Team members: Ethan Chang, Alessandro Castillo

```
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
from sklearn.model_selection import train_test_split
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

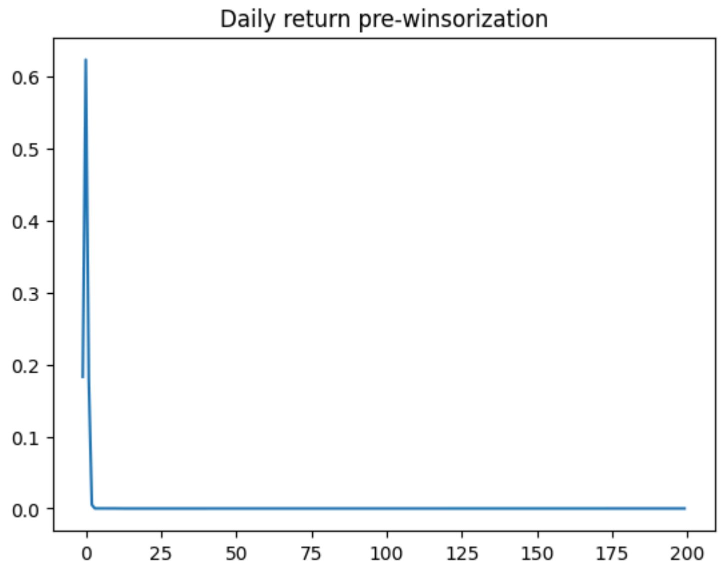
```
model_dir = './drive/MyDrive/FACTOR_MODEL/'

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

frames = {}

for year in [2004,2005,2006]:
  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]:
  fil = model_dir + "covariance." + str(year) + ".pickle.bz2"
  covariance.update(pd.read_pickle(fil))
```

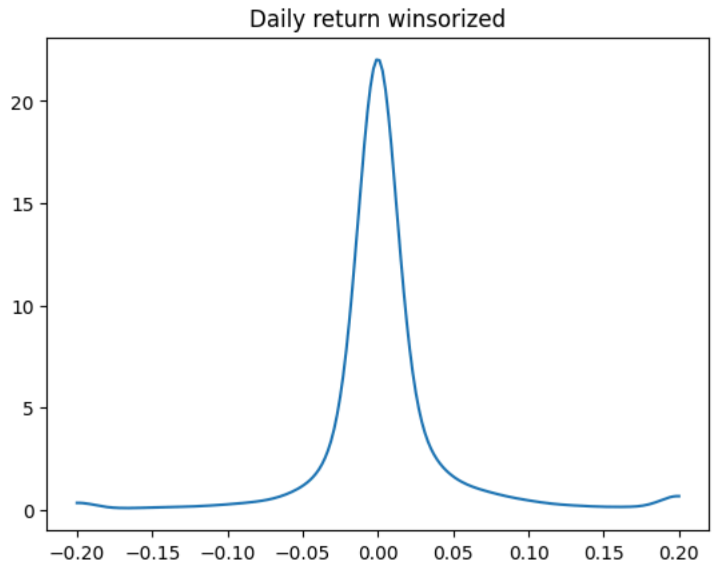
```
industry_factors = ['AERODEF', 'AIRLINES', 'ALUMSTEL', 'APPAREL', 'AUTO',
'BANKS','BEVTOB', 'BIOLIFE', 'BLDGPROD','CHEM', 'CNSTENG',
'CNSTMACH', 'CNSTMATL', 'COMMEQP', 'COMPELEC',
'COMSVCS', 'CONGLOM', 'CONTAINR', 'DISTRIB',
'DIVFIN', 'ELECEQP', 'ELECUTIL', 'FOODPROD', 'FOODRET', 'GASUTIL',
'HLTHEQP', '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']
```

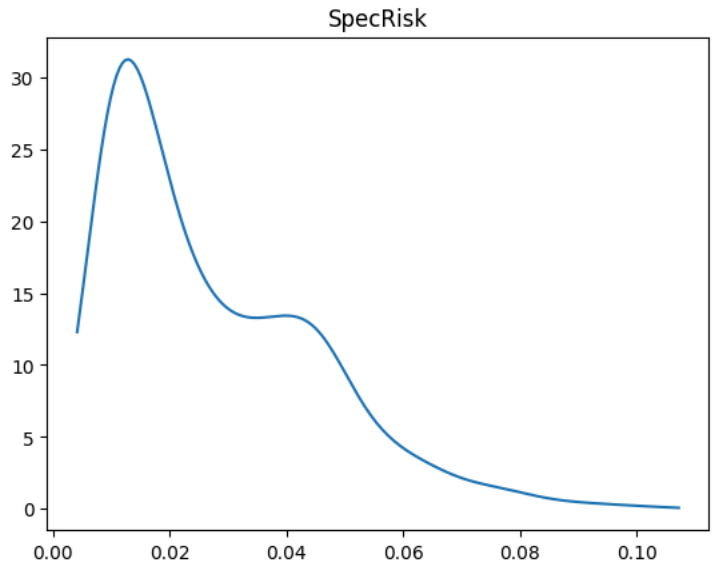
```
def wins(x,a,b):
  return(np.where(x <= a,a, np.where(x >= b, b, x)))

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
```

```
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')
```







```
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)

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)



my_date = '20040102'

# estu = estimation universe
estu = get_estu(frames[my_date])
estu['Ret'] = wins(estu['Ret'], -0.25, 0.25)

rske = risk_exposures(estu)
F = diagonal_factor_cov(my_date, rske)
X = np.asarray(rske)
D = np.asarray( (estu['SpecRisk'] / (100 * math.sqrt(252))) ** 2 )

kappa = 1e-5

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


F, X, D

D.shape
```

```
(2265,)
```

Problem 0

```
  for date in frames.keys():
      estu = get_estu(frames[date])
      rske = risk_exposures(estu)
      rske_indices = rske.design_info.column_names
      estu = estu.iloc[:len(rske_indices),:]



  frames['20040102']
```

	1DREVRSL	AERODEF	AIRLINES	ALUMSTEL	APPAREL	AUTO	BANKS	BETA	BEVTOB	BIOLIFE	...	SPTYSTOR	STREVRSL	SpecRisk	TE
0	-0.032	0.0	0.0	0.0	0.0	0.0	0.00	-2.177	0.0	0.0	...	0.0	0.548	9.014505	
1	0.684	0.0	0.0	0.0	0.0	0.0	0.00	-2.045	0.0	0.0	...	0.0	0.986	19.304651	
2	0.235	0.0	0.0	0.0	0.0	0.0	0.00	-2.010	0.0	0.0	...	0.0	-0.256	19.802556	
3	0.759	0.0	0.0	0.0	0.0	0.0	0.00	-0.948	0.0	0.0	...	0.0	1.841	31.274403	
4	0.674	0.0	0.0	0.0	0.0	0.0	0.00	-2.148	0.0	0.0	...	0.0	-0.588	13.533480	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12430	-0.953	0.0	0.0	0.0	0.0	0.0	0.97	-2.009	0.0	0.0	...	0.0	0.430	9.095567	
12431	-0.808	0.0	0.0	0.0	0.0	0.0	0.97	-2.066	0.0	0.0	...	0.0	0.561	8.739695	
12432	-0.754	0.0	0.0	0.0	0.0	0.0	0.97	-2.053	0.0	0.0	...	0.0	0.734	9.565838	
12433	-0.647	0.0	0.0	0.0	0.0	0.0	0.97	-2.022	0.0	0.0	...	0.0	0.680	9.394557	
12434	0.331	0.0	0.0	0.0	0.0	0.0	0.00	-1.985	0.0	0.0	...	0.0	-0.519	9.591641	

12435 rows × 93 columns

Problem 1

```
def residual_returns(frames):
  for date, frame in frames.items():
    estu = get_estu(frame)
    estu['Ret'] = wins(estu['Ret'], -0.25, 0.25)
    rske = risk_exposures(estu)
    X = np.asarray(rske)
    X_pseudoinverse = np.linalg.pinv(X)
    estu['Y'] = estu['Ret'] - np.dot(X, np.dot(X_pseudoinverse, estu['Ret']))

    frames[date] = estu

residual_returns(frames)
```

Problem 2

```
train_dates = [date for date in frames.keys() if int(date) < 20060101]
test_dates = [date for date in frames.keys() if int(date) >= 20060101]

train_frames = pd.concat([frames[date] for date in train_dates])
test_frames = pd.concat([frames[date] for date in test_dates])

candidate_alphas = ['STREVRSL', 'LTREVRSL', 'INDMOM', 'EARNQLTY', 'EARNYILD',
                    'MGMTQLTY', 'PROFIT', 'SEASON', 'SENTMT']
X_train = train_frames[candidate_alphas]
y_train = train_frames['Y']
X_test = test_frames[candidate_alphas]
y_test = test_frames['Y']


from sklearn.linear_model import ElasticNetCV
from sklearn.metrics import mean_squared_error

# cross validation, k=5
elastic_net = ElasticNetCV(cv=5, l1_ratio=[0.1, 0.5, 0.9], random_state=42)
elastic_net.fit(X_train, y_train)


y_pred_elastic = elastic_net.predict(X_test)
mse_elastic = mean_squared_error(y_test, y_pred_elastic)

print("Elastic Net MSE:", mse_elastic)
```

```
Elastic Net MSE: 0.0002876524424251106
```

```
from sklearn.model_selection import TimeSeriesSplit
# learned about from https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np

# time series split for cross validation, k=5
tscv = TimeSeriesSplit(n_splits=5)
cv_mse = []

for train_index, val_index in tscv.split(X_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    scaler = StandardScaler()
    X_train_fold = scaler.fit_transform(X_train_fold)
    X_val_fold = scaler.transform(X_val_fold)

    # neural network
    model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train_fold.shape[1],)),
        Dense(32, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

    model.fit(X_train_fold, y_train_fold, validation_data=(X_val_fold, y_val_fold),
              epochs=10, batch_size=32, verbose=0)

    y_val_pred = model.predict(X_val_fold)
    cv_mse.append(mean_squared_error(y_val_fold, y_val_pred))

print("Average CV MSE:", np.mean(cv_mse))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6374/6374 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6374/6374 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6374/6374 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6374/6374 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6374/6374 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
Average CV MSE: 0.0002682627590003899
```

Problem 3

```
def portfolio_optimization(mu, D, X, F, kappa):
    D_inverse = np.linalg.inv(D)
    F_inverse = np.linalg.inv(F)

    Sigma_inv = D_inverse - D_inverse @ X @ np.linalg.inv(F_inverse + X.T @ D_inverse @ X) @ X.T @ D_inverse

    h_star = (1 / kappa) * Sigma_inv @ mu
    return h_star
```

Problem 4

```
def backtest(frames, model, kappa):
    cumulative_profit = 0
    cumulative_profits = []
    daily_profits = []
    daily_profits = []
    long_market_values = []
    short_market_values = []
    daily_risks = []
    idiosyncratic_risks = []

    for date, frame in frames.items():
        if int(date) < 20060101:
          continue
        else:
          print(date)

        estu = get_estu(frame)
        rske = risk_exposures(estu)
        X = np.asarray(rske)
        D = np.diag(np.asarray( (estu['SpecRisk'] / (100 * math.sqrt(252))) ** 2 ))
        Ret = frame['Ret'].values
        alphas = frame[candidate_alphas]
        mu = model.predict(alphas)
        F =diagonal_factor_cov(date, rske)
        h_star = portfolio_optimization(mu, D, X, F, kappa=kappa)


        daily_profit = np.dot(h_star, Ret)
        cumulative_profit += daily_profit
        long_market_value = np.sum(h_star[h_star > 0])
        short_market_value = np.sum(h_star[h_star < 0])
        portfolio_variance = h_star.T @ (D + X @ F @ X.T) @ h_star
        daily_risk = np.sqrt(portfolio_variance)
        idiosyncratic_variance = h_star.T @ D @ h_star
        idiosyncratic_risk_percent = (idiosyncratic_variance / portfolio_variance) * 100

        daily_profits.append(daily_profit)
        cumulative_profits.append(cumulative_profit)
        long_market_values.append(long_market_value)
        short_market_values.append(short_market_value)
        daily_risks.append(daily_risk)
        idiosyncratic_risks.append(idiosyncratic_risk_percent)

    return {
        "cumulative_profits": cumulative_profits,
        "daily_profits": daily_profits,
        "long_market_values": long_market_values,
        "short_market_values": short_market_values,
        "daily_risks": daily_risks,
        "idiosyncratic_risks": idiosyncratic_risks,
    }

graphing = backtest(frames, elastic_net, kappa)


dates = range(len(graphing["cumulative_profits"]))
cumulative_profit = graphing["cumulative_profits"]
daily_profit = graphing["daily_profits"]
daily_risk = graphing["daily_risks"]
idiosyncratic_risk_percent = graphing["idiosyncratic_risks"]
long_market_values = graphing["long_market_values"]
short_market_values = graphing["short_market_values"]

plt.figure(figsize=(10, 6))
plt.plot(dates, cumulative_profit, label="Cumulative Profit")
plt.title("Cumulative Profit")
plt.xlabel("Time (Days)")
plt.ylabel("Cumulative Profit")
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(dates, daily_risk, label="Daily Risk")
plt.title("Daily Portfolio Risk")
plt.xlabel("Time (Days)")
plt.ylabel("Risk")
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(dates, idiosyncratic_risk_percent, label="Idiosyncratic Risk (%)")
```

```
    plt.plot(dates, idiosyncratic_risk_percent, label="Idiosyncratic Risk (%)")
    plt.title("Idiosyncratic Risk Percentage")
    plt.xlabel("Time (Days)")
    plt.ylabel("Idiosyncratic Risk (%)")
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(10, 6))
    plt.plot(dates, long_market_values, label="Long Market Value")
    plt.title("Long Market Values")
    plt.xlabel("Time (Days)")
    plt.ylabel("Market Value")
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(10, 6))
    plt.plot(dates, short_market_values, label="Short Market Value")
    plt.title("Short Market Values")
    plt.xlabel("Time (Days)")
    plt.ylabel("Market Value")
    plt.legend()
    plt.grid(True)
    plt.show()
```