```python
In [26]: import pandas as pd
         import tensorflow as tf
         from tensorflow import keras
         from keras.models import Sequential
         from keras.layers import Dense, Activation, Dropout
         from keras.utils import to_categorical
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import LabelEncoder, StandardScaler
         from sklearn.metrics import confusion_matrix
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline
         sns.set_style("whitegrid")
         plt.style.use('fivethirtyeight')
```

```python
In [3]: # Load the data
        df=pd.read_csv('college.csv', header=0)
```

```python
In [4]: # Dropping columns
        features=list(df.columns.values)
        features.remove("Private")

        print(features)
```

```
['School', 'Apps', 'Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F_Undergrad',
'P_Undergrad', 'Outstate', 'Room_Board', 'Books', 'Personal', 'PhD', 'Terminal',
'S_F_Ratio', 'perc_alumni', 'Expend', 'Grad_Rate']
```

```python
In [5]: features.remove("School")
        print(features)
```

```
['Apps', 'Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F_Undergrad', 'P_Undergr
ad', 'Outstate', 'Room_Board', 'Books', 'Personal', 'PhD', 'Terminal', 'S_F_Rati
o', 'perc_alumni', 'Expend', 'Grad_Rate']
```

```python
In [6]: # Creating input features and target variables
        X=df[features]
        y=df['Private']
```

```
In [7]:  # Correcting skewed data - Grad rate has an outlier that showed 120% for one of the
         schools
         plt.figure(figsize=(10, 10))

         df.loc[df.Private == 'Yes', 'Grad_Rate'].hist(label="Private College", bins=30)
         df.loc[df.Private == 'No', 'Grad_Rate'].hist(label="Public College", bins=30)

         plt.xlabel('Graduation Rate')
         plt.legend()
```

Out[7]: <matplotlib.legend.Legend at 0x2093c3a58c8>
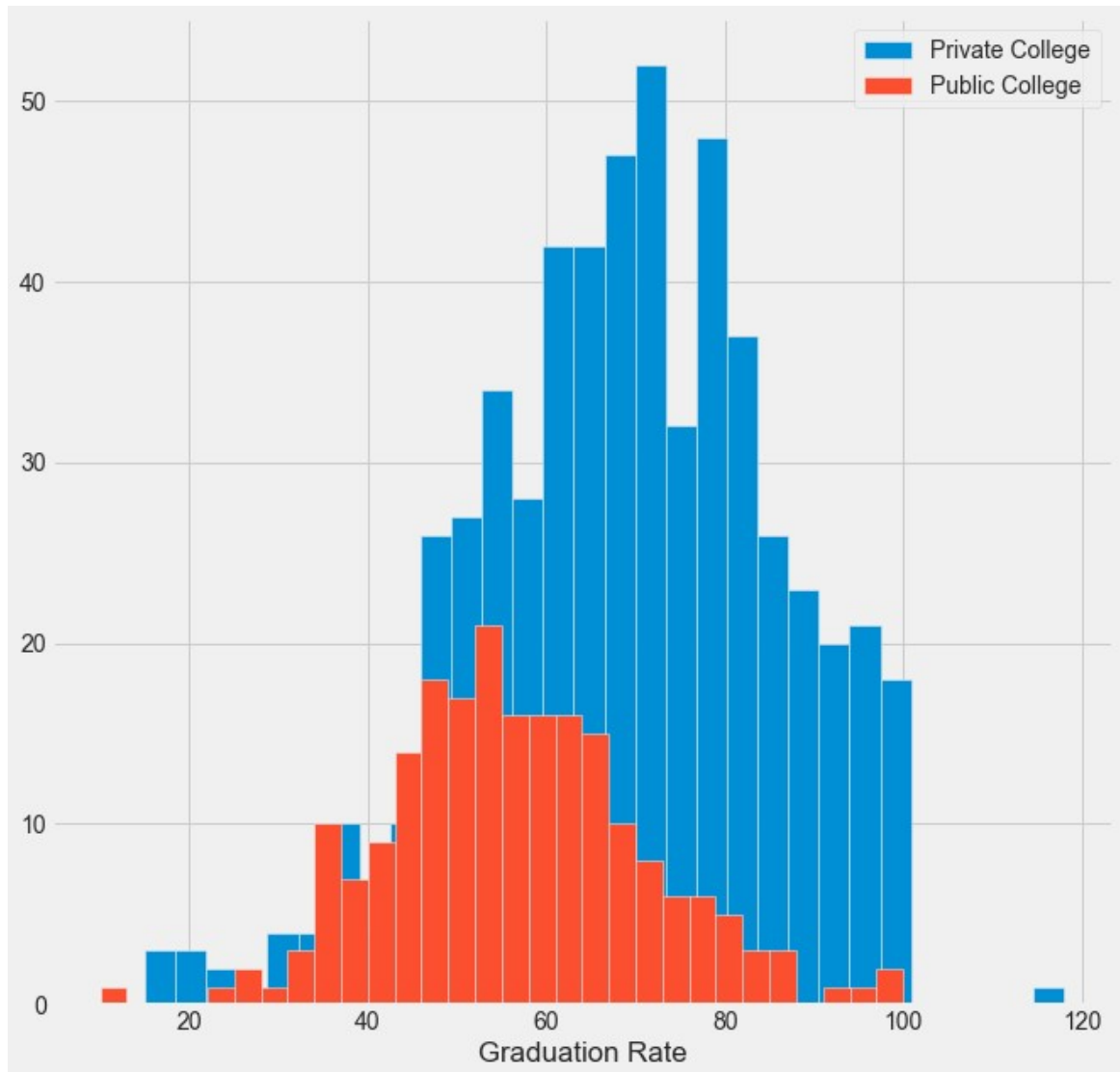

```
In [8]:  df.loc[df.Grad_Rate > 100]
```

Out[8]:

|	School	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F_Undergrad	P_Undergrad	Outstate	Ro
95	Cazenovia College	Yes	3847	3433	527	9	35	1010	12	9384	

```
In [9]:  df.loc[df.Grad_Rate>100, 'Grad_Rate']=100
```

```
In [10]:  plt.figure(figsize=(10, 10))

          df.loc[df.Private == 'Yes', 'Grad_Rate'].hist(label="Private College", bins=30)
          df.loc[df.Private == 'No', 'Grad_Rate'].hist(label="Public College", bins=30)

          plt.xlabel('Graduation Rate')
          plt.legend()
```

Out[10]: <matplotlib.legend.Legend at 0x2093c7cdb08>
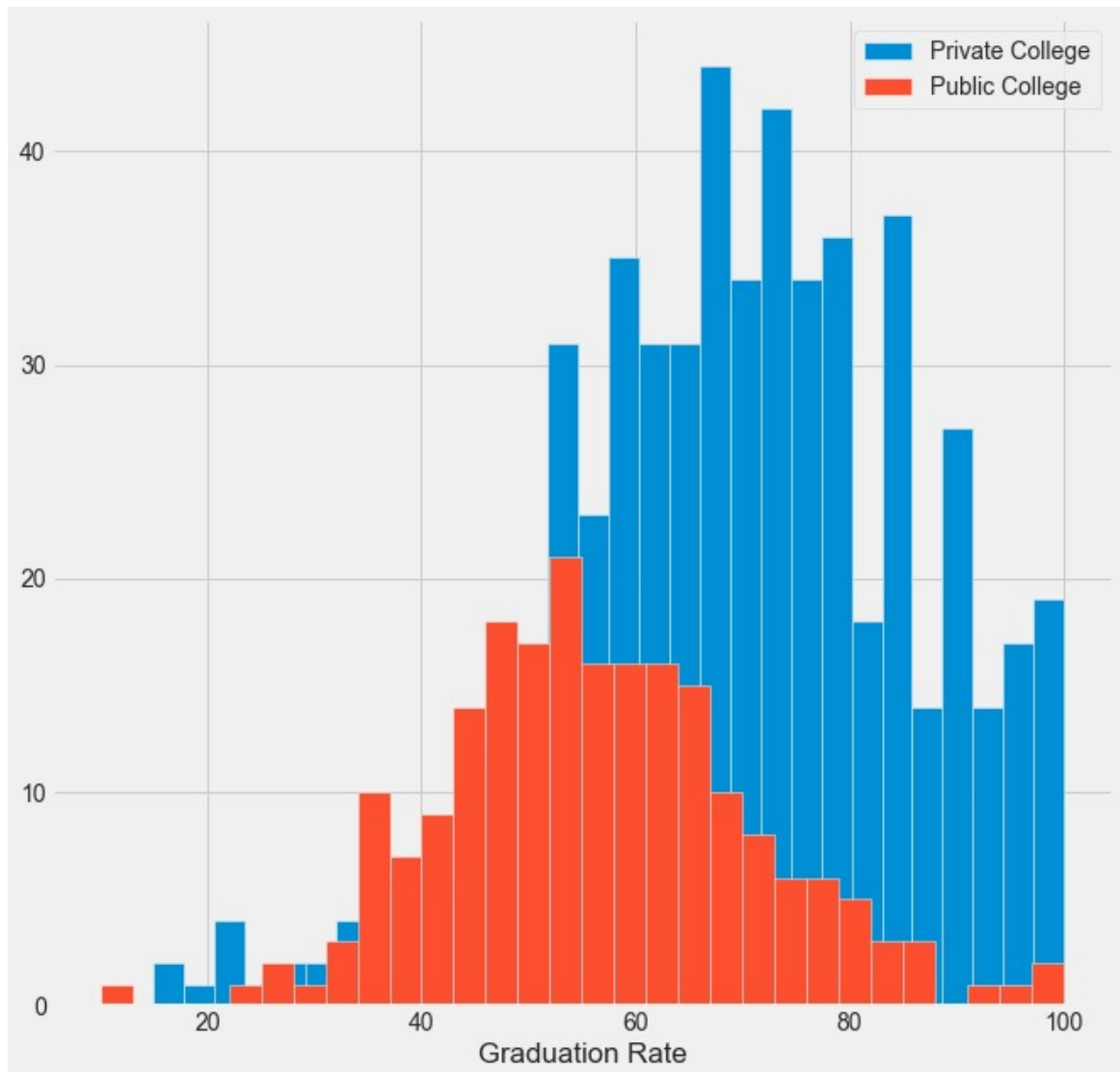
```
In [11]:  # Checking for outliers in 'Apps'
          plt.figure(figsize=(10, 10))

          df.loc[df.Private == 'Yes', 'Apps'].hist(label="Private College", bins=30)
          df.loc[df.Private == 'No', 'Apps'].hist(label="Non Private College", bins=30)

          plt.xlabel('Number of Applications')
          plt.legend()
```
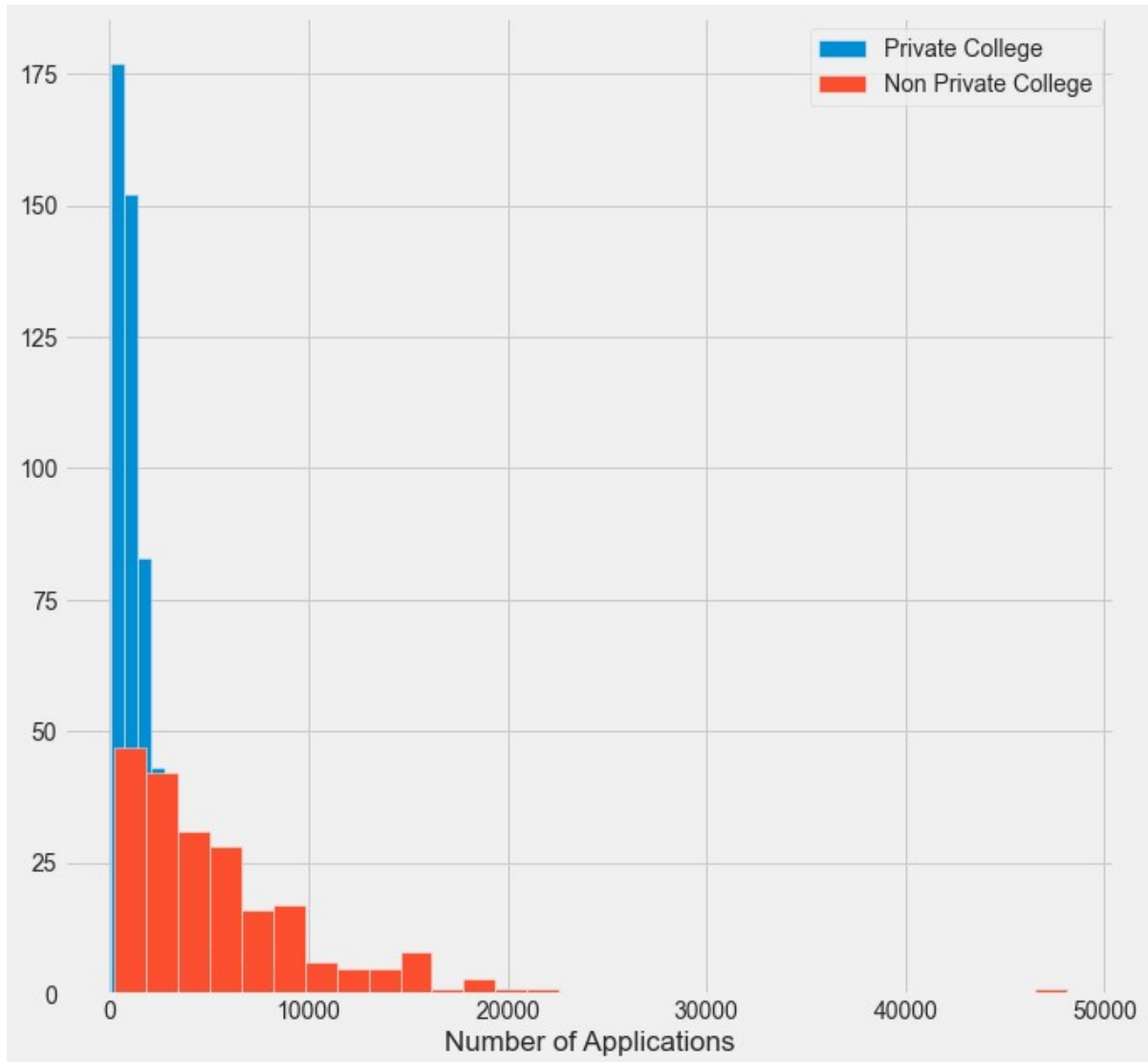
Out[11]: <matplotlib.legend.Legend at 0x2093c8deec8>
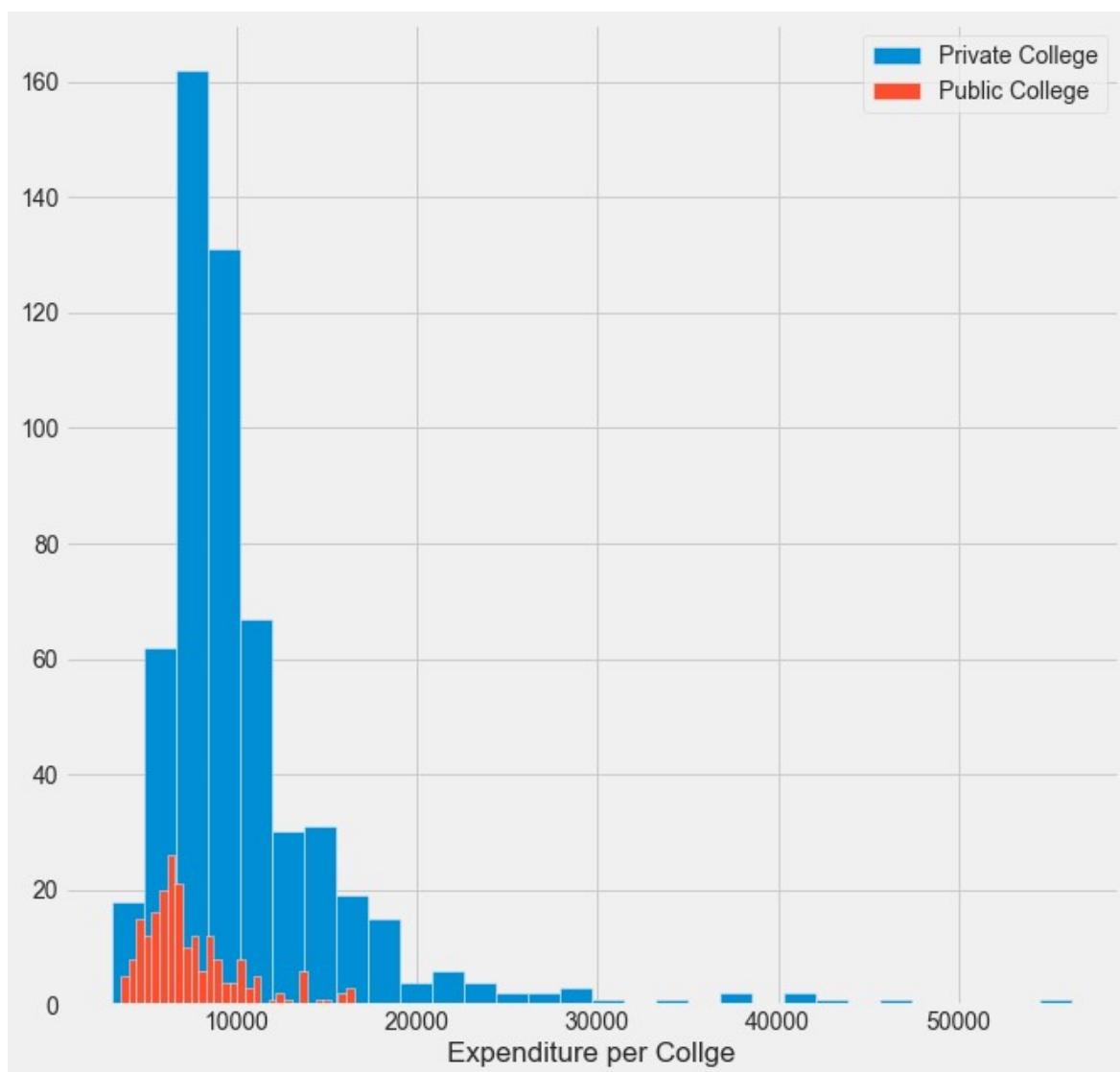


```
In [12]:  df.loc[df.Apps > 20000]
```

Out[12]:

	School	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F_Undergrad	P_Undergrad	Outstate	R
59	Boston University	Yes	20192	13007	3810	45	80	14971	3113	18420	
461	Purdue University at West Lafayette	No	21804	18744	5874	29	60	26213	4065	9556	
483	Rutgers at New Brunswick	No	48094	26330	4520	36	79	21401	3712	7410	

```
In [13]:  # Checking for outliers in 'Expend'
          plt.figure(figsize=(10, 10))

          df.loc[df.Private == 'Yes', 'Expend'].hist(label="Private College", bins=30)
          df.loc[df.Private == 'No', 'Expend'].hist(label="Public College", bins=30)

          plt.xlabel('Expenditure per Collge')
          plt.legend()
```

Out[13]: <matplotlib.legend.Legend at 0x2093cca3508>

```
In [14]: df.loc[df.Expend > 40000]
```

Out[14]:

|	School	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F_Undergrad	P_Undergrad	Outstate
20	Antioch University	Yes	713	661	252	25	44	712	23	15476
284	Johns Hopkins University	Yes	8474	3446	911	75	94	3566	1569	18800
720	Wake Forest University	Yes	5661	2392	903	75	88	3499	172	13850
728	Washington University	Yes	7654	5259	1254	62	93	4879	1274	18350
775	Yale University	Yes	10705	2453	1317	95	99	5217	83	19840

```
In [15]: # Comparing 'out-of-state' tuition with Boxplots
         import matplotlib.pyplot as plt
         fig = plt.figure(figsize=(10, 10))
         ax = fig.add_subplot(111)

         sns.boxplot(y="Outstate", x="Private", data=df)
         ax.set_xlabel('Private')
         ax.set_ylabel('Outstate Tution')
         ax.set_title('Outstate Tution vs College Type')
         plt.show()
```
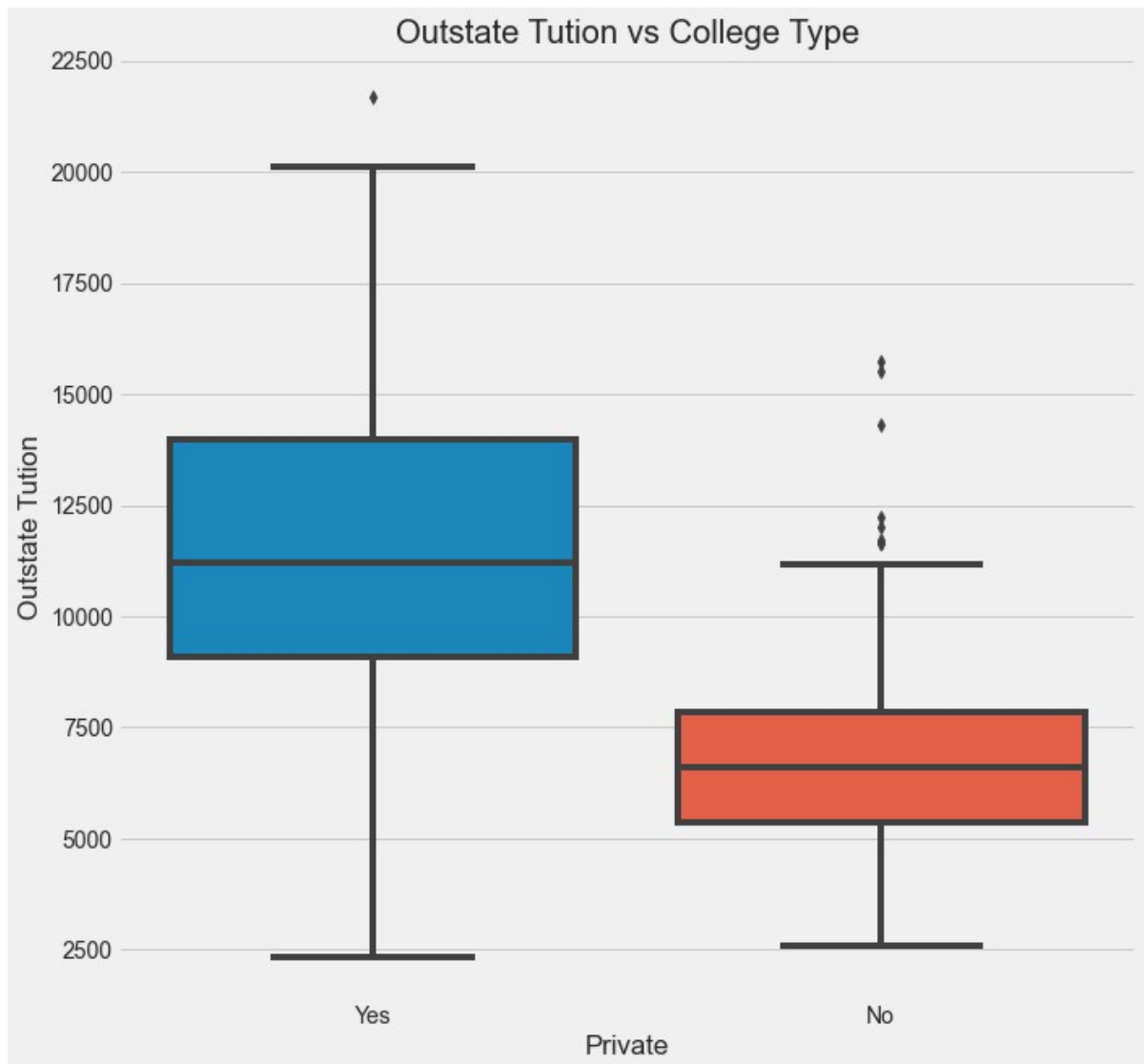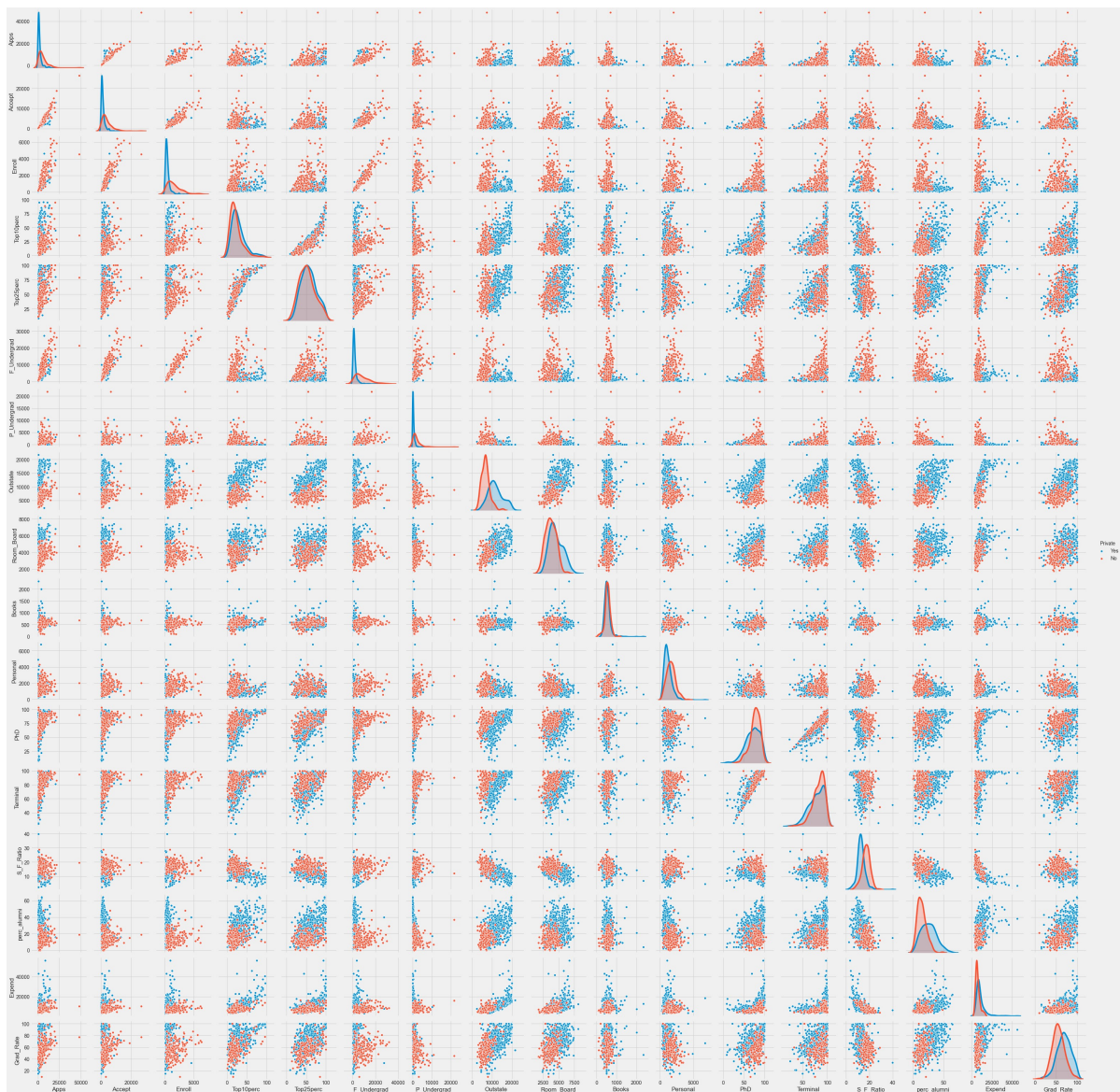
In [16]:
```python
# # Comparing % faculty with 'PhD' with Boxplots
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111)

sns.boxplot(x="Private", y="PhD", data=df)
ax.set_xlabel('Private')
ax.set_ylabel('PhD')
ax.set_title('PhD Graduates vs Non-PhD Graduates')
plt.show()
```
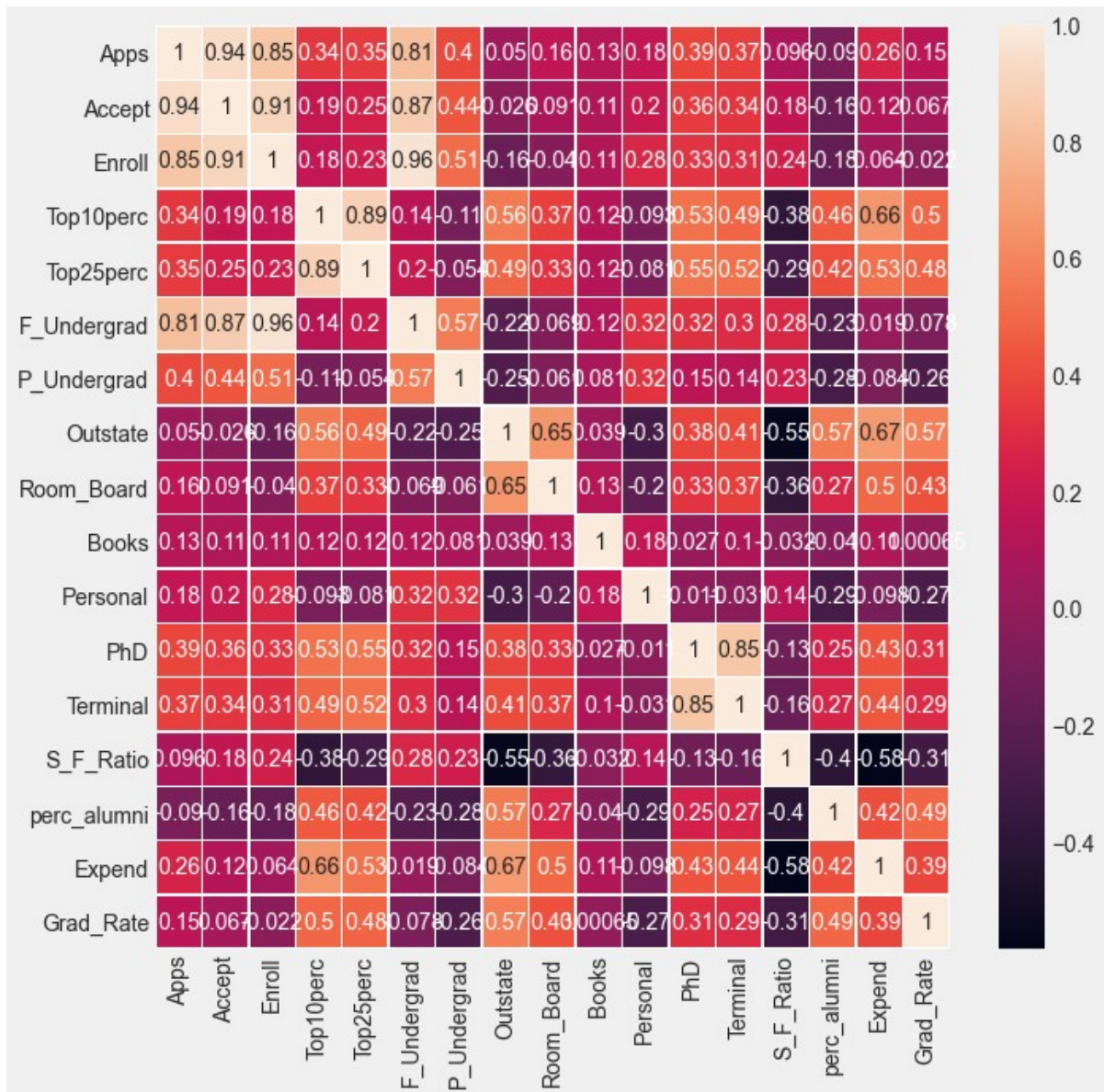

PhD Graduates vs Non-PhD Graduates
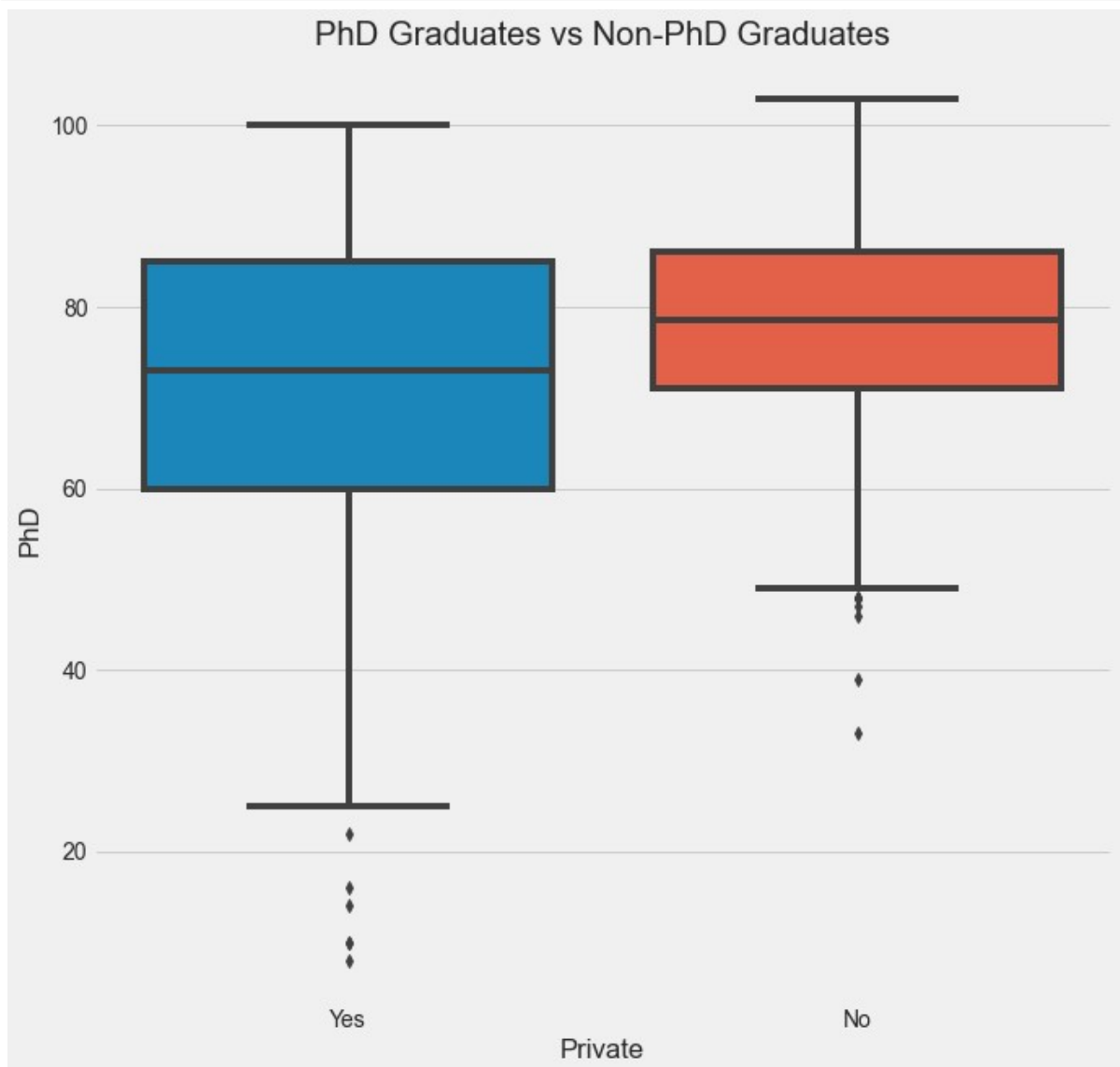
In [17]: ```python
# Identifying correlations with the target variable
sns.pairplot(df, hue='Private')
```

Out[17]: `<seaborn.axisgrid.PairGrid at 0x2093cdb5d48>`

```
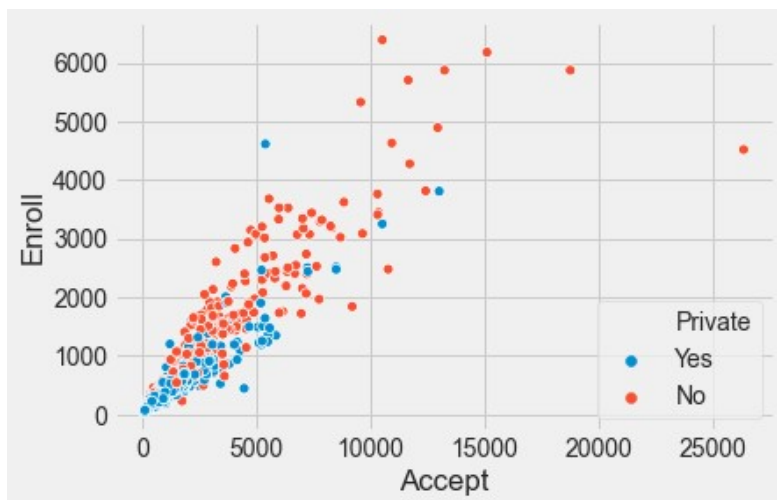In [18]:   # Identifying the features most related to the target vaiable
           # using numerical values
           fig, ax = plt.subplots(figsize=(10,10))
           sns.heatmap(df.corr(), annot=True, linewidth=.5, ax=ax)
```

Out[18]:   <matplotlib.axes._subplots.AxesSubplot at 0x20947f01f48>

```
In [19]:  # Correlation between 'Accept' and 'Enroll'
          sns.scatterplot(x=df["Accept"], y=df["Enroll"], data=df, hue="Private")
          plt.figure(figsize=(12, 12))
```

Out[19]: <Figure size 864x864 with 0 Axes>

<Figure size 864x864 with 0 Axes>

```
In [20]:  # Correlation between 'out-of-state tuition' and
          # % of fuculty with 'Ph.D.'
          sns.scatterplot(x=df["Grad_Rate"], y=df["Outstate"], data=df, hue="Private")
          plt.figure(figsize=(12, 12))
```

Out[20]: <Figure size 864x864 with 0 Axes>

<Figure size 864x864 with 0 Axes>

```
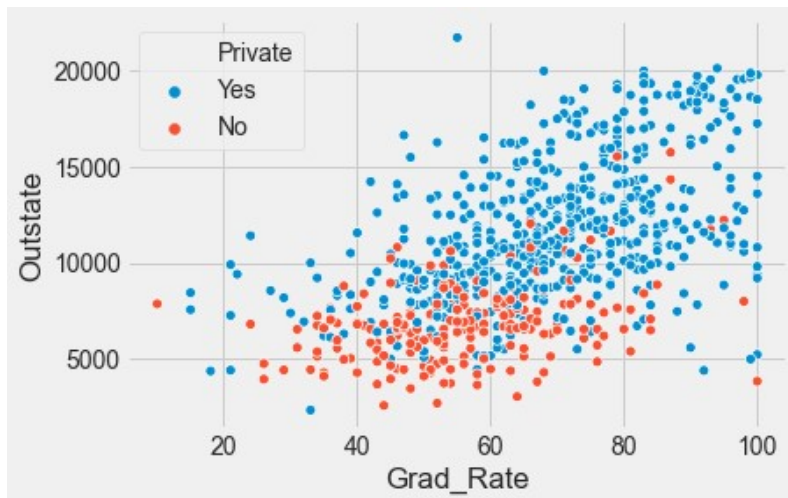In [21]:  # Chenging the target variable from string to numerical data
          y=df['Private'].astype('category').cat.codes
```

```
In [22]: print(y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
772    0
773    1
774    1
775    1
776    1
Length: 777, dtype: int8
```

```
In [23]: df.head()
```

Out[23]:

	School	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F_Undergrad	P_Undergrad	Outstate	Roor
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	
1	Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	
2	Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	
3	Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	
4	Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	

```
In [24]: df.Private.value_counts()
```

```
Out[24]: Yes    565
         No     212
         Name: Private, dtype: int64
```

```
In [30]: # Standardizing the input features
         scaler = StandardScaler()
         X = scaler.fit_transform(X)
         X
```

```
Out[30]: array([[-3.46881819e-01, -3.21205453e-01, -6.35089011e-02, ...,
                 -8.67574189e-01, -5.01910084e-01, -3.18251941e-01],
                [-2.10884040e-01, -3.87029908e-02, -2.88584214e-01, ...,
                 -5.44572203e-01,  1.66109850e-01, -5.51261842e-01],
                [-4.06865631e-01, -3.76317928e-01, -4.78121319e-01, ...,
                  5.85934748e-01, -1.77289956e-01, -6.67766793e-01],
                ...,
                [-2.33895071e-01, -4.23771558e-02, -9.15087008e-02, ...,
                 -2.21570217e-01, -2.56241250e-01, -9.59029170e-01],
                [ 1.99171118e+00,  1.77256262e-01,  5.78332661e-01, ...,
                  2.12019418e+00,  5.88797079e+00,  1.95359460e+00],
                [-3.26765760e-03, -6.68715889e-02, -9.58163623e-02, ...,
                  4.24433755e-01, -9.87115613e-01,  1.95359460e+00]])
```

```
In [31]:  # Split input features and target variable into training and testing data sets
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
          te=1)
          X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
          random_state=1)
```

```
In [33]:  # Use a Sequential model to build a Neural Netwrok.
          # Use a dense library to build 'input', 'hidden' and 'output' layers
          model=Sequential()
          model.add(Dense(60, input_dim=17, activation='relu'))
          model.add(Dense(1, activation='sigmoid'))
```

```
In [34]:  # Compile the Neural Network, we use 'Adam' to optimize our neural network
          model.compile(loss='binary_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
```

```
In [35]: # Fit the training data to the model, using a batch_size of 20, iterating over 20 E
         pochs
         seqModel=model.fit(X_train, y_train, epochs=20, batch_size=20, validation_data=(X_v
         al, y_val))
```

```
Train on 434 samples, validate on 109 samples
Epoch 1/20
434/434 [==============================] - 0s 375us/step - loss: 0.5608 - accura
cy: 0.7373 - val_loss: 0.4663 - val_accuracy: 0.8716
Epoch 2/20
434/434 [==============================] - 0s 87us/step - loss: 0.4095 - accurac
y: 0.9101 - val_loss: 0.3531 - val_accuracy: 0.9266
Epoch 3/20
434/434 [==============================] - 0s 72us/step - loss: 0.3284 - accurac
y: 0.9217 - val_loss: 0.2872 - val_accuracy: 0.9266
Epoch 4/20
434/434 [==============================] - 0s 108us/step - loss: 0.2783 - accura
cy: 0.9240 - val_loss: 0.2454 - val_accuracy: 0.9358
Epoch 5/20
434/434 [==============================] - 0s 72us/step - loss: 0.2456 - accurac
y: 0.9332 - val_loss: 0.2165 - val_accuracy: 0.9358
Epoch 6/20
434/434 [==============================] - 0s 72us/step - loss: 0.2228 - accurac
y: 0.9355 - val_loss: 0.1951 - val_accuracy: 0.9450
Epoch 7/20
434/434 [==============================] - 0s 51us/step - loss: 0.2059 - accurac
y: 0.9424 - val_loss: 0.1806 - val_accuracy: 0.9358
Epoch 8/20
434/434 [==============================] - 0s 72us/step - loss: 0.1938 - accurac
y: 0.9447 - val_loss: 0.1695 - val_accuracy: 0.9450
Epoch 9/20
434/434 [==============================] - 0s 87us/step - loss: 0.1843 - accurac
y: 0.9470 - val_loss: 0.1611 - val_accuracy: 0.9450
Epoch 10/20
434/434 [==============================] - 0s 144us/step - loss: 0.1764 - accura
cy: 0.9493 - val_loss: 0.1533 - val_accuracy: 0.9450
Epoch 11/20
434/434 [==============================] - 0s 106us/step - loss: 0.1695 - accura
cy: 0.9539 - val_loss: 0.1475 - val_accuracy: 0.9450
Epoch 12/20
434/434 [==============================] - 0s 102us/step - loss: 0.1646 - accura
cy: 0.9539 - val_loss: 0.1445 - val_accuracy: 0.9541
Epoch 13/20
434/434 [==============================] - 0s 56us/step - loss: 0.1593 - accurac
y: 0.9539 - val_loss: 0.1399 - val_accuracy: 0.9450
Epoch 14/20
434/434 [==============================] - 0s 72us/step - loss: 0.1550 - accurac
y: 0.9539 - val_loss: 0.1371 - val_accuracy: 0.9541
Epoch 15/20
434/434 [==============================] - 0s 108us/step - loss: 0.1521 - accura
cy: 0.9539 - val_loss: 0.1389 - val_accuracy: 0.9450
Epoch 16/20
434/434 [==============================] - 0s 114us/step - loss: 0.1483 - accura
cy: 0.9562 - val_loss: 0.1348 - val_accuracy: 0.9541
Epoch 17/20
434/434 [==============================] - 0s 72us/step - loss: 0.1452 - accurac
y: 0.9539 - val_loss: 0.1333 - val_accuracy: 0.9450
Epoch 18/20
434/434 [==============================] - 0s 56us/step - loss: 0.1424 - accurac
y: 0.9539 - val_loss: 0.1317 - val_accuracy: 0.9450
Epoch 19/20
434/434 [==============================] - 0s 108us/step - loss: 0.1399 - accura
cy: 0.9562 - val_loss: 0.1300 - val_accuracy: 0.9450
Epoch 20/20
434/434 [==============================] - 0s 87us/step - loss: 0.1373 - accurac
y: 0.9562 - val_loss: 0.1282 - val_accuracy: 0.9450
```

```
In [36]:   # Evaluate the model
           eval_model=model.evaluate(X_train, y_train)
           eval_model

           434/434 [==============================] - 0s 0us/step

Out[36]:   [0.13494041170667392, 0.9562212228775024]


In [37]:   # Predict the output for the test dataset
           y_pred=model.predict(X_test)
           y_pred =(y_pred>0.5)
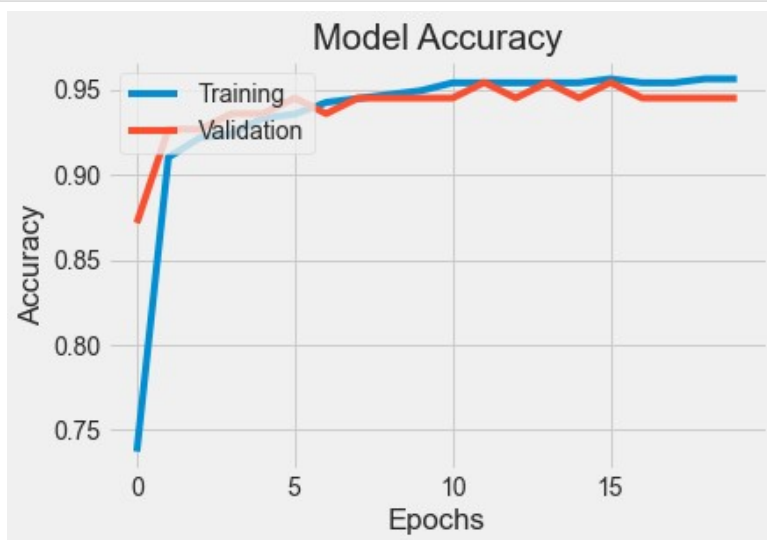

In [38]:   # check the accuracy on the test dataset
           cm = confusion_matrix(y_test, y_pred)
           print(cm)

           [[ 54  10]
            [  8 162]]
```

Summary

The True positive and True negative result is 218 out of a toal 234 observations, in other words the accuracy for the test dataset is 93%. This means that we can predict whether a school is private or public (based on a combination of features) accurately 93% of the time.

```
In [39]:   # Plot model accuracy
           plt.plot(seqModel.history['accuracy'])
           plt.plot(seqModel.history['val_accuracy'])
           plt.title('Model Accuracy')
           plt.ylabel('Accuracy')
           plt.xlabel('Epochs')
           plt.legend(['Training','Validation'],loc='upper left')
           plt.show()
```

In [40]:
```python
# Plot model loss
plt.plot(seqModel.history['loss'])
plt.plot(seqModel.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Training', 'Validation'], loc='upper left')
plt.show()
```
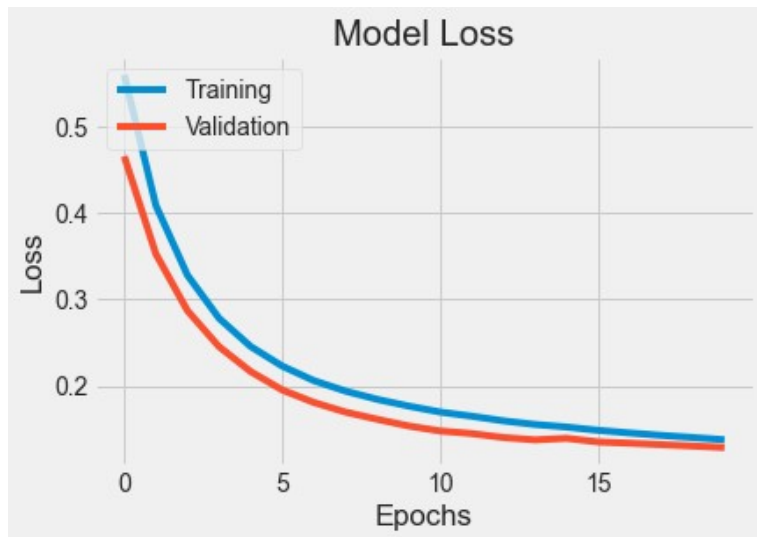

In [43]:
```python
# Save the model and architecture to a file
model.save('model.h5')
print('Save model to disk')
```

```
Save model to disk
```

In [ ]: