic['Election_date']=[day0]
            dic['code']=[stock]


            b=df[df['Date']<day0]
            b=b[b['Date']>=day_test_b]
            a=df[df['Date']>=day0]
            a=a[a['Date']<=day_test_a]
            dic['239_diff']=b.mean()['diff']
            dic['30_diff']=a.mean()['diff']
            dic['std']=a.std()['diff']

            df2=pd.DataFrame(dic)
            frame=frame.append(df2, ignore_index=True)
    demo_before=frame
    demo_before['t_stats']=(demo_before['30_diff']-demo_before['239_diff'])*
    (test_after**0.5)/demo_before['std']
    demo_before=demo_before.merge(event_party,left_on='Election_date',
    right_on='Date', how='left')
    demo_before= demo_before.drop(columns = "Date")


    t_90 = t.ppf(0.9,test_after-1)
    demo_sig=demo_before
    demo_sig['positive_sig']=demo_sig['t_stats'] >= t_90
    demo_sig['positive_insig']=(demo_sig['t_stats'] < t_90) & \
    (demo_sig['t_stats'] >= 0)
    demo_sig['negative_sig']=demo_sig['t_stats'] <= -t_90
    demo_sig['negative_insig']=(demo_sig['t_stats'] > -t_90) & \
    (demo_sig['t_stats'] < 0)
    demo_sig=demo_sig.iloc[:,[6,7,8,9,10,11]]
    demo_sig=demo_sig.groupby(by=['Party','not_change']).mean()

    return (demo_before, demo_sig)

## Run the function
demo_before,demo_sig=change_date_demo(240, 90,60,30)
repu_before,repu_sig=change_date_repu(240, 90,60,30)

repu_sig=repu_sig.reset_index()
```

```
repu_sig['Party_Before']=['Republican', 'Democratic', 'Democratic',
'Republican']
repu_sig=repu_sig.drop(columns='not_change').loc[:,['Party_Before','Party',
'positive_sig','positive_insig','negative_insig','negative_sig']]

demo_sig=demo_sig.reset_index()
demo_sig['Party_Before']=['Republican', 'Democratic', 'Democratic',
'Republican']
demo_sig=demo_sig.drop(columns='not_change').loc[:,['Party_Before','Party',
'positive_sig','positive_insig','negative_insig','negative_sig']]

print(repu_sig)
print(demo_sig)
```

## copula_simulation.R

```
## IAQF 2020
##
## PART 1: Simulate Portfolio Returns with t-Copula
## Fit each asset with GPD lower tail and t-distributed upper tail


rm(list = ls())

library(copula)
library(fGarch) # need for standardized t density
library(MASS) # need for fitdistr and kde2d
library(fCopulae)
library(BatchGetSymbols)
library(dplyr)
library(tidyr)
library(ggplot2)
library(evir)
library(bizdays)

compute_copula = function(start, end, port, title) {
  # set dates
  election_day = start
  cal = create.calendar(name = "mycal", weekdays=c("saturday", "sunday"))
  last.date = as.Date(end)
  first.date = offset(election_day, -50, cal)

  freq.data = 'daily'
  calc_log_rets = TRUE

  # Read in Data
  data = BatchGetSymbols(tickers = port,
                         first.date = first.date,
                         last.date = last.date,
                         freq.data = freq.data,
                         cache.folder = file.path(tempdir(),
                                                  'BGS_Cache') )

  if (calc_log_rets) {
    df = data$df.tickers
    df = df[,c("ref.date", "ticker", "price.adjusted")]
    data_wide = spread(df, ticker, price.adjusted)
    data_wide = data_wide[complete.cases(data_wide),]
```

```
  rownames(data_wide) = data_wide[,"ref.date"]
  data_wide = select(data_wide, -ref.date)
  data_wide = apply(data_wide, 2, function(x){diff(log(x))})
} else {
  # Construct Data frame of returns per asset
  df = data$df.tickers
  df = df[,c("ref.date", "ticker", "ret.adjusted.prices")]
  data_wide = spread(df, ticker, ret.adjusted.prices)
  data_wide = data_wide[complete.cases(data_wide), ]
  rownames(data_wide) = data_wide[,"ref.date"]
  data_wide = select(data_wide, -ref.date)
  data_wide = data_wide[-1,]
}

