Data Processing

Imports

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot as plt
import seaborn as sns
```

```
c:\Users\Hunter\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarn
ing: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' curren
tly installed).
  from pandas.core import (
```

Read the project_data.csv file

In [ ]:

```
df = pd.read_csv("data/project_data.csv")

df.head()
```

Out[ ]:

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration
0	192.168.10.5-104.16.207.165-54865-443-6	104.16.207.165	443	192.168.10.5	54865	6	7/7/2017 3:30	3
1	192.168.10.5-104.16.28.216-55054-80-6	104.16.28.216	80	192.168.10.5	55054	6	7/7/2017 3:30	109
2	192.168.10.5-104.16.28.216-55055-80-6	104.16.28.216	80	192.168.10.5	55055	6	7/7/2017 3:30	52
3	192.168.10.16-104.17.241.25-46236-443-6	104.17.241.25	443	192.168.10.16	46236	6	7/7/2017 3:30	34
4	192.168.10.5-104.19.196.102-54863-443-6	104.19.196.102	443	192.168.10.5	54863	6	7/7/2017 3:30	3

5 rows × 85 columns

Data cleaning and manipulation

1. Remove the column with majority NaN values
2. Use isnull() to figure out the number of NaN values per column

In [ ]:
```
df.columns = df.columns.str.strip()
df.dropna(axis=0, inplace=True)
df.isnull().sum().sum()
#df.isna()
```

Out[ ]:
```
0
```

1. Dropping unnecessary columns

In [ ]:
```
df.drop(columns=['Source IP'], inplace=True)
df.drop(columns=['Destination IP'], inplace=True)
df.drop(columns=['Source Port'], inplace=True)
df.drop(columns=['Destination Port'], inplace=True)
df.drop(columns=['Flow ID'], inplace=True)
df.drop(columns=['Timestamp'], inplace=True)
```

1. Convert labels from 'benign' and 'ddos' to 0 and 1

In [ ]:
```
df['Label_encoded'] = df['Label'].map({'BENIGN': 0, 'DDoS': 1})
df.drop(columns=['Label'], inplace=True)
df.head()
```

Out[ ]:

	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	mi
0	6	3	2	0	12	0	6	6	6.0	0.0	...	
1	6	109	1	1	6	6	6	6	6.0	0.0	...	
2	6	52	1	1	6	6	6	6	6.0	0.0	...	
3	6	34	1	1	6	6	6	6	6.0	0.0	...	
4	6	3	2	0	12	0	6	6	6.0	0.0	...	

5 rows × 79 columns

In [ ]:
```
#print tail of the data
df.tail()
```

Out[ ]:

	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	.
225740	6	61	1	1	6	6	6	6	6.0	0.0	.
225741	6	72	1	1	6	6	6	6	6.0	0.0	.
225742	6	75	1	1	6	6	6	6	6.0	0.0	.
225743	6	48	2	0	12	0	6	6	6.0	0.0	.
225744	6	68	1	1	6	6	6	6	6.0	0.0	.

5 rows × 79 columns

1. Converted the column 'Protocol' to dummy variable using pandas get_dummies

In [ ]:

```
cols = ['Protocol']
df_dummies = pd.get_dummies(df, columns=cols , prefix=cols,prefix_sep='_')

#show the head
df_dummies.head()
```

Out[ ]:

	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	...	Activ Ma
0	3	2	0	12	0	6	6	6.0	0.0	0	...	
1	109	1	1	6	6	6	6	6.0	0.0	6	...	
2	52	1	1	6	6	6	6	6.0	0.0	6	...	
3	34	1	1	6	6	6	6	6.0	0.0	6	...	
4	3	2	0	12	0	6	6	6.0	0.0	0	...	

5 rows × 81 columns

Data preparation for Logistic Regression

- Correlation heatmap

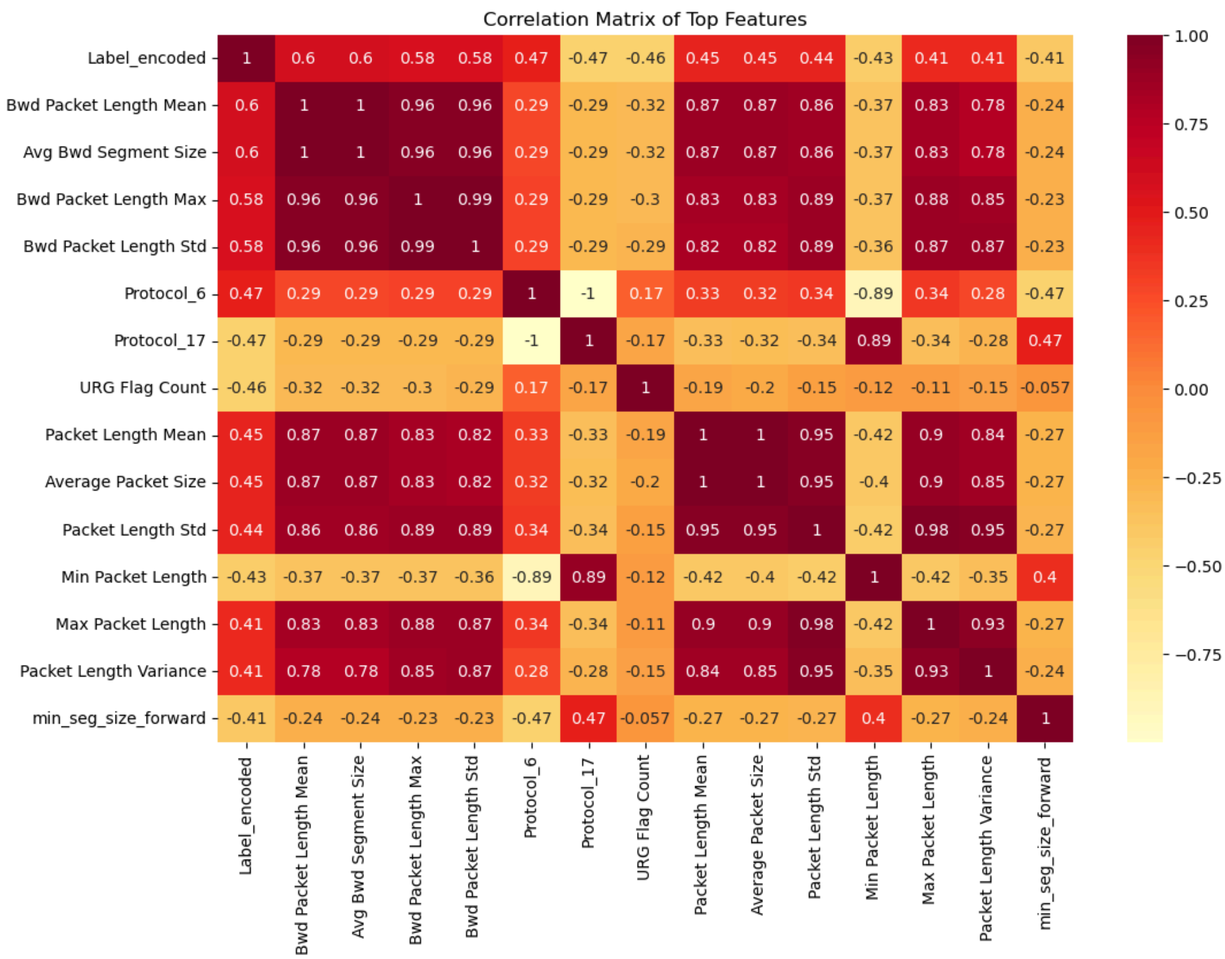
In [ ]:

```
#plt.figure(figsize=(18, 12))
#sns.heatmap(df.corr(), annot=True, cmap='YlOrRd')
#plt.show()
```

In [ ]:

```
corr_matrix = df_dummies.corr().abs()
top_features = corr_matrix['Label_encoded'].sort_values(ascending=False).head(15).inde
```

```
top_corr = df_dummies[top_features].corr()

plt.figure(figsize=(12, 8))
sns.heatmap(top_corr, annot=True, cmap='YlOrRd')
plt.title("Correlation Matrix of Top Features")
plt.show()
```



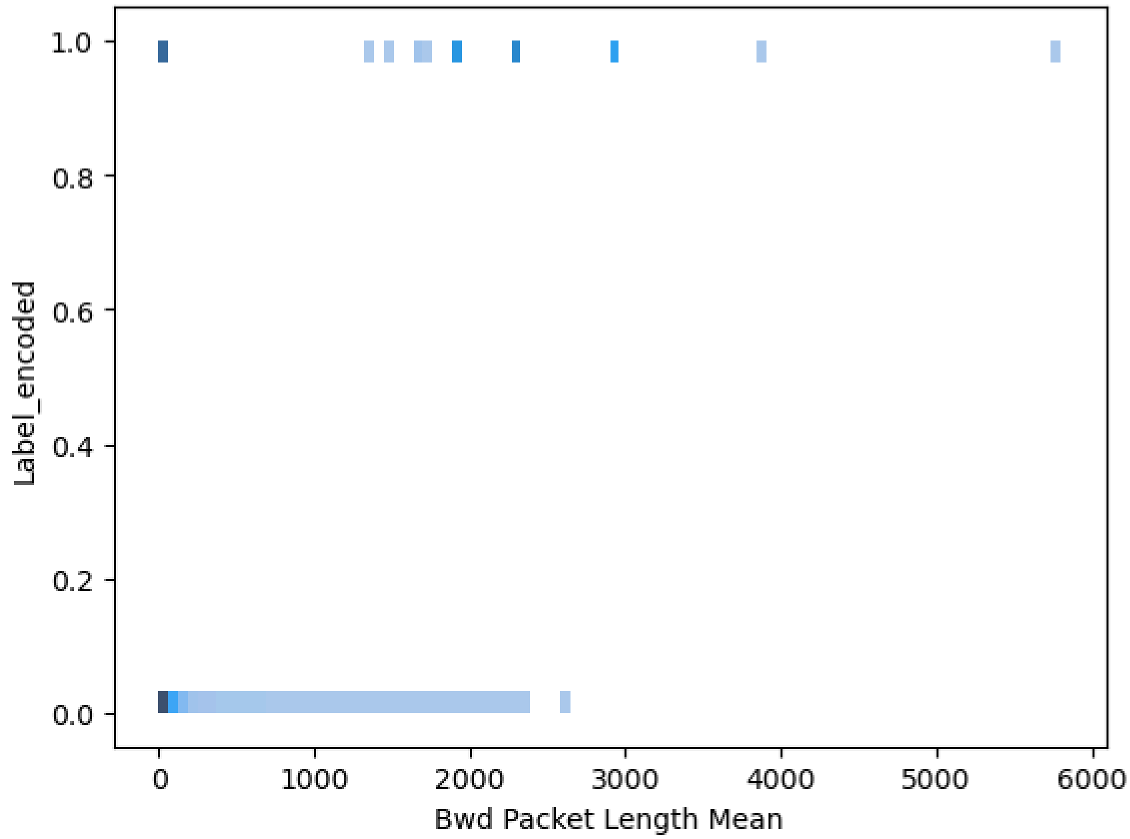
In [ ]:

```
sns.histplot(data = df_dummies , x ='Bwd Packet Length Mean',y = 'Label_encoded')
#sns.histplot(data = df , x ='Bwd Packet Length Std',y = 'Label_encoded')
#sns.histplot(data = df , x ='Bwd Packet Length Max',y = 'Label_encoded')
#sns.histplot(data = df , x ='Avg Bwd Segment Size',y = 'Label_encoded')
```

```
C:\Users\tobyu\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tobyu\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[ ]:

```
<Axes: xlabel='Bwd Packet Length Mean', ylabel='Label_encoded'>
```



1. Split the data into features (X) and target (y):

In [ ]:
```
X = df.drop(['Label_encoded'], axis=1)
y = df['Label_encoded']
```

1. Handle categorical variables based on highest correlation

In [ ]:
```
X = df_dummies[['Bwd Packet Length Mean', 'Avg Bwd Segment Size']]
```

1. Use StandardScaler from sklearn to transform the x dataframe.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

1. Split dataset into train and test data use train_test_split with test_size = 0.2 and random_state = 42

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random
```

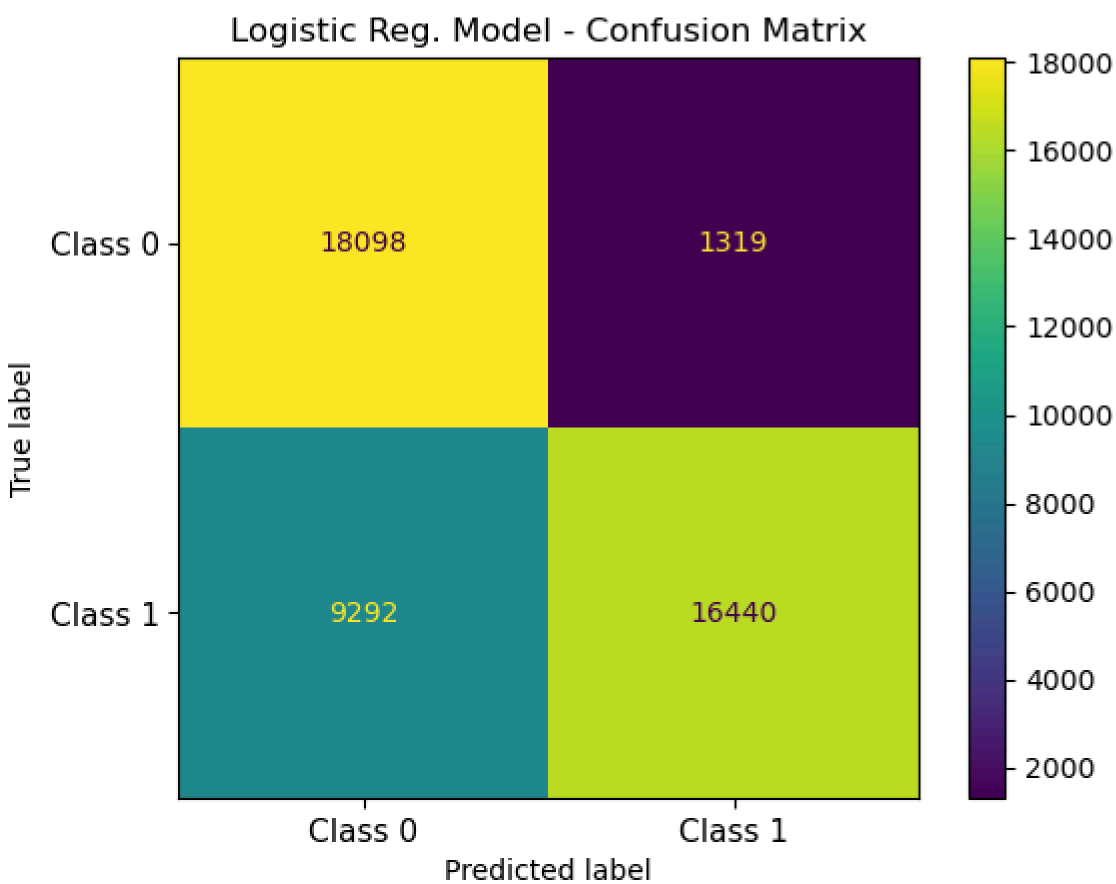
Classification Model 1: Logistic Regression

1. Create a logistic regression model using sklearn
2. Fit the model with the train data
3. Get the score from the model using test data
4. Plot confusion matrix using [ConfusionMatrixDisplay]

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Create a logistic regression model using sklearn library
clf=LogisticRegression()
clf.fit(X_train,y_train)

#print score for test data
print(clf.score(X_test,y_test))
```

```
0.7649781833484685
```

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

cm = ConfusionMatrixDisplay.from_estimator(clf,X_test, y_test)
#plt.figure()
#plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("Logistic Reg. Model - Confusion Matrix")
plt.xticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.yticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.show()
```



Evaluating the Model

In [ ]:
```
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.76
```

Classification Model 2: K Nearest Neighbor Classifier

1. Create a KNN model using sklearn library, and initialize n_neighbors
2. Fit the model with the train data
3. Predict the values from test data
4. Print out the score from training and test data
5. Repeat Step 1.- 4. for a range of `n_neighbors` values (k in kNN) from 1 to 30.

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier


# Define KNN model
for k in range(1,30):

    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit KNN model on xtrain, ytrain from above
    knn.fit(X_train, y_train)

    #predict y values from xtest
    y_pred=knn.predict(X_test)

    #print score for test data
    print("K: ",k,"Train Score: ",knn.score(X_train,y_train), "Test Score: ",knn.score
```

```
K:  1 Train Score:  0.9414315141313016 Test Score:  0.9407295842654322
K:  2 Train Score:  0.9414204394436076 Test Score:  0.9407295842654322
K:  3 Train Score:  0.9413816780366794 Test Score:  0.9407074353806286
K:  4 Train Score:  0.793213431381235 Test Score:  0.7941039668652683
K:  5 Train Score:  0.9413706033489856 Test Score:  0.9407074353806286
K:  6 Train Score:  0.9413706033489856 Test Score:  0.9407074353806286
K:  7 Train Score:  0.9413373792859041 Test Score:  0.9406631376110213
K:  8 Train Score:  0.9413373792859041 Test Score:  0.9406631376110213
K:  9 Train Score:  0.9413373792859041 Test Score:  0.9406631376110213
K:  10 Train Score:  0.9413373792859041 Test Score:  0.9406631376110213
K:  11 Train Score:  0.9413041552228227 Test Score:  0.9406409887262176
K:  12 Train Score:  0.9413041552228227 Test Score:  0.9406409887262176
K:  13 Train Score:  0.9412709311597413 Test Score:  0.940618839841414
K:  14 Train Score:  0.9412709311597413 Test Score:  0.940618839841414
K:  15 Train Score:  0.9412432444405068 Test Score:  0.940618839841414
K:  16 Train Score:  0.9412432444405068 Test Score:  0.940618839841414
K:  17 Train Score:  0.9412377070966599 Test Score:  0.9405966909566104
K:  18 Train Score:  0.9412377070966599 Test Score:  0.9405966909566104
K:  19 Train Score:  0.9411989456897315 Test Score:  0.940552393187003
K:  20 Train Score:  0.9411989456897315 Test Score:  0.940552393187003
K:  21 Train Score:  0.9411823336581908 Test Score:  0.940552393187003
K:  22 Train Score:  0.9411823336581908 Test Score:  0.940552393187003
K:  23 Train Score:  0.9411657216266501 Test Score:  0.9405080954173958
K:  24 Train Score:  0.9411657216266501 Test Score:  0.9405080954173958
K:  25 Train Score:  0.9411380349074157 Test Score:  0.9404859465325921
K:  26 Train Score:  0.9411380349074157 Test Score:  0.9404859465325921
K:  27 Train Score:  0.9411214228758749 Test Score:  0.9404637976477884
K:  28 Train Score:  0.9411214228758749 Test Score:  0.9404637976477884
K:  29 Train Score:  0.9410937361566404 Test Score:  0.9404637976477884
```

1. Create a KNN Classifier model using the best value of k found from previous question.
2. Train the model using xtrain, ytrain values.
3. Plot confusion matrix for the xtest and ytest,

In [ ]:

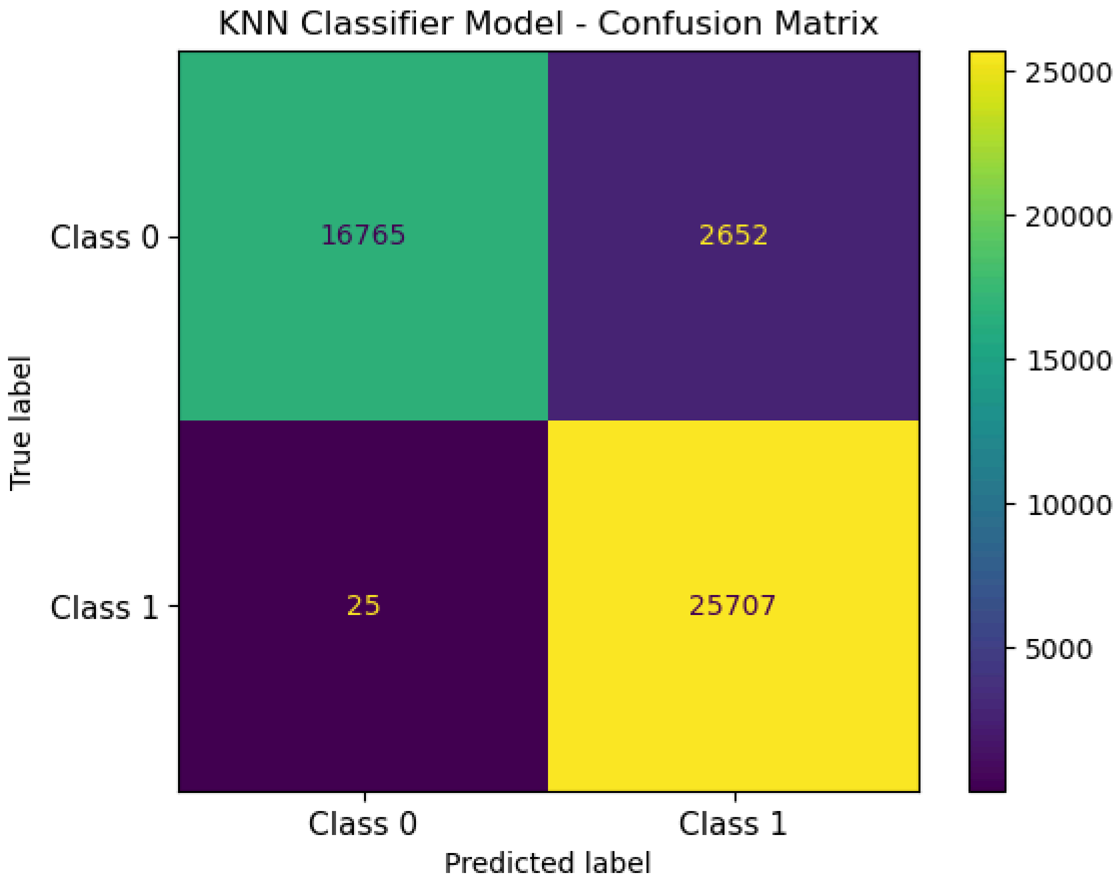
```
knn_best = KNeighborsClassifier(n_neighbors=6)

knn_best.fit(X_train, y_train)
```

Out[ ]:

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

In [ ]:

```
cm = ConfusionMatrixDisplay.from_estimator(knn_best,X_test, y_test)
#plt.figure()
#plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("KNN Classifier Model - Confusion Matrix")
plt.xticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.yticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.show()
```



In [ ]:
```
print ("Accuracy is ",knn_best.score(X_test,y_test))
```

```
Accuracy is  0.9407074353806286
```

In [ ]:
```
from sklearn.model_selection import cross_val_score

for k in range(1,30):

    knn_crossval = KNeighborsClassifier(n_neighbors=k)

    # Use sklearn for 5 fold cross validation
    scores_cv=cross_val_score(knn_crossval,X_train,y_train,cv=5)

    # print the scores from different folds
    print(scores_cv)
```

```
[0.7926576  0.94124976 0.94149731 0.94091589 0.94257711]
[0.79290678 0.94124976 0.94149731 0.94094357 0.94257711]
[0.79290678 0.94124976 0.94149731 0.9408882  0.94257711]
[0.79290678 0.94124976 0.94149731 0.9408882  0.94257711]
[0.79290678 0.94122207 0.94144194 0.94086051 0.94249405]
[0.79290678 0.94122207 0.94144194 0.94086051 0.94249405]
[0.79290678 0.94119439 0.94141425 0.94086051 0.94249405]
[0.79290678 0.94119439 0.94141425 0.94086051 0.94249405]
[0.79290678 0.9411667  0.94135888 0.94083283 0.94243867]
[0.79290678 0.9411667  0.94135888 0.94083283 0.94243867]
[0.9405576  0.94111133 0.94133119 0.94083283 0.94241099]
[0.9405576  0.94111133 0.94133119 0.94083283 0.94241099]
[0.94050223 0.94111133 0.94130351 0.94077745 0.94241099]
[0.94050223 0.94111133 0.94130351 0.94083283 0.94241099]
[0.94047454 0.94111133 0.94122044 0.94069439 0.94241099]
[0.94047454 0.94111133 0.94122044 0.94069439 0.94241099]
[0.94044686 0.94108364 0.94116507 0.9406667  0.94241099]
[0.94044686 0.94108364 0.94116507 0.9406667  0.94241099]
[0.94044686 0.94102827 0.94113738 0.94063902 0.94235561]
[0.94044686 0.94102827 0.94113738 0.94063902 0.94235561]
[0.94044686 0.9409729  0.94105432 0.94058364 0.94232793]
[0.94044686 0.9409729  0.94105432 0.94058364 0.94232793]
[0.94044686 0.9409729  0.94099895 0.94058364 0.94227255]
[0.79276835 0.9409729  0.94099895 0.94058364 0.94227255]
[0.94041917 0.94094521 0.94097126 0.94052827 0.94221718]
[0.79274066 0.94094521 0.94097126 0.94052827 0.94221718]
[0.79271298 0.94094521 0.94091589 0.94052827 0.94218949]
[0.79271298 0.94094521 0.94091589 0.94052827 0.94218949]
[0.79271298 0.94094521 0.94086051 0.94044521 0.94218949]
```

