Part 2 Midterm ADTA 5550 Deep Learning with Big Data

In [29]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /opt/conda/lib/python3.7/site-packages
(1.15.5)
Requirement already satisfied: wrapt>=1.11.1 in /opt/conda/lib/python3.7/site-package
s (from tensorflow) (1.14.1)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /opt/conda/lib/python3.
7/site-packages (from tensorflow) (1.1.2)
Requirement already satisfied: tensorboard<1.16.0,>=1.15.0 in /opt/conda/lib/python3.
7/site-packages (from tensorflow) (1.15.0)
Requirement already satisfied: google-pasta>=0.1.6 in /opt/conda/lib/python3.7/site-p
ackages (from tensorflow) (0.2.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/conda/lib/python3.7/site-pack
ages (from tensorflow) (2.1.0)
Requirement already satisfied: numpy<1.19.0,>=1.16.0 in /opt/conda/lib/python3.7/site
-packages (from tensorflow) (1.18.5)
Requirement already satisfied: protobuf>=3.6.1 in /opt/conda/lib/python3.7/site-packa
ges (from tensorflow) (3.20.3)
Requirement already satisfied: h5py<=2.10.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (2.10.0)
Requirement already satisfied: astor>=0.6.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (0.8.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/conda/lib/python3.7/site-pac
kages (from tensorflow) (3.3.0)
Requirement already satisfied: tensorflow-estimator==1.15.1 in /opt/conda/lib/python
3.7/site-packages (from tensorflow) (1.15.1)
Requirement already satisfied: absl-py>=0.7.0 in /opt/conda/lib/python3.7/site-packag
es (from tensorflow) (0.8.1)
Requirement already satisfied: gast==0.2.2 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (0.2.2)
Requirement already satisfied: six>=1.10.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (1.16.0)
Requirement already satisfied: grpcio>=1.8.6 in /opt/conda/lib/python3.7/site-package
s (from tensorflow) (1.50.0)
Requirement already satisfied: keras-applications>=1.0.8 in /opt/conda/lib/python3.7/
site-packages (from tensorflow) (1.0.8)
Requirement already satisfied: wheel>=0.26 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (0.37.1)
Requirement already satisfied: werkzeug>=0.11.15 in /opt/conda/lib/python3.7/site-pac
kages (from tensorboard<1.16.0,>=1.15.0->tensorflow) (2.2.2)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/python3.7/site-packa
ges (from tensorboard<1.16.0,>=1.15.0->tensorflow) (3.4.1)
Requirement already satisfied: setuptools>=41.0.0 in /opt/conda/lib/python3.7/site-pa
ckages (from tensorboard<1.16.0,>=1.15.0->tensorflow) (59.8.0)
Requirement already satisfied: importlib-metadata>=4.4 in /opt/conda/lib/python3.7/si
te-packages (from markdown>=2.6.8->tensorboard<1.16.0,>=1.15.0->tensorflow) (4.11.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/conda/lib/python3.7/site-pac
kages (from werkzeug>=0.11.15->tensorboard<1.16.0,>=1.15.0->tensorflow) (2.1.1)
Requirement already satisfied: typing-extensions>=3.6.4 in /opt/conda/lib/python3.7/s
ite-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<1.16.0,>=1.1
5.0->tensorflow) (4.4.0)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packages (f
rom importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<1.16.0,>=1.15.0->tensorflo
w) (3.10.0)
Note: you may need to restart the kernel to use updated packages.
```

In [30]:

```python
import pandas as pd
import numpy as np
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split   #Train and Test data
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.metrics import accuracy_score
```

In [31]:
```
# Load the dataset
data = pd.read_csv("pima_diabetes.csv")

data.head()
```

Out[31]:

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

In [32]:
```
# Assuming the data is in a CSV file without header
file_path = 'pima_diabetes.csv' # Replace with the path to your CSV file
column_names = ['Preg', 'Plas', 'Pres', 'Skin', 'Test', 'Mass', 'Pedi', 'Age', 'Class'

# Load the CSV file into a pandas DataFrame with the specified column names
pima_diabetes_dataframe = pd.read_csv(file_path, header=None, names=column_names)

# Now your dataframe will have the columns named as specified
```

In [33]:
```
pima_diabetes_dataframe.head()
```

Out[33]:

	Preg	Plas	Pres	Skin	Test	Mass	Pedi	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

PART II: MLPs (Fully Connected Neural Networks) with Keras (50 Points)

In [34]:
```
# Perform integer encoding
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

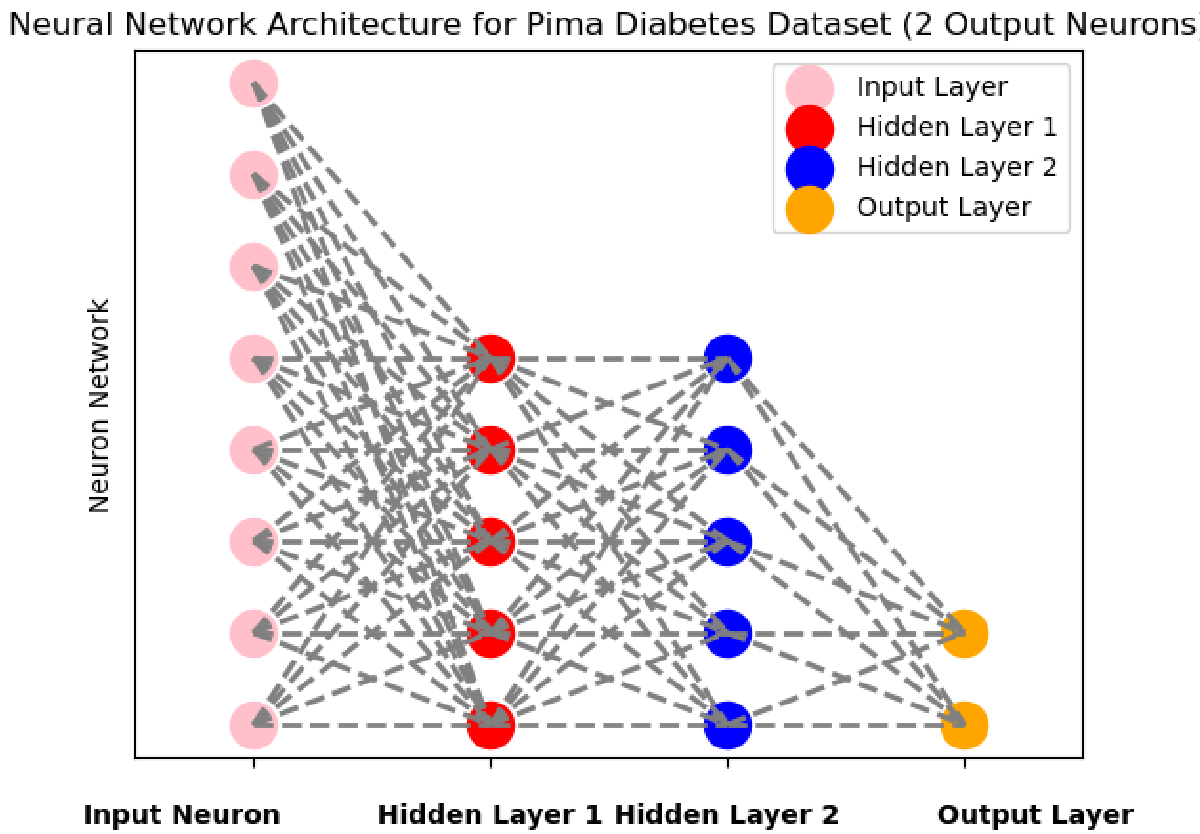
```
from sklearn.preprocessing import StandardScaler

# Load the dataset
#pima_diabetes_dataframe = pd.read_csv('/content/pima_diabetes.csv')

# Split the dataset into features and target
X = pima_diabetes_dataframe.drop(['Class'], axis=1)
y = pima_diabetes_dataframe['Class']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.4, random

# Create and train the MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(5,), max_iter=1000, alpha=0.01, random_state=4
mlp.fit(X_train, y_train)

# Make predictions on the test set
y_pred = mlp.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Calculate normalized accuracy score
normalized_accuracy = accuracy_score(y_test, y_pred, normalize=True)

# Calculate normalized accuracy score
normalized_accuracy = accuracy_score(y_test, y_pred, normalize=True)


print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Normalized Accuracy: {:.2f}%".format(normalized_accuracy * 100))
```

```
Accuracy: 77.27%
Normalized Accuracy: 77.27%
```

In [35]:

```
import matplotlib.pyplot as plt
import numpy as np

# Define the MLP classifier architecture
hidden_layer_sizes = (5, 5)  # Two hidden layers, each with 5 neurons
input_dim = 8  # Number of input features
output_dim = 2  # Number of output classes

# Create the figure and axis objects
fig, ax = plt.subplots()

# Plot the input layer
ax.scatter(np.zeros(input_dim), np.arange(input_dim), color='pink', label='Input Layer
# Plot the first hidden layer
ax.scatter(np.ones(hidden_layer_sizes[0]) * 1, np.arange(hidden_layer_sizes[0]), color
# Plot the second hidden layer
ax.scatter(np.ones(hidden_layer_sizes[1]) * 2, np.arange(hidden_layer_sizes[1]), color
# Plot the output layer
ax.scatter(np.ones(output_dim) * 3, np.arange(output_dim), color='orange', label='Outp

# Connect the input layer to the first hidden layer
```

```python
for i in range(input_dim):
    for j in range(hidden_layer_sizes[0]):
        ax.plot([0, 1], [i, j], color='gray', linewidth=2, linestyle='dashed')

# Connect the first hidden layer to the second hidden layer
for i in range(hidden_layer_sizes[0]):
    for j in range(hidden_layer_sizes[1]):
        ax.plot([1, 2], [i, j], color='gray', linewidth=2, linestyle='dashed')

# Connect the second hidden layer to the output layer
for i in range(hidden_layer_sizes[1]):
    for j in range(output_dim):
        ax.plot([2, 3], [i, j], color='gray', linewidth=2, linestyle='dashed')

# Add labels to the layers
ax.text(-0.3, -1, 'Input Neuron', ha='center', va='center', fontweight='bold')
ax.text(1, -1, 'Hidden Layer 1', ha='center', va='center', fontweight='bold')
ax.text(2, -1, 'Hidden Layer 2', ha='center', va='center', fontweight='bold')
ax.text(3.3, -1, 'Output Layer', ha='center', va='center', fontweight='bold')

# Set the axis limits and labels
ax.set_xlim(-0.5, 3.5)
ax.set_xticks([0, 1, 2, 3])
ax.set_xticklabels(['', '', '', ''])
ax.set_yticks([])
ax.set_ylabel('Neuron Network')

ax.set_title('Neural Network Architecture for Pima Diabetes Dataset (2 Output Neurons)

# Add a legend
ax.legend(loc='upper right')

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

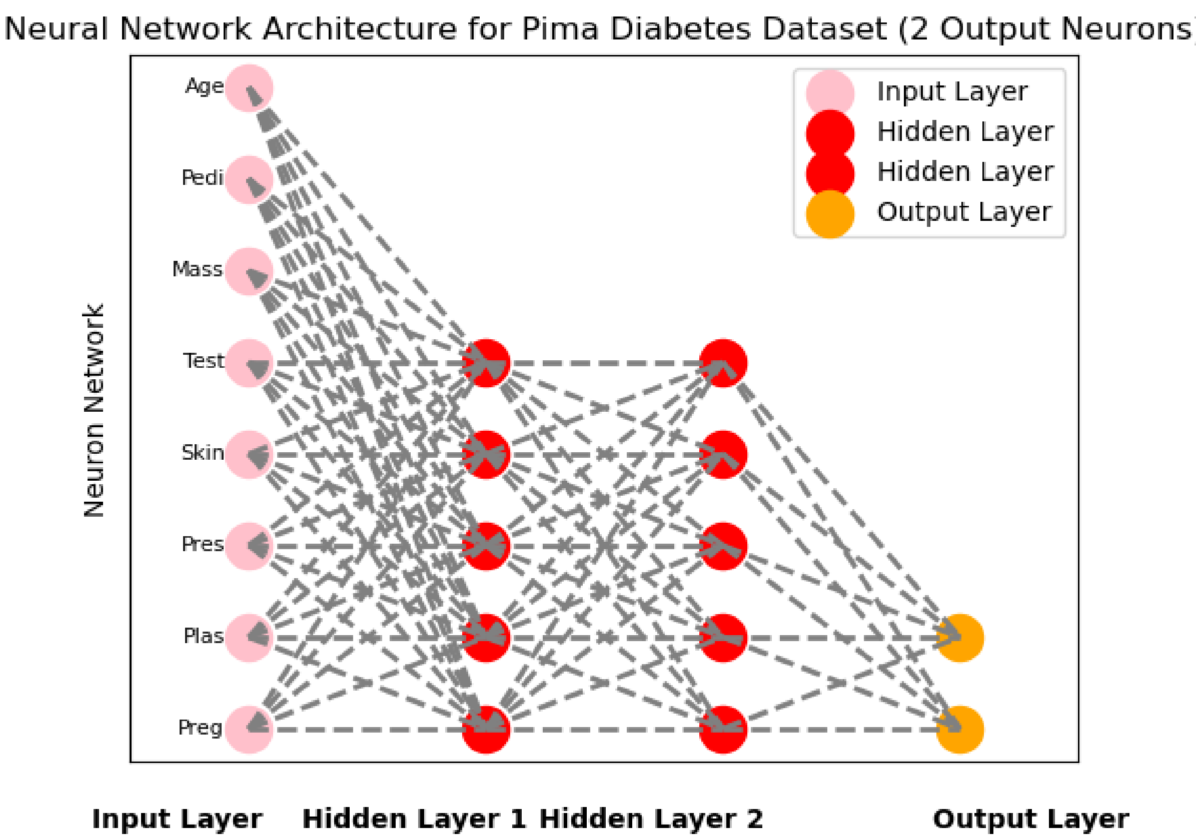


MLP classifier architecture and Feature Names

In [36]:

```
import matplotlib.pyplot as plt
import numpy as np

# Define the MLP classifier architecture
hidden_layer_sizes = (5, 5)  # Number of hidden units in each hidden layer
input_dim = 8  # Number of input features
output_dim = 2  # Number of output classes (assuming binary classification)

# Feature names for the input layer
feature_names = ['Preg', 'Plas', 'Pres', 'Skin', 'Test', 'Mass', 'Pedi', 'Age']

# Create the figure and axis objects
fig, ax = plt.subplots()

# Plot the input layer
input_neurons = ax.scatter(np.zeros(input_dim), np.arange(input_dim), color='pink', la
# Plot the hidden layers
for layer in range(len(hidden_layer_sizes)):
    ax.scatter(np.ones(hidden_layer_sizes[layer]) * (layer + 1), np.arange(hidden_laye
# Plot the output layer
ax.scatter(np.ones(output_dim) * (len(hidden_layer_sizes) + 1), np.arange(output_dim),

# Connect the layers
for layer in range(len(hidden_layer_sizes) + 1):
    for i in range(input_dim if layer == 0 else hidden_layer_sizes[layer - 1]):
        for j in range(hidden_layer_sizes[layer] if layer < len(hidden_layer_sizes) el
```

```
            ax.plot([layer, layer + 1], [i, j], color='gray', linewidth=2, linestyle='

# Add feature names to the input neurons
for i, name in enumerate(feature_names):
    ax.text(-0.1, i, name, ha='right', va='center', fontsize=8)

# Add labels to the layers
ax.text(-0.3, -1, 'Input Layer', ha='center', va='center', fontweight='bold')
for layer in range(len(hidden_layer_sizes)):
    ax.text(layer + 0.7, -1, f'Hidden Layer {layer + 1}', ha='center', va='center', fc
ax.text(len(hidden_layer_sizes) + 1.3, -1, 'Output Layer', ha='center', va='center', f

# Set the axis limits and labels
ax.set_xlim(-0.5, len(hidden_layer_sizes) + 1.5)
ax.set_xticks([])
ax.set_yticks([])
ax.set_ylabel('Neuron Network')

ax.set_title('Neural Network Architecture for Pima Diabetes Dataset (2 Output Neurons)

# Add a legend
ax.legend(loc='upper right')

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



In [37]:
```
pima_diabetes_dataframe.head()
```

Out[37]:

	Preg	Plas	Pres	Skin	Test	Mass	Pedi	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [38]:

```
#Ecploratory data analysis
print(pima_diabetes_dataframe.shape)
```

```
(768, 9)
```

In [39]:

```
print(pima_diabetes_dataframe.dtypes)
```

```
Preg       int64
Plas       int64
Pres       int64
Skin       int64
Test       int64
Mass     float64
Pedi     float64
Age        int64
Class      int64
dtype: object
```

In [40]:

```
# datset informatation
pima_diabetes_dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Preg    768 non-null    int64  
 1   Plas    768 non-null    int64  
 2   Pres    768 non-null    int64  
 3   Skin    768 non-null    int64  
 4   Test    768 non-null    int64  
 5   Mass    768 non-null    float64
 6   Pedi    768 non-null    float64
 7   Age     768 non-null    int64  
 8   Class   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Preprocess the dataset

In [41]:

```
# We can clean the dateset and missing values
pima_diabetes_dataframe.isnull().sum()

# Handle missing values if any
#data.dropna(inplace=True)
```

Out[41]:
```
Preg     0
Plas     0
Pres     0
Skin     0
Test     0
Mass     0
Pedi     0
Age      0
Class    0
dtype: int64
```

In [42]:
```
# Find the columns name
print(pima_diabetes_dataframe.columns)
```

```
Index(['Preg', 'Plas', 'Pres', 'Skin', 'Test', 'Mass', 'Pedi', 'Age', 'Class'], dtype
='object')
```

Perform the exploratory data analysis (EDA) in the dataset.

In [43]:
```
# print shape of dataset
print('Number of Instances : ', pima_diabetes_dataframe.shape[0])
print('Number of features : ', pima_diabetes_dataframe.shape[1])
```

```
Number of Instances :  768
Number of features :  9
```

In [44]:
```
# show statistical summary of numerical data
pima_diabetes_dataframe.describe().T
```

Out[44]:

	count	mean	std	min	25%	50%	75%	max
Preg	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Plas	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
Pres	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
Skin	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Test	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
Mass	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
Pedi	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

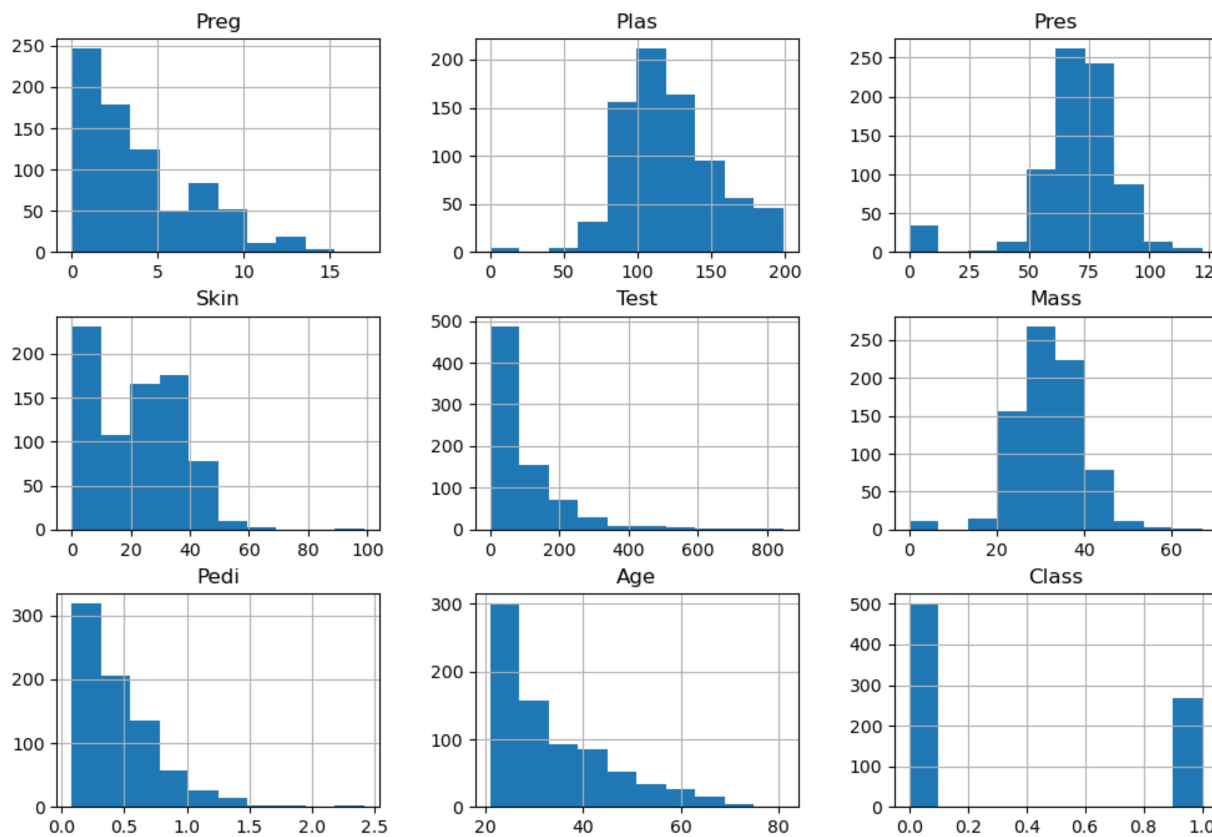
In [45]:
```
pima_diabetes_dataframe.describe()
```

Out[45]:

	Preg	Plas	Pres	Skin	Test	Mass	Pedi	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Univariate Analysis

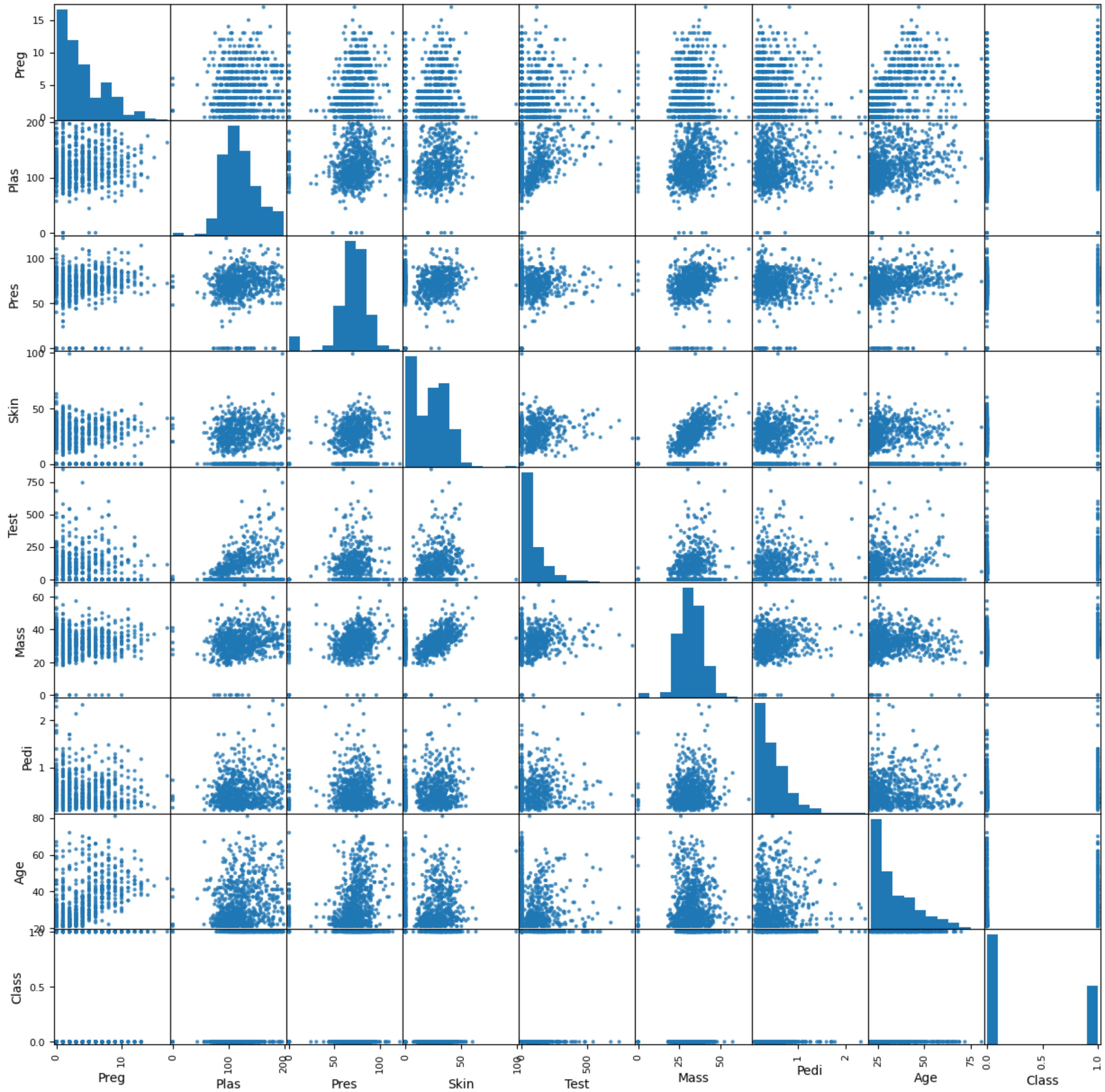
In [46]:
```
# Histogram
pima_diabetes_dataframe.hist(figsize=(12,8))
pyplot.show()
```



Scatter Plots

In [47]:
```
# Scatter plots
scatter_matrix(pima_diabetes_dataframe, alpha=0.8, figsize=(15,15))
```

```
pyplot.show()
```



MLP Model

In [48]:
```
#Designing an MLP model.
def create_model():
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

In [49]:
```
# Create the KerasClassifier

model = KerasClassifier(build_fn=create_model, epochs=10, batch_size=10, verbose=0)
```

Evaluate the model using the 10-fold cross-validation

In [50]:
```
# Evaluate the model using 10-fold cross-validation

scores = cross_val_score(model, X, y, cv=10)
accuracy_training = np.mean(scores)
accuracy_training = np.std(scores)


accuracy_evaluation = np.mean(scores)
accuracy_evaluation = np.std(scores)
```

In [51]:
```
# Evaluate the model using cross-validation predictions
y_pred = cross_val_predict(model, X, y, cv=10)

accuracy_evaluation = accuracy_score(y, y_pred)
accuracy_tranining = accuracy_score(y, y_pred)
```

In [52]:
```
print("Accuracy level from the training process: %.2f%%" % (accuracy_training*100))
print("Accuracy level from the evaluation process: %.2f%%" % (accuracy_evaluation*100)
```

```
Accuracy level from the training process: 5.05%
Accuracy level from the evaluation process: 64.97%
```

In [53]:
```
print("Mean Cross-Validation Accuracy:", accuracy_training)
print("Standard Deviation of Cross-Validation Accuracy:", accuracy_evaluation)
```

```
Mean Cross-Validation Accuracy: 0.05047670667827923
Standard Deviation of Cross-Validation Accuracy: 0.6497395833333334
```

Part 3 ADTA 5550 Deep Learning with Big Data

In [54]:
```
# Next File Part 3
```

In [55]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /opt/conda/lib/python3.7/site-packages
(1.15.5)
Requirement already satisfied: six>=1.10.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (1.16.0)
Requirement already satisfied: h5py<=2.10.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (2.10.0)
Requirement already satisfied: wrapt>=1.11.1 in /opt/conda/lib/python3.7/site-package
s (from tensorflow) (1.14.1)
Requirement already satisfied: absl-py>=0.7.0 in /opt/conda/lib/python3.7/site-packag
es (from tensorflow) (0.8.1)
Requirement already satisfied: grpcio>=1.8.6 in /opt/conda/lib/python3.7/site-package
s (from tensorflow) (1.50.0)
Requirement already satisfied: protobuf>=3.6.1 in /opt/conda/lib/python3.7/site-packa
ges (from tensorflow) (3.20.3)
Requirement already satisfied: tensorflow-estimator==1.15.1 in /opt/conda/lib/python
3.7/site-packages (from tensorflow) (1.15.1)
Requirement already satisfied: termcolor>=1.1.0 in /opt/conda/lib/python3.7/site-pack
ages (from tensorflow) (2.1.0)
Requirement already satisfied: google-pasta>=0.1.6 in /opt/conda/lib/python3.7/site-p
ackages (from tensorflow) (0.2.0)
Requirement already satisfied: keras-applications>=1.0.8 in /opt/conda/lib/python3.7/
site-packages (from tensorflow) (1.0.8)
Requirement already satisfied: tensorboard<1.16.0,>=1.15.0 in /opt/conda/lib/python3.
7/site-packages (from tensorflow) (1.15.0)
Requirement already satisfied: numpy<1.19.0,>=1.16.0 in /opt/conda/lib/python3.7/site
-packages (from tensorflow) (1.18.5)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /opt/conda/lib/python3.
7/site-packages (from tensorflow) (1.1.2)
Requirement already satisfied: gast==0.2.2 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (0.2.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/conda/lib/python3.7/site-pac
kages (from tensorflow) (3.3.0)
Requirement already satisfied: wheel>=0.26 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (0.37.1)
Requirement already satisfied: astor>=0.6.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow) (0.8.1)
Requirement already satisfied: werkzeug>=0.11.15 in /opt/conda/lib/python3.7/site-pac
kages (from tensorboard<1.16.0,>=1.15.0->tensorflow) (2.2.2)
Requirement already satisfied: setuptools>=41.0.0 in /opt/conda/lib/python3.7/site-pa
ckages (from tensorboard<1.16.0,>=1.15.0->tensorflow) (59.8.0)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/python3.7/site-packa
ges (from tensorboard<1.16.0,>=1.15.0->tensorflow) (3.4.1)
Requirement already satisfied: importlib-metadata>=4.4 in /opt/conda/lib/python3.7/si
te-packages (from markdown>=2.6.8->tensorboard<1.16.0,>=1.15.0->tensorflow) (4.11.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/conda/lib/python3.7/site-pac
kages (from werkzeug>=0.11.15->tensorboard<1.16.0,>=1.15.0->tensorflow) (2.1.1)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packages (f
rom importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<1.16.0,>=1.15.0->tensorflo
w) (3.10.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /opt/conda/lib/python3.7/s
ite-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<1.16.0,>=1.1
5.0->tensorflow) (4.4.0)
Note: you may need to restart the kernel to use updated packages.
```

In [56]:

```python
import pandas as pd
import numpy as np
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split   #Train and Test data
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [57]:

```
# Load the dataset
import os
cwd = os.getcwd()
print(cwd)
path = cwd + '/Data/'
print(path)
#df = path
file = path + 'pima_diabetes.csv'
#df = pd.read_csv(file, header=None).values

pima_diabetes_data = pd.read_csv("pima_diabetes.csv", header=None).values

#dataset = pd.read_csv(file, header=None).values

X = pima_diabetes_data[:, :-1]
y = pima_diabetes_data[:, -1]

# Preprocess the data
scaler = StandardScaler()
X = scaler.fit_transform(X)
y = np.reshape(y, (-1, 1))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

```
/home/yogchaudhary9/JPTR_NTBK
/home/yogchaudhary9/JPTR_NTBK/Data/
```

In [58]:

```
pima_diabetes_data
```

Out[58]:

```
array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
```

In [59]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.constraints import max_norm  # Import max_norm constraint

# Define the model architecture
model = Sequential()
model.add(Dense(32, input_dim=X_train.shape[1], activation='relu', kernel_constraint=m
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu', kernel_constraint=max_norm(3)))  # Example max_
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu', kernel_constraint=max_norm(3)))  # Example max_
```

```
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Complie Model

In [60]:
```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [61]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Define the model
model = Sequential()

