In [1]:

```
# Some neccessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

In [2]:

```
Bank_note = pd.read_csv('BankNote_Authentication.csv')
Bank_note.head()
```

Out[2]:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [3]:

```
Bank_note.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   variance  1372 non-null   float64
 1   skewness  1372 non-null   float64
 2   curtosis  1372 non-null   float64
 3   entropy   1372 non-null   float64
 4   class     1372 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
```

In [4]:

```
Bank_note.describe()
```

Out[4]:

	variance	skewness	curtosis	entropy	class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

In [5]:

```
# Check for missing value
Bank_note.isnull().sum()/len(Bank_note)*100
```

Out[5]:

```
variance    0.0
skewness    0.0
curtosis    0.0
entropy     0.0
class       0.0
dtype: float64
```

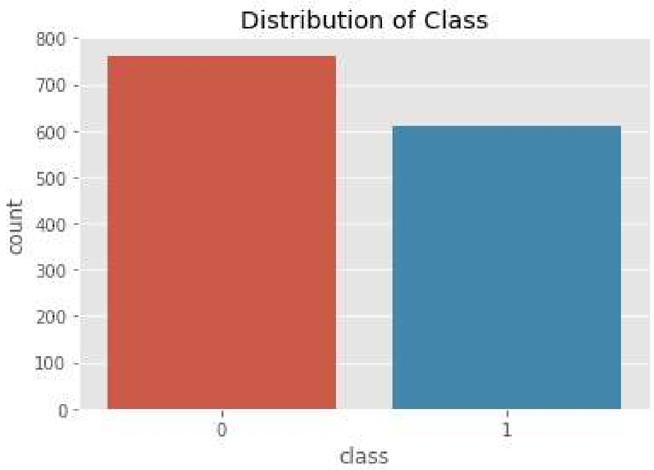
Vizualization of the data

In [6]:

```
# Target Variable
plt.style.use('ggplot')
sns.countplot(data = Bank_note , x = 'class')
plt.title('Distribution of Class')
```

Out[6]:

```
Text(0.5, 1.0, 'Distribution of Class')
```



Data preprocessing Steps

In [7]:

```
# Split data into target & predictors
X = Bank_note.iloc[: , :4]
y = Bank_note.iloc[: , -1]
```

In [8]:

```
# Standard Scaling ,important for convergence of the neural network
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [9]:

```
# Split the dataset into training and testing
X_train, X_test, Y_train, Y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=4)

n_features = X.shape[1]
n_classes = 2
```

In [10]:

```
# Convert the target labels to one-hot encoded format
Y_train = tf.keras.utils.to_categorical(Y_train, num_classes=2)
```

In [11]:

```
Y_test = tf.keras.utils.to_categorical(Y_test, num_classes=2)
```

In [12]:

```
# neural network architecture for adaptive activation function (AF) selection

initializer0 = keras.initializers.RandomUniform(minval = -0.003, maxval =0.0)
initializer1 = keras.initializers.RandomUniform(minval = -0.003, maxval =0.0)

class Adaact(keras.layers.Layer):
    def __init__(self):
        super(Adaact, self).__init__()
        self.k0 = self.add_weight(name='k0', shape = (), initializer=initializer0, trainable=True)
        self.k1 = self.add_weight(name='k1', shape = (), initializer=initializer1, trainable=True)

    def call(self, inputs):
        return self.k0 + tf.multiply(inputs, self.k1)
```

In [13]:

```
np.random.seed(1)
```

In [14]:

```
# Build model with fully connected layers with dropout regulation
model = Sequential()
model.add(layers.Dense(25, input_dim=n_features))
act = Adaact()
model.add(act)
model.add(layers.Dropout(0.1))
model.add(layers.Dense(n_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 25)                125

 adaact (Adaact)             (None, 25)                2

 dropout (Dropout)           (None, 25)                0

 dense_1 (Dense)             (None, 2)                 52

=================================================================
Total params: 179
Trainable params: 179
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
import datetime
batch_size = 5
epochs = 100

model.compile(loss="binary_crossentropy", optimizer = keras.optimizers.Adam(learning_rate = 0.001), metrics

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs, validation_split=0.2, verbose=1
```

```
Epoch 1/100
176/176 [==============================] - 4s 12ms/step - loss: 0.6262 - accuracy: 0.7856
- val_loss: 0.4996 - val_accuracy: 0.8364
Epoch 2/100
176/176 [==============================] - 1s 6ms/step - loss: 0.3448 - accuracy: 0.8962 -
val_loss: 0.2483 - val_accuracy: 0.9182
Epoch 3/100
176/176 [==============================] - 1s 6ms/step - loss: 0.1812 - accuracy: 0.9373 -
val_loss: 0.1305 - val_accuracy: 0.9636
Epoch 4/100
176/176 [==============================] - 0s 3ms/step - loss: 0.1061 - accuracy: 0.9647 -
val_loss: 0.0791 - val_accuracy: 0.9818
Epoch 5/100
176/176 [==============================] - 0s 2ms/step - loss: 0.0728 - accuracy: 0.9749 -
val_loss: 0.0572 - val_accuracy: 0.9818
Epoch 6/100
176/176 [==============================] - 0s 2ms/step - loss: 0.0569 - accuracy: 0.9772 -
val_loss: 0.0472 - val_accuracy: 0.9818
Epoch 7/100
```

In [16]:

```
# parameter updates during training
parameter_updates = history.history['loss']
print(parameter_updates)
```

```
[0.6261903047561646, 0.3448285758495331, 0.18124817311763763, 0.1061161682009697, 0.072807341
81404114, 0.05689842626452446, 0.05023106187582016, 0.042717136442661285, 0.044297214597463б
1, 0.039925675839185715, 0.04003143683075905, 0.036097317934036255, 0.03450433164834976, 0.03
456991910934448, 0.03755985200405121, 0.034785423427820206, 0.03494922816753387 5, 0.032089304
17895317, 0.03273605927824974, 0.029516370967030525, 0.02748020738363266, 0.0291885342448949
8, 0.029104694724082947, 0.03373458981513977, 0.030214454978704453, 0.029009433463215828, 0.0
33283770084381104, 0.026095576584339142, 0.02900168113410473, 0.026819398626685143, 0.0266080
92710375786, 0.03330490365624428, 0.027282165363430977, 0.029476206749677658, 0.0273567810654
6402, 0.026554759591817856, 0.027292057871818542, 0.02948649227619171, 0.02622881717979908,
0.026660239323973656, 0.021387604996562004, 0.024269217625260353, 0.025518273934721947, 0.024
028358981013298, 0.030812887474894524, 0.023990895599126816, 0.024592721834778786, 0.02480297
7219223976, 0.024202303960919 38, 0.029533570632338524, 0.028178056702017784, 0.02755907550454
1397, 0.030502453446388245, 0.02718379534780979, 0.025319790467619896, 0.028918184340000153,
0.026618460193276405, 0.02867400273680687, 0.02715141884982586, 0.028261803090572357, 0.02470
07068246603, 0.027037978172302246, 0.03074176423251629, 0.023135755211114883, 0.026379359886 0
50224, 0.026155896484851837, 0.02387145720422268, 0.026583200320601463, 0.026884529739618 3,
0.023311931639909744, 0.027949374169111252, 0.026531649753451347, 0.025653796270489693, 0.026
84391662478447, 0.021825456991791725, 0.026076989248394966, 0.025577634572982788, 0.025362234
562635422, 0.02691815234720707, 0.023286515846848488, 0.026399897411465645, 0.028523474931716
92, 0.025843506678938866, 0.024095602333545685, 0.025104766711592674, 0.025554519146680832,
0.0256079062819480 9, 0.024610135704278946, 0.026773866266012192, 0.025588402524590492, 0.0247
04288691282272, 0.022267434746026993, 0.024958722293376923, 0.024913746863603592, 0.026283372
193574905, 0.026179606094956398, 0.023963002488017082, 0.025048557668924 33, 0.026499593630433
083, 0.025324976071715355]
```

