Name: Argha Kamal Samanta

Roll number: 21EC30012

Department: Electronics & Electrical Communication Engineering

Kaggle username: arghakamalsamanta

Introduction:

- **Objective**

  The mission of this challenge is to leverage advanced feature engineering and sophisticated modeling techniques to forecast stock returns during earnings announcement periods. The objective is to achieve precise and reliable predictions while avoiding the common pitfalls of overfitting.

- **Overview of the Task**

  The task involves meticulous data analysis to uncover hidden patterns and create novel features that enhance the model's predictive power. By applying expertise in quantitative finance, the goal is to develop sophisticated models capable of accurately forecasting stock returns during crucial earnings announcement periods.

Techniques Explored:

1. XGBoost with 0-imputation

```
params = {
    'objective': 'reg:squarederror',
    'max_depth': 6,
    'eta': 0.3,
    'eval_metric': 'rmse'
}
```

Validation MSE: 0.004379940009694669

2. **Linear regression with median impute**

```
# Create a LinearRegression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Validation MSE: 0.0041627172984071 86

3. **XGBoost with median impute**

```
xg_reg = xgb.XGBRegressor(objective='reg:squarederror',
                          colsample_bytree=0.3, learning_rate=0.1,
                          max_depth=5, alpha=10, n_estimators=100)
```

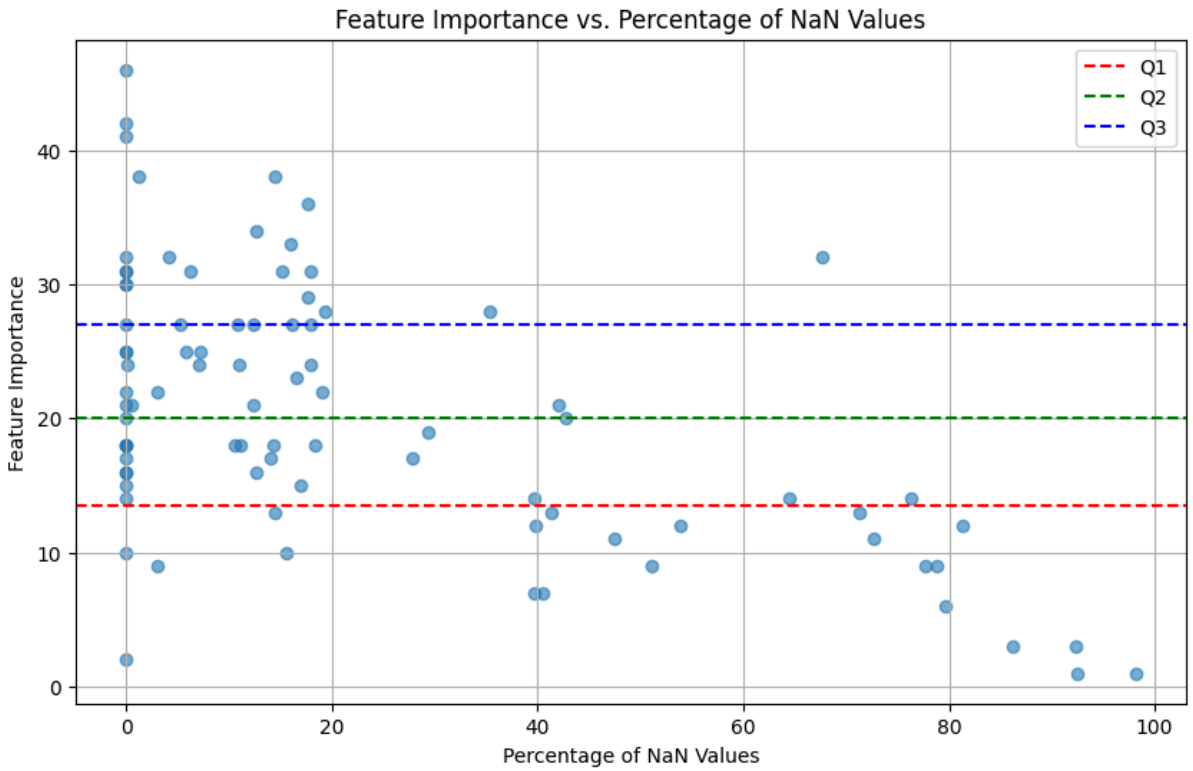
Validation MSE: 0.0041608592417112 78

4. **Feature importance using the model mentioned above**

```
feature_importance = xg_reg.get_booster().get_score(importance_type='weight')
```

	Features	Importance	NaN %
1	f2	46.00	0.00
0	f1	42.00	0.00
29	f30	41.00	0.00
42	f43	38.00	1.18
26	f27	38.00	14.43
83	f84	36.00	17.63
15	f16	34.00	12.63
50	f51	33.00	15.99

5. Scatter plot of Feature Importance vs missing value %




```
q1 = 13.5, q2 = 20.0, q3 = 27.0
```

6. Column drop using the scatter plot

```
cols_to_drop = []
for _, row in xgb_imp.iterrows():
    if row['Importance'] < (q1 + q2)/2 and row['NaN %'] > 50:
        cols_to_drop.append(row['Features'])
cols_to_drop
```

7. Applied XGBoost after dropping columns

