Stock Return and Volatility Modeling with Earning Report Keyword Analysis

March 19, 2025

[1]:
```
import os
import glob
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.feature_selection import VarianceThreshold
from datetime import datetime, timedelta
from collections import defaultdict
from math import prod
from ast import literal_eval

pd.set_option("max_colwidth", 40)
pd.options.display.max_rows = 4000
```

1 1. Classification of Keywords

In this section, I create an excel file with brand new columns for each Category in the latter section of the Data spreadsheet.

[2]:
```
# Import files

txt_files = glob.glob(os.path.join("Daily_Return Data", "*.txt"))
df_data_temp = pd.read_excel(open('Data.xlsx', 'rb'))

# Filter irrelevant data as instructed
```

```
df_data = df_data_temp.copy()
df_data = df_data[df_data['documentType'] == 'EARNINGS_CALL']
df_data = df_data[df_data['country'] == 'US']

# Forward fill any missing identifiers by referencing earnings calls with the␣
↪same id
# Remove event columns as instructed
# Populate nan keydriver cells with neutral score of 0
# Populate nan Method2 Category cells with neutral scores of [0,0]

df_data = df_data.sort_values(by=['companyId', 'mainIdentifier isin'])
df_data['mainIdentifier isin'] = df_data.groupby(['companyId',␣
↪'companyName'])['mainIdentifier isin'].transform(lambda x: x.ffill())

event_col = [col for col in df_data.columns if 'eventcount' in col.lower()]
df_data = df_data.drop(columns = event_col)

score_col = [col for col in df_data.columns if 'score' in col.lower()]
df_data[score_col] = df_data[score_col].fillna(0)

df_data.iloc[:, 205:] = df_data.iloc[:, 205:].fillna('[0, 0]')
sorted_m2 = sorted(list(df_data.iloc[:, 205:].columns))
```

[3]:
```
# Group Method2 Category by prefix
# Calculate the aggregate [positive,negative] score for each Category in a␣
↪dynamically created column

def sum_tuple_group(row, cols):
    total_first = 0
    total_second = 0
    for col in cols:
        val = row[col]
        if isinstance(val, str):
            try:
                val = literal_eval(val)
            except Exception as e:
                continue
        total_first += val[0]
        total_second += val[1]
    return [total_first, total_second]


grouped_cols = defaultdict(list)
for col in sorted_m2:
    prefix = col.split(" - ")[0]
    grouped_cols[prefix].append(col)
```

```
for prefix, cols in grouped_cols.items():
    new_col_name = f"{prefix}_Total"
    df_data[new_col_name] = df_data.apply(lambda row: sum_tuple_group(row,␣
↪cols), axis=1)
```

[4]:
```
# Export result as csv or xlsx

df_data.to_csv('question1.csv')
```

2 2. Modeling forward stock returns with keyDriver scores

Here, I merge the earning calls table with the returns table, and calculate the forward returns after each earning call.

After doing feature selection, I evaluate the performance of several traditional machine learning models.

- Reprocess the earning calls dataframe in case someone wants to start here
- Filter unnecessary documents and regions, fill in missing mainIdentifier isin
- Remove events, populate missing scores
- Add eventDate column which either will provide the same day as the earning call if the earning call is made before closing hours, or next day if the call is after hours

[5]:
```
df_data2 = df_data_temp.copy()
df_data2 = df_data2[df_data2['documentType'] == 'EARNINGS_CALL']
df_data2 = df_data2[df_data2['country'] == 'US']

df_data2 = df_data2.sort_values(by=['companyId', 'mainIdentifier isin'])
df_data2['mainIdentifier isin'] = df_data2.groupby(['companyId',␣
↪'companyName'])['mainIdentifier isin'].transform(lambda x: x.ffill())
df_data2 = df_data2.rename(columns={'mainIdentifier isin': 'fsym_isin'})

event_col = [col for col in df_data2.columns if 'eventcount' in col.lower()]
df_data2 = df_data2.drop(columns = event_col)

score_col = [col for col in df_data2.columns if 'score' in col.lower()]
df_data2[score_col] = df_data2[score_col].fillna(0)

def roll_date(timestamp):
    dt = datetime.fromisoformat(timestamp)
    cutoff_time = dt.replace(hour=16, minute=0, second=0, microsecond=0)

    if dt >= cutoff_time:
        dt += timedelta(days=1)

    return dt.date().isoformat()
```

```python
df_data2['eventDate'] = df_data2['eventTime'].apply(roll_date)
```

