Imports & Settings

In [68]:

```
# Core tools
import os
import numpy as np
import pandas as pd
from pandas.errors import EmptyDataError
import joblib

# Visualization tools
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Modeling tools
import xgboost
from xgboost import XGBRegressor, XGBRFRegressor
import scipy.stats as stats
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Ignoring warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
sns.set_theme()
```

Baseline OLS Model

Importing Data

In [3]:

```
df = pd.read_csv('../data/preprocessed_data/MSFT_preprocessed.csv')
df.head()
```

Out[3]:

	Report Date	Revenue	Cost of Revenue	Gross Profit	Operating Expenses	Selling, General & Administrative	Research & Development	Operating Income (Loss)	Non-Operating Income (Loss)	Pretax Income (Loss), Adj.	...	Cash from (Repurchase of) Equity	Net (
0	2009-06-30	1.309900e+10	-2.586000e+09	1.051300e+10	-6.816000e+09	-4.591000e+09	-2.225000e+09	3.697000e+09	155000000.0	3852000000	...	1.210000e+08	2.69
1	2009-09-30	1.292000e+10	-2.842000e+09	1.007800e+10	-5.596000e+09	-3.531000e+09	-2.065000e+09	4.482000e+09	283000000.0	4765000000	...	-1.292000e+09	-2.19
2	2009-12-31	1.902200e+10	-3.628000e+09	1.539400e+10	-6.881000e+09	-4.802000e+09	-2.079000e+09	8.513000e+09	370000000.0	8883000000	...	-3.138000e+09	-4.27
3	2010-03-31	1.450300e+10	-2.755000e+09	1.174800e+10	-6.575000e+09	-4.355000e+09	-2.220000e+09	5.173000e+09	168000000.0	5341000000	...	-1.601000e+09	-2.72
4	2010-06-30	1.603900e+10	-3.170000e+09	1.286900e+10	-6.939000e+09	-4.589000e+09	-2.350000e+09	5.930000e+09	94000000.0	6024000000	...	-2.927000e+09	-4.09

5 rows × 59 columns

Dropping Columns

The two object columns (`Report Date` and `Price Date`) are not needed for modeling purposes. Additionally, the `Open` , `High` , `Low` , `Adj Close` , and `Volume` columns are highly correlated with the `Close` column. Dropping these columns are necessary for modeling purposes.

In [4]:

```
df = df.drop(columns=['Report Date', 'Price Date', 'Open', 'High', 'Low', 'Adj Close', 'Volume'])
```

Fitting Model

In [5]:

```python
def fit_model(df, target='Close'):
    '''
    Description:
    ------------
    Takes a dataframe and returns a fitted OLS model with price as the dependent variable.

    Parameters:
    ------------
    df : pandas.DataFrame
        This dataframe should include all of the predictors and the target column.

    target: str
        The name of the column being predicted (dependent variable).

    Example:
    ------------
    >>> fit_model(df)
    <statsmodels.regression.linear_model.RegressionResultsWrapper>
    '''
    predictors = df.drop(columns=[target])
    predictors = sm.add_constant(predictors)
    model = sm.OLS(df[target], predictors).fit()
    return model

baseline = fit_model(df)
baseline.summary()
```

Out[5]:

