Preprocessing and Analysis

BUSI 722: Data-Driven Finance II

Kerry Back, Rice University

Outline

1. Build dataset of features, returns, and targets as before
2. Add preprocessing of features

- Standardize features relative to other stocks at the same date
- Add interactions of features

3. Train, predict, and form portfolios in loop as before
4. Interpret model

- Feature importances
- Shapley values
- Features of best and worst portfolios

5. Analyze portfolio returns

- Mean-variance frontiers of SPY, best, and worst.

- Build dataset of features, returns, and targets as before
- Add preprocessing of features
    - Features standardized relative to other stocks at the same date
    - Add interactions of features
- Interpret model
    - Feature importances
    - Shapley values
    - Features of best and worst portfolios
- Evaluate portfolio returns
    -

1. Create dataset as before

In [ ]:

```
import numpy as np
import pandas as pd
from sqlalchemy import create_engine
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
```

In [2]:

```python
server = 'fs.rice.edu'
database = 'stocks'
username = 'stocks'
password = '6LAZH1'
driver = 'SQL+Server'
string = f"mssql+pyodbc://{username}:{password}@{server}/{database}"
try:
    conn = create_engine(string + "?driver='SQL+Server'").connect()
except:
    try:
        conn = create_engine(string + "?driver='ODBC+Driver+18+for+SQL+Server
    except:
        import pymssql
        string = f"mssql+pymssql://{username}:{password}@{server}/{database}"
        conn = create_engine(string).connect()
```

In [3]:

```
sep_weekly = pd.read_sql(
    """
    select date, ticker, closeadj, closeunadj, volume, lastupdated from sep_we
    where date >= '2010-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
sep_weekly = sep_weekly.groupby(["ticker", "date"]).last()
sep_weekly = sep_weekly.drop(columns=["lastupdated"])

ret = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_change()
ret.name = "ret"

price = sep_weekly.closeunadj
price.name = "price"

volume = sep_weekly.volume
volume.name = "volume"
```

In [4]:

```
ret_annual = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_chan
ret_monthly = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_cha
mom = (1 + ret_annual) / (1 + ret_monthly) - 1
mom.name = "mom"
```

In [5]:

```
weekly = pd.read_sql(
    """
    select date, ticker, pb, marketcap, lastupdated from weekly
    where date>='2010-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
weekly = weekly.groupby(["ticker", "date"]).last()
weekly = weekly.drop(columns=["lastupdated"])

pb = weekly.pb
pb.name = "pb"
marketcap = weekly.marketcap
marketcap.name = "marketcap"
```

In [6]:

```
sf1 = pd.read_sql(
    """
    select datekey as date, ticker, assets, netinc, equity, lastupdated from
    where datekey>='2010-01-01' and dimension='ARY' and assets>0 and equity>0
    order by ticker, datekey, lastupdated
    """,
    conn,
)
sf1 = sf1.groupby(["ticker", "date"]).last()
sf1 = sf1.drop(columns=["lastupdated"])

# change dates to Fridays
from datetime import timedelta
sf1 = sf1.reset_index()
sf1.date =sf1.date.map(
    lambda x: x + timedelta(4 - x.weekday())
)
sf1 = sf1.set_index(["ticker", "date"])
sf1 = sf1[~sf1.index.duplicated()]

assets = sf1.assets
assets.name = "assets"
netinc = sf1.netinc
netinc.name = "netinc"
equity = sf1.equity
equity.name = "equity"

equity = equity.groupby("ticker", group_keys=False).shift()
roe = netinc / equity
```

In [7]:

```
df = pd.concat(
    (
        ret,
        mom,
        volume,
        price,
        pb,
        marketcap,
        roe,
        assetgr
        ),
        axis=1
    )
df["ret"] = df.groupby("ticker", group_keys=False).ret.shift(-1)
df["roe"] = df.groupby("ticker", group_keys=False).roe.ffill()
df["assetgr"] = df.groupby("ticker", group_keys=False).assetgr.ffill()
df = df[df.price >= 5]
df = df.dropna()

df = df.reset_index()
df.date = df.date.astype(str)
df = df[df.date >= "2012-01-01"]

df["target1"] = df.groupby("date", group_keys=False).ret.apply(
    lambda x: x - x.median()
)
df["target2"] = df.groupby("date", group_keys=False).ret.apply(
    lambda x: 100*x.rank(pct=True)
)
```