In [17]:

```
# final parameter values at the end of training
final_parameter_values = model.get_weights()
print(final_parameter_values)
```

```
[array([[-0.14139467, -0.6394828 ,  0.31086674,  0.16637956, -0.50747466,
        -0.19739258, -0.49401265, -0.7229469 , -0.5440975 ,  0.22787444,
         0.78157616, -0.6254811 ,  0.7499037 ,  0.21823817, -1.0567837 ,
        -0.546263  , -0.18043919,  0.9456196 ,  0.85682625, -0.8117007 ,
         0.6038053 ,  0.5656989 ,  0.45649683,  0.729022  , -0.4619017 ],
       [-0.333675  , -0.34122488,  0.5024505 ,  0.8188201 , -0.57741845,
        -0.8782533 , -0.53477436, -0.95673245, -0.7562165 ,  1.0780644 ,
         0.6432481 , -0.6840831 ,  0.37103435, -0.13566802, -0.45237586,
        -0.7888445 , -0.3138447 ,  0.32529494,  0.5159587 , -0.38933602,
         0.9703745 ,  0.5053425 ,  0.5583918 ,  0.32627723, -0.7993878 ],
       [-0.24711555, -0.50590044,  0.17560421,  0.65469784, -0.38280398,
        -0.39394075, -0.64142466, -0.8874544 , -0.52671075,  0.83008873,
         0.76970416, -0.26751631,  0.21184896, -0.01683507, -0.71619326,
        -0.64384085, -0.48783186,  0.6587785 ,  0.69404566, -0.70196104,
         0.8092501 ,  0.49358338,  0.48421577,  0.12108159, -0.5474645 ],
       [ 0.09426061,  0.06463203,  0.3085331 ,  0.03685943,  0.19213645,
        -0.25716838, -0.2769125 , -0.1500468 , -0.23843782, -0.19559354,
        -0.23130761,  0.0374947 , -0.27489498, -0.00360605,  0.03679829,
        -0.02110022, -0.21735343, -0.09453958, -0.14330882, -0.03632021,
        -0.04136774,  0.2546833 ,  0.10636898, -0.13246648, -0.30004793]],
      dtype=float32), array([-0.22702327, -0.22792327,  0.30351952,  0.29987508, -0.16188428,
       -0.31755787, -0.21196893, -0.33271062, -0.28991583,  0.3091811 ,
        0.26126334, -0.2328347 ,  0.25098428, -0.01719618, -0.25270256,
       -0.24555854, -0.265023  ,  0.19237454,  0.22555193, -0.15294755,
        0.351322  ,  0.2654592 ,  0.3152896 ,  0.23208576, -0.30924782],
      dtype=float32), 0.04507808, -1.3688513, array([[ 0.4159409 , -0.37696254],
       [ 0.98860645, -1.2390804 ],
       [-0.13326763,  0.536805  ],
       [-1.1862712 ,  0.85979825],
       [ 0.48511687, -0.71396714],
       [ 0.5659488 , -0.33973733],
       [ 0.42215988, -0.77675647],
       [ 1.7946576 , -1.6395364 ],
       [ 0.9057746 , -1.1399345 ],
       [-0.6994393 ,  0.9232474 ],
       [-1.0663705 ,  0.9482057 ],
       [ 0.6386752 , -0.18999629],
       [-0.18520314,  0.68033075],
       [ 0.10959896,  0.10321756],
       [ 0.91730034, -0.74829155],
       [ 1.4218578 , -1.6239287 ],
       [ 0.31383672, -0.28191727],
       [-1.0704436 ,  1.1054313 ],
       [-1.4360706 ,  0.9999816 ],
       [ 0.7548888 , -0.8558407 ],
       [-1.3041137 ,  1.6202729 ],
       [-0.8379797 ,  0.5620404 ],
       [-1.8263292 ,  1.346389  ],
       [-0.45884988,  0.25136098],
       [ 0.76089567, -0.8224173 ]], dtype=float32), array([ 0.14973214, -0.1726101 ], dtype=f
loat32)]
```

In [18]:

```
total_parameters = model.count_params()
print(total_parameters)
```

```
179
```

In [19]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.016286909580230713
Test accuracy: 0.996363639831543
```

In [20]:

```
# Optimal Value of k0,k1 & k2
print("AF coefficients (weights) {}".format(act.get_weights()))
```

```
AF coefficients (weights) [0.04507808, -1.3688513]
```

In [21]:

```
from sklearn.metrics import f1_score
# training and test loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [22]:

```
print(train_loss[-1])
```

```
0.025324976071715355
```

In [23]:

```
# training and testing accuracy
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
```

In [24]:

```
print(train_accuracy[-1])
```

```
0.9874572157859802
```

In [25]:

```
# predictions on the test set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
```

```
9/9 [==============================] - 0s 2ms/step
```

In [26]:

```
#  F1-Score
f1 = f1_score(np.argmax(Y_test, axis=1), y_pred, average='weighted')
```

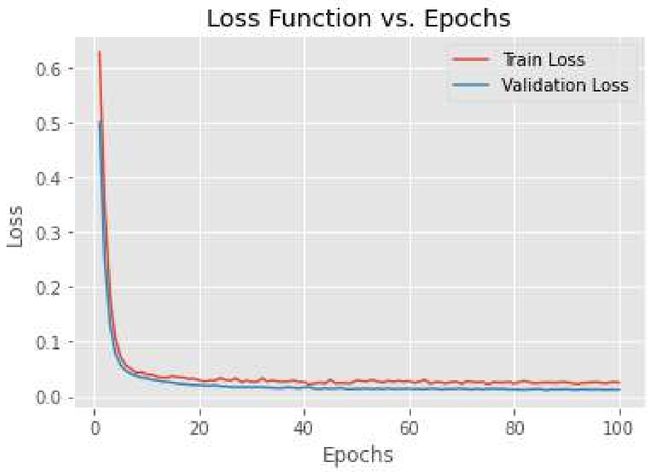
In [27]:

```
# Print test loss, test accuracy, and F1-Score
score = model.evaluate(X_test, Y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
print("F1-Score:", f1)
```

```
Test loss: 0.016286909580230713
Test accuracy: 0.996363639831543
F1-Score: 0.9963656013017715
```

In [28]:

```
# Plot of loss function vs. epochs
plt.plot(range(1, epochs + 1), train_loss, label='Train Loss')
plt.plot(range(1, epochs + 1), val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Function vs. Epochs')
plt.legend()
plt.show()
```



In [ ]:

```

```