OLS Regression Results

Dep. Variable:	Close	**R-squared:**	0.997
Model:	OLS	**Adj. R-squared:**	0.982
Method:	Least Squares	**F-statistic:**	66.11
Date:	Sat, 28 Aug 2021	**Prob (F-statistic):**	1.76e-05
Time:	21:59:32	**Log-Likelihood:**	-93.237
No. Observations:	42	**AIC:**	258.5
Df Residuals:	6	**BIC:**	321.0
Df Model:	35		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	66.8006	132.787	0.503	0.633	-258.118	391.720
Revenue	0.0053	0.011	0.503	0.633	-0.020	0.031
Cost of Revenue	0.0053	0.011	0.503	0.633	-0.020	0.031
Gross Profit	-0.0040	0.008	-0.503	0.633	-0.023	0.015
Operating Expenses	2.976e-05	5.92e-05	0.503	0.633	-0.000	0.000
Selling, General & Administrative	0.0013	0.003	0.503	0.633	-0.005	0.008
Research & Development	0.0013	0.003	0.503	0.633	-0.005	0.008
Operating Income (Loss)	0.0044	0.009	0.503	0.633	-0.017	0.026
Non-Operating Income (Loss)	0.0057	0.011	0.503	0.633	-0.022	0.033
Pretax Income (Loss), Adj.	-0.0057	0.011	-0.503	0.633	-0.033	0.022
Pretax Income (Loss)	-0.0042	0.008	-0.503	0.633	-0.025	0.016
Income Tax (Expense) Benefit, Net	-0.0042	0.008	-0.503	0.633	-0.025	0.016
Income (Loss) from Continuing Operations	0.0006	0.001	0.503	0.633	-0.002	0.004
Net Income	0.0035	0.007	0.503	0.633	-0.014	0.021
Net Income (Common)	0.0010	0.002	0.503	0.633	-0.004	0.006
Cash, Cash Equivalents & Short Term Investments	-1.638e-09	3.19e-09	-0.514	0.626	-9.44e-09	6.16e-09
Accounts & Notes Receivable	6.871e-10	4.07e-09	0.169	0.871	-9.27e-09	1.06e-08
Inventories	-1.615e-09	7.45e-09	-0.217	0.836	-1.98e-08	1.66e-08
Total Current Assets	0.0030	0.006	0.503	0.633	-0.011	0.017
Property, Plant & Equipment, Net	0.0001	0.000	0.503	0.633	-0.000	0.001
Long Term Investments & Receivables	0.0001	0.000	0.503	0.633	-0.000	0.001
Other Long Term Assets	0.0001	0.000	0.503	0.633	-0.000	0.001
Total Noncurrent Assets	0.0029	0.006	0.503	0.633	-0.011	0.017
Total Assets	0.0054	0.011	0.503	0.633	-0.021	0.032
Payables & Accruals	-2.997e-10	3.86e-09	-0.078	0.941	-9.75e-09	9.15e-09
Short Term Debt	-2.858e-10	1.67e-09	-0.171	0.870	-4.38e-09	3.81e-09
Total Current Liabilities	0.0007	0.001	0.503	0.633	-0.003	0.004
Long Term Debt	-5.993e-10	1.34e-09	-0.447	0.671	-3.88e-09	2.68e-09
Total Noncurrent Liabilities	0.0007	0.001	0.503	0.633	-0.003	0.004
Total Liabilities	-0.0153	0.030	-0.503	0.633	-0.089	0.059
Share Capital & Additional Paid-In Capital	-0.0183	0.036	-0.503	0.633	-0.107	0.071
Retained Earnings	-0.0183	0.036	-0.503	0.633	-0.107	0.071
Total Equity	0.0037	0.007	0.503	0.633	-0.014	0.022
Total Liabilities & Equity	0.0062	0.012	0.503	0.633	-0.024	0.036
Net Income/Starting Line	0.0011	0.002	0.503	0.633	-0.004	0.006
Depreciation & Amortization.1	0.0020	0.004	0.503	0.633	-0.008	0.012
Non-Cash Items	0.0020	0.004	0.503	0.633	-0.008	0.012
Change in Working Capital	0.0035	0.007	0.503	0.633	-0.013	0.020
Change in Accounts Receivable	-0.0015	0.003	-0.503	0.633	-0.009	0.006
Change in Inventories	-0.0015	0.003	-0.503	0.633	-0.009	0.006
Change in Accounts Payable	-0.0015	0.003	-0.503	0.633	-0.009	0.006
Change in Other	-0.0015	0.003	-0.503	0.633	-0.009	0.006
Net Cash from Operating Activities	-0.0020	0.004	-0.503	0.633	-0.012	0.008
Change in Fixed Assets & Intangibles	1.02e-08	8.02e-09	1.272	0.251	-9.43e-09	2.98e-08

Net Change in Long Term Investment	1.28e-09	2.67e-09	0.479	0.649	-5.26e-09	7.83e-09
Net Cash from Acquisitions & Divestitures	1.201e-09	2.75e-09	0.436	0.678	-5.54e-09	7.94e-09
Net Cash from Investing Activities	-5.196e-08	4.3e-08	-1.209	0.272	-1.57e-07	5.32e-08
Dividends Paid	-9.086e-09	2.54e-08	-0.357	0.733	-7.13e-08	5.31e-08
Cash from (Repayment of) Debt	2.793e-09	6.06e-09	0.461	0.661	-1.2e-08	1.76e-08
Cash from (Repurchase of) Equity	3.882e-09	6.39e-09	0.608	0.566	-1.17e-08	1.95e-08
Net Cash from Financing Activities	-5.357e-08	4.08e-08	-1.314	0.237	-1.53e-07	4.62e-08
Net Change in Cash	5.065e-08	4.22e-08	1.199	0.276	-5.27e-08	1.54e-07

Omnibus:	0.610	**Durbin-Watson:**	2.966
Prob(Omnibus):	0.737	**Jarque-Bera (JB):**	0.608
Skew:	0.263	**Prob(JB):**	0.738
Kurtosis:	2.736	**Cond. No.**	1.07e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The input rank is higher than the number of observations.
[3] The condition number is large, 1.07e+16. This might indicate that there are
strong multicollinearity or other numerical problems.

Evaluating Fitness

In [6]:

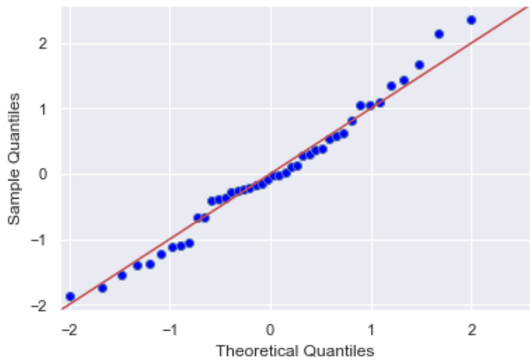
```
def print_rmse(df, target='Close', decimals=2):
    '''
    Description:
    ------------
    Takes a dataframe, splits it into train/test data, fits it to a linear regression model,
    then calculates and prints the RMSE for both the train and test portions rounded to two
    decimal places.

    Parameters:
    ------------
    df : pandas.DataFrame
        This dataframe should include all of the predictors and the target column.

    target: str
        The name of the column being predicted (dependent variable).

    decimals: int
        The number of decimals to round the output to.

    Example:
    ------------
    >>> print_rmse(df)
    Train RMSE: 100,000.00
    Test RMSE:  101,250.00
    '''
    X = df.drop(columns=[target])
    y = df[target]

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=42)
    linreg = LinearRegression()
    linreg.fit(X_train, y_train)

    y_pred_train = linreg.predict(X_train)
    y_pred_test = linreg.predict(X_test)

    rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
    rmse_test = mean_squared_error(y_test, y_pred_test, squared=False)

    print('Train RMSE:', round(rmse_train, decimals))
    print('Test  RMSE:', round(rmse_test, decimals))

print_rmse(df)
```

```
Train RMSE: 0.0
Test  RMSE: 54.76
```

Plotting Residuals

In [7]:

```python
def qqplot(model):
    '''
    Description:
    ------------
    Takes an OLS model and returns a Q-Q plot of the model residuals.

    Parameters:
    ------------
    model : a fitted statsmodels.api.OLS() model

    Example:
    ------------
    >>> get_qqplot(model)
    <matplotlib.figure.Figure>
    '''
    return sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)

qqplot(baseline);
```



Observations

- The baseline model results are too good to be true (R^2 ~99.7%) as evidenced by the p-values of each of the independent variables
- The large condition number indicates that there's an issue with multicollinearity
- There are large scaling differences between the independent variables
- The model exhibits a large degree of overfitness based on the train and test RMSE values
- While the residuals are normally distributed, this model cannot be trusted

Investigating Multicollinearity

Correlation Matrix

Renaming the columns to shortened versions in order to increase the space available for the matrix squares:

In [8]:

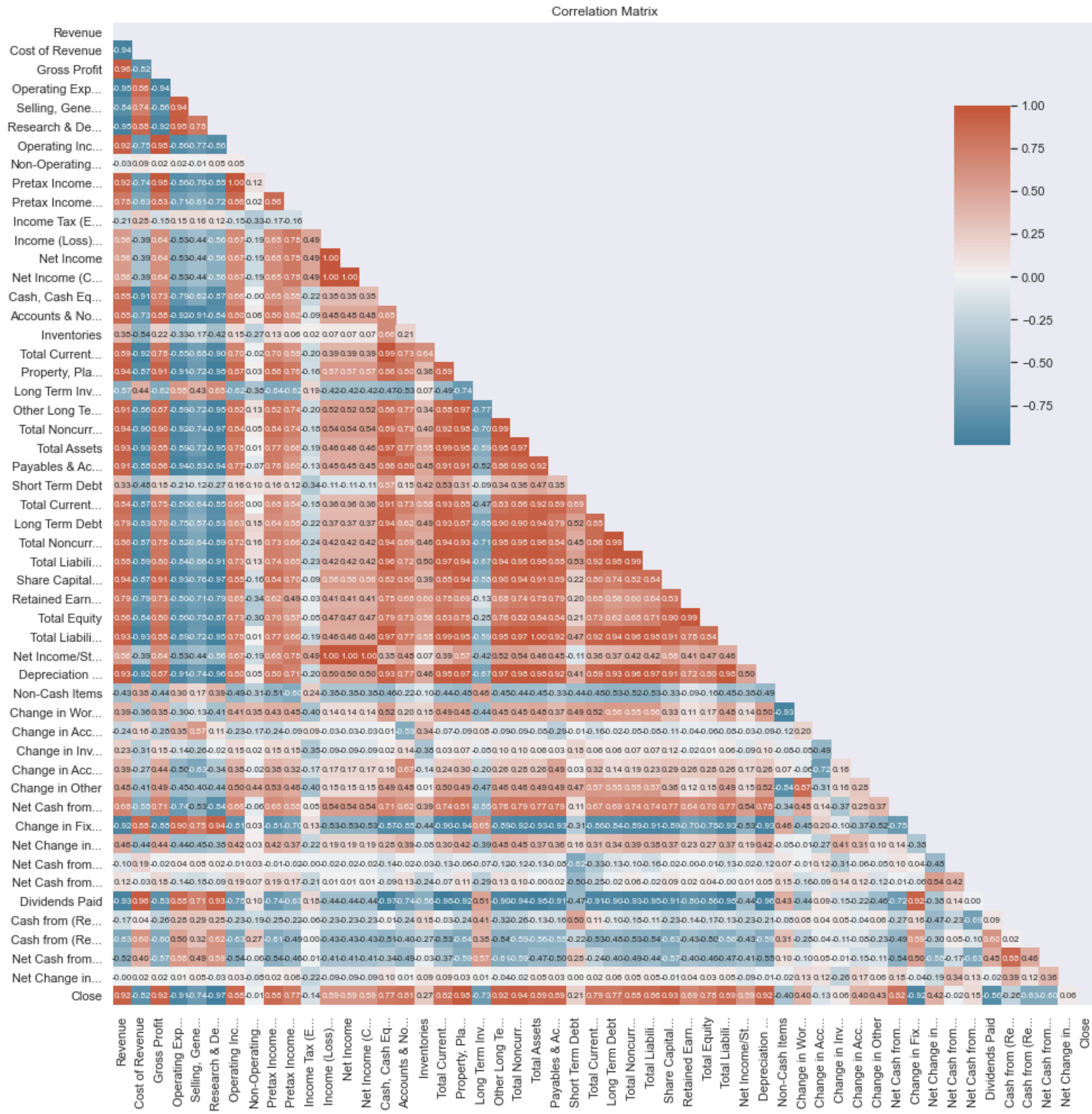
```python
df_temp = df.copy()
df_temp.columns = [col[:13]+'...' if len(col) > 15 else col for col in df.columns]
df_temp.columns
```

Out[8]:

```
Index(['Revenue', 'Cost of Revenue', 'Gross Profit', 'Operating Exp...',
       'Selling, Gene...', 'Research & De...', 'Operating Inc...',
       'Non-Operating...', 'Pretax Income...', 'Pretax Income...',
       'Income Tax (E...', 'Income (Loss)...', 'Net Income',
       'Net Income (C...', 'Cash, Cash Eq...', 'Accounts & No...',
       'Inventories', 'Total Current...', 'Property, Pla...',
       'Long Term Inv...', 'Other Long Te...', 'Total Noncurr...',
       'Total Assets', 'Payables & Ac...', 'Short Term Debt',
       'Total Current...', 'Long Term Debt', 'Total Noncurr...',
       'Total Liabili...', 'Share Capital...', 'Retained Earn...',
       'Total Equity', 'Total Liabili...', 'Net Income/St...',
       'Depreciation ...', 'Non-Cash Items', 'Change in Wor...',
       'Change in Acc...', 'Change in Inv...', 'Change in Acc...',
       'Change in Other', 'Net Cash from...', 'Change in Fix...',
       'Net Change in...', 'Net Cash from...', 'Net Cash from...',
       'Dividends Paid', 'Cash from (Re...', 'Cash from (Re...',
       'Net Cash from...', 'Net Change in...', 'Close'],
      dtype='object')
```

In [9]:

```
fig, ax = plt.subplots(figsize=(14, 14))
cbar_ax = fig.add_axes([0.85, 0.6, 0.05, 0.3])
ax.set_title('Correlation Matrix')

mask = np.triu(np.ones_like(df.corr()))
cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(df_temp.corr(), ax=ax, mask=mask, annot=True, fmt='.2f',
            annot_kws={'fontsize': 8}, square=True, cmap=cmap,
            cbar_ax=cbar_ax)
plt.tight_layout()
```



XGBoost

In [25]:

```
X = df.drop(columns=['Close'])
y = df.Close
```

XGBRegressor

Baseline

In [26]:

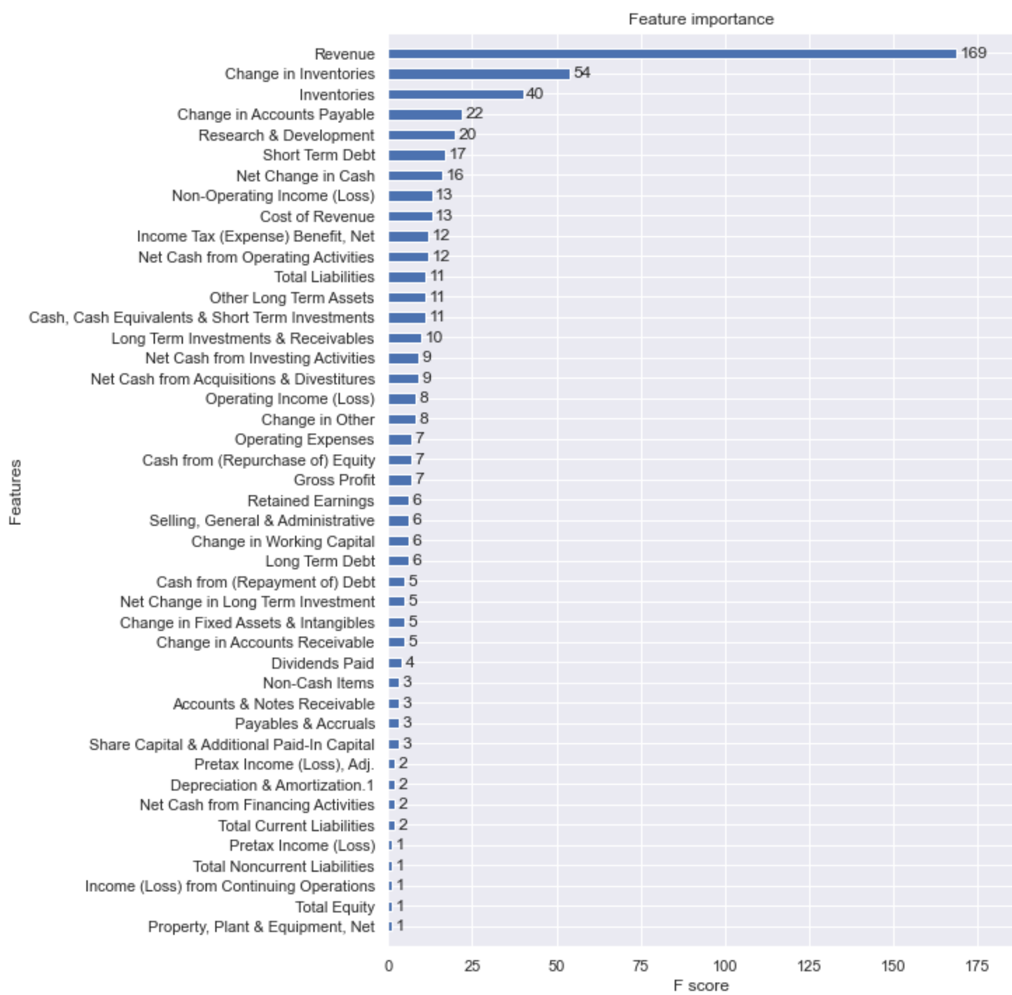
```
xgbr = xgboost.XGBRegressor(random_state=42)
xgbr.fit(X, y)
```

Out[26]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [27]:

```
fig, ax = plt.subplots(figsize=(8, 12))
xgboost.plot_importance(xgbr, ax=ax, height=0.5);
```



Hyperparameter Tuning

In [28]:

```
xgbr_params = {'max_depth': [4, 5, 6],
               'colsample_bylevel': [0.2, 0.5, 0.8],
               'colsample_bytree': [0.2, 0.5, 0.8],
               'gamma': [1, 25, 50],
               'learning_rate': [0.05, 0.25, 0.50],
               'random_state': [42]}
```

In [29]:

```
xgbr_gs = GridSearchCV(xgbr, param_grid=xgbr_params, n_jobs=-1, verbose=1)
xgbr_gs.fit(X, y)
```

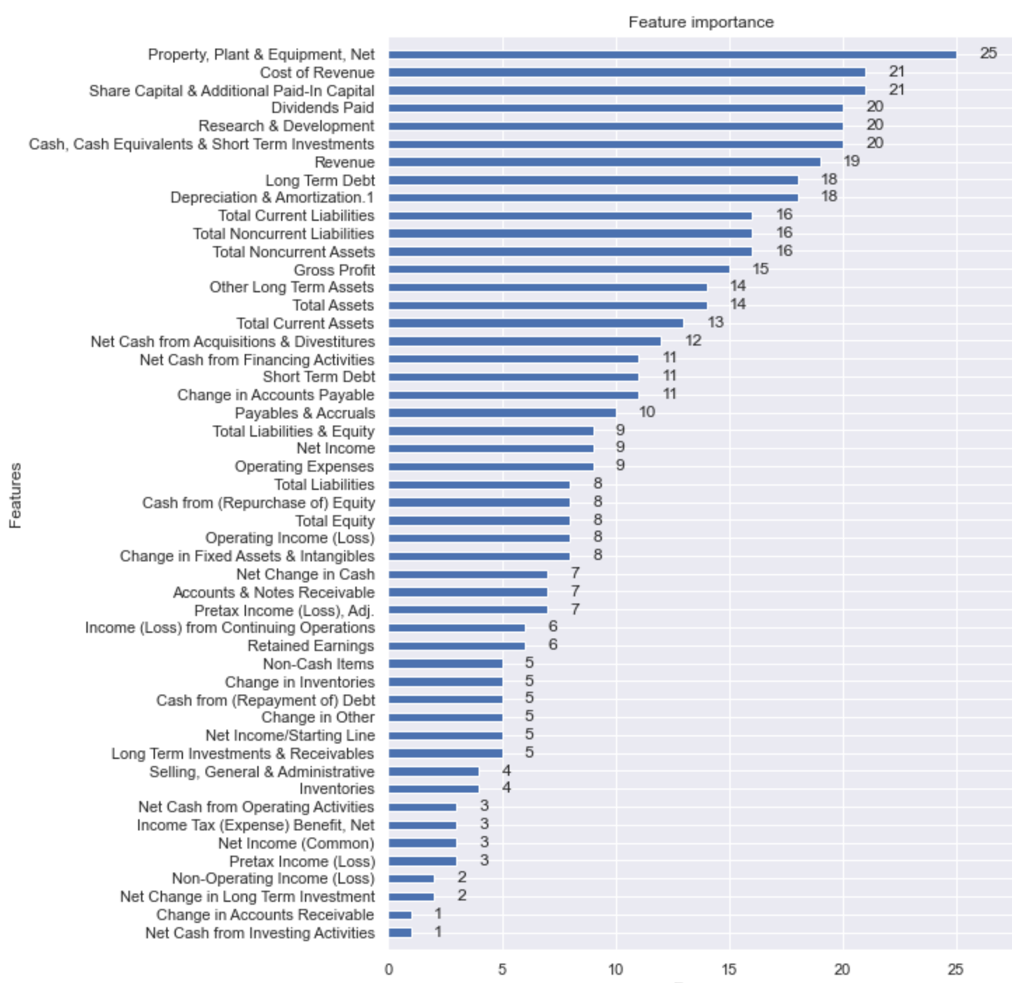
```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 1172 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done 1200 out of 1215 | elapsed:    3.4s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 1215 out of 1215 | elapsed:    3.4s finished
```

Out[29]:

```
GridSearchCV(estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                    colsample_bylevel=1, colsample_bynode=1,
                                    colsample_bytree=1, gamma=0, gpu_id=-1,
                                    importance_type='gain',
                                    interaction_constraints='',
                                    learning_rate=0.300000012, max_delta_step=0,
                                    max_depth=6, min_child_weight=1,
                                    missing=nan, monotone_constraints='()',
                                    n_estimators=100, n_jobs=0,
                                    num_parallel_tree=1, random_state=42,
                                    reg_alpha=0, reg_lambda=1,
                                    scale_pos_weight=1, subsample=1,
                                    tree_method='exact', validate_parameters=1,
                                    verbosity=None),
             n_jobs=-1,
             param_grid={'colsample_bylevel': [0.2, 0.5, 0.8],
                         'colsample_bytree': [0.2, 0.5, 0.8],
                         'gamma': [1, 25, 50],
                         'learning_rate': [0.05, 0.25, 0.5],
                         'max_depth': [4, 5, 6], 'random_state': [42]},
             verbose=1)
```

In [30]:

```
fig, ax = plt.subplots(figsize=(8, 12))
xgboost.plot_importance(xgbr_gs.best_estimator_, ax=ax, height=0.5);
```



XGBRFRegressor

Baseline

In [31]:

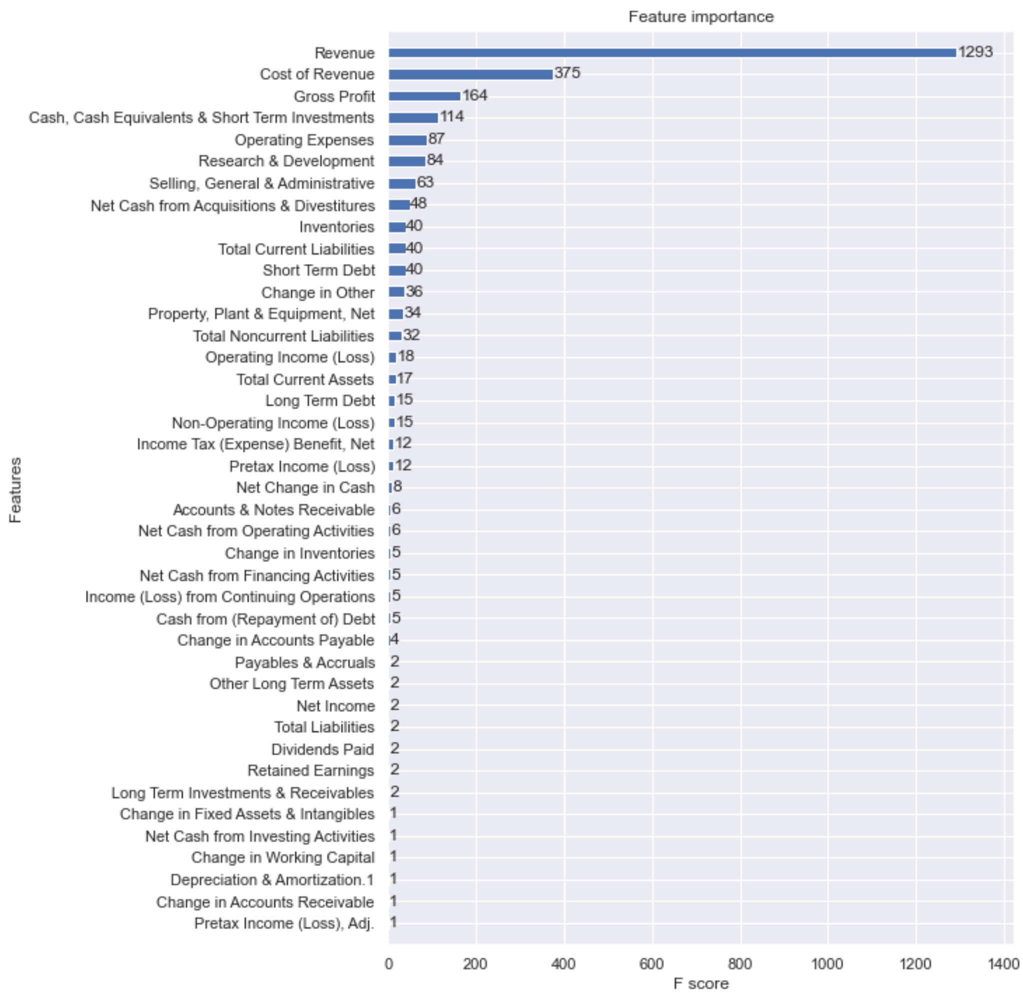
```
xgb_rf = XGBRFRegressor(random_state=42)
xgb_rf.fit(X, y)
```

Out[31]:

```
XGBRFRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
               colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain',
               interaction_constraints='', max_delta_step=0, max_depth=6,
               min_child_weight=1, missing=nan, monotone_constraints='()',
               n_estimators=100, n_jobs=0, num_parallel_tree=100,
               objective='reg:squarederror', random_state=42, reg_alpha=0,
               scale_pos_weight=1, tree_method='exact', validate_parameters=1,
               verbosity=None)
```

In [32]:

```
fig, ax = plt.subplots(figsize=(8, 12))
xgboost.plot_importance(xgb_rf, height=0.5, ax=ax);
```



Hyperparameter Tuning

In [48]:

```
xgb_rf_params = {'max_depth': [4, 5, 6, 7, 8],
                 'colsample_bylevel': [0.2, 0.5, 0.8],
                 'colsample_bytree': [0.2, 0.5, 0.8],
                 'gamma': [1, 25, 50],
                 'learning_rate': [0.01, 0.05, 0.25, 0.50],
                 'random_state': [42]}
```

In [49]:

```
xgb_rf_gs = GridSearchCV(xgb_rf, param_grid=xgb_rf_params, n_jobs=-1, verbose=1)
xgb_rf_gs.fit(X, y)
```

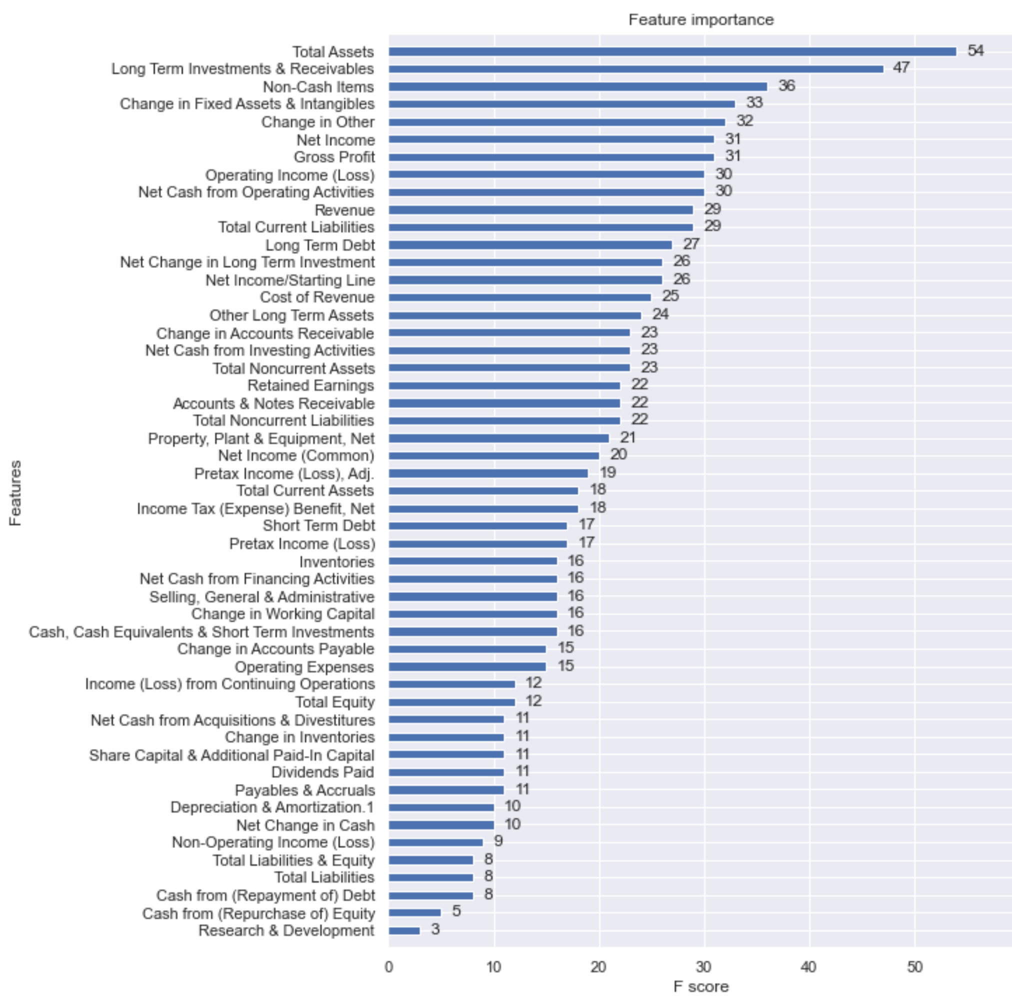
```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 2160 tasks      | elapsed:    4.3s
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed:    5.9s finished
```

Out[49]:

```
GridSearchCV(estimator=XGBRFRegressor(base_score=0.5, booster='gbtree',
                                      colsample_bylevel=1, colsample_bytree=1,
                                      gamma=0, gpu_id=-1,
                                      importance_type='gain',
                                      interaction_constraints='',
                                      max_delta_step=0, max_depth=6,
                                      min_child_weight=1, missing=nan,
                                      monotone_constraints='()',
                                      n_estimators=100, n_jobs=0,
                                      num_parallel_tree=100,
                                      objective='reg:squarederror',
                                      random_state=42, reg_alpha=0,
                                      scale_pos_weight=1, tree_method='exact',
                                      validate_parameters=1, verbosity=None),
             n_jobs=-1,
             param_grid={'colsample_bylevel': [0.2, 0.5, 0.8],
                         'colsample_bytree': [0.2, 0.5, 0.8],
                         'gamma': [1, 25, 50],
                         'learning_rate': [0.01, 0.05, 0.25, 0.5],
                         'max_depth': [4, 5, 6, 7, 8], 'random_state': [42]},
             verbose=1)
```

In [50]:

```
fig, ax = plt.subplots(figsize=(8, 12))
xgboost.plot_importance(xgb_rf_gs.best_estimator_, height=0.5, ax=ax);
```



In [51]:

```
xgb_rf_gs.best_params_
```

Out[51]:

```
{'colsample_bylevel': 0.2,
 'colsample_bytree': 0.2,
 'gamma': 50,
 'learning_rate': 0.5,
 'max_depth': 4,
 'random_state': 42}
```

Exporting Models

In [55]:

```
snp_tickers = pd.read_csv('../data/sp500.csv').Symbol.to_list()
```

In [77]:

```
for i, ticker in enumerate(snp_tickers):
    # Manually catching BF.B
    if ticker == 'BF.B':
        ticker = 'BF-B'

#     # Skipping files already generated
#     if os.path.isfile(f'../models/{ticker}_model.joblib'):
#         continue

    try:
        # Loading in data
        df = pd.read_csv(f'../data/preprocessed_data/{ticker}_preprocessed.csv')
        df = df.drop(columns=['Report Date', 'Price Date', 'Open', 'High', 'Low', 'Adj Close', 'Volume'])
        df.columns = [col.replace('.', '') for col in df.columns]  # Prevents issue with the dashboard

        # Setting features and target
        X = df.drop(columns=['Close'])
        y = df.Close

    except EmptyDataError as e:
        continue

    except KeyError as e:
        print(f'KeyError issue with {ticker} - skipping')
        continue

    # Modeling
    try:
        params = {'max_depth': [4, 5, 6, 7, 8],
                  'colsample_bylevel': [0.2, 0.5, 0.8],
                  'colsample_bytree': [0.2, 0.5, 0.8],
                  'gamma': [1, 25, 50],
                  'learning_rate': [0.01, 0.05, 0.25, 0.50],
                  'random_state': [42]}
        model = GridSearchCV(xgb_rf, param_grid=params, n_jobs=-1, verbose=0)
        model.fit(X, y)

    except ValueError:
        print(f'Modeling error with {ticker} - skipping')
        continue

    # Exporting model
    best = model.best_estimator_
    joblib.dump(best, f'../models/{ticker}_model.joblib')

    # Printing progress
    print(f'{i+1}/{len(snp_tickers)} -- modeled and saved {ticker.ljust(5)}', end='\r')
```

```
KeyError issue with CZR - skipping
KeyError issue with CDW - skipping
KeyError issue with DOW - skipping
KeyError issue with IQV - skipping
Modeling error with TER - skipping
KeyError issue with UA - skipping
Modeling error with ZBRA - skipping
505/505 -- modeled and saved ZTS
```