In [ ]:

```
```

Classification Model 3: Decision Tree Classifier

1. Create a DecisionTreeClassifier model
2. Fit the model with train data
3. Predict the test data using the trained model
4. Calculate the Mean Squared Error (MSE) of the model's prediction
5. Print the precison recall curve for the test data with minimum MSE value from the trained model
6. Plot Confusion Matrix
7. Display accuracy of prediction

In [ ]:

```
#Create Decision Tree Classifier
clf_1 = DecisionTreeClassifier(random_state = 30, criterion = 'entropy')

#Fit and Score Classifier Model
clf_1.fit(X_train, y_train)
clf_1.score(X_test, y_test)
```

Out[ ]:

```
0.9407295842654322
```

In [ ]:

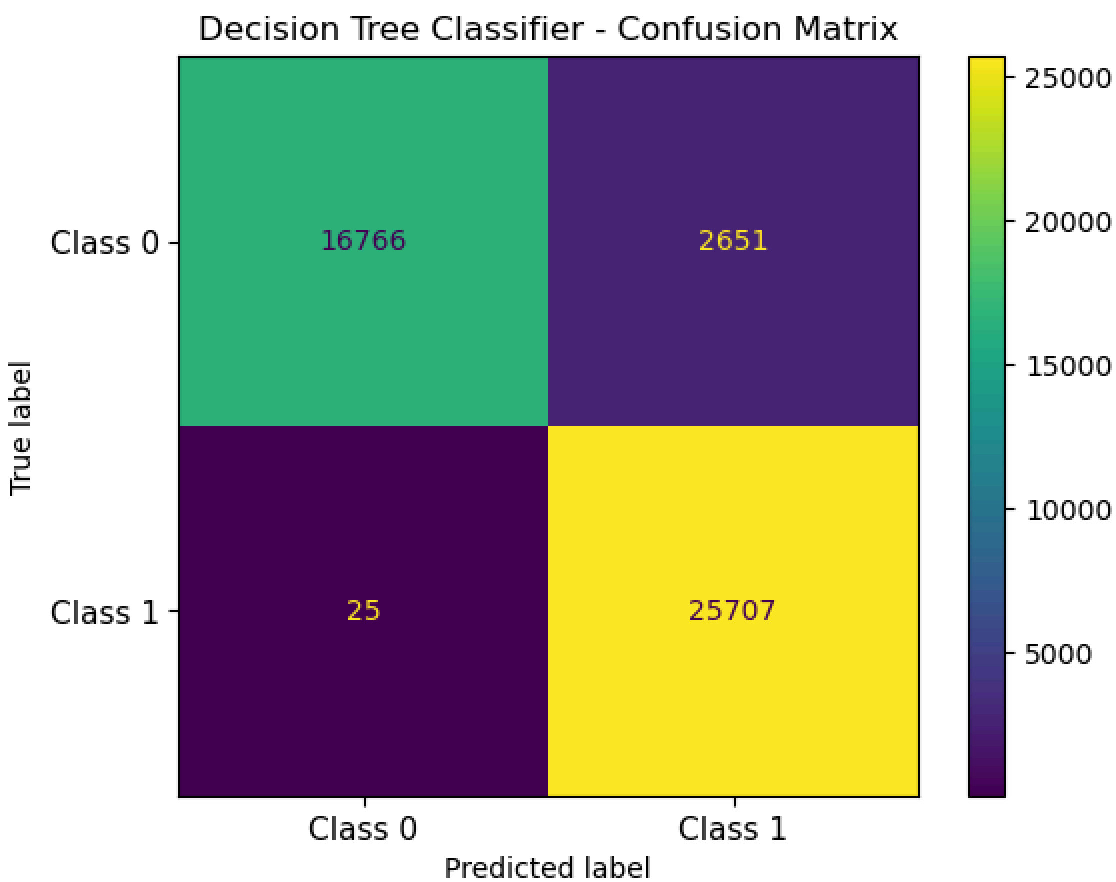
```
from sklearn.metrics import mean_squared_error
```

```
#Predict
y_1 = clf_1.predict(X_test)

#Calculate mean_squared_error
print(mean_squared_error(y_test, y_1))
```

```
0.05927041573456777
```

In [ ]:

```
cm = ConfusionMatrixDisplay.from_estimator(clf_1, X_test, y_test)
plt.title("Decision Tree Classifier - Confusion Matrix")
plt.xticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.yticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.show()
```



In [ ]:

```
accuracy = accuracy_score(y_test, y_1)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.94
```

In [ ]: