In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.signal import find_peaks
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, confusi
from sklearn.metrics import accuracy_score, precision_score, recall_score,
```

In [2]:
```python
#import the train and test datasets
df0 = pd.read_csv('mitbih_train.csv', header = None)
df1 = pd.read_csv('mitbih_test.csv', header = None )
```

In [3]:
```python
#show the number of columns in each dataframe
columns_df0 = df0.columns
print("'mitbih_train.csv: ", columns_df0)
print('     ')
print('####################################################################
print('             ')
columns_df1 = df1.columns
print("mitbih_test.csv: ", columns_df1)
```

```
'mitbih_train.csv:  Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,
8,   9,
            ...
            178, 179, 180, 181, 182, 183, 184, 185, 186, 187],
           dtype='int64', length=188)

########################################################################
####

mitbih_test.csv:  Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,
8,   9,
            ...
            178, 179, 180, 181, 182, 183, 184, 185, 186, 187],
           dtype='int64', length=188)
```

In [4]:
```python
# concatenate the dataframes to make one dataframe dfs
dfs = [df0, df1]
dfs = pd.concat(dfs, axis=0)
```

In [5]:
```
dfs.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8
0	0.977941	0.926471	0.681373	0.245098	0.154412	0.191176	0.151961	0.085784	0.058824
1	0.960114	0.863248	0.461538	0.196581	0.094017	0.125356	0.099715	0.088319	0.074074
2	1.000000	0.659459	0.186486	0.070270	0.070270	0.059459	0.056757	0.043243	0.054054
3	0.925414	0.665746	0.541436	0.276243	0.196133	0.077348	0.071823	0.060773	0.066298
4	0.967136	1.000000	0.830986	0.586854	0.356808	0.248826	0.145540	0.089202	0.117371

5 rows × 188 columns

In [6]:
```
dfs.info()#data shows all the information about the datafarme including nu
#and the data types and amount of memory
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109446 entries, 0 to 21891
Columns: 188 entries, 0 to 187
dtypes: float64(188)
memory usage: 157.8 MB
```

In [7]:
```
# check for missing values, we can see that there are no null values in th
dfs.isnull().sum()
```

Out[7]:
```
0      0
1      0
2      0
3      0
4      0
      ..
183    0
184    0
185    0
186    0
187    0
Length: 188, dtype: int64
```

In [8]:

```python
# finding out the number of unique values in each column
# we can see that in column 187 there are five unique values which stand f
#of the different ecg classes
dfs.nunique()
```

Out[8]:

```
0      26912
1      48772
2      57132
3      46919
4      44736
       ...
183     1128
184     1033
185      961
186      912
187        5
Length: 188, dtype: int64
```

In [9]:

```python
dfs.shape #shape of the merged dataframe showing 109446, rows and 188 colu
```

Out[9]:

```
(109446, 188)
```

In [10]:

```python
#randomise the dataframe and reset index
dfs = dfs.sample(frac=1).reset_index(drop = True)
```

In [ ]:

```python

```

In [11]:
```python
dfs.head(20)
```

Out[11]:

	0	1	2	3	4	5	6	7	
0	1.000000	0.691667	0.166667	0.013889	0.038889	0.036111	0.005556	0.002778	0.00000
1	1.000000	0.774487	0.232346	0.143508	0.129841	0.104784	0.077449	0.091116	0.05922
2	1.000000	0.892982	0.428070	0.128070	0.182456	0.198246	0.157895	0.154386	0.15614
3	1.000000	0.831099	0.310992	0.040214	0.040214	0.053619	0.029491	0.008043	0.00536
4	1.000000	0.907692	0.712821	0.533333	0.307692	0.153846	0.123077	0.174359	0.16410
5	0.975332	0.747628	0.018975	0.011385	0.000000	0.024668	0.062619	0.096774	0.09297
6	1.000000	0.820475	0.495549	0.197329	0.000000	0.060831	0.129080	0.111276	0.09198
7	0.995614	0.894737	0.320175	0.006579	0.160088	0.175439	0.094298	0.072368	0.07236
8	0.898734	0.857595	0.496835	0.000000	0.072785	0.205696	0.300633	0.306962	0.34177
9	0.000000	0.010336	0.060724	0.142119	0.241602	0.310078	0.383721	0.454780	0.49870
10	0.990148	0.889984	0.431855	0.014778	0.050903	0.210181	0.231527	0.210181	0.21510
11	1.000000	0.810127	0.396624	0.143460	0.092827	0.118143	0.088608	0.059072	0.05063
12	1.000000	0.892473	0.403226	0.000000	0.190860	0.314516	0.370968	0.368280	0.39516
13	0.969453	0.959807	0.403537	0.000000	0.205788	0.336013	0.316720	0.300643	0.31028
14	1.000000	0.409302	0.000000	0.106977	0.083721	0.088372	0.097674	0.097674	0.08837
15	1.000000	0.801252	0.425665	0.142410	0.000000	0.065728	0.143975	0.139280	0.11737
16	1.000000	0.916493	0.538622	0.089770	0.000000	0.167015	0.248434	0.229645	0.21503
17	0.841492	0.784382	0.712121	0.651515	0.579254	0.489510	0.412587	0.313520	0.22843
18	0.930124	0.934783	0.388199	0.017081	0.250000	0.333851	0.343168	0.336957	0.33229
19	0.864564	0.923933	0.465677	0.089054	0.031540	0.005566	0.000000	0.024119	0.08163

20 rows × 188 columns

In [12]:
```python
dfs = dfs.rename(columns={187: 'classes'})
```

In [13]:
```
# describe the dfs dataframe in terms of metrics like count, mean, standar
dfs.describe()
```

Out[13]:

	0	1	2	3	4	
count	109446.000000	109446.000000	109446.000000	109446.000000	109446.000000	109446
mean	0.891170	0.758909	0.424503	0.219602	0.201237	0
std	0.239657	0.221190	0.227561	0.207248	0.177191	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.922252	0.682648	0.251014	0.048853	0.082418	0
50%	0.991202	0.826560	0.430174	0.166355	0.147842	0
75%	1.000000	0.910868	0.579832	0.342707	0.259045	0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1

8 rows × 188 columns

In [14]:
```
#convert the last column to integer
dfs['classes']=dfs['classes'].astype(int)
```

In [15]:
```
# get the number of unique values in the last column and use it to plot a
# to visualise the categories
categories =dfs['classes'].value_counts()
categories
```

Out[15]:
```
0    90589
4     8039
2     7236
1     2779
3      803
Name: classes, dtype: int64
```

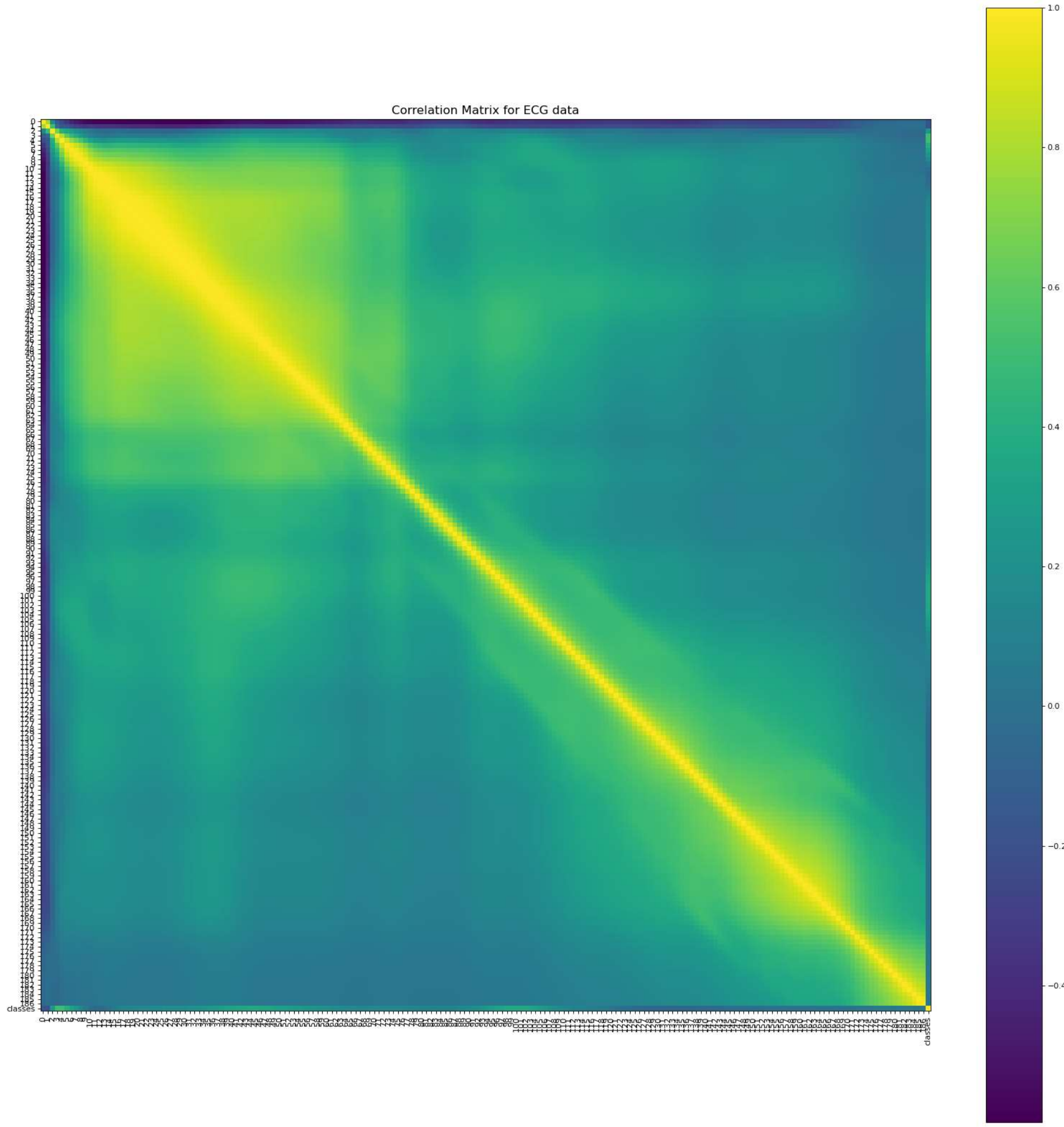
In [ ]:
```

```

In [17]:

```python
# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = 'ECG data'
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns w
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or const
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecol
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()
```

In [18]:
```
plotCorrelationMatrix(dfs, 25)
```

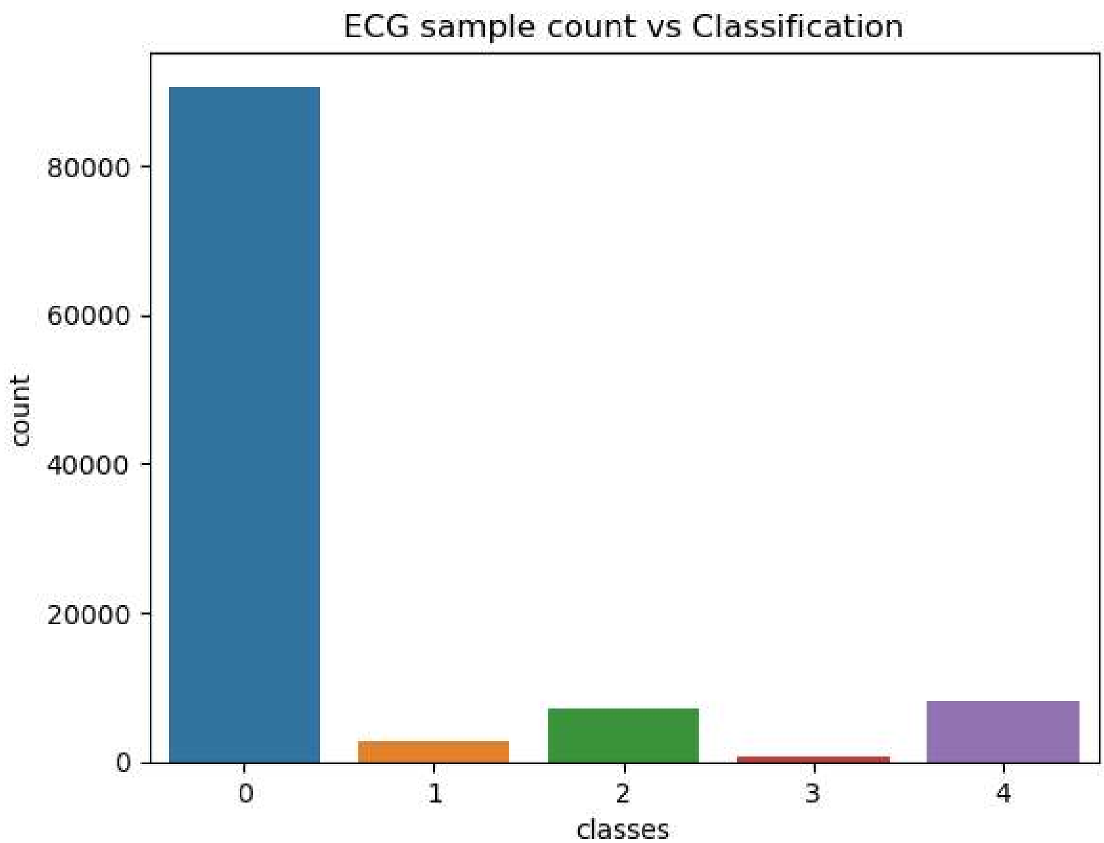
```
C:\Users\HP\AppData\Local\Temp\ipykernel_12656\1307321360.py:4: FutureWar
ning: In a future version of pandas all arguments of DataFrame.dropna wil
l be keyword-only.
  df = df.dropna('columns') # drop columns with NaN
```



In [ ]:

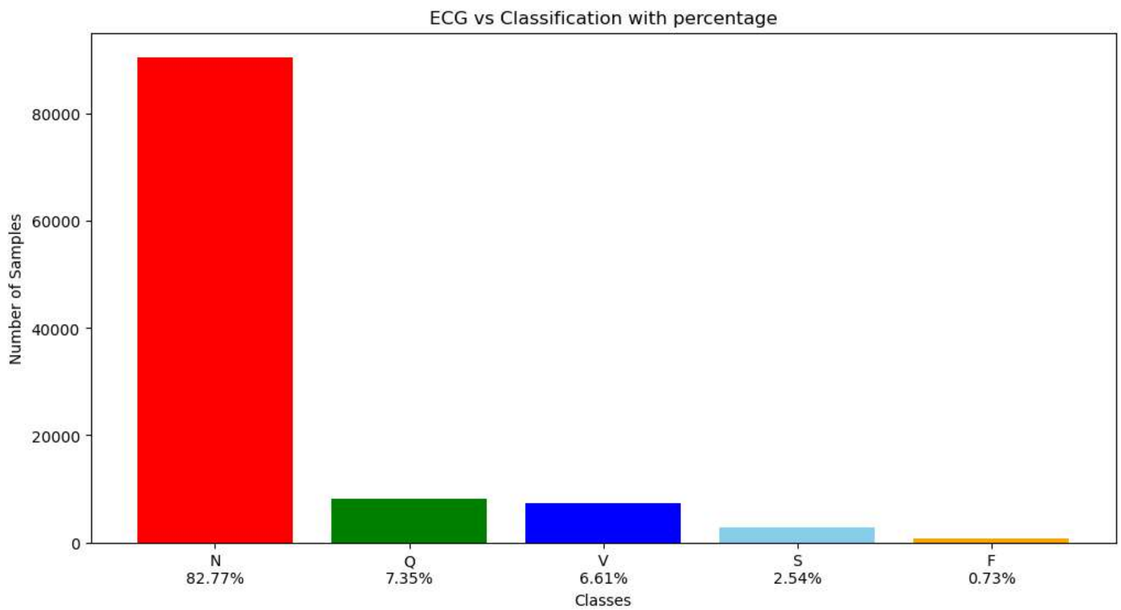
In [19]:

```
# performing a countplot to visualise the data based on the number of clas
sns.countplot(x='classes', data=dfs)
plt.title('ECG sample count vs Classification')
plt.show()
```



In [20]:

```
#use categories to plot a bar chart to visualise the distribution categori
# Classes: ['N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4]

# Create an array with the positions of the bars on the x-axis
positions = np.arange(len(categories))

# Create the bar chart
for i in categories:
    i = categories / len(dfs) * 100

plt.figure(figsize=(12,6))
plt.bar(positions, categories, color=['red','green','blue','skyblue','oran
plt.title('ECG vs Classification with percentage')
plt.xlabel('Classes')
plt.ylabel('Number of Samples')

# Create names on the x-axis
plt.xticks(positions, [f'N\n{i.values[0]:.2f}%',f'Q\n{i.values[1]:.2f}%',
                       f'V\n{i.values[2]:.2f}%',f'S\n{i.values[3]:.2f}%',
                       f'F\n{i.values[4]:.2f}%'])

# Show the graph
plt.show()
```



the graph shows that the category 'N' which means normal beats is in the overwhelming majority