- Upload the returns dataset, and reorganize with pivot for each row to have its own mainIdentifier isin
- Merge the earnings and returns datasets where both returns are provided and there also a matching earning call of the same identifier

[6]:
```python
df_list = []
for file in txt_files:
    df = pd.read_csv(file, delimiter="\t", header=None, encoding="utf-8")
    df_list.append(df)

df_ret = pd.concat(df_list, ignore_index=True)
df_filter = df_ret[df_ret[1] != 'DATE']
df_returns = df_filter.rename(columns={0: "BENCHMARK_ID",
                                       1: "DATE",
                                       2: "SECURITY_ID",
                                       3: "Weight",
                                       4: "p_price_returns",
                                       5: "gd_class_gics_h",
                                       6: "fsym_security_perm_id",
                                       7: "p_symbol",
                                       8: "fsym_isin"})



df_returns = df_returns.sort_values(by=['fsym_isin','DATE'])
df_retwide = df_returns.pivot_table(index = ['fsym_isin'],
                           columns = 'DATE',
                           values = 'p_price_returns')

df_retwide = df_retwide.fillna(1)

df_merge = pd.merge(df_data2, df_retwide, on=['fsym_isin'], how='inner')
```

- With the eventDate provided earlier as the starting point, calculate several ranges of forward returns for each earning call

[7]:
```python
def oneday_ret(row):
    event_date = row['eventDate']
    future_dates = [col for col in s_retcol if pd.to_datetime(col) >=␣
 ↪event_date]

    if len(future_dates) > 0:
        res = future_dates[0]
        return row[res]
    else:
        return 0
```

```python
def nday_ret(row, days):
    event_date = row['eventDate']
    future_dates = [col for col in s_retcol if pd.to_datetime(col) >=␣
 ↪event_date]

    if len(future_dates) >= days:
        res = [row[col] for col in future_dates[:days]]
        cumres = (1 + pd.Series(res)).prod() - 1
        return cumres
    else:
        return None


df_merge['eventDate'] = pd.to_datetime(df_merge['eventDate'])
retcol = [col for col in df_merge.columns if col[:4].isdigit()]
retdates = pd.to_datetime(retcol)
s_retcol = [col for _, col in sorted(zip(retdates, retcol))]

df_merge['fow_1d'] = df_merge.apply(oneday_ret, axis=1)
df_merge['fow_3d'] = df_merge.apply(lambda row: nday_ret(row, 3), axis=1)
df_merge['fow_7d'] = df_merge.apply(lambda row: nday_ret(row, 7), axis=1)

df_merge[['fow_1d', 'fow_3d', 'fow_7d']] = df_merge[['fow_1d', 'fow_3d',␣
 ↪'fow_7d']].fillna(0)
df_merge.head()

# binarize into positive return or not

df_merge['fow_1d_bin'] = df_merge['fow_1d'].apply(lambda x: 1 if x > 0 else 0)
df_merge['fow_3d_bin'] = df_merge['fow_3d'].apply(lambda x: 1 if x > 0 else 0)
df_merge['fow_7d_bin'] = df_merge['fow_7d'].apply(lambda x: 1 if x > 0 else 0)
```

Given that we are looking at a dataset with many features and also happens to be very sparse, there are a few steps we should take. - We start by removing highly correlated features as a best practice. - Next, as a blanket approach to filtering sparsely populated features, we can filter by each feature's variance below a threshold, removing features that have little change and therefore little contribution. - Scale the independent variables.

