ECG Heartbeat Classification Datset

This notebook involves the making of machine learning models to classify the given data of obtained as an heartbeat ECG into differen classes. We'll undergo machine learning processes to classify them. AS given in the dataset, we are given 5 different classes of heartbeat as [N:0, S:1, V:2 , F:3, Q:4]

- N: Non-Ectopic Beats
- S: Superventrical Ectopic Beats
- V: Ventricular Ectopic Beats
- F: Fusion Beats
- Q: Unknown Beats

The **CNN Algotithm** that we'll implement will classigy the given heartbeat into one of these classes

In [1]:

```
# importing libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os, tqdm, re, time, itertools, sys
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matri
from keras.layers import Conv2D, Conv1D, MaxPooling2D, MaxPooling1D, Flatten, Batc
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.callbacks import CSVLogger, ModelCheckpoint
import matplotlib.pyplot as plt
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

Loading the Data

The first step is to load the data in our memory. We'll load all data provided to us in our notebook, and then start the machine learning process

In [3]:

```
# loading all data files into memory
start = time.time()

data_train = pd.read_csv('../input/heartbeat/mitbih_train.csv', header=None)
data_test = pd.read_csv('../input/heartbeat/mitbih_test.csv', header=None)
abnormal = pd.read_csv('../input/heartbeat/ptbdb_abnormal.csv', header=None)
normal = pd.read_csv('../input/heartbeat/ptbdb_normal.csv', header=None)

end = time.time()
print('Time taken: %.3f seconds' % (end-start))

print('Data loaded........')
```

```
Time taken: 10.418 seconds
Data loaded........
```

In [4]:
```
normal = normal.drop([187], axis=1)
abnormal = abnormal.drop([187], axis=1)
```

EDA (Exploratory Data Analysis)

In this step, we will undergo an EDA (Exploratory Data Analysis) to get brief understanding of our data. We are given a data concerned with the ECG of a patient, classified into normal and abnormal classes. We'll make some plots to see the variations in the heart rate of a patient with normal and abnormal ECG.

In [5]:
```
data_train.isnull().sum().to_numpy()
```

Out[5]:
```
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])
```

In [6]:
```
# checking the dataset shape
abnormal.shape, normal.shape
```

Out[6]:
```
((10506, 187), (4046, 187))
```

We have a total of **10506** rows and **188** columns for abormal & **4045** rows and **188** columns of a normal ECG in our data

In [7]:
```
# view first 4 rows of data
data_train.head(4)
```

Out[7]:

	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	0.049(
1	0.960114	0.863248	0.461538	0.196581	0.094017	0.125356	0.099715	0.088319	0.074074	0.082(
2	1.000000	0.659459	0.186486	0.070270	0.070270	0.059459	0.056757	0.043243	0.054054	0.045(
3	0.925414	0.665746	0.541436	0.276243	0.196133	0.077348	0.071823	0.060773	0.066298	0.058(

4 rows × 188 columns

As it can be seen, the data is composed of columns (features) that contain the floating point numbers that represent the heart rate.

In [8]:
```
# checking the columns
data_train.columns
```

Out[8]:
```
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 [9]:
```
abnormal.shape, normal.shape
```

Out[9]:
```
((10506, 187), (4046, 187))
```

In [10]:
```
# view first 2 rows of abnormal ECG data
abnormal.head(2)
```

Out[10]:

	0	1	2	3	4	5	6	7	8	
0	0.932233	0.869679	0.886186	0.929626	0.908775	0.933970	0.801043	0.749783	0.687229	0.635
1	1.000000	0.606941	0.384181	0.254237	0.223567	0.276836	0.253430	0.184826	0.153349	0.121

2 rows × 187 columns

In [11]:
```
# view first 2 rows of normal data|
normal.head(2)
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	.
0	1.0	0.900324	0.358590	0.051459	0.046596	0.126823	0.133306	0.119125	0.110616	0.113047	.
1	1.0	0.794681	0.375387	0.116883	0.000000	0.171923	0.283859	0.293754	0.325912	0.345083	.

2 rows × 187 columns

In [12]:
```
flatten_y = abnormal.values
flatten_y = flatten_y[:, 5:70].flatten()
flatten_y
```

Out[12]:
```
array([0.93397045, 0.80104256, 0.7497828 , ..., 0.06976745, 0.06078224,
       0.06606765])
```

Data Visualization

For better comprehension, we'll plot the data of normal and abnormal ECG rate to see how the curves look like. Given below are some plots of normal and abnormal ECG rate.

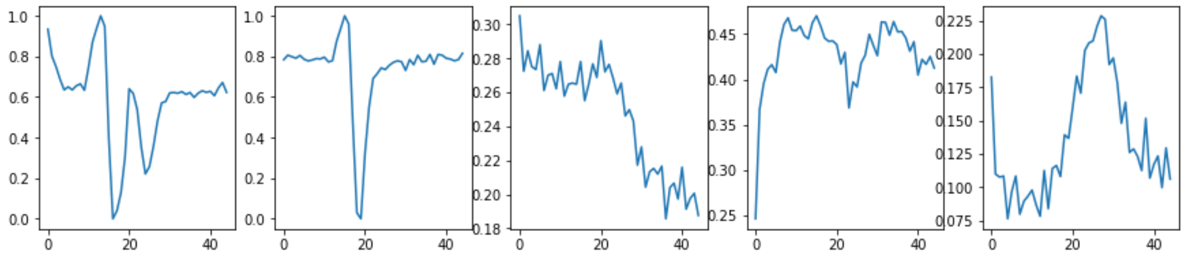
Abormal ECG Visualization

Below are some plots showing the ECG Curve of those persons who have an abnormal ECG rate

In [13]:
```
plt.figure(figsize=(15, 3))
plt.title('ECG Visualization of Abormal Persons')
plt.subplot(1, 5, 1)
plt.plot(abnormal.values[0][5:50])
plt.subplot(1, 5, 2)
plt.plot(abnormal.values[10][5:50])
plt.subplot(1, 5, 3)
plt.plot(abnormal.values[20][5:50])
```

```
plt.subplot(1, 5, 4)
plt.plot(abnormal.values[40][5:50])
plt.subplot(1, 5, 5)
plt.plot(abnormal.values[44][5:50])
```

Out[13]: [<matplotlib.lines.Line2D at 0x7f880af2e790>]

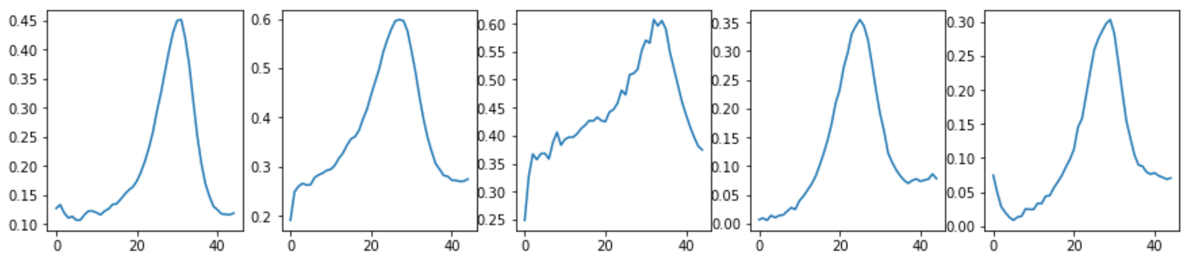


Normal ECG Visualization

Below are the graphs showing the ECG rate of normal persons

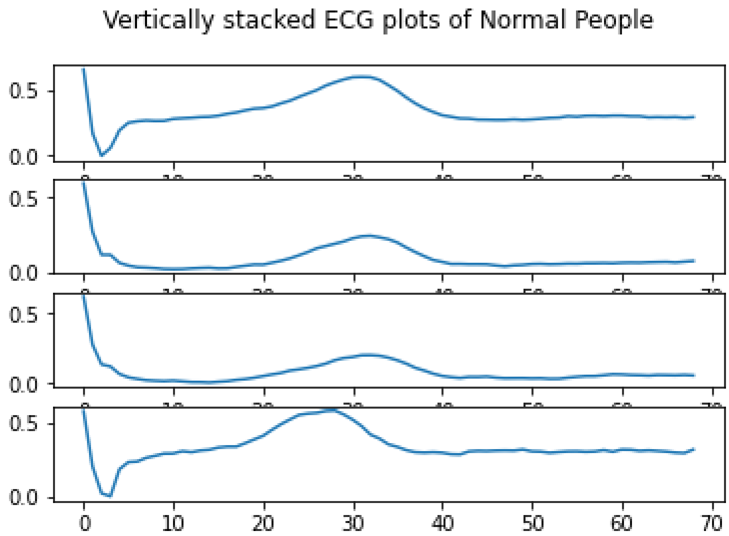
In [14]:
```
plt.figure(figsize=(15, 3))
plt.title('ECG Visualization of Normal Persons')
plt.subplot(1, 5, 1)
plt.plot(normal.values[0][5:50])
plt.subplot(1, 5, 2)
plt.plot(normal.values[10][5:50])
plt.subplot(1, 5, 3)
plt.plot(normal.values[20][5:50])
plt.subplot(1, 5, 4)
plt.plot(normal.values[40][5:50])
plt.subplot(1, 5, 5)
plt.plot(normal.values[77][5:50])
```

Out[14]: [<matplotlib.lines.Line2D at 0x7f880ab25a90>]



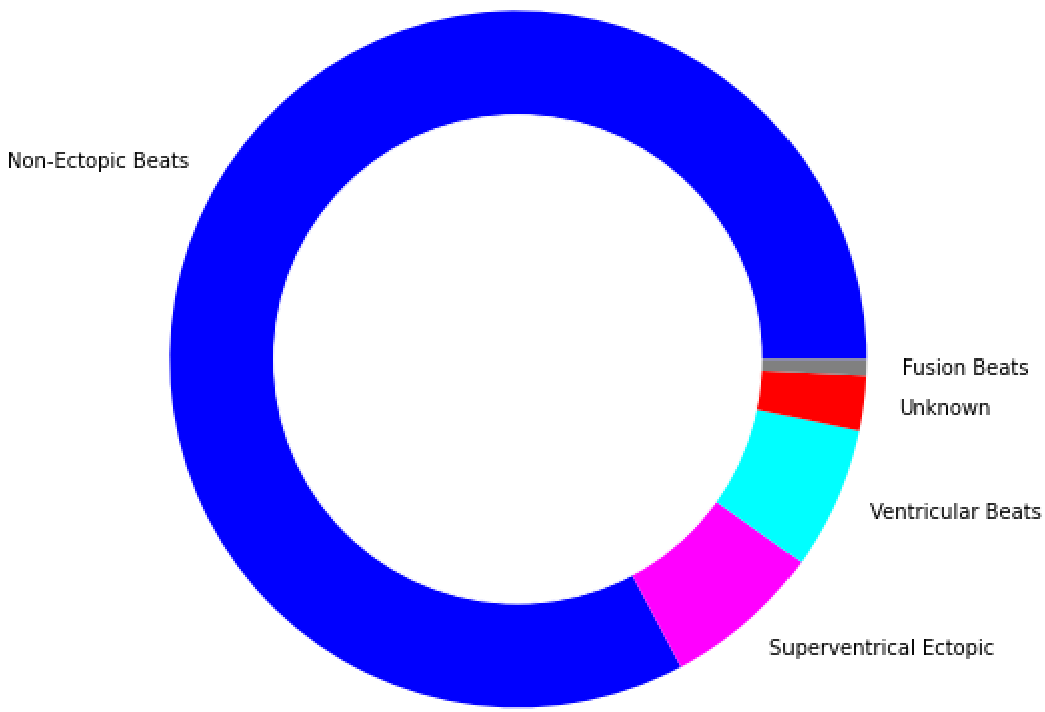
In [15]:
```
fig, axs = plt.subplots(4)
fig.suptitle('Vertically stacked ECG plots of Normal People')
axs[0].plot(normal.values[10][1:70])
axs[1].plot(normal.values[55][1:70])
axs[2].plot(normal.values[87][1:70])
axs[3].plot(normal.values[98][1:70])
```

Out[15]: [<matplotlib.lines.Line2D at 0x7f880a91c850>]



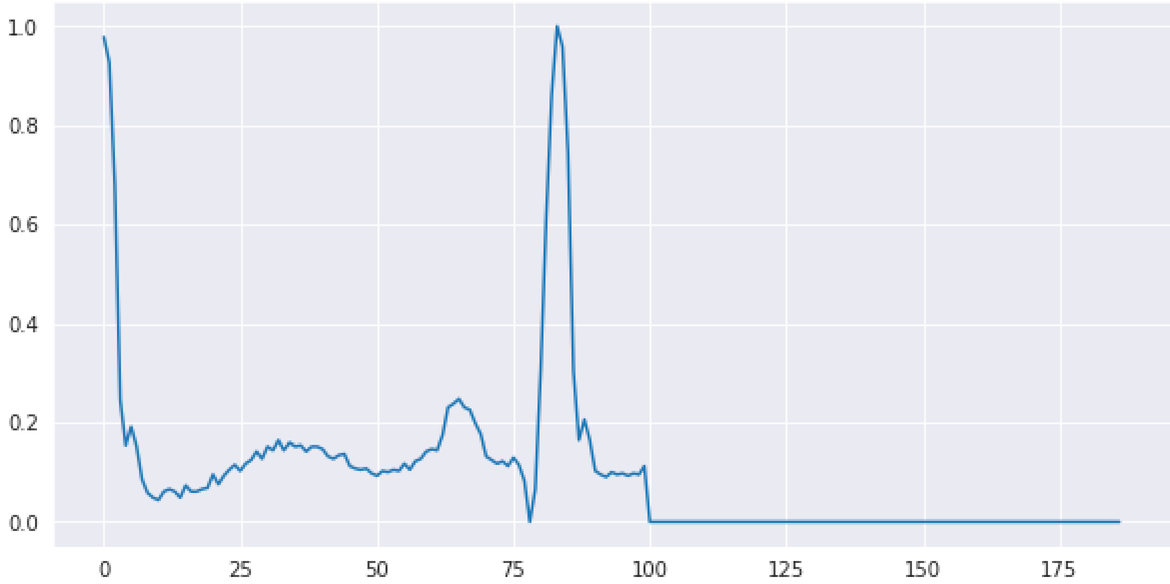
In [16]:
```
# viewing the distribution of beats in our dataset
plt.figure(figsize=(10, 8))
circle = plt.Circle((0, 0), 0.7, color='white')
plt.pie(data_train[187].value_counts(), labels=['Non-Ectopic Beats', 'Superventrica
                                                'Unknown', 'Fusion Beats'], colors

p = plt.gcf()
p.gca().add_artist(circle)
```

Out[16]:
```
<matplotlib.patches.Circle at 0x7f880a8c09d0>
```



In [17]:
```
sns.set_style('darkgrid')
plt.figure(figsize=(10, 5))
plt.plot(data_train.iloc[0, 0:187])
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x7f880a7fc7d0>]
```



Conclusion:

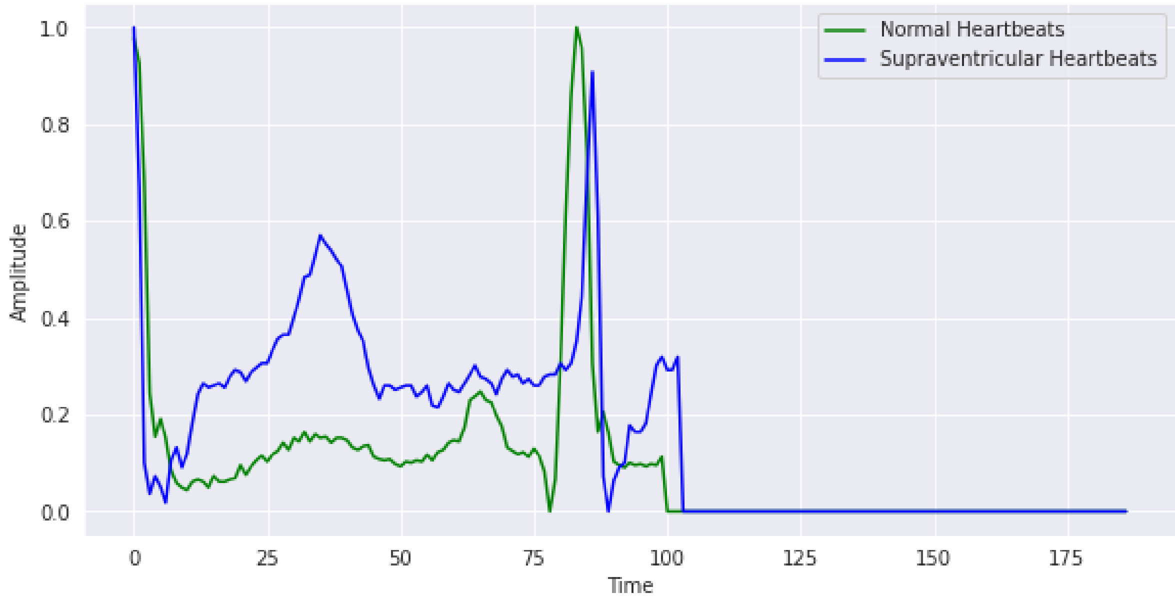
We can conclude from above figures that the persons having normal ECG rate, the figures are following a **bell-curve** pattern. The ECG of abnormal persons show othe types of curves. We'll use this information to make our machine learning model for classification.

Data Preprocessing

At this stage, we'll undergo some data preprocessing process to see if the data needs to be cleaned. Cleaned data is requried for model fitting in next phases.

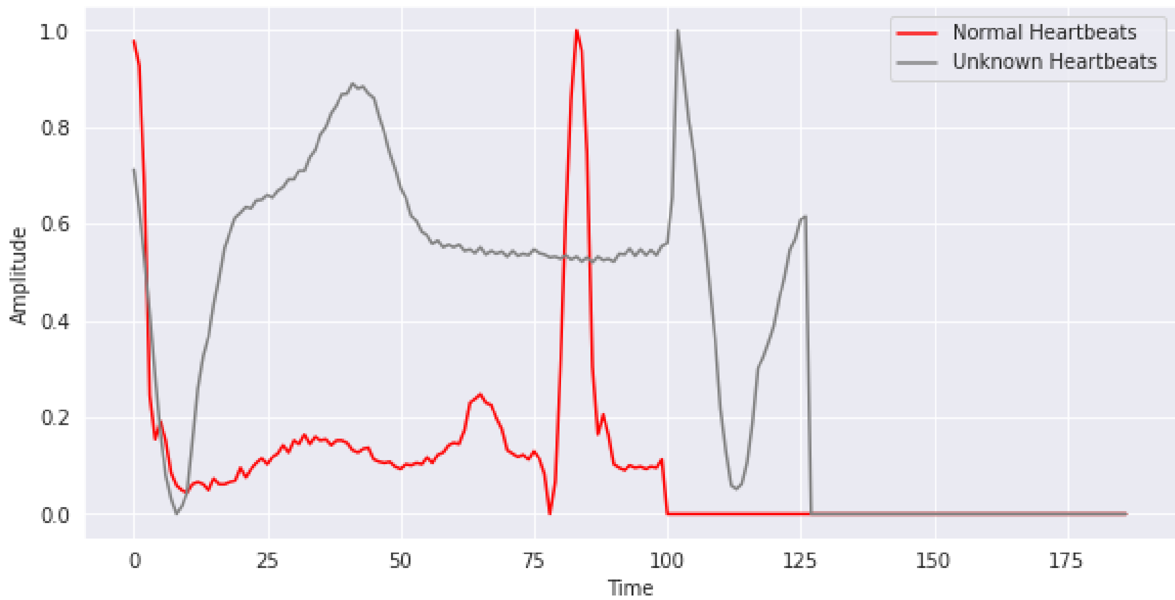
In [18]:
```
# making the class labels for our dataset
data_1 = data_train[data_train[187] == 1]
data_2 = data_train[data_train[187] == 2]
data_3 = data_train[data_train[187] == 3]
data_4 = data_train[data_train[187] == 4]

sns.set_style('darkgrid')
plt.figure(figsize=(10, 5))
plt.plot(data_train.iloc[0, 0:187], color='green', label='Normal Heartbeats')
plt.plot(data_1.iloc[0, 0:187], color='blue', label='Supraventricular Heartbeats')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
```

Out[18]:
```
<matplotlib.legend.Legend at 0x7f880a490690>
```



In [19]:
```
sns.set_style('darkgrid')
plt.figure(figsize=(10, 5))
plt.plot(data_train.iloc[0, 0:187], color='red', label='Normal Heartbeats')
plt.plot(data_4.iloc[0, 0:187], color='grey', label='Unknown Heartbeats')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x7f880a476f90>
```



In [20]:
```
y_abnormal = np.ones(abnormal.shape[0])
y_abnormal = pd.DataFrame(y_abnormal)

y_normal = np.zeros(normal.shape[0])
y_normal = pd.DataFrame(y_normal)

# merging the original dataframe
X = pd.concat([abnormal, normal], sort=True)
y = pd.concat([y_abnormal, y_normal], sort=True)
```

In [21]:
```
print(X.shape)
print(y.shape)
```

```
(14552, 187)
(14552, 1)
```

In [22]:
```
# checking if there are some null values in data
normal.isnull().sum().to_numpy()
```

Out[22]:
```
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])
```

In [23]:
```
# checking if there are some null values in abnormal patient data
abnormal.isnull().sum().to_numpy()
```

Out[23]:
```
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])
```

The output of above cell shows that there are no null values in our data, and the data can now be used for model fitting. We have two types of datasets, normal and abnormal, and they'll be used for model fitting.

Data Argumentation

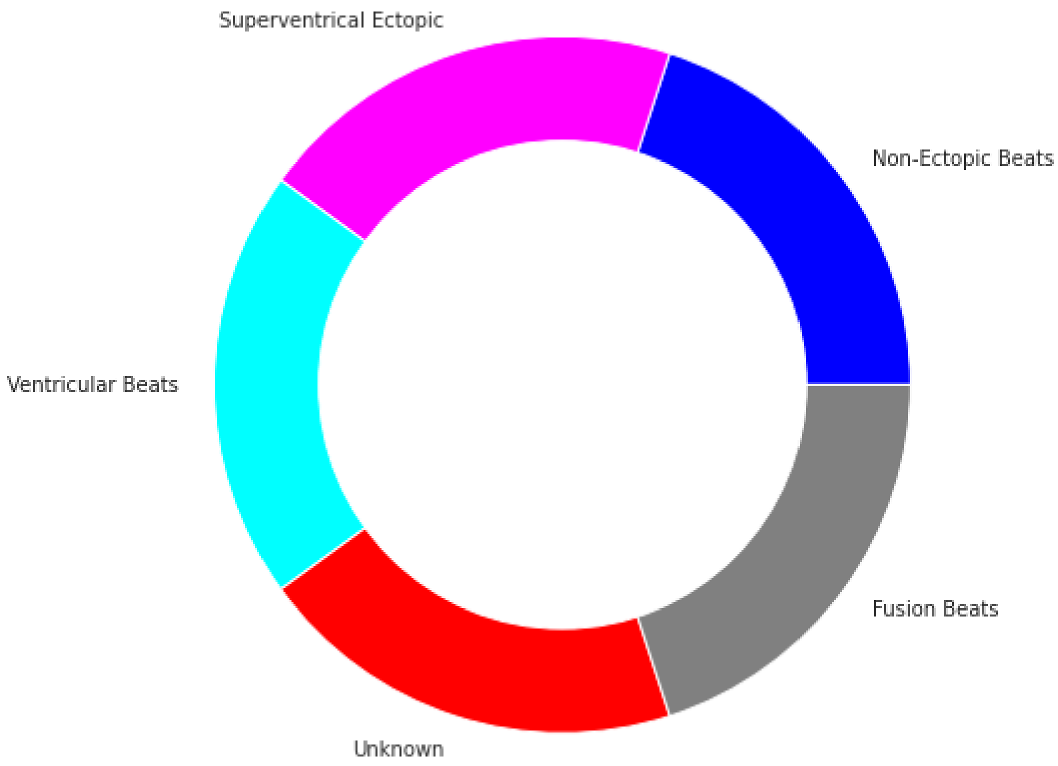
Since our data in biased, we need to use data argumentation on it so that we can remove bias from data and make equal distributions.

In [24]:
```
from sklearn.utils import resample
data_1_resample = resample(data_1, n_samples=20000,
                           random_state=123, replace=True)
data_2_resample = resample(data_2, n_samples=20000,
                           random_state=123, replace=True)
data_3_resample = resample(data_3, n_samples=20000,
                           random_state=123, replace=True)
data_4_resample = resample(data_4, n_samples=20000,
                           random_state=123, replace=True)
data_0 = data_train[data_train[187] == 0].sample(n=20000, random_state=123)
```

In [25]:
```
train_dataset = pd.concat([data_0, data_1_resample, data_2_resample, data_3_resamp
                          data_4_resample])
```

In [26]:
```
# viewing the distribution of beats in our dataset
plt.figure(figsize=(10, 8))
circle = plt.Circle((0, 0), 0.7, color='white')
plt.pie(train_dataset[187].value_counts(), labels=['Non-Ectopic Beats', 'Supervent
                                                   'Unknown', 'Fusion Beats'], colors

p = plt.gcf()
p.gca().add_artist(circle)
```

Out[26]:
```
<matplotlib.patches.Circle at 0x7f880740a2d0>
```



Making X & Y Variables

In [27]:
```
target_train = train_dataset[187]
target_test = data_test[187]
target_train.unique()
```

Out[27]:
```
array([0., 1., 2., 3., 4.])
```

In [28]:
```
y_train = to_categorical(target_train)
y_test = to_categorical(target_test)
y_train[:4]
```

Out[28]:
```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.]], dtype=float32)
```

Data Splicing

This stage involves the data split into train & test sets. The training data will be used for training our model, and the testing data will be used to check the performance of model on unseen dataset. We're using a split of **80-20**, i.e., **80%** data to be used for training & **20%** to be used for testing purpose.

In [29]:
```
# making train & test splits
X_train = train_dataset.iloc[:, :-1].values
X_test = data_test.iloc[:, :-1].values
```

In [30]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(100000, 187)
(21892, 187)
(100000, 5)
(21892, 5)
```

Applying the Model

We are making use of following models to make our classification:

- Random Forest Classification
- Support Vector Machines (SVM)
- Convolutional Neural Network (CNN)

Steps:

- We will instantiate the model
- After intantiation, the model will be fit to training data
- After then, the model will be tested on useen data to make predictions

Convolutional Neural Network (CNN)

We will apply the CNN algorithm to our data to generate prediction results. First, we need to reshape our data for CNN. We will use 1-dimensional CNN for our model, reshaping our data as per the dimensins of our CNN>

In [31]:
```
X_train = X_train.reshape(len(X_train), X_train.shape[1], 1)
X_test = X_test.reshape(len(X_test), X_test.shape[1], 1)
X_train.shape, X_test.shape
```

Out[31]:
```
((100000, 187, 1), (21892, 187, 1))
```

In [32]:
```
# making the deep learning function
def model():
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=6, activation='relu',
                    padding='same', input_shape=(187, 1)))
    model.add(BatchNormalization())

    # adding a pooling layer
    model.add(MaxPooling1D(pool_size=(3), strides=2, padding='same'))

    model.add(Conv1D(filters=64, kernel_size=6, activation='relu',
                    padding='same', input_shape=(187, 1)))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(3), strides=2, padding='same'))

    model.add(Conv1D(filters=64, kernel_size=6, activation='relu',
                    padding='same', input_shape=(187, 1)))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(3), strides=2, padding='same'))
```

```
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(5, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accu
    return model
```

In [33]:
```
model = model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 187, 64)           448
_________________________________________________________________
batch_normalization (BatchNo (None, 187, 64)           256
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 94, 64)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 94, 64)            24640
_________________________________________________________________
batch_normalization_1 (Batch (None, 94, 64)            256
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 47, 64)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 47, 64)            24640
_________________________________________________________________
batch_normalization_2 (Batch (None, 47, 64)            256
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 24, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 64)                98368
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 325
=================================================================
Total params: 153,349
Trainable params: 152,965
Non-trainable params: 384
_________________________________________________________________
```

In [34]:
```
logger = CSVLogger('logs.csv', append=True)
his = model.fit(X_train, y_train, epochs=50, batch_size=32,
          validation_data=(X_test, y_test), callbacks=[logger])
```

```
Epoch 1/50
3125/3125 [==============================] - 22s 5ms/step - loss: 0.2956 - accurac
y: 0.8936 - val_loss: 0.2277 - val_accuracy: 0.9227
Epoch 2/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0843 - accurac
y: 0.9701 - val_loss: 0.0985 - val_accuracy: 0.9717
Epoch 3/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0533 - accurac
y: 0.9816 - val_loss: 0.1062 - val_accuracy: 0.9693
Epoch 4/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0383 - accurac
y: 0.9869 - val_loss: 0.1110 - val_accuracy: 0.9709
Epoch 5/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0342 - accurac
y: 0.9886 - val_loss: 0.1410 - val_accuracy: 0.9621
Epoch 6/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0274 - accurac
y: 0.9914 - val_loss: 0.1396 - val_accuracy: 0.9658
Epoch 7/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0221 - accurac
y: 0.9931 - val_loss: 0.1554 - val_accuracy: 0.9691
Epoch 8/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0202 - accurac
y: 0.9941 - val_loss: 0.1255 - val_accuracy: 0.9741
Epoch 9/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0178 - accurac
y: 0.9946 - val_loss: 0.1475 - val_accuracy: 0.9725
Epoch 10/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0166 - accurac
y: 0.9949 - val_loss: 0.1853 - val_accuracy: 0.9719
Epoch 11/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0160 - accurac
y: 0.9955 - val_loss: 0.1161 - val_accuracy: 0.9801
Epoch 12/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0131 - accurac
y: 0.9960 - val_loss: 0.1270 - val_accuracy: 0.9788
Epoch 13/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0125 - accurac
y: 0.9964 - val_loss: 0.1431 - val_accuracy: 0.9751
Epoch 14/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0095 - accurac
y: 0.9971 - val_loss: 0.1400 - val_accuracy: 0.9794
Epoch 15/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0126 - accurac
y: 0.9967 - val_loss: 0.1305 - val_accuracy: 0.9807
Epoch 16/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0094 - accurac
y: 0.9969 - val_loss: 0.1554 - val_accuracy: 0.9785
Epoch 17/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0100 - accurac
y: 0.9971 - val_loss: 0.1505 - val_accuracy: 0.9799
Epoch 18/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0092 - accurac
y: 0.9971 - val_loss: 0.1334 - val_accuracy: 0.9820
Epoch 19/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0098 - accurac
y: 0.9976 - val_loss: 0.1724 - val_accuracy: 0.9768
Epoch 20/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0089 - accurac
y: 0.9975 - val_loss: 0.1640 - val_accuracy: 0.9767
Epoch 21/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0068 - accurac
y: 0.9978 - val_loss: 0.1602 - val_accuracy: 0.9791
Epoch 22/50
```

```
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0071 - accurac
y: 0.9980 - val_loss: 0.1505 - val_accuracy: 0.9819
Epoch 23/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0081 - accurac
y: 0.9978 - val_loss: 0.1842 - val_accuracy: 0.9732
Epoch 24/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0064 - accurac
y: 0.9982 - val_loss: 0.1602 - val_accuracy: 0.9829
Epoch 25/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0060 - accurac
y: 0.9983 - val_loss: 0.1750 - val_accuracy: 0.9785
Epoch 26/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0075 - accurac
y: 0.9985 - val_loss: 0.1884 - val_accuracy: 0.9789
Epoch 27/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0080 - accurac
y: 0.9979 - val_loss: 0.1721 - val_accuracy: 0.9808
Epoch 28/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0090 - accurac
y: 0.9978 - val_loss: 0.1575 - val_accuracy: 0.9827
Epoch 29/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0042 - accurac
y: 0.9987 - val_loss: 0.1721 - val_accuracy: 0.9803
Epoch 30/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0065 - accurac
y: 0.9982 - val_loss: 0.1672 - val_accuracy: 0.9810
Epoch 31/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0054 - accurac
y: 0.9986 - val_loss: 0.1958 - val_accuracy: 0.9794
Epoch 32/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0062 - accurac
y: 0.9986 - val_loss: 0.2086 - val_accuracy: 0.9756
Epoch 33/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0095 - accurac
y: 0.9978 - val_loss: 0.1930 - val_accuracy: 0.9781
Epoch 34/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0046 - accurac
y: 0.9988 - val_loss: 0.2047 - val_accuracy: 0.9772
Epoch 35/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0060 - accurac
y: 0.9984 - val_loss: 0.1807 - val_accuracy: 0.9803
Epoch 36/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0037 - accurac
y: 0.9990 - val_loss: 0.1912 - val_accuracy: 0.9804
Epoch 37/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0050 - accurac
y: 0.9986 - val_loss: 0.2132 - val_accuracy: 0.9751
Epoch 38/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0050 - accurac
y: 0.9986 - val_loss: 0.2197 - val_accuracy: 0.9780
Epoch 39/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0049 - accurac
y: 0.9989 - val_loss: 0.2026 - val_accuracy: 0.9808
Epoch 40/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0031 - accurac
y: 0.9991 - val_loss: 0.1995 - val_accuracy: 0.9802
Epoch 41/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0046 - accurac
y: 0.9988 - val_loss: 0.1768 - val_accuracy: 0.9810
Epoch 42/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0041 - accurac
y: 0.9990 - val_loss: 0.1886 - val_accuracy: 0.9808
Epoch 43/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0046 - accurac
```

```
y: 0.9988 - val_loss: 0.1712 - val_accuracy: 0.9813
Epoch 44/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0036 - accurac
y: 0.9990 - val_loss: 0.1804 - val_accuracy: 0.9809
Epoch 45/50
3125/3125 [==============================] - 14s 5ms/step - loss: 0.0044 - accurac
y: 0.9991 - val_loss: 0.1887 - val_accuracy: 0.9803
Epoch 46/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0024 - accurac
y: 0.9994 - val_loss: 0.2462 - val_accuracy: 0.9762
Epoch 47/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0057 - accurac
y: 0.9986 - val_loss: 0.2002 - val_accuracy: 0.9795
Epoch 48/50
3125/3125 [==============================] - 14s 4ms/step - loss: 0.0034 - accurac
y: 0.9993 - val_loss: 0.2047 - val_accuracy: 0.9795
Epoch 49/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0047 - accurac
y: 0.9989 - val_loss: 0.2159 - val_accuracy: 0.9787
Epoch 50/50
3125/3125 [==============================] - 15s 5ms/step - loss: 0.0033 - accurac
y: 0.9992 - val_loss: 0.2109 - val_accuracy: 0.9809
```

In [41]:

```
model.evaluate(X_test, y_test)
```

```
685/685 [==============================] - 2s 2ms/step - loss: 0.2109 - accuracy:
0.9809
```

Out[41]:

```
[0.2108655869960785, 0.9809062480926514]
```

Graphical Visualization of Predictions

In [42]:

```
history = his.history
history.keys()
```

Out[42]:

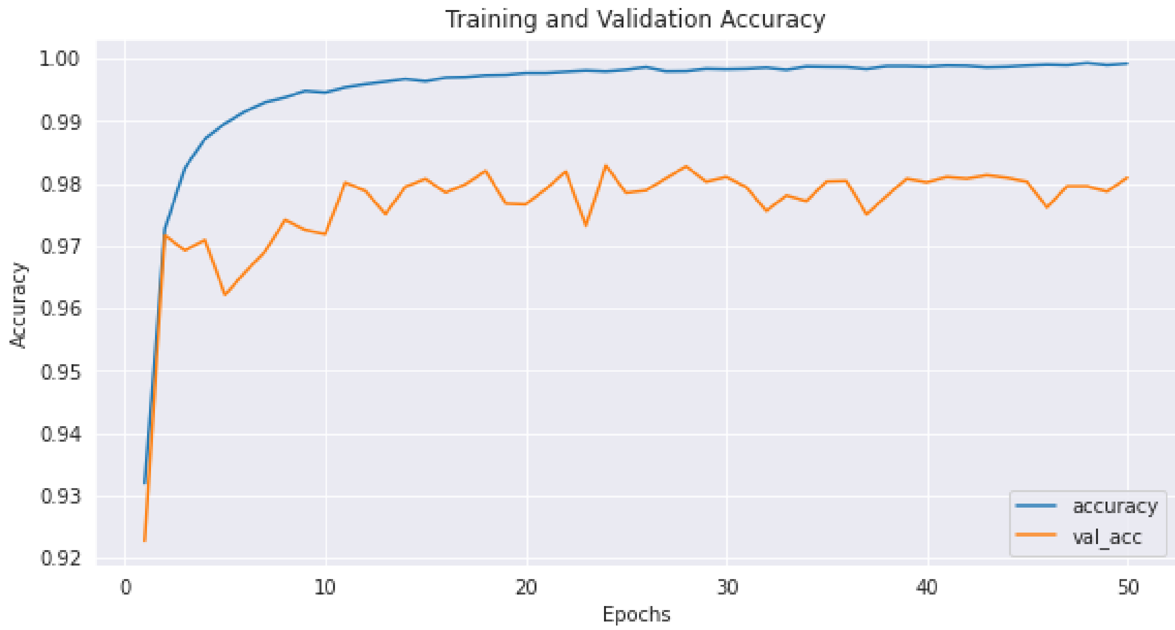
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

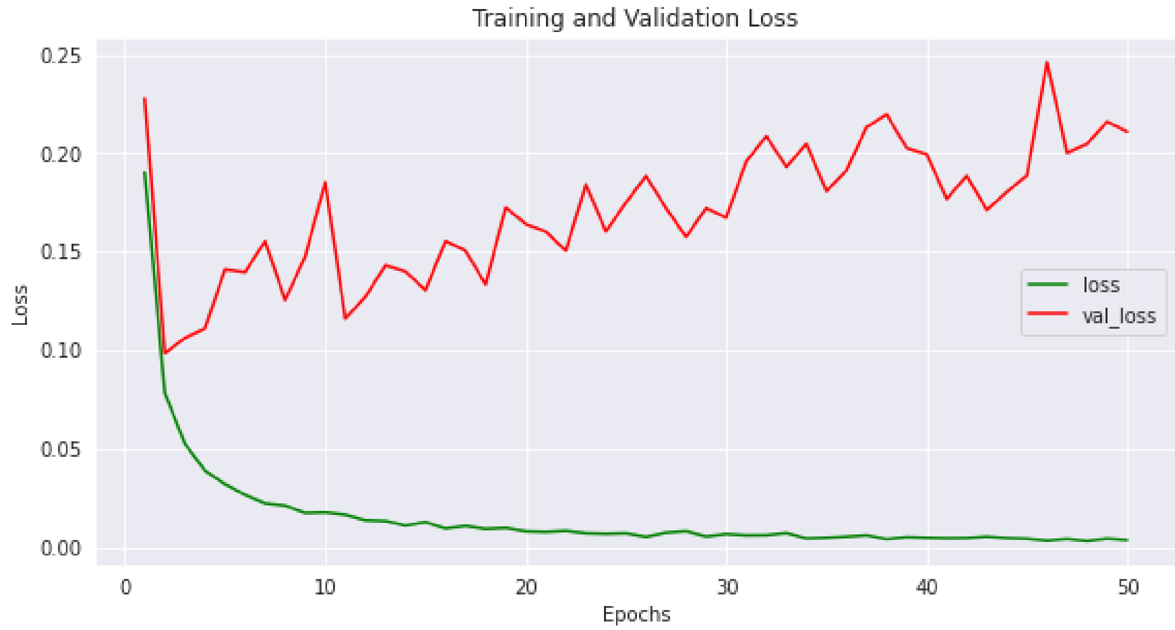
In [44]:

```
epochs = range(1, len(history['loss']) + 1)
acc = history['accuracy']
loss = history['loss']
val_acc = history['val_accuracy']
val_loss = history['val_loss']

plt.figure(figsize=(10, 5))
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.plot(epochs, acc, label='accuracy')
plt.plot(epochs, val_acc, label='val_acc')
plt.legend()

plt.figure(figsize=(10, 5))
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(epochs, loss, label='loss', color='g')
plt.plot(epochs, val_loss, label='val_loss', color='r')
plt.legend()
```

Out[44]:

```
<matplotlib.legend.Legend at 0x7f87de0e2990>
```



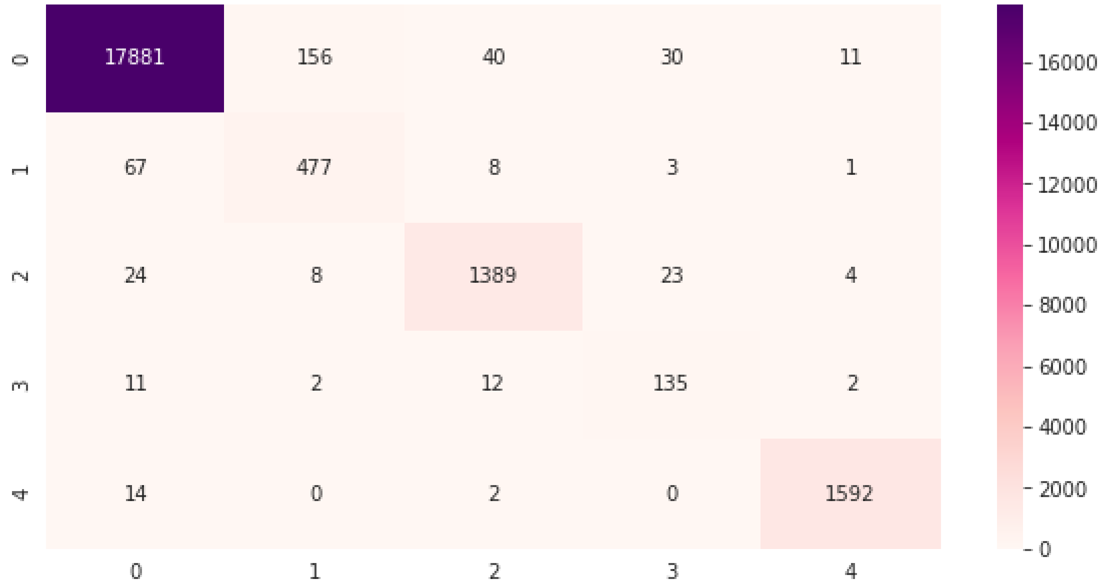


In [45]:
```
y_pred = model.predict(X_test)
y_hat = np.argmax(y_pred, axis = 1)
confusion_matrix(np.argmax(y_test, axis = 1), y_hat)
```

Out[45]:
```
array([[17881,   156,    40,    30,    11],
       [   67,   477,     8,     3,     1],
       [   24,     8,  1389,    23,     4],
       [   11,     2,    12,   135,     2],
       [   14,     0,     2,     0,  1592]])
```

In [46]:
```
plt.figure(figsize=(10, 5))
sns.heatmap(confusion_matrix(np.argmax(y_test, axis = 1), y_hat), annot=True, fmt=
```

Out[46]:
```
<AxesSubplot:>
```



1. Random Forest Classifier

In this section, we use Random Forest Classifier to fit to our training dataset & predict results.

In [40]:
```
# instantiate the classifier and fit to training data
start = time.time()

rf = RandomForestClassifier()
rf.fit(X_train, y_train)

end = time.time()
print('Time Taken: %.3f seconds' % (end-start))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-04bfba4042f5> in <module>
      3 
      4 rf = RandomForestClassifier()
----> 5 rf.fit(X_train, y_train)
      6 
      7 end = time.time()

/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in fit(self, X, y, sample_weight)
    302             )
    303         X, y = self._validate_data(X, y, multi_output=True,
--> 304                                    accept_sparse="csc", dtype=DTYPE)
    305         if sample_weight is not None:
    306             sample_weight = _check_sample_weight(sample_weight, X)

/opt/conda/lib/python3.7/site-packages/sklearn/base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    430                 y = check_array(y, **check_y_params)
    431             else:
--> 432                 X, y = check_X_y(X, y, **check_params)
    433             out = X, y
    434 

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    800                     ensure_min_samples=ensure_min_samples,
    801                     ensure_min_features=ensure_min_features,
--> 802                     estimator=estimator)
    803     if multi_output:
    804         y = check_array(y, accept_sparse='csr', force_all_finite=True,

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    639         if not allow_nd and array.ndim >= 3:
    640             raise ValueError("Found array with dim %d. %s expected <= 2."
--> 641                              % (array.ndim, estimator_name))
    642 
    643         if force_all_finite:

ValueError: Found array with dim 3. Estimator expected <= 2.
```

The model has been successfully fit to our training data. Let's check its performance on test set.

In [ ]:
```
# viewing params of random forest
rf.get_params()
```

In [ ]:
```
# making predictions on test set
start = time.time()

y_pred = rf.predict(X_test)

end = time.time()
print('Time Taken: %.3f seconds' % (end-start))

y_pred[:10]
```

In [ ]:
```
# check accuracy
print('Accuracy on train data: %.4f' % rf.score(X_train, y_train))
print('Accuracy on test data %.4f' % rf.score(X_test, y_test))
```

2. Support Vector Machine (SVM)

In this section, we'll fit SVM on our training dataset and check its performance on test data.

In [ ]:
```
# instantiate ^ fit SVM to train data
svc = SVC()
svc.fit(X_train, y_train)
```

Model has been successfully fit to train data

In [ ]:
```
# check parameters of SVM
svc.get_params()
```

In [ ]:
```
# predict on test set
y_pred_svc = svc.predict(X_test)
y_pred_svc[:5]
```

In [ ]:
```
# check accuracy of SVM
print('Acc train %.4f' % svc.score(X_train, y_train))
print('Acc test %.4f' % svc.score(X_test, y_test))
```

As given, we are getting **92%** accuracy on training & **90%** accuracy on test dataset using **Support Vector Machines (SVM)**.

Prediction Results

In this section, we'll visualize the results obtained by **SVM** & **Random Forest**. The results are:

- Random Forest got an accuracy of **99%** on training data and **97%** on test data
- SVM got an accuracy of **92%** on train data and **90%** on test data
- We'll also plot confusion matrices of respective classifiers

In [ ]:
```
acc_rf = rf.score(X_test, y_test)
acc_svm = rf.score(X_test, y_test)
classifiers = ['Random Forest', 'SVM']
plt.title('Accuracy of SVM & Random Forest on Test Data')
plt.ylabel('Accuracy')
plt.bar(classifiers, [acc_rf, acc_svm])
```

In [ ]:
```
# print classification report of SVM anf RF
print('Classification Report of Random Forest:')
print('')
print(classification_report(y_test, y_pred))
```

In [ ]:
```
# print classification report of SVM anf RF
print('Classification Report of SVM:')
print('')
print(classification_report(y_test, y_pred_svc))
```

Confusion Matrices

Below are the confusion matrices obtained as a result of classification from Random Forest & SVM:

Confuion Matrix for Random Forest

In [ ]:
```
# confusion matrix for Random Forest
plot_confusion_matrix(rf, X_test, y_test)
```

Confuion Matrix for SVM

In [ ]:
```
# confusion matrix for Random Forest
plot_confusion_matrix(svc, X_test, y_test)
```