In [ ]:

```

```

define a function to choose one sample per class

In [21]:
```
def indi_sample(dfs):
    return dfs.sample(1)
sample_per_class = dfs.groupby('classes', group_keys = False).apply(indi_s
```

In [22]:
```
sample_per_class
```

Out[22]:

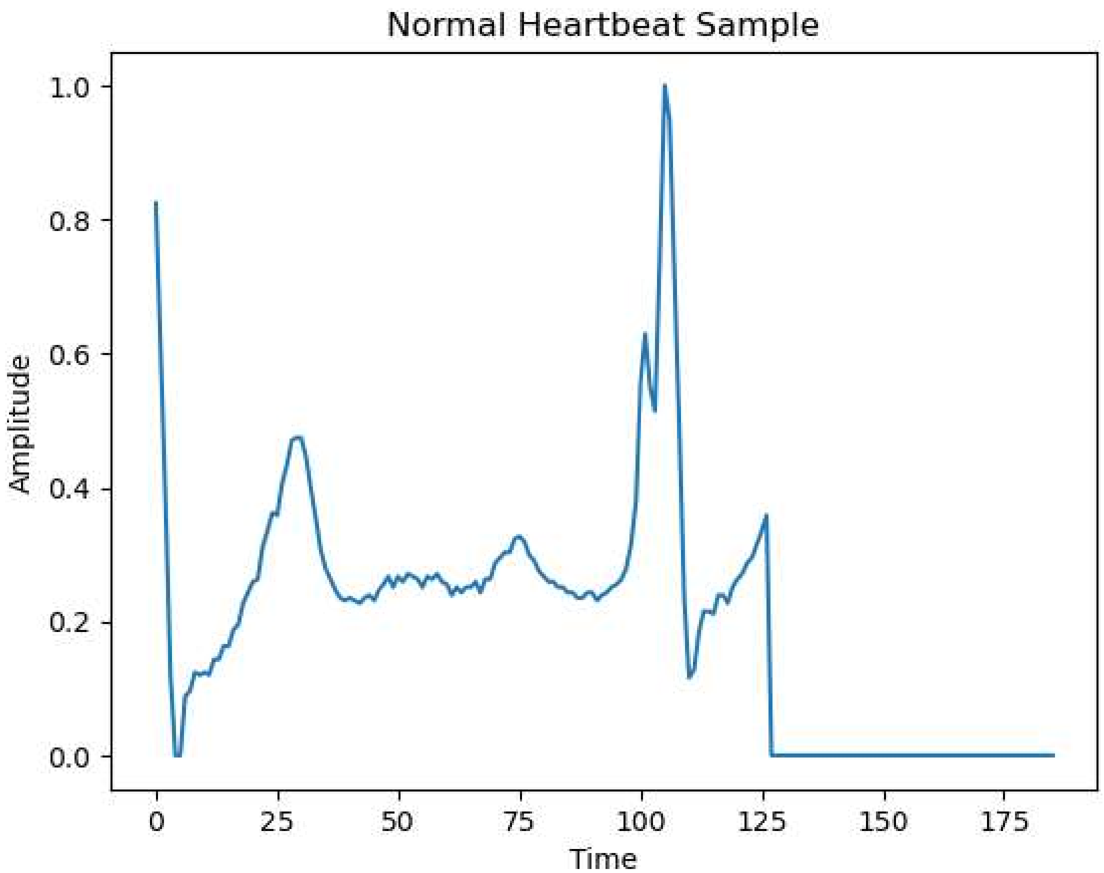
	0	1	2	3	4	5	6	7	
65298	0.824701	0.601594	0.374502	0.119522	0.000000	0.000000	0.087649	0.095618	0.12
6956	1.000000	0.882096	0.253275	0.021834	0.065502	0.126638	0.109170	0.157205	0.21
68100	0.000000	0.038462	0.108392	0.178322	0.269231	0.356643	0.444056	0.510490	0.56
65635	1.000000	0.837545	0.368231	0.252708	0.169675	0.086643	0.090253	0.093863	0.07
15414	0.496259	0.416459	0.379052	0.376559	0.336658	0.311721	0.264339	0.224439	0.17

5 rows × 188 columns

from the dataframe above, we can have a look at individual examples of each ecg class

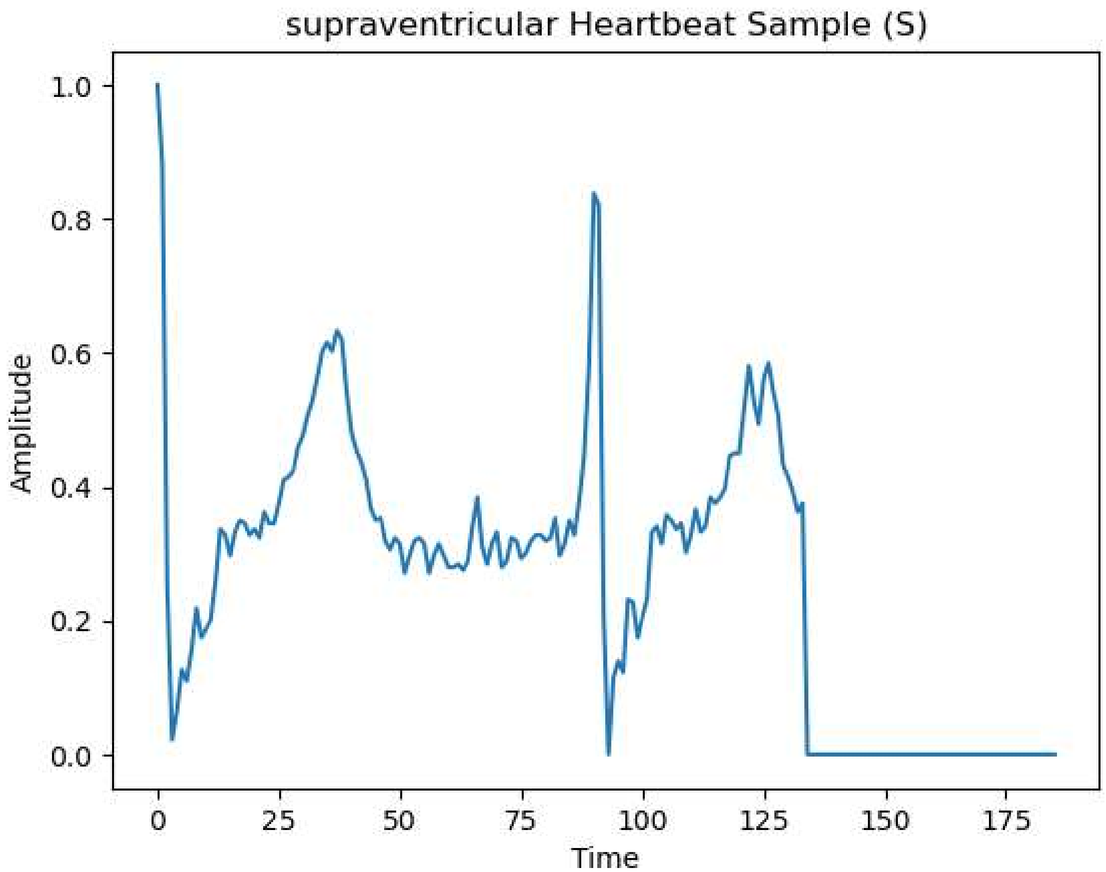
In [23]:
```
# class N normal beat
plt.plot(sample_per_class.iloc[0,:186])
plt.title('Normal Heartbeat Sample')
plt.xlabel('Time')
plt.ylabel('Amplitude')
```

Out[23]:
```
Text(0, 0.5, 'Amplitude')
```



In [24]:
```
# class S supraventricular beat
plt.plot(sample_per_class.iloc[1,:186])
plt.title('supraventricular Heartbeat Sample (S)')
plt.xlabel('Time')
plt.ylabel('Amplitude')
```

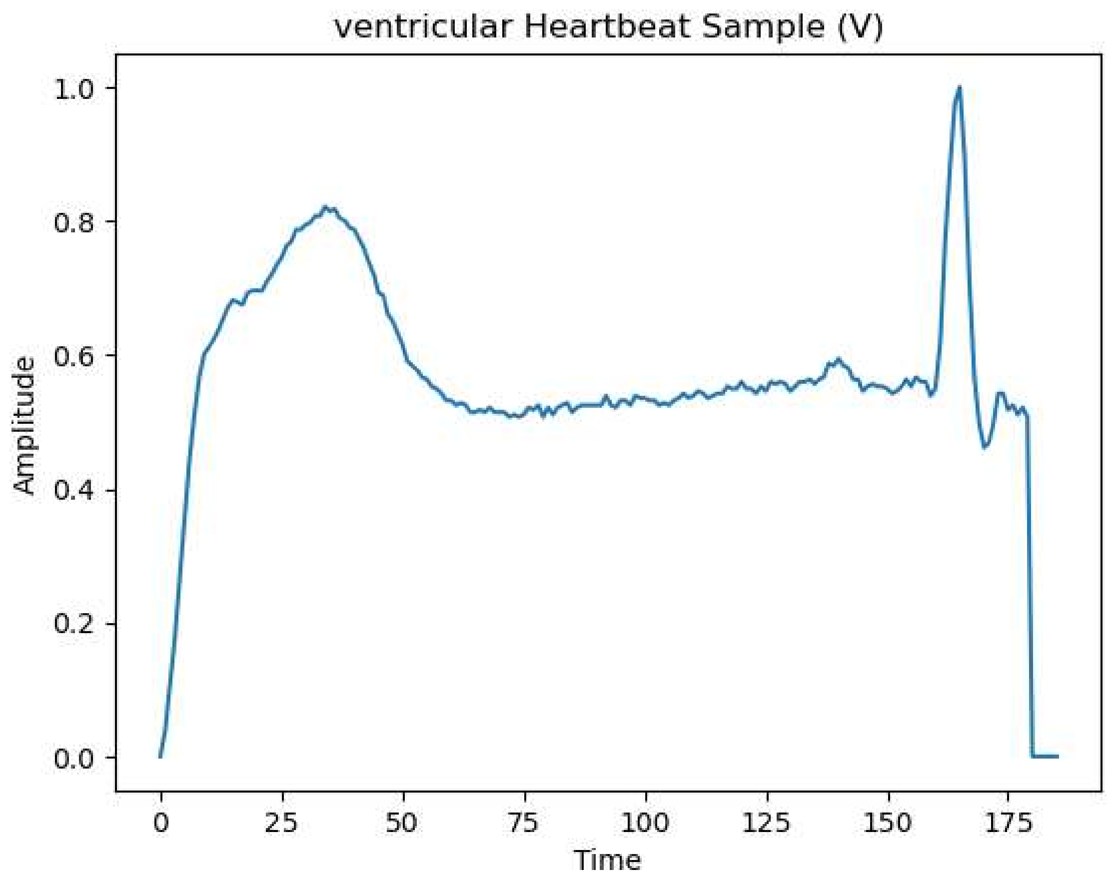
Out[24]:
```
Text(0, 0.5, 'Amplitude')
```



In [25]:

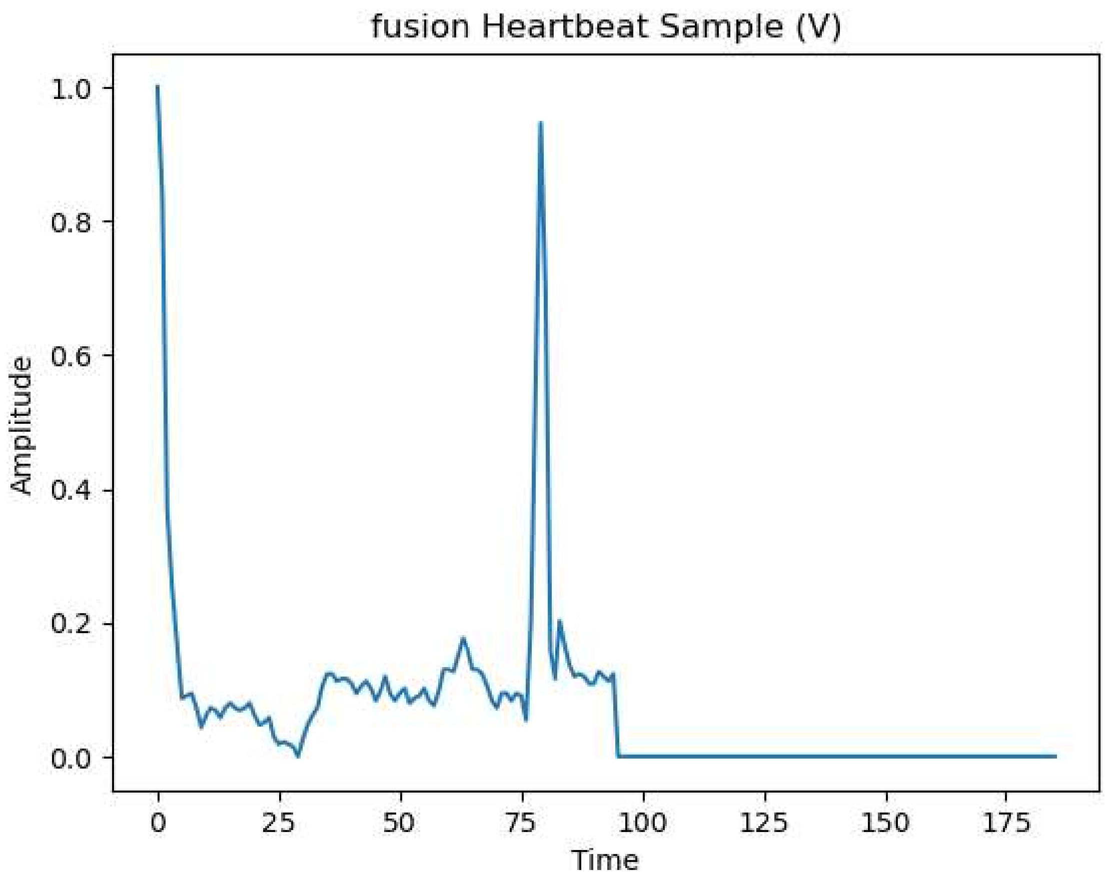
```python
# class V ventricular beat
plt.plot(sample_per_class.iloc[2,:186])
plt.title('ventricular Heartbeat Sample (V)')
plt.xlabel('Time')
plt.ylabel('Amplitude')
```

Out[25]: Text(0, 0.5, 'Amplitude')



In [26]:
```
# class F  fusion beat
plt.plot(sample_per_class.iloc[3,:186])
plt.title('fusion Heartbeat Sample (V)')
plt.xlabel('Time')
plt.ylabel('Amplitude')
```