# Add the input layer
model.add(Dense(32, activation='relu', input_dim=X_train.shape[1]))

# Add the first hidden layer
model.add(Dense(32, activation='relu'))

# Add dropout regularization
model.add(Dropout(0.2))

# Add the second hidden layer
model.add(Dense(32, activation='relu'))

# Add dropout regularization
model.add(Dropout(0.2))

# Add the output layer
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Train Model

In [62]:
```
# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_te
```

```
Train on 614 samples, validate on 154 samples
Epoch 1/100
614/614 [==============================] - 2s 3ms/step - loss: 0.6593 - accuracy: 0.6
221 - val_loss: 0.6228 - val_accuracy: 0.7013
Epoch 2/100
614/614 [==============================] - 0s 250us/step - loss: 0.5827 - accuracy:
0.7150 - val_loss: 0.5671 - val_accuracy: 0.7013
Epoch 3/100
614/614 [==============================] - 0s 247us/step - loss: 0.5232 - accuracy:
0.7508 - val_loss: 0.5335 - val_accuracy: 0.7338
Epoch 4/100
614/614 [==============================] - 0s 239us/step - loss: 0.5048 - accuracy:
0.7573 - val_loss: 0.5181 - val_accuracy: 0.7208
Epoch 5/100
614/614 [==============================] - 0s 240us/step - loss: 0.4798 - accuracy:
0.7557 - val_loss: 0.5176 - val_accuracy: 0.7208
Epoch 6/100
614/614 [==============================] - 0s 265us/step - loss: 0.4631 - accuracy:
0.7671 - val_loss: 0.5150 - val_accuracy: 0.7403
Epoch 7/100
614/614 [==============================] - 0s 281us/step - loss: 0.4677 - accuracy:
0.7687 - val_loss: 0.5186 - val_accuracy: 0.7273
Epoch 8/100
614/614 [==============================] - 0s 252us/step - loss: 0.4652 - accuracy:
0.7834 - val_loss: 0.5206 - val_accuracy: 0.7403
Epoch 9/100
614/614 [==============================] - 0s 240us/step - loss: 0.4625 - accuracy:
0.7801 - val_loss: 0.5225 - val_accuracy: 0.7468
Epoch 10/100
614/614 [==============================] - 0s 238us/step - loss: 0.4466 - accuracy:
0.7785 - val_loss: 0.5233 - val_accuracy: 0.7532
Epoch 11/100
614/614 [==============================] - 0s 240us/step - loss: 0.4431 - accuracy:
0.7850 - val_loss: 0.5260 - val_accuracy: 0.7338
Epoch 12/100
614/614 [==============================] - 0s 227us/step - loss: 0.4420 - accuracy:
0.7769 - val_loss: 0.5269 - val_accuracy: 0.7403
Epoch 13/100
614/614 [==============================] - 0s 223us/step - loss: 0.4466 - accuracy:
0.7720 - val_loss: 0.5260 - val_accuracy: 0.7597
Epoch 14/100
614/614 [==============================] - 0s 222us/step - loss: 0.4404 - accuracy:
0.7834 - val_loss: 0.5310 - val_accuracy: 0.7532
Epoch 15/100
614/614 [==============================] - 0s 226us/step - loss: 0.4393 - accuracy:
0.7899 - val_loss: 0.5298 - val_accuracy: 0.7662
Epoch 16/100
614/614 [==============================] - 0s 216us/step - loss: 0.4331 - accuracy:
0.7850 - val_loss: 0.5332 - val_accuracy: 0.7403
Epoch 17/100
614/614 [==============================] - 0s 231us/step - loss: 0.4288 - accuracy:
0.7980 - val_loss: 0.5359 - val_accuracy: 0.7403
Epoch 18/100
614/614 [==============================] - 0s 231us/step - loss: 0.4220 - accuracy:
0.8029 - val_loss: 0.5365 - val_accuracy: 0.7338
Epoch 19/100
614/614 [==============================] - 0s 232us/step - loss: 0.4339 - accuracy:
0.7752 - val_loss: 0.5291 - val_accuracy: 0.7597
Epoch 20/100
614/614 [==============================] - 0s 232us/step - loss: 0.4274 - accuracy:
```

```
0.7997 - val_loss: 0.5336 - val_accuracy: 0.7532
Epoch 21/100
614/614 [==============================] - 0s 229us/step - loss: 0.4322 - accuracy: 
0.8062 - val_loss: 0.5346 - val_accuracy: 0.7468
Epoch 22/100
614/614 [==============================] - 0s 232us/step - loss: 0.4183 - accuracy: 
0.7899 - val_loss: 0.5338 - val_accuracy: 0.7597
Epoch 23/100
614/614 [==============================] - 0s 246us/step - loss: 0.4217 - accuracy: 
0.7932 - val_loss: 0.5356 - val_accuracy: 0.7468
Epoch 24/100
614/614 [==============================] - 0s 261us/step - loss: 0.4208 - accuracy: 
0.8029 - val_loss: 0.5416 - val_accuracy: 0.7597
Epoch 25/100
614/614 [==============================] - 0s 253us/step - loss: 0.4178 - accuracy: 
0.7980 - val_loss: 0.5432 - val_accuracy: 0.7597
Epoch 26/100
614/614 [==============================] - 0s 267us/step - loss: 0.4120 - accuracy: 
0.8013 - val_loss: 0.5395 - val_accuracy: 0.7597
Epoch 27/100
614/614 [==============================] - 0s 231us/step - loss: 0.4181 - accuracy: 
0.8046 - val_loss: 0.5437 - val_accuracy: 0.7597
Epoch 28/100
614/614 [==============================] - 0s 268us/step - loss: 0.4193 - accuracy: 
0.7915 - val_loss: 0.5490 - val_accuracy: 0.7532
Epoch 29/100
614/614 [==============================] - 0s 282us/step - loss: 0.4171 - accuracy: 
0.8094 - val_loss: 0.5444 - val_accuracy: 0.7403
Epoch 30/100
614/614 [==============================] - 0s 239us/step - loss: 0.4186 - accuracy: 
0.7980 - val_loss: 0.5444 - val_accuracy: 0.7532
Epoch 31/100
614/614 [==============================] - 0s 239us/step - loss: 0.4051 - accuracy: 
0.8013 - val_loss: 0.5456 - val_accuracy: 0.7468
Epoch 32/100
614/614 [==============================] - 0s 244us/step - loss: 0.4005 - accuracy: 
0.8160 - val_loss: 0.5433 - val_accuracy: 0.7597
Epoch 33/100
614/614 [==============================] - 0s 243us/step - loss: 0.4048 - accuracy: 
0.8013 - val_loss: 0.5470 - val_accuracy: 0.7532
Epoch 34/100
614/614 [==============================] - 0s 249us/step - loss: 0.3964 - accuracy: 
0.8192 - val_loss: 0.5398 - val_accuracy: 0.7532
Epoch 35/100
614/614 [==============================] - 0s 244us/step - loss: 0.3967 - accuracy: 
0.7964 - val_loss: 0.5447 - val_accuracy: 0.7532
Epoch 36/100
614/614 [==============================] - 0s 253us/step - loss: 0.3990 - accuracy: 
0.8127 - val_loss: 0.5490 - val_accuracy: 0.7532
Epoch 37/100
614/614 [==============================] - 0s 242us/step - loss: 0.4077 - accuracy: 
0.8176 - val_loss: 0.5501 - val_accuracy: 0.7532
Epoch 38/100
614/614 [==============================] - 0s 247us/step - loss: 0.3982 - accuracy: 
0.8143 - val_loss: 0.5458 - val_accuracy: 0.7403
Epoch 39/100
614/614 [==============================] - 0s 222us/step - loss: 0.3974 - accuracy: 
0.8143 - val_loss: 0.5423 - val_accuracy: 0.7468
Epoch 40/100
614/614 [==============================] - 0s 241us/step - loss: 0.3849 - accuracy: 
```

```
0.8208 - val_loss: 0.5512 - val_accuracy: 0.7403
Epoch 41/100
614/614 [==============================] - 0s 235us/step - loss: 0.3872 - accuracy:
0.8078 - val_loss: 0.5576 - val_accuracy: 0.7597
Epoch 42/100
614/614 [==============================] - 0s 245us/step - loss: 0.3800 - accuracy:
0.8127 - val_loss: 0.5612 - val_accuracy: 0.7403
Epoch 43/100
614/614 [==============================] - 0s 239us/step - loss: 0.3892 - accuracy:
0.8094 - val_loss: 0.5611 - val_accuracy: 0.7468
Epoch 44/100
614/614 [==============================] - 0s 240us/step - loss: 0.3888 - accuracy:
0.8046 - val_loss: 0.5682 - val_accuracy: 0.7468
Epoch 45/100
614/614 [==============================] - 0s 221us/step - loss: 0.3812 - accuracy:
0.8225 - val_loss: 0.5697 - val_accuracy: 0.7468
Epoch 46/100
614/614 [==============================] - 0s 230us/step - loss: 0.3808 - accuracy:
0.8306 - val_loss: 0.5714 - val_accuracy: 0.7662
Epoch 47/100
614/614 [==============================] - 0s 232us/step - loss: 0.3873 - accuracy:
0.8322 - val_loss: 0.5706 - val_accuracy: 0.7468
Epoch 48/100
614/614 [==============================] - 0s 235us/step - loss: 0.3767 - accuracy:
0.8225 - val_loss: 0.5601 - val_accuracy: 0.7403
Epoch 49/100
614/614 [==============================] - 0s 234us/step - loss: 0.3762 - accuracy:
0.8225 - val_loss: 0.5686 - val_accuracy: 0.7532
Epoch 50/100
614/614 [==============================] - 0s 246us/step - loss: 0.3746 - accuracy:
0.8274 - val_loss: 0.5803 - val_accuracy: 0.7662
Epoch 51/100
614/614 [==============================] - 0s 233us/step - loss: 0.3735 - accuracy:
0.8322 - val_loss: 0.5790 - val_accuracy: 0.7468
Epoch 52/100
614/614 [==============================] - 0s 246us/step - loss: 0.3668 - accuracy:
0.8371 - val_loss: 0.5817 - val_accuracy: 0.7468
Epoch 53/100
614/614 [==============================] - 0s 249us/step - loss: 0.3683 - accuracy:
0.8355 - val_loss: 0.5837 - val_accuracy: 0.7532
Epoch 54/100
614/614 [==============================] - 0s 248us/step - loss: 0.3723 - accuracy:
0.8306 - val_loss: 0.5823 - val_accuracy: 0.7468
Epoch 55/100
614/614 [==============================] - 0s 255us/step - loss: 0.3622 - accuracy:
0.8355 - val_loss: 0.5821 - val_accuracy: 0.7468
Epoch 56/100
614/614 [==============================] - 0s 229us/step - loss: 0.3595 - accuracy:
0.8420 - val_loss: 0.5916 - val_accuracy: 0.7468
Epoch 57/100
614/614 [==============================] - 0s 229us/step - loss: 0.3532 - accuracy:
0.8355 - val_loss: 0.5897 - val_accuracy: 0.7403
Epoch 58/100
614/614 [==============================] - 0s 222us/step - loss: 0.3527 - accuracy:
0.8534 - val_loss: 0.5931 - val_accuracy: 0.7532
Epoch 59/100
614/614 [==============================] - 0s 237us/step - loss: 0.3592 - accuracy:
0.8355 - val_loss: 0.5977 - val_accuracy: 0.7403
Epoch 60/100
614/614 [==============================] - 0s 233us/step - loss: 0.3652 - accuracy:
```

```
0.8290 - val_loss: 0.6021 - val_accuracy: 0.7338
Epoch 61/100
614/614 [==============================] - 0s 236us/step - loss: 0.3412 - accuracy: 
0.8404 - val_loss: 0.6071 - val_accuracy: 0.7468
Epoch 62/100
614/614 [==============================] - 0s 235us/step - loss: 0.3544 - accuracy: 
0.8420 - val_loss: 0.6083 - val_accuracy: 0.7468
Epoch 63/100
614/614 [==============================] - 0s 249us/step - loss: 0.3562 - accuracy: 
0.8355 - val_loss: 0.6014 - val_accuracy: 0.7468
Epoch 64/100
614/614 [==============================] - 0s 258us/step - loss: 0.3447 - accuracy: 
0.8485 - val_loss: 0.6061 - val_accuracy: 0.7468
Epoch 65/100
614/614 [==============================] - 0s 252us/step - loss: 0.3408 - accuracy: 
0.8502 - val_loss: 0.6171 - val_accuracy: 0.7532
Epoch 66/100
614/614 [==============================] - 0s 263us/step - loss: 0.3342 - accuracy: 
0.8550 - val_loss: 0.6040 - val_accuracy: 0.7468
Epoch 67/100
614/614 [==============================] - 0s 235us/step - loss: 0.3421 - accuracy: 
0.8485 - val_loss: 0.5977 - val_accuracy: 0.7468
Epoch 68/100
614/614 [==============================] - 0s 256us/step - loss: 0.3470 - accuracy: 
0.8436 - val_loss: 0.6072 - val_accuracy: 0.7468
Epoch 69/100
614/614 [==============================] - 0s 260us/step - loss: 0.3303 - accuracy: 
0.8632 - val_loss: 0.6130 - val_accuracy: 0.7532
Epoch 70/100
614/614 [==============================] - 0s 245us/step - loss: 0.3272 - accuracy: 
0.8599 - val_loss: 0.6248 - val_accuracy: 0.7532
Epoch 71/100
614/614 [==============================] - 0s 255us/step - loss: 0.3411 - accuracy: 
0.8502 - val_loss: 0.6250 - val_accuracy: 0.7403
Epoch 72/100
614/614 [==============================] - 0s 245us/step - loss: 0.3318 - accuracy: 
0.8485 - val_loss: 0.6304 - val_accuracy: 0.7273
Epoch 73/100
614/614 [==============================] - 0s 240us/step - loss: 0.3293 - accuracy: 
0.8681 - val_loss: 0.6236 - val_accuracy: 0.7532
Epoch 74/100
614/614 [==============================] - 0s 236us/step - loss: 0.3280 - accuracy: 
0.8502 - val_loss: 0.6228 - val_accuracy: 0.7403
Epoch 75/100
614/614 [==============================] - 0s 234us/step - loss: 0.3164 - accuracy: 
0.8599 - val_loss: 0.6305 - val_accuracy: 0.7403
Epoch 76/100
614/614 [==============================] - 0s 251us/step - loss: 0.3251 - accuracy: 
0.8567 - val_loss: 0.6255 - val_accuracy: 0.7338
Epoch 77/100
614/614 [==============================] - 0s 271us/step - loss: 0.3232 - accuracy: 
0.8583 - val_loss: 0.6132 - val_accuracy: 0.7273
Epoch 78/100
614/614 [==============================] - 0s 248us/step - loss: 0.3137 - accuracy: 
0.8583 - val_loss: 0.6218 - val_accuracy: 0.7273
Epoch 79/100
614/614 [==============================] - 0s 246us/step - loss: 0.3173 - accuracy: 
0.8616 - val_loss: 0.6292 - val_accuracy: 0.7338
Epoch 80/100
614/614 [==============================] - 0s 247us/step - loss: 0.3133 - accuracy: 
```

```
0.8664 - val_loss: 0.6481 - val_accuracy: 0.7273
Epoch 81/100
614/614 [==============================] - 0s 247us/step - loss: 0.3161 - accuracy:
0.8599 - val_loss: 0.6292 - val_accuracy: 0.7338
Epoch 82/100
614/614 [==============================] - 0s 245us/step - loss: 0.3173 - accuracy:
0.8583 - val_loss: 0.6318 - val_accuracy: 0.7403
Epoch 83/100
614/614 [==============================] - 0s 223us/step - loss: 0.3111 - accuracy:
0.8746 - val_loss: 0.6300 - val_accuracy: 0.7532
Epoch 84/100
614/614 [==============================] - 0s 221us/step - loss: 0.3056 - accuracy:
0.8811 - val_loss: 0.6418 - val_accuracy: 0.7468
Epoch 85/100
614/614 [==============================] - 0s 235us/step - loss: 0.3093 - accuracy:
0.8730 - val_loss: 0.6393 - val_accuracy: 0.7403
Epoch 86/100
614/614 [==============================] - 0s 232us/step - loss: 0.3279 - accuracy:
0.8550 - val_loss: 0.6528 - val_accuracy: 0.7403
Epoch 87/100
614/614 [==============================] - 0s 241us/step - loss: 0.2984 - accuracy:
0.8648 - val_loss: 0.6450 - val_accuracy: 0.7403
Epoch 88/100
614/614 [==============================] - 0s 239us/step - loss: 0.3072 - accuracy:
0.8746 - val_loss: 0.6546 - val_accuracy: 0.7468
Epoch 89/100
614/614 [==============================] - 0s 233us/step - loss: 0.2856 - accuracy:
0.8713 - val_loss: 0.6510 - val_accuracy: 0.7468
Epoch 90/100
614/614 [==============================] - 0s 223us/step - loss: 0.2898 - accuracy:
0.8844 - val_loss: 0.6625 - val_accuracy: 0.7403
Epoch 91/100
614/614 [==============================] - 0s 228us/step - loss: 0.3002 - accuracy:
0.8681 - val_loss: 0.6690 - val_accuracy: 0.7403
Epoch 92/100
614/614 [==============================] - 0s 230us/step - loss: 0.3023 - accuracy:
0.8681 - val_loss: 0.6634 - val_accuracy: 0.7468
Epoch 93/100
614/614 [==============================] - 0s 222us/step - loss: 0.2890 - accuracy:
0.8876 - val_loss: 0.6456 - val_accuracy: 0.7532
Epoch 94/100
614/614 [==============================] - 0s 255us/step - loss: 0.2952 - accuracy:
0.8860 - val_loss: 0.6520 - val_accuracy: 0.7403
Epoch 95/100
614/614 [==============================] - 0s 242us/step - loss: 0.2938 - accuracy:
0.8730 - val_loss: 0.6603 - val_accuracy: 0.7338
Epoch 96/100
614/614 [==============================] - 0s 244us/step - loss: 0.2830 - accuracy:
0.8795 - val_loss: 0.6799 - val_accuracy: 0.7468
Epoch 97/100
614/614 [==============================] - 0s 227us/step - loss: 0.2961 - accuracy:
0.8746 - val_loss: 0.6874 - val_accuracy: 0.7532
Epoch 98/100
614/614 [==============================] - 0s 226us/step - loss: 0.2801 - accuracy:
0.8909 - val_loss: 0.6802 - val_accuracy: 0.7403
Epoch 99/100
614/614 [==============================] - 0s 242us/step - loss: 0.2708 - accuracy:
0.8876 - val_loss: 0.6779 - val_accuracy: 0.7532
Epoch 100/100
```

```
614/614 [==============================] - 0s 256us/step - loss: 0.2655 - accuracy:
0.8844 - val_loss: 0.6972 - val_accuracy: 0.7338
```

Evaluate Model

In [63]:

```
# Evaluate the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
154/154 [==============================] - 0s 140us/step
Accuracy: 73.38
```

In [64]:

```
# cross-validation model
print(" Cross-Validation Accuracy Training: ", _, accuracy )
#print(" Mean Cross-Validation Accuracy Evaluation: ", _, accuracy.mean() )
```

```
 Cross-Validation Accuracy Training:  0.6971944671172601 0.7337662577629089
```

In [65]:

```
# pleases Feedback and Thank you somuch
```