In [459]:
```
# Importing Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

# Model Evaluation Metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from scipy import stats
np.warnings.filterwarnings('ignore')

# Importing Models
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import xgboost as xgb
```

BUILDING LINEAR REGRESSION MODEL

TRAINING AND TESTING

In [460]:
```
# Loading Dataset

df=pd.read_csv('G1.csv')
df_test=pd.read_csv('G2.csv')
```

In [461]:
```
df.head()
```

Out[461]:

	TimeStamp	AssetGroup	Identifier	F2	F3	F4	F5	q	F7	F8	...	F24	F25	F26	F27	F28	F29	F30	F31
0	1998-12-31	G1	86a1f1ee	bc068363	Capital Goods	Diversified Capital Goods	1	1	B1	14.0	...	0	1	7.0	0.0	1	4.0	0.0	1
1	1998-12-31	G1	de5b9bca	e5fb34b9	Telecommunications	Telecom - Wireline Integrated & Services	6	1	BB2	12.0	...	-4	-4	NaN	NaN	-4	NaN	NaN	-4
2	1998-12-31	G1	2a0a4ee3	62835655	Basic Industry	Chemicals	1	1	B1	14.0	...	-4	-4	NaN	NaN	-4	NaN	NaN	-4
3	1998-12-31	G1	a67fb965	81f811c5	Transportation	Rail	2	0	B2	15.0	...	-4	-4	NaN	NaN	-4	NaN	NaN	-4
4	1998-12-31	G1	52f697cc	2becce58	Basic Industry	Building Materials	4	0	BB2	12.0	...	1	1	7.0	0.0	1	4.0	0.0	1

5 rows × 35 columns

In [462]:
```
df_test.head()
```

Out[462]:

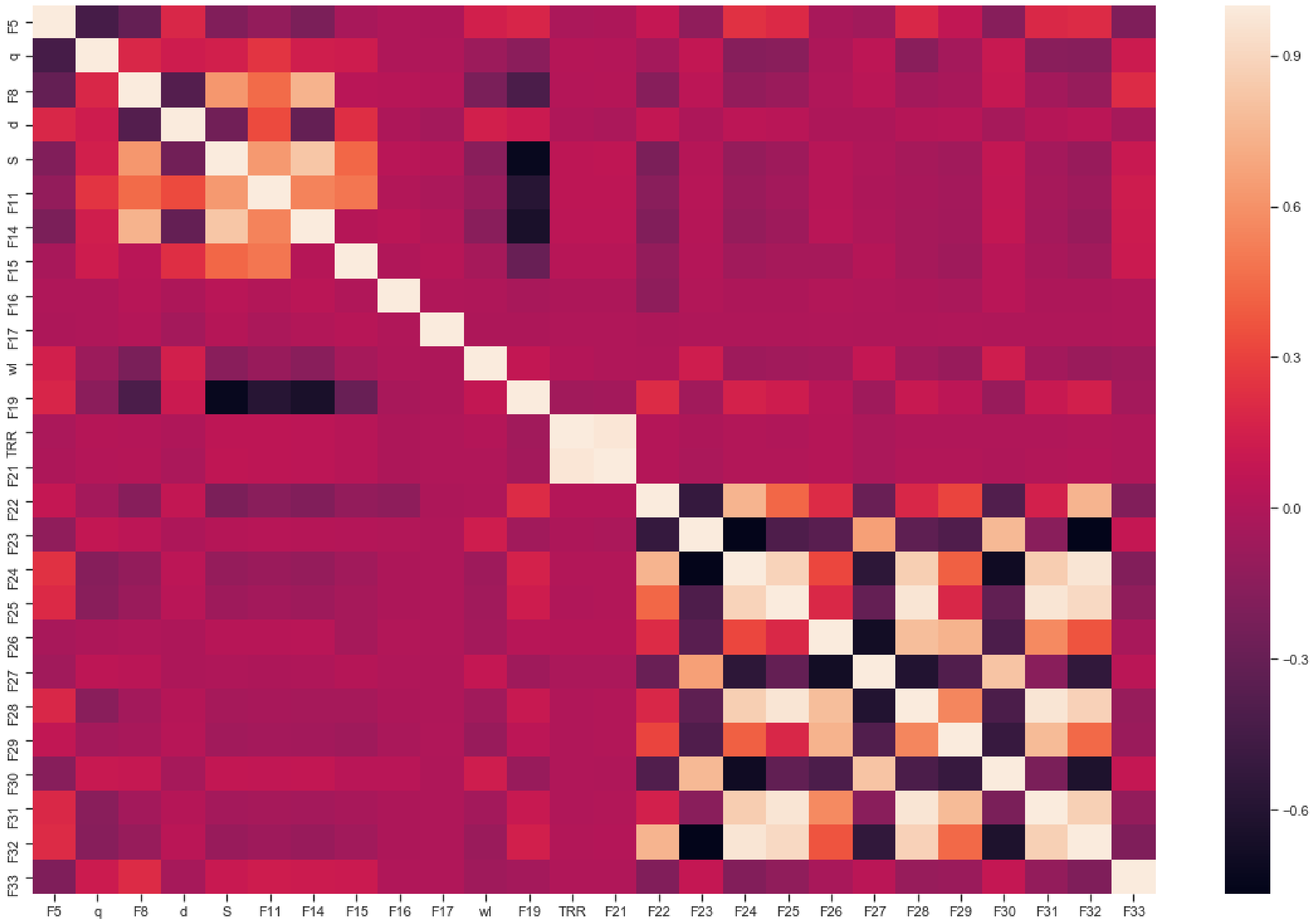
	TimeStamp	AssetGroup	Identifier	F2	F3	F4	F5	q	F7	F8	...	F24	F25	F26	F27	F28	F29	F30
0	1997-12-31	G2	2050c8b3	134e3828	Telecommunications	Telecom - Integrated/Services	2	0	B2	15.0	...	-3	0	3.0	7.0	0	0.0	6.0
1	1997-12-31	G2	69bd765f	134e3828	Telecommunications	Telecom - Integrated/Services	2	1	B2	15.0	...	-3	0	3.0	7.0	0	0.0	6.0
2	1997-12-31	G2	87093289	5b42c631	Services	Building & Construction	2	0	B1	14.0	...	-3	0	3.0	7.0	0	0.0	6.0
3	1997-12-31	G2	4ce55a61	45ef6ce3	Media	Media - Broadcast	1	1	B2	15.0	...	-1	1	4.0	0.0	0	0.0	0.0
4	1997-12-31	G2	0aa1ccf8	5b42c631	Services	Building & Construction	2	1	B1	14.0	...	-3	0	3.0	7.0	0	0.0	6.0

5 rows × 35 columns

Correlation Heatmap

In [463]:
```
# # Correlation Heatmap

plt.subplots(figsize=(20,13 ))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=False)
```

Out[463]: <matplotlib.axes._subplots.AxesSubplot at 0x1a3add3470>



Finding Correlation of features with Target Variable (TRR)

In [464]:
```
# Correlation of features with target feature (Return)
correlation_matrix['TRR'].sort_values(ascending = False)
```

Out[464]:
```
TRR     1.000000
F21     0.976311
S       0.060496
F11     0.059398
F14     0.050005
F15     0.027253
F26     0.023759
q       0.020033
F22     0.017913
F8      0.017284
wI      0.015083
F32     0.008805
F24     0.006245
F17     0.003687
F29     0.003516
F28     0.003441
F31     0.002817
F25     0.002541
F30     0.000393
F33    -0.000862
d      -0.006628
F16    -0.011144
F23    -0.011933
F5     -0.015971
F27    -0.019085
F19    -0.058083
Name: TRR, dtype: float64
```

We found that F21 is the only feature having Correlation > 0.9 with TRR

We Choose top 4 highest correlated features

In [465]:
```
# # Retaining numerical features only (1st Attempt)
# numerical_df = df.drop(['TimeStamp', 'Identifier','F7', 'F13', 'AssetGroup', 'F2', 'F3', 'F4', 'F12', 'F15', 'F22', 'F23', 'F26', 'F27', 'F29', 'F30'], axis=1)
# numerical_df_test = df_test.drop(['TimeStamp', 'Identifier','F7', 'F13', 'AssetGroup', 'F2', 'F3', 'F4', 'F12', 'F15', 'F22', 'F23', 'F26', 'F27', 'F29', 'F30'], axis=1)

# Retaining Top 4 features
numerical_df = df[['F21','S','F11','F14', 'TRR']]
numerical_df_test = df_test[['F21','S','F11','F14', 'TRR']]
```

In [466]:
```
numerical_df.describe()
```

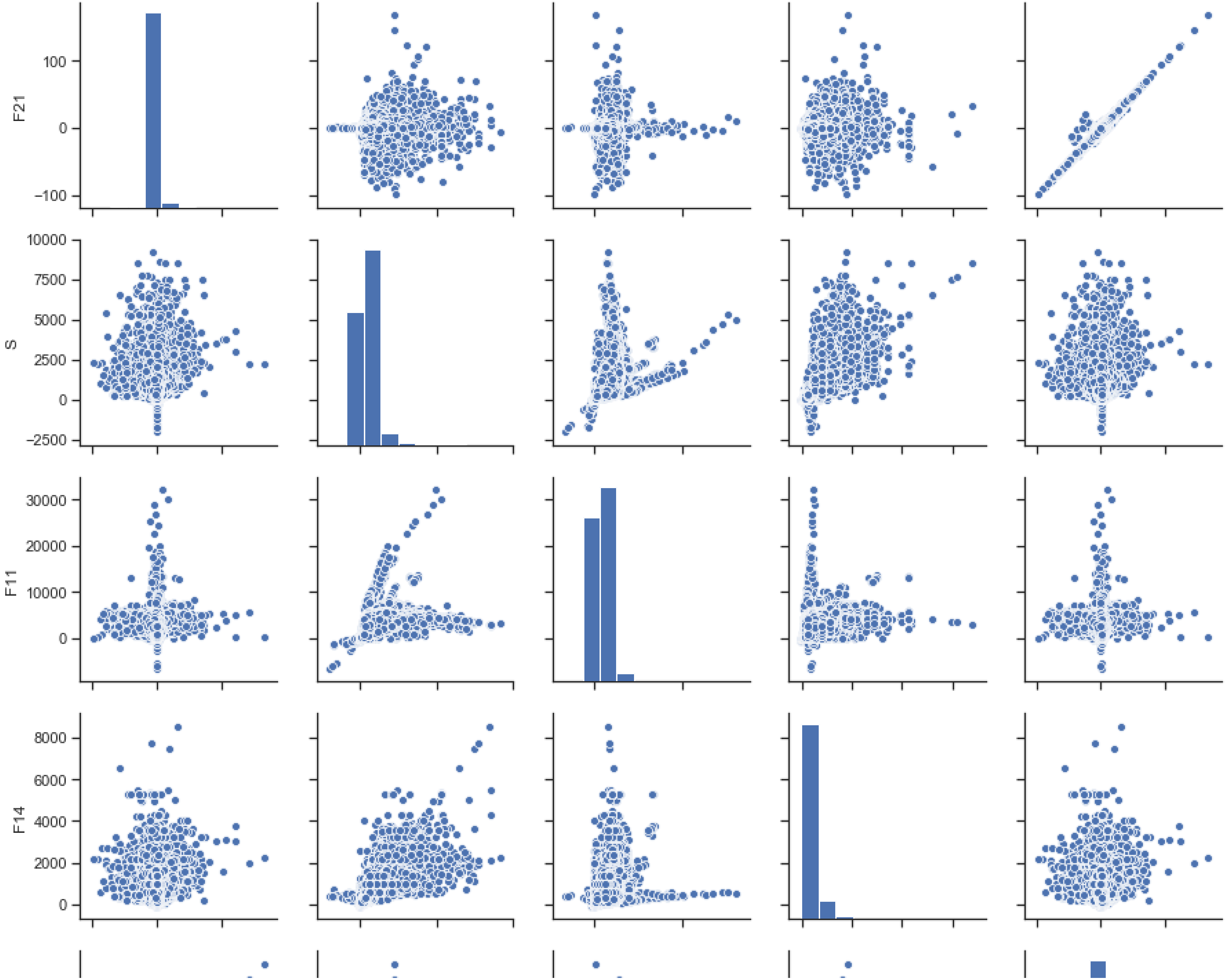
Out[466]:

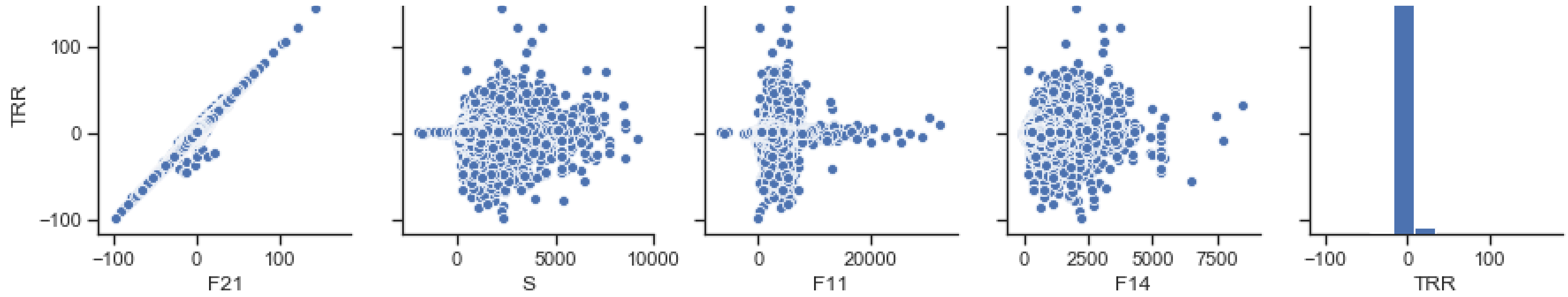
	F21	S	F11	F14	TRR
count	100483.000000	100483.000000	100483.000000	100483.000000	100483.000000
mean	0.258733	472.465064	1714.639863	436.339858	0.496910
std	5.371406	528.222779	1401.960279	401.651737	5.342896
min	-98.008000	-1980.000000	-6514.200000	-47.000000	-98.000000
25%	-0.517000	178.000000	632.940000	187.000000	-0.357000
50%	0.281000	325.000000	1390.146000	318.000000	0.542000
75%	1.328000	576.000000	2439.505000	569.500000	1.606000
max	166.665000	9180.000000	32101.083000	8511.000000	166.667000

Pair Plots between features

We can clearly see that, Plot b/w F21 vs TRR depicts high positive Correlation

In [424]:
```
# PairPlot between F21 and TRR (highly correlated)
# NOTE: Running this snippet takes around 2-3 Minutes.

sns.set(style="ticks")
sns.pairplot(numerical_df)
plt.savefig('pairplots_coloured')
```


F21
100
0
−100
S
10000
7500
5000
2500
0
−2500
F11
30000
20000
10000
0
F14
8000
6000
4000
2000
0



Removing outliers using Z-Score Technique

In [467]:
```
numerical_df = numerical_df[(np.abs(stats.zscore(numerical_df)) < 3).all(axis=1)]
numerical_df_test = numerical_df_test.dropna()
numerical_df_test = numerical_df_test[(np.abs(stats.zscore(numerical_df_test)) < 3).all(axis=1)]
```

In [468]:
```
numerical_df.shape
```

Out[468]:
```
(95757, 5)
```

In [469]:
```
numerical_df_test.shape
```

Out[469]:
```
(65998, 5)
```

In [470]:
```
# Preparing input and target arrays for the model

# Separating out the target
y = numerical_df.loc[:,['TRR']].values
y_new= numerical_df_test.loc[:,['TRR']].values

# Separating out the Input
x = numerical_df.drop(['TRR'], axis=1).values
x_new=numerical_df_test.drop(['TRR'],axis=1).values
```

FEATURE ENGINEERING

Scaling Features

In [471]:
```
x = StandardScaler().fit_transform(x)
x_new = StandardScaler().fit_transform(x_new)
```

In [472]:
```
# 1. Applying PCA (not required)

# pca = PCA(n_components=10)
# principalComponents = pca.fit_transform(x)
# principalComponents_new = pca.fit_transform(x_new)

# principalDf = pd.DataFrame(data = principalComponents)
# principalDf_new = pd.DataFrame(data = principalComponents_new)

# TRR = pd.DataFrame(data=y)
# X = np.array(principalDf)
# X_new = np.array(principalDf_new)
```

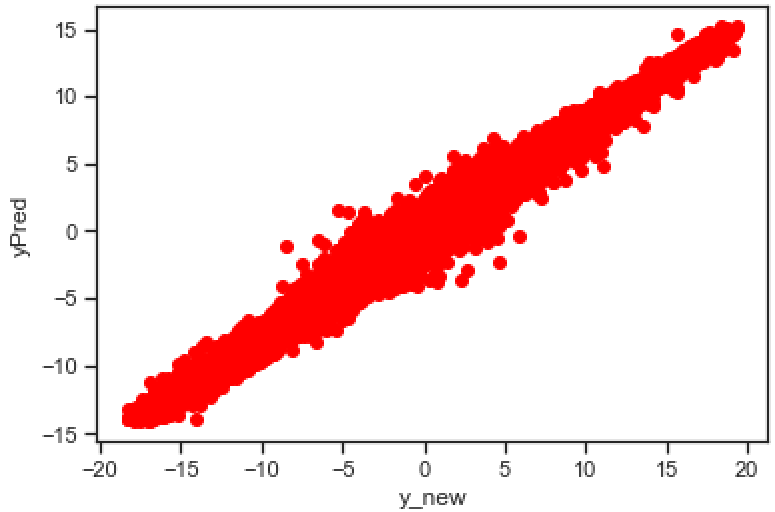
In [473]:
```
# Converting into array for model
X = np.array(x)
X_new = np.array(x_new)
```

Applying Linear Regression Model

In [474]:
```
model = LinearRegression().fit(X, y)

print("Multi Linear Regression using PCA :")

yPred = model.predict(X_new)

print("mean squared error is : " + str(mean_squared_error(y_new, yPred)))
print("R^2 value is : " + str(r2_score(y_new, yPred)))

plt.scatter(y_new, yPred, color='red')
plt.xlabel('y_new')
plt.ylabel('yPred')
```

```
Multi Linear Regression using PCA :
mean squared error is : 0.7763567455867468
R^2 value is : 0.9206360252866739
```

Out[474]: Text(0, 0.5, 'yPred')

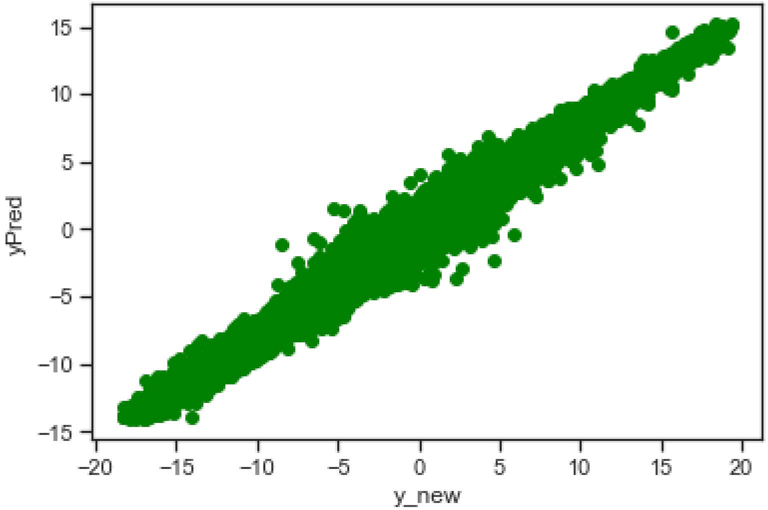


Gradient Boosting using XG-Boost

In [475]:
```
print("Gradient Boosting :")

xgdmat = xgb.DMatrix(X,y)
our_params = {'eta':0.1,'seed':0,'subsample':0.8,'colsample_bytree':0.8,'objective':'reg:squarederror','max_d
epth':3,'min_child_weight':1}
final_gb = xgb.train(our_params,xgdmat)
tesdmat = xgb.DMatrix(X_new)
y_pred = final_gb.predict(tesdmat)

print("Mean sqaured error:" + str(mean_squared_error(y_new,y_pred)))
print("R^2 score :" + str(r2_score(y_new,y_pred)))

plt.scatter(y_new, yPred, color='green')
plt.xlabel('y_new')
plt.ylabel('yPred')
```

```
Gradient Boosting :
Mean sqaured error:3.317360241681815
R^2 score :0.6608789762793252
```

Out[475]:
```
Text(0, 0.5, 'yPred')
```



We can Clearly see that Linear Regression Model Works far better than XG-Boost.

PEDICTING RETURNS USING TRAINED MODEL

In [445]:
```
# PREDIT RETURNS

months = ['2016-01-31','2016-02-29','2016-03-31','2016-04-30','2016-05-31','2016-06-30','2016-07-31','2016-08-31',
            '2016-09-30','2016-10-31','2016-11-30','2016-12-31','2017-01-31','2017-02-28','2017-03-31','2017-04-30','2017-05-31','2017-06-30','2017-07-31','2017-08-31',
            '2017-09-30','2017-10-31','2017-11-30','2017-12-31','2018-01-31','2018-02-28','2018-03-31','2018-04-30','2018-05-31','2018-06-30','2018-07-31','2018-08-31',
            '2018-09-30','2018-10-31','2018-11-30']
stocks = []

for i in range(0, len(months)):
    stocksDF = df_test[df_test['TimeStamp']==months[i]][['F21','S','F11','F14']]
    stocksDF = stocksDF.dropna()
    stocksDF = stocksDF[(np.abs(stats.zscore(stocksDF)) < 3).all(axis=1)]

    x = stocksDF.values
    x = StandardScaler().fit_transform(x)
    X = np.array(x)

    returns = model.predict(X)
    returns = returns.transpose()
    stocks.append(returns[0])
```

In [446]:
```
stocks = pd.DataFrame(data=stocks).iloc[:, 0:400]
stocks['Months'] = months
stocks.set_index('Months', inplace=True)
```

In [447]:
```
stocks.to_csv('predicted_stocks.csv')
```

In [448]:
```
stocks
```

Out[448]:

Months	0	1	2	3	4	5	6	7	8	9	...	390	391	
2016-01-31	-1.064075	-1.980852	0.643682	0.975394	1.523719	0.676484	-2.240913	4.643552	-1.681585	2.099946	...	-0.118219	0.631477	0.
2016-02-29	1.236957	1.403737	2.519866	-3.650119	-1.895302	-1.702685	4.879950	-1.758377	0.532961	-0.293455	...	-1.017750	-1.550674	-1.
2016-03-31	-1.708303	-1.599713	-1.296885	-1.709868	-1.584342	-0.600672	-0.721768	-1.051162	-1.296362	-0.246934	...	5.665293	6.471781	1.
2016-04-30	4.426131	-5.227894	1.581452	3.734957	0.310967	0.565480	1.006582	3.209713	0.893781	1.205560	...	1.109316	2.201871	-0.
2016-05-31	0.482746	0.124049	3.374706	0.968938	1.087234	-3.293495	1.466974	10.347726	7.644094	-6.315749	...	2.047965	-0.342362	1.
2016-06-30	3.751863	0.203278	1.678704	-7.313451	1.293308	-1.540526	0.367314	-4.372270	-1.630925	-1.027476	...	-0.679458	10.151568	-1.
2016-07-31	1.173334	-1.419655	1.752400	3.938241	-0.718527	-0.542570	-1.114226	-1.103356	-1.832726	2.115933	...	2.226617	2.131549	-1.
2016-08-31	1.203405	0.107329	3.130945	-2.162963	0.398774	2.525931	-1.039352	-0.846990	-1.864126	0.438213	...	0.032450	1.791382	1.
2016-09-30	-1.533112	7.443263	1.438134	0.872181	-0.805087	1.797281	-1.022359	-0.863389	1.464756	0.642637	...	-2.118449	1.834585	-2.
2016-10-31	-0.873246	-0.950045	-0.541938	3.060396	0.948985	-4.273043	-1.580536	-0.180231	-0.550394	0.761423	...	0.726583	-2.095292	1.
2016-11-30	1.166947	-0.914919	-2.107762	1.026485	1.189798	4.215140	0.803487	5.092062	-1.375378	2.286877	...	-1.634392	-1.682967	-1.
2016-12-31	-0.337516	-0.359161	0.678695	-0.349821	3.057935	0.576095	1.937550	1.157069	-1.508958	-1.087182	...	-0.867544	0.659811	1.
2017-01-31	1.654599	-0.132457	0.864241	0.765883	0.529671	0.401642	-0.925294	1.340229	0.313116	-0.906073	...	-0.009329	-0.286296	-1.
2017-02-28	1.980106	1.391754	1.304432	0.193348	2.292781	2.132909	-2.700351	2.798651	0.871421	1.793262	...	8.095858	1.618285	8.
2017-03-31	-1.427682	-1.334107	-1.704285	-0.904351	-2.034351	-0.947038	1.604677	3.351632	1.893913	8.498827	...	1.727614	-0.198172	-1.

	0	1	2	3	4	5	6	7	8	9	...	390	391	
Months														
2017-04-30	-7.216011	1.031624	3.871621	-0.053184	5.538793	0.909771	-0.118858	0.412573	0.365817	1.220704	...	1.237609	4.714986	-0.
2017-05-31	0.397094	1.220182	-0.483483	-0.459840	-1.537129	-4.604775	0.462660	-0.645662	-0.349818	5.268383	...	2.495575	-0.518144	-0.
2017-06-30	-0.470007	0.114416	3.255923	0.437616	-1.022626	0.561894	16.427445	8.944439	1.432916	1.875033	...	1.088606	-0.497985	0.
2017-07-31	0.181384	0.542860	0.421713	-1.307839	-1.897947	3.177442	-1.812016	0.989782	1.064224	1.677674	...	2.742645	5.098989	0.
2017-08-31	-0.066208	-0.285502	-0.768059	-1.315670	10.792680	-1.246141	-1.097757	-1.071718	-0.150284	-0.033383	...	-7.297064	2.211845	-1.
2017-09-30	-0.298843	-1.988551	-1.057641	-0.435691	1.333060	1.810108	0.723702	-1.018624	-0.134721	1.749160	...	0.313691	1.388758	-0.
2017-10-31	2.360076	0.949338	1.932139	4.092743	0.736166	2.247921	-1.740225	2.177945	-0.062553	2.112980	...	0.507492	0.203856	1.
2017-11-30	-0.568924	1.030452	1.509808	-1.468123	2.764231	0.138701	2.364114	-1.332872	1.925931	0.171254	...	-0.724040	1.419332	-0.
2017-12-31	2.712608	-0.972675	0.766733	-1.612262	-0.174286	-1.853094	-2.112412	-1.796098	0.340650	-2.195414	...	1.930892	0.982182	-6.
2018-01-31	-1.032606	-0.170226	1.435702	1.687953	2.241946	-2.941250	-0.542954	3.468505	3.208014	4.251696	...	0.862224	1.653488	2.
2018-02-28	1.451341	3.084981	-2.339006	1.536137	2.107943	-1.566456	2.928036	2.505583	3.574783	-1.238405	...	1.685151	2.084093	-0.
2018-03-31	-0.742891	-2.307243	0.896646	1.630425	-0.579697	0.300364	2.098274	-0.462311	-0.315294	2.690433	...	0.971026	-0.846174	2.
2018-04-30	2.398130	1.539342	2.665579	-5.837084	1.283074	2.916097	-2.020995	2.010891	-1.325661	0.710036	...	2.720842	1.802095	2.
2018-05-31	1.605734	0.918214	-2.625116	3.854188	1.075546	1.476819	1.273072	-0.062588	-3.559595	1.973795	...	1.858147	1.704687	4.
2018-06-30	-0.508055	-0.352081	1.250319	-0.335732	-2.150423	-0.598170	-1.923884	-0.066686	0.389553	-0.103138	...	1.208812	2.241970	-0.

Months	0	1	2	3	4	5	6	7	8	9	...	390	391	
2018-07-31	-3.709656	-0.704590	3.784479	1.551871	1.516044	1.221064	1.575642	-1.338070	0.737259	-1.101382	...	1.454971	1.468334	-0.
2018-08-31	-2.125058	1.154835	-0.165798	-1.640554	-0.213928	1.052459	2.232745	-4.281518	0.923669	2.925432	...	0.478082	2.315572	12.
2018-09-30	-0.885803	0.972763	2.680431	1.498846	-0.225780	0.808457	2.753934	4.533666	2.832032	2.619015	...	1.547886	0.841125	1.
2018-10-31	1.743694	2.453796	0.877272	2.240769	-0.751477	2.440556	1.593360	-0.755927	0.557834	1.678471	...	1.608451	1.040771	-2.
2018-11-30	-0.407710	-8.456073	-2.538831	0.723214	1.620739	4.393334	0.789359	1.786383	1.815526	0.851766	...	1.503730	-0.589081	2.

35 rows × 400 columns

FINDING OPTIMAL WEIGHTS

Using Predicted Stock Returns

In [449]:
```
df = pd.read_csv('predicted_stocks.csv')
# Building Portfolio using 4 stocks
df = df.iloc[:, 0:5]
df.head()
```

Out[449]:

	Months	0	1	2	3
0	2016-01-31	-1.064075	-1.980852	0.643682	0.975394
1	2016-02-29	1.236957	1.403737	2.519866	-3.650119
2	2016-03-31	-1.708303	-1.599713	-1.296885	-1.709868
3	2016-04-30	4.426131	-5.227894	1.581452	3.734957
4	2016-05-31	0.482746	0.124049	3.374706	0.968938

Statistical Properties of stocks

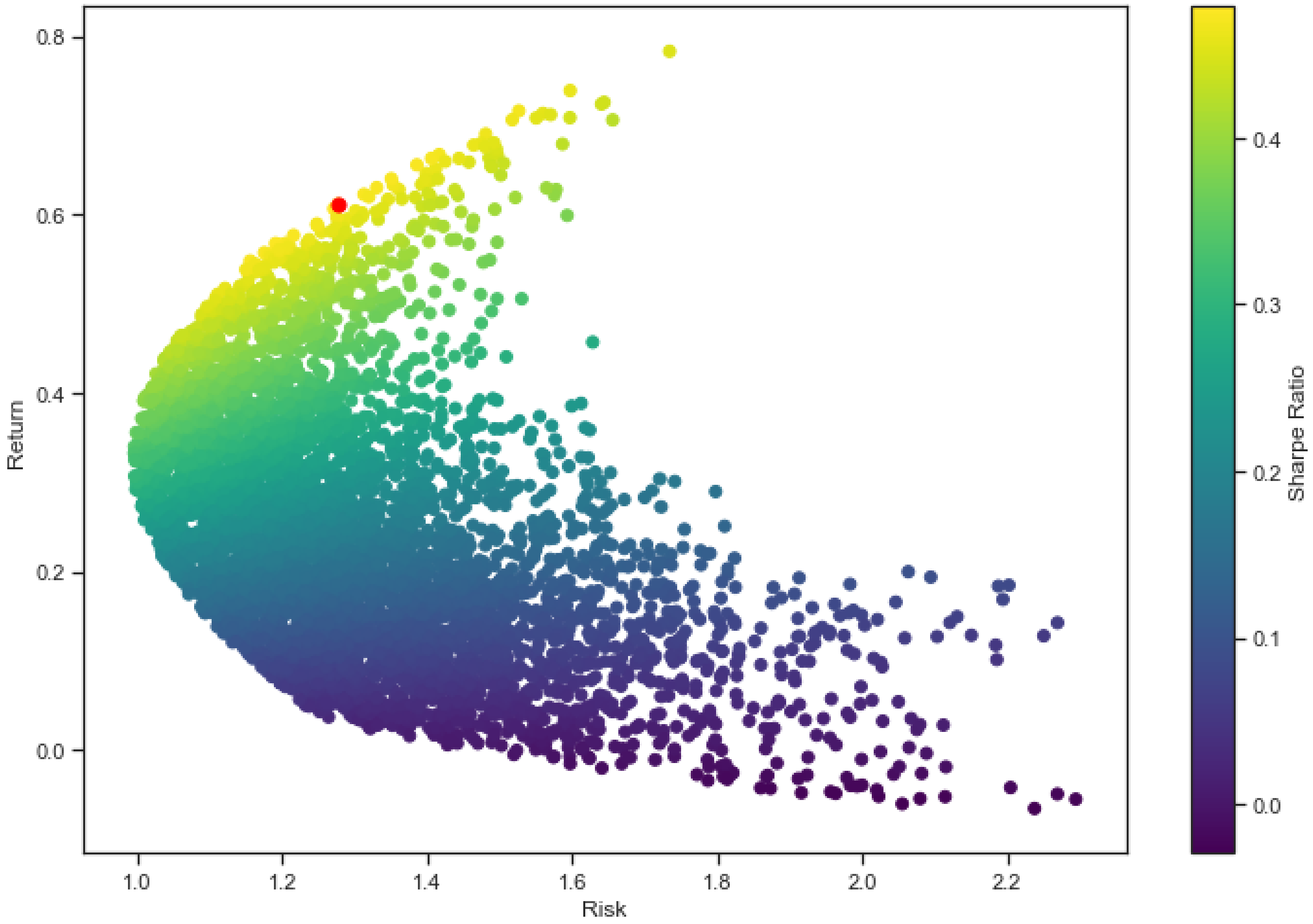
In [450]:
```
mean_return = df.mean(axis=0)  # Stocks Return
std_dev = df.std(axis=0)  # Stocks Risk (Standard Deviation)
cov_matrix = np.matrix(df.cov())  # Stocks Covariance Matrix
corr_matrix = df.corr()  # Stocks Correlation Matrix
```

Building Portfolios

In [451]:
```
import random  # For generating weights

ports = 5000  # Count of Total Portfolios
(m, n) = df.shape

portfolios = []  # Set of Portfolios
all_weights = []  # Set of Portfolio Weights

for i in range(1, ports):
#    Generating Weights
    w = [np.sqrt(random.random()*random.random())*(random.random()*50)
         for i in range(1,n)]
    s = sum(w)
    weight = [ i/s for i in w ]
    all_weights.append(weight)
#     Portfolio Properties
    portfolio_return = np.dot(weight, mean_return)
    variance = np.matmul(np.matmul(weight, cov_matrix), np.transpose(weight))
    portfolio_std_dev = np.sqrt(variance[0,0])
    sharpe_ratio = portfolio_return / portfolio_std_dev  # Assuming Rf=0
#     Add Portfolio to the list
    portfolios.append((portfolio_return, portfolio_std_dev, sharpe_ratio))

portfolios = pd.DataFrame(portfolios,
                          columns=['Return', 'Std. Dev.', 'Sharpe Ratio'])
```

Plotting Portfolios and finding Optimal Weights

In [454]:
```
# Plot Portfolios
plt.figure(figsize=(12,8))
plt.scatter(portfolios.iloc[:,1], portfolios.iloc[:,0],
            c=portfolios.iloc[:,2], cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Risk')
plt.ylabel('Return')

# Find Point with Maximum sharpe Ratio
idx = np.argmax(portfolios.iloc[:,2])
plt.scatter(portfolios.iloc[idx, 1], portfolios.iloc[idx, 0], c='red', s=50) # Plotting the point (red dot)

plt.show()
```



Optimal Portfolio

In [455]:
```
# Return Weights with Max Sharpe Ratio
opt_weight = all_weights[idx]
opt_weight
```

Out[455]:
```
[0.16449135664403777,
 0.009532201282198751,
 0.7003787493298251,
 0.1255976927439384]
```

Thank You!