interpretability

May 10, 2024

1 Interpretability

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys

sys.path.insert(1, "/Users/simon/Documents/II/Dissertation/")
%load_ext autoreload
%autoreload 2
```

[ ]:
```
stocks = ["NVDA", "JPM", "HD", "UNH"]
features = [
    "log_return",
    "log_return_open",
    "log_return_high",
    "log_return_low",
    "log_return_volume",
    "sma",
    "wma",
    "ema",
    "dema",
    "tema",
    "aroon",
    "rsi",
    "willr",
    "cci",
    "ad",
    "mom",
    "slowk",
    "slowd",
    "macd",
    "fed_funds_rate",
    "^N225",
    "^IXIC",
    "^FTSE",
    "^SPX",
```

```
    "^DJI",
]
```

1.1 Model Agnostic Methods

Permutation feature importance on deep learning models

Starting with LSTM model

[ ]:
```python
from src.evaluate import permutation_importance

dfs = []
for stock in ["NVDA", "JPM", "UNH", "HD"]:
    df = permutation_importance("LSTM", stock)
    dfs.append(df)
dfs = pd.concat(dfs)
dfs = dfs.groupby(level=0).mean()
display(dfs)
```

```
Loading LSTM_NVDA.
Rank 1: trial no. 0, value: 0.5537848472595215. Run completed at 2024-04-29
18:01:30.993116
Loading LSTM_NVDA.
Rank 1: trial no. 0, value: 0.5537848472595215. Run completed at 2024-04-29
18:01:30.993116
Loading LSTM_JPM.
Rank 1: trial no. 3, value: 0.6175298690795898. Run completed at 2024-04-29
18:13:38.976207
Loading LSTM_JPM.
Rank 1: trial no. 3, value: 0.6175298690795898. Run completed at 2024-04-29
18:13:38.976207
Loading LSTM_UNH.
Rank 1: trial no. 11, value: 0.5896414518356323. Run completed at 2024-04-29
20:23:47.878337
Loading LSTM_UNH.
Rank 1: trial no. 11, value: 0.5896414518356323. Run completed at 2024-04-29
20:23:47.878337
Loading LSTM_HD.
Rank 1: trial no. 4, value: 0.6055777072906494. Run completed at 2024-04-29
18:19:41.955427
Loading LSTM_HD.
Rank 1: trial no. 4, value: 0.6055777072906494. Run completed at 2024-04-29
18:19:41.955427
```

	RMSE	Accuracy	Avg. daily return	Risk adj. return
^DJI	0.00006412	1.70000000	0.00040338	0.02809956
^FTSE	0.00002575	1.10000000	0.00035024	0.02595172
^IXIC	0.00003057	0.40000000	0.00023086	0.02152301
^N225	0.00002271	2.40000000	0.00046137	0.03695707

^SPX	0.00002844	1.30000000	0.00037648	0.02756674
ad	0.00003609	0.60000000	-0.00001082	-0.00212652
aroon	0.00001893	-0.50000000	-0.00017383	-0.01627432
cci	0.00002091	0.60000000	0.00004210	0.00451493
dema	-0.00000213	0.30000000	-0.00001718	-0.00237491
ema	0.00000732	0.30000000	-0.00000340	-0.00399535
fed_funds_rate	0.00004464	2.20000000	0.00033264	0.02805641
log_return	-0.00000931	0.20000000	-0.00002239	-0.00557027
log_return_high	0.00000427	0.50000000	0.00002165	-0.00290638
log_return_low	0.00001073	-0.30000000	-0.00007294	-0.00753752
log_return_open	-0.00000563	0.20000000	0.00000356	-0.00373820
log_return_volume	0.00000929	0.10000000	-0.00005041	-0.00520350
macd	0.00001762	0.90000000	-0.00005187	-0.01622437
mom	0.00003174	0.70000000	0.00000821	-0.00090274
rsi	0.00000885	0.60000000	-0.00003810	-0.00492711
slowd	0.00001508	0.70000000	0.00011847	0.00824941
slowk	0.00002274	0.60000000	0.00003260	-0.00053751
sma	0.00000607	-0.60000000	-0.00010817	-0.01150514
tema	0.00001056	0.20000000	-0.00009030	-0.01144403
willr	0.00002341	0.70000000	0.00000058	-0.00012027
wma	0.00000381	-0.20000000	-0.00008983	-0.00908731

We can see RMSE is change minimally by perturbations, so we ignore. We normalise other columns for ease of comparison.

[ ]:
```python
df = dfs.copy()
df = df[["Accuracy", "Avg. daily return", "Risk adj. return"]]
df = (df - df.mean()) / df.std()
df["Mean"] = df.mean(axis=1)
df = df.sort_values("Mean")
df
```

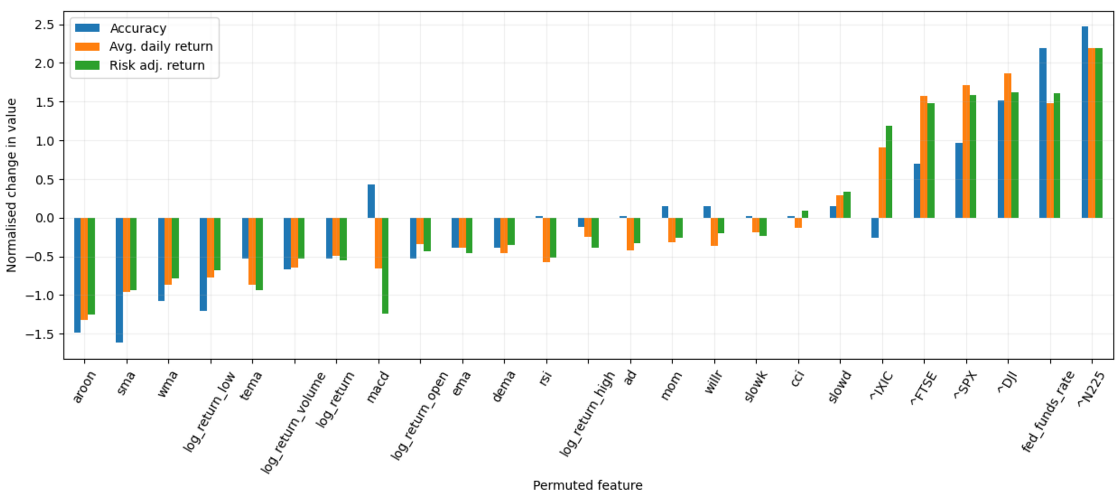
[ ]:

	Accuracy	Avg. daily return	Risk adj. return	Mean
aroon	-1.48250348	-1.32831239	-1.24822796	-1.35301461
sma	-1.61876299	-0.96482383	-0.94029241	-1.17462641
wma	-1.07372495	-0.86327923	-0.78417868	-0.90706095
log_return_low	-1.20998446	-0.76977775	-0.68411222	-0.88795814
tema	-0.52868690	-0.86592872	-0.93634702	-0.77698755
log_return_volume	-0.66494641	-0.64509908	-0.53340975	-0.61448508
log_return	-0.52868690	-0.48995289	-0.55709080	-0.52524353
macd	0.42512967	-0.65317340	-1.24500266	-0.49101546
log_return_open	-0.52868690	-0.34628410	-0.43879833	-0.43792311
ema	-0.39242739	-0.38482895	-0.45540183	-0.41088606
dema	-0.39242739	-0.46110388	-0.35077385	-0.40143504
rsi	0.01635114	-0.57695258	-0.51556367	-0.35872170
log_return_high	-0.11990837	-0.24615949	-0.38508954	-0.25038580
ad	0.01635114	-0.42589678	-0.33473583	-0.24809382
mom	0.15261065	-0.32058653	-0.25571938	-0.14123175

willr	0.15261065	-0.36278781	-0.20519705	-0.13845807
slowk	0.01635114	-0.18552028	-0.23213676	-0.13376863
cci	0.01635114	-0.13294877	0.09408765	-0.00750333
slowd	0.15261065	0.28979802	0.33521470	0.25920779
^IXIC	-0.25616788	0.91199833	1.19226149	0.61603065
^FTSE	0.69764870	1.57287904	1.47821374	1.24958049
^SPX	0.97016772	1.71814232	1.58249208	1.42360070
^DJI	1.51520576	1.86707671	1.61689469	1.66639239
fed_funds_rate	2.19650331	1.47546299	1.61410861	1.76202497
^N225	2.46902234	2.18805903	2.18880476	2.28196204

[ ]:
```
fig, ax = plt.subplots(figsize=(15, 5))
df[["Accuracy", "Avg. daily return", "Risk adj. return"]].plot(kind="bar",␣
 ↪ax=ax)
ax.xaxis.set_tick_params(rotation=60)
ax.grid(True, alpha=0.2)
ax.set_ylabel("Normalised change in value")
ax.set_xlabel("Permuted feature")
```

[ ]: Text(0.5, 0, 'Permuted feature')



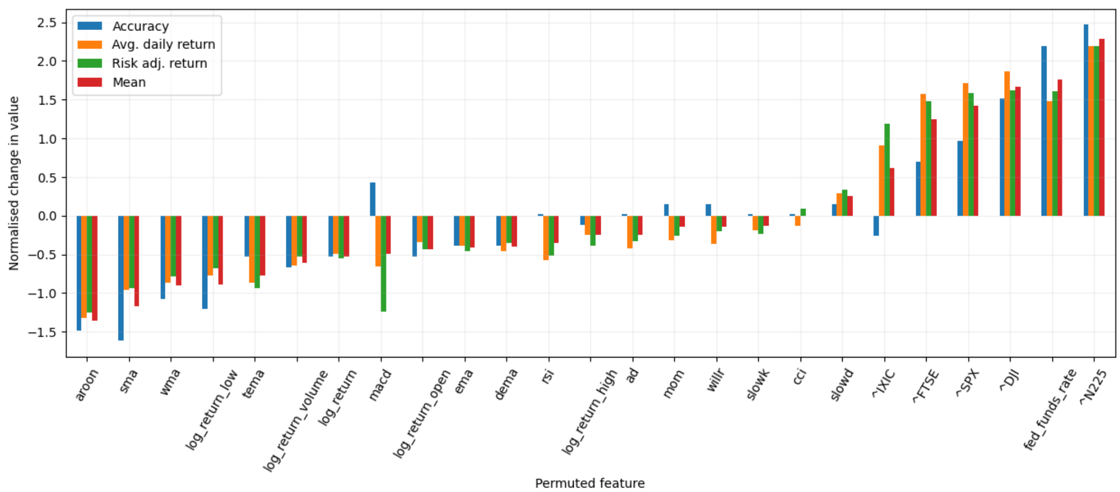
Now, the rest of the models.

[ ]:
```
dfs = []
for model in ["LSTM", "CNN", "ConvLSTM"]:
    df = []
    for stock in ["NVDA", "JPM", "UNH", "HD"]:
        df.append(permutation_importance(model, stock))
    df = (
        pd.concat(df)
```

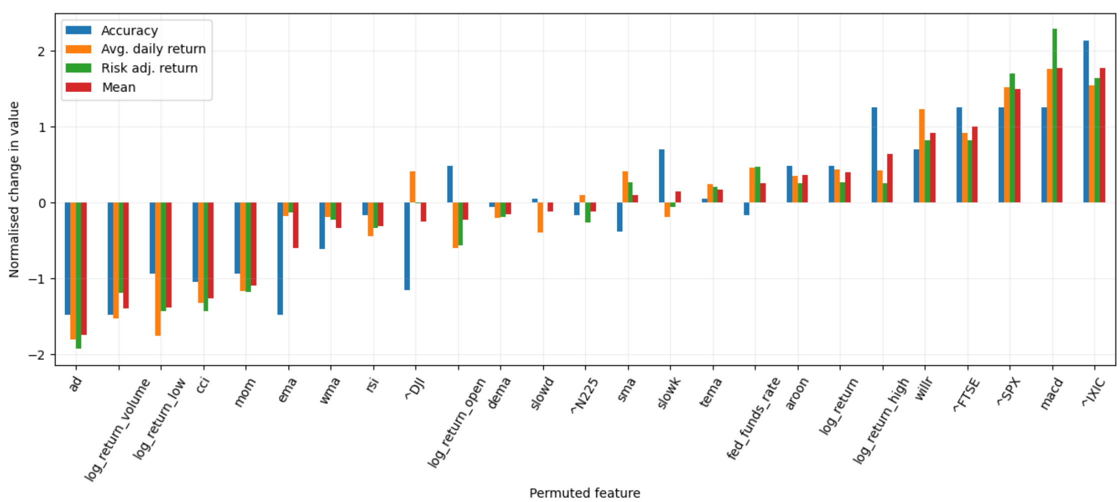
```python
        .groupby(level=0)
        .mean()[["Accuracy", "Avg. daily return", "Risk adj. return"]]
    )
    df = (df - df.mean()) / df.std()
    df["Mean"] = df.mean(axis=1)
    df = df.sort_values("Mean")
    dfs.append(df)
    fig, ax = plt.subplots(figsize=(15, 5))
    df.plot(kind="bar", ax=ax)
    ax.xaxis.set_tick_params(rotation=60)
    ax.grid(True, alpha=0.2)
    ax.set_ylabel("Normalised change in value")
    ax.set_xlabel("Permuted feature")
dfs = pd.concat(dfs)
dfs = dfs.groupby(level=0).mean()
plt.show()
```

```
Loading LSTM_NVDA.
Rank 1: trial no. 0, value: 0.5537848472595215. Run completed at 2024-04-29
18:01:30.993116
Loading LSTM_NVDA.
Rank 1: trial no. 0, value: 0.5537848472595215. Run completed at 2024-04-29
18:01:30.993116
Loading LSTM_JPM.
Rank 1: trial no. 3, value: 0.6175298690795898. Run completed at 2024-04-29
18:13:38.976207
Loading LSTM_JPM.
Rank 1: trial no. 3, value: 0.6175298690795898. Run completed at 2024-04-29
18:13:38.976207
Loading LSTM_UNH.
Rank 1: trial no. 11, value: 0.5896414518356323. Run completed at 2024-04-29
20:23:47.878337
Loading LSTM_UNH.
Rank 1: trial no. 11, value: 0.5896414518356323. Run completed at 2024-04-29
20:23:47.878337
Loading LSTM_HD.
Rank 1: trial no. 4, value: 0.6055777072906494. Run completed at 2024-04-29
18:19:41.955427
Loading LSTM_HD.
Rank 1: trial no. 4, value: 0.6055777072906494. Run completed at 2024-04-29
18:19:41.955427
Loading CNN_NVDA.
Rank 1: trial no. 11, value: 0.5737051963806152. Run completed at 2024-04-29
20:22:15.814904
Loading CNN_NVDA.
Rank 1: trial no. 11, value: 0.5737051963806152. Run completed at 2024-04-29
20:22:15.814904
```

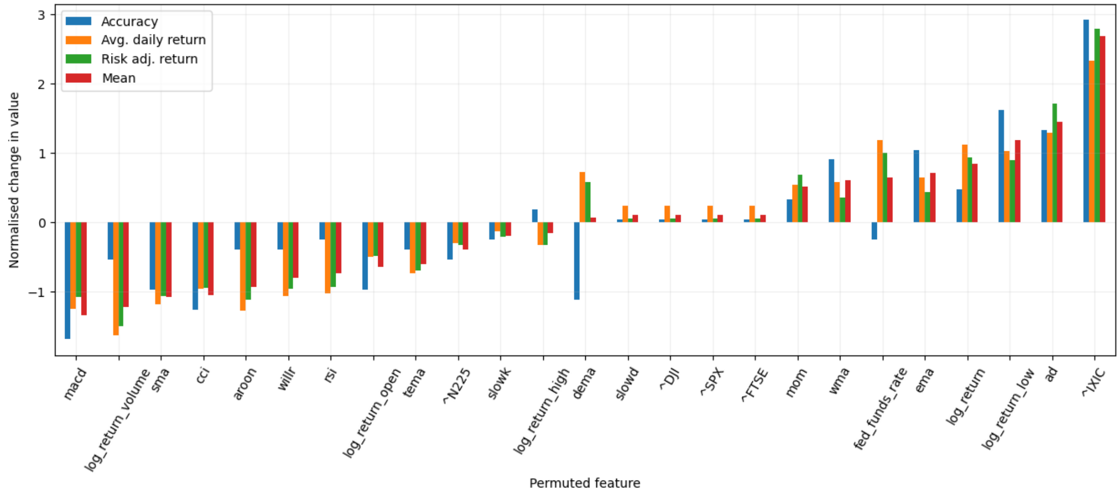
```
Loading CNN_JPM.
Rank 1: trial no. 9, value: 0.5737051963806152. Run completed at 2024-04-29
18:34:47.643948
Loading CNN_JPM.
Rank 1: trial no. 9, value: 0.5737051963806152. Run completed at 2024-04-29
18:34:47.643948
Loading CNN_UNH.
Rank 1: trial no. 8, value: 0.5816733241081238. Run completed at 2024-04-29
18:44:16.965952
Loading CNN_UNH.
Rank 1: trial no. 8, value: 0.5816733241081238. Run completed at 2024-04-29
18:44:16.965952
Loading CNN_HD.
Rank 1: trial no. 0, value: 0.5498008131980896. Run completed at 2024-04-29
18:35:17.552823
Loading CNN_HD.
Rank 1: trial no. 0, value: 0.5498008131980896. Run completed at 2024-04-29
18:35:17.552823
Loading ConvLSTM_NVDA.
Rank 1: trial no. 2, value: 0.518652081489563. Run completed at 2024-04-29
18:47:06.218729
Loading ConvLSTM_NVDA.
Rank 1: trial no. 2, value: 0.518652081489563. Run completed at 2024-04-29
18:47:06.218729
Loading ConvLSTM_JPM.
Rank 1: trial no. 26, value: 0.5697211027145386. Run completed at 2024-04-29
20:35:01.199259
Loading ConvLSTM_JPM.
Rank 1: trial no. 26, value: 0.5697211027145386. Run completed at 2024-04-29
20:35:01.199259
Loading ConvLSTM_UNH.
Rank 1: trial no. 0, value: 0.5577689409255981. Run completed at 2024-04-29
16:26:05.723444
Loading ConvLSTM_UNH.
Rank 1: trial no. 0, value: 0.5577689409255981. Run completed at 2024-04-29
16:26:05.723444
Loading ConvLSTM_HD.
Rank 1: trial no. 6, value: 0.5231999158859253. Run completed at 2024-04-29
19:04:03.502803
Loading ConvLSTM_HD.
Rank 1: trial no. 6, value: 0.5231999158859253. Run completed at 2024-04-29
19:04:03.502803
```



Accuracy
Avg. daily return
Risk adj. return
Mean
Normalised change in value
Permuted feature
aroon
sma
wma
log_return_low
tema
log_return_volume
log_return
macd
log_return_open
ema
dema
rsi
log_return_high
ad
mom
willr
slowk
cci
slowd
^IXIC
^FTSE
^SPX
^DJI
fed_funds_rate
^N225



Accuracy
Avg. daily return
Risk adj. return
Mean
Normalised change in value
Permuted feature
ad
log_return_volume
log_return_low
cci
mom
ema
wma
rsi
^DJI
log_return_open
dema
slowd
^N225
sma
slowk
tema
fed_funds_rate
aroon
log_return
log_return_high
willr
^FTSE
^SPX
macd
^IXIC



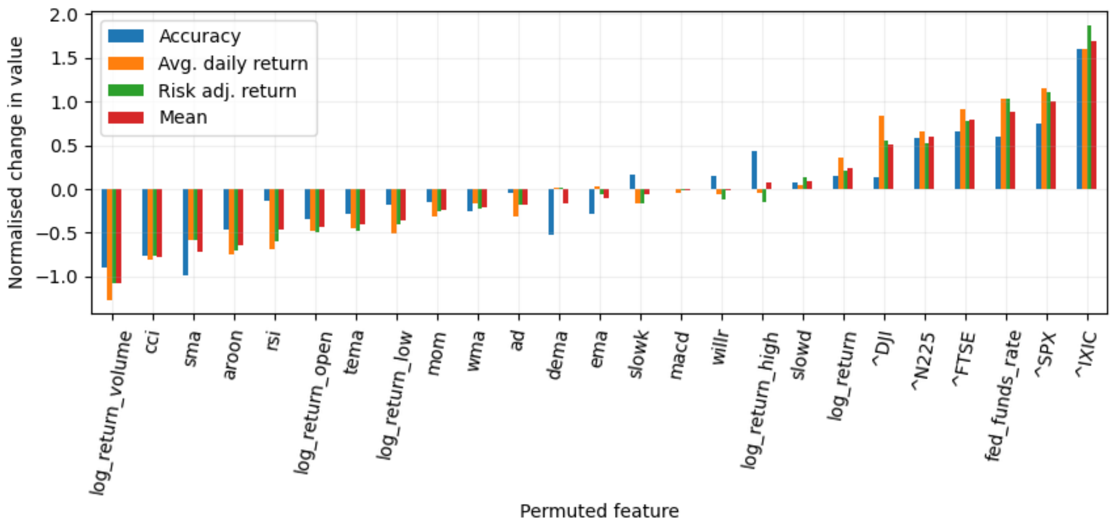
Averaging across stocks

[ ]:
```
df = dfs.copy()

df = df.sort_values("Mean")
fig, ax = plt.subplots(figsize=(10, 3))
df.plot(kind="bar", ax=ax)
ax.xaxis.set_tick_params(rotation=80)
ax.grid(True, alpha=0.2)
ax.set_ylabel("Normalised change in value")
ax.set_xlabel("Permuted feature")
df
```

[ ]:

	Accuracy	Avg. daily return	Risk adj. return	Mean
log_return_volume	-0.89585437	-1.26626483	-1.07317136	-1.07843018
cci	-0.76259351	-0.80684962	-0.76188032	-0.77710782
sma	-0.99231667	-0.58180066	-0.57679983	-0.71697239
aroon	-0.46246681	-0.75119884	-0.70462078	-0.63942881
rsi	-0.13414064	-0.68235326	-0.59439234	-0.47029541
log_return_open	-0.33655843	-0.48358601	-0.49506894	-0.43840446
tema	-0.29072772	-0.45468559	-0.47595069	-0.40712134
log_return_low	-0.17466970	-0.50320276	-0.40901811	-0.36229686
mom	-0.15253991	-0.31574550	-0.25006063	-0.23944868
wma	-0.25963814	-0.15869160	-0.22007282	-0.21280085
ad	-0.04465612	-0.31373110	-0.18581180	-0.18139967
dema	-0.52189593	0.02121311	0.01031735	-0.16345516
ema	-0.27693087	0.02653988	-0.05221220	-0.10086773
slowk	0.15825890	-0.16991831	-0.16924758	-0.06030233
macd	-0.00280808	-0.04822143	-0.01187932	-0.02096961
willr	0.15567112	-0.06608816	-0.11553389	-0.00865031

log_return_high	0.43961163	-0.04909692	-0.15573339	0.07826044
slowd	0.08039433	0.04306811	0.13255327	0.08533857
log_return	0.14351777	0.35488613	0.21694026	0.23844805
^DJI	0.13254333	0.83747877	0.55586126	0.50862779
^N225	0.58740119	0.66141556	0.52961896	0.59281190
^FTSE	0.66412303	0.91017052	0.78601732	0.78677029
fed_funds_rate	0.59257675	1.04025799	1.02898849	0.88727441
^SPX	0.75496271	1.16039315	1.11302765	1.00946117
^IXIC	1.59873612	1.59601139	1.87812946	1.69095899



1.2 Linear model

[ ]:
```
from src.models.statistical.Linear import data
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LassoLarsIC
from sklearn.pipeline import make_pipeline
```

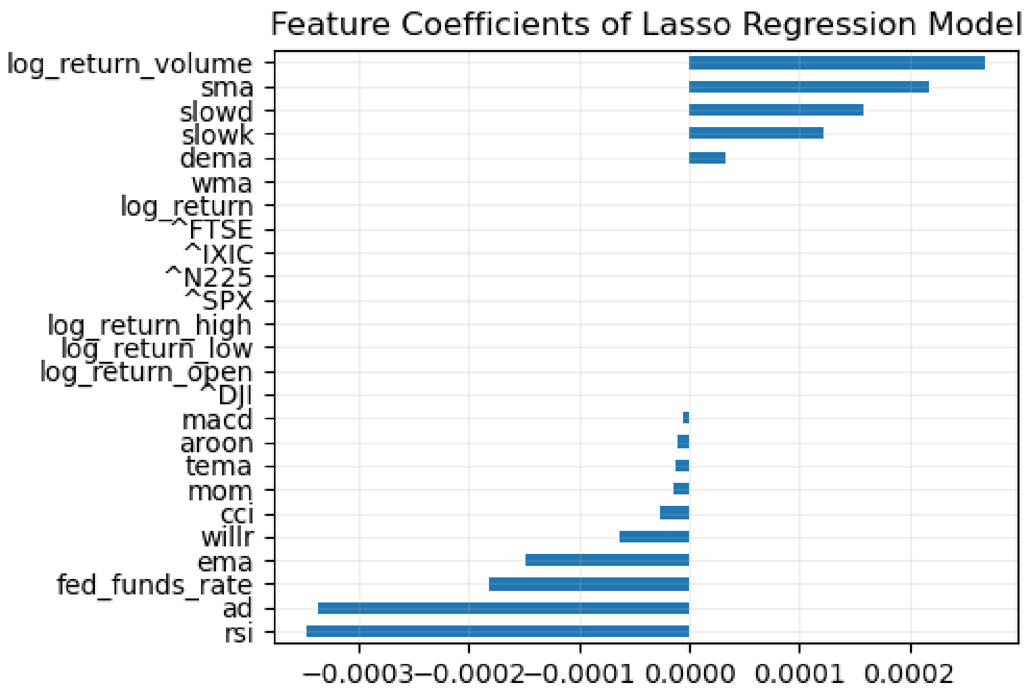
[ ]:
```
coef_series = []

no_reg = []
L1 = []
L2 = []
for s in ["NVDA", "JPM", "HD", "UNH"]:
    X_train, y_train, X_val, y_val, X_test, y_test = data(s, features)

    model = LinearRegression(fit_intercept=True)
    model.fit(X_train, y_train)
    series = pd.Series(index=X_train.columns, data=model.coef_).rename("Linear")
    no_reg.append(series)
```

```python
    model = Lasso(fit_intercept=True, alpha=0.0001)
    # model = LassoLarsIC(fit_intercept=True, criterion="aic")
    model.fit(X_train, y_train)
    series = pd.Series(index=X_train.columns, data=model.coef_).rename("Lasso")
    L1.append(series)

    model = Ridge(fit_intercept=True, alpha=0.01)
    model.fit(X_train, y_train)
    series = pd.Series(index=X_train.columns, data=model.coef_).rename("Ridge")
    L2.append(series)

coef_series.append(pd.concat(no_reg).groupby(level=0).mean())
coef_series.append(pd.concat(L1).groupby(level=0).mean())
coef_series.append(pd.concat(L2).groupby(level=0).mean())

df = pd.concat(coef_series, axis=1)
df = df.sort_values(by="Lasso")
df
```

[ ]:

	Linear	Lasso	Ridge
rsi	-0.00038000	-0.00034654	-0.00037256
ad	-0.00019192	-0.00033687	-0.00020284
fed_funds_rate	-0.00023750	-0.00018194	-0.00023633
ema	-0.00021096	-0.00014861	-0.00021201
willr	0.00040780	-0.00006327	0.00039673
cci	0.00020521	-0.00002586	0.00020647
mom	-0.00005198	-0.00001379	-0.00004656
tema	0.00023459	-0.00001230	0.00023786
aroon	0.00022882	-0.00001051	0.00022524
macd	-0.00003162	-0.00000514	-0.00002526
^DJI	0.05974673	0.00000000	-0.00016239
log_return_open	-0.04368791	0.00000000	-0.04231992
log_return_low	0.01856993	0.00000000	0.01703705
log_return_high	0.03483383	0.00000000	0.03370565
^SPX	-0.19801942	0.00000000	-0.10428030
^N225	0.02072377	0.00000000	0.02051037
^IXIC	0.03642702	0.00000000	0.00466998
^FTSE	-0.06111706	0.00000000	-0.06031058
log_return	-0.03744253	0.00000000	-0.03728044
wma	0.00002363	0.00000000	0.00002540
dema	0.00017859	0.00003273	0.00017730
slowk	0.00025950	0.00012155	0.00026006
slowd	0.00015419	0.00015863	0.00015540
sma	0.00048784	0.00021751	0.00048822
log_return_volume	0.00023915	0.00026785	0.00023590

```
[ ]: fig, ax = plt.subplots()
     df["Lasso"].plot(
         kind="barh",
         figsize=(5, 4),
         title="Feature Coefficients of Lasso Regression Model",
         ax=ax,
     )
     ax.grid(True, alpha=0.2)
```

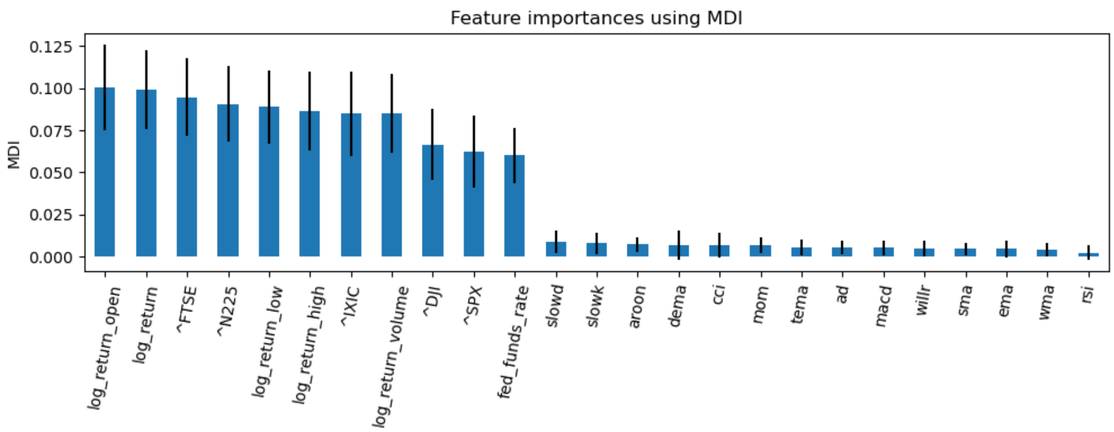


1.3 Random forest

```
[ ]: from src.models.statistical.RandomForest import data
     from sklearn.ensemble import RandomForestRegressor

     dfs = []
     for s in stocks:
         X_train, y_train, X_val, y_val, X_test, y_test = data(s, features)
         model = RandomForestRegressor()
         model.fit(X_train, y_train)
         df = pd.DataFrame(
             {
                 "Feature Importance": model.feature_importances_,
                 "Std": np.std(
                     [tree.feature_importances_ for tree in model.estimators_],
```

```
                axis=0,
            ),
        },
        index=X_train.columns,
    )
    dfs.append(df)
dfs = pd.concat(dfs)
dfs = dfs.groupby(level=0).mean().sort_values("Feature Importance",␣
↪ascending=False)
```

[ ]:
```
fig, ax = plt.subplots(figsize=(10, 4))
dfs["Feature Importance"].plot.bar(yerr=dfs["Std"], ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("MDI")
ax.xaxis.set_tick_params(rotation=80)
fig.tight_layout()
```

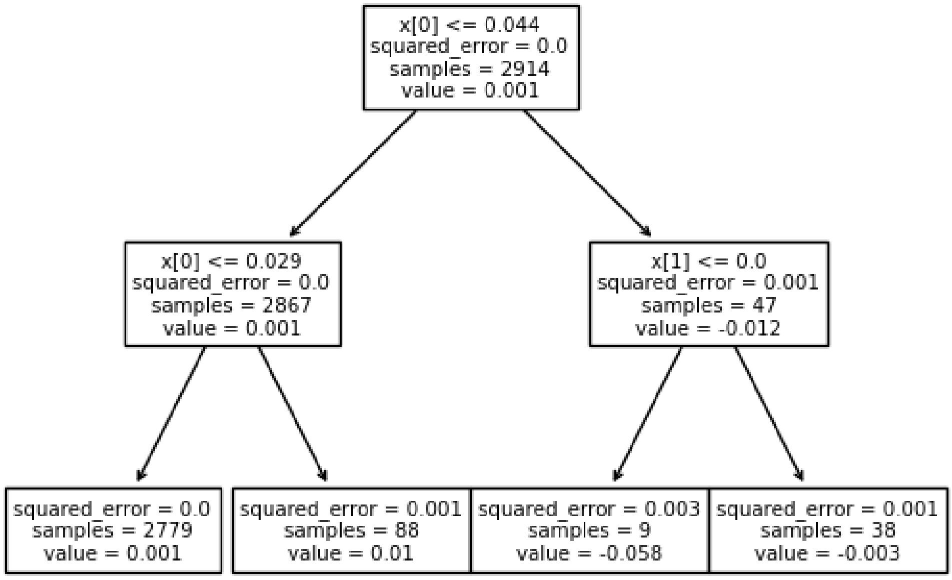


Visualising individual tree

[ ]:
```
from sklearn import tree

for s in stocks:
    fs = ["log_return", "sma", "rsi", "fed_funds_rate"]
    X_train, y_train, X_val, y_val, X_test, y_test = data(s, fs)
    model = RandomForestRegressor(max_depth=2)
    model.fit(X_train, y_train)
    print(
        pd.Series(index=X_train.columns, data=model.feature_importances_).
↪sort_values()
    )
tree.plot_tree(model.estimators_[1])
```

```
rsi               0.00000000
sma               0.09064458
fed_funds_rate    0.21337907
log_return        0.69597635
dtype: float64
rsi               0.00000000
sma               0.01086256
fed_funds_rate    0.23123626
log_return        0.75790118
dtype: float64
rsi               0.00000000
fed_funds_rate    0.08667353
sma               0.08768303
log_return        0.82564344
dtype: float64
rsi               0.00000000
sma               0.07860289
fed_funds_rate    0.23727181
log_return        0.68412530
dtype: float64
```

[ ]:
```
[Text(0.5, 0.8333333333333334, 'x[0] <= 0.044\nsquared_error = 0.0\nsamples =
2914\nvalue = 0.001'),
 Text(0.25, 0.5, 'x[0] <= 0.029\nsquared_error = 0.0\nsamples = 2867\nvalue =
0.001'),
 Text(0.125, 0.16666666666666666, 'squared_error = 0.0\nsamples = 2779\nvalue =
0.001'),
 Text(0.375, 0.16666666666666666, 'squared_error = 0.001\nsamples = 88\nvalue =
0.01'),
 Text(0.75, 0.5, 'x[1] <= 0.0\nsquared_error = 0.001\nsamples = 47\nvalue =
-0.012'),
 Text(0.625, 0.16666666666666666, 'squared_error = 0.003\nsamples = 9\nvalue =
-0.058'),
 Text(0.875, 0.16666666666666666, 'squared_error = 0.001\nsamples = 38\nvalue =
-0.003')]
```


x[0] <= 0.044
squared_error = 0.0
samples = 2914
value = 0.001
x[0] <= 0.029
squared_error = 0.0
samples = 2867
value = 0.001
x[1] <= 0.0
squared_error = 0.001
samples = 47
value = -0.012
squared_error = 0.0
samples = 2779
value = 0.001
squared_error = 0.001
samples = 88
value = 0.01
squared_error = 0.003
samples = 9
value = -0.058
squared_error = 0.001
samples = 38
value = -0.003