I am doing classification problem on the vitamins part in Australian Food Composition Database - Nutrient information. The input feature variables are component, Limit/s of reporting per 100 g and output feature variable is core component. The target variable would be that a component is core component or not.

For determining the correlation between the input feature variables (component and Limit/s of reporting per 100 g) and the output feature variable (core component), I am using point-biserial correlation as a correlation measure. To cite the standard for the strength of the correlation, I referred to common guidelines for interpreting correlation coefficients. One widely used guideline mentioned in Neter, J., Kutner, M. H., Nachtsheim, C. J., & Wasserman, W. (1996). Applied linear statistical models (4th ed.). Irwin suggests the following thresholds for correlation strength:

- 0 to 0.3: Weak correlation
- 0.3 to 0.7: Moderate correlation
- 0.7 to 1.0: Strong correlation

Data Loading and Column selecting

I read the selected columns, are component, Limit/s of reporting per 100 g and core component on Jupyter Notebook by using pandas library of python.

```
import pandas as pd
usecols=['Component', 'Limit/s of reporting per 100 g', 'Core component']
df = pd.read_excel('Rel_2_Nutrient_details.xlsx', sheet_name='Vitamins',usecols=usecols)
```

Data Pre-processing

Now, I am doing data preprocessing on the dataframe to do one hot encoding of the component column and handling range values in Limits of reporting per 100 g column.

```
df_encoded = pd.get_dummies(df, columns=['Component'])
df_encoded['Limit/s of reporting per 100 g'] = df_encoded['Limit/s of reporting per 100
    lambda x: np.mean([float(val) for val in str(x).split('-')]) if '-' in str(x) else f
```

Preparting the input features X and target variable y,

```
X = df_encoded.drop('Core component', axis=1)
y = df_encoded['Core component']
```

Splitting the dataset to training and testing, Training dataset consists of 80% of the dataset,

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

Filling the missing values in input features of training and testing dataset by using mean of all values. Updating the target variable accordingly. Gave zero value to the core component empty values and 1 to the ✓

```
X_train = X_train.fillna(X_train.mean())
y_train = y_train[X_train.index]
y_train = y_train.fillna(0)
y_train = y_train.replace('ü', 1)
```

```
X_test = X_test.fillna(X_test.mean())
y_test = y_test[X_test.index]
y_test = y_test.fillna(0)
y_test = y_test.replace('ü', 1)
```

Applying ML models,

Initializing and training the logistic regression model,

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
```

Making predictions on the test set using logistic regression,

```
logistic_pred = logistic_model.predict(X_test)
```

Evaluating the logistic regression model,

```
logistic_accuracy = accuracy_score(y_test, logistic_pred)
print("Logistic Regression Accuracy:", logistic_accuracy)
```

```
Logistic Regression Accuracy: 0.625
```

Printing the precision, recall, F1-score and support by using classification report of the sklearn library,

```
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test, logistic_pred)
print("Classification Report:")
print(classification_rep)
```

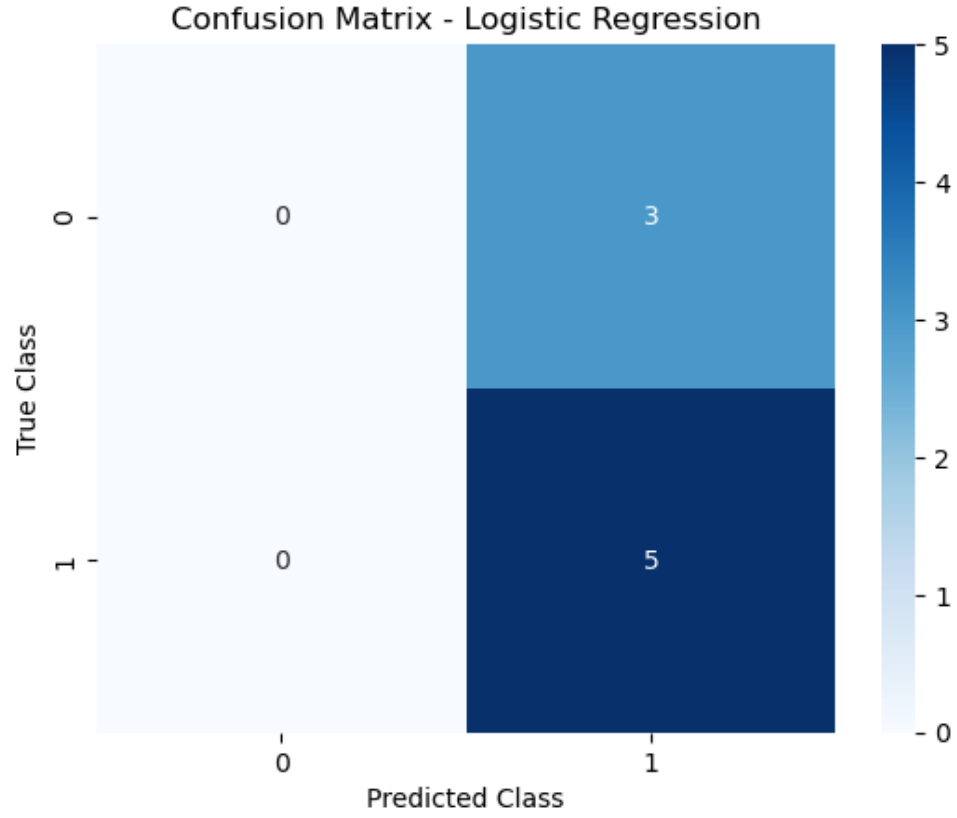
```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.62      1.00      0.77         5

    accuracy                           0.62         8
   macro avg       0.31      0.50      0.38         8
weighted avg       0.39      0.62      0.48         8
```

Computing and plotting the confusion matrix,

```
logistic_confusion = confusion_matrix(y_test, logistic_pred)
sns.heatmap(logistic_confusion, annot=True, cmap='Blues', fmt='g')
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```



Now applying Multi-Layer Perceptron on it,

Defining the network architecture using Tensorflow and keras library of python, hidden layers are 2 with 100 nodes and ReLU activation function. Output layer will have 1 node and sigmoid is applied as activation function as it is binary classification.

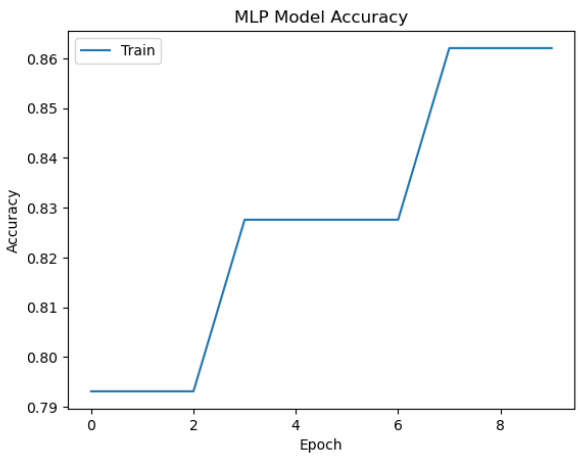
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
MLP = Sequential()
MLP.add(InputLayer(input_shape=(X_train.shape[1],)))
MLP.add(Dense(100, activation='relu'))
MLP.add(Dense(100, activation='relu'))
MLP.add(Dense(1, activation='sigmoid'))
```

Compiling the model by using binary cross entropy as the loss as it is binary classification problem, Adam optimizer is used as it showed best accuracy. Then the model is trained by using batch size of 2 and 10 epochs. The accuracy during training is also plotted below.

```
MLP.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = MLP.fit(X_train, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
1/1 [==============================] - 1s 569ms/step - loss: 0.6428 - accuracy: 0.7931
Epoch 2/10
1/1 [==============================] - 0s 7ms/step - loss: 0.6361 - accuracy: 0.7931
Epoch 3/10
1/1 [==============================] - 0s 11ms/step - loss: 0.6296 - accuracy: 0.7931
Epoch 4/10
1/1 [==============================] - 0s 8ms/step - loss: 0.6233 - accuracy: 0.8276
Epoch 5/10
1/1 [==============================] - 0s 7ms/step - loss: 0.6168 - accuracy: 0.8276
Epoch 6/10
1/1 [==============================] - 0s 11ms/step - loss: 0.6101 - accuracy: 0.8276
Epoch 7/10
1/1 [==============================] - 0s 11ms/step - loss: 0.6035 - accuracy: 0.8276
Epoch 8/10
1/1 [==============================] - 0s 7ms/step - loss: 0.5967 - accuracy: 0.8621
Epoch 9/10
1/1 [==============================] - 0s 10ms/step - loss: 0.5898 - accuracy: 0.8621
Epoch 10/10
1/1 [==============================] - 0s 10ms/step - loss: 0.5827 - accuracy: 0.8621
```



Making predictions on the test set using Multilayer perceptron model and evaluating the MLP model. Printing the precision, recall, F1-score and support by using classification report of the sklearn library,

```
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:")
print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.62      1.00      0.77         5

    accuracy                           0.62         8
   macro avg       0.31      0.50      0.38         8
weighted avg       0.39      0.62      0.48         8
```

Now, Finding and plotting the confusion matrix of the evaluation,

```
mlp_confusion = confusion_matrix(y_test, y_pred)
sns.heatmap(mlp_confusion, annot=True, cmap='Blues', fmt='g')
plt.title("Confusion Matrix - MultiLayer Perceptron")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```