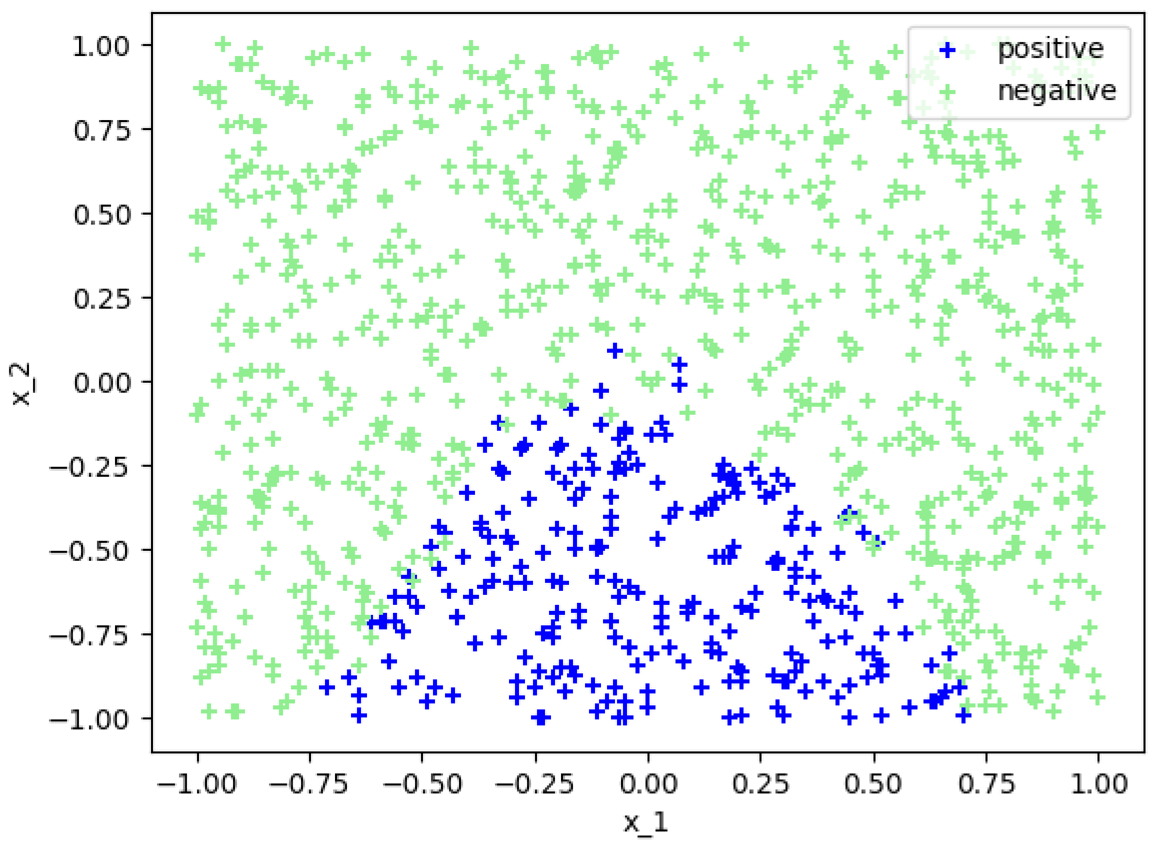
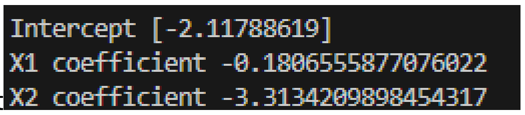
1) # id:25--50—25 : this is the data code I was told to append to the start of the report

   a) Question A

      i) I did X[y == 1] and X[y == -1] to separate the positive and negative values from the dataI then used matplotlib.pyplot to scatter the positive and negative values making the positive ones blue and the negative ones green. The data seems to follow a curved decision boundary

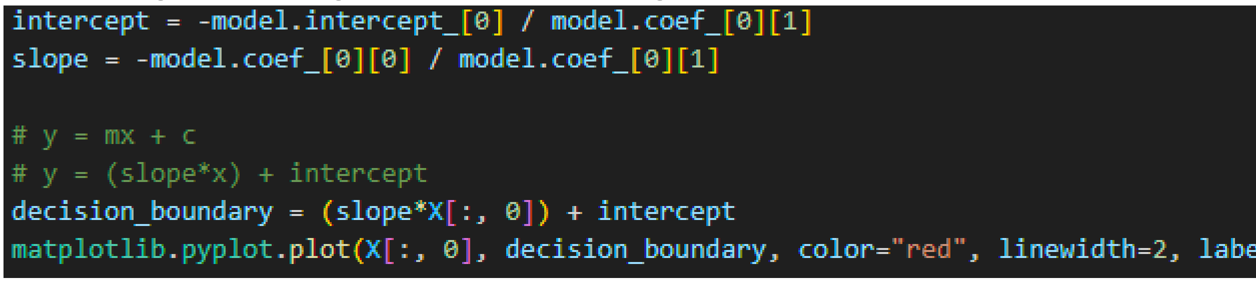


      ii) Using sklearn I created a logisticRegression model for the using X(thefeatures/inputs) and Y(the target/outputs) and printed the Intercept, X1 coefficient and X2 coefficient 

```
Intercept [-2.11788619]
X1 coefficient -0.1806555877076022
X2 coefficient -3.3134209898454317
```

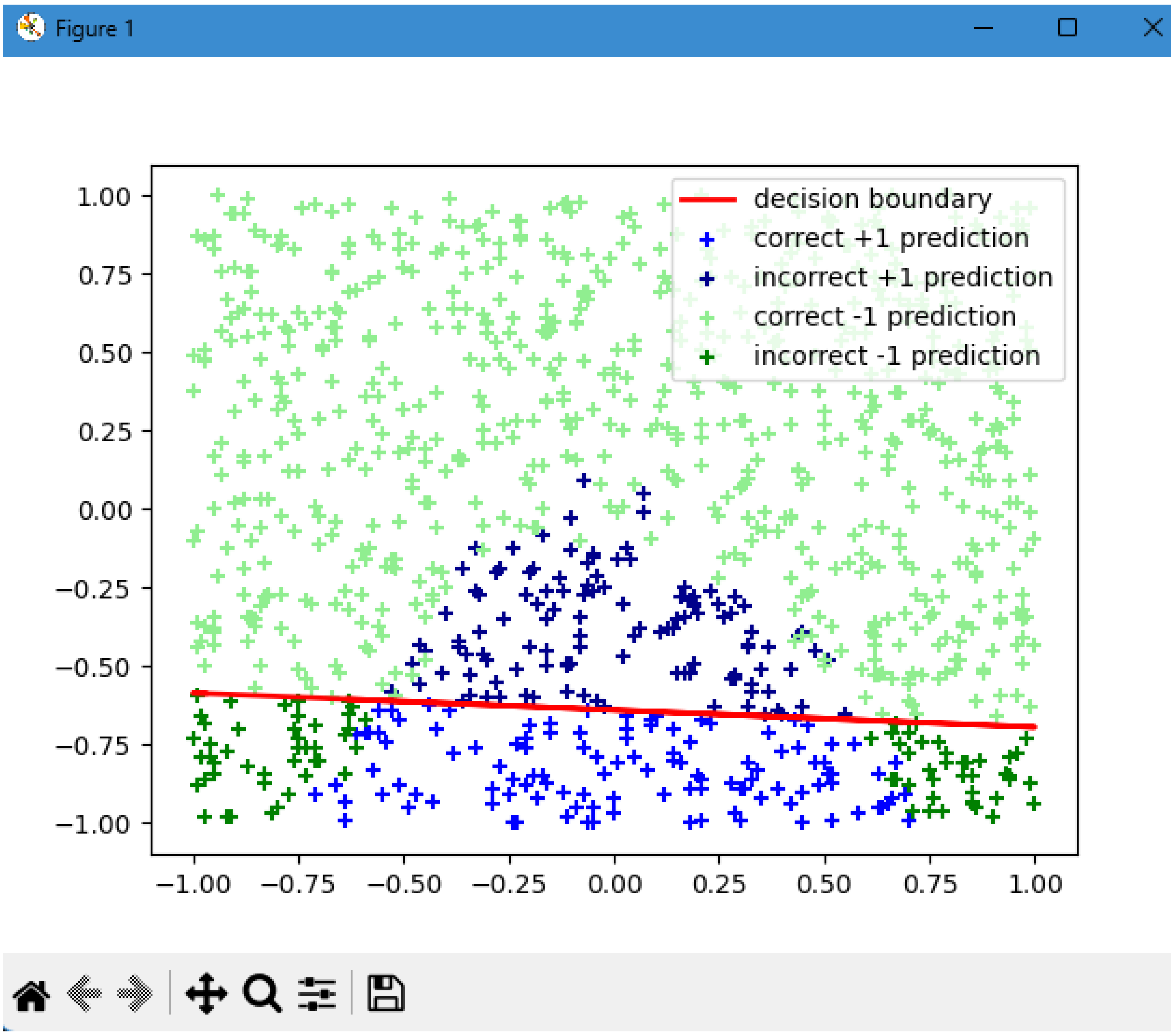
. The X1 coefficient is very small, showing that it currently plays very little impact in the predictions. The X2 coefficient is very large, showing that the second feature is much more impactful in the result. The intercept lies in the middle, showing that the X2 coefficient can

      iii) By recreating y=mx+c using the parameters of the regression model



```
intercept = -model.intercept_[0] / model.coef_[0][1]
slope = -model.coef_[0][0] / model.coef_[0][1]

# y = mx + c
# y = (slope*x) + intercept
decision_boundary = (slope*X[:, 0]) + intercept
matplotlib.pyplot.plot(X[:, 0], decision_boundary, color="red", linewidth=2, labe
```

. I was able to create the decision_boundary and plot it on the graph in red

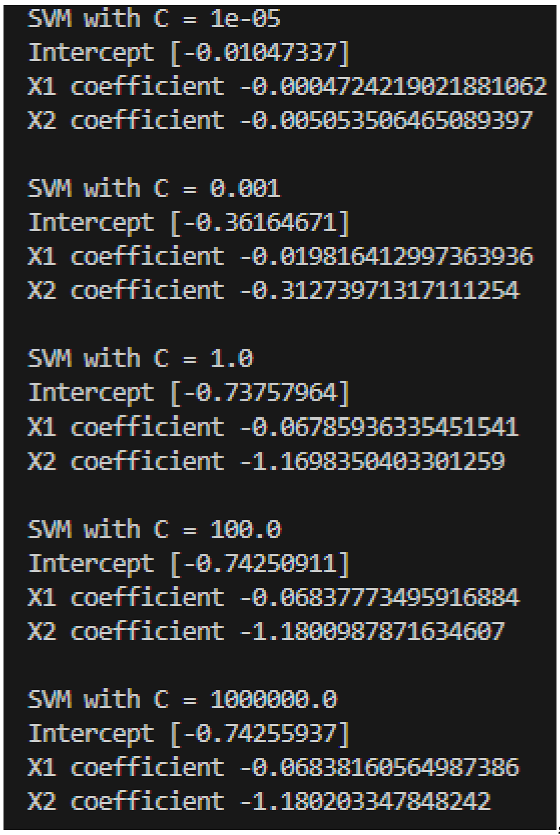


I then created 4 categories of points depending on the result of the training data and the prediction of the model and colored them accordingly, this completely aligned with the decision boundary, thus proving one another. From the decision boundary we can notice a few things that align with the parameters we printed earlier. The X1 coefficient being so small compared to the X2 coefficient causes a very small slope of the line.

iv) The predictors and training data are greatly misaligned, this is very obviously because the data seems to follow a nonlinear curve while the regression model is capable of a linear regression

