What is the Cost Function of Logistic Regression? Explain in Detail.

Logistic Regression models what the probability of an input belongs to a class. It's a statistical methodology for binary classification tasks. To tackle the question above, we'll answer in two parts. The first section will be a mathematical explanation of the cost function, while the second is a written explanation. Throughout this question, we use resource: Supervised Learning II by Kannan Singaravelu. I'll be highlighting the slides used when going through each mathematical step.

Mathematical Explanation

Hypothesis for Logistic Regression: Slides 8-10

For logistical regression, we can link any real number into a (0,1) interval. This is known as the hypothesis and is defined using sigmoid (or logistic) function:

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where:

- $h_\theta(x)$ is the predicted probability.
- θ is the parameter vector.
- $\sigma(z)$ is the sigmoid function.

Probability of y Given x: Slide 10

For a single training example (x, y), the conditional probabilities are:

$$P(y \mid x) = \begin{cases} \hat{y} = h_\theta(x) & \text{if } y = 1 \\ 1 - \hat{y} = 1 - h_\theta(x) & \text{if } y = 0 \end{cases}$$

where $\hat{y}$ is the predicted probability.

Bernoulli Random Variable: Slide 25

As we are using the labels 0 and 1, then Y can be modeled as a Bernoulli random variable $Y \sim \text{Ber}(p)$, where $p = \sigma(\theta^T x)$.

The probability $P(Y = y \mid X = x)$ can then be written as the below:

$$P(Y = y \mid X = x) = \sigma(\theta^T x)^y [1 - \sigma(\theta^T x)]^{(1-y)}$$

Likelihood and Log-Likelihood: Slide 24

When looking at the probability of observing the data set, we can see that it's the product of the probabilities of an individual observation with the below formulas:

$$L(\theta) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)}; \theta)$$

Taking the natural logarithm gives us the log-likelihood which we can see below:

$$\log L(\theta) = \sum_{i=1}^{m} \log P(y^{(i)} \mid x^{(i)}; \theta)$$

Substituting the equations, we get:

$$\log L(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \log(\sigma(\theta^T x^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(\theta^T x^{(i)})) \right]$$

Loss Function: Slide 19 and 26

The negative log-likelihood is used as the loss function, which we need to minimize as our goal:

$$L(\hat{y}, y) = -\log P(y \mid x) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \right]$$

Cost Function for Logistic Regression: Slide 26

The cost function $J(\theta)$ for the entire training set is the average of the loss function over all training examples:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

where m is the number of training examples, $y^{(i)}$ is the actual label, and $\hat{y}^{(i)}$ is the predicted probability.

Written Explanation

In logistic regression, for our hypothesis function defined above, it's mapping real numbers to a probability set (0,1). The means by which we're mapping is by using the sigmoid function. We care about this outcome as the function is giving us the predicted probability that our target is 1 from our input feature. Effectively, we're altering the output into a probability. This is because the sigmoid function which is representing our hypothesis is applying to our linear combination of inputs.

For this probability which is for our target variable, we define it in two ways. The first way is if our target is 1, then our probability is the predicted value. On the other hand, if our target is 0, it is one minus our predicted value. For Bernoulli random variables, our method described for the target variable aligns with the modeling. This is because the probability of our target being 1 is given by the sigmoid function.

When using the logistic regression model, the probability of observing the training data is the product of the probabilities mapped to each training example. In order for us to simplify the product into a sum, we take natural log of the likelihood which gives us the log likelihood. This is especially important as it'll make it easier to maximize computationally. We then take the log likelihood and substitute it with the sigmoid function to express it in the terms of the models parameters and the input features.

Finally, deriving the negative log likelihood gives us our cost (or loss) function. Our goal during the training is to minimize this as the loss function can be described as the difference between our probabilities which are predicted and the actual target value. In other words, it's the average of the loss over the training examples. The importance of our cost function is it'll act as our object when optimizing using gradient descent. By definition, our cost function is convex which when using optimization techniques, we can find the global minimum. The results of this is when optimizing, we are able to get the best parameter values for our given model.

Optimization

We use logistic regression in order to find the parameter θ so that we can minimize our cost function $J(\theta)$. This is done using gradient descent which is as an iterative optimization that will change our parameters in the direction of our negative gradient of the cost function:

$$\theta := \theta - \alpha \nabla_\theta J(\theta)$$

where α is the learning rate, and $\nabla_\theta J(\theta)$ is the gradient of the cost function.

Resources used was Mathematics Toolbox for Machine Learning by Panos Parpas, slides 21, 22, 45, 54

Conclusion

Our cost function (which can also referred to as the log loss or more typically, binary cross entropy loss) can be defined as the difference between the probabilities which are predicted and our actual labels. This would ultimately measure the loss. When we minimize our cost function, we can make our trained logistic regression model to make more accurate predictions. The convexity of the cost function ensures that when using optimizations, such as gradient descent, we can efficiently find the optimal parameters.

Resources used:

Mathematics Toolbox for Machine Learning by Panos Parpas

Supervised Learning I by Kannan Singaravelu

Supervised Learning II by Kannan Singaravelu

What are voting classifiers in ensemble learning?

Throughout this question, we use resource: Extreme Gradient Boosting by Kannan Singaravelu Python Lab, lecture Supervised Learning 1 by Kannan Singaravelu, and lecture Decision Trees and Ensemble by Dr. Panos Parpas. Using slides from Decision Trees and Ensemble, I'll be annotating the slides I used information from through this written work.

Before we dive into Voting Classifiers, lets first define Ensemble Learning. This is a powerful machine learning technique which takes multiple base models (or learners) and uses their predictions in order to improve the overall model performance. As we are using a combination of models instead of a single one, we're able to use the strengths of others to help the gap of some of the other weaknesses. This in turn helps lower issues such as high variance, bias and overfitting.

Some ensemble techniques include bagging (which is also known as bootstrap aggregating) and boosting.

For Bagging, we use random sampling with replacement (or bootstrapping) which trains our models on different parts of our data. The goal of doing this is to reduce the overall variance. For regression, we take an average of our predictions and for classifications, we have a voting methodology on our output. In this technique, a common example is using Random Forest which will take these combinations of our predictions in order to help control fitting and improve our overall accuracy (Slide 24).

Within Random Forests (which is a specific type of bagging), we use decision trees as our model. Each tree will take a different segment of our data to train on and in our in our final result, we aggregate all the tree's predictions to get our final output. By doing this, we get a smoothening of our predictions which would help reduce overfitting. (Slide 29)

The other typical technique to Ensemble learning is boosting. This method focuses on reducing our models bias. Here, each of our models tries to correct the errors of the previous one by being trained sequentially. Essentially, we're combining weak models to form an even stronger model. Some of the more common boosting algorithms are AdaBoost and Gradient Boosting (Slide 32).

AdaBoost is an iterative process which works by taking incorrectly classified instances and adjusting their weights such that the next model can work on a more difficult case. Gradient Boosting on the other hand is a process which involves optimizing a loss function where sequentially, each model works to correct the errors made by the previous model. (Slides 37 and 44).

For Ensemble learning, voting classifiers are a specific type which combines multiple modules (or classifiers) in order to have our model increase it's performance. The reason this is powerful is because we're aggregating the predictions of many base models such that we have a more accurate final prediction. This combination provides an accuracy which is better than using an individual model. (Slide 29)

When talking about voting classifiers, we can break these into two main types:

The first type is Hard Voting. In this method, our individual base models each makes their prediction, where our final prediction is the one which gets the majority of the votes. Basically, the distinct value which is predicted by the most models is then picked as our final solution. This becomes a simple method as our models are given an equal vote.

The second type is Soft Voting. Unlike Hard Voting where we need a majority vote, every models predicted probability of our distinct value is considered. We then take the average of our probabilities, and the final prediction is selected based off of the highest average of our probabilities. As we are accounting for the confidence of the models predictions, we generally see better performance than Hard Voting.

When individual models make errors and are diverse, we see the impact of how voting classifiers become effective. Given we're using a combination of models, we're able to use the strengths of each model which can be used to compensate for the weakness

of others. From our diversification, our predictions becoming more reliable and accurate. (Slide 24 and 26)

Resources used:

Decision Trees and Ensemble by Dr. Panos Parpas

Supervised Learning 1 by Kannan Singaravelu

Extreme Gradient Boosting by Kannan Singaravelu, Python Lab

For Hard and Soft Voting - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

Gradient boosting, random forests, bagging, voting - https://scikit-learn.org/stable/modules/ensemble.html

Follow the 7-steps to model building

7 steps to to model building *Resource used: Introduction to Machine Learning using Scikit-learn by Kannan Singaravelu*

Steps	Workflow	Remarks
Step 1	Ideation	Predict next trading day index price from the given dataset
Step 2	Data Collection	Load the dataset from Yahoo Finance
Step 3	Exploratory Data Analysis	Study summary statistics
Step 4	Cleaning Dataset	Ensure no null values are present
Step 5	Transformation	Perform feature scaling based on EDA
Step 6	Modeling	Building and training SVM
Step 7	Metrics	Validating the model performance

In [ ]:
```
import yfinance as yf
import pandas as pd
import numpy as np
from pylab import plt
```

```
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=FutureWarning)

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,MinMaxScaler

from sklearn.feature_selection import SelectKBest,f_classif,RFECV,chi2,SelectFromModel
import shap
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.linear_model import Lasso, Ridge, ElasticNet, LogisticRegression, LinearRegression

from sklearn.metrics import (precision_recall_curve,
                             classification_report,
                             roc_curve,
                             RocCurveDisplay,
                             ConfusionMatrixDisplay,
                             mean_squared_error,
                             accuracy_score,
                             f1_score,
                             r2_score,
                             auc
                             )

from sklearn.svm import SVC
```

First, we download QQQ data from Yahoo Finance for dates 2015-01-01 to 2024-05-17 in order to get a range of values each day

In [ ]:
```
QQQ_data = yf.download("QQQ", start="2015-01-01", end="2024-05-17", period = '1d')
```

```
[**********100%%**********]  1 of 1 completed
```

In [ ]:
```
QQQ_data.head()
```

Out[ ]:

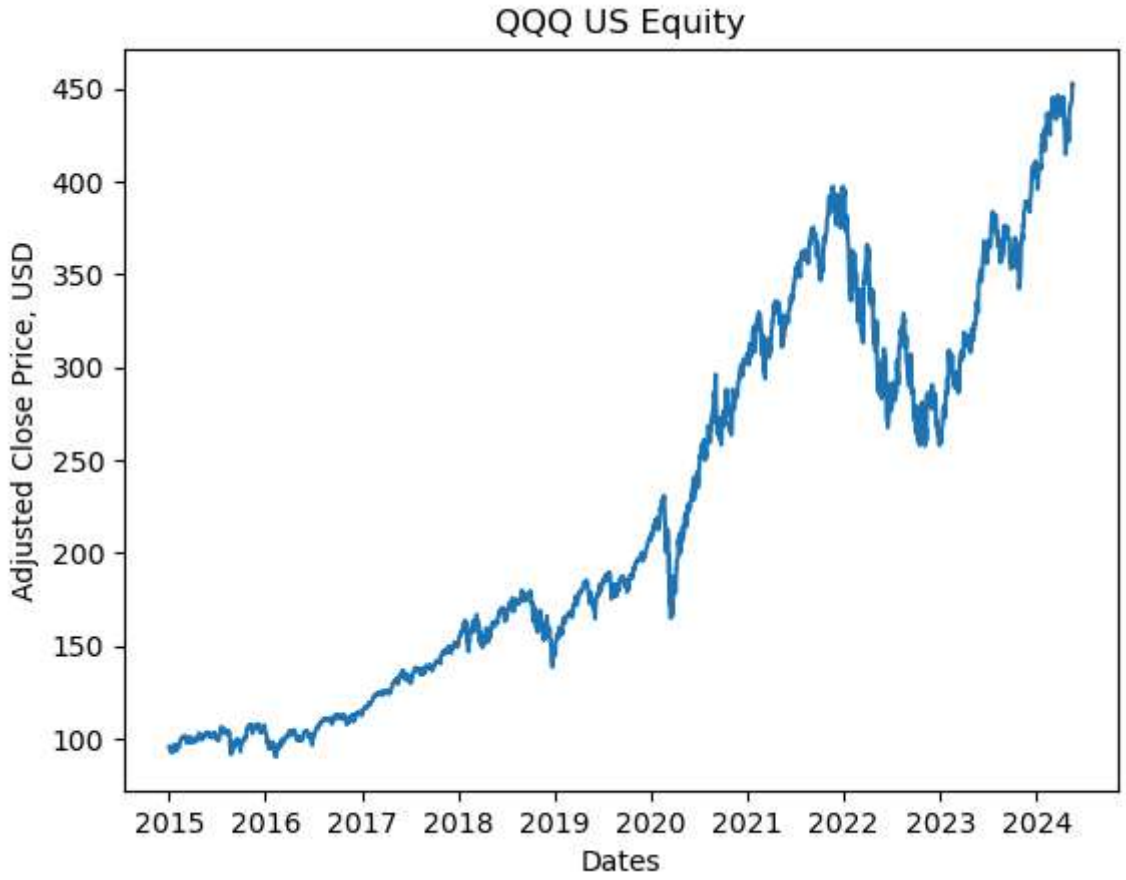
Date	Open	High	Low	Close	Adj Close	Volume
2015-01-02	103.760002	104.199997	102.440002	102.940002	95.704704	31314600
2015-01-05	102.489998	102.610001	101.139999	101.430000	94.300842	36521300
2015-01-06	101.580002	101.750000	99.620003	100.070000	93.036453	66205500
2015-01-07	100.730003	101.599998	100.489998	101.360001	94.235764	37577400
2015-01-08	102.220001	103.500000	102.110001	103.300003	96.039429	40212600

To ensure the data is clean without any null values, we do a summation check to verify there are no missing values. In addition, it's important to understand how our raw data set looks like. For this, we run a graph on our main metric, Adj Close.

In [ ]:
```
# Check for missing values
QQQ_data.isnull().sum()
```

Out[ ]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [ ]:
```
plt.plot(QQQ_data['Adj Close'])

plt.title('QQQ US Equity')
plt.ylabel('Adjusted Close Price, USD')
plt.xlabel('Dates');
```



Creating Labels & Features

We can use data transformation on our original set. In doing so, we create features which can help machine learning algorithms perform better. The process involves taking the raw data from QQQ on Yahoo finance, and transforming it into more variables which can help our predictive power. For these transformations, I used the ones described in question 3 which can be found below:

Feature	Formula	Description
O-C, H-L	Open - Close, High - Low	intraday price range

Feature	Formula	Description
Sign	$\text{sign}\left[r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)\right]$	sign of return or momentum
Past Returns	$r_{t-1}, r_{t-2}, \dots$	lagged returns
Momentum	$P_t - P_{t-k}$	price change over k period
Moving Average	$\text{SMA}_i = \frac{1}{n}\sum_{i=0}^{n-1} P_{t-i}$	simple moving average
Exponential MA	$\text{EMA}_i = \text{EMA}_{t-1} + \alpha[P_t - \text{EMA}_{t-1}]$	recursive, $\alpha = \frac{2}{(N_{obs}+1)}$

For Past Return, Momentum and moving average, I have these ranging from 1 to 11 days to predict short term trends. Some of these features may be highly correlated which we will check later in this paper.

Now that we have our features created for predictions, we need an object to predict off of. We will use the label y which will be our predictive object.

$$y_t = \begin{cases} 1, & \text{if } p_{t+1} > 0.9975 \times p_t \\ 0, & \text{otherwise} \end{cases}$$

In [ ]:
```
# Features
# Resources used: Logistic Regression & Linear Regression Python Lab by Kannan Singaravelu
#Percentage return
QQQ_data['r'] = QQQ_data['Adj Close'].pct_change()

# O-C
QQQ_data['OC'] = QQQ_data.Open - QQQ_data.Close
# H-L
QQQ_data['HL'] = QQQ_data.High - QQQ_data.Low

# Sign
```

```
QQQ_data['sign'] = np.sign(np.log(QQQ_data['Adj Close']).diff())

n = [i for i in range(1,11)]
for i in n:

    # Past Return
    QQQ_data['PastReturn' + str(i)] = QQQ_data.r.shift(i)

    # Momentum
    QQQ_data['Momentum' + str(i)] = QQQ_data['Adj Close'].diff(i)

    # Moving Average
    QQQ_data['SMA' + str(i)] = QQQ_data['Adj Close'].rolling(window=i).mean()

# Exponential MA
alpha = 2 / (len(QQQ_data + 1))
QQQ_data['EMA'] = QQQ_data['Adj Close'][0]
for i in range(1, len(QQQ_data)):
    QQQ_data.iloc[i, len(QQQ_data.columns)-1] = (1 - alpha) * QQQ_data.iloc[i - 1, len(QQQ_data.columns)-1] + alpha *

QQQ_data['y'] = np.where(QQQ_data['Adj Close'].shift(-1) > 0.9975 * QQQ_data['Adj Close'], 1, 0)

# Show the New dataset
QQQ_data = QQQ_data.dropna()
```

Whats interesting using np.sign is it'll return an element set of returns which is -1 if x < 0, 0 if x==0, 1 if x > 0. Saying this, as all values may be potentially important or additive for underlying predictions, we will not remove any values.

In [ ]:
```
np.unique(np.array(QQQ_data['sign']))
```

Out[ ]:
```
array([-1.,  0.,  1.])
```

Visual Inspection

Now, lets us look at our new dataset which has our features and categorical variables that we are using.

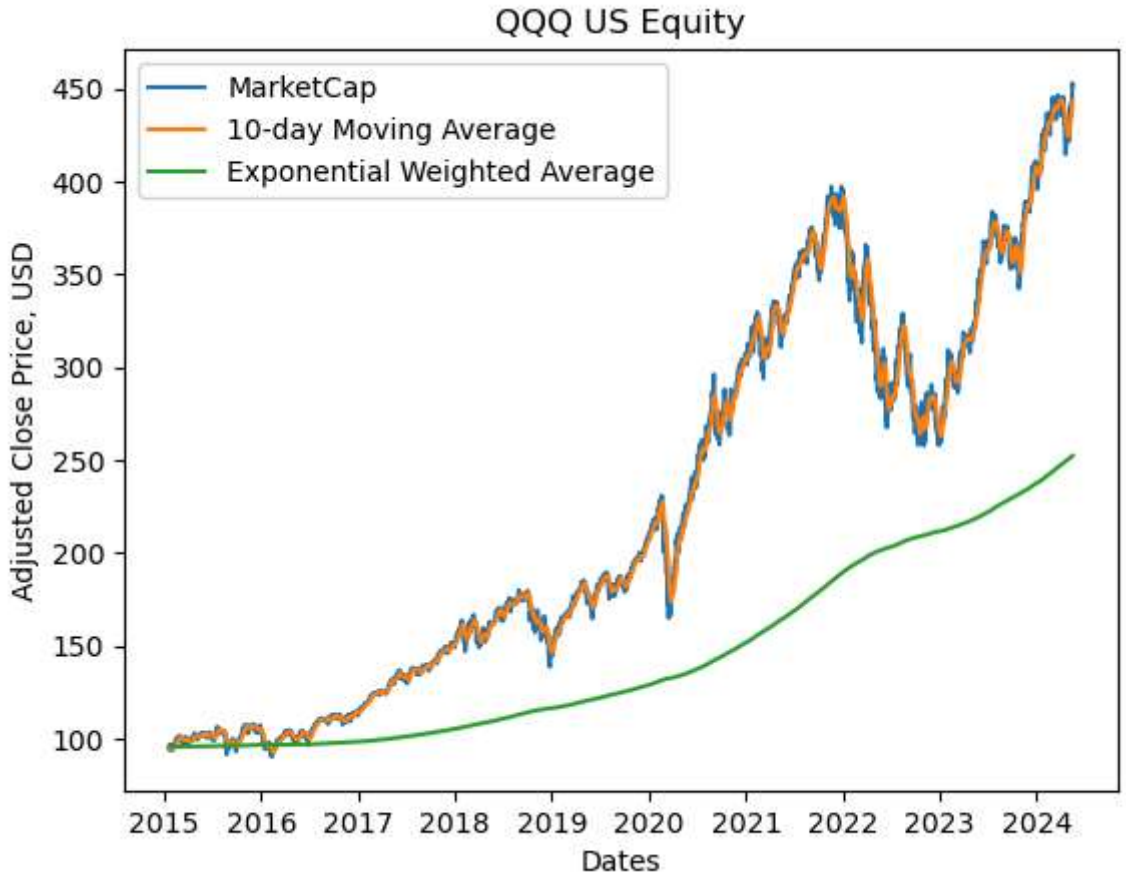
In [ ]:
```
QQQ_data.head()
```

Out[ ]:

Date	Open	High	Low	Close	Adj Close	Volume	r	OC	HL	sign	...	Momentu
2015-01-20	101.430000	101.879997	100.290001	101.620003	94.477493	30964100	0.007935	-0.190002	1.589996	1.0	...	0.241
2015-01-21	101.320000	102.620003	100.959999	102.139999	94.960945	40128800	0.005117	-0.820000	1.660004	1.0	...	-1.078
2015-01-22	102.540001	104.139999	101.639999	104.029999	96.718109	40887200	0.018504	-1.489998	2.500000	1.0	...	1.310
2015-01-23	104.019997	104.580002	103.720001	104.260002	96.931938	34783000	0.002211	-0.240005	0.860001	1.0	...	2.519
2015-01-26	104.139999	104.330002	103.610001	104.139999	96.820358	19960900	-0.001151	0.000000	0.720001	-1.0	...	2.435

5 rows × 42 columns

In [ ]:
```
plt.plot(QQQ_data['Adj Close'], label='MarketCap')
plt.plot(QQQ_data['SMA10'], label = '10-day Moving Average')
plt.plot(QQQ_data['EMA'], label='Exponential Weighted Average')
plt.legend()

plt.title('QQQ US Equity')
plt.ylabel('Adjusted Close Price, USD')
plt.xlabel('Dates');
```

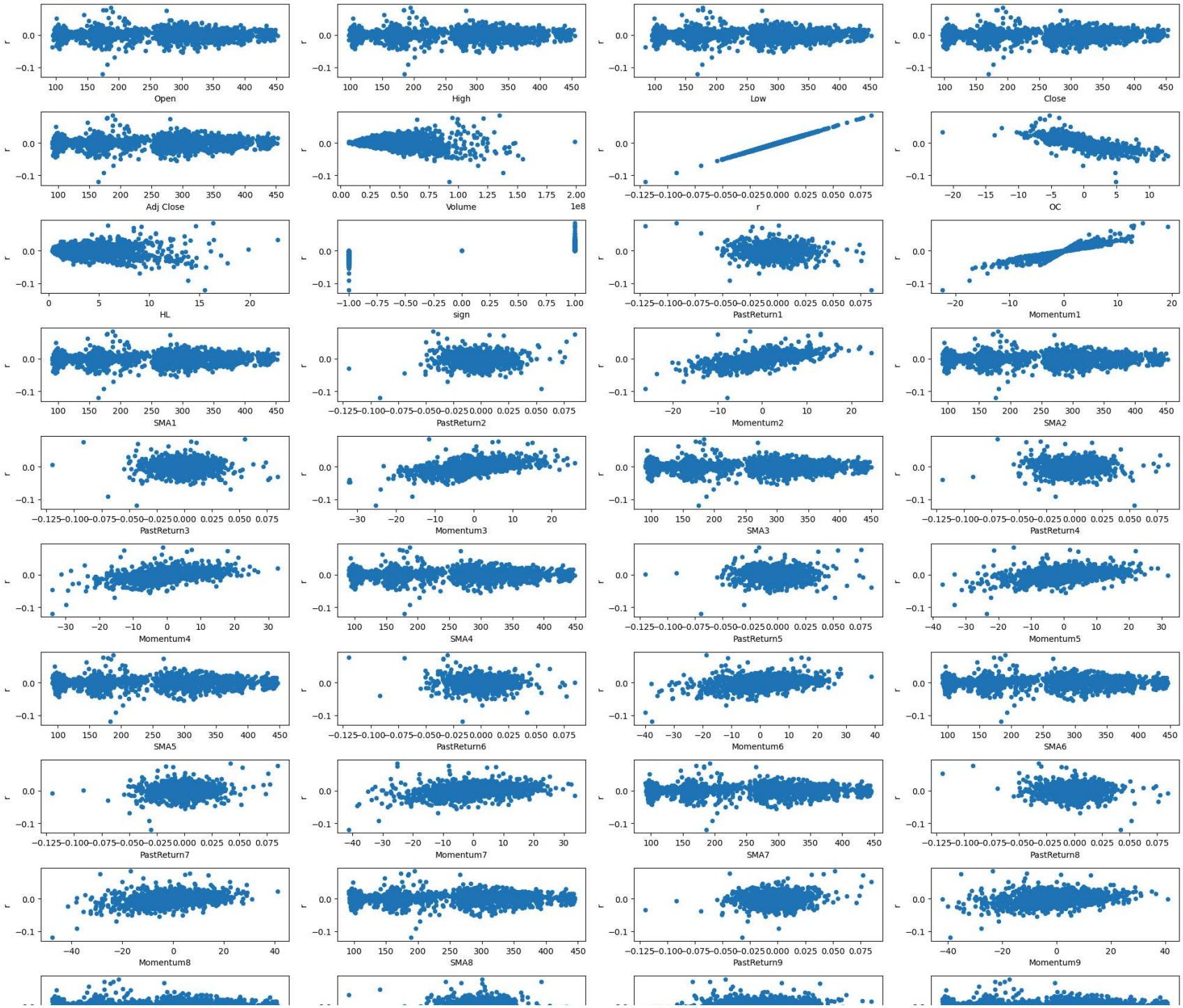


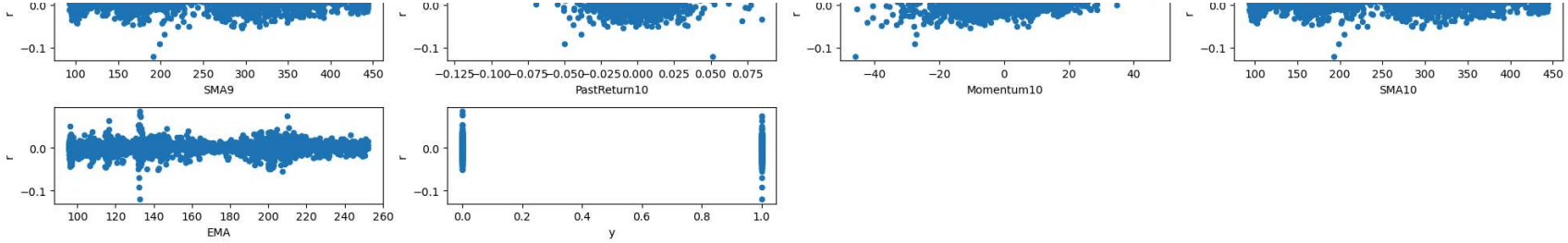
Plotting 'Adj Close', 'SMA10' and 'EMA', we can see how close these values are in terms of upward trend movements. Interestingly, the 10 day moving average aligns with our adjusted close in close approximation with one another. Further, the exponential weighted average follows the underlying trend which is a positive slopping distribution over time. During the Covid crash of 2021, we also see EMA drastically flattening which is accounting for the drop in market prices. While I could have added additional metrics to our graph, I found that many did not add substance which is why they are excluded here.

Correlation of Metrics

Our features may include ones which are highly correlated with one another. Lets observe to see if we can see underlying scatter plot or numerical trends which would indicate this.

In [ ]:
```
num_features = QQQ_data.shape[1]

num_cols = 4
num_rows = (num_features // num_cols) + (1 if num_features % num_cols != 0 else 0)

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, 20))

for idx, feature in enumerate(QQQ_data.columns):
    row = idx // num_cols
    col = idx % num_cols
    QQQ_data.plot(feature, "r", subplots=True, kind="scatter", ax=axes[row, col])

for i in range(num_features, num_rows * num_cols):
    fig.delaxes(axes.flatten()[i])

plt.tight_layout()
plt.show()
```


r
Open
High
Low
Close
Adj Close
Volume
1e8
r
OC
HL
sign
PastReturn1
Momentum1
SMA1
PastReturn2
Momentum2
SMA2
PastReturn3
Momentum3
SMA3
PastReturn4
Momentum4
SMA4
PastReturn5
Momentum5
SMA5
PastReturn6
Momentum6
SMA6
PastReturn7
Momentum7
SMA7
PastReturn8
Momentum8
SMA8
PastReturn9
Momentum9



In [ ]:
```
QQQ_data.iloc[:,2:].corr()
```

Out[ ]:

	Low	Close	Adj Close	Volume	r	OC	HL	sign	PastReturn1	Momentum1
Low	1.000000	0.999826	0.999739	0.337065	0.008691	-0.006644	0.579500	-0.002216	0.018462	0.027584
Close	0.999826	1.000000	0.999909	0.343897	0.018482	-0.019662	0.587512	0.004474	0.015331	0.038203
Adj Close	0.999739	0.999909	1.000000	0.345217	0.018119	-0.019604	0.587808	0.003866	0.015062	0.037933
Volume	0.337065	0.343897	0.345217	1.000000	-0.190050	0.124393	0.800751	-0.186448	-0.191762	-0.177345
r	0.008691	0.018482	0.018119	-0.190050	1.000000	-0.738166	-0.126607	0.689671	-0.112188	0.934596
OC	-0.006644	-0.019662	-0.019604	0.124393	-0.738166	1.000000	0.102090	-0.549939	0.063445	-0.809153
HL	0.579500	0.587512	0.587808	0.800751	-0.126607	0.102090	1.000000	-0.125444	-0.129310	-0.125267
sign	-0.002216	0.004474	0.003866	-0.186448	0.689671	-0.549939	-0.125444	1.000000	-0.028693	0.650690
PastReturn1	0.018462	0.015331	0.015062	-0.191762	-0.112188	0.063445	-0.129310	-0.028693	1.000000	-0.085144
Momentum1	0.027584	0.038203	0.037933	-0.177345	0.934596	-0.809153	-0.125267	0.650690	-0.085144	1.000000
SMA1	0.999739	0.999909	1.000000	0.345217	0.018119	-0.019604	0.587808	0.003866	0.015062	0.037933
PastReturn2	0.017148	0.015425	0.015107	-0.145847	0.026910	-0.006653	-0.113526	0.002596	-0.112420	0.015883
Momentum2	0.048686	0.054359	0.054017	-0.259702	0.620300	-0.553368	-0.181181	0.462632	0.621537	0.683310
SMA2	0.999773	0.999767	0.999863	0.348329	0.002624	-0.006192	0.590175	-0.006926	0.016482	0.021363
PastReturn3	0.015807	0.014369	0.014086	-0.121084	-0.025518	0.015511	-0.079030	-0.016846	0.026789	-0.026429
Momentum3	0.062274	0.065769	0.065336	-0.294036	0.521171	-0.455794	-0.214996	0.376571	0.460758	0.565746
SMA3	0.999645	0.999612	0.999711	0.351433	-0.001583	-0.002304	0.592490	-0.010324	0.006620	0.016573
PastReturn4	0.015336	0.013797	0.013556	-0.139143	-0.029861	-0.011072	-0.098058	-0.007973	-0.025531	-0.014032
Momentum4	0.073012	0.075239	0.074748	-0.316231	0.443927	-0.392503	-0.229146	0.325617	0.412810	0.480529
SMA4	0.999482	0.999452	0.999555	0.354152	-0.003812	-0.000374	0.594669	-0.011952	0.002408	0.014159
PastReturn5	0.015681	0.014683	0.014408	-0.106702	0.024850	0.015878	-0.058309	-0.023265	-0.030115	0.016088
Momentum5	0.082292	0.083549	0.083017	-0.345298	0.393372	-0.362274	-0.252432	0.289160	0.360154	0.432454

	Low	Close	Adj Close	Volume	r	OC	HL	sign	PastReturn1	Momentum1
SMA5	0.999315	0.999297	0.999403	0.356564	-0.004975	0.000683	0.596574	-0.012865	-0.000220	0.012920
PastReturn6	0.014040	0.012771	0.012562	-0.101259	-0.076253	-0.000490	-0.066459	-0.003068	0.024677	-0.061215
Momentum6	0.090396	0.091058	0.090480	-0.361232	0.368398	-0.323410	-0.256071	0.255814	0.323339	0.400979
SMA6	0.999145	0.999144	0.999254	0.358909	-0.005664	0.001479	0.598449	-0.013419	-0.001827	0.012113
PastReturn7	0.014819	0.014958	0.014637	-0.095928	0.100212	-0.057964	-0.053395	0.015652	-0.075776	0.081279
Momentum7	0.098055	0.098099	0.097493	-0.379394	0.318193	-0.305800	-0.268178	0.236513	0.310351	0.352814
SMA7	0.998974	0.998990	0.999104	0.361106	-0.006238	0.001951	0.600120	-0.013714	-0.002899	0.011492
PastReturn8	0.013509	0.011508	0.011300	-0.073528	-0.120016	0.067859	-0.046534	-0.027825	0.100474	-0.087729
Momentum8	0.104364	0.104442	0.103780	-0.388739	0.329318	-0.307954	-0.267512	0.225072	0.264469	0.358322
SMA8	0.998809	0.998844	0.998962	0.363199	-0.006394	0.002331	0.601715	-0.013903	-0.003776	0.011261
PastReturn9	0.015339	0.015086	0.014758	-0.074995	0.119375	-0.049324	-0.067908	0.077490	-0.120269	0.108820
Momentum9	0.111163	0.110535	0.109848	-0.393791	0.278304	-0.272296	-0.270680	0.208748	0.282119	0.313419
SMA9	0.998645	0.998694	0.998815	0.365188	-0.006824	0.002839	0.603170	-0.014090	-0.004214	0.010807
PastReturn10	0.014414	0.012933	0.012626	-0.067761	-0.051642	0.043709	-0.055419	-0.048861	0.119124	-0.042584
Momentum10	0.116519	0.115748	0.115005	-0.396280	0.300966	-0.274245	-0.280405	0.224371	0.233194	0.332561
SMA10	0.998481	0.998549	0.998673	0.367025	-0.006850	0.003059	0.604529	-0.014192	-0.004842	0.010694
EMA	0.924891	0.925551	0.929884	0.403845	-0.001342	-0.012597	0.600928	-0.026466	-0.001461	0.016186
y	-0.059872	-0.061087	-0.060892	-0.086027	-0.009296	0.000529	-0.098658	0.024420	0.036045	-0.002858

40 rows × 40 columns

From the scatter plot and correlation matrix, we do see some interesting observations. The pair plot shows the relationship between various metrics such as open, close, volume and other moving averages. Given we have a cluster of data points without linearity,

there's a suggestion that there could be a complex relationship in our variables which are not intuitively obvious. As expected that our matrix is presenting, Close, Low and Adj Close are almost perfectly correlated given they're price measurements. While momentum and past returns does not have a strong relationship with price measurements, we do see some correlation which indicates usefulness in forecasting price movements. Looking across various simple moving averages (or SMA's), we do see strong positive correlations among themselves and price measures which showcases how they track our underlying price data. Finally, whats unexpected is our volume indicator as it has a lower correlation with price measures. While volume is important, this indicator does not follow price movements making it independent from immediate price changes.

Feature and object to predict set up

For our feature set, we'll define our data set as 'X' dropping the raw data along with additionals such as percentage returns 'r'. While doing so, we'll also define 'y' as our object to predict

In [ ]:
```
# Feature data set
X = QQQ_data.drop(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'r', 'y'], axis = 1)
# Object to predict
y = QQQ_data.y
```

In [ ]:
```
pd.Series(y).value_counts()
```

Out[ ]:
```
y
1    1561
0     787
Name: count, dtype: int64
```

In [ ]:
```
X.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
OC	2348.0	-0.062909	2.800008	-21.649994	-1.179993	-0.139999	0.889988	12.839996
HL	2348.0	3.516840	2.869889	0.360001	1.290007	2.660004	5.000000	22.779999
sign	2348.0	0.119250	0.992217	-1.000000	-1.000000	1.000000	1.000000	1.000000
PastReturn1	2348.0	0.000773	0.013866	-0.119788	-0.005082	0.001195	0.007812	0.084706
Momentum1	2348.0	0.152575	3.431727	-22.447952	-0.933699	0.199169	1.495003	19.225159
SMA1	2348.0	223.251299	103.489923	90.456673	131.317940	185.067398	313.878021	452.899994
PastReturn2	2348.0	0.000760	0.013865	-0.119788	-0.005105	0.001190	0.007804	0.084706
Momentum2	2348.0	0.306005	4.689841	-26.101562	-1.397182	0.455669	2.203438	24.380249
SMA2	2348.0	223.175011	103.439047	90.597561	131.271547	184.569973	314.390121	452.440002
PastReturn3	2348.0	0.000755	0.013865	-0.119788	-0.005118	0.001179	0.007804	0.084706
Momentum3	2348.0	0.455948	5.679564	-32.090729	-1.662878	0.581085	2.834629	25.864685
SMA3	2348.0	223.098439	103.391977	90.644524	131.252448	184.603755	314.370061	450.269999
PastReturn4	2348.0	0.000754	0.013865	-0.119788	-0.005118	0.001164	0.007804	0.084706
Momentum4	2348.0	0.604456	6.462585	-33.970306	-1.833305	0.814697	3.321922	32.805847
SMA4	2348.0	223.022667	103.347248	90.668005	131.298321	184.427338	314.518234	448.472496
PastReturn5	2348.0	0.000749	0.013867	-0.119788	-0.005126	0.001153	0.007804	0.084706
Momentum5	2348.0	0.752517	7.154548	-37.223358	-1.997768	0.972717	3.844156	31.843750
SMA5	2348.0	222.947502	103.304015	90.945044	131.281377	184.135918	314.563582	447.189996
PastReturn6	2348.0	0.000745	0.013868	-0.119788	-0.005127	0.001148	0.007804	0.084706
Momentum6	2348.0	0.899711	7.799631	-39.848984	-2.029165	1.158493	4.228348	38.626007
SMA6	2348.0	222.872715	103.261687	91.145393	131.356028	183.987996	314.654504	446.161662
PastReturn7	2348.0	0.000753	0.013873	-0.119788	-0.005127	0.001153	0.007812	0.084706

	count	mean	std	min	25%	50%	75%	max
Momentum7	2348.0	1.046228	8.315409	-41.203262	-1.985935	1.445274	4.673637	33.568878
SMA7	2348.0	222.798269	103.219894	91.571577	131.315772	183.971886	314.891546	445.289995
PastReturn8	2348.0	0.000759	0.013875	-0.119788	-0.005127	0.001164	0.007829	0.084706
Momentum8	2348.0	1.193623	8.900016	-47.662903	-1.961578	1.552155	5.323849	41.277771
SMA8	2348.0	222.724119	103.178570	92.057912	131.431339	184.012238	314.678761	444.736252
PastReturn9	2348.0	0.000748	0.013877	-0.119788	-0.005133	0.001153	0.007812	0.084706
Momentum9	2348.0	1.341499	9.354650	-42.119904	-1.914299	1.676552	5.711254	40.981995
SMA9	2348.0	222.650070	103.137552	92.437214	131.467873	184.040512	314.706163	444.333333
PastReturn10	2348.0	0.000734	0.013875	-0.119788	-0.005145	0.001148	0.007804	0.084706
Momentum10	2348.0	1.486806	9.912733	-45.909149	-2.083601	1.960732	6.157803	46.463013
SMA10	2348.0	222.576043	103.096849	92.786672	131.430291	183.988173	314.517390	444.110001
EMA	2348.0	144.632903	48.786828	95.687511	100.388935	124.861326	191.038874	252.328437

Split the data

Now that we have our data set and features, it's important for us to split the data into two segments: Training and testing. Training is used to train our model while the testing set is used to see how well it performed. This is important as it'll help us not have overfitting. Overfitting is when the training set does well, but performs poorly on new data.

In [ ]:
```
Test_set_percentage = 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=Test_set_percentage,shuffle=False)
```

Base Model

Before diving into the modeling, lets define Support Vector Machines (SVM). The resource used through these details are from *Module 4 Lecture 1 Support Vector Machines by Paul Wilmott*. During the math specifications, I'll be noting the slides used during that lecture. An SVM is a supervised learning classification and regression technique. This method will divide our data classifications using a hyperplane. The idea behind this is to maximimze our margin between classes. In other words, we're taking the distance

between our hyperplane and the nearest data point from either class (which is also known as support vectors). As an example, if we are on a grid and have blue dots on the left side, and red dots on the right side, the hyperplane is a boundary which separates the data to designate them as different classes. The margin mentioned signifies as the distance between this hyperplane and our closest data from either class (the red or blue sets described). For Margin, we can designate it into two groups: Hard and Soft Margin.

For Hard Margin, some of the requirements include the data set being correctly classified along with the data points being outside of the margin. The advantage of this method is if our data is linearly separable, we will classify our training data set perfectly. Saying this, hard margin is very sensitive to outliers to the extent that if a single point is not properly classified and or within the margin, SVM can't find a solution. From the lecture notes, we can detail the math as below:

Hyperplane Equation

From Slide 9:

$$\theta^T x + \theta_0 = 0$$

Here, θ is the vector orthogonal to the hyperplane, and θ_0 is the bias term.

Margin Constraints

from slide 10:

For the two classes labeled as $+1$ and -1:

$$\theta^T x^{(i)} + \theta_0 \geq 1 \quad \text{for} \quad y^{(i)} = +1$$
$$\theta^T x^{(i)} + \theta_0 \leq -1 \quad \text{for} \quad y^{(i)} = -1$$

These can be combined into:

$$y^{(i)}(\theta^T x^{(i)} + \theta_0) \geq 1$$

Optimization Problem

From Slide 11: The objective is to maximize the margin, which is equivalent to minimizing $|\theta|$:

$$\min_{\theta,\theta_0} \frac{1}{2}|\theta|^2$$

Subject to the constraint:

$$y^{(i)}(\theta^T x^{(i)} + \theta_0) \geq 1 \quad \forall i$$

Soft Margin on the other hand acts differently from Hard Margin. Instead of needing to be linearly separable with perfect classification, soft margin is able to handle non linearly separable data while also allowing for misclassification. The advantage of this is we are not sensitive to outliers. Given we allow misclassification, we use hyperparameters in order to tune our model. With this however, we do try to minimize these errors through our cost function. From our lecture notes, we detail the formulas for soft margin below:

Loss Function

From Slide 20:

$$J = \frac{1}{N}\sum_{i=1}^{N} \max\left(1 - y^{(i)}(\theta^T x^{(i)} + \theta_0), 0\right) + \lambda|\theta|^2$$

Here, N is the number of samples, and λ is a regularization parameter. The purpose of this is it'll control the trade-off between maximizing the margin and minimizing the error of our classification.

Optimization Problem

The objective is to minimize the loss function J:

$$\min_{\theta,\theta_0} \left(\frac{1}{N}\sum_{i=1}^{N} \max\left(1 - y^{(i)}(\theta^T x^{(i)} + \theta_0), 0\right) + \lambda|\theta|^2 \right)$$

As our task is to predict stock movement, we could use two different models, Support Vector Classification (SVC) and Support Vector Regression (SVR). SVC focuses on the categorical outcome of our data or in other words, we're looking at our target variable and predicting if it'll move up or down. We maximize our margin by finding the hyperplane which best separates our data. This is to ensure we get the widest possible gap between the support vectors of our classes. SVR on the other hand will be used for continuous outcomes such as predicting numerical values of the equities price. For the scope of this assignment, we will be using SVC to predict positive moves.

Within these models, we can alter the Kernel and Scalers. A kernal is a function which transforms data into a higher dimensional space to make them easier to classify, while scaler normalizes our data to make our features contribute equally to our models performance. For our kernel, we can use: 'linear' which is best used when the data is linearly separable, 'poly' for when there's a polynomial relationship between features, 'rbf' when we need to capture non linearity, and 'sigmoid' for when we want to map our inputs between 0 and 1 making the data binary classifications. As for scalers, a few options available are: 'StandardScaler()' for when we want to 'standardize' our features by getting rid of the mean and scaling to unit variance, 'MinMaxScaler()' for when we want to scale our features into a range such as o and 1, 'RobustScaler()' for when we want to scale using median and interquartile ranges making our data more robust against outliers, and finally 'MaxAbsScaler()' when we need to scale the data to -1 and 1 inclusively for maximum absolute value in each feature.

For our analysis on the best model to use, we will be observing train and test accuracy scores, confusion matrix's, receiver operator characteristic curves (ROC) and a classification report. For information on this topic, I used Python lab 'Trend prediction using Logistic Regression' by Kannan Singaravelu. Descriptions below are found directly in his lecture without modification. Credit to these descriptions are reserved to Kannan Singaravelu and are used exclusively to clearly illustrate what these metrics are.

Confusion Matrix is a table used to describe the performance of a classification model on a set of test data for which the true values are known.

Outcome	Position
True Negative	upper-left
False Negative	lower-left
False Positive	upper-right
True Positive	lower-right

True Positive is an outcome where the model correctly predicts the positive class. Similarly, a **True Negative** is an outcome where the model correctly predicts the negative class.

False Positive is an outcome where the model incorrectly predicts the positive class. And a **False Negative** is an outcome where the model incorrectly predicts the negative class.

Receiver Operator Characterisitc Curve (ROC) The area under the ROC curve (AUC) is a measure of how well a model can distinguish between two classes. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate

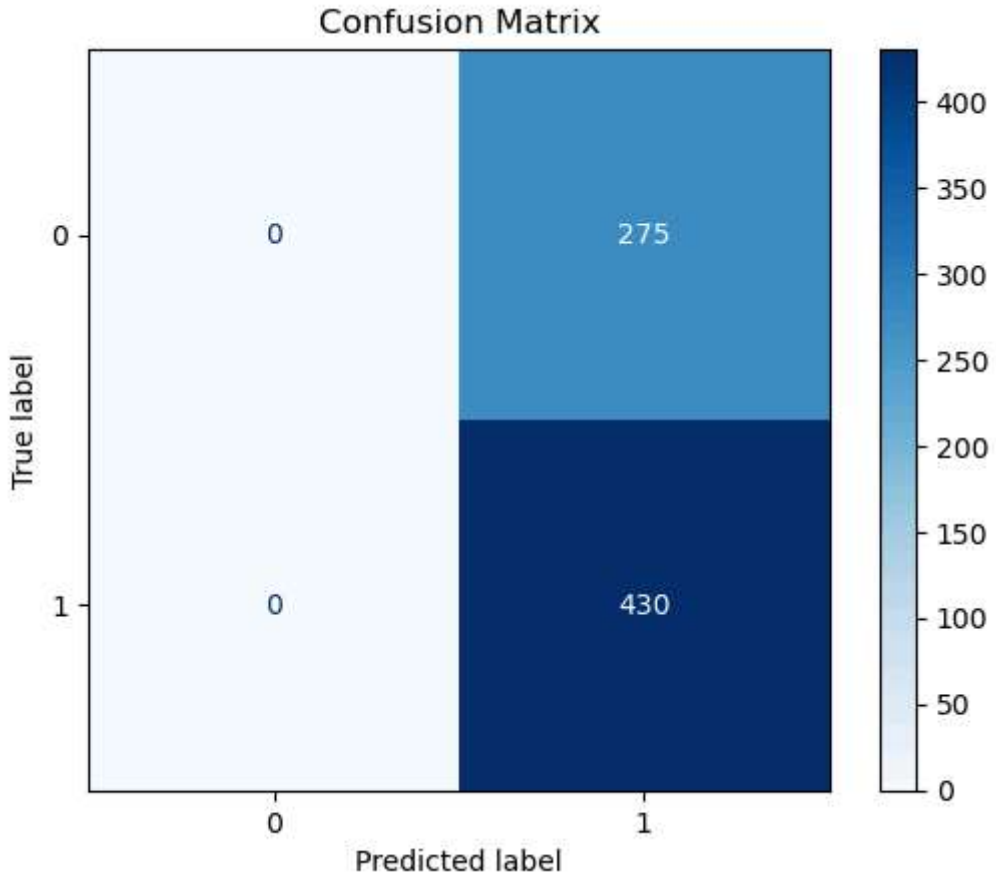
(FPR) at various classification thresholds.

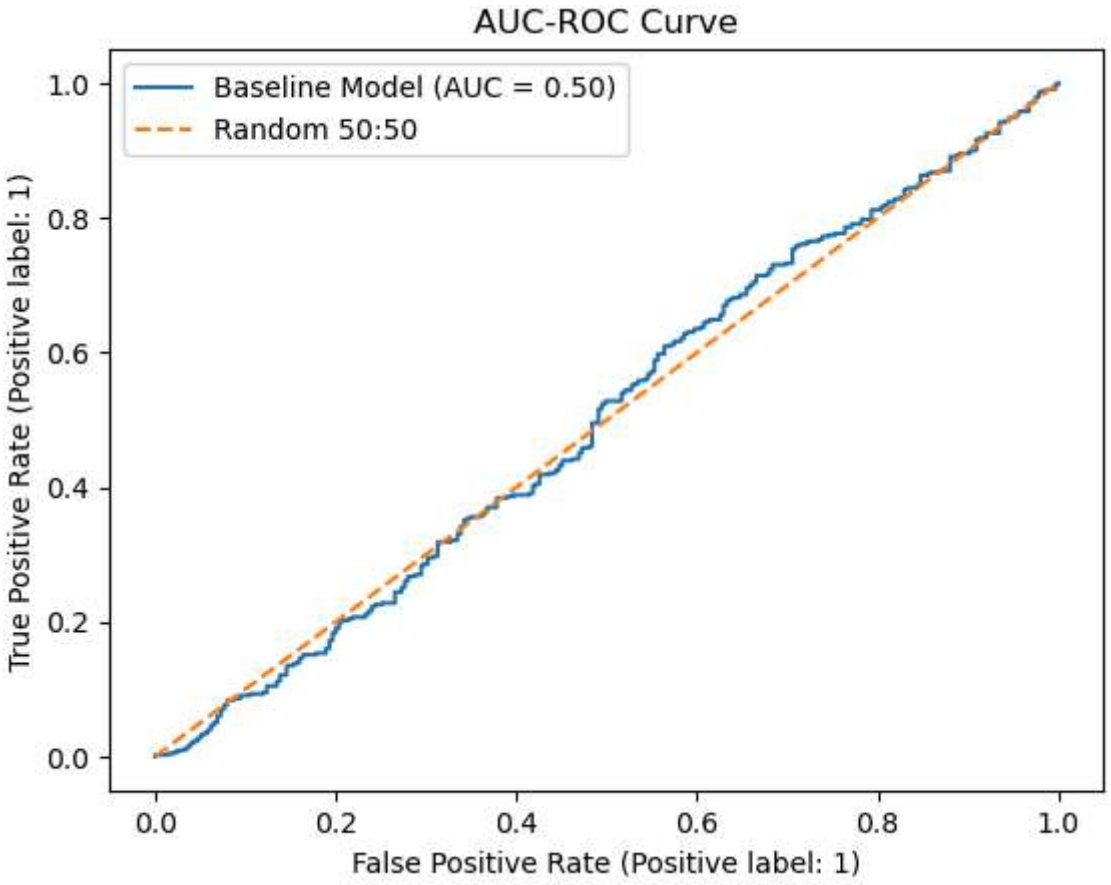
Classification Report A classification report is used to measure the quality of predictions from a classification algorithm.

Finally, adding to the concept of hard and soft margin, Pipeline sets a default value of 'c=1.0' which means that our base model will be using soft margin. We will dive into this later in the paper.

In [ ]:
```
classifier = Pipeline([
    ('scaler', MinMaxScaler()),
    ('svm', SVC(kernel='rbf', probability=True))
])

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
acc_train = accuracy_score(y_train, classifier.predict(X_train))
acc_test = accuracy_score(y_test, y_pred)
print(f'Baseline Model -- Train Accuracy: {acc_train:0.4}, Test Accuracy: {acc_test:0.4}')

ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test, cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()

RocCurveDisplay.from_estimator(classifier, X_test, y_test, name='Baseline Model')
plt.plot([0, 1], [0, 1], linestyle="--", label='Random 50:50')
plt.legend()
plt.title("AUC-ROC Curve")
plt.show()

print(classification_report(y_test, y_pred))
```

```
Baseline Model -- Train Accuracy: 0.6902, Test Accuracy: 0.6099
```


Confusion Matrix
0
275
0
430
True label
0
1
Predicted label
400
350
300
250
200
150
100
50
0



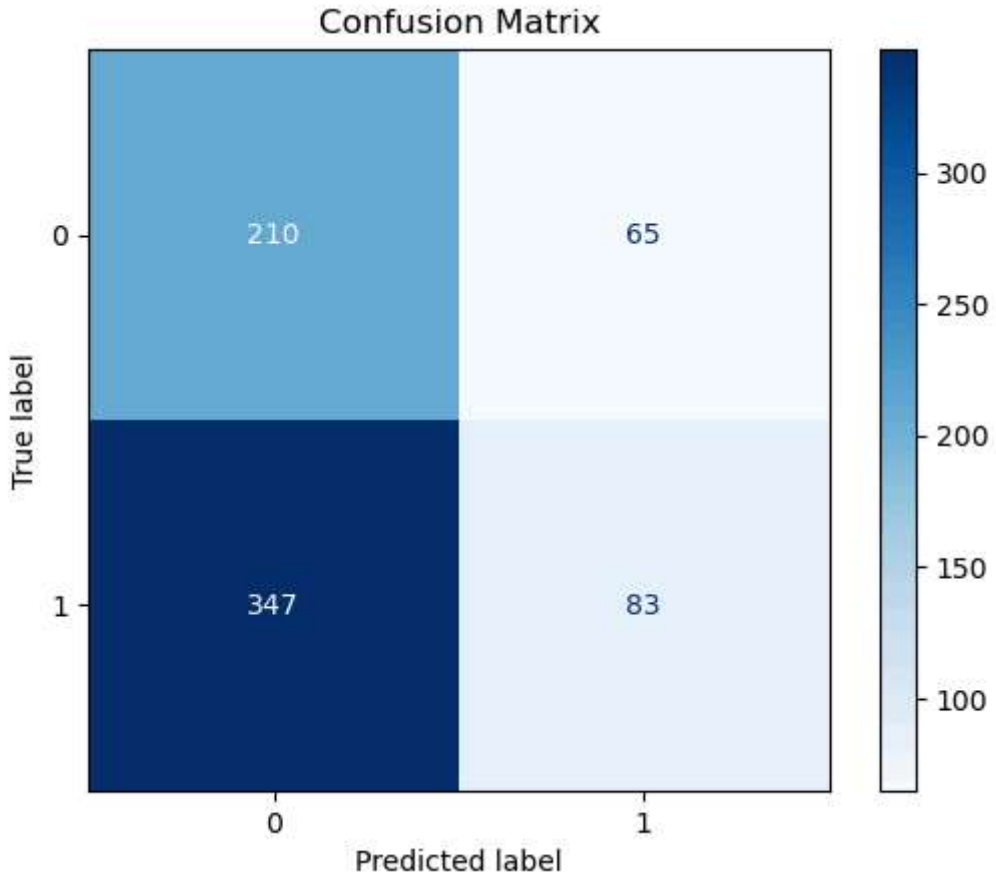
	precision	recall	f1-score	support
0	0.00	0.00	0.00	275
1	0.61	1.00	0.76	430
accuracy			0.61	705
macro avg	0.30	0.50	0.38	705
weighted avg	0.37	0.61	0.46	705

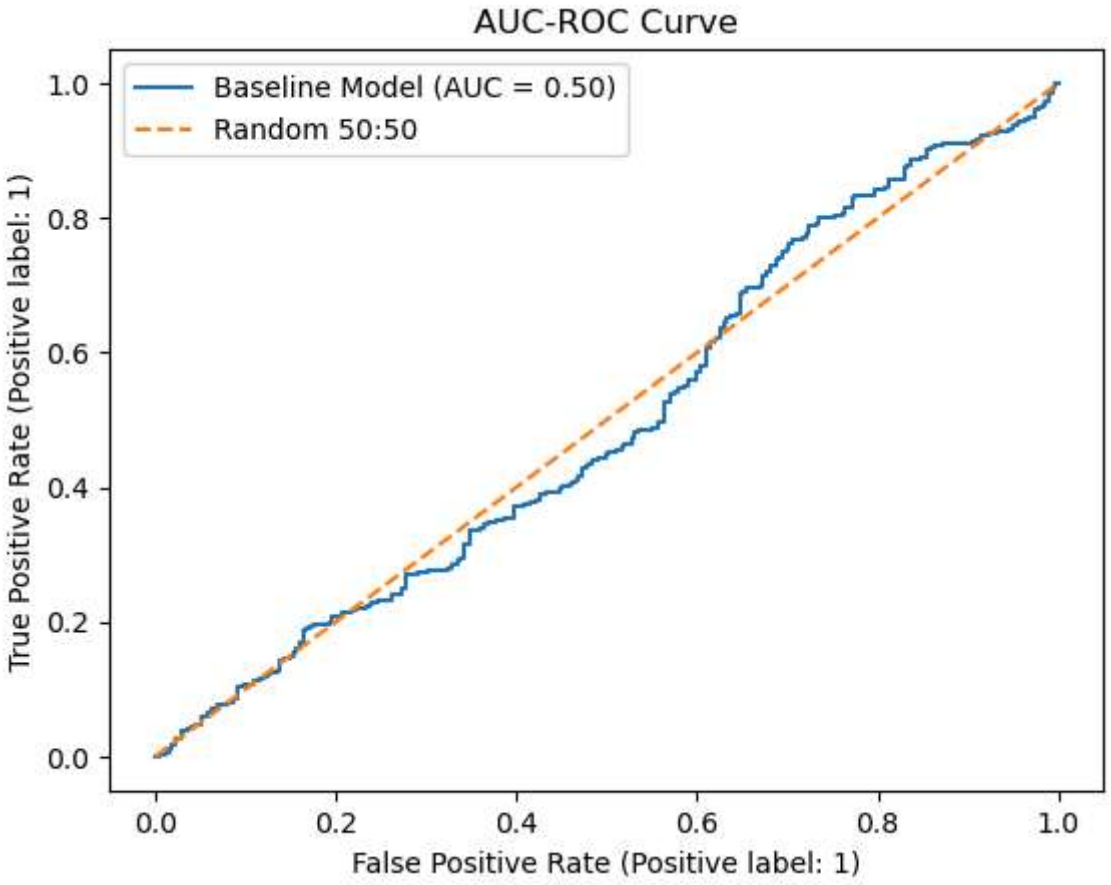
The base model does demonstrate a high test and training accuracy score, but this can be misleading. For instance in the confusion matrix, we can see our predicted label of 0 is being ignored. In order to address this problem, we will implement Class_weight which will adjust the penalties for misclassifications to balance each class. By using 'Class_weights='Balanced', we will automatically

determine the appropriate weights given our class frequencies of training data. In addition, I'll add 'probability=True' as it'll ensure that SVM classifers can output probabilities for each class, rather than just the class labels.

In [ ]:
```
classifier = Pipeline([
    ('scaler', MinMaxScaler()),
    ('svm', SVC(kernel='rbf', class_weight='balanced', probability=True))
])

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
acc_train = accuracy_score(y_train, classifier.predict(X_train))
acc_test = accuracy_score(y_test, y_pred)
print(f'Baseline Model -- Train Accuracy: {acc_train:0.4}, Test Accuracy: {acc_test:0.4}')

ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test, cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()

RocCurveDisplay.from_estimator(classifier, X_test, y_test, name='Baseline Model')
plt.plot([0, 1], [0, 1], linestyle="--", label='Random 50:50')
plt.legend()
plt.title("AUC-ROC Curve")
plt.show()

print(classification_report(y_test, y_pred))
```

```
Baseline Model -- Train Accuracy: 0.6348, Test Accuracy: 0.4156
```


Confusion Matrix
210
65
347
83
0
1
True label
0
1
Predicted label
300
250
200
150
100



	precision	recall	f1-score	support
0	0.38	0.76	0.50	275
1	0.56	0.19	0.29	430
accuracy			0.42	705
macro avg	0.47	0.48	0.40	705
weighted avg	0.49	0.42	0.37	705

Lets compare between the two base models created. The first one has a higher test and training accuracy compared to the second model. Saying this, the issue with the first model is while it can correctly predict all positive cases, it incorrectly predicts all negative cases which is concerning. The second model has a balanced confusion matrix showing that it can better determine true positives

and true negatives. For this reason, while base model 2 has a lower test accuracy, we will proceed with using class_weight='balanced' along with it's metrics as a ground level comparison.

Filtering our Feature Set

Resource used when speaking on filtration methods is Module 4 on Supervised Learning 1 by Kannan Singaravelu.

For this exam, I will be using SelectKBest, RFECV, LASSO, Ridge and ElasticNet.

SelectKBest looks at the k highest scores, and selects those features accordingly. In other words, the best features are selected based on univariate statistical tests. *Resource used: Module 4 Lecture 4 exercise solutions sheet*

RFECV uses recursive feature elimination along with cross validation to automatically use the optimal set of features. *Resource used: Exercise solutions for Module 4 lecture 4.* Cross validation is a process which uses re-sampling in order to evaluate the models performance (slide 49).

LASSO (Least Absolute Shrinkage and Selection Operator) is a type of linear regression. This filter adds absolute value of the magnitude of the coefficients as a penalty to the model (also known as L1 penalty). The overall goal of this is to minimize the cost function.

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(w_0 + \sum_{j=1}^{p} x_{ij}w_j\right)\right)^2 + \lambda\sum_{j=1}^{p}|w_j|$$

Here, λ is the regularization penalty. For our coefficients, it'll control how much the shrinkage will be. As λ increases, more coefficients are shrunk towards zero, which will then select a more basic model by removing some features. (Slide 57)

Ridge is another type of linear regression which also has a regularization term. This term however penalizes the sum of the squared coefficients instead of their absolute values.

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(w_0 - \sum_{j=1}^{p} x_{ij}w_j\right)\right)^2 + \lambda\sum_{j=1}^{p} w_j^2$$

Much like LASSO, the first term represents the residual sum of squares (RSS), while the second term, $\lambda\sum_{j=1}^{p} w_j^2$, is the L2 penalty term that shrinks coefficients towards zero. (Slide 59)

ElasticNet is the final linear regression type that we will be using which combines the properties from both LASSO and Ridge. Adding L1 and L2 penalties to the cost function, it helps with multicollinearity.

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(w_0 + \sum_{j=1}^{p} x_{ij}w_j\right)\right)^2 + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{p} w_j^2 + \alpha\sum_{j=1}^{p}|w_j|\right)$$

Here:

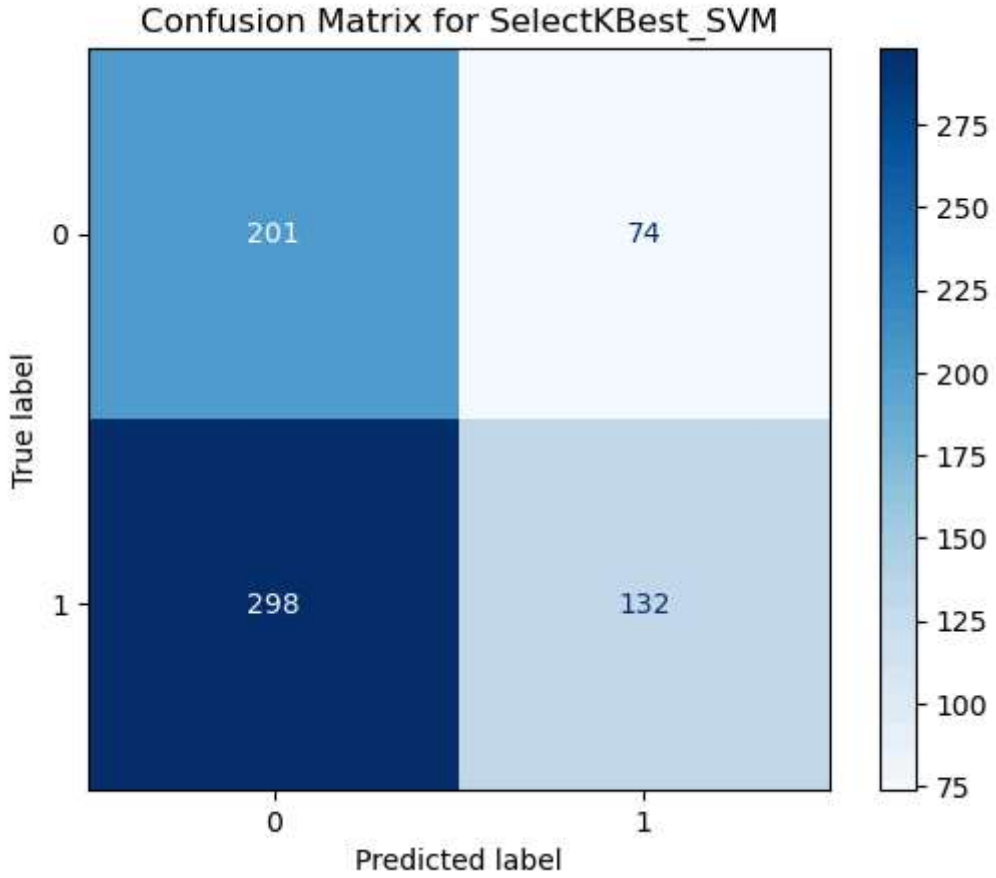
- λ is the regularization for controlling our shrinkage.
- α controls the balance between L1 (LASSO) and L2 (Ridge) penalties:
  - When $\alpha = 1$, ElasticNet becomes LASSO.
  - When $\alpha = 0$, ElasticNet becomes Ridge.
  - For $0 < \alpha < 1$, the penalty is a mix of L1 and L2 norms. (Slide 61)

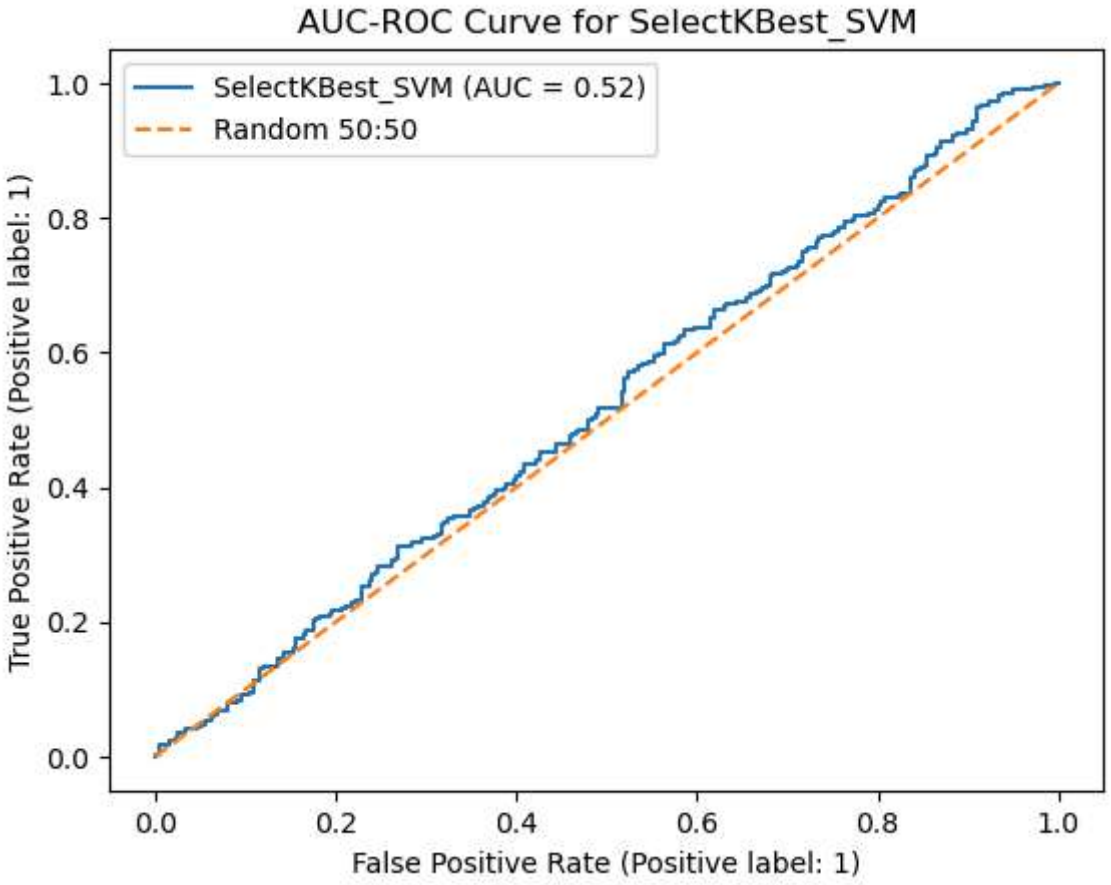
Points to note about the code below. Each pipeline includes a balanced class weight, minmaxscaler, kernal = 'rbf' and probability = true in order to match the base models parameters. In addition, I have made lasso alpha = 0.001 as if we increase this value, all features will be filtered through. I wanted ElasticNet to have a balance between L1 and L2 penalization which is why alpha = 0.5. For consistency with ElasticNets alpha, I have also included ridge to be alpha = 0.5

In [ ]:
```
pipelines = {
    'SelectKBest_SVM': Pipeline([
        ('scaler', MinMaxScaler()),
        ('selector', SelectKBest(score_func=f_classif, k=10)),
        ('classifier', SVC(kernel='rbf', class_weight='balanced', probability=True))
    ]),
    'RFECV_SVM': Pipeline([
        ('scaler', MinMaxScaler()),
        ('selector', RFECV(estimator=SVC(kernel='linear'), step=1, cv=5, min_features_to_select=10)),
        ('classifier', SVC(kernel='rbf', class_weight='balanced', probability=True))
    ]),
    'Lasso_SVM': Pipeline([
        ('scaler', MinMaxScaler()),
        ('feature_selection', SelectFromModel(Lasso(alpha=0.001))),
        ('classifier', SVC(kernel='rbf', class_weight='balanced', probability=True))
    ]),
    'Ridge_SVM': Pipeline([
        ('scaler', MinMaxScaler()),
        ('feature_selection', SelectFromModel(Ridge(alpha=0.5))),
```

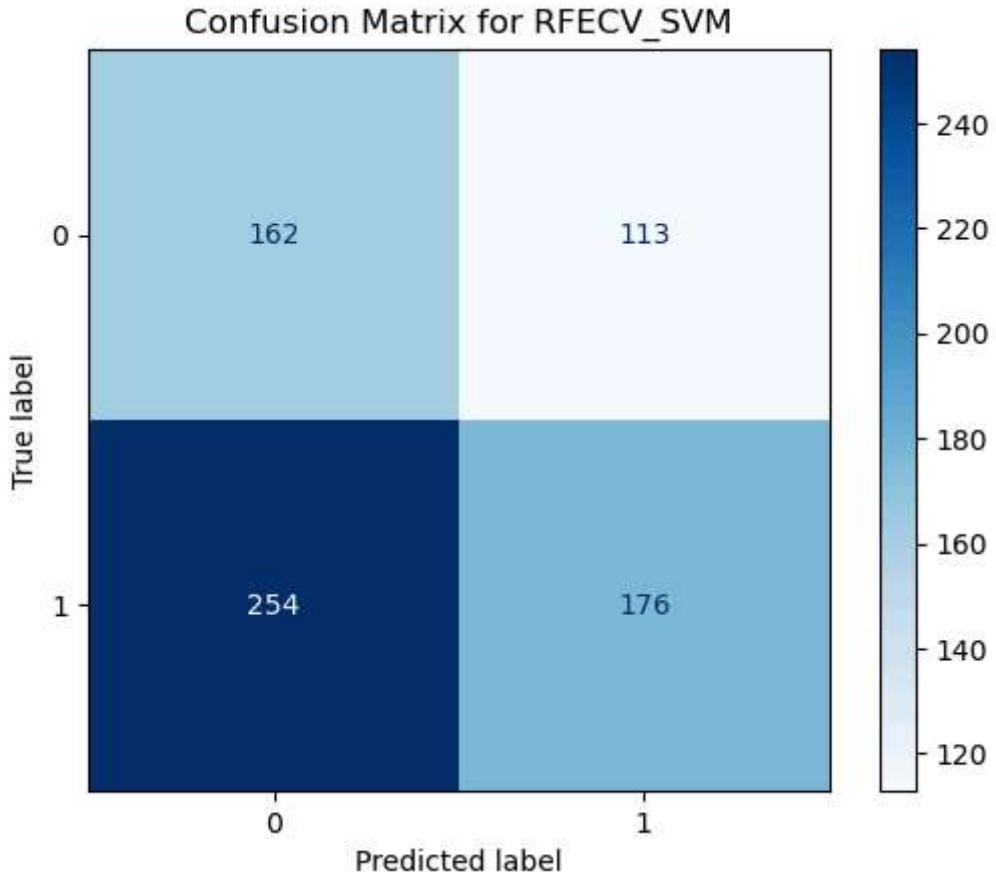
```
        ('classifier', SVC(kernel='rbf', class_weight='balanced', probability=True))
    ]),
    'ElasticNet_SVM': Pipeline([
        ('scaler', MinMaxScaler()),
        ('feature_selection', SelectFromModel(ElasticNet(alpha=0.5, l1_ratio=0.5))),
        ('classifier', SVC(kernel='rbf', class_weight='balanced', probability=True))
    ])
}

for name, pipeline in pipelines.items():

    print(f'\n{"="*10} {name} {"="*10}\n')

    pipeline.fit(X_train, y_train)

    y_pred = pipeline.predict(X_test)

    acc_train = accuracy_score(y_train, pipeline.predict(X_train))
    acc_test = accuracy_score(y_test, y_pred)

    print(f'{name} -- Train Accuracy: {acc_train:.4f}, Test Accuracy: {acc_test:.4f}\n')
    print(f'Classification Report for {name}:')
    print(classification_report(y_test, y_pred, zero_division=0))

    disp = ConfusionMatrixDisplay.from_estimator(pipeline, X_test, y_test, cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix for {name}')
    plt.show()

    disp = RocCurveDisplay.from_estimator(pipeline, X_test, y_test, name=name)
    plt.title(f'AUC-ROC Curve for {name}')
    plt.plot([0, 1], [0, 1], linestyle="--", label='Random 50:50')
    plt.legend()
    plt.show()
```

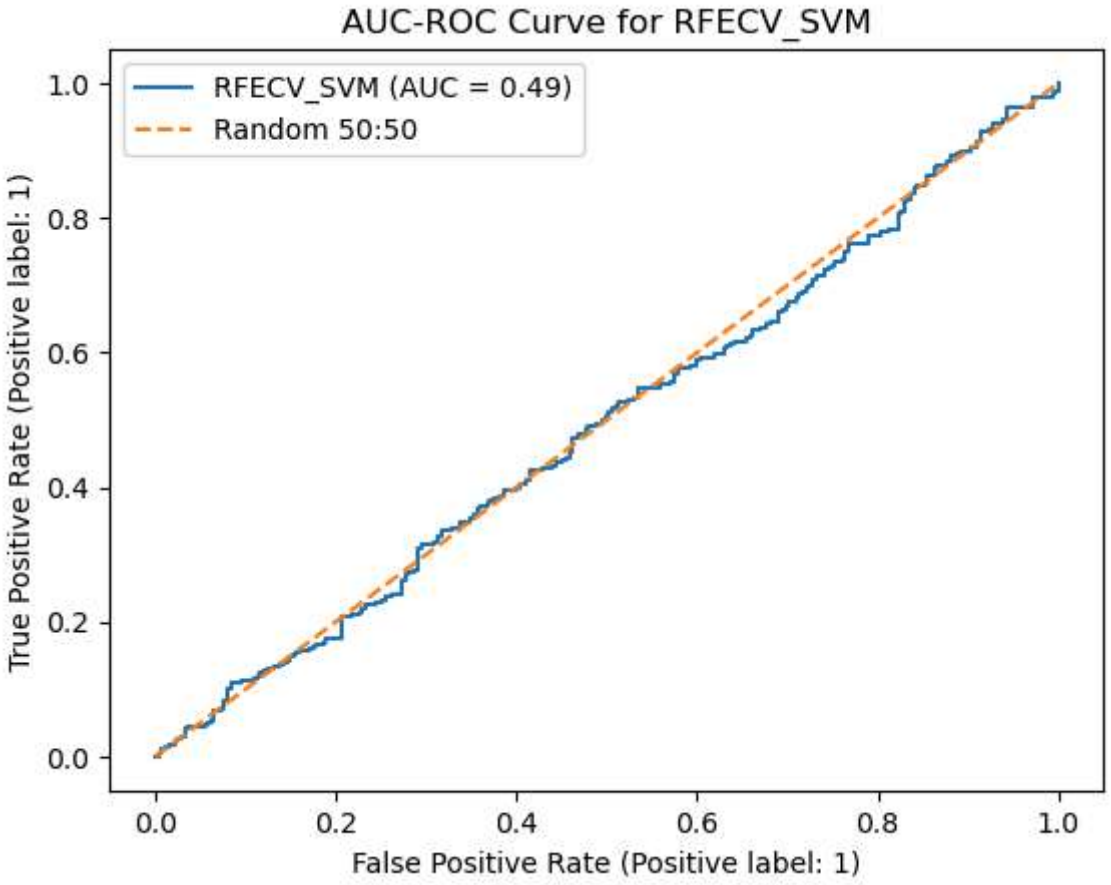
```
========== SelectKBest_SVM ==========

SelectKBest_SVM -- Train Accuracy: 0.6330, Test Accuracy: 0.4723

Classification Report for SelectKBest_SVM:
              precision    recall  f1-score   support

           0       0.40      0.73      0.52       275
           1       0.64      0.31      0.42       430

    accuracy                           0.47       705
   macro avg       0.52      0.52      0.47       705
weighted avg       0.55      0.47      0.46       705
```



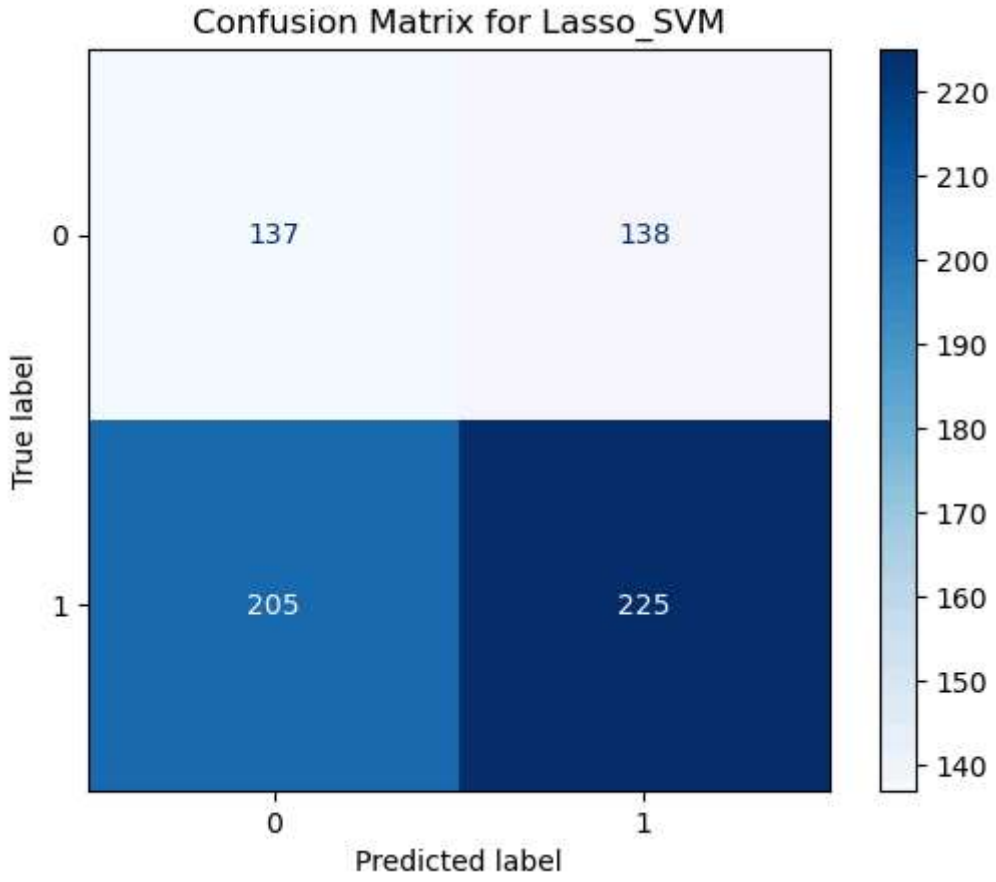


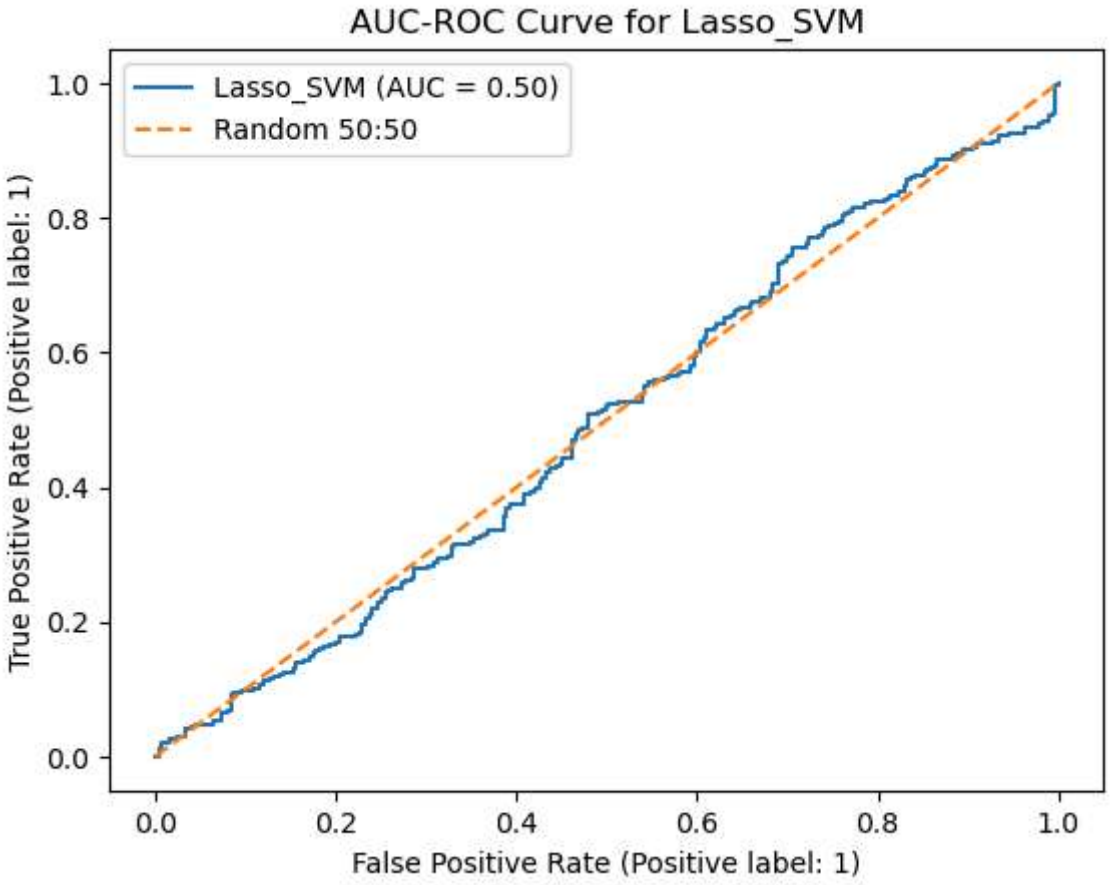

```
========== RFECV_SVM ==========

RFECV_SVM -- Train Accuracy: 0.6579, Test Accuracy: 0.4794

Classification Report for RFECV_SVM:
              precision    recall  f1-score   support

           0       0.39      0.59      0.47       275
           1       0.61      0.41      0.49       430

    accuracy                           0.48       705
   macro avg       0.50      0.50      0.48       705
weighted avg       0.52      0.48      0.48       705
```


Confusion Matrix for RFECV_SVM
True label
Predicted label
0
1
0
1
162
113
254
176
240
220
200
180
160
140
120

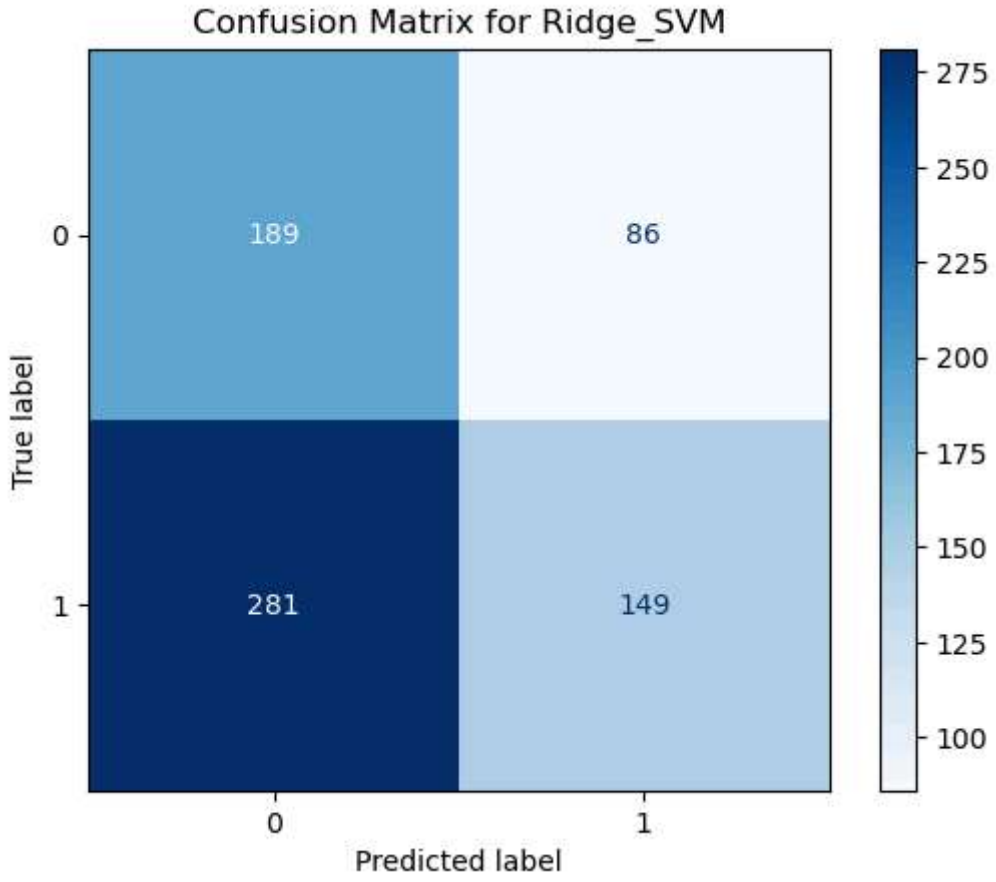


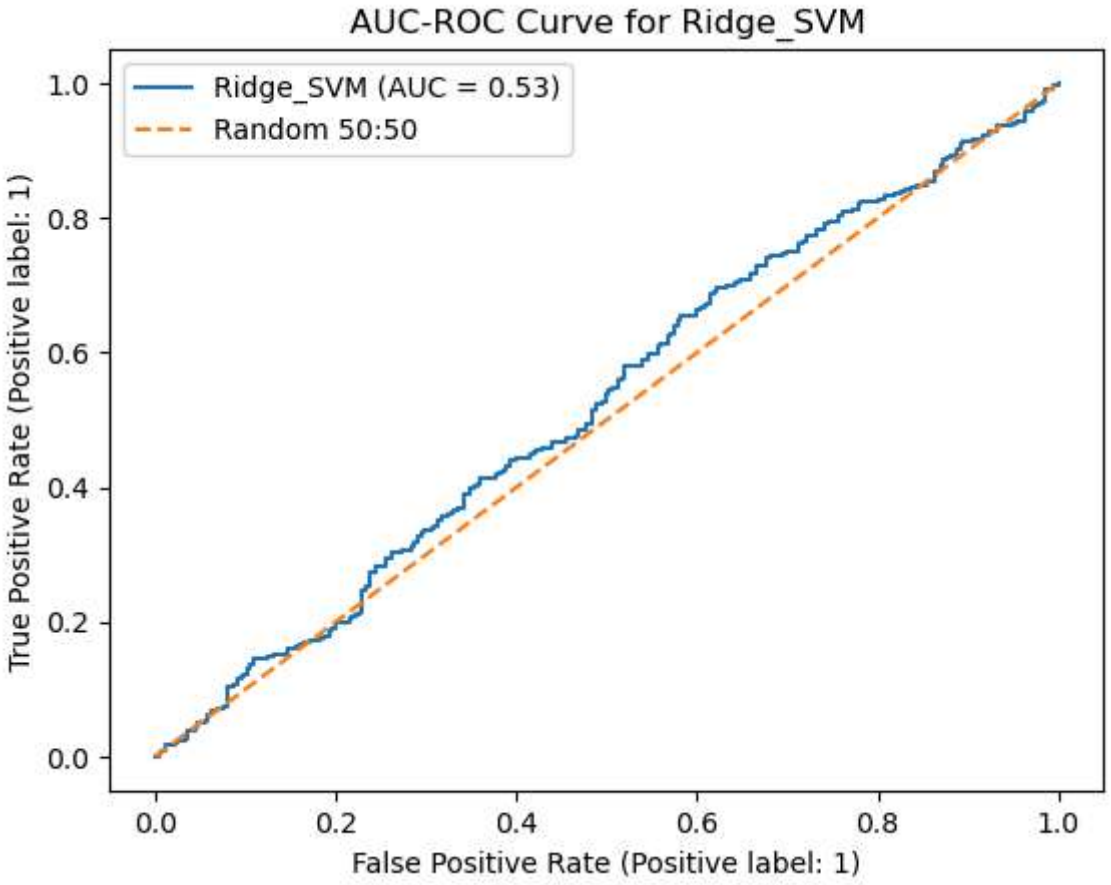

```
========== Lasso_SVM ==========

Lasso_SVM -- Train Accuracy: 0.6598, Test Accuracy: 0.5135

Classification Report for Lasso_SVM:
              precision    recall  f1-score   support

           0       0.40      0.50      0.44       275
           1       0.62      0.52      0.57       430

    accuracy                           0.51       705
   macro avg       0.51      0.51      0.51       705
weighted avg       0.53      0.51      0.52       705
```


Confusion Matrix for Lasso_SVM
137
138
205
225
0
1
True label
0
1
Predicted label
220
210
200
190
180
170
160
150
140

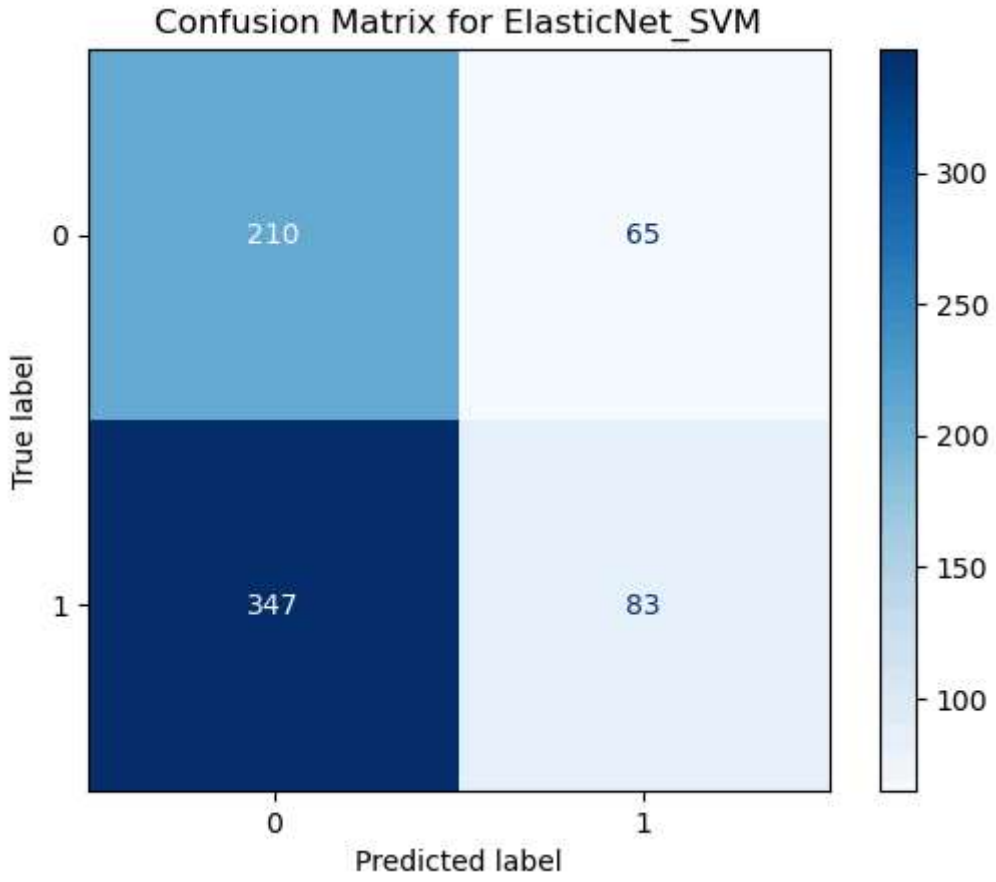


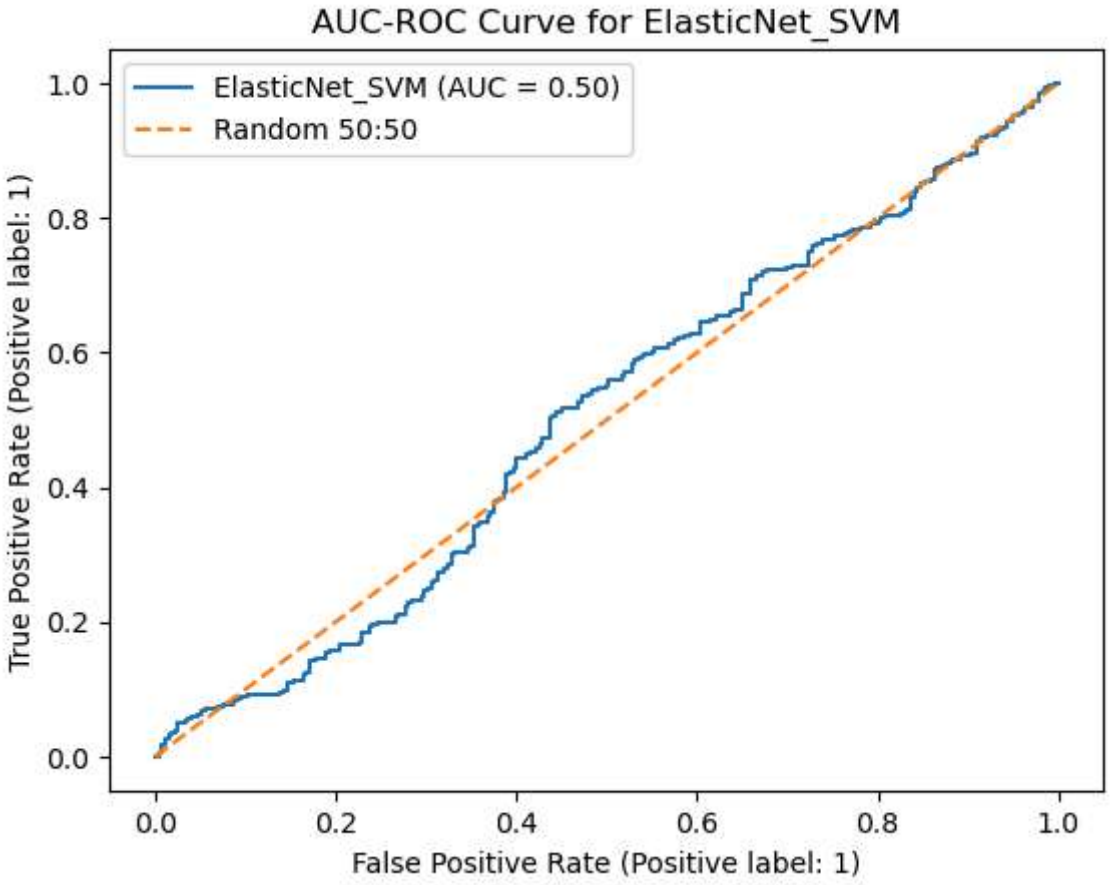

```
========== Ridge_SVM ==========

Ridge_SVM -- Train Accuracy: 0.6592, Test Accuracy: 0.4794

Classification Report for Ridge_SVM:
              precision    recall  f1-score   support

           0       0.40      0.69      0.51       275
           1       0.63      0.35      0.45       430

    accuracy                           0.48       705
   macro avg       0.52      0.52      0.48       705
weighted avg       0.54      0.48      0.47       705
```


Confusion Matrix for Ridge_SVM
189
86
281
149
0
1
True label
0
1
Predicted label
275
250
225
200
175
150
125
100




```
========== ElasticNet_SVM ==========

ElasticNet_SVM -- Train Accuracy: 0.6348, Test Accuracy: 0.4156

Classification Report for ElasticNet_SVM:
              precision    recall  f1-score   support

           0       0.38      0.76      0.50       275
           1       0.56      0.19      0.29       430

    accuracy                           0.42       705
   macro avg       0.47      0.48      0.40       705
weighted avg       0.49      0.42      0.37       705
```


Confusion Matrix for ElasticNet_SVM
210
65
347
83
0
1
True label
0
1
Predicted label
300
250
200
150
100



Which is the best model?

Out of the 5 models used, the two which stick out when comparing accuracy tests alone are LASSO and Ridge. LASSO has a higher test accuracy along with a more balanced precision and recall which can be observed in the classification report. While Ridge does have a better recall for class 0, we see an impact on class 1 which falters. Finally, even though Ridge has a slightly higher AUC score, LASSO presents a more balanced confusion matrix with fewer false negatives and false positives.

Given LASSO is a more balanced model overall, we will use this for hyper-parameterization.

Hyper-parameters is a means of tuning the model such as changing C, kernel, degree, gamma and more. In doing so, we can influence the models performance to have an increase in accuracy scores. The way we will move forward with this technique is by

using GridSearchCV. This is a tool for hyperparameter optimization which takes many combinations to identify which is best to configure for our model. It uses cross-validation to see what parameters would produce the best performance. The parameters chosen here are alpha, C, gamma, and kernel.

Alpha will control the strength of LASSO regularization. The smaller the value, the more features will be retained.

C is the regularization for SVM, where it's inversely proportional to the strength of regularization. In other words, we're seeing how much SVM allows for misclassifications. The lower the value, the stronger our regularization will be. This acts as our distinction between hard and soft margin being used where the smaller the value, the softer the margin.

Gamma states how much a single training example has on our model. It'll help in balancing between under and over fitting. The lower values mean 'far' influence and higher values mean 'close' influence.

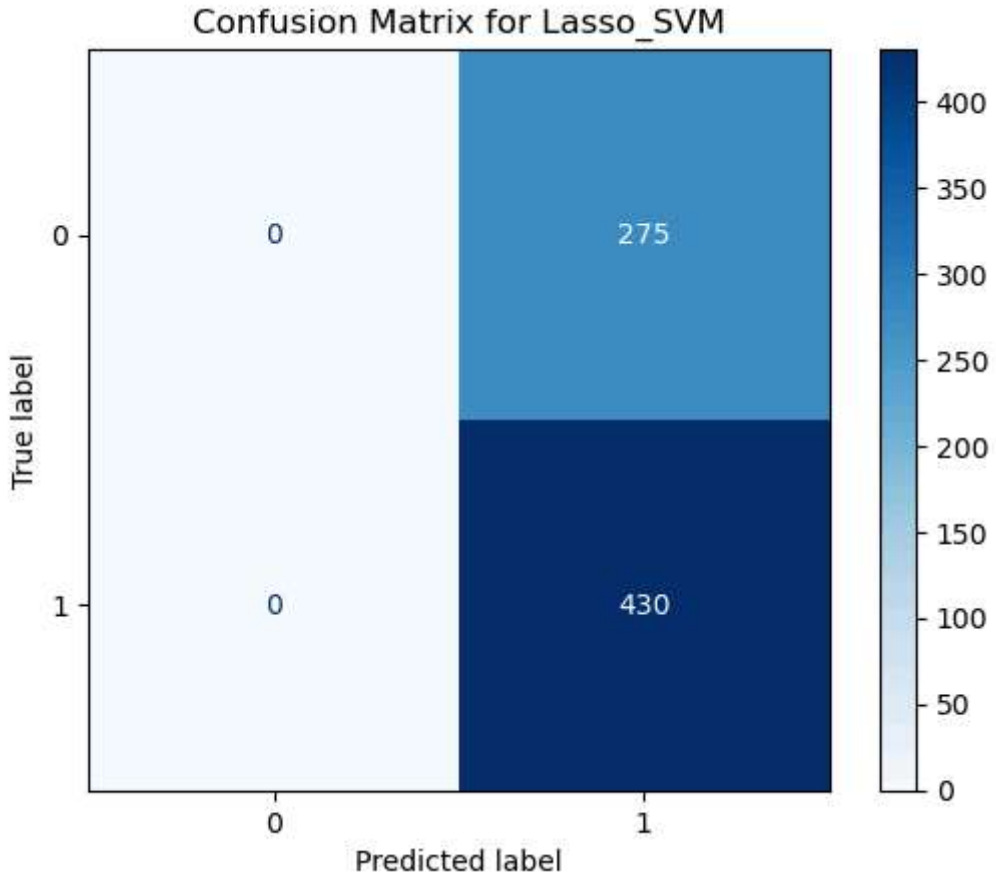
Kernel will capture how the function defines the transformation of the inputs.

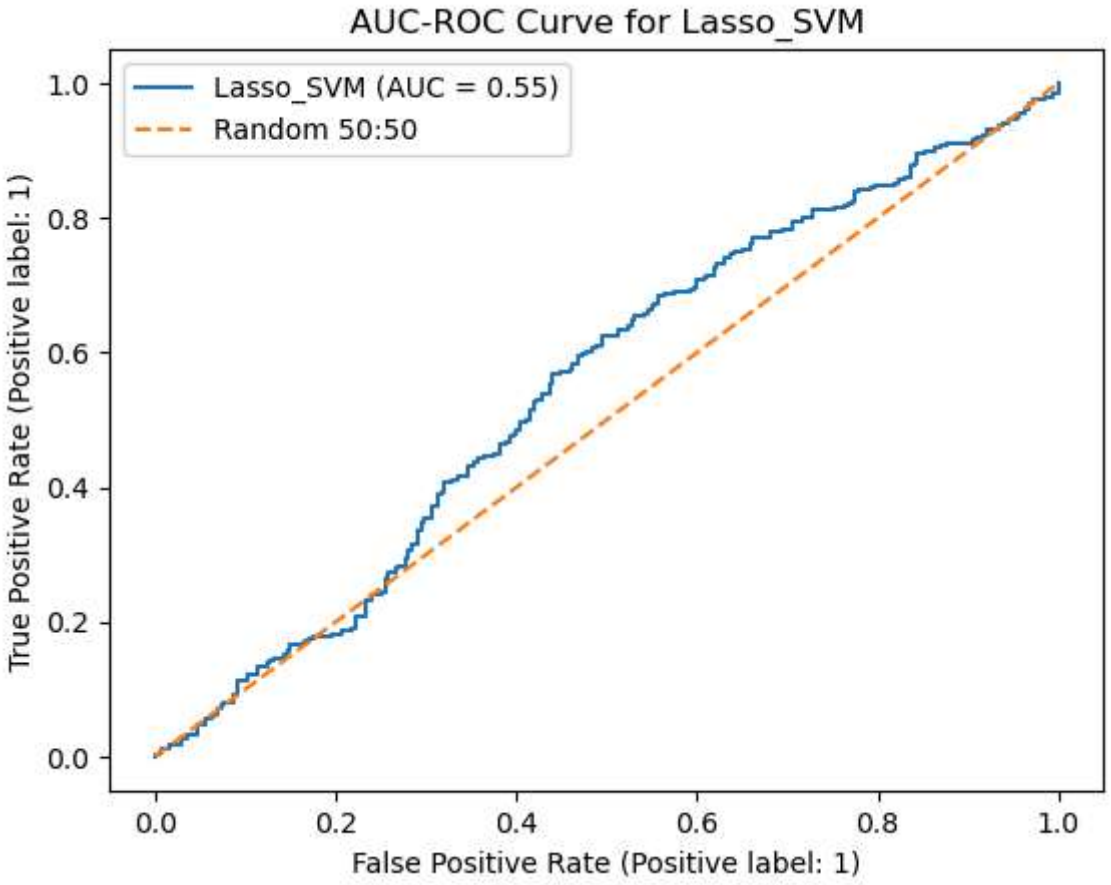
Hyper-parameter Tuning

In [ ]:
```
lasso_svm_pipeline = pipelines['Lasso_SVM']

param_grid = {
    'feature_selection__estimator__alpha': [0.0001, 0.001, 0.01, 0.1, 1],
    'classifier__C': [0.01, 0.1, 1, 10, 100],
    'classifier__gamma': [0.001, 0.01, 0.1, 1, 10],
    'classifier__kernel': ['rbf', 'linear', 'sigmoid']
}

grid = GridSearchCV(lasso_svm_pipeline, param_grid, refit=True, cv=3, verbose=0)

grid.fit(X_train, y_train)

best_params = grid.best_params_
best_score = grid.best_score_

print(f"Best parameters: {best_params}")
print(f"Best cross-validation score: {best_score}")

y_pred = grid.predict(X_test)
```

```
acc_train = accuracy_score(y_train, grid.predict(X_train))
acc_test = accuracy_score(y_test, y_pred)

print(f'Lasso_SVM -- Train Accuracy: {acc_train:.4f}, Test Accuracy: {acc_test:.4f}\n')
print('Classification Report for Lasso_SVM:')
print(classification_report(y_test, y_pred, zero_division=0))

disp = ConfusionMatrixDisplay.from_estimator(grid, X_test, y_test, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Lasso_SVM')
plt.show()

disp = RocCurveDisplay.from_estimator(grid, X_test, y_test, name='Lasso_SVM')
plt.title('AUC-ROC Curve for Lasso_SVM')
plt.plot([0, 1], [0, 1], linestyle="--", label='Random 50:50')
plt.legend()
plt.show()
```

```
Best parameters: {'classifier__C': 0.01, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf', 'feature_selection_
_estimator__alpha': 0.0001}
Best cross-validation score: 0.6883754342420724
Lasso_SVM -- Train Accuracy: 0.6884, Test Accuracy: 0.6099

Classification Report for Lasso_SVM:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       275
           1       0.61      1.00      0.76       430

    accuracy                           0.61       705
   macro avg       0.30      0.50      0.38       705
weighted avg       0.37      0.61      0.46       705
```


Confusion Matrix for Lasso_SVM
0
275
0
430
True label
0
1
Predicted label
400
350
300
250
200
150
100
50
0



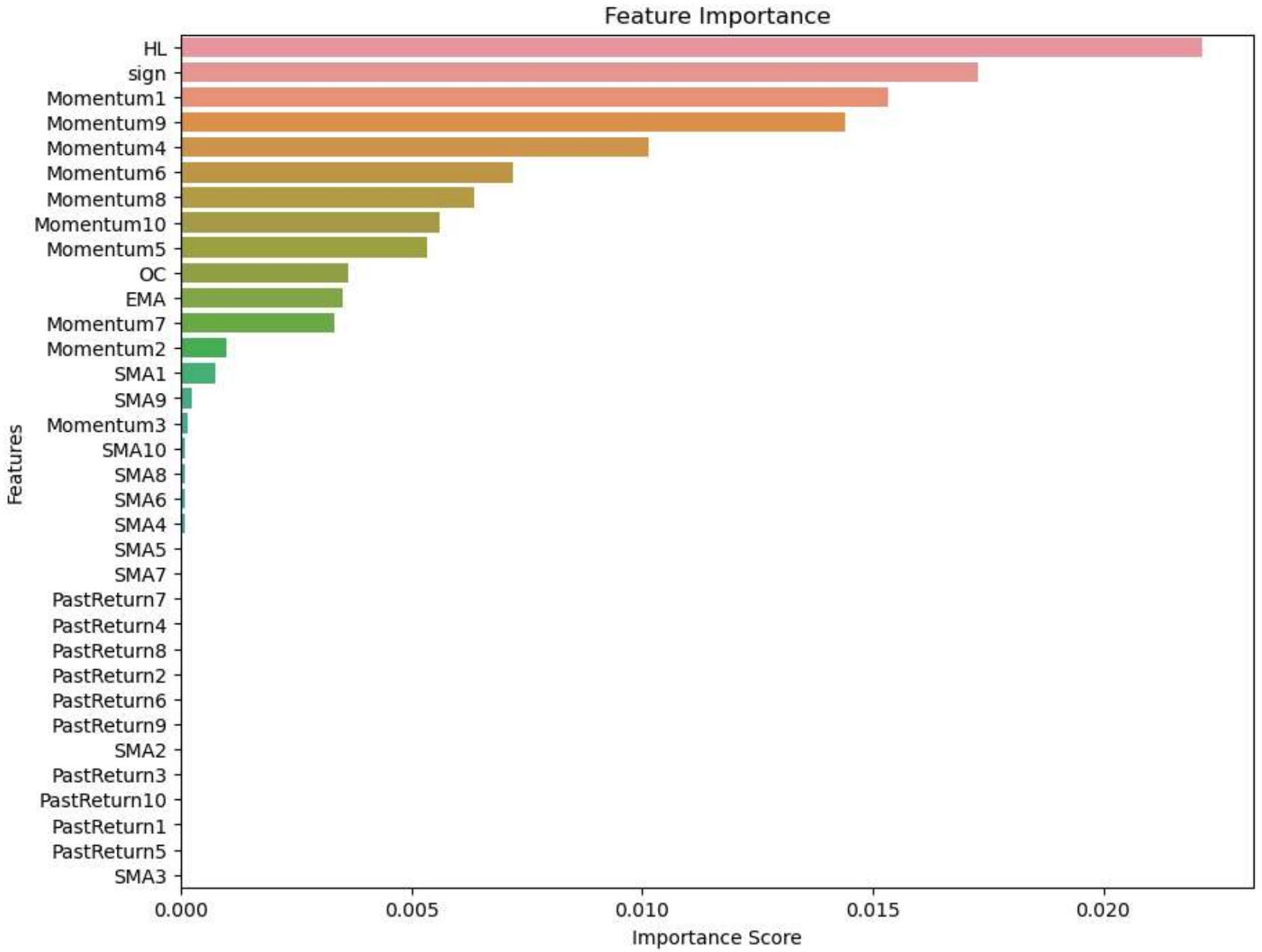
When using GridSearchCV so that we can find the best parameters for hyperparameter tuning, we've found that we should be using 'classifier__C': 0.01, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf', 'feature_selection__estimator__alpha': 0.0001'. There's a strong improvement of test accuracy scoring that's increased by ~0.1 while also seeing increases in our AUC-ROC score as well. The issue with our tuned LASSO model lies in the confusion matrix.

When viewing our matrix data, all instances has been classified as the positive class which suggests overfitting. This also means that our model fails to differentiate between positive and negative classes. Even though our test accuracy has improved, this can be seen as misleading given the system has a bias towards the majority class. Adding to this, further investigations would be needed to balance the models precision and recall. Future enhancements to this model to make it better represented towards being used in a financial landscape would be adding different hyperparameter ranges, class weight adjustments or even additional feature selections to ensure the most relevant ones are being used. Also, given our model is choosing to use 'c=0.01', this indicates we're

using soft margin allowing for high misclassifications. In future testings, we could take away 0.01 from the options. For this paper, I'll leave it in our current model to demonstrate that while machine learning is powerful, we need to understand the outputs that our code is providing us.

Lastly, from Python Lab Gradient Boosting for Price Predictions by Kannan Singaravelu, we learned about feature importance. What this technique does is we will look at all our input features, and calculate a score for which ones the models thinks is the most important. A node in this instance can be calculated by taking the number of samples that reach the node and dividing by the total number of samples. The highest value (which is the most important feature) is calculated by the decrease in node impurity weighted by the probability of reaching that node. Applying this concept to our tuned model, we can see the below results:

In [ ]:
```
best_model = grid.best_estimator_

lasso = best_model.named_steps['feature_selection'].estimator_
lasso.fit(X_train, y_train)
feature_importances = pd.DataFrame({
    'Importance Score': np.abs(lasso.coef_),
    'Features': X_train.columns
}).sort_values(by='Importance Score', ascending=False)

fig, ax = plt.subplots(figsize=(10, 8))
sns.barplot(x='Importance Score', y='Features', data=feature_importances)
ax.set_title('Feature Importance')
plt.show()
```



As we can see from our feature importance, given our tuned LASSO regression model, it highlights 'HL', 'sign' and various momentums being the top influential features. Momentum can reflect investor sentiment and ongoing market trends as it's

calculated on the difference in adjusted closing prices over different lag periods. In many ways, it shows the strength of the stock price movement over time. While these have higher scores indicating a strong impact on the predictions models, we can see all 'PastReturn' features play almost no role in determining our predictive outcome. Unlike momentum, past returns capture the historical returns over time and is based on performance. Given our model is not putting weight onto this feature, this suggests that historical percentage returns are less predictive of future price movements. Given we're using SVC which is used to determine 'up' or 'down' movements, this would make sense as past returns does not capture directional strength and trend to the effectiveness as momentum does.