There are a few models to try with sparse data: - Linear Regression as a baseline check - Lasso Regularization to penalize certain features - RandomForest and XGBoost, as decision trees are well-equipped for sparse data

We'll evaluate the performance of each model by looking at their R^2 value, and attempt to identify reoccurring features with high importance to see if there are any that can be used to predict postitive forward returns.

[8]:
```
# remove correlated features

drive_col = [col for col in df_merge.columns if 'keydriver' in col.lower()]

df_corr = df_merge[drive_col]
mat = df_corr.corr()

to_remove = set()
for i in range(len(mat.columns)):
    for j in range(i):
        if abs(mat.iloc[i, j]) > 0.9:
            colname = mat.columns[i]
            to_remove.add(colname)

print(to_remove)

df_tpp = df_merge[drive_col].drop(columns=to_remove)
```

```
{'Qna Deception keyDriver positiveScore', 'Total Exec Change keyDriver
positiveScore', 'Answer Exec Change keyDriver positiveScore', 'Answer Deception
keyDriver negativeScore', 'Answer Guidance keyDriver positiveScore', 'Total
Capital Raise Returns keyDriver positiveScore', 'Answer Headwinds Tailwinds
keyDriver negativeScore', 'Total Exec Change keyDriver negativeScore', 'Total
Wage keyDriver score'}
```

- Due to there being many features, some features may be very sparse and be mostly 0s. In that event, we filter and remove the features with low variance.

[9]:
```
selector = VarianceThreshold(threshold=0.01)  # Remove features with very low␣
↪variance
df_reduced = df_tpp.iloc[:, selector.fit(df_tpp).get_support()]
```

2.1 Linear Regression

[10]:
```
X = df_reduced
feature_names = df_reduced.columns

#y1d = df_merge['fow_1d']
#y3d = df_merge['fow_3d']
#y7d = df_merge['fow_7d']

y1d = df_merge['fow_1d_bin']
y3d = df_merge['fow_3d_bin']
y7d = df_merge['fow_7d_bin']
```

[11]:
```
forwards = [(1, y1d), (3, y3d), (5, y7d)]

for num, table in forwards:
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, table, test_size=0.
↪2, random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    feature_importance = np.abs(model.coef_)
    df_feat = pd.DataFrame({'Feature': feature_names, 'Importance':␣
↪feature_importance})
    df_feat = df_feat.sort_values(by="Importance", ascending=False)


    print(f"{num}-Day Forward Return Linear Regression")
    print("R^2:", r2)
    print("MSE:", mse)
    print(df_feat.head(10))
    print(" ")
```

```
1-Day Forward Return Linear Regression
R^2: -0.21224036354801634
MSE: 0.30305084025085477
                                  Feature  Importance
136  Presentation Irregularities keyDrive…    1.289802
130     Total Irregularities keyDriver score    1.115469
139  Answer Merger Acquisition keyDriver …    0.694026
137  Presentation Irregularities keyDrive…    0.621898
131  Total Irregularities keyDriver negat…    0.536375
145           Qna Exec Change keyDriver score    0.490342
146  Qna Exec Change keyDriver negativeScore    0.483078
138  Presentation Irregularities keyDrive…    0.447071
133    Answer Irregularities keyDriver score    0.432729
124  Answer Capital Raise Returns keyDriv…    0.385358

3-Day Forward Return Linear Regression
R^2: -0.5255556743718135
MSE: 0.3598636870530623
                                  Feature  Importance
145           Qna Exec Change keyDriver score    1.323129
139  Answer Merger Acquisition keyDriver …    1.227862
133    Answer Irregularities keyDriver score    1.031828
146  Qna Exec Change keyDriver negativeScore    0.835097
135  Answer Irregularities keyDriver posi…    0.791380
```

```
147  Qna Exec Change keyDriver positiveScore    0.567497
142  Question Irregularities keyDriver score    0.565293
127       Qna Irregularities keyDriver score    0.524743
140  Answer Merger Acquisition keyDriver …    0.505078
143  Question Irregularities keyDriver ne…    0.446560

5-Day Forward Return Linear Regression
R^2: -0.3829553723353947
MSE: 0.28637615597580396
                                     Feature  Importance
145          Qna Exec Change keyDriver score    1.406079
133    Answer Irregularities keyDriver score    1.163594
146  Qna Exec Change keyDriver negativeScore    0.991046
135  Answer Irregularities keyDriver posi…    0.644708
134  Answer Irregularities keyDriver nega…    0.614579
139  Answer Merger Acquisition keyDriver …    0.596427
125  Answer Capital Raise Returns keyDriv…    0.466832
147  Qna Exec Change keyDriver positiveScore    0.450984
141  Answer Merger Acquisition keyDriver …    0.403433
137  Presentation Irregularities keyDrive…    0.400318
```