2. Preprocessing of Features

Features relative to peers

We are predicting relative performance. It makes sense to use relative features: how does a stock compare to other stocks at the same date? There are multiple options:

- standard scaler (subtract mean and divide by std dev)
- quantile transformer (map to normal or uniform distribution)
- rank (quantile transformer to uniform distribution)

In [23]:

```python
features = [
    "mom",
    "volume",
    "pb",
    "marketcap",
    "roe",
    "assetgr"
]

for f in features:
    df[f] = df.groupby("date", group_keys=False)[f].apply(
        lambda x: x.rank(pct=True)
    )
```

Interactions

It may be useful to include interactions of features (x1*x2 for all features x1 and x2). For example, a high value of x1 may predict a high return only if it is coupled with a high value of x2. We could add interactions manually to the dataframe but it is easier to use PolynomialFeatures.

Pipeline

We can put preprocessing steps that are to be applied to the entire training set in a pipeline with the model and fit and predict from the pipeline.

In [21]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

poly = PolynomialFeatures(degree=2, interaction_only=True)
model = RandomForestRegressor(max_depth=4)
pipe = make_pipeline(poly, model)
```

3. Train, predict and form portfolios as before

- Only change is to use fit and predict using the pipeline.

In [24]:

```
train_years = 5 # num years of past data to use for training
train_freq = 3  # num years between training
target = "target2"

years = range(2012+train_years, 2024, train_freq)
df2 = None
for i, year in enumerate(years):
    print(year)
    start_train = f"{year-train_years}-01-01"
    start_predict = f"{year}-01-01"
    if year == years[-1]:
        stop_predict = "2100-01-01"
    else:
        stop_predict = f"{years[i+1]}-01-01"
    past = df[(df.date >= start_train) & (df.date < start_predict)]
    future = df[(df.date>=start_predict) & (df.date<stop_predict)].copy()
    pipe.fit(X=past[features], y=past[target])
    future["predict"] = pipe.predict(X=future[features])
    df2 = pd.concat((df2, future))

df2.head()
```

```
2017
```

```
c:\Users\kerry\AppData\Local\Programs\Python\Python310\lib\site-packa
ges\sklearn\base.py:443: UserWarning: X has feature names, but Random
ForestRegressor was fitted without feature names
  warnings.warn(
```

In [10]:

```
num_stocks = 50

grouped = df2.groupby("date", group_keys=False).predict
starting_from_best = grouped.rank(ascending=False, method="first")
best = df2[starting_from_best <= num_stocks]
best_rets = best.groupby("date", group_keys=True).ret.mean()
best_rets.index = pd.to_datetime(best_rets.index)

starting_from_worst = grouped.rank(ascending=True, method="first")
worst = df2[starting_from_worst <= num_stocks]
worst_rets = worst.groupby("date", group_keys=True).ret.mean()
worst_rets.index = pd.to_datetime(worst_rets.index)

all_rets = df2.groupby("date", group_keys=True).ret.mean()
all_rets.index = pd.to_datetime(all_rets.index)
```

4. Interpret

Find feature importances for last trained model

In [12]:

```
importances = pd.Series(
    model.feature_importances_,
    index=features
)
importances.round(3)
```

Out[12]:

```
mom          0.041
volume       0.304
pb           0.029
marketcap    0.170
roe          0.456
assetgr      0.000
dtype: float64
```

Extract best, worst, and all stocks in last portfolios

In [13]:

