Import Libraries

In [83]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import accuracy_score

%matplotlib inline
```

Import CSV File

In [84]:
```
credit_card_data = pd.read_csv('creditCard.csv')
```

In [85]:
```
credit_card_data.head()
```

Out[85]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0

5 rows × 31 columns

In [86]:
```
credit_card_data.describe
```

Out[86]:
```
<bound method NDFrame.describe of            Time         V1         V2
V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278
...          ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
...          ...       ...       ...       ...       ...       ...     ...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

       Class
0          0
1          0
2          0
3          0
4          0
...      ...
284802     0
284803     0
284804     0
284805     0
284806     0

[284807 rows x 31 columns]>
```

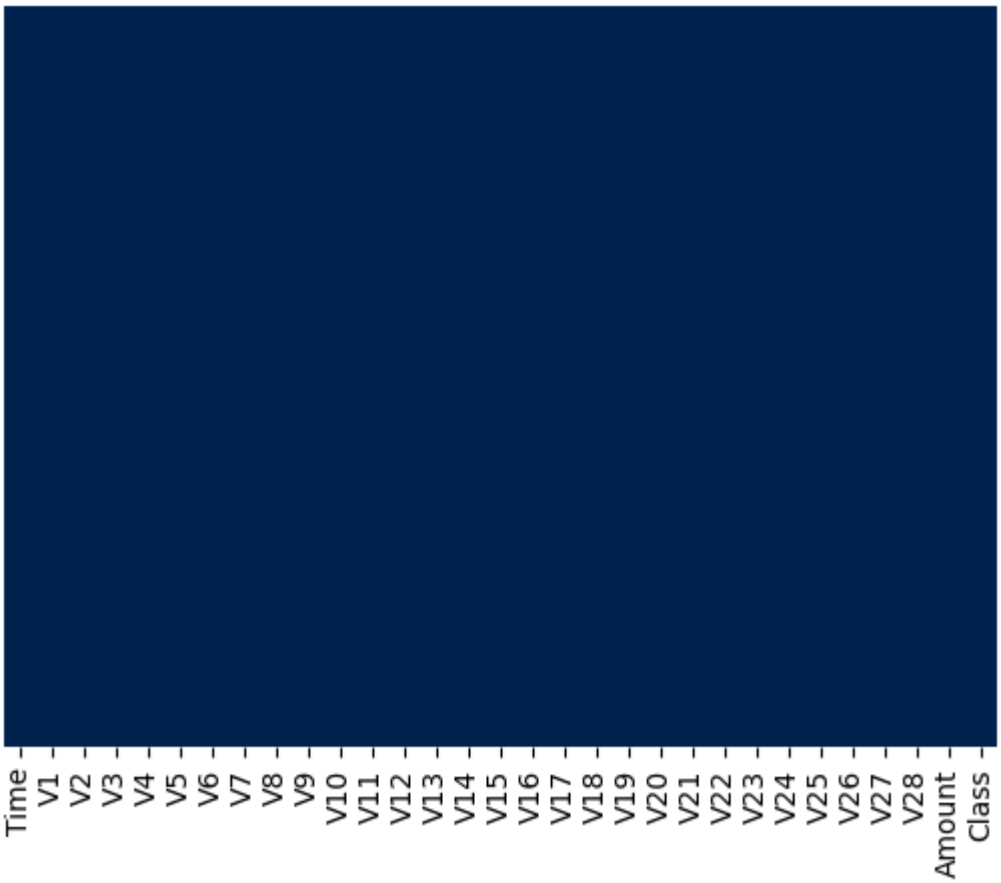
In [87]:
```
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Data Analysis

In [88]:
```
sns.heatmap(credit_card_data.isnull(),yticklabels=False,cbar=False,cmap='civid
```

Out[88]:
```
<Axes: >
```

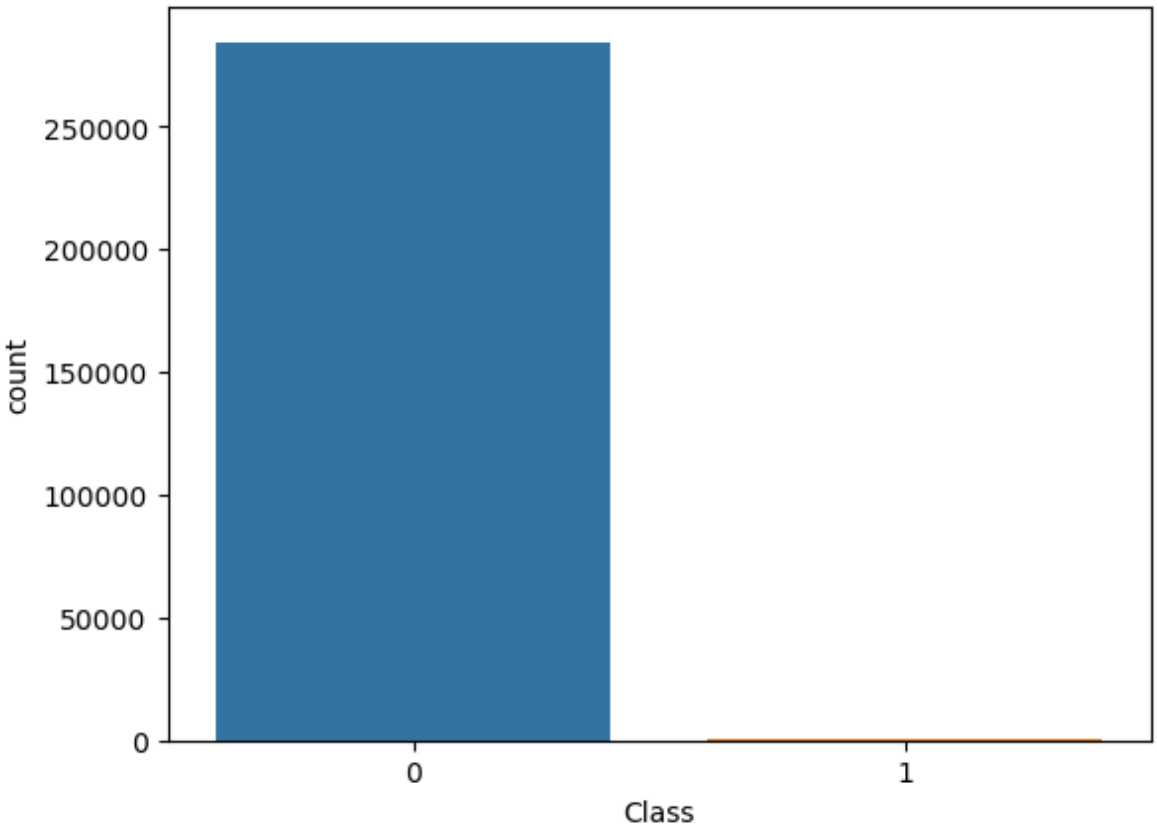


In [89]:
```
credit_card_data.columns
```

Out[89]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [90]:
```
sns.countplot(x='Class', data =credit_card_data)
```

Out[90]:
```
<Axes: xlabel='Class', ylabel='count'>
```



In [91]:
```
credit_card_data.isnull().sum()
```

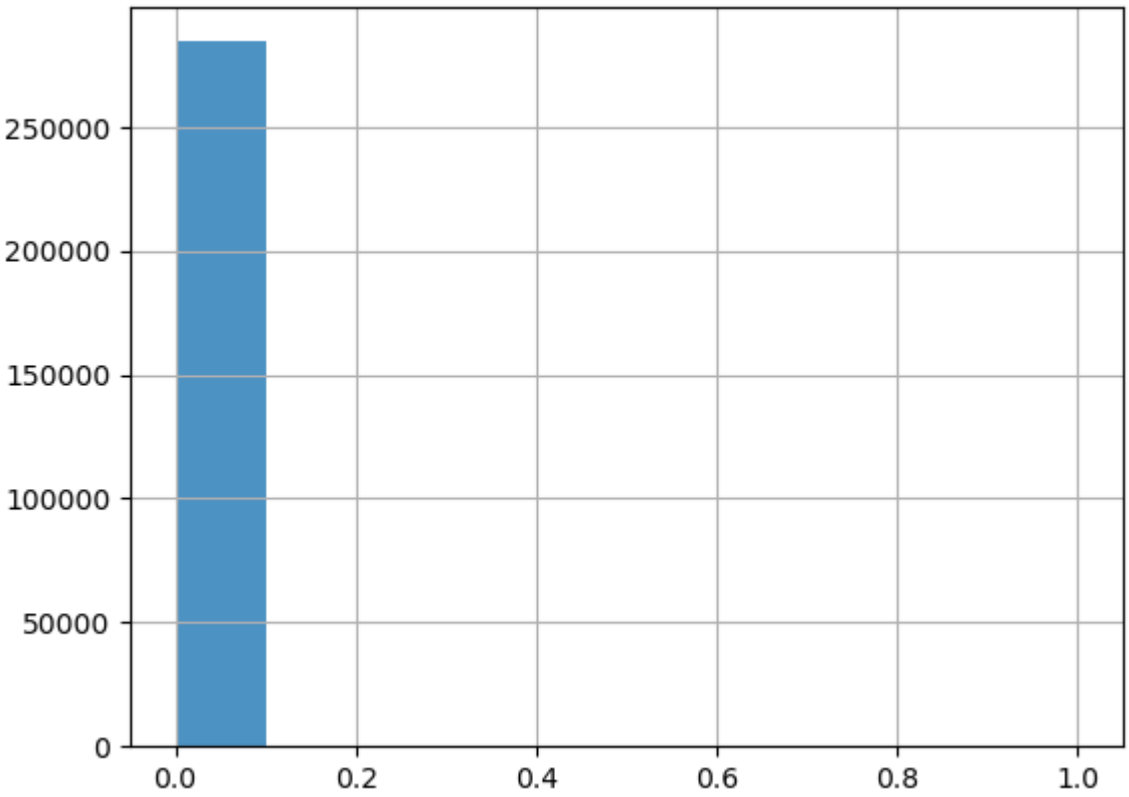
Out[91]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [92]:
```
credit_card_data['Class'].value_counts()
```

Out[92]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [93]:
```
credit_card_data['Class'].hist(bins=10, alpha=0.8)
```

Out[93]: <Axes: >



The Dataset is Highly Imblanced

0--> Normal Transaction

1--> Fraud Transaction

In [94]:
```
normal_transaction = credit_card_data[credit_card_data.Class ==0 ]
fraud_transaction = credit_card_data[credit_card_data.Class ==1 ]
```

In [95]:
```
normal_transaction.describe()
```

Out[95]:

	Time	V1	V2	V3	V4	V
count	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.0000
mean	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.0054
std	47484.015786	1.929814	1.636146	1.459429	1.399333	1.3569
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.7433
25%	54230.000000	-0.917544	-0.599473	-0.884541	-0.850077	-0.6893
50%	84711.000000	0.020023	0.064070	0.182158	-0.022405	-0.0534
75%	139333.000000	1.316218	0.800446	1.028372	0.737624	0.6121
max	172792.000000	2.454930	18.902453	9.382558	16.875344	34.8016

8 rows × 31 columns

In [96]:
```
fraud_transaction.describe()
```

Out[96]:

	Time	V1	V2	V3	V4	V5	V6
count	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000
mean	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737
std	47835.365138	6.783687	4.291216	7.110937	2.873318	5.372468	1.858124
min	406.000000	-30.552380	-8.402154	-31.103685	-1.313275	-22.105532	-6.406267
25%	41241.500000	-6.036063	1.188226	-8.643489	2.373050	-4.792835	-2.501511
50%	75568.500000	-2.342497	2.717869	-5.075257	4.177147	-1.522962	-1.424616
75%	128483.000000	-0.419200	4.971257	-2.276185	6.348729	0.214562	-0.413216
max	170348.000000	2.132386	22.057729	2.250210	12.114672	11.095089	6.474115

8 rows × 31 columns

In [97]:
```
print(normal_transaction.shape)
print(fraud_transaction.shape)
```

```
(284315, 31)
(492, 31)
```

In [98]:
```
credit_card_data.groupby('Class').mean()
```

Out[98]:

Class	Time	V1	V2	V3	V4	V5	V6	V7
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

In [99]:
```
normal_transaction_sample = normal_transaction.sample(n=492)
normal_transaction_sample.head()
```

Out[99]:

	Time	V1	V2	V3	V4	V5	V6	V7	
100287	67474.0	0.693107	-1.685440	1.261139	-0.171384	-2.323730	-0.666678	-0.788288	-0.1
37827	39128.0	-1.319421	0.988224	0.804973	-0.603870	-0.551572	-0.550180	1.695602	-0.3
253977	156521.0	2.257111	-1.337892	-1.006524	-1.687687	-1.002215	-0.280004	-1.125956	-0.(
278075	168024.0	0.304924	-0.532798	-0.024353	-3.038937	-0.048588	-0.560653	-0.005788	-0.2
25027	33484.0	-1.199076	-1.245462	1.654279	-0.798890	-2.247533	0.027414	1.304699	-1.3

5 rows × 31 columns

In [100]:
```
new_data = pd.concat([normal_transaction_sample,fraud_transaction], axis=0)
```

In [101]:
```
new_data.head()
```

Out[101]:

	Time	V1	V2	V3	V4	V5	V6	V7	
100287	67474.0	0.693107	-1.685440	1.261139	-0.171384	-2.323730	-0.666678	-0.788288	-0.1
37827	39128.0	-1.319421	0.988224	0.804973	-0.603870	-0.551572	-0.550180	1.695602	-0.3
253977	156521.0	2.257111	-1.337892	-1.006524	-1.687687	-1.002215	-0.280004	-1.125956	-0.(
278075	168024.0	0.304924	-0.532798	-0.024353	-3.038937	-0.048588	-0.560653	-0.005788	-0.2
25027	33484.0	-1.199076	-1.245462	1.654279	-0.798890	-2.247533	0.027414	1.304699	-1.3

5 rows × 31 columns

In [102]:
```
new_data.tail()
```

Out[102]:

	Time	V1	V2	V3	V4	V5	V6	V7	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.69
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.24
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.21
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.05
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.06

5 rows × 31 columns

In [103]:
```
new_data['Class'].value_counts()
```

Out[103]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [104]:
```
X = new_data[['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']]
Y = new_data['Class']
```

Train and Test Split

In [105]:
```
from sklearn.model_selection import train_test_split
```

In [106]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size = .2, random_s
                                                 )
```

In [107]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Train and Predicting

In [108]:
```
from sklearn.linear_model import LogisticRegression
```

In [109]:
```
model = LogisticRegression()
```

In [110]:
```
model.fit(X_train,Y_train)
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[110]:
```
LogisticRegression()
```

In [120]:
```
train_predict = model.predict(X_train)
```

Evaluation

In [124]:
```
train_data_accuracy = accuracy_score(train_predict,Y_train)
print('Accuracy on Training Data = ',train_data_accuracy  )
```

```
Accuracy on Training Data =  0.9390088945362135
```

In [125]:
```
test_predict = model.predict(X_test)
test_data_accuracy = accuracy_score(test_predict, Y_test)
print('Accuracy on Test Data = ', test_data_accuracy  )
```

```
Accuracy on Test Data =  0.9390862944162437
```

In [126]:
```
from sklearn.metrics import classification_report
```

In [127]:
```
print(classification_report(Y_test, test_predict))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94        95
           1       0.98      0.90      0.94       102

    accuracy                           0.94       197
   macro avg       0.94      0.94      0.94       197
weighted avg       0.94      0.94      0.94       197
```

In [128]:
```
from sklearn.metrics import confusion_matrix
```

In [129]:
```
print(confusion_matrix(Y_test,test_predict))
```

```
[[93  2]
 [10 92]]
```

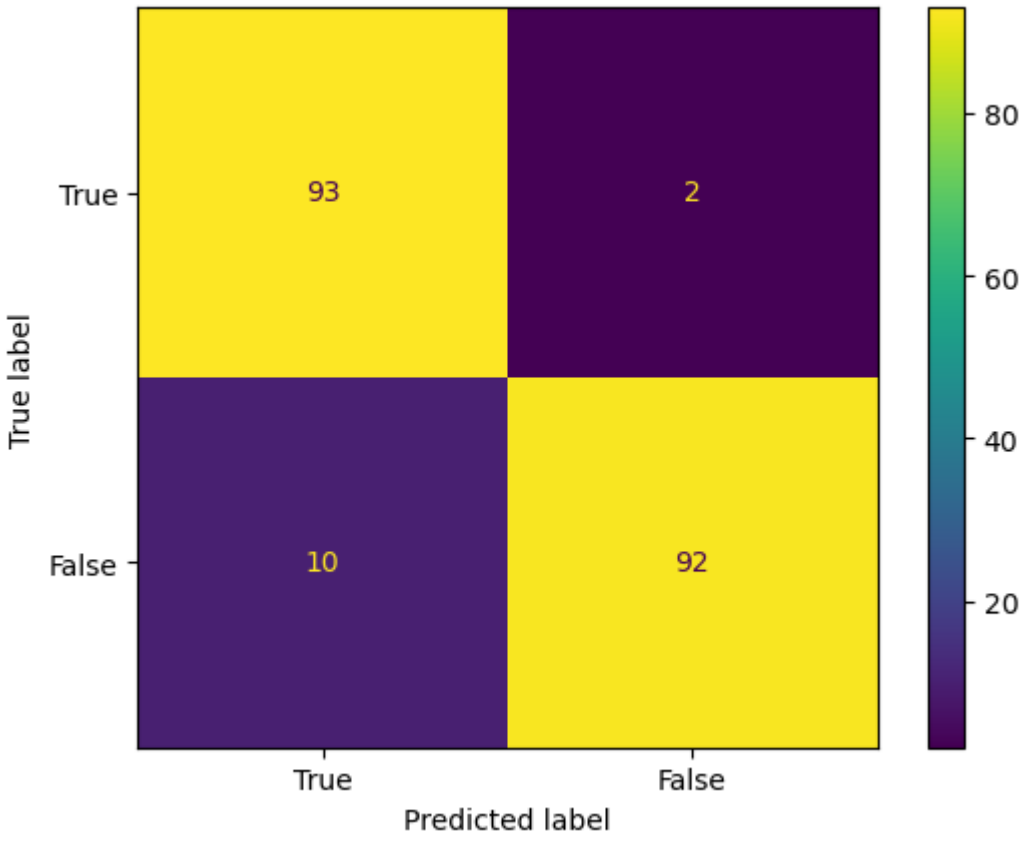
In [142]:
```
from    sklearn.metrics import    accuracy_score, precision_score, recall_scor
# Calculate the accuracy
accuracy =    accuracy_score(Y_test, test_predict)
# Calculate the precision
precision =    precision_score(Y_test,test_predict)
# Calculate the recall
recall =    recall_score(Y_test,test_predict)
# Calculate the f1 score
f1 =    f1_score(Y_test, test_predict)
# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9390862944162437
Precision: 0.9787234042553191
Recall: 0.9019607843137255
F1 Score: 0.9387755102040817
```

In [140]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

In [133]:
```
conf_matrix = confusion_matrix(Y_test,test_predict)
dis = ConfusionMatrixDisplay(confusion_matrix= conf_matrix, display_labels=[Tr

dis.plot()
plt.show()
```



In [137]:
```
group_names = ['True Positive','False Positive','False Negative','True Negativ
group_counts = ["{0:0.0f}".format(value) for value in conf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for    value in    conf_matrix.fl
labels = [f"{v1}\n{v2}\n{v3}" for    v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels =    np.asarray(labels).reshape(2,2)
sns.heatmap(conf_matrix, annot=labels, fmt='', cmap='summer')
```

Out[137]: <Axes: >

In [ ]: