In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
```

```
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy versi
on >=1.17.3 and <1.25.0 is required for this version of SciPy (detected version
1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Load Dataset

In [2]:
```
df = pd.read_csv('Breastcancer.csv')
```

In [3]:
```
# dropping unncessary columns
df.drop("Unnamed: 32",axis=1,inplace=True)
```

EDA

In [4]:
```
df.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_r
0	842302	M	17.99	10.38	122.80	1001.0	0.1
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1

5 rows × 32 columns

In [5]:
```
#checking missing values
df.isnull().sum()
```

Out[5]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

In [6]:
```
df.shape
```

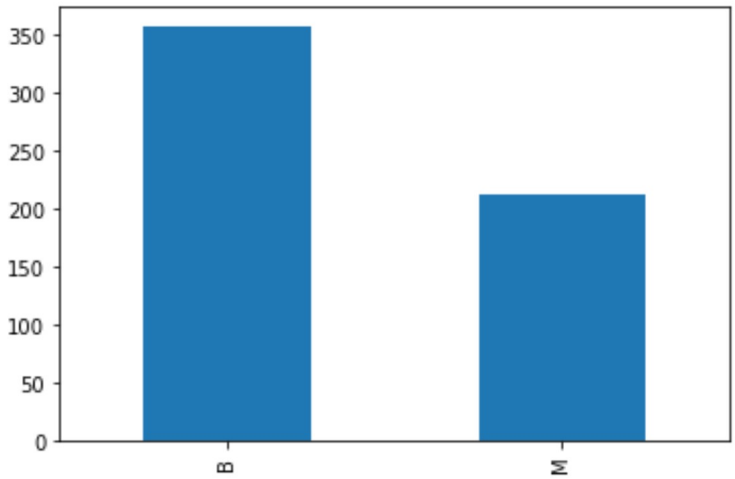
Out[6]:
```
(569, 32)
```

In [7]:
```
df.dtypes
```

Out[7]:
```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

In [8]:
```
df.diagnosis.value_counts().plot.bar()
```

Out[8]:
```
<AxesSubplot:>
```



Split the dataset

In [58]:
```
## Data Preprocessing
X=df.iloc[:,2:].values
y=df.iloc[:,1].values
```

In [60]:
```
from sklearn.preprocessing import LabelEncoder,StandardScaler
labelencode = LabelEncoder()
y=labelencode.fit_transform(y)
```

In [11]:
```
#train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=12
```

In [12]:
```
#applying standard scaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [13]:
```
X_train.shape, X_test.shape
```

Out[13]:
```
((398, 30), (171, 30))
```

Model 1

In [14]:
```
# importing tensorflow and Keras
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
from keras import backend
```

```
2024-05-14 14:30:12.606340: I tensorflow/core/util/port.cc:113] oneDNN custom op
erations are on. You may see slightly different numerical results due to floatin
g-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-05-14 14:30:12.609713: I external/local_tsl/tsl/cuda/cudart_stub.cc:31] Cou
ld not find cuda drivers on your machine, GPU will not be used.
2024-05-14 14:30:12.641047: E external/local_xla/xla/stream_executor/cuda/cuda_d
nn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for
plugin cuDNN when one has already been registered
2024-05-14 14:30:12.641090: E external/local_xla/xla/stream_executor/cuda/cuda_f
ft.cc:607] Unable to register cuFFT factory: Attempting to register factory for
plugin cuFFT when one has already been registered
2024-05-14 14:30:12.642691: E external/local_xla/xla/stream_executor/cuda/cuda_b
las.cc:1515] Unable to register cuBLAS factory: Attempting to register factory f
or plugin cuBLAS when one has already been registered
2024-05-14 14:30:12.649494: I external/local_tsl/tsl/cuda/cudart_stub.cc:31] Cou
ld not find cuda drivers on your machine, GPU will not be used.
2024-05-14 14:30:12.650657: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in perform
ance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI AVX512_BF16 FMA,
in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-05-14 14:30:13.245058: W tensorflow/compiler/tf2tensorrt/utils/py_utils.c
c:38] TF-TRT Warning: Could not find TensorRT
```

In [62]:
```
# setting up the layers of Neural Network

model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(30,)),
                          keras.layers.Dense(1, activation='relu'),
                          keras.layers.Dense(2, activation='sigmoid')
])
```

In [63]:
```
# compiling the Neural Network

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [64]:
```
print(model.summary())
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 30)                0

 dense_39 (Dense)            (None, 1)                 31

 dense_40 (Dense)            (None, 2)                 4

=================================================================
Total params: 35 (140.00 Byte)
Trainable params: 35 (140.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [65]:
```
#!pip3 install keras-visualizer

from keras_visualizer import visualizer
visualizer(model, file_format='png', view=True, settings=None)
```

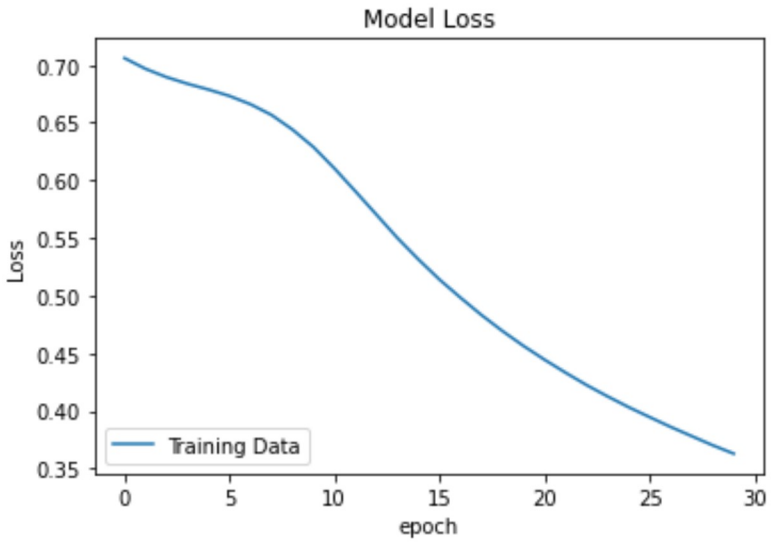
In [54]:
```
# training the Neural Network

history = model.fit(X_train, y_train, validation_split=0.1, epochs=30)
```

```
Epoch 1/30
12/12 [==============================] - 0s 9ms/step - loss: 0.7060 - accuracy:
0.6425 - val_loss: 0.6980 - val_accuracy: 0.6000
Epoch 2/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6967 - accuracy:
0.6425 - val_loss: 0.6924 - val_accuracy: 0.6000
Epoch 3/30
12/12 [==============================] - 0s 2ms/step - loss: 0.6895 - accuracy:
0.6425 - val_loss: 0.6882 - val_accuracy: 0.6000
Epoch 4/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6838 - accuracy:
0.6425 - val_loss: 0.6847 - val_accuracy: 0.6000
Epoch 5/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6787 - accuracy:
0.6453 - val_loss: 0.6813 - val_accuracy: 0.6000
Epoch 6/30
12/12 [==============================] - 0s 2ms/step - loss: 0.6732 - accuracy:
0.6620 - val_loss: 0.6770 - val_accuracy: 0.6500
Epoch 7/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6660 - accuracy:
0.7179 - val_loss: 0.6709 - val_accuracy: 0.6500
Epoch 8/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6567 - accuracy:
0.7654 - val_loss: 0.6623 - val_accuracy: 0.7000
Epoch 9/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6440 - accuracy:
0.8045 - val_loss: 0.6507 - val_accuracy: 0.7750
Epoch 10/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6287 - accuracy:
0.8575 - val_loss: 0.6361 - val_accuracy: 0.7750
Epoch 11/30
12/12 [==============================] - 0s 3ms/step - loss: 0.6102 - accuracy:
0.8827 - val_loss: 0.6194 - val_accuracy: 0.8250
Epoch 12/30
12/12 [==============================] - 0s 3ms/step - loss: 0.5903 - accuracy:
0.8966 - val_loss: 0.6019 - val_accuracy: 0.8000
Epoch 13/30
12/12 [==============================] - 0s 3ms/step - loss: 0.5701 - accuracy:
0.9022 - val_loss: 0.5838 - val_accuracy: 0.8500
Epoch 14/30
12/12 [==============================] - 0s 3ms/step - loss: 0.5498 - accuracy:
0.9078 - val_loss: 0.5663 - val_accuracy: 0.8500
Epoch 15/30
12/12 [==============================] - 0s 3ms/step - loss: 0.5313 - accuracy:
0.9162 - val_loss: 0.5502 - val_accuracy: 0.8500
Epoch 16/30
12/12 [==============================] - 0s 3ms/step - loss: 0.5138 - accuracy:
0.9134 - val_loss: 0.5350 - val_accuracy: 0.9000
Epoch 17/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4981 - accuracy:
0.9246 - val_loss: 0.5212 - val_accuracy: 0.9000
Epoch 18/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4831 - accuracy:
0.9274 - val_loss: 0.5078 - val_accuracy: 0.9000
Epoch 19/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4692 - accuracy:
0.9302 - val_loss: 0.4954 - val_accuracy: 0.9000
Epoch 20/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4563 - accuracy:
0.9302 - val_loss: 0.4846 - val_accuracy: 0.9000
Epoch 21/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4444 - accuracy:
0.9302 - val_loss: 0.4739 - val_accuracy: 0.9000
```

```
Epoch 22/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4331 - accuracy:
0.9330 - val_loss: 0.4635 - val_accuracy: 0.8750
Epoch 23/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4224 - accuracy:
0.9330 - val_loss: 0.4538 - val_accuracy: 0.8750
Epoch 24/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4125 - accuracy:
0.9385 - val_loss: 0.4453 - val_accuracy: 0.8750
Epoch 25/30
12/12 [==============================] - 0s 3ms/step - loss: 0.4032 - accuracy:
0.9385 - val_loss: 0.4379 - val_accuracy: 0.8750
Epoch 26/30
12/12 [==============================] - 0s 3ms/step - loss: 0.3946 - accuracy:
0.9385 - val_loss: 0.4307 - val_accuracy: 0.8750
Epoch 27/30
12/12 [==============================] - 0s 3ms/step - loss: 0.3861 - accuracy:
0.9413 - val_loss: 0.4239 - val_accuracy: 0.8750
Epoch 28/30
12/12 [==============================] - 0s 3ms/step - loss: 0.3781 - accuracy:
0.9441 - val_loss: 0.4173 - val_accuracy: 0.8750
Epoch 29/30
12/12 [==============================] - 0s 3ms/step - loss: 0.3702 - accuracy:
0.9497 - val_loss: 0.4112 - val_accuracy: 0.8750
Epoch 30/30
12/12 [==============================] - 0s 3ms/step - loss: 0.3629 - accuracy:
0.9497 - val_loss: 0.4054 - val_accuracy: 0.9000
```

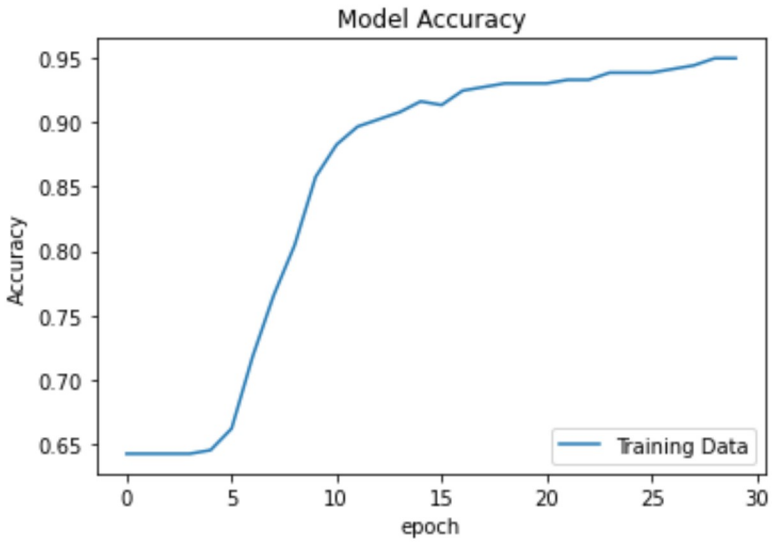
In [55]:
```
plt.plot(history.history["loss"])

plt.title("Model Loss")
plt.ylabel("Loss")
plt.xlabel("epoch")

plt.legend(["Training Data" ,"Validation Data"], loc="lower left")
```

Out[55]:
```
<matplotlib.legend.Legend at 0x7a0f1b499510>
```



In [56]:
```
plt.plot(history.history["accuracy"])

plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("epoch")

plt.legend(["Training Data" ,"Validation Data"], loc="lower right")
```

Out[56]: <matplotlib.legend.Legend at 0x7a0f6030c400>



Model 2

In [22]:
```
# Creating the model
model2 = tf.keras.Sequential([
            tf.keras.layers.Flatten(input_shape=(30,)),
            tf.keras.layers.Dense(units=3, activation = "relu"),
            tf.keras.layers.Dense(units=2, activation = "sigmoid")
])
# compiling the Neural Network
model2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
history2= model2.fit(X_train, y_train, epochs=30)
```

```
Epoch 1/30
13/13 [==============================] - 0s 773us/step - loss: 0.6388 - accuracy
: 0.6106
Epoch 2/30
13/13 [==============================] - 0s 623us/step - loss: 0.5888 - accuracy
: 0.6935
Epoch 3/30
13/13 [==============================] - 0s 594us/step - loss: 0.5541 - accuracy
: 0.7613
Epoch 4/30
13/13 [==============================] - 0s 624us/step - loss: 0.5284 - accuracy
: 0.8015
Epoch 5/30
13/13 [==============================] - 0s 597us/step - loss: 0.5083 - accuracy
: 0.8492
Epoch 6/30
13/13 [==============================] - 0s 612us/step - loss: 0.4917 - accuracy
: 0.8794
Epoch 7/30
13/13 [==============================] - 0s 577us/step - loss: 0.4780 - accuracy
: 0.8995
Epoch 8/30
13/13 [==============================] - 0s 635us/step - loss: 0.4664 - accuracy
: 0.9146
Epoch 9/30
13/13 [==============================] - 0s 603us/step - loss: 0.4559 - accuracy
: 0.9196
Epoch 10/30
13/13 [==============================] - 0s 575us/step - loss: 0.4461 - accuracy
: 0.9296
Epoch 11/30
13/13 [==============================] - 0s 629us/step - loss: 0.4369 - accuracy
: 0.9422
Epoch 12/30
13/13 [==============================] - 0s 656us/step - loss: 0.4281 - accuracy
: 0.9472
Epoch 13/30
13/13 [==============================] - 0s 703us/step - loss: 0.4196 - accuracy
: 0.9523
Epoch 14/30
13/13 [==============================] - 0s 696us/step - loss: 0.4115 - accuracy
: 0.9523
Epoch 15/30
13/13 [==============================] - 0s 694us/step - loss: 0.4034 - accuracy
: 0.9548
Epoch 16/30
13/13 [==============================] - 0s 665us/step - loss: 0.3956 - accuracy
: 0.9548
Epoch 17/30
13/13 [==============================] - 0s 601us/step - loss: 0.3882 - accuracy
: 0.9598
Epoch 18/30
13/13 [==============================] - 0s 737us/step - loss: 0.3811 - accuracy
: 0.9573
Epoch 19/30
13/13 [==============================] - 0s 755us/step - loss: 0.3744 - accuracy
: 0.9598
Epoch 20/30
13/13 [==============================] - 0s 720us/step - loss: 0.3678 - accuracy
: 0.9598
Epoch 21/30
13/13 [==============================] - 0s 675us/step - loss: 0.3614 - accuracy
: 0.9598
```

```
Epoch 22/30
13/13 [==============================] - 0s 1ms/step - loss: 0.3553 - accuracy:
0.9623
Epoch 23/30
13/13 [==============================] - 0s 752us/step - loss: 0.3494 - accuracy
: 0.9623
Epoch 24/30
13/13 [==============================] - 0s 716us/step - loss: 0.3436 - accuracy
: 0.9648
Epoch 25/30
13/13 [==============================] - 0s 914us/step - loss: 0.3380 - accuracy
: 0.9648
Epoch 26/30
13/13 [==============================] - 0s 719us/step - loss: 0.3325 - accuracy
: 0.9673
Epoch 27/30
13/13 [==============================] - 0s 641us/step - loss: 0.3271 - accuracy
: 0.9673
Epoch 28/30
13/13 [==============================] - 0s 745us/step - loss: 0.3218 - accuracy
: 0.9673
Epoch 29/30
13/13 [==============================] - 0s 696us/step - loss: 0.3167 - accuracy
: 0.9673
Epoch 30/30
13/13 [==============================] - 0s 626us/step - loss: 0.3117 - accuracy
: 0.9673
```

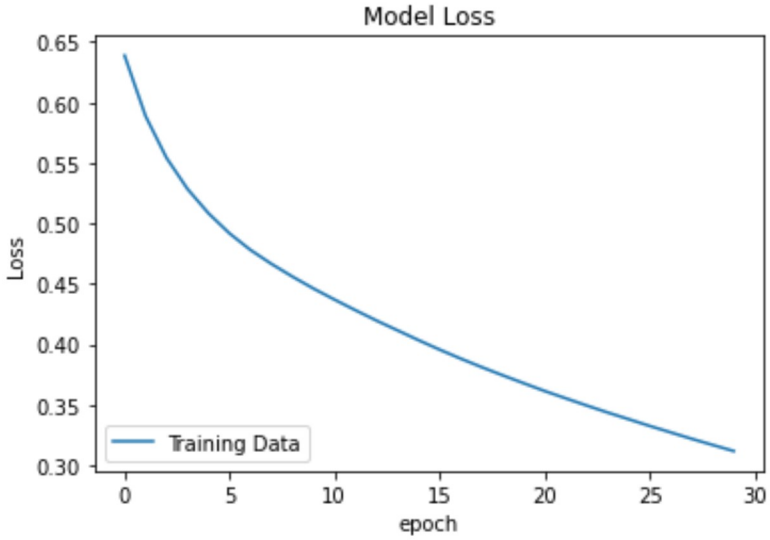
In [23]:
```
visualizer(model2, file_format='png', view=True, settings=None)
```

In [25]:
```
print(model2.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 30)                0

 dense_2 (Dense)             (None, 3)                 93

 dense_3 (Dense)             (None, 2)                 8

=================================================================
Total params: 101 (404.00 Byte)
Trainable params: 101 (404.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

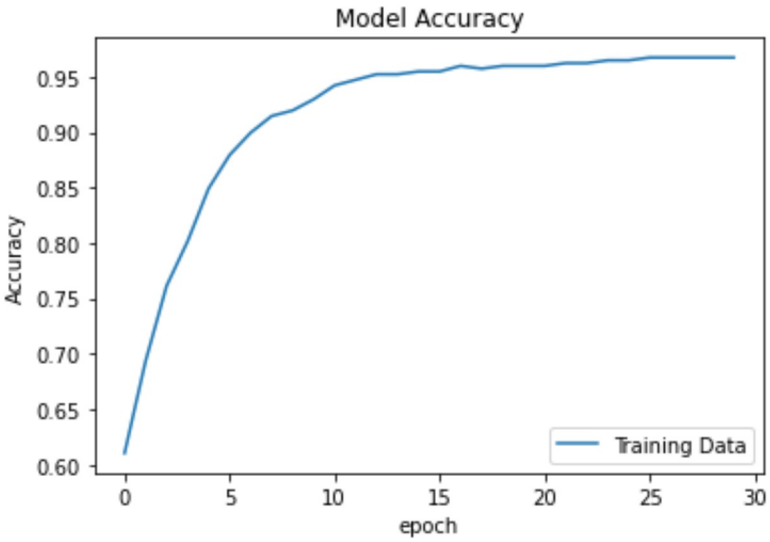
In [24]:
```
plt.plot(history2.history["loss"])

plt.title("Model Loss")
plt.ylabel("Loss")
plt.xlabel("epoch")

plt.legend(["Training Data" ,"Validation Data"], loc="lower left")
```

Out[24]:
```
<matplotlib.legend.Legend at 0x7a0f6031eef0>
```



In [26]:
```
plt.plot(history2.history["accuracy"])

plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("epoch")

plt.legend(["Training Data" ,"Validation Data"], loc="lower right")
```

Out[26]:
```
<matplotlib.legend.Legend at 0x7a0f60083190>
```



Model 3

In [27]:

```
# Creating the model
model3 = tf.keras.Sequential([
            tf.keras.layers.Flatten(input_shape=(30,)),
            tf.keras.layers.Dense(units=6, activation = "relu"),
            tf.keras.layers.Dense(units=2, activation = "relu"),
            tf.keras.layers.Dense(units=2, activation = "sigmoid")
])
# compiling the Neural Network
model3.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
history3= model3.fit(X_train, y_train, epochs=30)
```

```
Epoch 1/30
13/13 [==============================] - 0s 935us/step - loss: 1.4723 - accuracy
: 0.3568
Epoch 2/30
13/13 [==============================] - 0s 695us/step - loss: 1.2197 - accuracy
: 0.3568
Epoch 3/30
13/13 [==============================] - 0s 739us/step - loss: 1.0189 - accuracy
: 0.3668
Epoch 4/30
13/13 [==============================] - 0s 684us/step - loss: 0.8718 - accuracy
: 0.3819
Epoch 5/30
13/13 [==============================] - 0s 685us/step - loss: 0.7683 - accuracy
: 0.4171
Epoch 6/30
13/13 [==============================] - 0s 728us/step - loss: 0.6901 - accuracy
: 0.4799
Epoch 7/30
13/13 [==============================] - 0s 760us/step - loss: 0.6369 - accuracy
: 0.5302
Epoch 8/30
13/13 [==============================] - 0s 785us/step - loss: 0.5967 - accuracy
: 0.6080
Epoch 9/30
13/13 [==============================] - 0s 617us/step - loss: 0.5651 - accuracy
: 0.6633
Epoch 10/30
13/13 [==============================] - 0s 682us/step - loss: 0.5400 - accuracy
: 0.7211
Epoch 11/30
13/13 [==============================] - 0s 631us/step - loss: 0.5185 - accuracy
: 0.7688
Epoch 12/30
13/13 [==============================] - 0s 725us/step - loss: 0.5002 - accuracy
: 0.8191
Epoch 13/30
13/13 [==============================] - 0s 744us/step - loss: 0.4841 - accuracy
: 0.8568
Epoch 14/30
13/13 [==============================] - 0s 637us/step - loss: 0.4699 - accuracy
: 0.8794
Epoch 15/30
13/13 [==============================] - 0s 651us/step - loss: 0.4566 - accuracy
: 0.8945
Epoch 16/30
13/13 [==============================] - 0s 627us/step - loss: 0.4447 - accuracy
: 0.9045
Epoch 17/30
13/13 [==============================] - 0s 780us/step - loss: 0.4334 - accuracy
: 0.9045
Epoch 18/30
13/13 [==============================] - 0s 765us/step - loss: 0.4233 - accuracy
: 0.9196
Epoch 19/30
13/13 [==============================] - 0s 651us/step - loss: 0.4137 - accuracy
: 0.9322
Epoch 20/30
13/13 [==============================] - 0s 779us/step - loss: 0.4049 - accuracy
: 0.9372
Epoch 21/30
13/13 [==============================] - 0s 789us/step - loss: 0.3965 - accuracy
: 0.9447
```

```
Epoch 22/30
13/13 [==============================] - 0s 695us/step - loss: 0.3887 - accuracy
: 0.9472
Epoch 23/30
13/13 [==============================] - 0s 662us/step - loss: 0.3813 - accuracy
: 0.9523
Epoch 24/30
13/13 [==============================] - 0s 682us/step - loss: 0.3741 - accuracy
: 0.9523
Epoch 25/30
13/13 [==============================] - 0s 651us/step - loss: 0.3673 - accuracy
: 0.9573
Epoch 26/30
13/13 [==============================] - 0s 732us/step - loss: 0.3607 - accuracy
: 0.9598
Epoch 27/30
13/13 [==============================] - 0s 641us/step - loss: 0.3543 - accuracy
: 0.9573
Epoch 28/30
13/13 [==============================] - 0s 764us/step - loss: 0.3480 - accuracy
: 0.9673
Epoch 29/30
13/13 [==============================] - 0s 650us/step - loss: 0.3419 - accuracy
: 0.9698
Epoch 30/30
13/13 [==============================] - 0s 798us/step - loss: 0.3359 - accuracy
: 0.9698
```

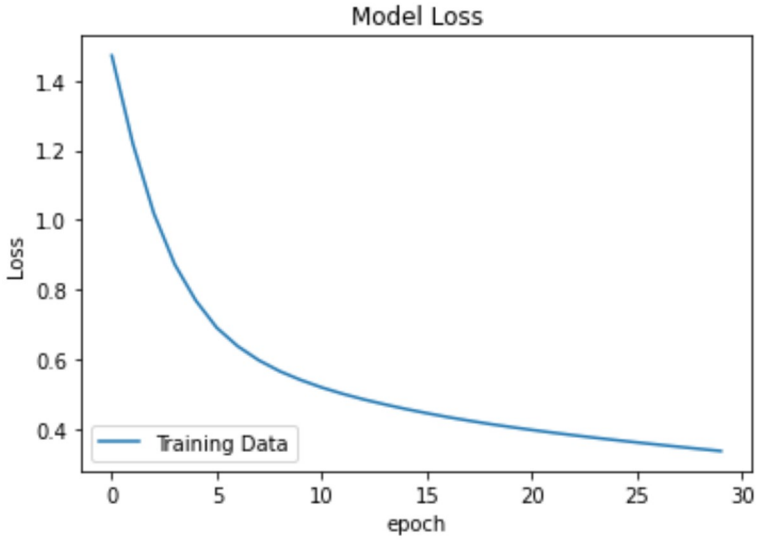
In [28]:
```
print(model3.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 30)                0

 dense_4 (Dense)             (None, 6)                 186

 dense_5 (Dense)             (None, 2)                 14

 dense_6 (Dense)             (None, 2)                 6

=================================================================
Total params: 206 (824.00 Byte)
Trainable params: 206 (824.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [29]:
```
visualizer(model3, file_format='png', view=True, settings=None)
```

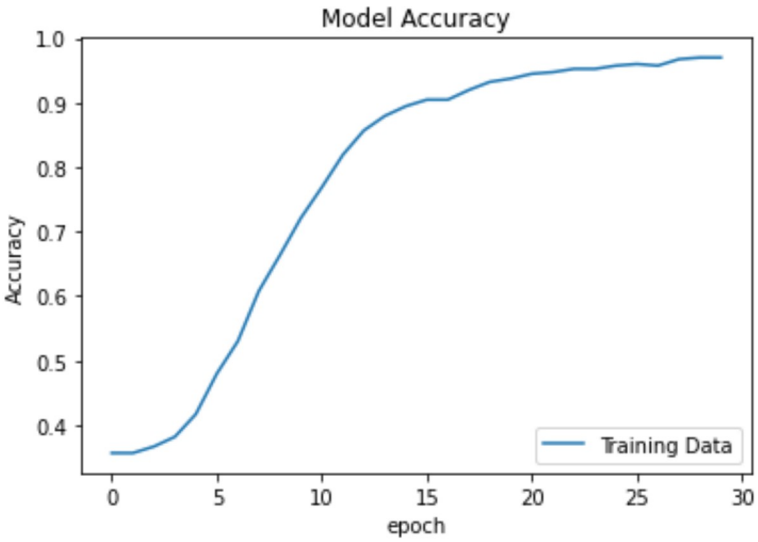
In [30]:
```
plt.plot(history3.history["loss"])

plt.title("Model Loss")
plt.ylabel("Loss")
plt.xlabel("epoch")

plt.legend(["Training Data" ,"Validation Data"], loc="lower left")
```

Out[30]:
```
<matplotlib.legend.Legend at 0x7a0f487e3400>
```



In [31]:
```
plt.plot(history3.history["accuracy"])

plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("epoch")

plt.legend(["Training Data" ,"Validation Data"], loc="lower right")
```

Out[31]:
```
<matplotlib.legend.Legend at 0x7a0f48f1f160>
```



Result comparison

In [32]:
```
print("Comparação da loss dos 3 modelos:")
print("Loss modelo 1: ", np.min(history.history["loss"]))
print("Loss modelo 2: ", np.min(history2.history["loss"]))
print("Loss modelo 3: ", np.min(history3.history["loss"]))
```

```
Comparação da loss dos 3 modelos:
Loss modelo 1:  0.3350701630115509
Loss modelo 2:  0.31169626116752625
Loss modelo 3:  0.3359304964542389
```

In [33]:
```
print("Comparação da accuracy dos 3 modelos:")
print("Accuracy modelo 1: ", np.max(history.history["accuracy"]))
print("Accuracy modelo 2: ", np.max(history2.history["accuracy"]))
print("Accuracy modelo 3: ", np.max(history3.history["accuracy"]))
```

```
Comparação da accuracy dos 3 modelos:
Accuracy modelo 1:  0.9441340565681458
Accuracy modelo 2:  0.9673366546630859
Accuracy modelo 3:  0.9698492288589478
```

Evaluate the models using Test data

In [34]:
```
loss1, accuracy1 = model.evaluate(X_test, y_test)
loss2, accuracy2 = model2.evaluate(X_test, y_test)
loss3, accuracy3 = model3.evaluate(X_test, y_test)
```

```
6/6 [==============================] - 0s 984us/step - loss: 0.3445 - accuracy:
0.9825
6/6 [==============================] - 0s 899us/step - loss: 0.2887 - accuracy:
0.9708
6/6 [==============================] - 0s 1ms/step - loss: 0.3032 - accuracy:
0.9883
```

In [35]:
```
Y_pred1 = model.predict(X_test)
Y_pred2 = model2.predict(X_test)
Y_pred3 = model3.predict(X_test)
```

```
6/6 [==============================] - 0s 820us/step
6/6 [==============================] - 0s 836us/step
6/6 [==============================] - 0s 848us/step
```

In [36]:
```
print(Y_pred3)
```

```
[[5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [6.14918098e-02 6.77211881e-01]
 [5.77959359e-01 4.22040641e-01]
 [2.80171223e-02 7.48670340e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.34848750e-01 4.32027251e-01]
 [5.74407876e-01 4.45700824e-01]
 [5.77959359e-01 4.22040641e-01]
 [2.98937596e-02 7.52984345e-01]
 [3.61981578e-02 8.49992096e-01]
 [5.77959359e-01 4.22040641e-01]
 [4.45007533e-02 7.40994573e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [1.61778799e-03 9.03748393e-01]
 [4.22773987e-01 4.67648864e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [2.11636517e-02 7.80546665e-01]
 [1.44494653e-01 6.38172090e-01]
 [5.77959359e-01 4.22040641e-01]
 [6.13770895e-02 6.88211679e-01]
 [5.77959359e-01 4.22040641e-01]
 [2.95078665e-01 6.20548308e-01]
 [5.77212453e-01 4.26995009e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [1.79293558e-01 6.51434302e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.24836242e-01 4.28166181e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [2.85342224e-02 7.49335468e-01]
 [3.33004534e-01 5.02045870e-01]
 [5.77682018e-01 4.23879236e-01]
 [3.01444624e-02 7.46167779e-01]
 [5.77959359e-01 4.22040641e-01]
 [4.57636528e-02 7.46321976e-01]
 [5.70550919e-01 4.71615613e-01]
 [8.75516608e-03 8.27483296e-01]
 [1.10939480e-04 9.71675038e-01]
 [3.76847386e-01 4.94048476e-01]
 [1.16582751e-01 6.65566623e-01]
 [1.50415123e-01 6.15349591e-01]
 [6.50429283e-04 9.24112439e-01]
 [5.43741643e-01 4.31998640e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
 [8.40252638e-02 6.61175311e-01]
 [5.77959359e-01 4.22040641e-01]
 [1.49645458e-03 8.94005477e-01]
 [1.75740127e-03 9.04248357e-01]
 [5.76687932e-01 4.30481434e-01]
 [8.19209218e-03 8.55610907e-01]
 [5.77959359e-01 4.22040641e-01]
 [5.77959359e-01 4.22040641e-01]
```

```
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[7.69285180e-05 9.70073640e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[2.49548510e-01 5.59813261e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[3.23305845e-01 5.08380771e-01]
[5.77959359e-01 4.22040641e-01]
[1.00023881e-01 6.51768863e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[8.23515584e-04 9.31306243e-01]
[4.31310125e-02 7.49235272e-01]
[1.76845327e-01 5.86128175e-01]
[5.68593025e-01 4.84815925e-01]
[4.90521230e-02 7.52586901e-01]
[3.62644275e-03 8.90804470e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[4.09923643e-02 7.84252644e-01]
[5.77959359e-01 4.22040641e-01]
[1.31970659e-01 6.22152388e-01]
[5.77959359e-01 4.22040641e-01]
[2.59812981e-01 5.75730562e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[7.06837920e-04 9.26462293e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.59370359e-03 8.85855436e-01]
[1.28887251e-01 7.83499241e-01]
[4.03523929e-02 7.22781360e-01]
[5.77959359e-01 4.22040641e-01]
[2.40196921e-02 7.79173195e-01]
[5.77959359e-01 4.22040641e-01]
[5.72034895e-01 4.52818781e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[2.09087636e-02 7.75091648e-01]
[5.77627540e-01 4.24240202e-01]
```

```
[1.70748413e-01 6.13898277e-01]
[2.04610780e-01 6.07342064e-01]
[5.77959359e-01 4.22040641e-01]
[5.65077543e-01 4.35270816e-01]
[2.03840015e-03 8.99174571e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[1.00454226e-01 6.70888305e-01]
[8.74917060e-02 6.74315572e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[3.08988839e-02 7.45183170e-01]
[9.38767102e-03 8.17617953e-01]
[1.12710828e-02 8.27176332e-01]
[5.77959359e-01 4.22040641e-01]
[1.37505340e-04 9.59641218e-01]
[5.77959359e-01 4.22040641e-01]
[1.10836532e-02 8.19107234e-01]
[5.77959359e-01 4.22040641e-01]
[4.51824740e-02 7.64120936e-01]
[5.06629646e-02 7.05935895e-01]
[5.71090639e-01 4.67980057e-01]
[1.67475009e-05 9.83766377e-01]
[5.77959359e-01 4.22040641e-01]
[2.90351987e-01 6.06881380e-01]
[5.77959359e-01 4.22040641e-01]
[5.76036274e-01 4.34821486e-01]
[1.57833710e-01 6.33929312e-01]
[5.77959359e-01 4.22040641e-01]
[1.83267757e-01 5.72047055e-01]
[4.57295886e-04 9.53726292e-01]
[8.79080296e-02 6.80167794e-01]
[5.64660847e-01 4.19259369e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[4.58992757e-02 7.31070280e-01]
[4.57303852e-01 4.64423686e-01]
[5.77959359e-01 4.22040641e-01]
[5.77959359e-01 4.22040641e-01]
[5.76747119e-01 4.30088043e-01]
[4.23274130e-01 4.82935667e-01]
[8.18702916e-04 9.29347813e-01]
[1.14033902e-02 8.00209939e-01]
[4.72357303e-01 4.87011939e-01]]
```

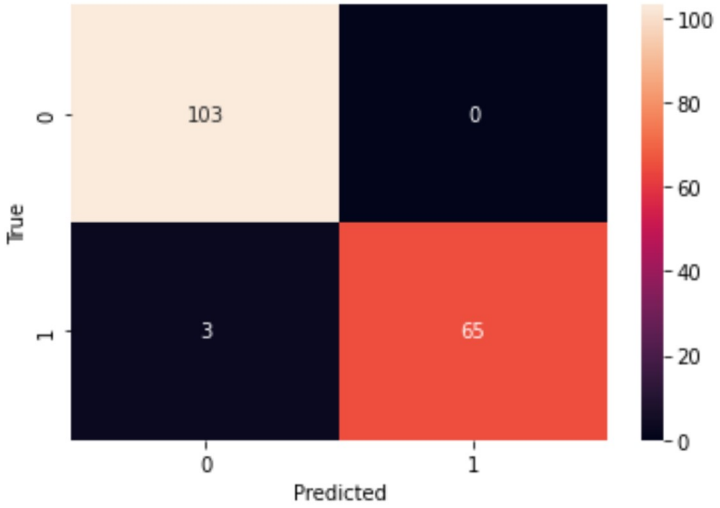
In [37]:

```python
# converting the prediction probability to class labels

Y_pred_labels1 = [np.argmax(i) for i in Y_pred1]
Y_pred_labels2 = [np.argmax(i) for i in Y_pred2]
Y_pred_labels3 = [np.argmax(i) for i in Y_pred3]
print(Y_pred_labels1)
```

```
[0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1
, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0,
0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1,
0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1,
1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
0, 1, 0, 0, 0, 0, 1, 1, 1, 1]
```

In [38]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print("Matriz de confusão do modelo 1")
CV = confusion_matrix(y_true=y_test,y_pred=Y_pred_labels1)
sns.heatmap(CV,annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("Métricas de avaliação do modelo 1")
print(classification_report(y_test, Y_pred_labels1))
```

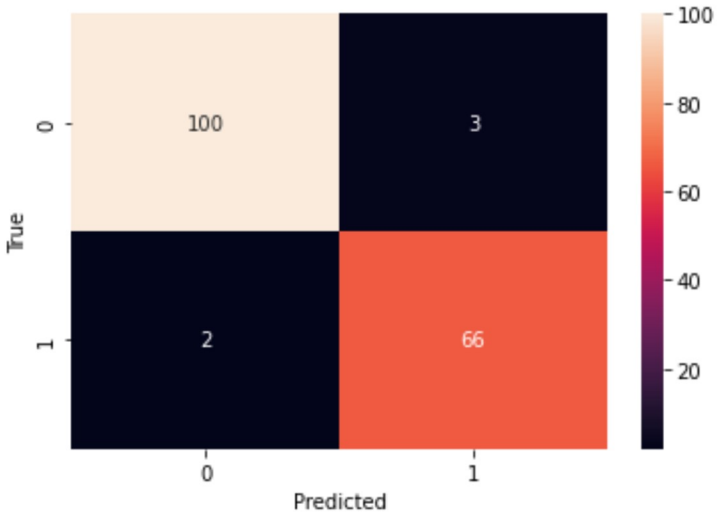
```
Matriz de confusão do modelo 1
```




```
Métricas de avaliação do modelo 1
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       103
           1       1.00      0.96      0.98        68

    accuracy                           0.98       171
   macro avg       0.99      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

In [39]:
```
print("Matriz de confusão do modelo 2")
CV2 = confusion_matrix(y_true=y_test,y_pred=Y_pred_labels2)
sns.heatmap(CV2,annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("Métricas de avaliação do modelo 2")
print(classification_report(y_test, Y_pred_labels2))
```

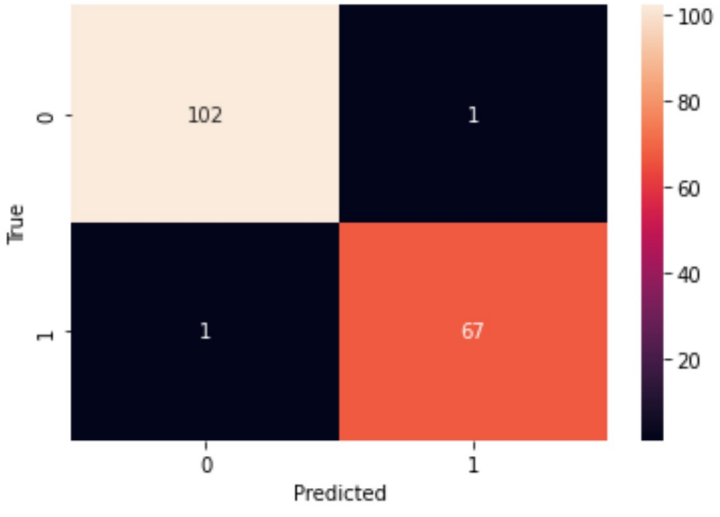
```
Matriz de confusão do modelo 2
```




```
Métricas de avaliação do modelo 2
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       103
           1       0.96      0.97      0.96        68

    accuracy                           0.97       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.97      0.97      0.97       171
```

In [40]:
```
print("Matriz de confusão do modelo 3")
CV3 = confusion_matrix(y_true=y_test,y_pred=Y_pred_labels3)
sns.heatmap(CV3, annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("Métricas de avaliação do modelo 3")
print(classification_report(y_test, Y_pred_labels3))
```

```
Matriz de confusão do modelo 3
```



Métricas de avaliação do modelo 3

	precision	recall	f1-score	support
0	0.99	0.99	0.99	103
1	0.99	0.99	0.99	68
accuracy			0.99	171
macro avg	0.99	0.99	0.99	171
weighted avg	0.99	0.99	0.99	171

Cross Validation

In [41]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
# Define the K-fold cross validator
num_folds = 10
kfold = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)

# K-fold Cross Validation model evaluation
fold_no = 1
acc_per_fold = []
loss_per_fold = []
f1score_per_fold = []

for train, test in kfold.split(X_train, y_train):

  # Define the model architecture
  model4 = tf.keras.Sequential([
      tf.keras.Input(shape=(30,)),
      tf.keras.layers.Dense(units=6, activation="relu"),
      tf.keras.layers.Dense(units=2, activation="relu"),
      tf.keras.layers.Dense(units=2, activation="sigmoid")
  ])


  # compiling the Neural Network
  model4.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

  # Fit data to model
  history4 = model4.fit(X_train, y_train,
              epochs=30,
              verbose=0)

  # Generate generalization metrics
  scores = model4.evaluate(X_test, y_test, verbose=0)
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])


  Y_pred4 = model4.predict(X_test)
  Y_pred_labels4 = [np.argmax(i) for i in Y_pred4]
  f1score_per_fold.append(f1_score(y_test, Y_pred_labels4, average=None))

  # Increase fold number
  fold_no = fold_no + 1

# == Provide average scores ==
print(f'Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print(f'> F1score: {np.mean(f1score_per_fold)}')
print('------------------------------------------------------------------------')
```

```
6/6 [==============================] - 0s 958us/step
6/6 [==============================] - 0s 837us/step
6/6 [==============================] - 0s 828us/step
6/6 [==============================] - 0s 844us/step
6/6 [==============================] - 0s 882us/step
6/6 [==============================] - 0s 748us/step
6/6 [==============================] - 0s 762us/step
6/6 [==============================] - 0s 824us/step
6/6 [==============================] - 0s 830us/step
6/6 [==============================] - 0s 926us/step
Average scores for all folds:
> Accuracy: 98.01169753074646 (+- 0.2864889355229654)
> Loss: 0.19774188697338105
> F1score: 0.9791184368212887
------------------------------------------------------------------------
```

In [42]:
```
visualizer(model4, file_format='png', view=True, settings=None)
```

Decision Tree

In [43]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
dt=DecisionTreeClassifier(criterion = "entropy",random_state=20)
dt.fit(X_train,y_train)
```

Out[43]:
```
DecisionTreeClassifier(criterion='entropy', random_state=20)
```

In [44]:
```
dt_pred=dt.predict(X_test)
dt_pred
```

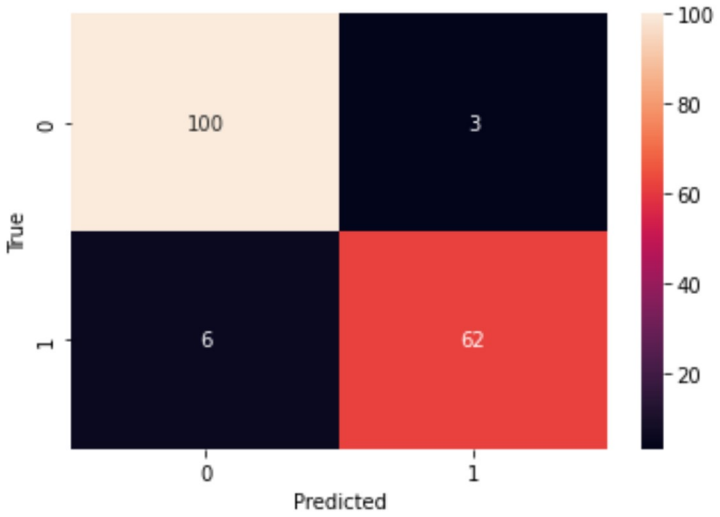
Out[44]:
```
array([0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1])
```

In [45]:
```
f1score_dt=f1_score(y_test, dt_pred, average=None)
print(f'> F1score DT: {np.mean(f1score_dt)}')
```

```
> F1score DT: 0.9446343130553656
```

In [46]:
```
print("Matriz de confusão da DT")
CV4 = confusion_matrix(y_true=y_test,y_pred=dt_pred)
sns.heatmap(CV4, annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("Métricas de avaliação da DT")
print(classification_report(y_test, dt_pred))
```

```
Matriz de confusão da DT
```




```
Métricas de avaliação da DT
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       103
           1       0.95      0.91      0.93        68

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

In [ ]: