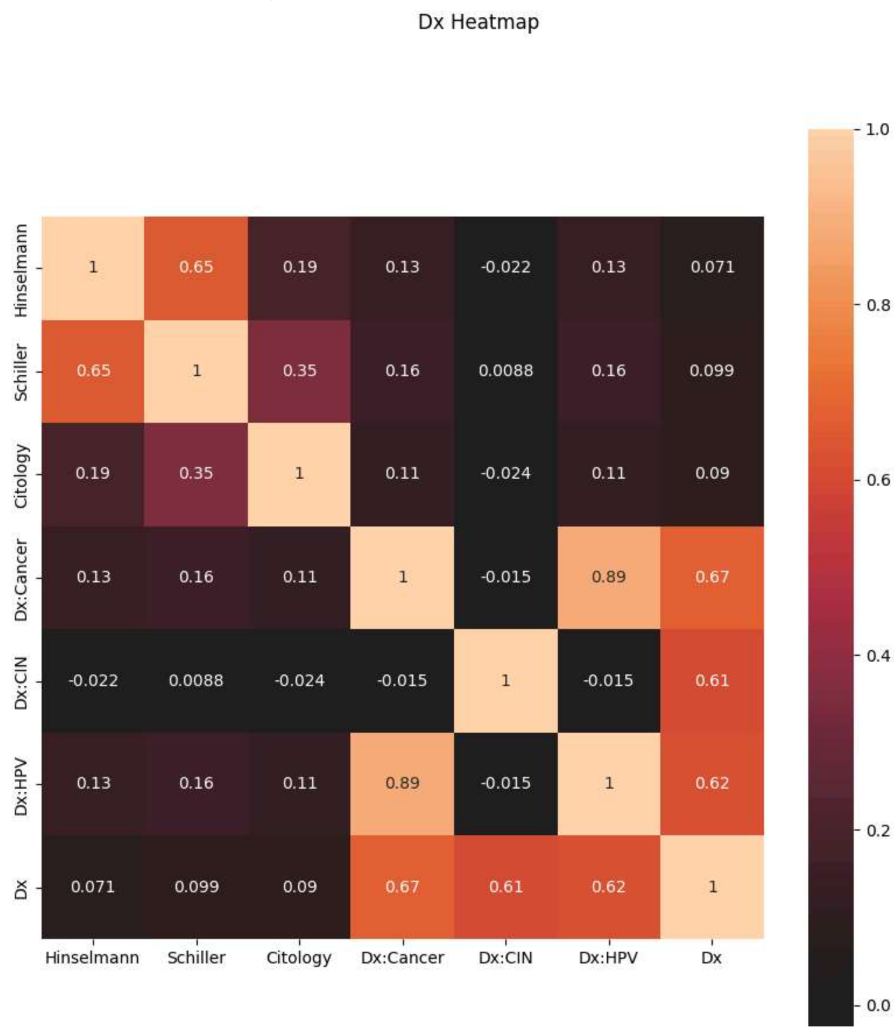
```
plt.figure(figsize=(10,10))
sns.heatmap(
    pd.concat([target_variables, c_dx], axis = 1).corr(),
    annot=True, center=0, square=True
)
plt.suptitle('Dx Heatmap')
```

```
Text(0.5, 0.98, 'Dx Heatmap')
```



Logistic Regression

```
# creating the X and Y arrays
X = cervical_cancer.drop(['Hinselmann', 'Schiller', 'Citology', 'Biopsy'], axis=1)
y = cervical_cancer['Hinselmann'] # Only Hinselmann would be used as dependent variable
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
X_train.shape, X_test.shape
```

```
((668, 32), (167, 32))
```

```
# Performing Feature Scaling
cols = X_train.columns
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns = [cols])
X_test = pd.DataFrame(X_test, columns = [cols])
X_train.describe()
```

	Age	Number_of_sexual_partners	First_sexual_intercourse	Num_of_pregnanci
count	668.000000	668.000000	668.000000	668.0000
mean	0.209966	0.059991	0.318293	0.2256
std	0.123068	0.065075	0.125395	0.1355
min	0.000000	0.000000	0.000000	0.0000
25%	0.121212	0.037037	0.227273	0.1000
50%	0.196970	0.037037	0.318182	0.2000
75%	0.287879	0.074074	0.363636	0.3000
max	1.000000	1.000000	1.000000	1.0000