```
last_date = df2.date.max()
best_last = best[best.date==last_date].copy()
worst_last = worst[worst.date==last_date].copy()
all_last = df2[df2.date==last_date].copy()

best_last["group"] = "best"
worst_last["group"] = "worst"
all_last["group"] = "all"

last = pd.concat((best_last, worst_last, all_last))
```

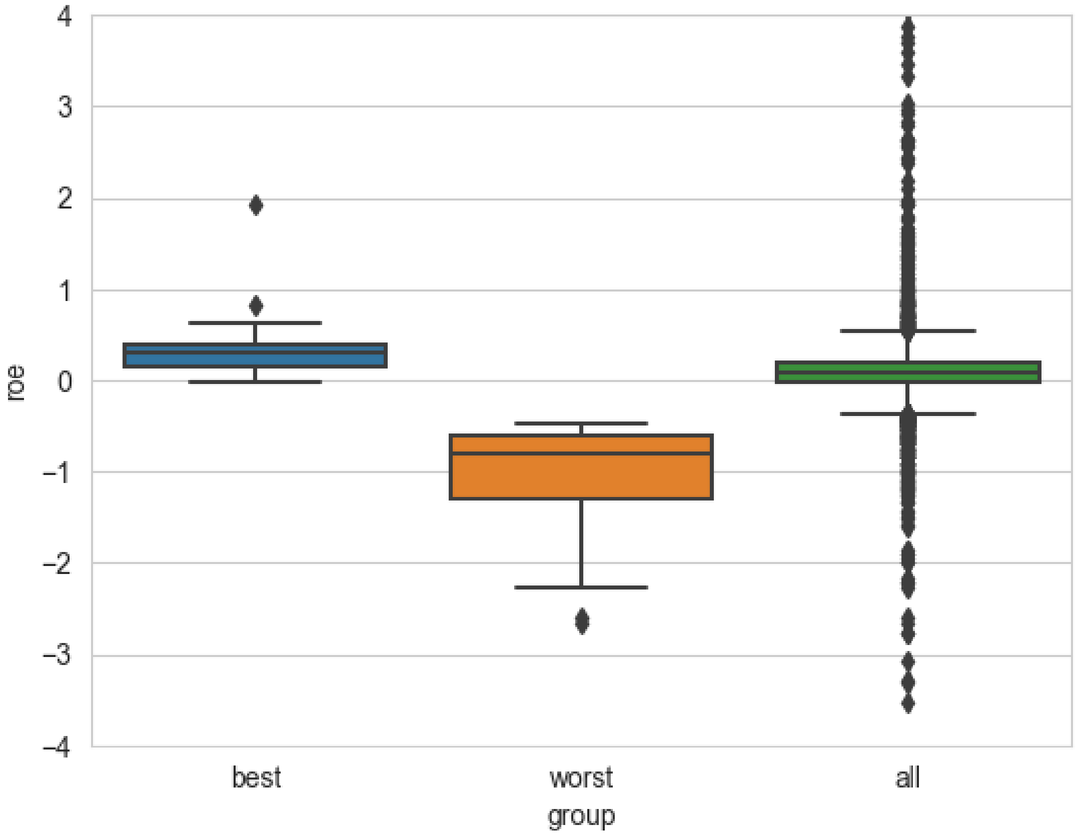
Compare features of best, worst, and all portfolios

In [14]:

```
sns.boxplot(last, x="group", y="roe")
plt.ylim((-4, 4))
```

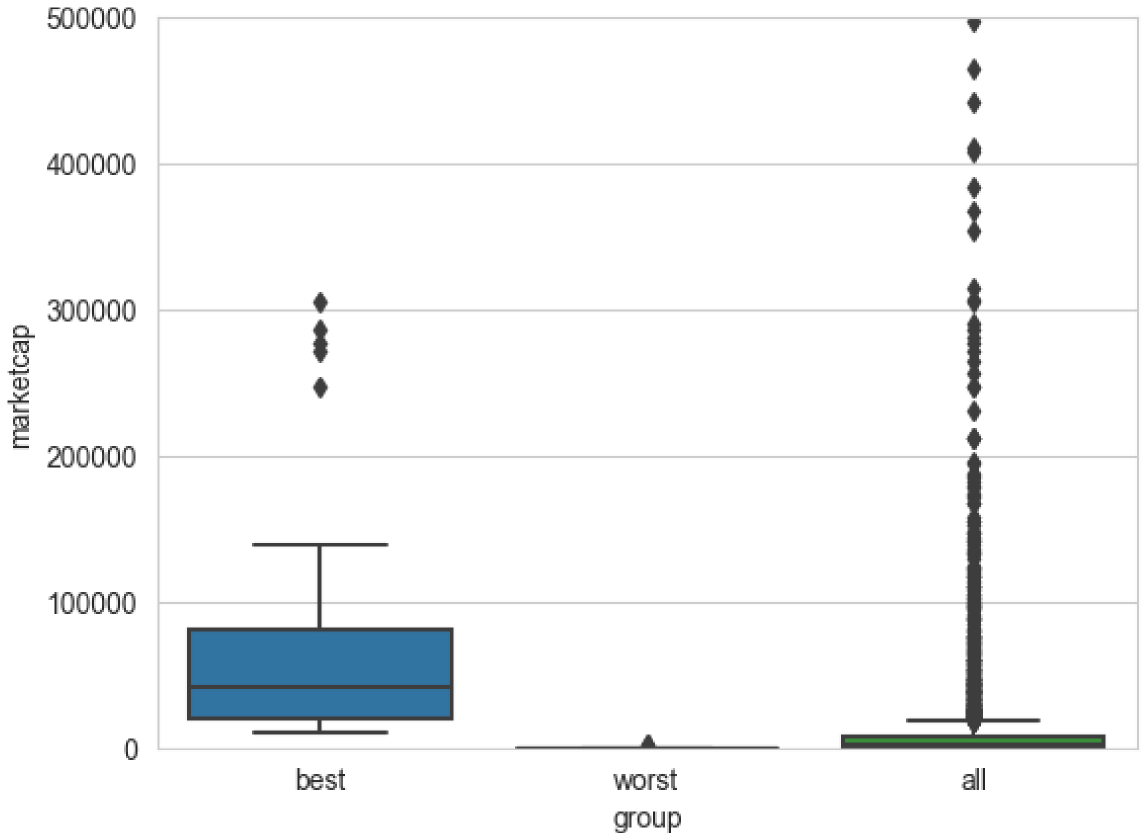
Out[14]:

```
(-4.0, 4.0)
```



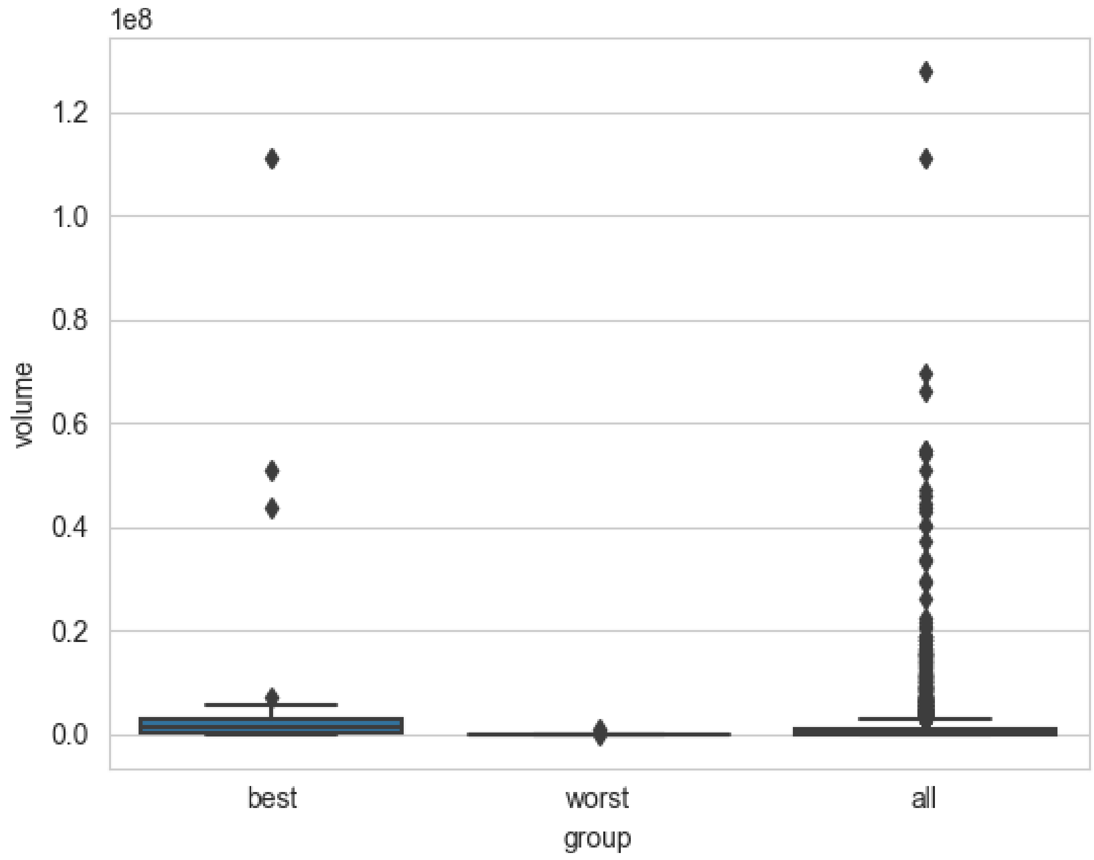
In [15]:

```
sns.boxplot(last, x="group", y="marketcap")
plt.ylim((0, 0.5e6))
plt.show()
```



In [16]:

```
sns.boxplot(last, x="group", y="volume")
# plt.ylim((0, 0.5e6))
plt.show()
```



5. Evaluate

Add SPY returns

In [17]:

```
import yfinance as yf

spy = yf.download("SPY", start=2017)["Adj Close"]
spy = pd.DataFrame(spy)
spy["date"] = spy.index.map(
    lambda x: x + timedelta(4 - x.weekday())
)
spy = spy.groupby(["date"])["Adj Close"].last()
spy = spy.pct_change()

rets = pd.concat((spy, best_rets, worst_rets), axis=1).dropna()
rets.columns = ["spy", "best", "worst"]
```

```
[*********************100%%**********************]  1 of 1 completed
```

Plot performance

In [ ]:

```
logy = False

(1+rets.best).cumprod().plot(label="Best", logy=logy)
(1+rets.spy).cumprod().plot(label="SPY", logy=logy)
(1+rets.worst).cumprod().plot(label="Worst", logy=logy)
plt.legend()
plt.show()
```

Find frontier of SPY, best, and worst

In [18]:

```
from cvxopt import matrix
from cvxopt.solvers import qp

cov = rets.cov()
means = rets.mean()

P = cov
A = np.array(
    [
        means,
        [1., 1., 1.]
    ]
)
P = matrix(P.to_numpy())
q = matrix(np.zeros((3, 1)))
A = matrix(A)

mns = []
vars = []
ports = []
for targ in np.linspace(0, 0.5/52, 50):
    b = matrix(
        np.array([targ, 1]).reshape(2, 1)
    )
    sol = qp(
        P=P,
        q=q,
        A=A,
        b=b
```

Find best portfolio with same risk as SPY

In [19]:

```
stdev = np.max([s for s in sds if s <= np.sqrt(52)*rets.spy.std()])
indx = np.where(sds==stdev)[0].item()
mean = mns[indx]
port = ports[indx]
print(port.round(2))
print(f"portfolio expected return is {mean:.1%}")
```

```
spy      0.53
best     0.92
worst   -0.44
dtype: float64
portfolio expected return is 38.8%
```

Long-only portfolios of SPY and best

In [ ]:

```
means = rets[["spy", "best"]].mean()
cov = rets[["spy", "best"]].cov()
ports = [np.array([w, 1-w]) for w in np.linspace(0, 1, 50)]
mns = [52 * means @ w for w in ports]
sds = [np.sqrt(52 * w @ cov @ w) for w in ports]

plt.plot(sds, mns, label=None)
plt.scatter(x=[np.sqrt(52)*rets.spy.std()], y=[52*rets.spy.mean()], label="SP
plt.scatter(x=[np.sqrt(52)*rets.best.std()], y=[52*rets.best.mean()], label="
plt.xlabel("Standard Deviation")
plt.ylabel("Expected Return")
plt.legend()
plt.show()
```