3 Lasso Regularization

[12]:
```
for num, table in forwards:
    X_train, X_test, y_train, y_test = train_test_split(X, table, test_size=0.
↪2, random_state=42)

    param_grid = {'alpha': np.logspace(-4, 1, 50)}

    lasso = Lasso()

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

    best_alpha = grid_search.best_params_['alpha']

    lasso_best = Lasso(alpha=best_alpha)
    lasso_best.fit(X_train, y_train)
    y_pred = lasso_best.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    feature_importance = np.abs(lasso_best.coef_)
```

```
    df_las_imp = pd.DataFrame({'Feature': feature_names, 'Importance':␣
↪feature_importance})
    df_las_imp = df_las_imp.sort_values(by="Importance", ascending=False)

    print(f"{num}-Day Forward Return LassoCV")
    print("R^2:", r2)
    print("MSE:", mse)
    print(df_las_imp.head(10))
    print(" ")
```

```
1-Day Forward Return LassoCV
R^2: 0.02413155549624013
MSE: 0.24395966424974183
                                     Feature  Importance
10     Qna Deception keyDriver negativeScore    0.003581
29          Answer Deception keyDriver score    0.000036
31    Question Capital Raise Returns keyDr…    0.000027
97    Presentation Headwinds Tailwinds key…    0.000000
98    Presentation Headwinds Tailwinds key…    0.000000
99     Answer Market Position keyDriver score    0.000000
100   Answer Market Position keyDriver neg…    0.000000
101   Answer Market Position keyDriver pos…    0.000000
102           Question Margin keyDriver score    0.000000
103   Question Margin keyDriver negativeScore    0.000000

3-Day Forward Return LassoCV
R^2: -0.015915358271915148
MSE: 0.23964451294909678
                                     Feature  Importance
36    Presentation Wage keyDriver positive…    0.006740
66    Total Headwinds Tailwinds keyDriver …    0.005899
65    Total Headwinds Tailwinds keyDriver …    0.004715
10     Qna Deception keyDriver negativeScore    0.003642
18    Total Merger Acquisition keyDriver p…    0.003199
14    Total Deception keyDriver negativeScore    0.001934
29          Answer Deception keyDriver score    0.000027
31    Question Capital Raise Returns keyDr…    0.000025
0                 Qna Wage keyDriver score    0.000000
100   Answer Market Position keyDriver neg…    0.000000

5-Day Forward Return LassoCV
R^2: -0.005223245264588838
MSE: 0.20815709214843767
                                     Feature  Importance
29          Answer Deception keyDriver score    0.000044
31    Question Capital Raise Returns keyDr…    0.000024
104   Question Margin keyDriver positiveScore    0.000000
```

```
97   Presentation Headwinds Tailwinds key…    0.000000
98   Presentation Headwinds Tailwinds key…    0.000000
99    Answer Market Position keyDriver score    0.000000
100  Answer Market Position keyDriver neg…    0.000000
101  Answer Market Position keyDriver pos…    0.000000
102          Question Margin keyDriver score    0.000000
103  Question Margin keyDriver negativeScore    0.000000
```

3.1 Random Forest

[13]:
```python
for num, table in forwards:
    X_train, X_test, y_train, y_test = train_test_split(X, table, test_size=0.
 ↪2, random_state=42)

    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    df_grad_imp = pd.DataFrame({"Feature": X.columns, "Importance": rf.
 ↪feature_importances_})
    df_grad_imp = df_grad_imp.sort_values(by="Importance", ascending=False)

    print(f"{num}-Day Forward Return Random Forest")
    print("R^2:", r2)
    print("MSE:", mse)
    print(df_grad_imp.head(10))
    print(" ")
```

```
1-Day Forward Return Random Forest
R^2: -0.05129175824175847
MSE: 0.26281491712707183
                                    Feature  Importance
9              Qna Deception keyDriver score    0.030538
19            Total Guidance keyDriver score    0.025018
22  Total Capital Raise Returns keyDrive…    0.021717
16  Total Merger Acquisition keyDriver s…    0.020661
29           Answer Deception keyDriver score    0.020611
27            Answer Guidance keyDriver score    0.019777
37   Presentation Deception keyDriver score    0.019405
52       Qna Market Position keyDriver score    0.019033
87    Presentation Guidance keyDriver score    0.018814
96  Presentation Headwinds Tailwinds key…    0.018164
```

```
3-Day Forward Return Random Forest
R^2: -0.06820681935817796
MSE: 0.2519795580110497
                                     Feature  Importance
9              Qna Deception keyDriver score    0.026530
10     Qna Deception keyDriver negativeScore    0.023715
61     Total Market Position keyDriver score    0.023457
64  Total Headwinds Tailwinds keyDriver …    0.022962
96  Presentation Headwinds Tailwinds key…    0.021703
22  Total Capital Raise Returns keyDrive…    0.019802
19            Total Guidance keyDriver score    0.019338
34         Presentation Wage keyDriver score    0.018937
46                 Qna CapEx keyDriver score    0.018712
27           Answer Guidance keyDriver score    0.018705

5-Day Forward Return Random Forest
R^2: -0.051718322523585325
MSE: 0.21778508287292817
                                     Feature  Importance
27           Answer Guidance keyDriver score    0.030829
96  Presentation Headwinds Tailwinds key…    0.024310
64  Total Headwinds Tailwinds keyDriver …    0.021355
29          Answer Deception keyDriver score    0.020693
3              Qna Guidance keyDriver score    0.019774
9              Qna Deception keyDriver score    0.019415
31  Question Capital Raise Returns keyDr…    0.019271
16  Total Merger Acquisition keyDriver s…    0.018800
19            Total Guidance keyDriver score    0.018421
22  Total Capital Raise Returns keyDrive…    0.018031
```

3.2 XGBoost Model

[14]:
```python
for num, table in forwards:

    X_train, X_test, y_train, y_test = train_test_split(X, table, test_size=0.
↪2, random_state=42)

    param_grid = {
        "n_estimators": [100, 200],
        "learning_rate": [0.01, 0.1],
        "max_depth": [3, 5, 7],
        "subsample": [0.8, 1.0]}

    xgb_model = xgb.XGBRegressor(objective="reg:squarederror", random_state=42)
```

```python
    grid_search = GridSearchCV(xgb_model, param_grid, cv=3, scoring="r2",␣
↪verbose=1, n_jobs=-1)
    grid_search.fit(X_train, y_train)

    best_xgb = grid_search.best_estimator_
    best_params = grid_search.best_params_
    y_pred = best_xgb.predict(X_test)

    r2_xgb = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    df_xgb_imp = pd.DataFrame({"Feature": X.columns, "Importance": best_xgb.
↪feature_importances_})
    df_xgb_imp = df_xgb_imp.sort_values(by="Importance", ascending=False)

    print(f"{num}-Day Forward Return LassoCV")
    print("R^2:", r2_xgb)
    print("MSE:", mse)
    print(df_xgb_imp.head(10))
    print(" ")
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
1-Day Forward Return LassoCV
R^2: -0.008499583855449577
MSE: 0.2521172000786341
                                      Feature  Importance
107  Question Headwinds Tailwinds keyDriv…    0.014660
112     Answer CapEx keyDriver negativeScore    0.012348
39   Presentation Deception keyDriver pos…    0.011403
117        Total Exec Change keyDriver score    0.010794
37    Presentation Deception keyDriver score    0.010672
54   Qna Market Position keyDriver positi…    0.010520
47         Qna CapEx keyDriver negativeScore    0.010519
96   Presentation Headwinds Tailwinds key…    0.010306
83   Presentation CapEx keyDriver positiv…    0.010299
52       Qna Market Position keyDriver score    0.010230

Fitting 3 folds for each of 24 candidates, totalling 72 fits
3-Day Forward Return LassoCV
R^2: -0.052880535815356566
MSE: 0.24836423737923372
                                      Feature  Importance
106  Question Headwinds Tailwinds keyDriv…    0.010688
30   Answer Deception keyDriver positiveS…    0.009308
39   Presentation Deception keyDriver pos…    0.009302
120  Question Merger Acquisition keyDrive…    0.009234
119  Question Merger Acquisition keyDrive…    0.009126
```

```
95   Presentation Capital Raise Returns k…    0.009043
62   Total Market Position keyDriver nega…    0.009026
122  Presentation Exec Change keyDriver n…    0.008993
14   Total Deception keyDriver negativeScore    0.008854
96   Presentation Headwinds Tailwinds key…    0.008813

Fitting 3 folds for each of 24 candidates, totalling 72 fits
5-Day Forward Return LassoCV
R^2: -0.011655558543950395
MSE: 0.209489066547485
                                     Feature  Importance
56       Total CapEx keyDriver negativeScore    0.015138
57       Total CapEx keyDriver positiveScore    0.014905
43                Qna Margin keyDriver score    0.013694
45        Qna Margin keyDriver positiveScore    0.013383
99    Answer Market Position keyDriver score    0.013108
17   Total Merger Acquisition keyDriver n…    0.013046
84       Presentation Margin keyDriver score    0.013036
46                 Qna CapEx keyDriver score    0.012995
65   Total Headwinds Tailwinds keyDriver …    0.012912
109  Question Guidance keyDriver negative…    0.012886
```

3.3 Part 2 Reflection

- Traditional machine learning models ineffective at capturing the relationship between key-driver feature scores and forward returns, demonstrated by the low R^2 values across all models.
- At best, the LassoCV regularization model returns positive R^2 values for the single-day forward return range.
- The overall keyDriver feature importance is very low for most models, with the only exceptions being seen in the Linear Regression model, with some of the most prevalent being 'Presentation Irregularities', and 'QNA Exec Change'
- If I had more time or started from scratch, I would experiment with approaches using deep learning models, which may be able to find the relationship between features and returns more effectively.

4 3. Modeling forward stock returns with constructedcategories

- For this section, I organize, preprocess, and evaluate the data with a variety of models.

[15]:
```
# Filter irrelevant data as instructed

df_data = df_data_temp.copy()
df_data = df_data[df_data['documentType'] == 'EARNINGS_CALL']
df_data = df_data[df_data['country'] == 'US']
```

```
# Forward fill any missing identifiers by referencing earnings calls with the
↪same id
# Remove event columns as instructed
# Populate nan keydriver cells with neutral score of 0
# Populate nan Method2 Category cells with neutral scores of [0,0]

df_data = df_data.sort_values(by=['companyId', 'mainIdentifier isin'])
df_data['mainIdentifier isin'] = df_data.groupby(['companyId',
↪'companyName'])['mainIdentifier isin'].transform(lambda x: x.ffill())
df_data = df_data.rename(columns={'mainIdentifier isin': 'fsym_isin'})

event_col = [col for col in df_data.columns if 'eventcount' in col.lower()]
df_data = df_data.drop(columns = event_col)

score_col = [col for col in df_data.columns if 'score' in col.lower()]
df_data = df_data.drop(columns = score_col)

df_data.iloc[:, 28:]
df_data.iloc[:, 28:] = df_data.iloc[:, 28:].fillna('[0, 0]')

# Group Method2 Category by prefix
# Calculate the aggregate [positive,negative] score for each Category in a
↪dynamically created column

for prefix, cols in grouped_cols.items():
    new_col_name = f"{prefix}_Total"
    df_data[new_col_name] = df_data.apply(lambda row: sum_tuple_group(row,
↪cols), axis=1)

cats = list(df_data.iloc[:,28:].columns)
```

- Since [0,0] is a very inconvenient string to work with, I create two new columns dynamically
- One column will be the positive score while the other depicts the negative score.
- Then merge with the returns dataset to filter out earnings without returns or unassociated returns

[16]:
```
def safe_parse_list(value):
    try:
        if isinstance(value, str) and value.startswith("[") and value.
↪endswith("]"):
            return literal_eval(value)
        else:
            return [0, 0]
    except (ValueError, SyntaxError):
        return [0, 0]
```

```
for col in cats:
    df_data[col] = df_data[col].apply(safe_parse_list)

    df_data[[f"{col} pos", f"{col} neg"]] = pd.DataFrame(df_data[col].tolist(),␣
 ↪index=df_data.index)
    df_data.drop(columns=[col], inplace=True)


df_merge3 = pd.merge(df_data, df_retwide, on=['fsym_isin'], how='inner')

df_inputs3 = df_merge3.iloc[:,28:770]
```

- Use correlation matrix to remove highly correlated variables, although the quantity of these is very low.
- Since these categories are very sparse, the variance threshold is effective at removing the majority of catergories that are mostly 0s.

[17]:
```
mat = df_inputs3.corr()

to_remove = set()
for i in range(len(mat.columns)):
    for j in range(i):
        if abs(mat.iloc[i, j]) > 0.9:
            colname = mat.columns[i]
            to_remove.add(colname)

print(to_remove)

df_tepp = df_inputs3.drop(columns=to_remove)

selector = VarianceThreshold(threshold=0.01)  # Remove features with very low␣
 ↪variance
df_rduced = df_tepp.iloc[:, selector.fit(df_tepp).get_support()]
```

```
{'Employment - Hypothetical neg', 'Spin Off Split Off neg'}
```

4.1 Linear Regression

[18]:
```
X = df_rduced
feature_names = df_rduced.columns

#y1d = df_merge['fow_1d']
#y3d = df_merge['fow_3d']
#y7d = df_merge['fow_7d']

y1d = df_merge['fow_1d_bin']
y3d = df_merge['fow_3d_bin']
y7d = df_merge['fow_7d_bin']
```

```python
for num, table in forwards:
    X_train, X_test, y_train, y_test = train_test_split(X, table, test_size=0.
 ↪2, random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    feature_importance = np.abs(model.coef_)
    df_feat = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪feature_importance})
    df_feat = df_feat.sort_values(by="Importance", ascending=False)


    print(f"{num}-Day Forward Return Linear Regression")
    print("R^2:", r2)
    print("MSE:", mse)
    print(df_feat.head(10))
    print(" ")
```

```
1-Day Forward Return Linear Regression
R^2: -0.26443322158008
MSE: 0.31609865647388213
                                  Feature  Importance
225              Product Trial Results pos    0.485794
236              Settlement - Question neg    0.439078
212                         Employment neg    0.411603
240                Dividend - Question pos    0.325062
216   Customer Spending - Hypothetical neg    0.308125
73                           Tailwinds neg    0.280274
196             Writedowns Impairments neg    0.239461
217    Merger Acquisition Announcement pos    0.236729
172         Financial Results - Question neg    0.204233
181                 Volume - Qualifier neg    0.202834

3-Day Forward Return Linear Regression
R^2: -0.56758924101893
MSE: 0.36977899498166394
                                  Feature  Importance
225              Product Trial Results pos    0.346474
216   Customer Spending - Hypothetical neg    0.339190
227           Debt Financing - Forecast pos    0.331088
```

```
221                  Product Approval pos    0.317388
150                       Competition pos    0.307625
208                           Lawsuit neg    0.306906
67                 Strategic Alliance neg    0.301037
154    Commodity Price - Hypothetical pos    0.272337
243             Facilities - Qualifier neg    0.262748
212                        Employment neg    0.236738

5-Day Forward Return Linear Regression
R^2: -0.9583001858064479
MSE: 0.4055159628982917
                                   Feature  Importance
244                  Catastrophe Loss neg    0.556881
227           Debt Financing - Forecast pos    0.539186
215         Customer Traffic - Question neg    0.387171
212                         Employment neg    0.379476
243              Facilities - Qualifier neg    0.312463
223  Merger Acquisition - Hypothetical neg    0.309424
235             Settlement - Question pos    0.303110
154     Commodity Price - Hypothetical pos    0.289330
137       Margin Commentary - Question pos    0.268737
221                  Product Approval pos    0.266522
```

4.2 LassoCV

[19]:
```python
for num, table in forwards:
    X_train, X_test, y_train, y_test = train_test_split(X, table, test_size=0.
 ↪2, random_state=42)

    param_grid = {'alpha': np.logspace(-4, 1, 50)}

    lasso = Lasso()

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

    best_alpha = grid_search.best_params_['alpha']

    lasso_best = Lasso(alpha=best_alpha)
    lasso_best.fit(X_train, y_train)
    y_pred = lasso_best.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
```

```python
    feature_importance = np.abs(lasso_best.coef_)

    df_las_imp = pd.DataFrame({'Feature': feature_names, 'Importance':␣
↪feature_importance})
    df_las_imp = df_las_imp.sort_values(by="Importance", ascending=False)

    print(f"{num}-Day Forward Return LassoCV")
    print("R^2:", r2)
    print("MSE:", mse)
    print(df_las_imp.head(10))
    print(" ")
```

```
1-Day Forward Return LassoCV
R^2: 0.006975290872181072
MSE: 0.2482485994858776
                               Feature  Importance
33  Business Commentary - Forecast neg    0.009162
2              Financial Commentary pos    0.009026
3              Financial Commentary neg    0.008282
15             AmenityQuestionTopic neg    0.006433
7                 Financial Results neg    0.006395
42              Category Commentary pos    0.004207
9                         Euphemism neg    0.002101
26                Margin Commentary pos    0.002028
17              Business Commentary neg    0.001393
16              Business Commentary pos    0.000853

3-Day Forward Return LassoCV
R^2: -0.015382980445472594
MSE: 0.23951893021832701
                                   Feature  Importance
115                            Weather neg    0.031161
58                 Capacity Production pos    0.028664
80                    Price - Question pos    0.020368
5                           Facilities neg    0.017516
17                 Business Commentary neg    0.015964
8                            Euphemism pos    0.014641
64                    Price - Forecast pos    0.013334
68   Financial Commentary - Qualifier pos    0.012982
72                           Tailwinds pos    0.012699
19    Financial Commentary - Forecast neg    0.012107

5-Day Forward Return LassoCV
R^2: -0.004062868514151940 5
MSE: 0.20791680656878614
                                 Feature  Importance
0                              Price pos         0.0
154  Commodity Price - Hypothetical pos         0.0
```

```
156            Price - Hypothetical neg        0.0
157             Cost - Hypothetical pos        0.0
158             Cost - Hypothetical neg        0.0
159                  Consumer Trend pos        0.0
160                  Consumer Trend neg        0.0
161                  Tax - Forecast pos        0.0
162                  Tax - Forecast neg        0.0
163                  Tax - Question neg        0.0
```

4.3 Part 3 Reflection

- Just as in Part 2, traditional machine learning models experience difficulty in finding the relationship between features and postive returns, with the LassoCV regularization model offering the best R^2 value for a single-day future.
- Regardless, the linear model identified some of the most important features to be 'Product Trials - Positive', 'Customer Spending - Negative', and 'Catastrophic Loss - Negative'.
- Contextually, these features make sense. A successful product trial, being the cornerstone of a company is an essential part of success. Meanwhile, the latter two cases highlight the ramifications of two negative occurrences to a company's earnings.

[ ]:

```

```