8 rows × 32 columns

```
# Training a Logistic Regression Model
logreg = LogisticRegression(solver='liblinear', random_state=0)
logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

```
# Predicting next results for TEST variables
y_pred_test = logreg.predict(X_test)
y_pred_test
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# Getting probability of getting 0
logreg.predict_proba(X_test)[:,0]
# it can be noticed that all the decimals are >0.9,
# which means that there is a very high likelihood of getting 0
```

```
array([0.96499774, 0.96812193, 0.96178213, 0.955815  , 0.90170716,
       0.97202728, 0.95733515, 0.97822406, 0.96974261, 0.95380856,
       0.78285408, 0.97282591, 0.96213718, 0.97139767, 0.95349241,
       0.95400283, 0.9617409 , 0.96660061, 0.9693478 , 0.95215969,
       0.93983049, 0.93778046, 0.96555495, 0.97014354, 0.97156259,
       0.94923841, 0.968736  , 0.95877157, 0.97493375, 0.96534649,
       0.92624292, 0.96636051, 0.95537433, 0.93973728, 0.9219655 ,
       0.96566973, 0.97117451, 0.95803919, 0.97901675, 0.95297406,
       0.95129823, 0.96199643, 0.95450791, 0.95959901, 0.94785386,
       0.95687697, 0.90279063, 0.95310214, 0.97246285, 0.96201477,
       0.96626321, 0.96506751, 0.92266776, 0.96539549, 0.95784011,
       0.94706479, 0.96207725, 0.967493  , 0.9710381 , 0.9535819 ,
       0.97463454, 0.9270629 , 0.96905479, 0.95194204, 0.95569159,
       0.96546064, 0.9485952 , 0.97270511, 0.96905738, 0.96232778,
       0.96933687, 0.96894982, 0.94925526, 0.96278429, 0.97107501,
       0.97406803, 0.96405313, 0.9344687 , 0.96260272, 0.96674386,
       0.96317831, 0.96494496, 0.93562235, 0.96705778, 0.94672523,
       0.9309969 , 0.9561994 , 0.95359612, 0.95276384, 0.95626473,
       0.91935853, 0.94675299, 0.93876364, 0.97230308, 0.9050622 ,
       0.96434028, 0.96673148, 0.9708371 , 0.98028568, 0.96705872,
       0.96820082, 0.94534362, 0.93357112, 0.94702052, 0.96626403,
       0.95023002, 0.96844403, 0.95916565, 0.96470221, 0.95153548,
       0.97405584, 0.95226863, 0.93855023, 0.95131513, 0.97053531,
       0.97195262, 0.9635906 , 0.94814468, 0.96478433, 0.9718506 ,
       0.97827541, 0.95440237, 0.96749883, 0.95128418, 0.96769256,
       0.9628513 , 0.95814777, 0.95626354, 0.96707181, 0.96505425,
       0.9689722 , 0.961836  , 0.95982967, 0.9626378 , 0.97635852,
       0.96708416, 0.96234493, 0.96759091, 0.97757895, 0.96150864,
       0.96586402, 0.96870273, 0.97397919, 0.94228874, 0.98551671,
       0.95661954, 0.96679722, 0.95520834, 0.97436977, 0.96832497,
       0.94521092, 0.91134901, 0.9717929 , 0.98072915, 0.97084674,
       0.97996196, 0.97007555, 0.95842026, 0.96092729, 0.94133172,
       0.97043439, 0.95268401, 0.96732373, 0.96127022, 0.95228539,
       0.96770342, 0.96441616])
```

```
# getting probabilty of getting 1
logreg.predict_proba(X_test)[:,1]
# it can be noticed that most of the decimals are < 0.09,
# which means that there is a very low likelihood of getting 1
```

```
array([0.03500226, 0.03187807, 0.03821787, 0.044185  , 0.09829284,
       0.02797272, 0.04266485, 0.02177594, 0.03025739, 0.04619144,
       0.21714592, 0.02717409, 0.03786282, 0.02860233, 0.04650759,
       0.04599717, 0.0382591 , 0.03339939, 0.0306522 , 0.04784031,
       0.06016951, 0.06221954, 0.03444505, 0.02985646, 0.02843741,
       0.05076159, 0.031264  , 0.04122843, 0.02506625, 0.03465351,
       0.07375708, 0.03363949, 0.04462567, 0.06026272, 0.0780345 ,
       0.03433027, 0.02882549, 0.04196081, 0.02098325, 0.04702594,
       0.04870177, 0.03800357, 0.04549209, 0.04040099, 0.05214614,
       0.04312303, 0.09720937, 0.04689786, 0.02753715, 0.03798523,
       0.03373679, 0.03493249, 0.07733224, 0.03460451, 0.04215989,
       0.05293521, 0.03792275, 0.032507  , 0.0289619 , 0.0464181 ,
       0.02536546, 0.0729371 , 0.03094521, 0.04805796, 0.04430841,
       0.03453936, 0.0514048 , 0.02729489, 0.03094262, 0.03767222,
       0.03066313, 0.03105018, 0.05074474, 0.03721571, 0.02892499,
       0.02593197, 0.03594687, 0.0655313 , 0.03739728, 0.03325614,
       0.03682169, 0.03505504, 0.06437765, 0.03294222, 0.05327477,
```

```
       0.0690031 , 0.0438006 , 0.04640388, 0.04723616, 0.04373527,
       0.08064147, 0.05324701, 0.06123636, 0.02769692, 0.0949378 ,
       0.03565972, 0.03326852, 0.0291629 , 0.01971432, 0.03294128,
       0.03179918, 0.05465638, 0.06642888, 0.05297948, 0.03373597,
       0.04976998, 0.03155597, 0.04083435, 0.03529779, 0.04846452,
       0.02594416, 0.04773137, 0.06144977, 0.04868487, 0.02946469,
       0.02804738, 0.0364094 , 0.05185532, 0.03521567, 0.0281494 ,
       0.02172459, 0.04559763, 0.03250117, 0.04871582, 0.03230744,
       0.0371487 , 0.04185223, 0.04373646, 0.03292819, 0.03494575,
       0.0310278 , 0.038164  , 0.04017033, 0.0373622 , 0.02364148,
       0.03291584, 0.03765507, 0.03240909, 0.02242105, 0.03849136,
       0.03413598, 0.03129727, 0.02602081, 0.05771126, 0.01448329,
       0.04338046, 0.03320278, 0.04479166, 0.02563023, 0.03167503,
       0.05478908, 0.08865099, 0.0282071 , 0.01927085, 0.02915326,
       0.02003804, 0.02992445, 0.04157974, 0.03907271, 0.05866828,
       0.02956561, 0.04731599, 0.03267627, 0.03872978, 0.04771461,
       0.03229658, 0.03558384])
```

```
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.9581
```

```
# Predicting next results for TRAIN variables
y_pred_train = logreg.predict(X_train)
y_pred_train
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0])
```

```
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_pred_train)))
```

```
Model accuracy score: 0.9581
```

```
# Checking for overfitting and underfitting
# getting set scores

print('Training set score: {0:0.4f}'.format(logreg.score(X_train, y_train)))
print('Test set score: {0:0.4f}'.format(logreg.score(X_test, y_test)))
# Since the set score are the same, there is not overfitting
```

```
Training set score: 0.9581
Test set score: 0.9581
```

```
# increasing the C to 100
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)
logreg100.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
# It can be noticed that the set score became higher in the training set but lower on the test set
```

```
Training set score: 0.9626
Test set score: 0.9521
```

```
logreg001 = LogisticRegression(C=0.01, solver='liblinear', random_state=0)
logreg001.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=0.01, random_state=0, solver='liblinear')
```

```
print('Training set score: {:.4f}'.format(logreg001.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(logreg001.score(X_test, y_test)))
# It can be noticed that there is no difference between a C of 1 and 0.01
```

```
Training set score: 0.9581
Test set score: 0.9581
```

```
# Comparing model accuracy with null accuracy
y_test.value_counts()
```

```
Hinselmann
0    160
1      7
Name: count, dtype: int64
```

```
null_accuracy = (160/167)
print('Null accuracy score: {:.4f}'.format(null_accuracy))
# It can be seen that the null accuracy score is the same as the training set score
```

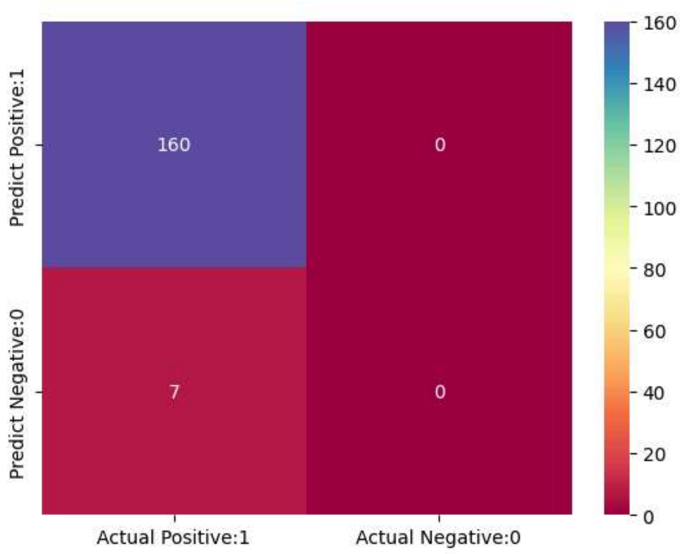
```
Null accuracy score: 0.9581
```

```
# Creating an displaying a confusion matrix
cm = confusion_matrix(y_test, y_pred_test)
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
# This shows that the model created 160 correct predictions and 7 incorrect predictions
```

```
Confusion matrix

 [[160   0]
 [  7   0]]

True Positives(TP) =  160

True Negatives(TN) =  0

False Positives(FP) =  0

False Negatives(FN) =  7
```

```
# Visualizing the confusion matrix
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1','Actual Negative:0'],
                         index=['Predict Positive:1','Predict Negative:0'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='Spectral')
```

```
<Axes: >
```




```
# Creates a classification report
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       160
           1       0.00      0.00      0.00         7

    accuracy                           0.96       167
   macro avg       0.48      0.50      0.49       167
weighted avg       0.92      0.96      0.94       167

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-c
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-c
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-c
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Gets the classification accuracy
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print('Classification accuracy: {:.4f}'.format(classification_accuracy))
```

```
Classification accuracy: 0.9581
```

```
# Gets classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error: {:.4f}'.format(classification_error))
# It can be seen that the classification accuracy is high while the error is low
```

```
Classification error: 0.0419
```

```
# Gets precision score
precision = TP / float(TP + FP)
print('Precision: {:.4f}'.format(precision))
# The result is a perfect precision score
```

```
Precision: 1.0000
```

```
# Gets Recall rate
recall = TP / float(TP + FN)
print('Recall or Sensitivity: {:.4f}'.format(recall))
```

```
Recall or Sensitivity: 0.9581
```

```
# Gets False Positive rate
false_positive_rate = FP / float(FP+TN)
print('False Positive: {:.4f}'.format(false_positive_rate))
# Error is encountered as FP is 0
```

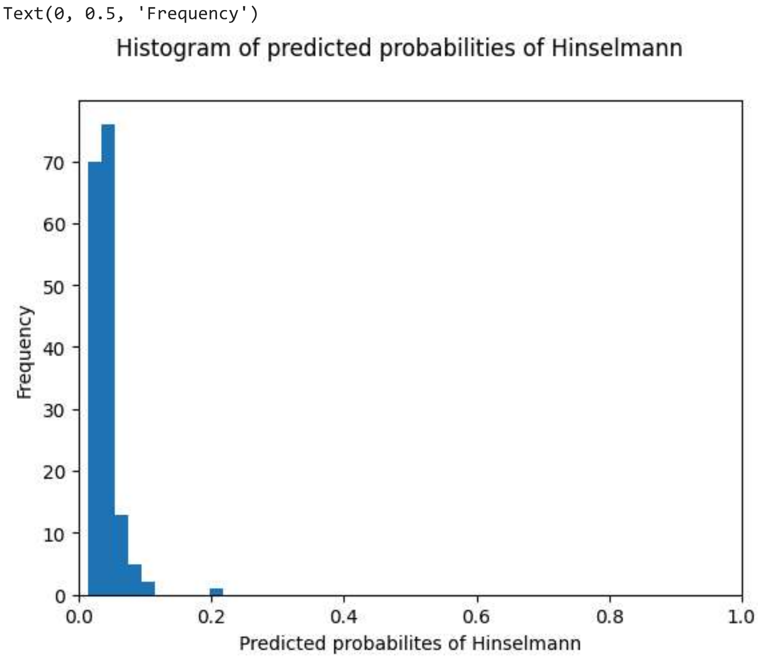
```
False Positive: nan
<ipython-input-118-648a66821cfc>:2: RuntimeWarning: invalid value encountered in divide
  false_positive_rate = FP / float(FP+TN)
```

```
# Gets Specificity rate
specificity = TN / (TN + FP)
print('Specificity: {:.4f}'.format(specificity))
# Error is encountered as TN is 0
```

```
Specificity: nan
<ipython-input-119-c484c783968f>:2: RuntimeWarning: invalid value encountered in scalar divide
  specificity = TN / (TN + FP)
```

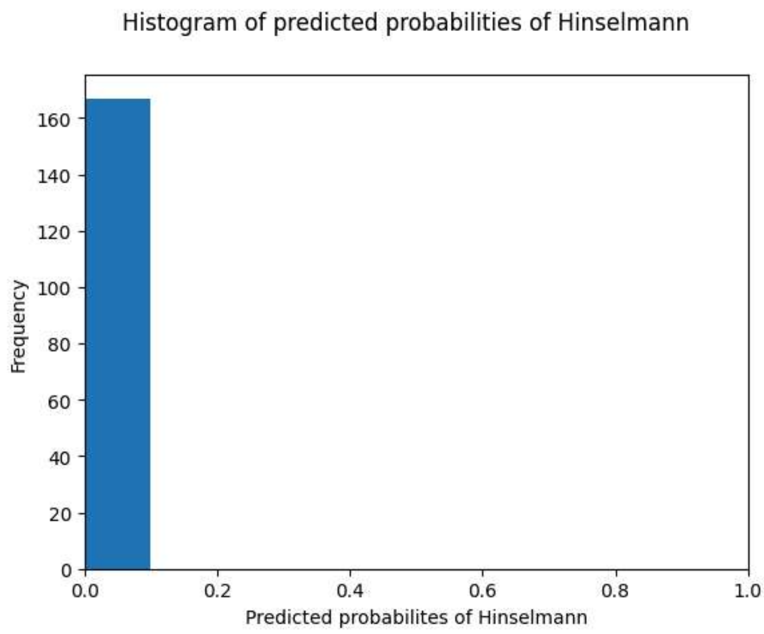
```
# Plots predicted probabilities to a histogram
y_pred1 = logreg.predict_proba(X_test)[:,1] # using the test variable
plt.hist(y_pred1, bins=10)
plt.suptitle('Histogram of predicted probabilities of Hinselmann')
plt.xlim(0,1)
plt.xlabel('Predicted probabilites of Hinselmann')
plt.ylabel('Frequency')
# The plot is highly skewed to 0,
# which means that the model predicts there a person won't have Hinselmann
```

```
Text(0, 0.5, 'Frequency')
```



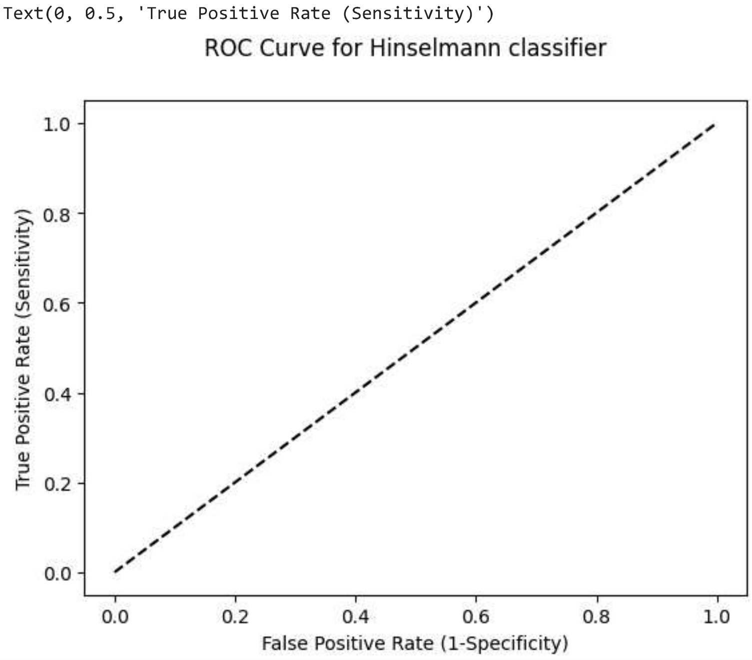

```
# Using 0.5 threshold
y_pred1 = y_pred1.reshape(-1,1)
y_pred2 = binarize(y_pred1, threshold = 0.5)
plt.hist(y_pred2, bins=10)
plt.suptitle('Histogram of predicted probabilities of Hinselmann')
plt.xlim(0,1)
plt.xlabel('Predicted probabilites of Hinselmann')
plt.ylabel('Frequency')
# With the threshold at 0.5, the plot becomes skewed to 0
```

```
Text(0, 0.5, 'Frequency')
```




```
# Creates an ROC Curve
fpr, tpr, thresholds = roc_curve(y_train, y_pred_train, pos_label = 'Yes') # Using train variables
plt.plot(fpr, tpr, linewidth = 2)
plt.plot([0,1], [0,1], 'k--')
plt.suptitle('ROC Curve for Hinselmann classifier')
plt.xlabel('False Positive Rate (1-Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
# It seems that there is an error as the false positive is 0
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_ranking.py:1029: UndefinedMetricWarning: No positive samples in y_true, true pc
  warnings.warn(
Text(0, 0.5, 'True Positive Rate (Sensitivity)')
```




```
# Gets the ROC AUC score
Cross_validated_ROC_AUC = cross_val_score(logreg, X_train, y_train, cv=5, scoring='roc_auc').mean()
print('Cross validated ROC AUC: {:.4f}'.format(Cross_validated_ROC_AUC))
# The result is not too high but is still higher than 0.5,
# therefore the model might not predict well
```

```
Cross validated ROC AUC: 0.6114
```

```
# Gets the 5-Fold Cross Validation score
scores = cross_val_score(logreg, X_train, y_train, cv=5, scoring='accuracy').mean()
print('Cross-validation score: {:.4f}'.format(scores))
# The result is higher than 0.9, which suggest high accuracy
```

```
Cross-validation score: 0.9581
```

```
# Using GridSearch CV
parameters = [{'penalty':['l1', 'l2']},
              {'C':[1,10,100,1000]}]
grid_search = GridSearchCV(estimator = logreg,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose = 0)
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

```
# Gets the best model
print('GridSearch CV best score: {:.4f}\n\n'.format(grid_search.best_score_))
print('Parameters that give the best results: \n\n',(grid_search.best_params_))
print('\n\nEstimator that was chosen by the search: \n\n',(grid_search.best_estimator_))
```

```
Cross-validation score: 0.9581


Parameters that give the best results:

 {'penalty': 'l1'}


Estimator that was chosen by the search:

 LogisticRegression(penalty='l1', random_state=0, solver='liblinear')
```

```
# Gets the GridSearch CV score on test set
print('GridSearch CV score on test set: {:.4f}\n\n'.format(grid_search.score(X_test, y_test)))
# The results are the same as with the normal model
```

```
GridSearch CV score on test set: 0.9581
```