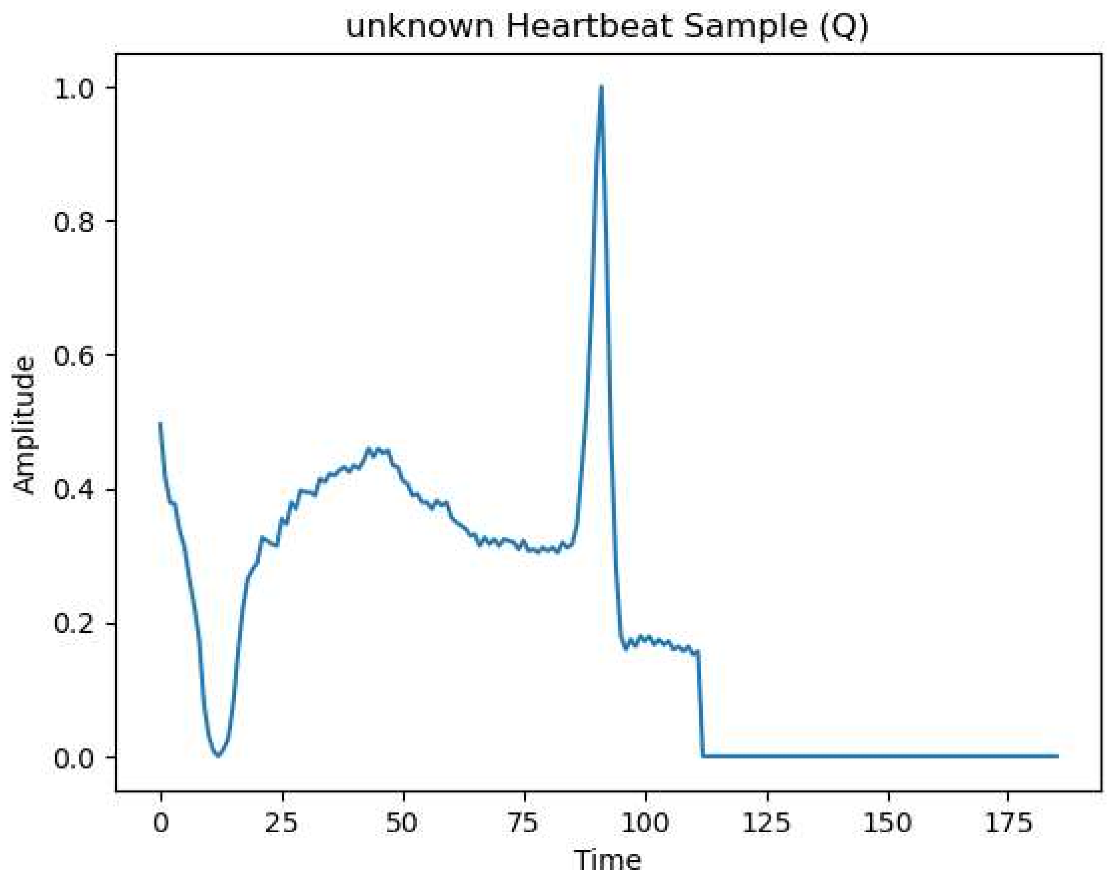
Out[26]: Text(0, 0.5, 'Amplitude')



In [27]:

```
# class Q is unknown beat
plt.plot(sample_per_class.iloc[4,:186])
plt.title('unknown Heartbeat Sample (Q)')
plt.xlabel('Time')
plt.ylabel('Amplitude')
```

Out[27]: Text(0, 0.5, 'Amplitude')



In [ ]:

Feature Selection

Here we seperate the dataframe into features and target variabes X and y

From observations, we know that the last column in the dataset represents the classes that signify the type of heartbeat signal Classes: ['N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4]. Thus, the 'classes' column will be dropped from the dataset and used as the target or output and the rest of the dataframe,X as the input .variable.

In [28]:
```python
# differentiate the X inputs from the y outputs
y = dfs['classes']
```

In [29]:
```python
y.shape
```

Out[29]:
```
(109446,)
```

In [30]:
```python
y
```

Out[30]:
```
0         0
1         0
2         0
3         0
4         0
         ..
109441    0
109442    0
109443    0
109444    0
109445    0
Name: classes, Length: 109446, dtype: int32
```

In [31]:
```python
X = dfs.drop('classes', axis = 1)
```

In [32]:
```python
X.shape
```

Out[32]:
```
(109446, 187)
```

In [33]:
```python
X
```

Out[33]:

	0	1	2	3	4	5	6	7	
0	1.000000	0.691667	0.166667	0.013889	0.038889	0.036111	0.005556	0.002778	0.0
1	1.000000	0.774487	0.232346	0.143508	0.129841	0.104784	0.077449	0.091116	0.0
2	1.000000	0.892982	0.428070	0.128070	0.182456	0.198246	0.157895	0.154386	0.1
3	1.000000	0.831099	0.310992	0.040214	0.040214	0.053619	0.029491	0.008043	0.0
4	1.000000	0.907692	0.712821	0.533333	0.307692	0.153846	0.123077	0.174359	0.1
...	...	...	...	...	...	...	...	...	
109441	0.916107	1.000000	0.640940	0.191275	0.083893	0.063758	0.057047	0.062081	0.0
109442	1.000000	0.830084	0.487465	0.069638	0.064067	0.203343	0.300836	0.364902	0.3
109443	0.992110	0.865878	0.299803	0.076923	0.029586	0.000000	0.007890	0.047337	0.1
109444	0.131105	0.205656	0.352185	0.465296	0.542416	0.652956	0.745501	0.781491	0.7
109445	0.980723	0.949398	0.539759	0.127711	0.065060	0.081928	0.084337	0.096386	0.0

109446 rows × 187 columns

Model Validation using RandomForest Classifier

In [86]:
```python
# Split the dataset into a training set and a val set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, ran
```

In [87]:
```python
# model validation using the random forrest clasifier
clf = RandomForestClassifier()
```

In [88]:
```python
clf.fit(X_train, y_train)
```

Out[88]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [89]:
```python
y_validation_prediction = clf.predict(X_val)
```

In [90]:
```python
acc = accuracy_score (y_val, y_validation_prediction) * 100
```

In [91]:
```python
print(f' Validation Accuracy: {acc :.2f}%')
```

```
 Validation Accuracy: 97.39%
```

In [92]:
```python
train_pred_RF = clf.predict(X_train)
```

In [ ]:
```python

```

In [93]:
```python
recall_RF = recall_score(y_val, y_validation_prediction, average = 'micro'
print(f' Validation Recall: {recall_RF :.2f}%')
```

```
 Validation Recall: 97.39%
```

In [94]:
```python
precision_RF = precision_score(y_val, y_validation_prediction, average = '
print(f' Validation precision: {precision_RF :.2f}%')
```

```
 Validation precision: 97.39%
```

In [95]:
```python
F_score_RF = f1_score(y_val, y_validation_prediction, average = 'micro' )
print(f' Validation F-score: {F_score_RF :.2f}%')
```

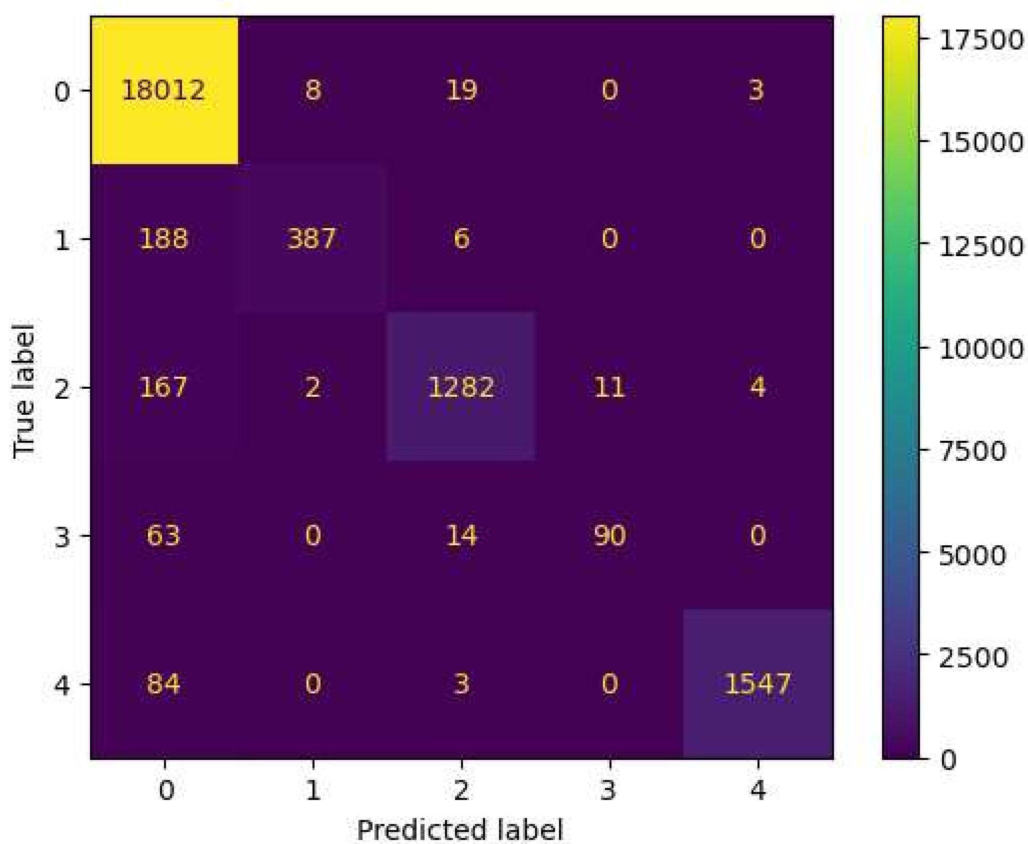
```
 Validation F-score: 97.39%
```

In [96]:
```python
# Generate a classification report
report = classification_report(y_val, y_validation_prediction)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99     18042
           1       0.97      0.67      0.79       581
           2       0.97      0.87      0.92      1466
           3       0.89      0.54      0.67       167
           4       1.00      0.95      0.97      1634

    accuracy                           0.97     21890
   macro avg       0.96      0.80      0.87     21890
weighted avg       0.97      0.97      0.97     21890
```

In [97]:
```python
# Plot the confusion matrix
conmat = confusion_matrix(y_val, y_validation_prediction, labels=clf.class
display = ConfusionMatrixDisplay(confusion_matrix=conmat, display_labels=c
display.plot()
```

Out[97]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20501a43190>



MLP

In [98]:
```
# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
```

In [99]:
```
clfMLP = MLPClassifier(hidden_layer_sizes=(100, 500,300,500, 100), activat
```

In [100]:
```
clfMLP.fit(X_train, y_train)
```

Out[100]:
```
MLPClassifier
MLPClassifier(alpha=0.25, hidden_layer_sizes=(100, 500, 300, 500, 100),
              max_iter=2000)
```

In [ ]:

In [101]:
```
# Make predictions on the test data
y_prediction = clfMLP.predict(X_test)
```

In [102]:
```
train_pred = clfMLP.predict(X_train)
```

In [103]:
```
train_acc_MLP = accuracy_score( y_train, train_pred) * 100
print(f'train Accuracy: {train_acc_MLP:.2f}%')
```

```
train Accuracy: 98.28%
```

In [ ]:

In [104]:
```
accMLP = accuracy_score(y_test, y_prediction) * 100
print(f'test Accuracy: {accMLP :.2f}%')
```

```
test Accuracy: 97.80%
```

In [105]:
```
recall_MLP = recall_score(y_test, y_prediction, average = 'micro') * 100
print(f' MLP Recall: {recall_MLP :.2f}%')

precision_MLP = precision_score(y_test, y_prediction, average = 'micro' )
print(f' MLP precision: {precision_MLP :.2f}%')

F_score_MLP = f1_score(y_test, y_prediction, average = 'micro' ) * 100
print(f' MLP F-score: {F_score_MLP :.2f}%')
```

```
 MLP Recall: 97.80%
 MLP precision: 97.80%
 MLP F-score: 97.80%
```

In [106]:

```python
# Generate a classification report
report_MLP = classification_report(y_test, y_prediction)
print(report_MLP)
```

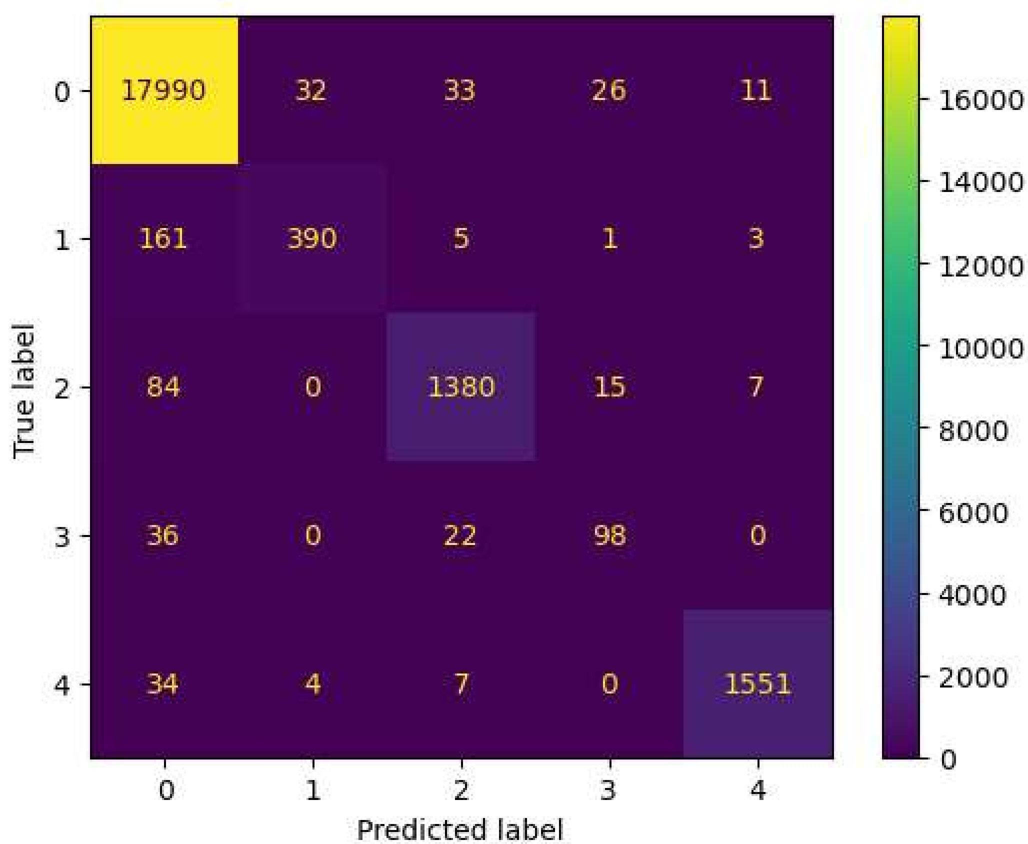
```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18092
           1       0.92      0.70      0.79       560
           2       0.95      0.93      0.94      1486
           3       0.70      0.63      0.66       156
           4       0.99      0.97      0.98      1596

    accuracy                           0.98     21890
   macro avg       0.91      0.84      0.87     21890
weighted avg       0.98      0.98      0.98     21890
```

In [107]:

```python
# Plot the confusion matrix
conmat_MLP = confusion_matrix(y_test, y_prediction, labels=clfMLP.classes_
display = ConfusionMatrixDisplay(confusion_matrix=conmat_MLP, display_labe
display.plot()
```

Out[107]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20512
db4d90>
```



SVM

In [57]:
```python
from sklearn.svm import SVC
```

In [58]:
```python
clf_SVC = SVC(kernel = 'poly')
```

In [59]:
```python
clf_SVC.fit(X_train, y_train)
```

Out[59]:
```
▾ SVC
SVC(kernel='poly')
```

In [60]:
```python
y_pred = clf_SVC.predict(X_test)
y_pred
```

Out[60]:
```
array([0, 0, 0, ..., 2, 2, 0])
```

In [61]:
```python
training_pred = clf_SVC.predict(X_train)
```

In [62]:
```python
training_accuracy_SVC = accuracy_score( y_train, training_pred) * 100
print(f'train Accuracy: {training_accuracy_SVC:.2f}%')
```

```
train Accuracy: 96.97%
```

In [64]:
```python
acc_SVC = accuracy_score(y_test, y_pred) * 100
print(f'test Accuracy: {acc_SVC :.2f}%')
```

```
test Accuracy: 96.53%
```

In [65]:
```python
recall_SVC = recall_score(y_test, y_pred, average = 'micro') * 100
print(f' MLP Recall: {recall_SVC :.2f}%')

precision_SVC = precision_score(y_test, y_pred, average = 'micro' ) * 100
print(f' MLP precision: {precision_SVC :.2f}%')

F_score_SVC = f1_score(y_test, y_pred, average = 'micro' ) * 100
print(f' MLP F-score: {F_score_SVC :.2f}%')
```

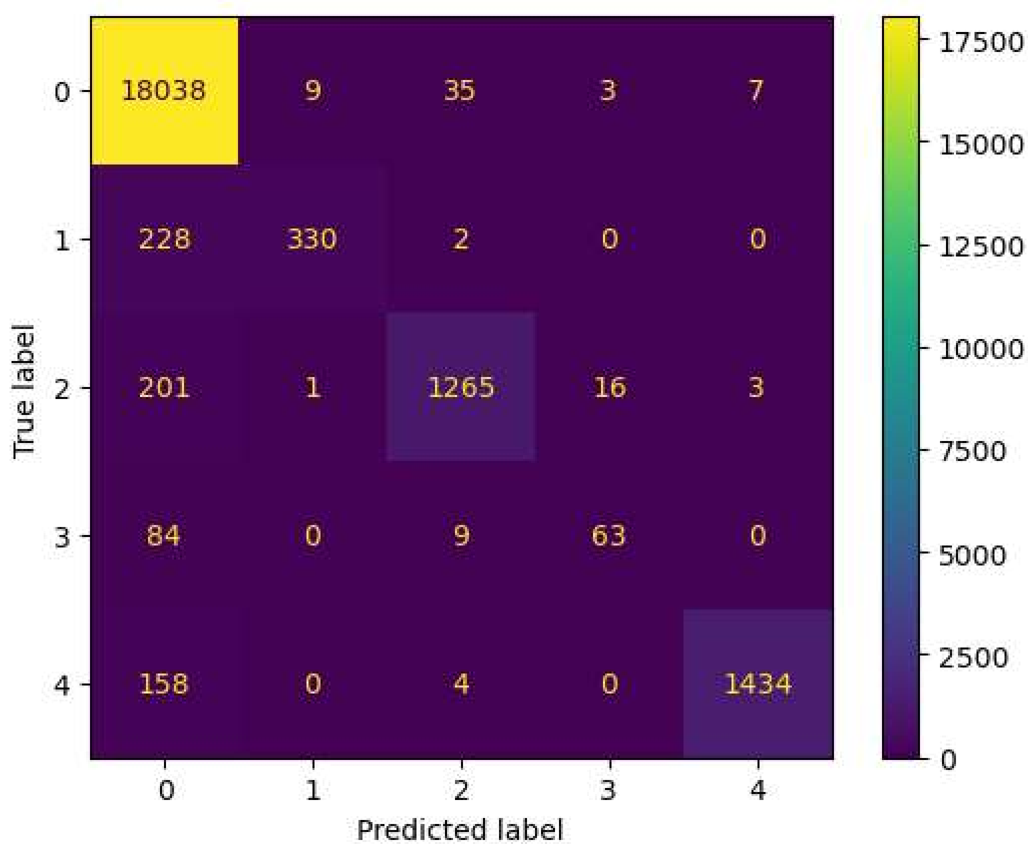
```
 MLP Recall: 96.53%
 MLP precision: 96.53%
 MLP F-score: 96.53%
```

In [66]:
```python
# Generate a classification report
report_SVC = classification_report(y_test, y_pred)
print(report_SVC)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     18092
           1       0.97      0.59      0.73       560
           2       0.96      0.85      0.90      1486
           3       0.77      0.40      0.53       156
           4       0.99      0.90      0.94      1596

    accuracy                           0.97     21890
   macro avg       0.93      0.75      0.82     21890
weighted avg       0.96      0.97      0.96     21890
```

In [67]:
```python
# Plot the confusion matrix
conmat_SVC = confusion_matrix(y_test, y_pred, labels=clf_SVC.classes_)
display = ConfusionMatrixDisplay(confusion_matrix=conmat_SVC, display_labe
display.plot()
```

Out[67]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20501
790d90>



Comparing the performance metrics on all the models

In [108]:

```
#table of metrics
model_eval= {'models':['MLP', 'SVM', 'RFS'],
'Accuracies':[accMLP,  acc_SVC, acc],
 'precisions':[precision_MLP, precision_SVC, precision_RF],
'recalls': [recall_MLP, recall_SVC, recall_RF],
'f_scores' : [F_score_MLP, F_score_SVC, F_score_RF]
}

model_table = pd.DataFrame(model_eval)
model_table
```

Out[108]:

	models	Accuracies	precisions	recalls	f_scores
0	MLP	97.802650	97.802650	97.802650	97.802650
1	SVM	96.528095	96.528095	96.528095	96.528095
2	RFS	97.386935	97.386935	97.386935	97.386935

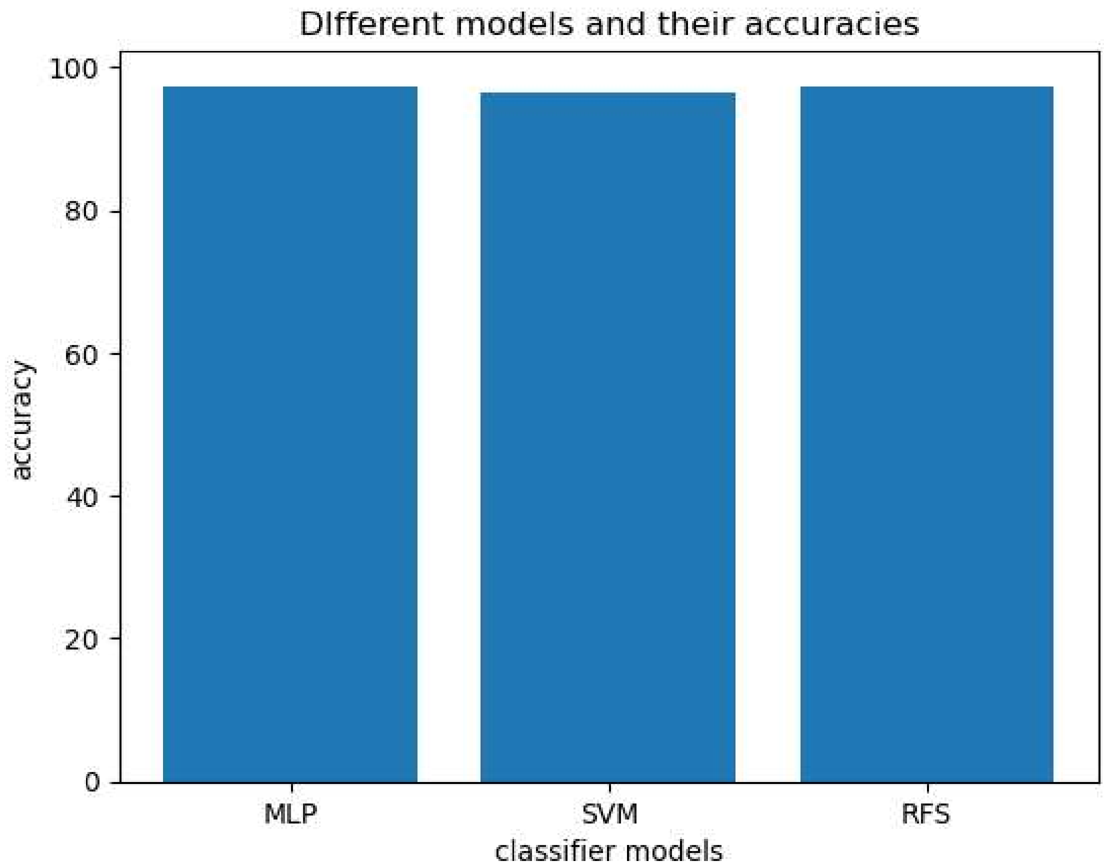
Accuracy

In [70]:

```
testAcc = [accMLP,  acc_SVC, acc]
precisions =[precision_MLP, precision_SVC, precision_RF]
recalls= [recall_MLP, recall_SVC, recall_RF]
f_scores = [F_score_MLP, F_score_SVC, F_score_RF]
```

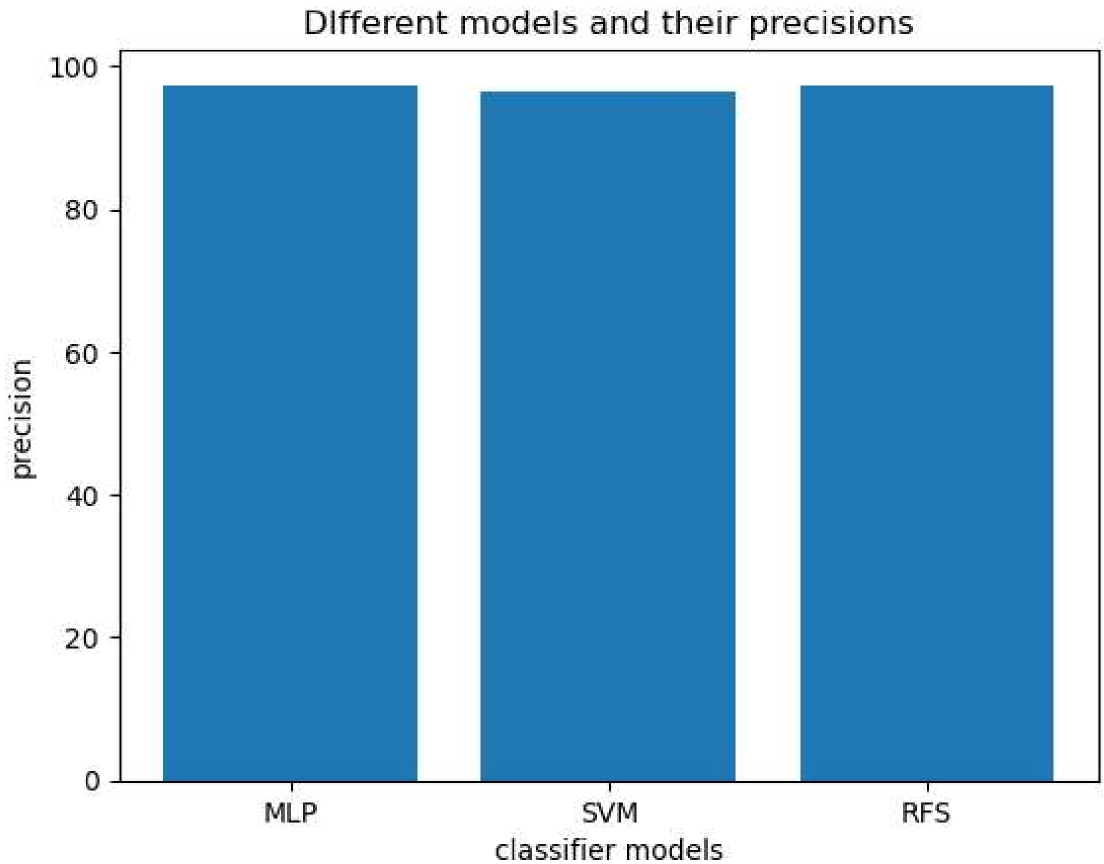
In [71]:

```
models = ['MLP', 'SVM', 'RFS']
plt.bar(models, testAcc)
plt.xlabel('classifier models')
plt.ylabel('accuracy')
plt.title('DIfferent models and their accuracies')
plt.show()
```



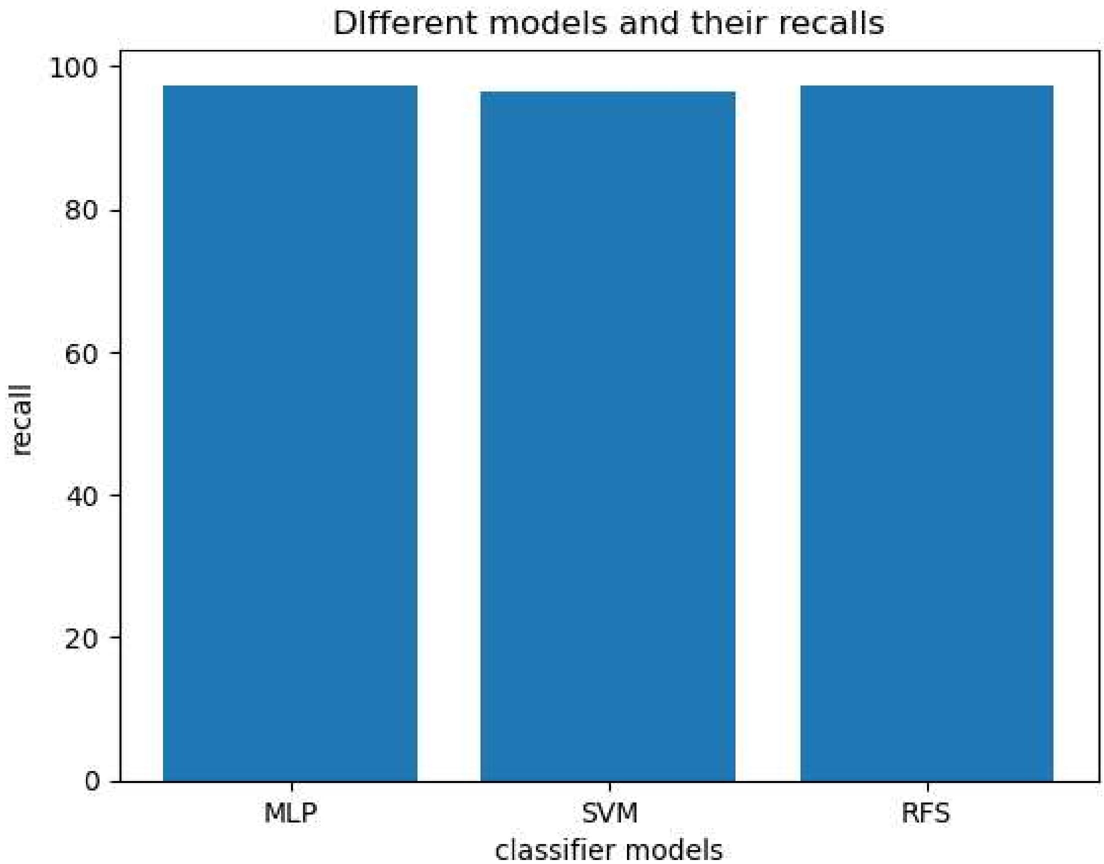
Precision

In [80]:
```
models = ['MLP', 'SVM', 'RFS']
plt.bar(models, precisions)
plt.xlabel('classifier models')
plt.ylabel('precision')
plt.title('DIfferent models and their precisions')
plt.show()
```



Recall

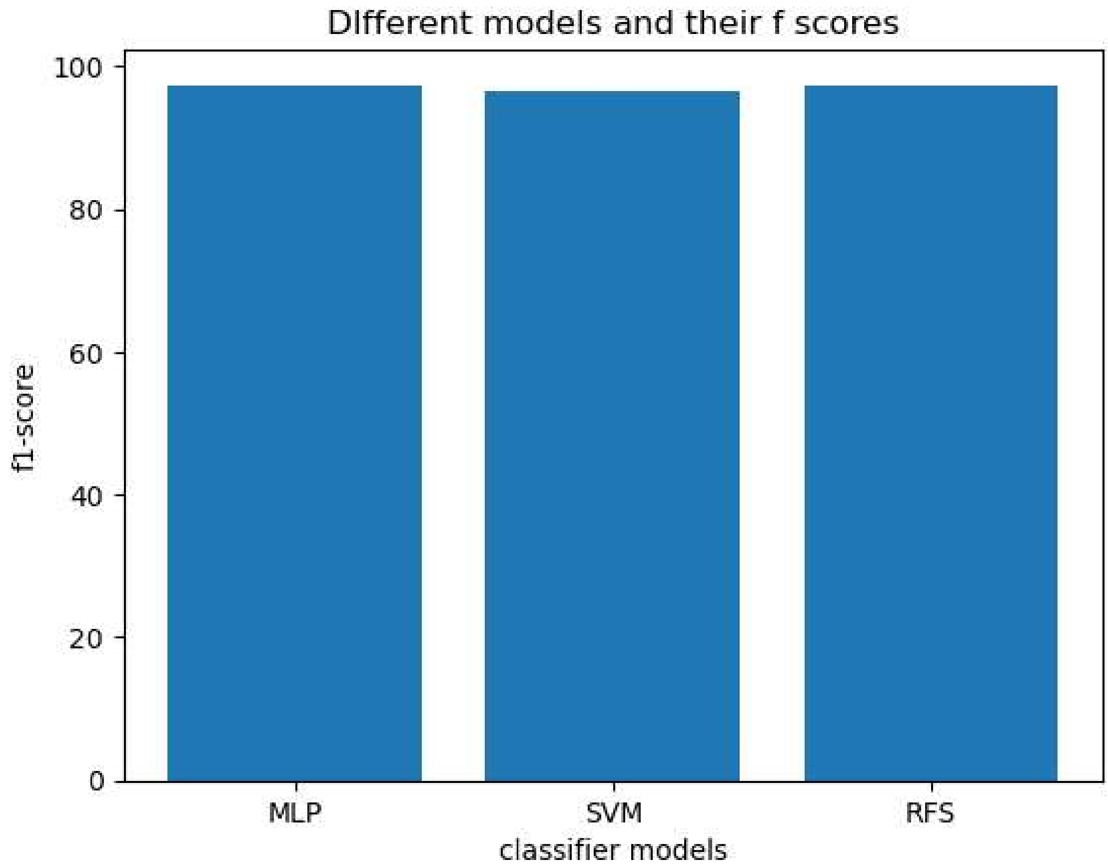
In [81]:
```
models = ['MLP', 'SVM', 'RFS']
plt.bar(models, recalls)
plt.xlabel('classifier models')
plt.ylabel('recall')
plt.title('DIfferent models and their recalls')
plt.show()
```



F-score

In [82]:

```
models = ['MLP', 'SVM', 'RFS']
plt.bar(models, f_scores)
plt.xlabel('classifier models')
plt.ylabel('f1-score')
plt.title('DIfferent models and their f scores')
plt.show()
```



In [ ]:

```

```