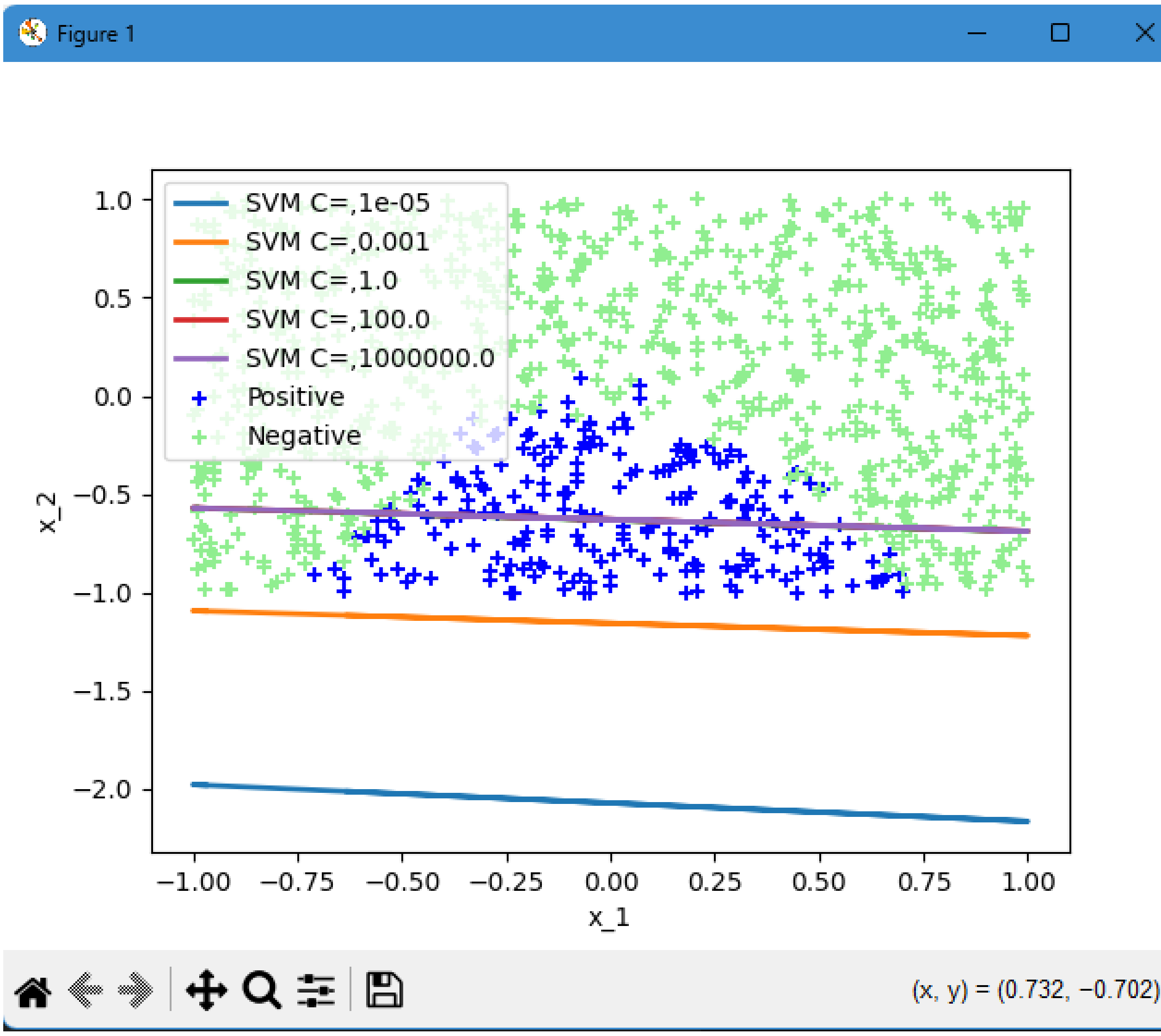
b) Using SVM classifiers

i) I created code to loop through a list of c values and put them all onto the graph

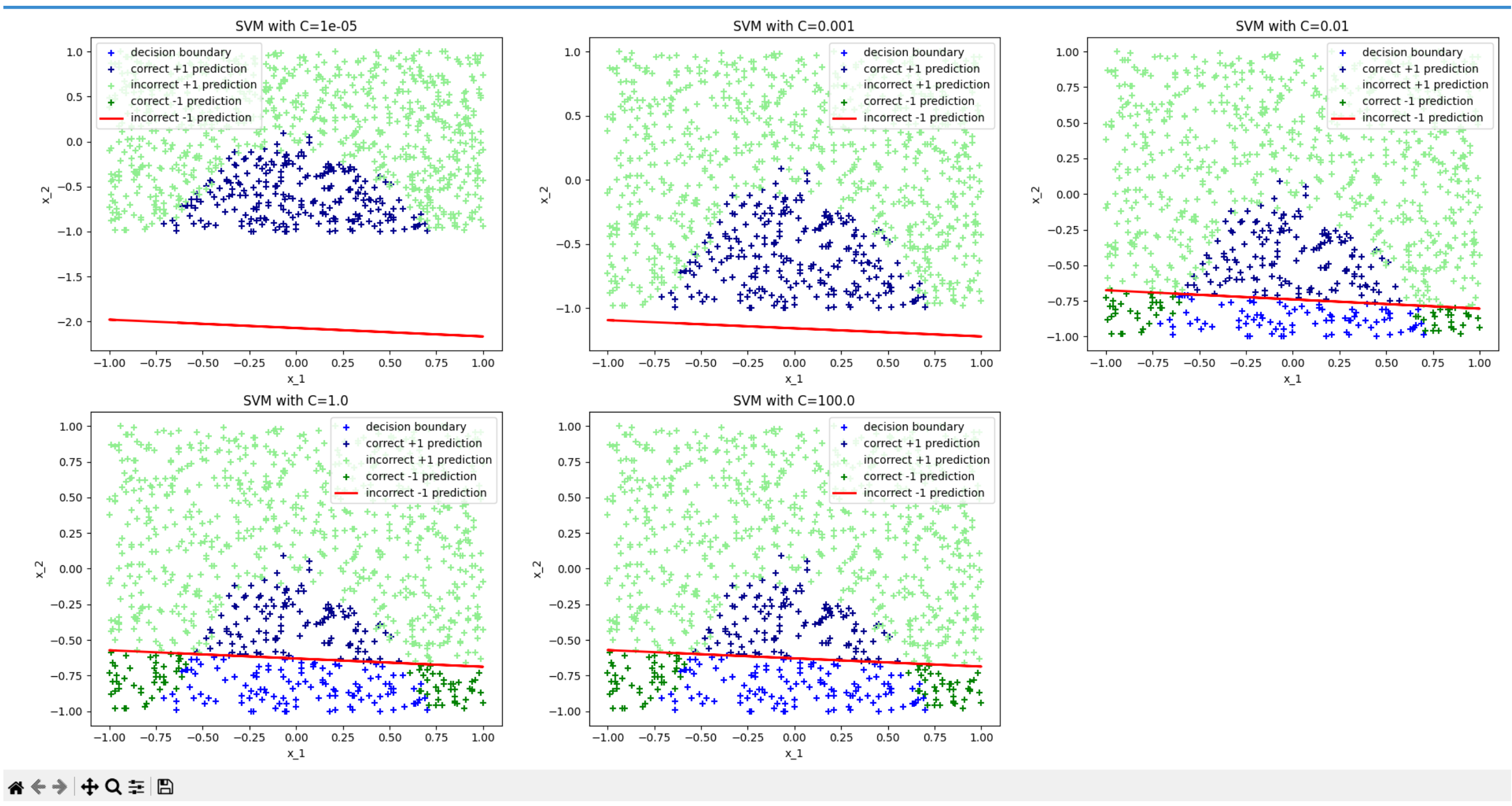


```
SVM with C = 1e-05
Intercept [-0.01047337]
X1 coefficient -0.0004724219021881062
X2 coefficient -0.005053506465089397

SVM with C = 0.001
Intercept [-0.36164671]
X1 coefficient -0.019816412997363936
X2 coefficient -0.31273971317111254

SVM with C = 1.0
Intercept [-0.73757964]
X1 coefficient -0.06785936335451541
X2 coefficient -1.1698350403301259

SVM with C = 100.0
Intercept [-0.74250911]
X1 coefficient -0.06837773495916884
X2 coefficient -1.1800987871634607

SVM with C = 1000000.0
Intercept [-0.74255937]
X1 coefficient -0.06838160564987386
X2 coefficient -1.180203347848242
```

, these values were returned, the biggest thing to notice is how the values don’t change much with higher values of C, as C approaches 0, it causes the intercept and both coefficients to also approach 0, but as it rises far above 1 it isnt influenced much. This can also be clearly seen on the graph.



This graph demonstrates how smaller numbers of C get increasingly lower down on the graph, while the larger values of C produce decision bounddaries that are almost inseperable. The smaller C allows more entries to qualify as “negative” thus lowering the decision much lower down

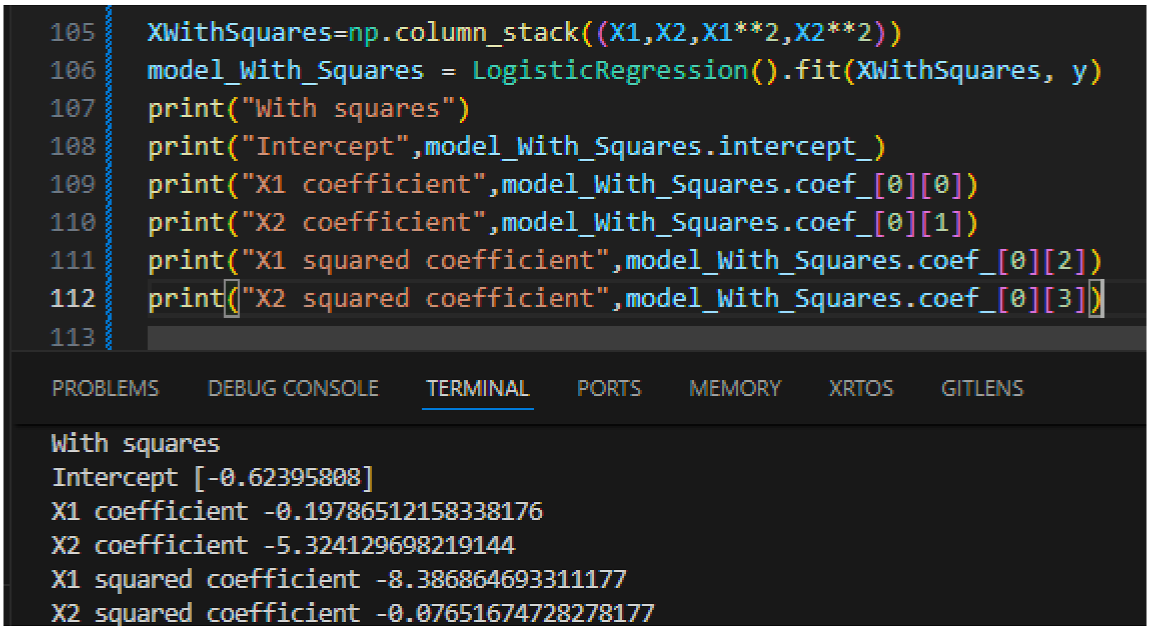


ii) Using recycled code from the previous prediction vs data comparison I was able to make 5 separate charts for each value of C. The chart clearly shows the same information that was shown when toghether on the same graph, where lowering C causes the margin to lower and thus allowing more entries to be predicted as positive. On this instance I added C=0.01 as it demonstrates the progression of the bar lowering, where it is allowing many more entries than C=1 but without allowing all of them

iii) A greater C means a lower margin, which ensures that the model is as accurate as possible for the data given, and thus a lower C value would result in a larger margin, which would become less accurate. Usually a high C value would run the risk of overfitting a model to a set of data. I believe in this instance that is not a concern as overfitting a liner model is very unlikely.

iv) While the intercept and coefficients of the highC-SVM models do differ from the logistic regression they are just scaled down , the predictions are identical and the decision_boundary is the same between the 2 once passed through the formula. On the other hand the lowC-SVM models are greatly different from the logistic regression, even outside of direct scaling.