```
xg_reg = xgb.XGBRegressor(objective='reg:squarederror',
                          colsample_bytree=0.3, learning_rate=0.1,
                          max_depth=5, alpha=10, n_estimators=100)
```

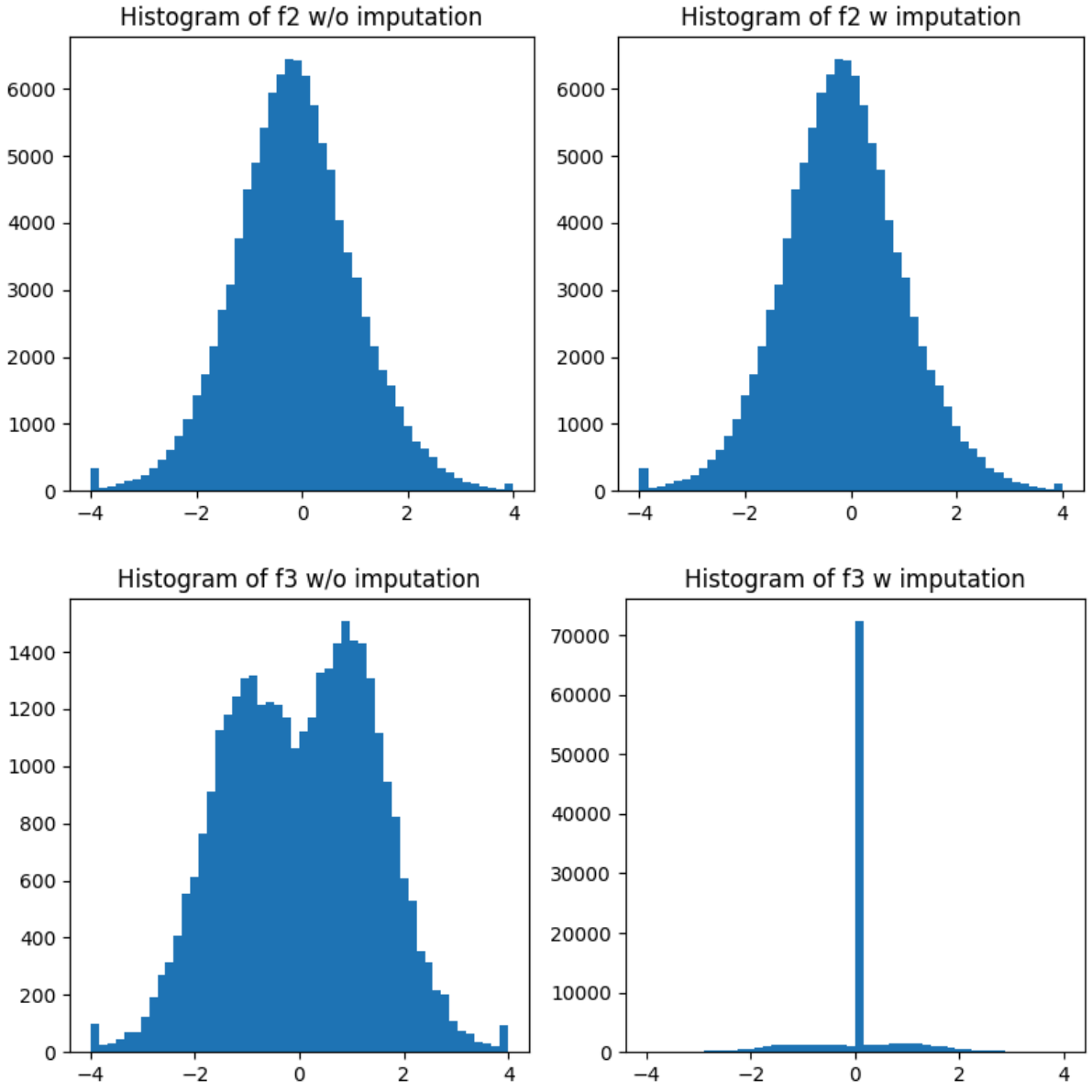
Validation MSE: 0.004158074503013681

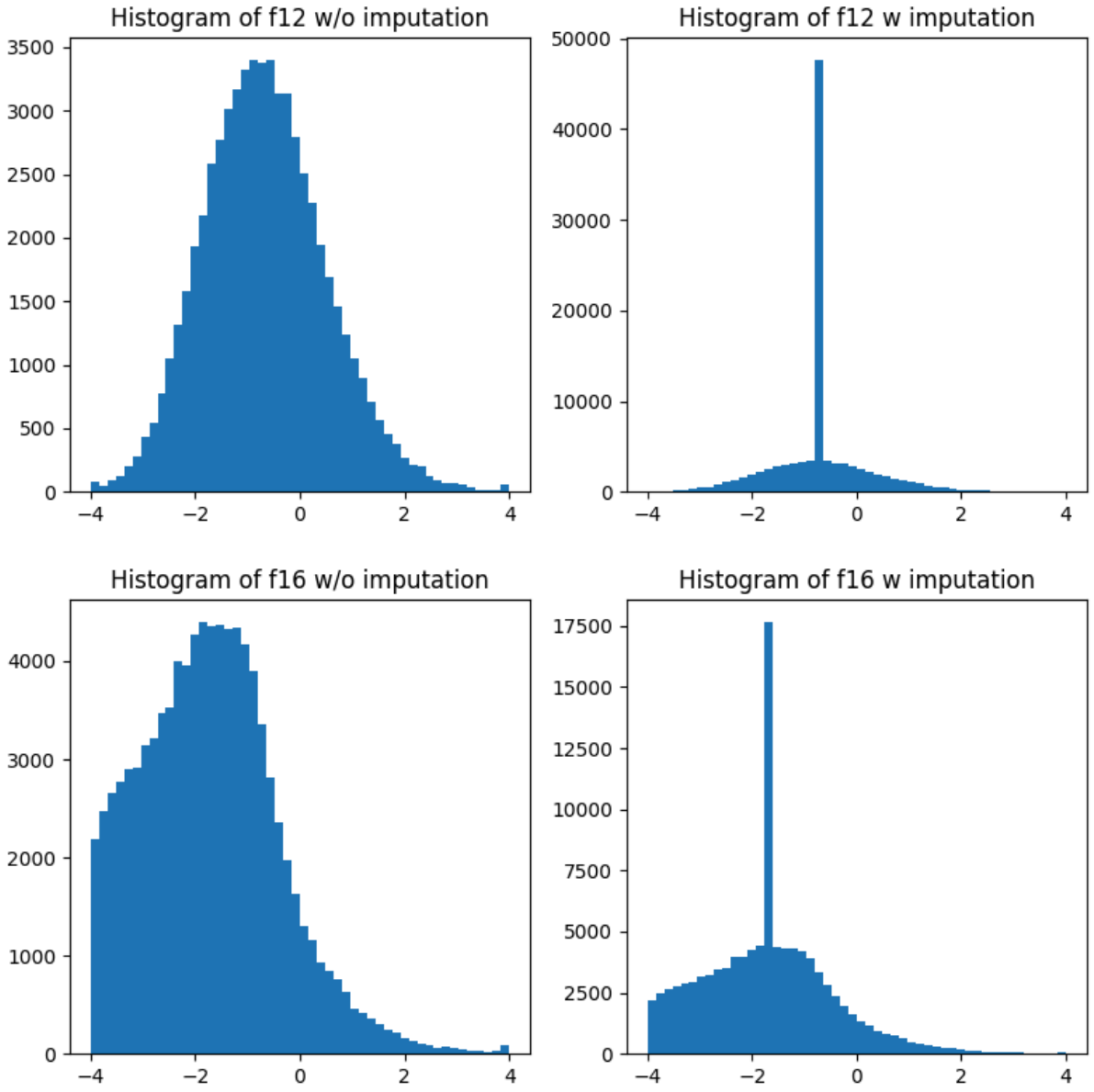
8. **Applied Linear regression after dropping columns**

```
# Create a LinearRegression model
model = LinearRegression()
```

Validation MSE: 0.0041617156704659 56

9. **Checked the effect of median imputation**





*** Median imputation is significantly affecting the original distribution of most of the features. Thus, more than median imputation is needed!**

10. XGBoost with dropped columns without median imputation

*** XGBoost can internally handle missing values optimally by maximizing the information gain.**
Validation MSE: 0.00414692142414937 *(MSE got improved!)

11. Hyperparameter tuning

```
# Initialize the model
xg_reg = xgb.XGBRegressor(objective='reg:squarederror')

# Define the parameter grid
param_grid = {
    'colsample_bytree': [0.2, 0.3, 0.4],
    'learning_rate': [0.01, 0.1],
    'max_depth': [2, 3, 5, 7],
    'alpha': [1, 10, 100],
    'n_estimators': [100, 200]
}

# Initialize the Grid Search
grid_search = GridSearchCV(estimator=xg_reg,
                           param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=3, verbose=1)

```

Validation MSE: 0.00414555434672254

12. XGBoost with no column dropping and no impute

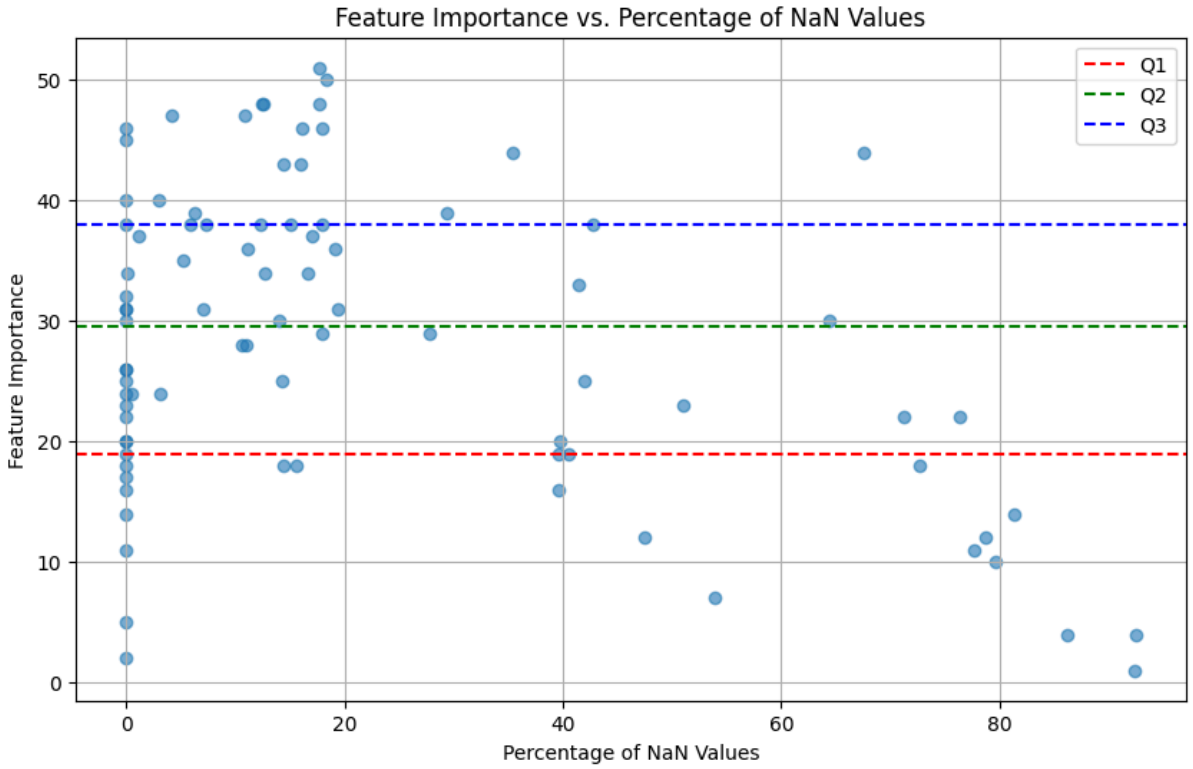
Validation MSE: 0.00414619116982528

13. Tuned the hyperparameters again to get the feature importance

	Features	Importance	NaN %
82	f84	51.00	17.63
80	f82	50.00	18.37
23	f24	48.00	17.71
15	f16	48.00	12.63
7	f8	48.00	12.39
62	f64	47.00	4.21
57	f59	47.00	10.84
22	f23	46.00	16.14

*** Different features than before!**

14. Scatter plot




```
1 (q1, q2, q3)

(19.0, 29.5, 38.0)
```

*q1, q2, q3 are the quartiles

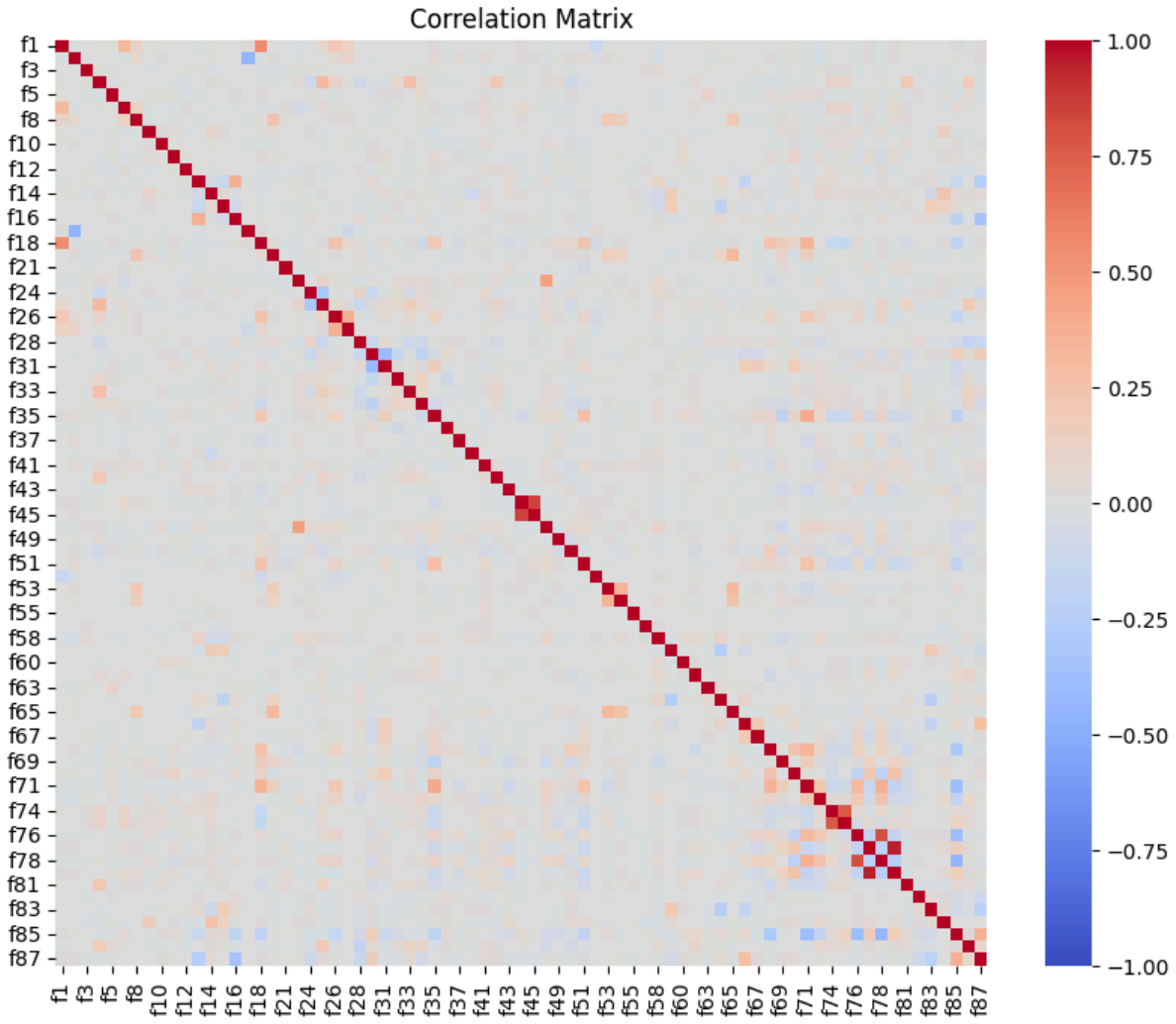
15. Dropped columns based on the recent scatter plot

```
cols_to_drop_no_impute = []
for _, row in xgb_imp_no_impute.iterrows():
    if row['Importance'] < (q1 + q2)/2 and row['NaN %'] > 50:
        cols_to_drop_no_impute.append(row['Features'])
cols_to_drop_no_impute
```

16. XGBoost with newly dropped columns with no imputation

Validation MSE: 0.004144750611941534

17. Correlation matrix of the remaining features



18. Further dropped one of the highly correlated features

```
threshold = 0.8

# Find pairs of highly correlated features
high_corr_pairs = [(feature1, feature2, corr_matrix.loc[feature1, feature2])
                   for feature1 in corr_matrix.columns
                   for feature2 in corr_matrix.columns
                   if feature1 != feature2 and abs(corr_matrix.loc[feature1, feature2]) > threshold]

# Display highly correlated pairs
high_corr_pairs
```

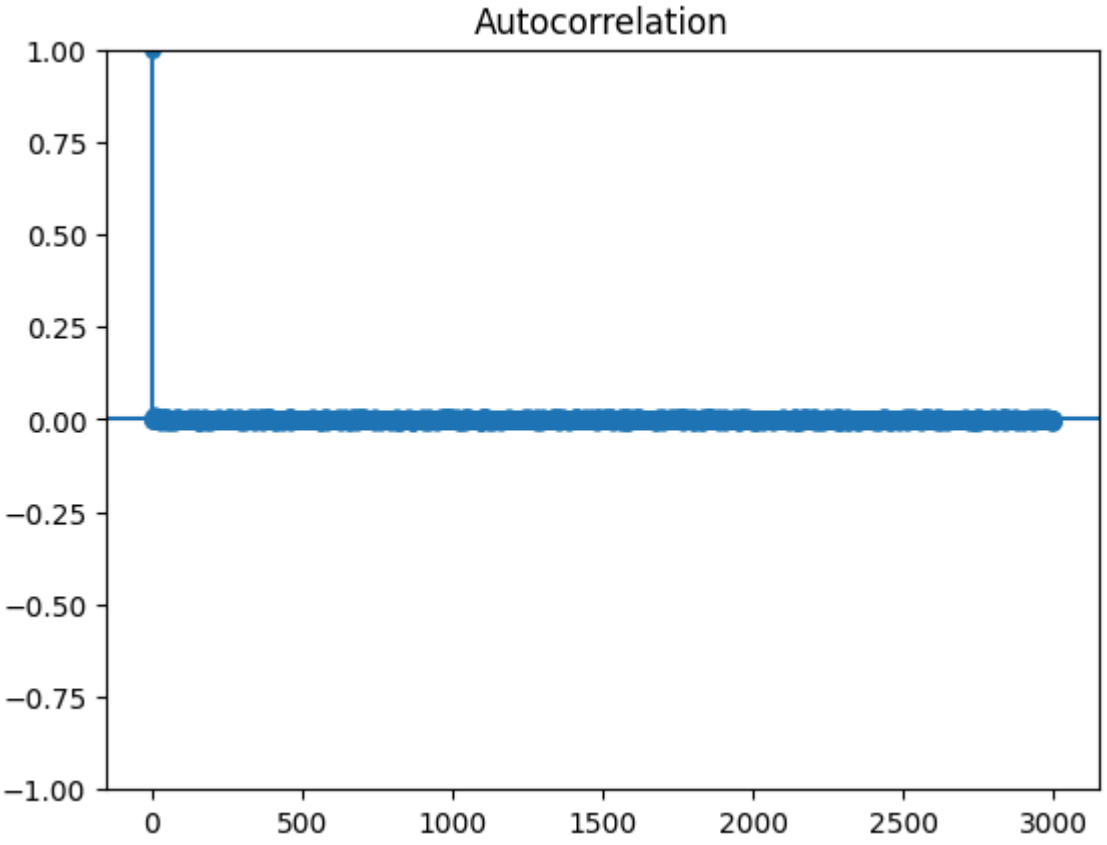
```
[('f44', 'f45', 0.8496947060205254),
 ('f45', 'f44', 0.8496947060205254),
 ('f76', 'f78', 0.8012811275932767),
 ('f77', 'f79', 0.9488068312182714),
 ('f78', 'f76', 0.8012811275932767),
 ('f79', 'f77', 0.9488068312182714)]
```

19. **XGBoost, after dropping highly correlated columns with no imputation**

Validation MSE: 0.004144225486120989

***** <u>Exploring time series forecasting</u> *****

20. **Autocorrelation of the target to check seasonality**



*** No significant seasonality in target**

21. **For prediction on any test data point, used the train data having row_id less than that of the test data point, thus not using the future**

*** Gave very poor result: MSE of 0.0070116 on submission**

22. Spline interpolation on target

*** Not applicable, as it requires unique data points in any feature to map the target with**

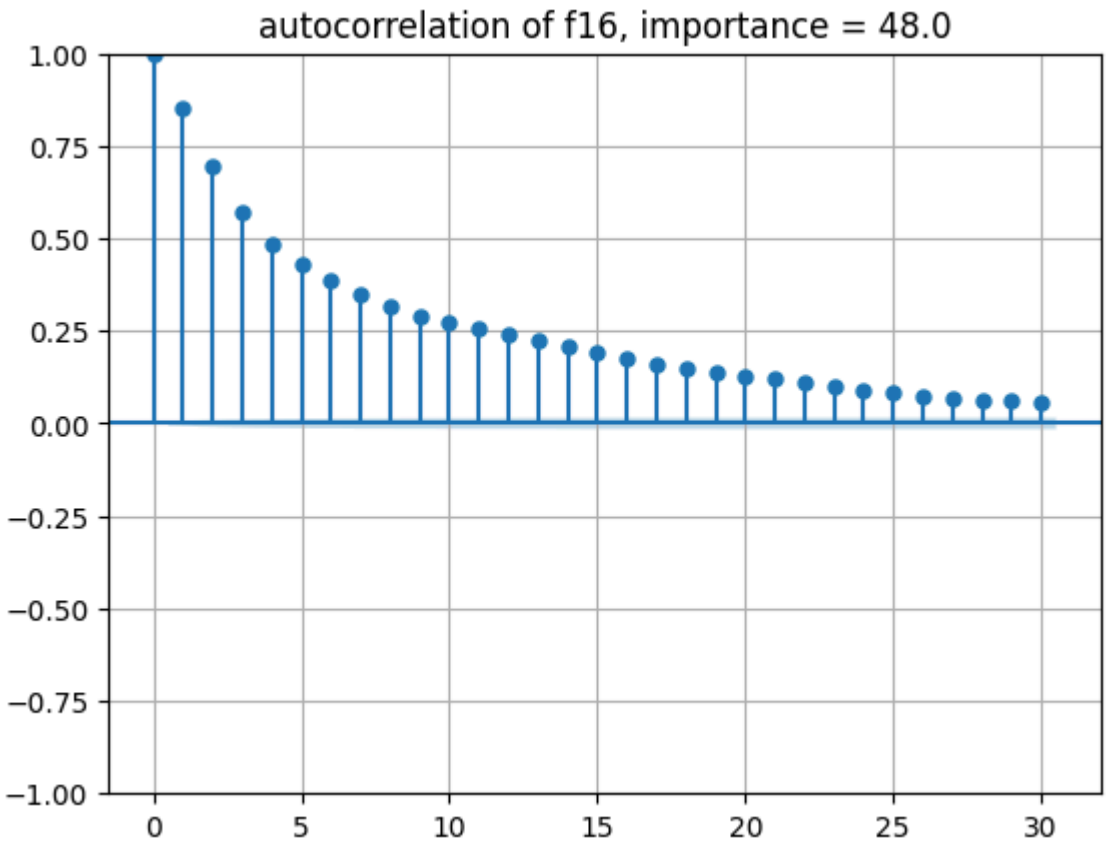
23. KNN Imputation

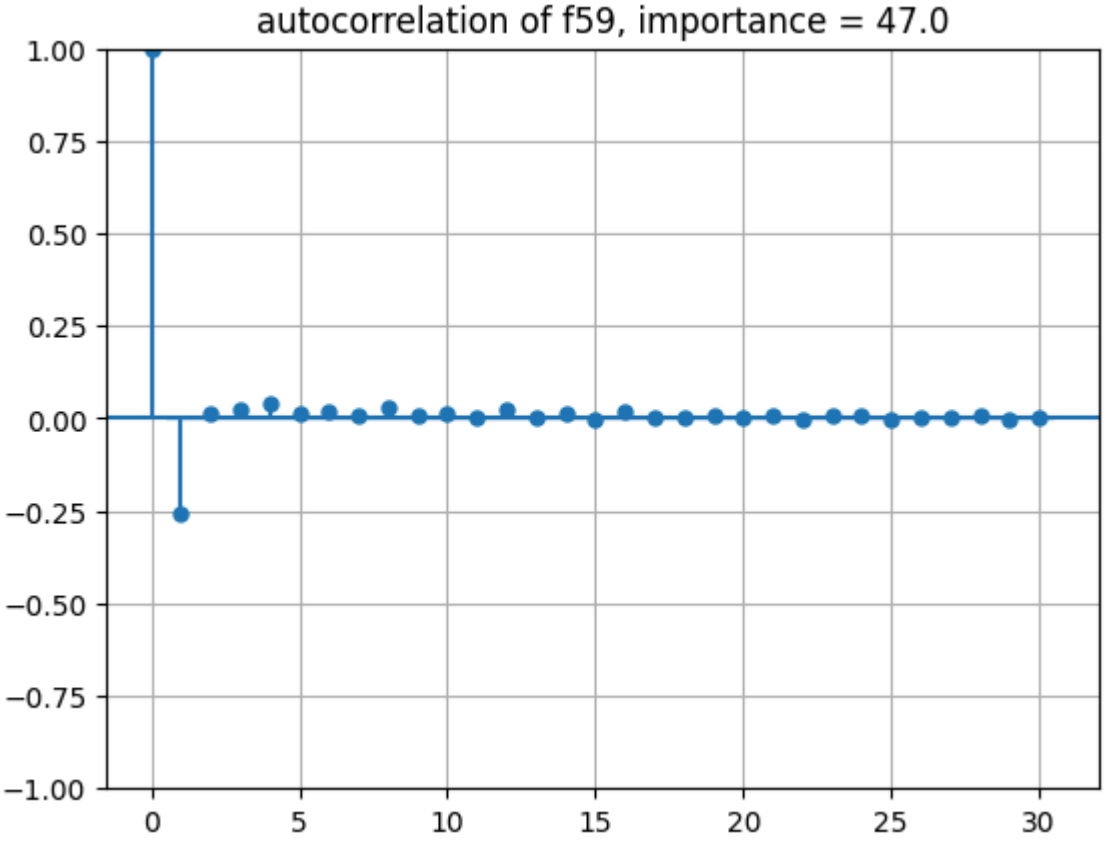
```
imputer = KNNImputer(n_neighbors=5)
data_imputed = imputer.fit_transform(df_new_drop_no_impute)
```

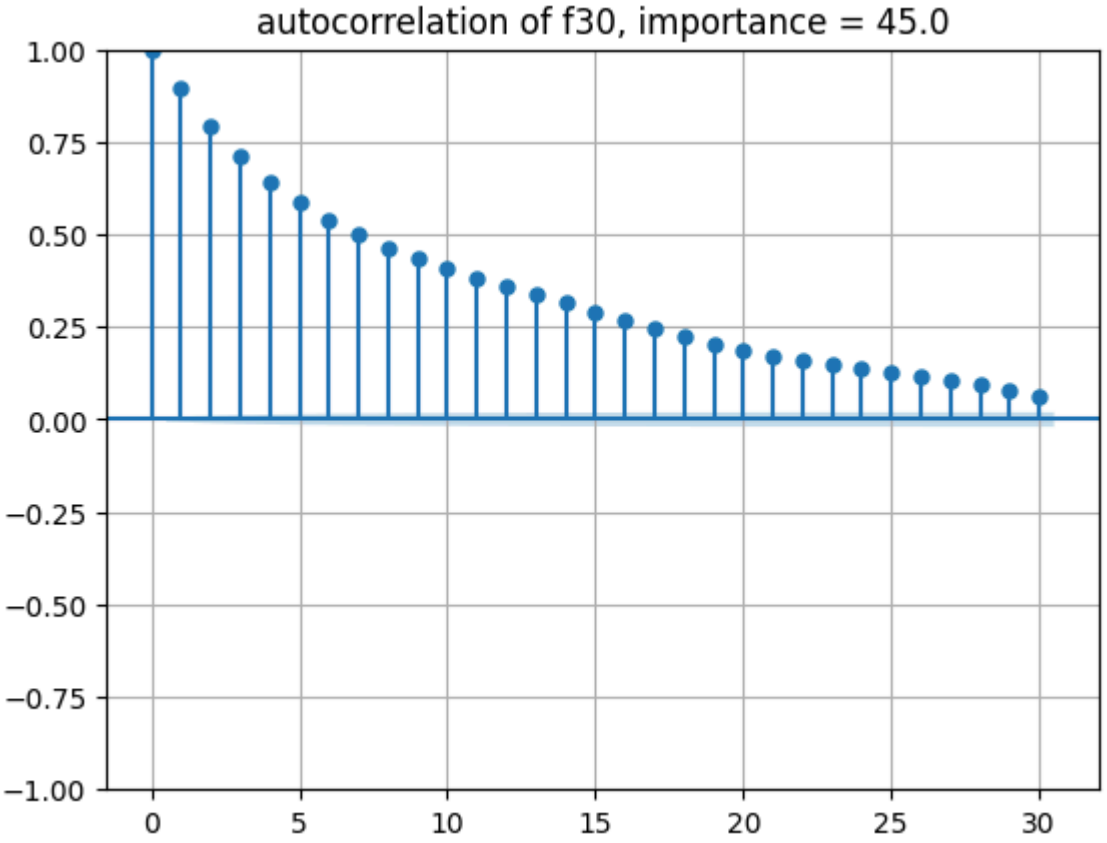
24. XGBoost on the KNN imputed features

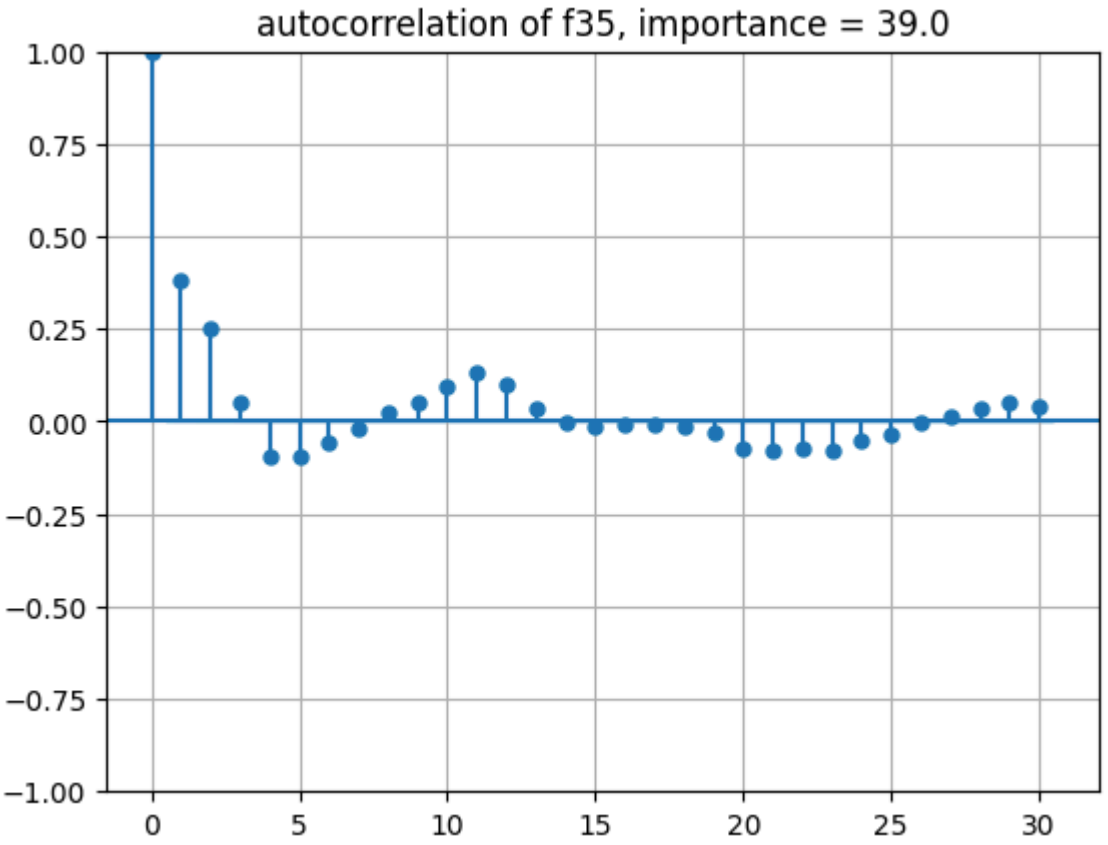
Validation MSE: 0.0041508024377114 54

25. Autocorrelation of some important features




autocorrelation of f59, importance = 47.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


autocorrelation of f30, importance = 45.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30



*** Some features have seasonality! Thus, created lagged features.**

26. Created lagged features

```python
def significant_lags(series, threshold=0.25, max_lags=2):
    # Calculate ACF values
    acf_values = acf(series.dropna(), nlags=max_lags)

    # Identify significant lags
    significant_lags = np.where(np.abs(acf_values) > threshold)[0]

    # Exclude lag 0 (autocorrelation with itself)
    significant_lags = significant_lags[significant_lags > 0]

    return significant_lags.tolist()
```

27. XGBoost on the lagged features

Validation MSE: 0.0041516693314999918

28. Tried Transformers on the target, considering it as a sequence to predict

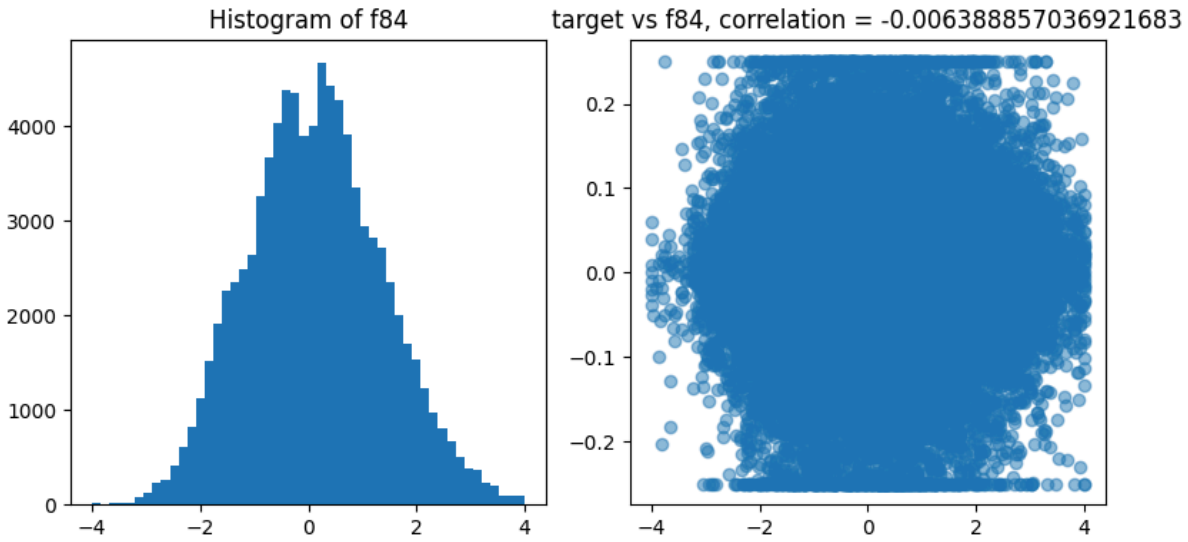
*** For a particular target to predict in validation, gave all the important features along with their respective targets to the encoder side and the row features of which we are to predict the target to the decoder side.**

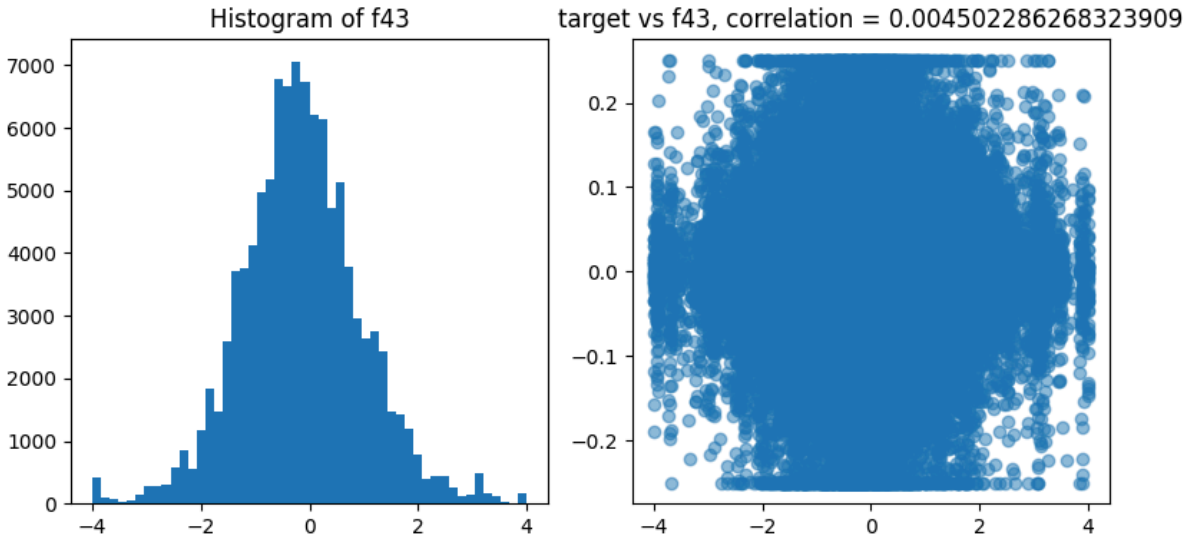
```python
class TransformerModel(nn.Module):
    def __init__(self, enc_input_dim, dcd_input_dim, model_dim, num_heads, num_layers, output_dim):
        super(TransformerModel, self).__init__()
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=model_dim, nhead=num_heads)
        self.encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.decoder_layer = nn.TransformerDecoderLayer(d_model=model_dim, nhead=num_heads)
        self.decoder = nn.TransformerDecoder(self.decoder_layer, num_layers=num_layers)
        self.linear = nn.Linear(model_dim, output_dim)
        self.enc_embedding = nn.Linear(enc_input_dim, model_dim)
        self.dcd_embedding = nn.Linear(dcd_input_dim, model_dim)

    def forward(self, src, tgt):
        src = self.enc_embedding(src.float())
        tgt = self.dcd_embedding(tgt.float())
        memory = self.encoder(src)
        output = self.decoder(tgt, memory)
        output = self.linear(output)
        return output
```

```python
# Hyperparameters
enc_input_dim = 80
dcd_input_dim = 79
model_dim = 128  # Transformer model dimension
num_heads = 4
num_layers = 2
output_dim = 1  # Single target value

model = TransformerModel(enc_input_dim, dcd_input_dim, model_dim, num_heads, num_layers, output_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 10  # Define the number of epochs
for epoch in range(num_epochs):
    print(f"starting epoch {epoch+1}/{num_epochs}")
    model.train()
    for i in range(1, len(train_data)):
        src = train_data[:i, :].unsqueeze(1).float()  # All previous rows (including targets), shape [seq_len, batch_size, f
        tgt = train_data[i:i+1, :-1].unsqueeze(1).float()  # Current row features, shape [1, batch_size, feature_dim]
        true_target = train_data[i:i+1, -1].unsqueeze(1).float()  # Current row target, shape [1, batch_size, 1]

        optimizer.zero_grad()
        output = model(src, tgt).float()
        loss = criterion(output.squeeze(), true_target.squeeze())
        loss.backward()
        optimizer.step()
    print(f"epoch {epoch+1}/{num_epochs} done")
```

*** Required RAM exceeded the quota!**

29. Feature distribution and the scatter plot of target vs important features





*** Very poor correlation with the target, even with the best features! Not suitable for linear regression!**

30. Checked correlation of different transformations and interactions of the features with the target

```
corr_dict = {}

for i in range(len(feats) - 1):
    for j in range(i+1, len(feats)):
        add_feat = df[feats[i]] + df[feats[j]]
        sub_feat = df[feats[i]] - df[feats[j]]
        mul_feat = df[feats[i]] * df[feats[j]]
        div_feat = df[feats[i]] / df[feats[j]].replace(0, 1e-9)

        add_corr = add_feat.corr(df['target'])
        sub_corr = sub_feat.corr(df['target'])
        mul_corr = mul_feat.corr(df['target'])
        div_corr = div_feat.corr(df['target'])

        corr_dict[f'{feats[i]} + {feats[j]}'] = add_corr
        corr_dict[f'{feats[i]} - {feats[j]}'] = sub_corr
        corr_dict[f'{feats[i]} * {feats[j]}'] = mul_corr
        corr_dict[f'{feats[i]} / {feats[j]}'] = div_corr
```

```
for feat in feats:
    log_corr = pd.Series(np.log(df[feat])).corr(df['target'])
    exp_corr = pd.Series(np.exp(df[feat])).corr(df['target'])
    sq_corr = pd.Series((df[feat])**2).corr(df['target'])
    sqrt_corr = pd.Series(np.sqrt(df[feat])).corr(df['target'])

    corr_dict[f'log({feat})'] = log_corr
    corr_dict[f'exp({feat})'] = exp_corr
    corr_dict[f'sq({feat})'] = sq_corr
    corr_dict[f'sqrt({feat})'] = sqrt_corr
```

```
1 low, up = 0.07, 0.3
2 imp_feats = [key for key, value in corr_dict.items() if low <= value]
3 imp_feats
```

```
['f20 + f41', 'f20 - f41', 'f41 / f47']
```

31. **Tried Exponential Moving Average on the target, as if it were a time series data**

```python
def get_EMA(array, span):
    alpha = 2 / (span + 1)
    ema = array[0]

    for i in range(1, len(array)):
        ema = alpha * array[i] + (1 - alpha) * ema

    return ema
```

```python
span = 10

for i in range(len(X)):
    if np.isnan(X[i]):
        past = X[:i]
        X[i] = get_EMA(past, span)
```

Validation MSE: 0.004764633349483802

32. **Tried LSTM**

```python
# Function to create sequences
def create_sequences(features, target, sequence_length):
    X, y = [], []
    for i in range(len(features) - sequence_length):
        X.append(features[i:i+sequence_length])
        y.append(target[i+sequence_length])
    return np.array(X), np.array(y)
```

```python
# Build the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(sequence_length, features.shape[1]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')
model.summary()
```

Validation MSE: 0.0052 *(Even worse than linear regression!)

33. LSTM on lagged features

***Even worse: Validation MSE: 0.0060**

34. Tried with neural network

***Better than LSTM but still worse than linear regression**
Validation MSE: 0.0048

35. Applied Recursive Feature Elimination(RFE) and Recursive Feature Elimination with Cross Validation(RFECV)

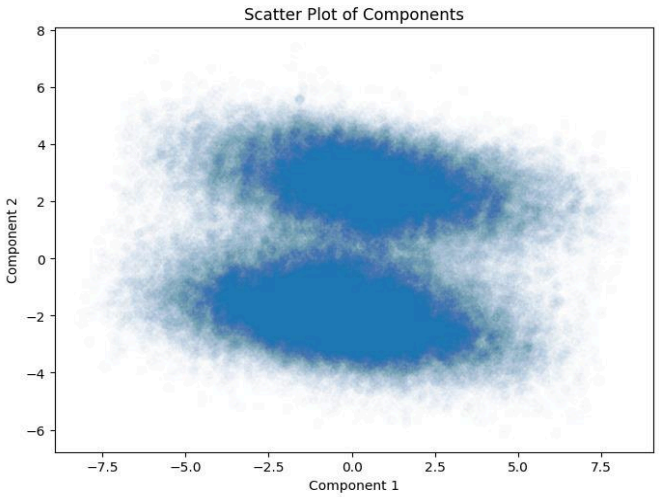
```
# Initialize the XGBRegressor
model = xgb.XGBRegressor(objective='reg:squarederror', colsample_bytree=0.3, learning_rate=0.1,
                         max_depth=5, alpha=10, n_estimators=300)

# Initialize RFE with the XGBRegressor
rfe = RFE(estimator=model, n_features_to_select=10, step=1)
```

```
# Initialize the XGBRegressor
model = xgb.XGBRegressor(objective='reg:squarederror', colsample_bytree=0.3, learning_rate=0.1,
                         max_depth=5, alpha=10, n_estimators=300)

# Initialize RFECV with the XGBRegressor
rfecv = RFECV(estimator=model, step=1, cv=5, scoring=make_scorer(mean_squared_error, greater_is_better=False))
```

```
Optimal number of features: 83
Selected Features: Index(['f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9',
```

Validation MSE: 0.0041443377665700 96

36. Applied PCA on the features and explored if there is any possibility of clusters



***There are two clusters!**

37. Added a cluster column using KMeans clustering, n = 2, and trained XGBoost separately on them

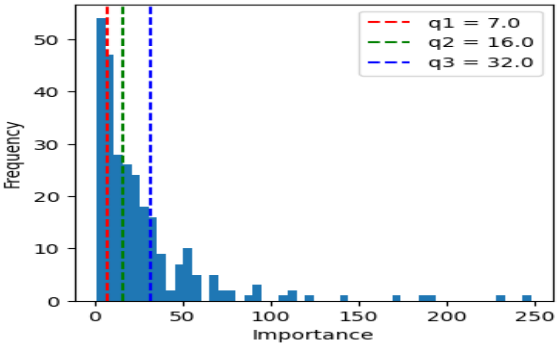
```
df_0
----------
Optimal number of features: 82
Selected Features: Index(['f1', 'f2', 'f3', 'f4', 'f5', 'f7', 'f8', 'f9', 'f10',
mse: 0.0042775353908237656
--------------------------------------------------------------------------------
df_1
----------
Optimal number of features: 80
Selected Features: Index(['f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', '
mse: 0.00426195764651821
```

Validation MSE on cluster 0: 0.0042775353908237656
Validation MSE on cluster 1: 0.00426195764651821

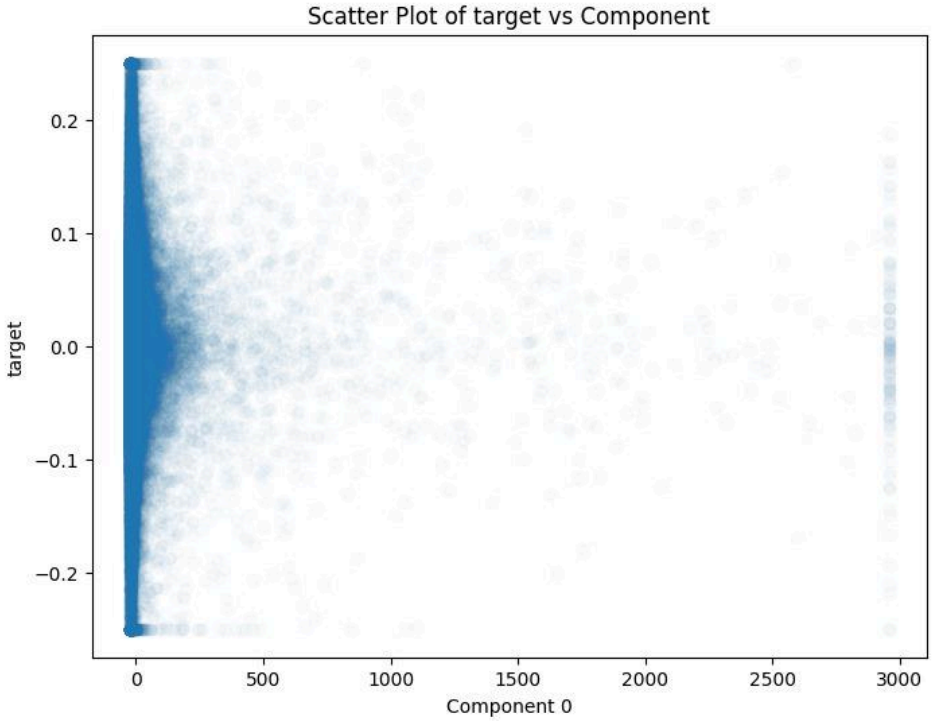
***** Trying with high n_estimators, and low tree depth *****

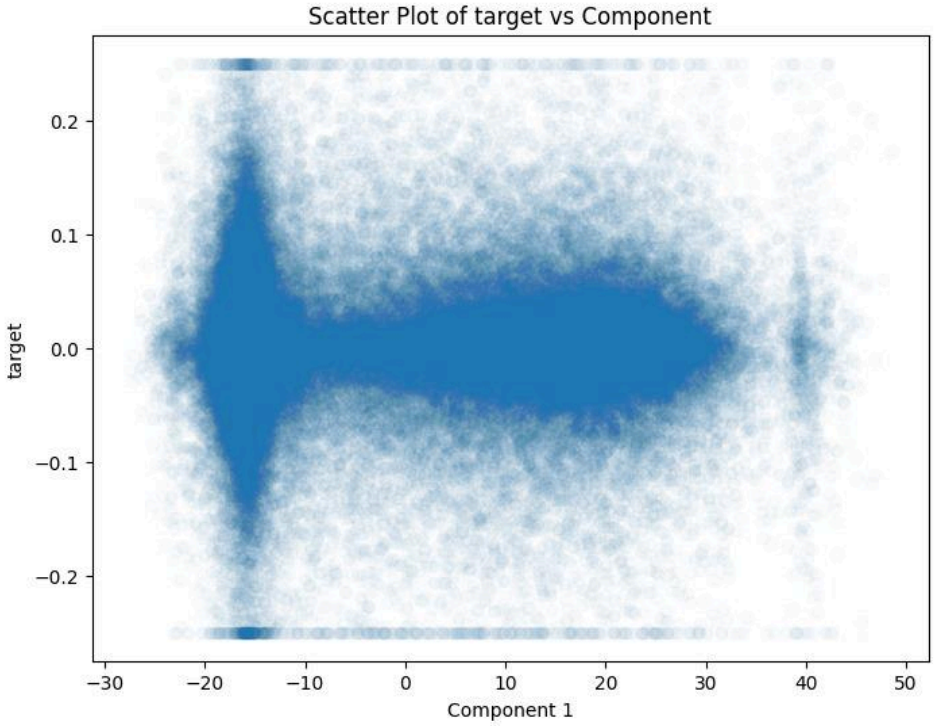
38. Log, exponential, square, and square root transformation of all features, then feature importance

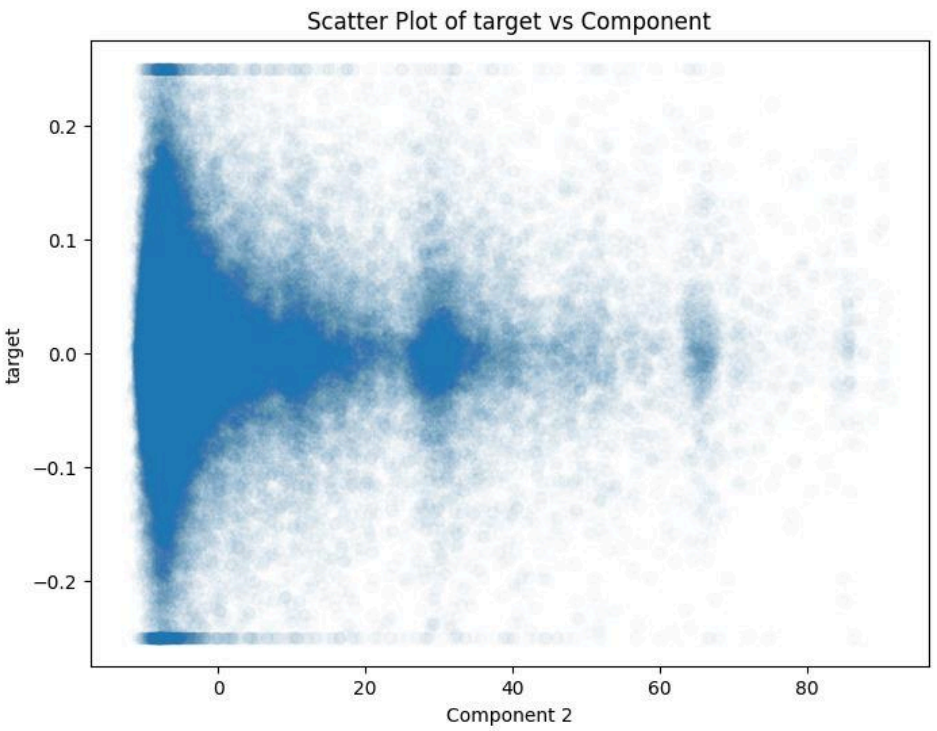
	Features	Importance
146	sq(f39)	248.00
114	sq(f24)	232.00
224	sq(f70)	190.00
164	sq(f50)	186.00
133	sq(f31)	170.00

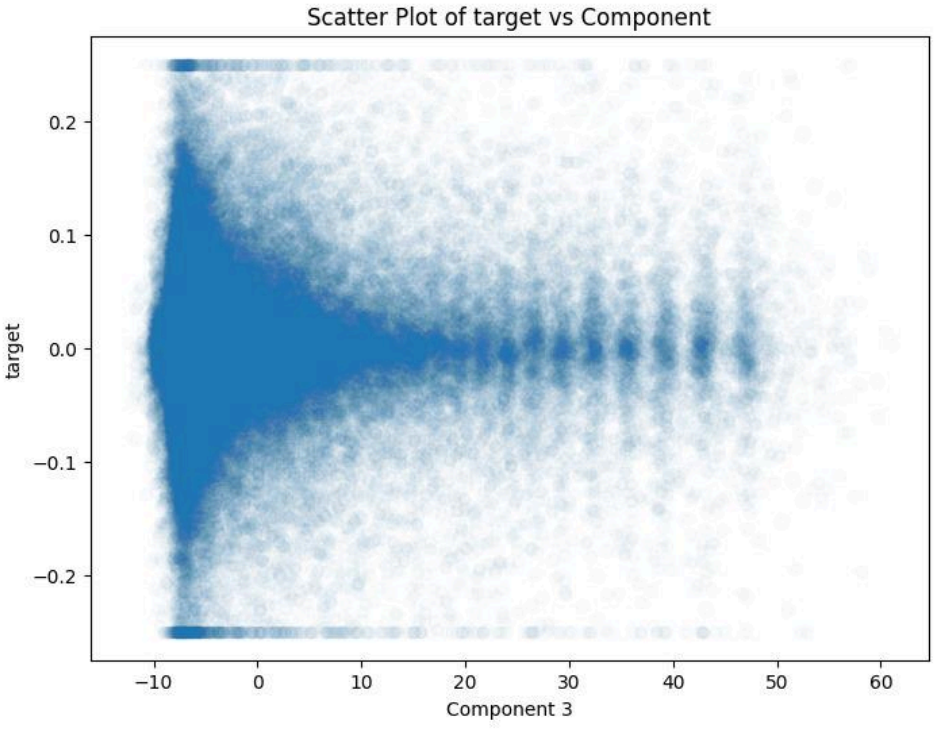


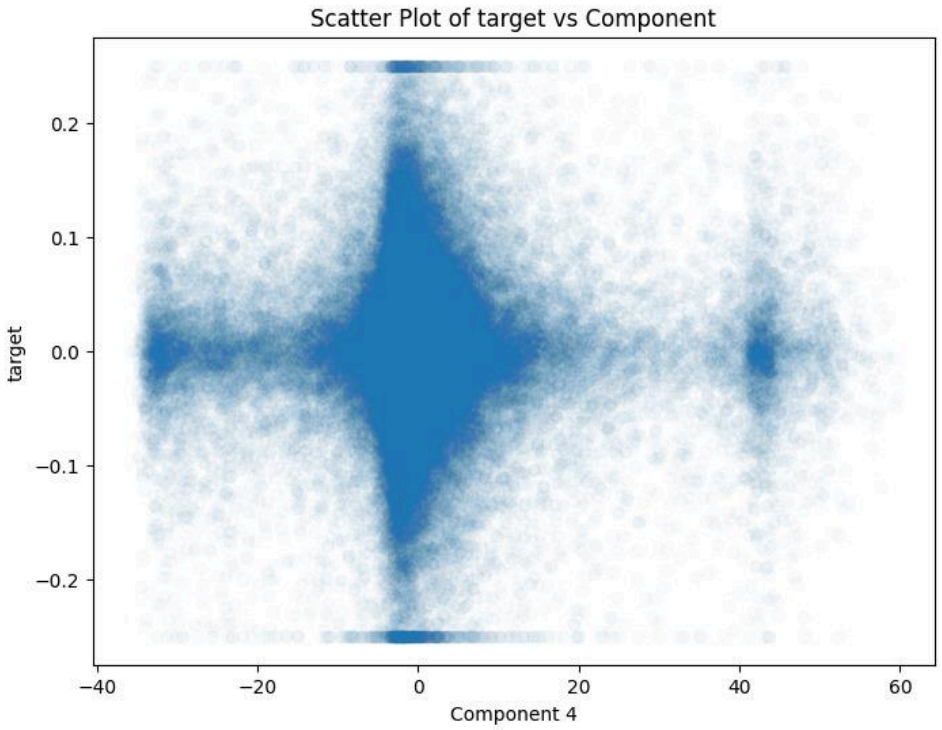
39. Took top 80 transformed features, applied PCA with n = 5






Scatter Plot of target vs Component
target
Component 2
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80


Scatter Plot of target vs Component
target
Component 3
0.2
0.1
0.0
−0.1
−0.2
−10
0
10
20
30
40
50
60



40. XGBoost on these important transformed features

Validation MSE: 0.004167197886135464

41. Tried GradientBoost with 2000 estimators, tree depth = 1

Validation MSE: 0.004162617513840202

42. Tried log(target + 1)

XGBoost Validation MSE: 0.004261299741872281
GradientBoost Validation MSE: 0.004268580004159017

43. Tried ensemble regression

Base models: XGBoost, LightGBM, GradientBoost
Final Model: XGBoost

```
# Stacking Regressor
stacking_regressor = StackingRegressor(
    estimators=[
        ('xgb', xgboost_model),
        ('lgb', lightgbm_model),
        ('gbr', gradientboost_model)
    ],
    final_estimator=final_model,
    passthrough=True  # use original features along with base model predictions
)
```

Validation MSE: 0.0041603194530779 24

44. Tried with creating a new feature based on the exponential moving average on the target

Validation MSE: 0.0025178290737984 68
***Though the Validation MSE is very good, performed the worst after submission, thus a high overfitting!**

Conclusion:

This report comprehensively explored how to predict stock returns during earnings announcement periods. By leveraging advanced feature engineering and various sophisticated modeling techniques, we aimed to achieve precise and reliable predictions. Our journey covered an extensive range of methods, from initial data preprocessing and imputation strategies to advanced machine learning models and time series analysis. Throughout the process, we maintained a focus on avoiding overfitting and ensuring the robustness of our models.

The challenge underscored the importance of meticulous data analysis and innovative feature creation in enhancing model performance. By systematically evaluating different approaches, we identified key insights and best practices that contributed to more accurate predictions. This rigorous exploration demonstrated the complexity of forecasting stock returns and highlighted the potential of quantitative finance techniques in tackling such challenges.

Overall, the research journey was invaluable, providing a deeper understanding of the interplay between earnings announcements and stock returns. The findings and methodologies documented in this report testify to the dedication and analytical rigor applied in this prestigious competition. Moving forward, these insights can guide future financial modeling and prediction efforts, paving the way for continued innovation in Quantitative Finance.