ML Group Project

December 22, 2023

Group Information

Siqi Yang (sy3153)

Rui Ji (rj2709)

Jinghong Zou (jz3704)

Yushan Zhang (yz4739)

Set up

[36]:
```
import pandas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from statsmodels.formula.api import ols
from scipy.stats import gaussian_kde
import scipy
import scipy.sparse
import patsy
from statistics import median
import bz2
import math
import os
```

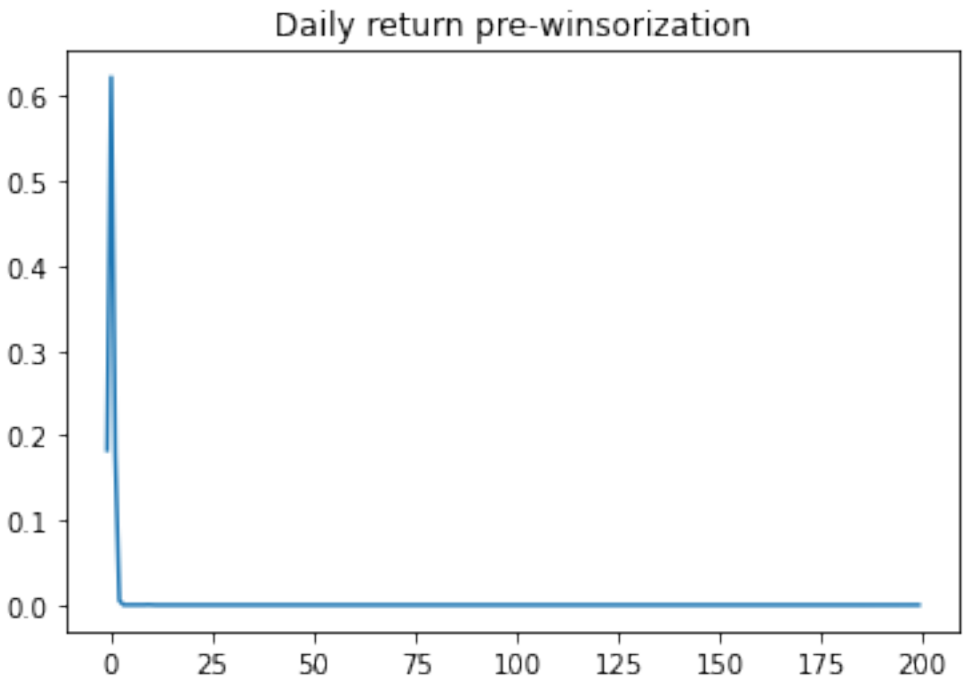
[2]:
```
model_dir = '/Users/raymo/Documents/FACTOR_MODEL/'

def sort_cols(test):
    return(test.reindex(sorted(test.columns), axis=1))

frames = {}
for year in [2003,2004,2005,2006,2007,2008,2009,2010]:
    fil = model_dir + "pandas-frames." + str(year) + ".pickle.bz2"
    frames.update(pd.read_pickle(fil))
for x in frames:
```

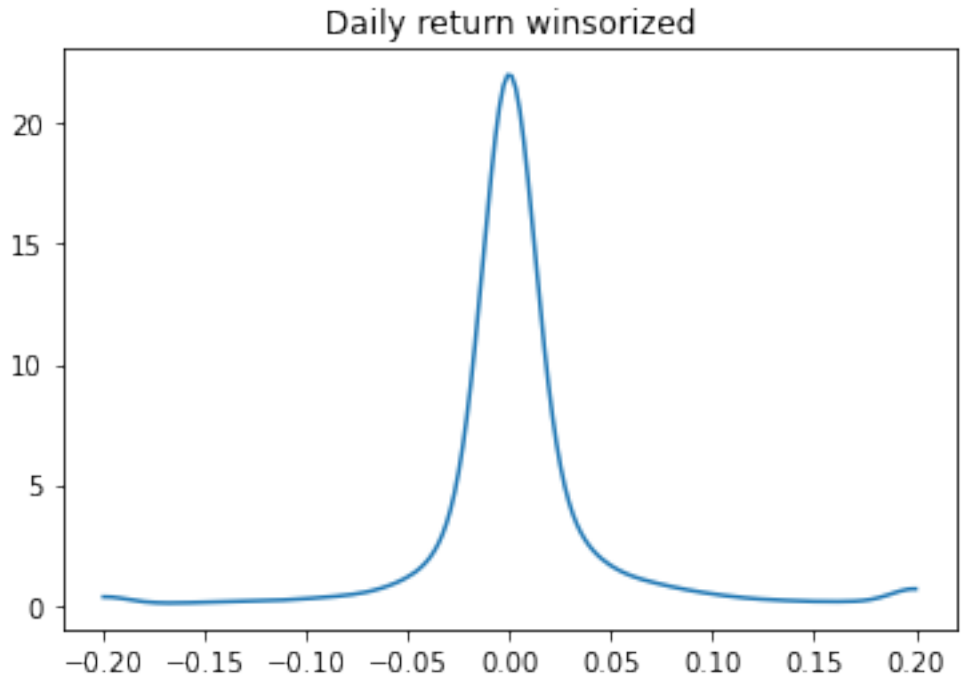
```
    frames[x] = sort_cols(frames[x])

covariance = {}
for year in [2003,2004,2005,2006,2007,2008,2009,2010]:
    fil = model_dir + "covariance." + str(year) + ".pickle.bz2"
    covariance.update(pd.read_pickle(fil))
```

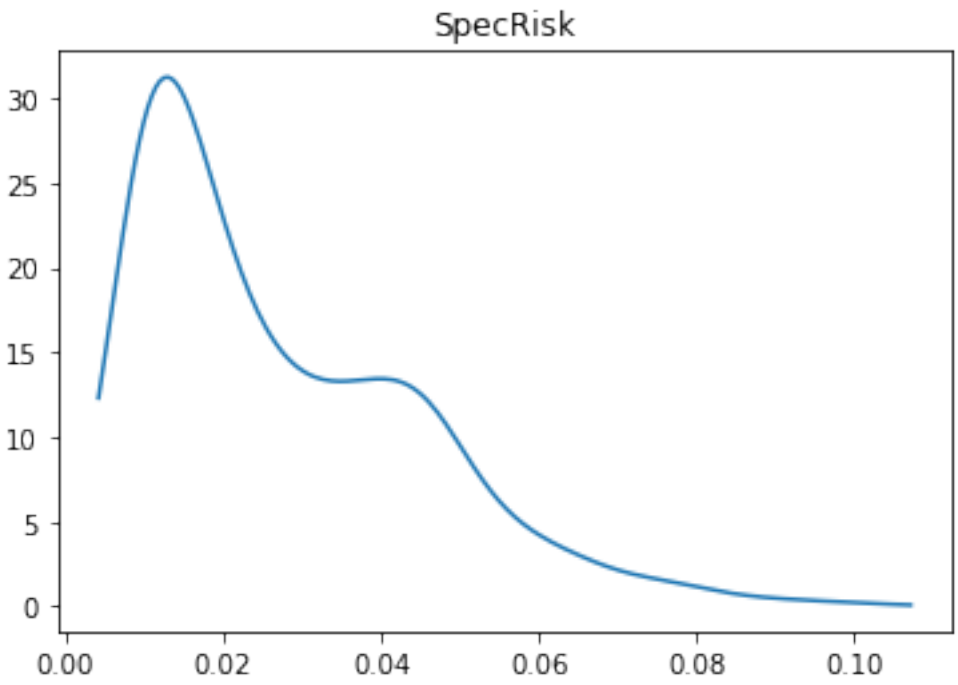
[3]:
```
industry_factors = ['AERODEF', 'AIRLINES', 'ALUMSTEL', 'APPAREL', 'AUTO',
       'BANKS','BEVTOB', 'BIOLIFE', 'BLDGPROD','CHEM', 'CNSTENG',
        'CNSTMACH', 'CNSTMATL', 'COMMEQP', 'COMPELEC',
       'COMSVCS', 'CONGLOM', 'CONTAINR', 'DISTRIB',
       'DIVFIN', 'ELECEQP', 'ELECUTIL', 'FOODPROD', 'FOODRET', 'GASUTIL',
       'HLTHEQP', 'HLTHSVCS', 'HOMEBLDG', 'HOUSEDUR','INDMACH',␣
 ↪'INSURNCE','INTERNET',
'LEISPROD', 'LEISSVCS', 'LIFEINS', 'MEDIA', 'MGDHLTH','MULTUTIL',
       'OILGSCON', 'OILGSDRL', 'OILGSEQP', 'OILGSEXP',
        'PAPER', 'PHARMA', 'PRECMTLS','PSNLPROD','REALEST',
       'RESTAUR', 'ROADRAIL','SEMICOND', 'SEMIEQP','SOFTWARE',
        'SPLTYRET', 'SPTYCHEM', 'SPTYSTOR', 'TELECOM', 'TRADECO',␣
 ↪'TRANSPRT','WIRELESS']

style_factors = ['BETA','SIZE','MOMENTUM','VALUE','LEVERAGE','LIQUIDTY']
```

[4]:
```
def wins(x,a,b):
    return(np.where(x <= a,a, np.where(x >= b, b, x)))
def clean_nas(df):
    numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()
    for numeric_column in numeric_columns:
        df[numeric_column] = np.nan_to_num(df[numeric_column])
    return df
```

[5]:
```
def density_plot(data, title):
    density = gaussian_kde(data)
    xs = np.linspace(np.min(data),np.max(data),200)
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs,density(xs))
    plt.title(title)
    plt.show()


test = frames['20040102']
density_plot(test['Ret'], 'Daily return pre-winsorization')
density_plot(wins(test['Ret'],-0.2,0.2), 'Daily return winsorized')
D = (test['SpecRisk'] / (100 * math.sqrt(252))) ** 2
density_plot(np.sqrt(D), 'SpecRisk')
```


Daily return pre-winsorization
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
25
50
75
100
125
150
175
200


Daily return winsorized
20
15
10
5
0
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20



[6]:
```
def get_estu(df):
    """Estimation universe definition"""
    estu = df.loc[df.IssuerMarketCap > 1e9].copy(deep=True)
    return estu
def colnames(X):
    """ return names of columns, for DataFrame or DesignMatrix """
    if(type(X) == patsy.design_info.DesignMatrix):
        return(X.design_info.column_names)
    if(type(X) == pandas.core.frame.DataFrame):
        return(X.columns.tolist())
    return(None)
```

[7]:
```
def diagonal_factor_cov(date, X):
    """Factor covariance matrix, ignoring off-diagonal for simplicity"""
    cv = covariance[date]
    k = np.shape(X)[1]
    Fm = np.zeros([k,k])
    for j in range(0,k):
        fac = colnames(X)[j]
        Fm[j,j] = (0.01**2) * cv.loc[(cv.Factor1==fac) & (cv.
 ↪Factor2==fac),"VarCovar"].iloc[0]
    return(Fm)

def risk_exposures(estu):
```

```
    """Exposure matrix for risk factors, usually called X in class"""
    L = ["0"]
    L.extend(style_factors)
    L.extend(industry_factors)
    my_formula = " + ".join(L)
    return patsy.dmatrix(my_formula, data = estu)
```

Problem 0

Using the get_estu method provided in the sample, we removed all non-estimation-universe rows. This action of modifying the data frame helps prepare for the following data manipulation.

[8]:
```
#problem 0
for key in frames:
    frames[key] = get_estu(frames[key])
```

Problem 1

For the existing daily dataframe, we calculated a new variable called Y, which is the residual of the variable. The formula is as the following:

$$Y = \text{Ret} - XX^{+}\text{Ret}$$

This new variable is calculated and appended with the dataframe. At the same time, we winsorize the Ret column in order to transform statistics by limiting extreme values in the statistical data to reduce the effect of possibly spurious outliers.

[9]:
```
#problem 1
for key in frames:
    frames[key]['Ret'] = wins(frames[key]['Ret'],-0.2,0.2)
    X = risk_exposures(frames[key])
    X_pinv = np.linalg.pinv(X)
    frames[key]['Y'] = frames[key]['Ret'] - np.dot(np.dot(X, X_pinv),␣
 ↪frames[key]['Ret'])
```

Problem 2

Before proceeding to the model selection part, we conduct a train-test split for the data. We let the training data to be 80% of the overall data while testing data accounts for the rest 20%. This paves the road for the following work in the cross validation part. At the same time, we vertically combine all the training data in order to make the following data analysis and model selection more efficient.

[10]:
```
#problem 2

from sklearn.model_selection import train_test_split

sorted_dates = sorted(frames.keys())
train_size = 0.8
train_dates, test_dates = train_test_split(sorted_dates, train_size=train_size,␣
 ↪shuffle=False)

train_data = [frames[date] for date in train_dates]
test_data = [frames[date] for date in test_dates]
panel = pd.concat(train_data)

candidate_alphas = [
'STREVRSL', 'LTREVRSL', 'INDMOM',
'EARNQLTY', 'EARNYILD', 'MGMTQLTY', 'PROFIT', 'SEASON', 'SENTMT']

X_train = panel[candidate_alphas]
y_train = panel['Y']
```

Condider list of 9 candudate alphas above. We try to find a linear model of the form

$$Y = f(\text{candidate alphas}) + \epsilon$$

Then we use cross-validation to optimize any tunable hyper-parameters. An instance of the Lasso regression model is created as 'model'. A grid search is performed using 'GridSearchCV' with 5-fold cross-validation and negative mean squared error as the scoring metric. The resulting grid_search object provides information about the best hyperparameter values for the Lasso model based on the specified alpha values.

The grid_search object is then fitted to the training data (X_train and y_train), enabling the determination of the best hyperparameter values based on the specified alpha values. This process facilitates the optimization of the Lasso regression model for the given financial data.

[11]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet, Lasso

model = Lasso()
```

```
param_grid = {
    'alpha': np.logspace(-5, 0, 50),
}

grid_search = GridSearchCV(model, param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

[11]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': array([1.00000000e-05, 1.26485522e-05,
1.59985872e-05, 2.02358965e-05,
       2.55954792e-05, 3.23745754e-05, 4.09491506e-05, 5.17947468e-05,
       6.55128557e-05, 8.28642773e-05, 1.04811313e-04, 1.32571137e-04,
       1.67683294e-04, 2.12095089e-04, 2.68269580e-04, 3.39322177e-04,
       4.29193426e-04, 5.42867544e-04, 6.86648845e-04, 8…
       7.19685673e-03, 9.10298178e-03, 1.15139540e-02, 1.45634848e-02,
       1.84206997e-02, 2.32995181e-02, 2.94705170e-02, 3.72759372e-02,
       4.71486636e-02, 5.96362332e-02, 7.54312006e-02, 9.54095476e-02,
       1.20679264e-01, 1.52641797e-01, 1.93069773e-01, 2.44205309e-01,
       3.08884360e-01, 3.90693994e-01, 4.94171336e-01, 6.25055193e-01,
       7.90604321e-01, 1.00000000e+00])},
             scoring='neg_mean_squared_error')
```

Then, we try one non-linear functional form for , again using cross-validation to optimize any tunable hyper-parameters.

Our work begins by filtering the frames dictionary based on the condition that the date is earlier than '20040701'. After filtering, we use new features (X_train_filtered) and the target variable (y_train_filtered) to find the non-linear model.

The core of the model involves configuring a machine learning pipeline that integrates polynomial features and elastic net regression. A parameter grid is defined to explore various degrees for polynomial features and l1 ratios for elastic net regression. The model establishes cross-validation with k-fold cross-validation, utilizing 5 splits, shuffling, and a specified random seed.

A grid search (GridSearchCV) indicates a potential search for optimal hyperparameters, the actual execution is replaced with the loading of a pre-trained result from a file named 'grid_search_non_linear.pkl'.

[12]:
```
#non-linear

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import ElasticNetCV
```

[151]:
```
# half year data
frames_filtered = {date: frames[date] for date in frames if date < '20040701'}

panel_filtered = pd.concat(frames_filtered.values())

candidate_alphas = ['STREVRSL', 'LTREVRSL', 'INDMOM', 'EARNQLTY', 'EARNYILD',␣
 ↪'MGMTQLTY', 'PROFIT', 'SEASON', 'SENTMT']
X_train_filtered = panel_filtered[candidate_alphas]
y_train_filtered = panel_filtered['Y']
```

[152]:
```
param_grid = {
    'polynomialfeatures__degree': [2, 3],
    'elasticnetcv__l1_ratio': [0.1, 0.5, 0.7, 0.9]
}


pipeline = make_pipeline(
    PolynomialFeatures(include_bias=False),
    ElasticNetCV(cv=5)
)

kf = KFold(n_splits=5, shuffle=True, random_state=42)
#grid_search_non_linear = GridSearchCV(pipeline, param_grid, cv=kf,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)
#grid_search_non_linear.fit(X_train_filtered, y_train_filtered)
```

[152]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
             estimator=Pipeline(steps=[('polynomialfeatures',
                                        PolynomialFeatures(include_bias=False)),
                                       ('elasticnetcv', ElasticNetCV(cv=5))]),
             n_jobs=-1,
             param_grid={'elasticnetcv__l1_ratio': [0.1, 0.5, 0.7, 0.9],
                         'polynomialfeatures__degree': [2, 3]},
             scoring='neg_mean_squared_error')
```

[156]:
```
import joblib
#joblib.dump(grid_search_non_linear, 'grid_search_non_linear.pkl')

#load pre-trained result
grid_search_non_linear = joblib.load("grid_search_non_linear.pkl")
```

[157]:
```
print("Best parameters:", grid_search_non_linear.best_params_)
print('-'*60)
best_model = grid_search_non_linear.best_estimator_
poly_features = best_model.named_steps['polynomialfeatures']
elasticnet_cv = best_model.named_steps['elasticnetcv']
coefficients = elasticnet_cv.coef_
```

```
for i, (name, coef) in enumerate(zip(X_train_filtered.columns, coefficients)):
    print(f"Factor: {name}, Degree: {poly_features.powers_[i]+1}, Coefficient:␣
 ↪{coef}")
```

```
Best parameters: {'elasticnetcv__l1_ratio': 0.9, 'polynomialfeatures__degree':
2}
------------------------------------------------------------
Factor: STREVRSL, Degree: [2 1 1 1 1 1 1 1 1], Coefficient:
0.00024400727814477536
Factor: LTREVRSL, Degree: [1 2 1 1 1 1 1 1 1], Coefficient: -0.0
Factor: INDMOM, Degree: [1 1 2 1 1 1 1 1 1], Coefficient: 9.8959767171312e-06
Factor: EARNQLTY, Degree: [1 1 1 2 1 1 1 1 1], Coefficient: -0.0
Factor: EARNYILD, Degree: [1 1 1 1 2 1 1 1 1], Coefficient: 0.0
Factor: MGMTQLTY, Degree: [1 1 1 1 1 2 1 1 1], Coefficient: 0.0
Factor: PROFIT, Degree: [1 1 1 1 1 1 2 1 1], Coefficient: 3.227932948507696e-06
Factor: SEASON, Degree: [1 1 1 1 1 1 1 2 1], Coefficient: 3.382754441745144e-05
Factor: SENTMT, Degree: [1 1 1 1 1 1 1 1 2], Coefficient: 3.813992051523156e-05
```

Problem 3

In order to be computationally efficient in calculating the inverse of the covariance of residuals, we build the inverse function based on the Woodbury matrix inversion lemma,

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

where

$$A = D, U = X, V = X', C = F$$

To construct our optimal portfolio, we obtain our dollar holdings of the portfolio according the optimal holding formulas in the lecture notes:

$$h^* = (\kappa\Sigma)^{-1}E[r]$$

where κ is a scalar factors to include risk aversion, Σ^{-1} is the inverse covariance of the return, and $E[r]$ is the expected returns.

In this project, we set

$$\kappa = 1e - 5$$

[134]:
```
#problem 3
def woodbury_inversion(A, U, C, V):
    A_inv = np.linalg.inv(A)
    C_inv = np.linalg.inv(C)
    return A_inv - A_inv @ U @ np.linalg.inv(C_inv + V @ A_inv @ U) @ V @ A_inv

def portfolio_weights(cov_inv, expected_returns, kappa = 1e-5):
    return np.dot(1/kappa*cov_inv, expected_returns)
```

Problem 4

For backtesting our portfolio optimization strategy, we follow the instruction to evaluate the strategy based on its profit, which is computed from the dot product of optimal holdings with Ret. In order to compute the optimal holdings, we first need to determine variables needed, $ Σ^{-1} $ and $E[r]$.

Here,

$$\Sigma^{-1} = D^{-1} - D^{-1}X(F^{-1} + X'D^{-1}X)^{-1}X'D^{-1}$$

follows the Woodbury matrix inversion; Also, X,D,F is obtained as suggested in the instruction.

$$E(r) = x \cdot \mu_f$$

where x is known from our candidate alphas. For μ_f, we set it to be the average of factor returns in the test sets in this project, as an approximate expected factor returns.

With all the parameters, we determine the optimal holdings. Note, we normalized h to be summed to 1.

For deriving daily risks and idiosyncratic percentages, we follow

$$h'\Sigma h = h'Dh + h'XFX'h$$

which expresses the portfolio's variance in terms of the idiosyncratic variance $h'Dh$, and

idiosyncratic percentages $= \frac{h'Dh}{h'\Sigma h}$.

For Long/Short market values, we consider Ret as a reflective variable of Market values of each asset, and consider the sign of h as an indicator of long/short position.

[135]:
```
#problem 4
f_hat_list = []
cov_inv_list = []
F_list = []
for my_date in test_dates:
    X = frames[my_date]
    x = X[candidate_alphas]
    n = len(x)
    F = diagonal_factor_cov(my_date, x)
    F_list.append(F)
    X_ = np.asarray(x)
    D = np.diag(np.asarray((X['SpecRisk'] / (100 * math.sqrt(252))) ** 2))
    cov_inv = woodbury_inversion(D, X_, F, X_.T)
    f_hat = np.linalg.pinv(x.T @ x) @ x.T @ X['Ret']
    f_hat = np.array(f_hat).reshape(1,len(f_hat))
    f_hat_list.append(f_hat)
    cov_inv_list.append(cov_inv)
```

```
miu_f = sum(f_hat_list)/len(f_hat_list)
```

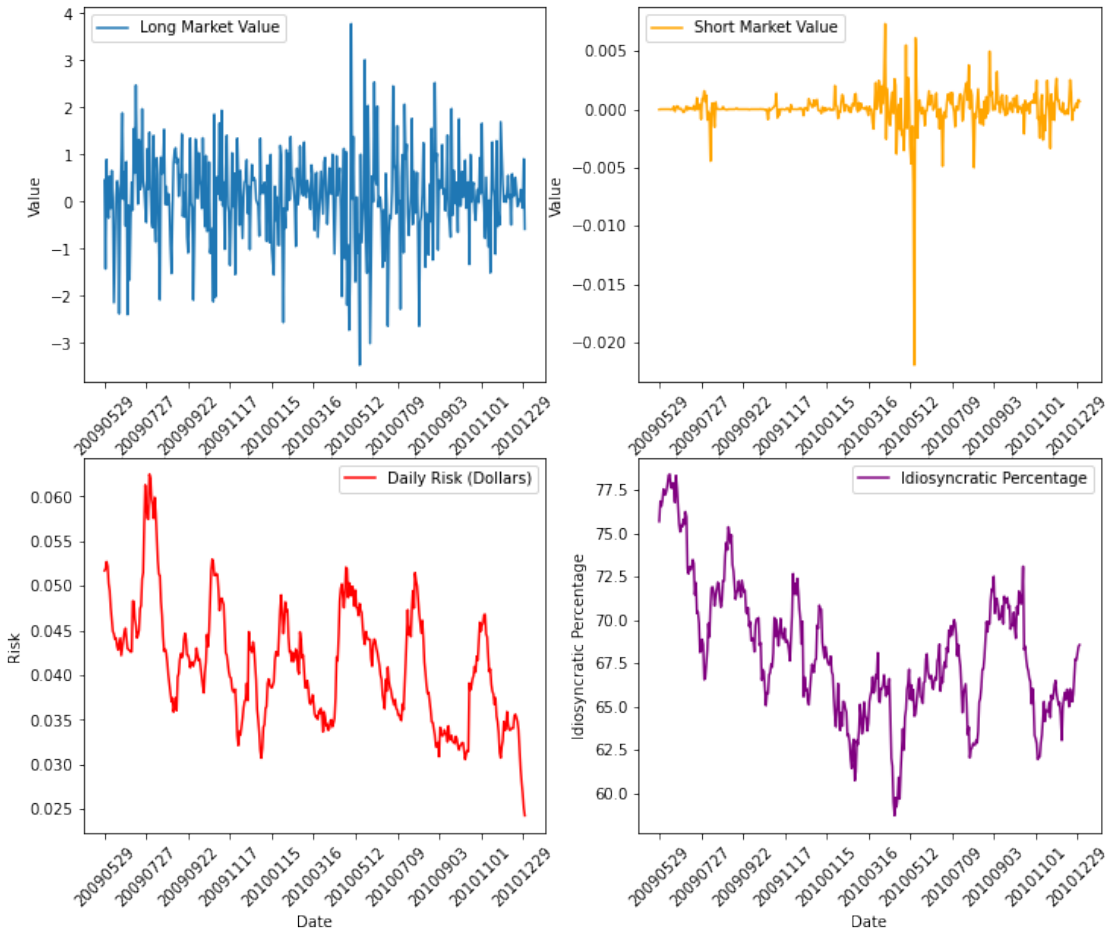
[149]:
```
#linear model
long_market_values = []
short_market_values = []
cumulative_profit = 0
daily_risks = []
idiosyncratic_percentage = []
profit = []
for idx, my_date in enumerate(test_dates):
    X = frames[my_date]
    x = X[candidate_alphas]
    n = len(x)
    F = F_list[idx]
    X_ = np.asarray(x)
    D = np.diag(np.asarray((X['SpecRisk'] / (100 * math.sqrt(252))) ** 2))
    cov_inv = cov_inv_list[idx]
    residual = grid_search.best_estimator_.predict(x)
    expected_returns = np.array(miu_f @ x.T + residual).reshape(n,)
    h = portfolio_weights(cov_inv, expected_returns)
    h = h/sum(h)*100
    long = [i if i > 0 else 0 for i in h] @ X['Ret']
    short = [-i if i < 0 else 0 for i in h] @ X['Ret']
    # Calculate daily risk in dollars
    daily_risk = np.sqrt(h @ D @ h.T + h.T @ X_ @ F @ X_.T @ h)

    # Calculate the percentage of risk that is idiosyncratic
    idiosyncratic_risk = np.sqrt(h @ D @ h.T)

    # Append results to lists for tracking
    long_market_values.append(long)
    short_market_values.append(short)
    cumulative_profit += np.dot(h, X['Ret'])
    profit.append(cumulative_profit)
    daily_risks.append(daily_risk)
    idiosyncratic_percentage.append(idiosyncratic_risk/daily_risk*100)
```

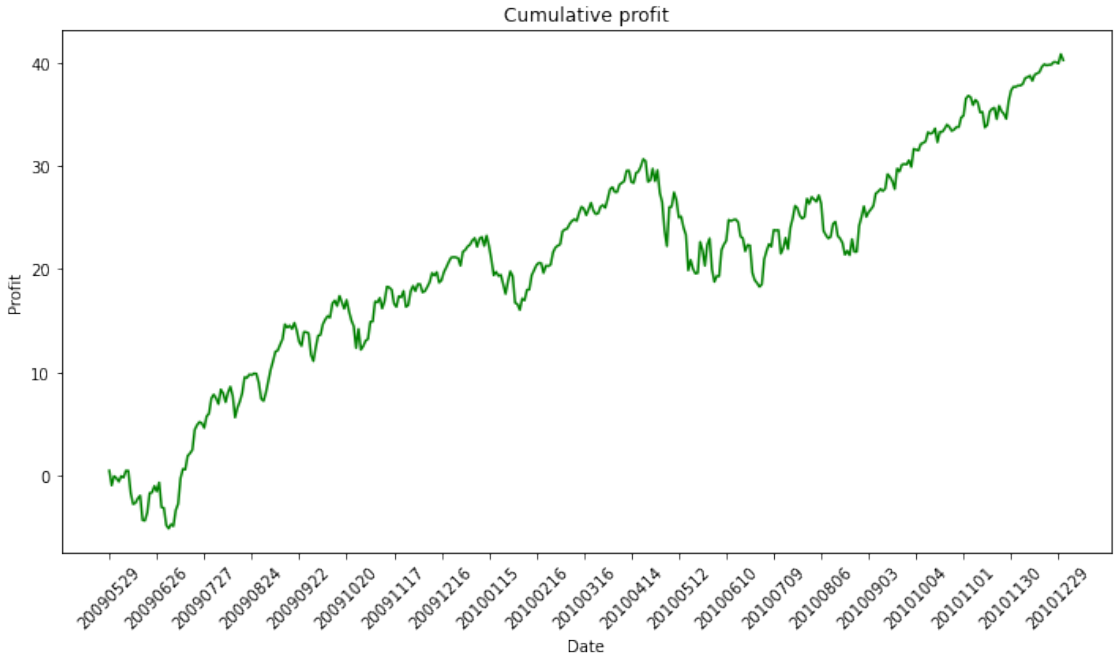
[158]:
```
#non-linear model
long_market_values2 = []
short_market_values2 = []
cumulative_profit = 0
daily_risks2 = []
idiosyncratic_percentage2 = []
profit2 = []
for idx, my_date in enumerate(test_dates):
    X = frames[my_date]
    x = X[candidate_alphas]
```

```
    n = len(x)
    F = F_list[idx]
    X_ = np.asarray(x)
    D = np.diag(np.asarray((X['SpecRisk'] / (100 * math.sqrt(252))) ** 2))
    cov_inv = cov_inv_list[idx]
    residual = grid_search_non_linear.best_estimator_.predict(x)
    expected_returns = np.array(miu_f @ x.T + residual).reshape(n,)
    h = portfolio_weights(cov_inv, expected_returns)
    h = h/sum(h)*100
    long = [i if i > 0 else 0 for i in h] @ X['Ret']
    short = [-i if i < 0 else 0 for i in h] @ X['Ret']
    # Calculate daily risk in dollars
    daily_risk = np.sqrt(h @ D @ h.T + h.T @ X_ @ F @ X_.T @ h)

    # Calculate the percentage of risk that is idiosyncratic
    idiosyncratic_risk = np.sqrt(h @ D @ h.T)

    # Append results to lists for tracking
    long_market_values2.append(long)
    short_market_values2.append(short)
    cumulative_profit += np.dot(h, X['Ret'])
    profit2.append(cumulative_profit)
    daily_risks2.append(daily_risk)
    idiosyncratic_percentage2.append(idiosyncratic_risk/daily_risk*100)
```

[165]:
```
#plots for linear model
plt.figure(figsize=(12, 10))

# Plot long market value
plt.subplot(2, 2, 1)
plt.plot(test_dates, long_market_values, label='Long Market Value')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()

# Plot short market value
plt.subplot(2, 2, 2)
plt.plot(test_dates, short_market_values, label='Short Market Value',␣
 ↪color='orange')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()

# Plot daily risk
plt.subplot(2, 2, 3)
```

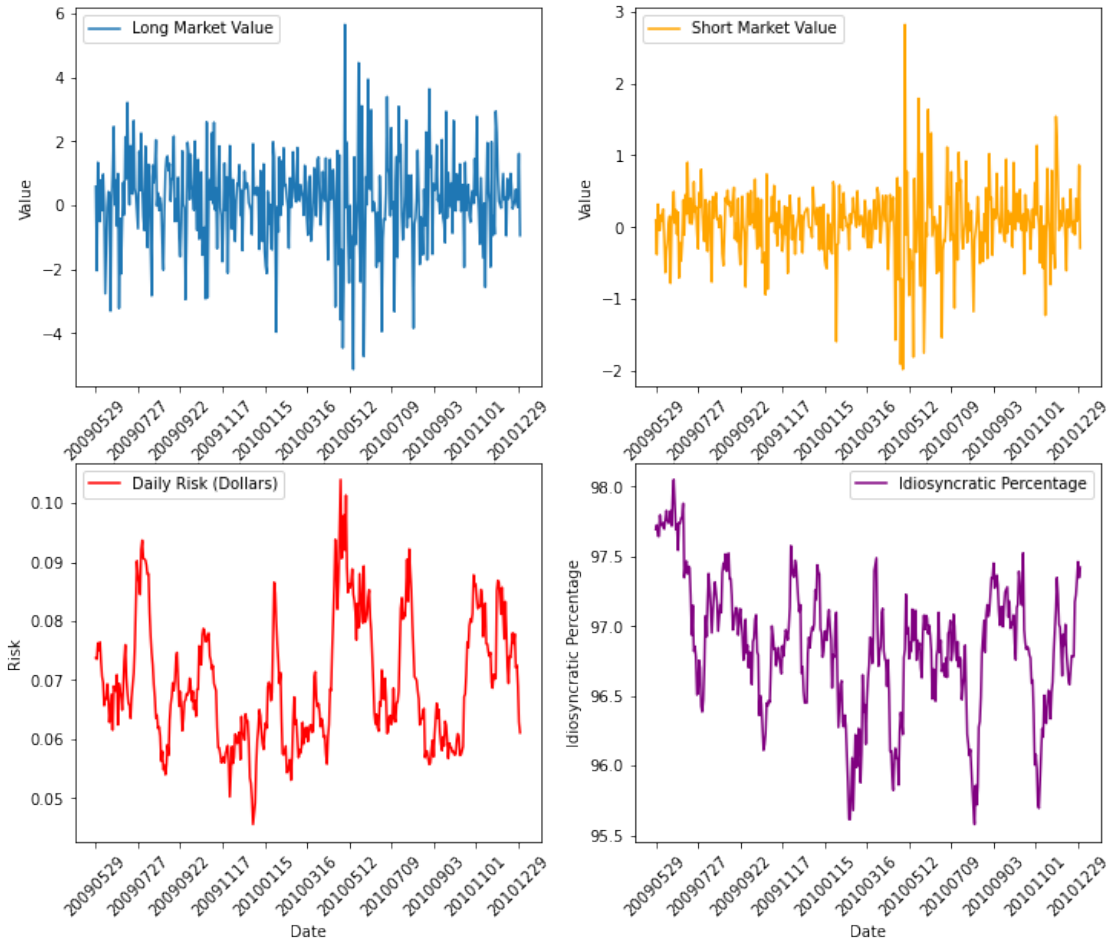
```
plt.plot(test_dates, daily_risks, label='Daily Risk (Dollars)', color='red')
plt.xlabel('Date')
plt.ylabel('Risk')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()

# Plot idiosyncratic percentage
plt.subplot(2, 2, 4)
plt.plot(test_dates, idiosyncratic_percentage, label='Idiosyncratic␣
 ↪Percentage', color='purple')
plt.xlabel('Date')
plt.ylabel('Idiosyncratic Percentage')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()


# Plot cumulative profit
plt.figure(figsize=(10, 6))
plt.plot(test_dates, profit, label='Cumulative Profit', color='green')
plt.title('Cumulative profit')
plt.xlabel('Date')
plt.ylabel('Profit')
plt.xticks(test_dates[::20], rotation = 45)


plt.tight_layout()
plt.show()
```

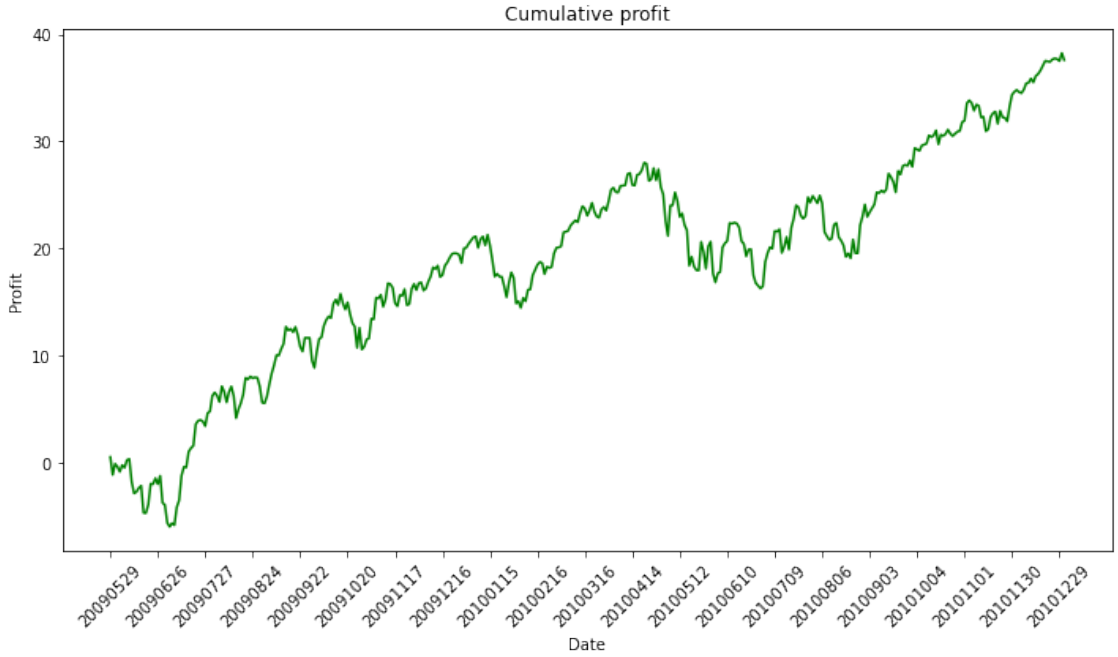

Long Market Value
Value
Short Market Value
Value
Daily Risk (Dollars)
Risk
Idiosyncratic Percentage
Idiosyncratic Percentage
Date
Date


Cumulative profit
Profit
Date

[166]:
```
#plots for non-linear model
plt.figure(figsize=(12, 10))

# Plot long market value
plt.subplot(2, 2, 1)
plt.plot(test_dates, long_market_values2, label='Long Market Value')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()

# Plot short market value
plt.subplot(2, 2, 2)
plt.plot(test_dates, short_market_values2, label='Short Market Value',␣
 ↪color='orange')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()

# Plot daily risk
plt.subplot(2, 2, 3)
plt.plot(test_dates, daily_risks2, label='Daily Risk (Dollars)', color='red')
plt.xlabel('Date')
plt.ylabel('Risk')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()

# Plot idiosyncratic percentage
plt.subplot(2, 2, 4)
plt.plot(test_dates, idiosyncratic_percentage2, label='Idiosyncratic␣
 ↪Percentage', color='purple')
plt.xlabel('Date')
plt.ylabel('Idiosyncratic Percentage')
plt.xticks(test_dates[::40], rotation = 45)
plt.legend()


# Plot cumulative profit
plt.figure(figsize=(10, 6))
plt.plot(test_dates, profit2, label='Cumulative Profit', color='green')
plt.title('Cumulative profit')
plt.xlabel('Date')
plt.ylabel('Profit')
```

```
plt.xticks(test_dates[::20], rotation = 45)

plt.tight_layout()
plt.show()
```




Cumulative profit
Profit
Date
40
30
20
10
0
20090529
20090626
20090727
20090824
20090922
20091020
20091117
20091216
20100115
20100216
20100316
20100414
20100512
20100610
20100709
20100806
20100903
20101004
20101101
20101130
20101229