c) Square of each feature

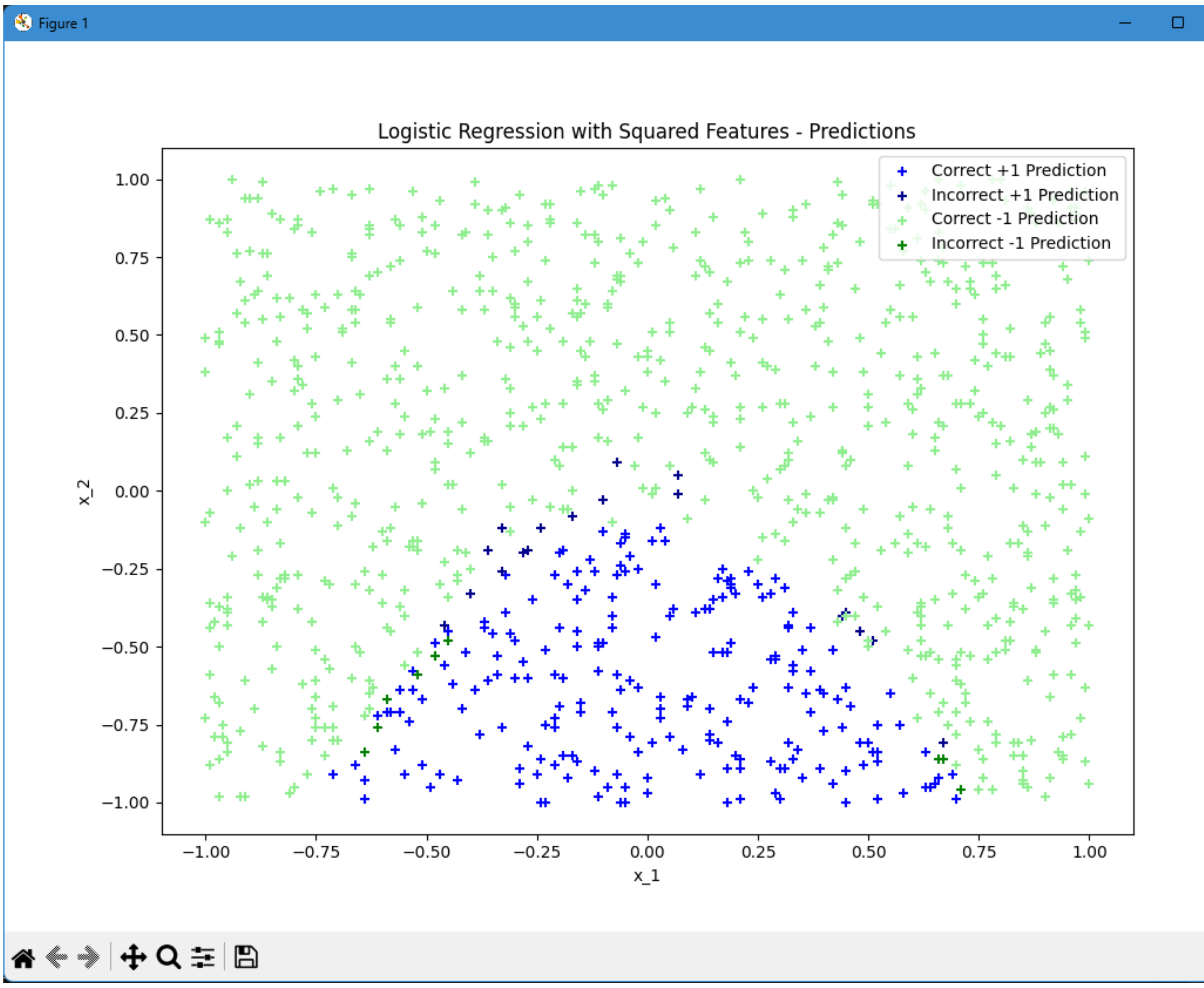


```python
XWithSquares=np.column_stack((X1,X2,X1**2,X2**2))
model_With_Squares = LogisticRegression().fit(XWithSquares, y)
print("With squares")
print("Intercept",model_With_Squares.intercept_)
print("X1 coefficient",model_With_Squares.coef_[0][0])
print("X2 coefficient",model_With_Squares.coef_[0][1])
print("X1 squared coefficient",model_With_Squares.coef_[0][2])
print("X2 squared coefficient",model_With_Squares.coef_[0][3])
```

```
With squares
Intercept [-0.62395808]
X1 coefficient -0.19786512158338176
X2 coefficient -5.324129698219144
X1 squared coefficient -8.386864693311177
X2 squared coefficient -0.07651674728278177
```

i) By recreating the data with the squared values we can see that the X1 squared coefficient skyrockets, the Intercept sinks and the X2 coefficient increases(downward) significantly. This shows that the graph should really be based on the value of X2 and the squared value of X1 almost exclusively

ii) Again recycling much of the code, I created the same system of colors for correct/incorrect positive/negative values, and finally acquired this table



, with only a handful of incorrect predictions(many of which visually seem to be outliers) we can see that this graph is a significantly better fit for the data than both the logistic regression and the SVM models. This demonstrates that the model always shouldve a nonlinear model in order to achieve better results.

iii) This classifier is significantly better than a baseline predictor. By comparing the length of the correct plotList vs the incorrect plotList I found that it had an accuracy of 97.3% accuracy(971 right out of 999). Compared to a baseline predictor (accuracy found by comparing the length of y==1 and y==-1) would have an accuracy of 76.6% (766 right out of 999), this shows that the model makes about 10 times less errors than a baseline predictor