# Filter Outliers
filter_outlier = function(x) {
  iqr = 5 * IQR(x)
  ind1 = which(x > iqr + quantile(x, c(0.75)))
  ind2 = which(x < -iqr + quantile(x, c(0.25)))
  ind = c(ind2, ind1)
  x[ind] = NA
  x
}

data_wide = apply(data_wide, 2, filter_outlier)
data_wide = data_wide[complete.cases(data_wide), ]

# Generating initial estimate of correlation for t-copula
corr_mat = cor(data_wide,method="spearman")
cor_tau = upper.tri(corr_mat, diag = FALSE)
cor_tau = corr_mat[cor_tau]

# Fit a t-distribution to each asset ----------------------------------------
# Prepare to fit copula
data1 = data.frame(data_wide[,1])
i = 1
for (col in colnames(data_wide)) {
  dat = sort(data_wide[,i])

  # Fraction of data to split
  split_quant = (which(dat > 0)[1] - 2) / length(dat)
  lower_quant = which.min(abs(dat - quantile(dat, split_quant)))

  # Fit GPD to lower tail
```

```
  u = quantile(-dat, c(1 - split_quant))
  gpd_est = gpd(-dat, threshold = u, method = c("ml"),
                information = c("observed"))
  beta = gpd_est$par.ests[2]
  xi = gpd_est$par.ests[1]

  # Convert upper tail negative returns to lower tail returns
  x = gpd_est
  dat1 <- as.numeric(gpd_est$data)
  threshold <- x$threshold
  plotmin <- threshold
  extend = 1.5
  plotmax <- max(dat1) * extend
  xx <- seq(from = 0, to = 1, length = 1000)
  z <- qgpd(xx, xi, threshold, beta)
  z <- pmax(pmin(z, plotmax), plotmin)
  y <- pgpd(z, xi, threshold, beta)
  prob <- x$p.less.thresh
  y <- (1 - prob) * (1 - y)
  y = sort(sample(y, size =lower_quant), decreasing = FALSE)

  # Estimate t-dsitribution
  est = as.numeric(fitdistr(dat,"t")$estimate)
  est[3] = max(2.1, est[3])
  est[2] = est[2]*sqrt(est[3] / (est[3]-2))
  t_cdf = pstd(dat[(lower_quant+1):length(dat)],
               mean = est[1], sd = est[2], nu = est[3])
  temp = data.frame(col = c(y, t_cdf))
  colnames(temp) = col
  data1 = cbind(data1, temp)
  i = i + 1
}
data1 = data1[,-1]


## THIS STEP TAKES ABOUT 5-10 MINUTES ----------------------------------------
# Fit t-copula, Note: choose(15,2) + 1 parameters to be estimated
cop_t_dim2 = tCopula(cor_tau, dim = nrow(corr_mat), dispstr = "un", df = 50)

ft1 = fitCopula(cop_t_dim2, data = data1,
                method = "ml",
                start = c(rep(0, length(cor_tau)), 50))
# ---------------------------------------------------------------------------
```

```r
# Set up graph for simulations -----------
election_ind = which(rownames(data_wide) %in% c(election_day))
draws = nrow(data_wide) - election_ind
#draws = 181
capital = 1e6

# Build cumulative returns of original series before the election
real_rets = cumprod(1 + rowMeans(exp(data_wide) - 1))
df_plot = data.frame(time = 1:length(real_rets), y = real_rets)

# Generate plot of cumulative returns
s = ggplot(df_plot, aes(x = time, y = y)) +
  geom_line(colour = "darkred")


# Generate multiple paths after election ----------------------------------
num_paths = 300
n = length(ft1@estimate)
rhos = ft1@estimate[1:n-1]
nu = ft1@estimate[n]
# q_split = 0.25

# data3 contains column vectors of return paths
data3 = data.frame(data_wide[1:draws,1])
for (path in 1:num_paths) {
  # Generate Random Variates from our copula
  ct = tCopula(rhos, dim = nrow(corr_mat), dispstr = "un", df = nu)
  uvsim = rCopula(draws, ct)

  # Recover simulation values based on inverse transformation
  data2 = data.frame(data_wide[1:draws,1])
  for (i in 1:nrow(corr_mat)) {
    # Re-estimate the data with our distributions
    dat = sort(data_wide[(nrow(data_wide) - draws):nrow(data_wide), i])
    q_split = (which(dat > 0)[1] - 2) / length(dat)

    # Set lower and upper indices to be estimated by both distributions
    gpd_ind = which(uvsim[, i] <= q_split)
    t_ind = which(uvsim[, i] > q_split)

    # Split simulations, this is done to estimate upper tail of returns
    g_dat = (1 - uvsim[gpd_ind, i] - (1 - q_split)) / q_split
    t_dat = (uvsim[t_ind, i] - q_split) / (1 - q_split)
```

```
    temp = rep(0, length(uvsim[, i]))
    lower_quant = which.min(abs(dat - quantile(dat, q_split)))

    # Fit t to upper tail
    est = as.numeric(fitdistr(dat,"t")$estimate)
    est[3] = max(2.1, est[3])
    est[2] = est[2]*sqrt(est[3] / (est[3] - 2))
    temp[t_ind] = qstd(uvsim[t_ind, i], mean = est[1], sd = est[2],
                       nu = est[3])

    # Fit GPD to lower tail
    u = quantile(-dat, c(1 - q_split))
    gpd_est = gpd(-dat, threshold = u, method = c("ml"),
                  information = c("observed"))
    beta = gpd_est$par.ests[2]
    xi = gpd_est$par.ests[1]
    temp[gpd_ind] = -qgpd(g_dat, xi = xi, mu=u, beta=beta)

    temp = data.frame(col = exp(temp) - 1)
    colnames(temp) = colnames(data_wide)[i]
    data2 = cbind(data2, temp)
  }
  data2 = data2[,-1]

  # Look at Cumulative returns, before and after simulation
  sim_returns = cumprod(1 + rowMeans(data2)) * real_rets[election_ind]
  data3 = cbind(data3, sim_returns)
  total = length(real_rets) + draws

  #color = c("black", "steelblue")
  #color = sample(color, size = 1, prob = c(0.5, 0.5))
  color = "steelblue"

  # Make all vectors of equal size
  sim_returns = c(rep(NA, election_ind -1),
                  c(real_rets[election_ind], sim_returns))
  df_plot_sim = data.frame(time = 1:length(sim_returns), z = sim_returns)
  s = s + geom_line(data = df_plot_sim,
                    aes(x = time, y = z), colour = color, alpha = 0.3)
}

data3 = data3[,-1]
data4 = data3 / real_rets[election_ind]
save(data1, data4, s, ft1, file = title)
```

```
}

# DATA SETUP ---------------------------------------------------------------------
# set tickers
democrat_port = c("APTV","KO","STZ","CSX","EL","EXC","FSLR","F",
                  "HD","MCD","NEE", "NSC","SPG","SPWR","WMT")

republican_port = c("GOOG","AMZN", "AXP","CVX","C","COP","FB","GILD",
                    "HON","MRO","MRK","PYPL","QCOM","CRM","V")

num_params = choose(length(democrat_port), 2) + 1
cal = create.calendar(name = "mycal", weekdays=c("saturday", "sunday"))
end = "2020-02-21"
election_day = offset(end, -280, cal)
title = "dem_sim.Rdata"

compute_copula(election_day, end, republican_port, title)
# bizdays(from = "2019-01-31", to = "2019-11-02", cal)

# set dates
election_day = election_day
cal = create.calendar(name = "mycal", weekdays=c("saturday", "sunday"))
last.date = as.Date("2020-02-21")
first.date = offset(election_day, -50, cal)

freq.data = 'daily'
calc_log_rets = TRUE

# Read in Data
data = BatchGetSymbols(tickers = democrat_port,
                       first.date = first.date,
                       last.date = last.date,
                       freq.data = freq.data,
                       cache.folder = file.path(tempdir(),
                                                'BGS_Cache') )

if (calc_log_rets) {
  df = data$df.tickers
  df = df[,c("ref.date", "ticker", "price.adjusted")]
  data_wide = spread(df, ticker, price.adjusted)
  data_wide = data_wide[complete.cases(data_wide),]
  rownames(data_wide) = data_wide[,"ref.date"]
  data_wide = select(data_wide, -ref.date)
  data_wide = apply(data_wide, 2, function(x){diff(log(x))})
```

```
} else {
  # Construct Data frame of returns per asset
  df = data$df.tickers
  df = df[,c("ref.date", "ticker", "ret.adjusted.prices")]
  data_wide = spread(df, ticker, ret.adjusted.prices)
  data_wide = data_wide[complete.cases(data_wide), ]
  rownames(data_wide) = data_wide[,"ref.date"]
  data_wide = select(data_wide, -ref.date)
  data_wide = data_wide[-1,]
}

# Filter Outliers
filter_outlier = function(x) {
  iqr = 5 * IQR(x)
  ind1 = which(x > iqr + quantile(x, c(0.75)))
  ind2 = which(x < -iqr + quantile(x, c(0.25)))
  ind = c(ind2, ind1)
  x[ind] = NA
  x
}

data_wide = apply(data_wide, 2, filter_outlier)
data_wide = data_wide[complete.cases(data_wide), ]

# Generating initial estimate of correlation for t-copula
trunc = 120
corr_mat = cor(data_wide[(nrow(data_wide) - trunc):nrow(data_wide),],
               method="spearman")
cor_tau = upper.tri(corr_mat, diag = FALSE)
cor_tau = corr_mat[cor_tau]

# Fit a t-distribution to each asset ----------------------------------------
# Prepare to fit copula
data1 = data.frame(data_wide[(nrow(data_wide) - trunc):nrow(data_wide),1])
i = 1
for (col in colnames(data_wide)) {
  #dat = sort(data_wide[,i])
  dat = sort(data_wide[(nrow(data_wide) - trunc):nrow(data_wide), i])

  # Fraction of data to split
  split_quant = (which(dat > 0)[1] - 2) / length(dat)
  lower_quant = which.min(abs(dat - quantile(dat, split_quant)))

  # Fit GPD to lower tail
```

```
  u = quantile(-dat, c(1 - split_quant))
  gpd_est = gpd(-dat, threshold = u, method = c("ml"),
                information = c("observed"))
  beta = gpd_est$par.ests[2]
  xi = gpd_est$par.ests[1]

  # Convert upper tail negative returns to lower tail returns
  x = gpd_est
  dat1 <- as.numeric(gpd_est$data)
  threshold <- x$threshold
  plotmin <- threshold
  extend = 1.5
  plotmax <- max(dat1) * extend
  xx <- seq(from = 0, to = 1, length = 1000)
  z <- qgpd(xx, xi, threshold, beta)
  z <- pmax(pmin(z, plotmax), plotmin)
  y <- pgpd(z, xi, threshold, beta)
  prob <- x$p.less.thresh
  y <- (1 - prob) * (1 - y)
  y = sort(sample(y, size =lower_quant), decreasing = FALSE)

  # Estimate t-dsitribution
  est = as.numeric(fitdistr(dat,"t")$estimate)
  est[3] = max(2.1, est[3])
  est[2] = est[2]*sqrt(est[3] / (est[3]-2))
  t_cdf = pstd(dat[(lower_quant+1):length(dat)],
               mean = est[1], sd = est[2], nu = est[3])
  temp = data.frame(col = c(y, t_cdf))
  colnames(temp) = col
  data1 = cbind(data1, temp)
  i = i + 1
}
data1 = data1[,-1]


## THIS STEP TAKES ABOUT 5-10 MINUTES ----------------------------------------
# Fit t-copula, Note: choose(15,2) + 1 parameters to be estimated
cop_t_dim2 = tCopula(cor_tau, dim = nrow(corr_mat), dispstr = "un", df = 80)

ft1 = fitCopula(cop_t_dim2, data = data1,
                method = "ml",
                start = c(rep(0, length(cor_tau)), 80))
# ---------------------------------------------------------------------------
```

```r
# Set up graph for simulations -----------
election_ind = which(rownames(data_wide) %in% c(election_day))
# draws = nrow(data_wide) - election_ind
draws = 181
capital = 1e6

# Build cumulative returns of original series before the election
real_rets = cumprod(1 + rowMeans(exp(data_wide) - 1))
df_plot = data.frame(time = 1:length(real_rets), y = real_rets)


# Generate multiple paths after election -------------------------------------
num_paths = 300
n = length(ft1@estimate)
rhos = ft1@estimate[1:n-1]
nu = ft1@estimate[n]
# q_split = 0.25

# data3 contains column vectors of return paths
data3 = data.frame(data_wide[1:draws,1])
for (path in 1:num_paths) {
  # Generate Random Variates from our copula
  ct = tCopula(rhos, dim = nrow(corr_mat), dispstr = "un", df = nu)
  uvsim = rCopula(draws, ct)

  # Recover simulation values based on inverse transformation
  data2 = data.frame(data_wide[1:draws,1])
  for (i in 1:nrow(corr_mat)) {
    # Re-estimate the data with our distributions
    dat = sort(data_wide[(nrow(data_wide) - draws):nrow(data_wide), i])
    q_split = (which(dat > 0)[1] - 2) / length(dat)

    # Set lower and upper indices to be estimated by both distributions
    gpd_ind = which(uvsim[, i] <= q_split)
    t_ind = which(uvsim[, i] > q_split)

    # Split simulations, this is done to estimate upper tail of returns
    g_dat = (1 - uvsim[gpd_ind, i] - (1 - q_split)) / q_split
    t_dat = (uvsim[t_ind, i] - q_split) / (1 - q_split)

    temp = rep(0, length(uvsim[, i]))
    lower_quant = which.min(abs(dat - quantile(dat, q_split)))

    # Fit t to upper tail
```

```
    est = as.numeric(fitdistr(dat,"t")$estimate)
    est[3] = max(2.1, est[3])
    est[2] = est[2]*sqrt(est[3] / (est[3] - 2))
    temp[t_ind] = qstd(uvsim[t_ind, i], mean = est[1], sd = est[2], nu = est[3])
    #temp[t_ind] = qstd(t_dat, mean = est[1], sd = est[2],
    #                   nu = est[3])

    # Fit GPD to lower tail
    u = quantile(-dat, c(1 - q_split))
    gpd_est = gpd(-dat, threshold = u, method = c("ml"),
                  information = c("observed"))
    beta = gpd_est$par.ests[2]
    xi = gpd_est$par.ests[1]
    temp[gpd_ind] = -qgpd(g_dat, xi = xi, mu=u, beta=beta)

    temp = data.frame(col = exp(temp) - 1)
    colnames(temp) = colnames(data_wide)[i]
    data2 = cbind(data2, temp)
  }
  data2 = data2[,-1]

  # Look at Cumulative returns, before and after simulation
  sim_returns = cumprod(1 + rowMeans(data2))
  data3 = cbind(data3, sim_returns)
}

data3 = data3[,-1]
#data4 = data3 / real_rets[election_ind]
save(data1, data3, ft1, file = title)
```

## structured_note.R

```
## IAQF 2020
##
## PART 2: Combine simulations and apply payoff function
rm(list = ls())
library(ggplot2)

load("obama_dem.Rdata")
obama_dem = data4
obama_dem = data4[,-c(28)]
obama_dem = obama_dem[-c(54, 55, 56),]


load("obama_rep.Rdata")
obama_rep = data4
# which.max(data4[nrow(data4),])
obama_rep = data4[-nrow(obama_rep),-c(132)]
obama_rep = obama_rep[-c(54, 55, 56),]


load("trump_demo.Rdata")
trump_dem = data4
trump_dem = data4[-nrow(trump_dem),]


load("trump_rep.Rdata")
trump_rep = data4
trump_rep = data4[,-c(42)]

s = s + ggtitle("Simulating Performance for Republican Portfolio
                (2016 Trump Eelection)") +
  xlab("Number of Business Days") +
  ylab("Cumulative Returns") +
  geom_vline(xintercept = 46) +
  geom_text(aes(x = 50, y = 0.9), label = "Election Day")

ggsave("c4.png", s)




load("dem_sim.Rdata")
dem_sim = data3
```

```r
load("rep_sim.Rdata")
rep_sim = data3

# Payoffs of Republican
r = 0.013
capital = 100
K_R = 150
K_D = 150
parameters = data.frame(mean = 0, sd = 0, K_R = K_R, K_D = K_D)

compute_payoffs = function(K_R, K_D) {
  samps = 50
  dem = sample(dem_sim[nrow(dem_sim),], 100)
  rep = sample(rep_sim[nrow(rep_sim),], 100)

  payoff_R_win = c()
  payoff_D_win = c()

  for (i in 1:length(dem)) {
    r_val = runif(1)
    # Simulate Republican Winning
    if (r_val > 0.5) {
      S_D = (capital / 2) * dem[,i] * sample(trump_dem[nrow(trump_dem),], samps)
      S_R = (capital / 2) * rep[,i] * sample(trump_rep[nrow(trump_rep),], samps)
      payoff_R_win =  c(payoff_R_win,
                        mean(0.5 *(apply(K_R - S_R, 2,
                                         function (x) max(x, 0)) -
                                   apply(K_D - S_D, 2,
                                         function (x) max(x, 0))) * exp(-r))
      )
    }

    if (r_val <= 0.5) {
      # Simulate Democrat Winning
      S_D = (capital / 2) * dem[,i] * sample(obama_dem[nrow(obama_dem),], samps)
      S_R = (capital / 2) * rep[,i] * sample(obama_rep[nrow(obama_rep),], samps)
      payoff_D_win =  c(payoff_D_win,
                        mean(0.5 *(apply(K_D - S_D, 2,
                                         function (x) max(x, 0)) -
                                   apply(K_R - S_R, 2,
                                         function (x) max(x, 0))) * exp(-r))
      )
    }
  }
```

```
  payoffs = c(payoff_R_win, payoff_D_win)
  temp_df = data.frame(mean = mean(payoffs), sd = sd(payoffs), K_R = K_R,
                       K_D = K_D)
  return(temp_df)
}

# Compute with multiple strike prices ----------------------------------------
for (kr in c(55:65)) {
  for (kd in c(55:65)) {
    temp_df = compute_payoffs(kr, kd)
    parameters = rbind(parameters, temp_df)
    show(c(kr,kd))
  }
}


# Plot Simulations to show example -------------------------------------------
library(bizdays)

# Simulate Republican
df = data.frame(time = 1:234, y = c(rep_sim[,1], rep(NA, 53)))
s = ggplot(df, aes(x = time, y = y)) +
  geom_line(colour = "black", alpha = 0.3)

for (i in 2:(ncol(rep_sim) - 1)) {
  if (i == 2) {
    for (j in 1:299) {
      df = data.frame(time = 1:234, y = c(rep(NA,180),
                                          rep_sim[181,i], rep_sim[181,i] *
                                            trump_rep[,j]))
      s = s + geom_line(data = df,
                        aes(x = time, y = y), colour = "darkred", alpha = 0.3)
    }
  } else if (i == 3) {
    for (j in 1:299) {
      df = data.frame(time = 1:234, y = c(rep(NA,180),
                                          rep_sim[181,i], rep_sim[181,i] *
                                            obama_rep[,j]))
      s = s + geom_line(data = df,
                        aes(x = time, y = y), colour = "steelblue", alpha = 0.3)
    }
  } else {
    df = data.frame(time = 1:234, y = c(rep_sim[,i], rep(NA, 53)))
    s = s + geom_line(data = df,
```

```
                      aes(x = time, y = y), colour = "black", alpha = 0.3)
  }

}

s1 = s + ggtitle("Simulation of outcomes for Republican Portfolio") +
  ylab("returns") +
  xlab("Number of Business Days")


# Simulate Democrat
df = data.frame(time = 1:234, y = c(dem_sim[,1], rep(NA, 53)))
s = ggplot(df, aes(x = time, y = y)) +
  geom_line(colour = "black", alpha = 0.3)

for (i in 2:(ncol(dem_sim) - 1)) {
  if (i == 17) {
    for (j in 1:299) {
      df = data.frame(time = 1:234, y = c(rep(NA,180),
                                          dem_sim[181,i], dem_sim[181,i] *
                                            trump_dem[,j]))
      s = s + geom_line(data = df,
                        aes(x = time, y = y), colour = "darkred", alpha = 0.3)
    }
  } else if (i == 12) {
    for (j in 1:299) {
      df = data.frame(time = 1:234, y = c(rep(NA,180),
                                          dem_sim[181,i], dem_sim[181,i] *
                                            obama_dem[,j]))
      s = s + geom_line(data = df,
                        aes(x = time, y = y), colour = "steelblue", alpha = 0.3)
    }
  } else {
    df = data.frame(time = 1:234, y = c(dem_sim[,i], rep(NA, 53)))
    s = s + geom_line(data = df,
                      aes(x = time, y = y), colour = "black", alpha = 0.3)
  }

}

s2 = s + ggtitle("Simulation of outcomes for Democrat Portfolio") +
  ylab("returns") +
  xlab("Number of Business Days")
```

```r
ggsave("demo_simulation.png", s2)
ggsave("rep_simulation.png", s1)


library(plotly)
parameters = parameters[-1,]
parameters$ratio = parameters$mean / parameters$sd
z_vals = matrix(0, nrow = 11, ncol = 11)
for (i in 1:11) {
  z_vals[i,] = parameters$ratio[(10*(i-1)+i):(10*(i) + i)]
}

df.list <- list(x = 55:65,
                y = 55:65,
                z = data.frame(z_vals))

data = data.frame(z_vals)

# Create Sharpe Ratio Surface
fig = plot_ly(data, x = 55:65, y = 55:65, z = z_vals, type = "surface")
fig <- fig %>% layout(
  title = "Surface plot of Payoff Sharpe Ratio",
  scene = list(
    xaxis = list(title = "K_R"),
    yaxis = list(title = "K_D"),
    zaxis = list(title = "Sharpe")
  ))

fig
```