In [1]:
```
import tensorflow as tf
import sys
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from transformers import AdamW, get_linear_schedule_with_warmup
import numpy as np
import time
import datetime
import random
from sklearn.metrics import accuracy_score, precision_score, recall_scc
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import os
```

```
WARNING:tensorflow:From C:\Users\tessa\anaconda3\Lib\site-packages\ker
as\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy
is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_ent
ropy instead.
```

In [2]:
```
import pandas as pd
import numpy as np

# Define the path to your Excel file within the "Data" folder
file_path = "./15-12 Final Training data set.xlsx"


# Read the Excel file into a DataFrame
df = pd.read_excel(file_path)
df
```

Out[2]:

	Sentences	Label
0	A t-test confirmed that no significant differe...	Correct
1	No significant difference in HBP scores betwee...	Correct
2	No significant difference was found in any oth...	Correct
3	it was therefore assumed that this minimal var...	Correct
4	no significant differences were observed in th...	Correct
...	...	...
295	as compared with CHO, 12 while there was no di...	Incorrect
296	As shown in Table 1, there were no differences...	Incorrect
297	Also, there was no effect of time (p = 0.552) ...	Incorrect
298	NEFA concentrations dropped from the baseline ...	Incorrect
299	Insulin concentrations (Figure 3b) did not dif...	Incorrect

300 rows × 2 columns

In [3]:
```
label_mapping = {'Incorrect': 0, 'Correct': 1}
df['Label'] = df['Label'].replace(label_mapping)
df
```

Out[3]:

	Sentences	Label
0	A t-test confirmed that no significant differe...	1
1	No significant difference in HBP scores betwee...	1
2	No significant difference was found in any oth...	1
3	it was therefore assumed that this minimal var...	1
4	no significant differences were observed in th...	1
...	...	...
295	as compared with CHO, 12 while there was no di...	0
296	As shown in Table 1, there were no differences...	0
297	Also, there was no effect of time (p = 0.552) ...	0
298	NEFA concentrations dropped from the baseline ...	0
299	Insulin concentrations (Figure 3b) did not dif...	0

300 rows × 2 columns

In [4]:
```
print('Positive samples: %d of %d (%.2f%%)' % (df.Label.sum(), len(df.L
```

```
Positive samples: 150 of 300 (50.00%)
```

In [77]:
```
# Get the lists of sentences and their labels.
sentences = df.Sentences.values
labels = df.Label.values
```

In [6]:
```
print(labels.dtype)
```

```
int64
```

In [7]:
```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower
model = BertTokenizer.from_pretrained('bert-base-uncased')
```

In [8]:
```
# Print the original sentence.
print(' Original: ', sentences[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize
```

```
 Original:  A t-test confirmed that no significant difference existed
between the two groups ( t(30) = 0.74, P=0 .
Tokenized:  ['a', 't', '-', 'test', 'confirmed', 'that', 'no', 'signif
icant', 'difference', 'existed', 'between', 'the', 'two', 'groups',
'(', 't', '(', '30', ')', '=', '0', '.', '74', ',', 'p', '=', '0',
'.']
Token IDs:  [1037, 1056, 1011, 3231, 4484, 2008, 2053, 3278, 4489, 583
9, 2090, 1996, 2048, 2967, 1006, 1056, 1006, 2382, 1007, 1027, 1014, 1
012, 6356, 1010, 1052, 1027, 1014, 1012]
```

In [9]:
```
# Tokenize all of the sentences and map the tokens to thier word IDs.
input_ids = []

# For every sentence...
for sent in sentences:
    # `encode` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    encoded_sent = tokenizer.encode(
                        sent
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_sent)

# Print sentence 0, now as a list of IDs.
print('Original: ', sentences[0])
print('Token IDs:', input_ids[0])
```

```
Original:  A t-test confirmed that no significant difference existed b
etween the two groups ( t(30) = 0.74, P=0 .
Token IDs: [101, 1037, 1056, 1011, 3231, 4484, 2008, 2053, 3278, 4489,
5839, 2090, 1996, 2048, 2967, 1006, 1056, 1006, 2382, 1007, 1027, 101
4, 1012, 6356, 1010, 1052, 1027, 1014, 1012, 102]
```

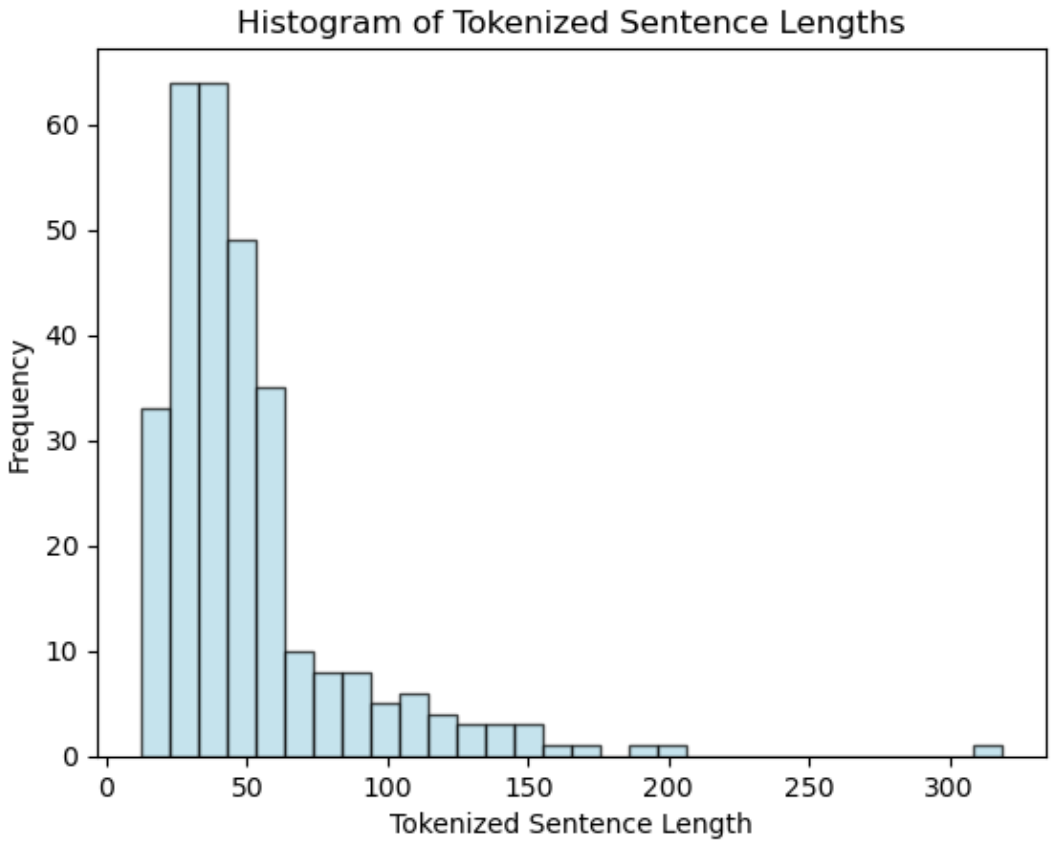
In [10]:
```
print('Max sentence length: ', max([len(sen) for sen in input_ids]))
```

```
Max sentence length:  319
```

In [11]:
```
average_length = sum(len(sen) for sen in input_ids) / len(input_ids)
print('Average sentence length:',average_length)
```

```
Average sentence length: 50.24
```

In [12]:
```
import matplotlib.pyplot as plt

# Calculate the lengths of tokenized sentences
sentence_lengths = [len(sen) for sen in input_ids]

# Create a histogram
plt.hist(sentence_lengths, bins=30, color='lightblue', edgecolor='black
plt.xlabel('Tokenized Sentence Length')
plt.ylabel('Frequency')
plt.title('Histogram of Tokenized Sentence Lengths')
plt.show()
```



In [13]:
```
cutoff_length = 150

# Count the number of sentences below the cutoff
sentences_below_cutoff = sum(1 for length in sentence_lengths if length

print(f'Number of sentences below {cutoff_length} tokens: {sentences_be
```

```
Number of sentences below 150 tokens: 295
```

In [14]:
```
cutoff_length = 150

# Find the sentences above the cutoff
long_sentences = [sentences[i] for i, length in enumerate(sentence_leng

# Print the long sentences with a line space between each
for idx, long_sentence in enumerate(long_sentences):
    print(f'Sentence {idx + 1} (Length: {len(long_sentence)}):\n{long_s
```

```
Sentence 1 (Length: 792):
Selfreport data and preliminary analyses  Oneway ANOVAs revealed no si
gnificant differences between  groups in their ages (M 18–22 years, S
D 1.23 –2.85), F (2, 46)  0.12, p F.05; weight (M 63.2 –68.4 kg, SD
9.60 –14.65), F (2, 46)  0.39, p F.05; height (M 168.30 cm–171.07 c
m, SD 9.07 –9.84), F (2,  46) 0.68, p F.05; training hours/day (M
2.66 –2.90 h, SD  0.57 –2.00), F (2, 46) 0.65, p F.05; training day
s/week (M  3.10 –3.42, SD 0.71 –1.40), F (2, 46) 0.38, p F.05; year
of experience  (M 5.07 –5.86, SD 2.46 –4.05), F (2, 46) 0.30, p F.0
5; or MIQR  scores (M 15.9 –18.34, SD 2.08 –3.25), F (2, 46) 2.65,
p F.05;  indicating no betweengroup differences in terms of age, weigh
t, height,  training hours/day, training days/week, year of experienc
e, and imagery ability.

Sentence 2 (Length: 396):
There was a small albeit significant increase in IL6,  8 and 10 concen
trations  pre to postmatch in both PLB (IL6: 0.83±0.92 Vs 2.91±1.40, I
L8: 2.16±1.22 Vs 3.91±1.61  and IL10: 2.51±2.14 Vs 0.61±0.50 pg.mL1) a
nd MC groups (IL6: 0.53±0.53 Vs 2.24±1.73,  IL8: 1.85±0.96 Vs 3.46±1.1
2 and IL10: 0.48±0.50 Vs 2.54±2.10 pg.mL1), although there  were no si
gnificant differences between groups (P<0.05).

Sentence 3 (Length: 472):
The difference between means  were tested at a significance level of P
<0.05.      13  Results  Match characteristics  There were no signifi
cant differences in absolute distance covered (6334±1924 vs 6596±177
m, P=0.75), relative distance covered (72.6±4.8 vs 79.3±5.5 m.min1, P=
0.009), total collisions  (28±11 vs 29±13, P=0.89), high speed running
(4457±1315 vs 4286±1532 m, P=0.78)  and  playing duration (67:10±19:7
vs 67:10±19:3 min, P=0.99, between the two matches.

Sentence 4 (Length: 403):
Gender and EAMC History Results We did not observe an interaction betw
een EAMC history and gender for [Na+]sw,F(1, 313) = 0.02, p= .88, Na+
swcontent, F(1, 307) = 2.03, p= .16, [K+]sw,F(1, 314) = 0.75, p=. 3 9
,K+ swcontent, F(1, 308) = 2.73, p=. 0 9 , [ C l–]sw,F(1, 265) = 0.60,
p=. 4 4 , C l– swcontent, F(1, 264) = 1.59, p= .21, SR BSA,F(1, 346) =
0.44, p=. 5 1 , o r S R , F(1, 346) = 0.83, p= .36.

Sentence 5 (Length: 345):
We did not observe a main effect for “EAMC history ”for [Na+]sw,F(1, 1
81) = 0.36, p= .55, Na+ swcontent, F(1, 180) = 0.30, p= .59, [K+]sw,F
(1, 182) = 0.33, p= .57, K+ swcontent, F(1, 180) = 0.46, p= .49, [Cl–]
sw,F(1, 182) = 0.01, p= .94, Cl– swcontent, F(1, 181) = 0.36, p= .55,
SR BSA, F(1, 194) = 0.01, p= .96, or SR, F(1, 194) = 0.01, p= .92.
```

In [15]:
```
# We will use some utility function from tensorflow(Tensorflow was my f
from keras.preprocessing.sequence import pad_sequences


MAX_LEN = 150

#Padding the input to the max length that is 150
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="post")
```

In [16]:
```
# Creating the attention masks
attention_masks = []

# For each sentence...
for sent in input_ids:

    # Create the attention mask.
    #   - If a token ID is 0, then it's padding, set the mask to 0.
    #   - If a token ID is > 0, then it's a real token, set the mask to
    att_mask = [int(token_id > 0) for token_id in sent]

    # Store the attention mask for this sentence.
    attention_masks.append(att_mask)
```

In [17]:
```
# # We will call the train_test_split() function from sklearn
# from sklearn.model_selection import train_test_split

# train_inputs, validation_inputs, train_labels, validation_labels = tr
#                                                             random_st
# # Performing same steps on the attention masks
# train_masks, validation_masks, _, _ = train_test_split(attention_mask
#                                           random_state=2018, test_

from sklearn.model_selection import train_test_split

# Split into training and temporary (remaining) data
train_inputs, temp_inputs, train_labels, temp_labels = train_test_split


# Further split the remaining data into validation and test sets
validation_inputs, test_inputs, validation_labels, test_labels = train_


# Repeat the same steps for attention masks
train_masks, temp_masks, _, _ = train_test_split(attention_masks, label
                                                 random_state=2018, test
validation_masks, test_masks, _, _ = train_test_split(temp_masks, temp_
                                                      random_state=2018
```

In [18]:
```
import numpy as np

# Count the labels in each set
train_label_counts = np.bincount(train_labels)
validation_label_counts = np.bincount(validation_labels)
test_label_counts = np.bincount(test_labels)

# Print the counts
print("Train label counts:", train_label_counts)
print("Validation label counts:", validation_label_counts)
print("Test label counts:", test_label_counts)
```

```
Train label counts: [128 112]
Validation label counts: [10 20]
Test label counts: [12 18]
```

In [19]:
```
#Converting the input data to the tensor , which can be feeded to the m
train_inputs = torch.tensor(train_inputs)
validation_inputs = torch.tensor(validation_inputs)

train_labels = torch.tensor(train_labels, dtype=torch.long)
validation_labels = torch.tensor(validation_labels, dtype=torch.long)

train_masks = torch.tensor(train_masks)
validation_masks = torch.tensor(validation_masks)
```

In [20]:
```
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,

#Creating the DataLoader which will help us to load data into the CPU
batch_size = 32

# Create the DataLoader for our training set.
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_


# Create the DataLoader for our validation set.
validation_data = TensorDataset(validation_inputs, validation_masks, va
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_
```

In [21]:

```
from transformers import BertForSequenceClassification, AdamW, BertConf

# Load BertForSequenceClassification, the pretrained BERT model with a
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 2,
    output_attentions = True,
    output_hidden_states = False, )

model
```

```
Some weights of BertForSequenceClassification were not initialized fro
m the model checkpoint at bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

Out[21]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tru
e)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=
True)
              (key): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (value): Linear(in_features=768, out_features=768, bias=
True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=
True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_af
fine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=T
rue)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=T
rue)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affi
ne=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

In [22]:
```
# AdamW is an optimizer which is a Adam Optimzier with weight-decay-fix
from transformers.optimization import AdamW

optimizer = AdamW(model.parameters(),
                  lr = 3e-5,
                  eps = 1e-8,
                )
```

```
C:\Users\tessa\anaconda3\Lib\site-packages\transformers\optimization.p
y:411: FutureWarning: This implementation of AdamW is deprecated and w
ill be removed in a future version. Use the PyTorch implementation tor
ch.optim.AdamW instead, or set `no_deprecation_warning=True` to disabl
e this warning
  warnings.warn(
```

In [23]:
```
from transformers import get_linear_schedule_with_warmup

# Number of training epochs (authors recommend between 2 and 4)
epochs = 5

# Total number of training steps is number of batches * number of epoch
total_steps = len(train_dataloader) * epochs

# Set the number of warm-up steps to 10% of the total steps
warmup_steps = int(0.1 * total_steps)

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = warmup_s
                                            num_training_steps = total_
scheduler
```

Out[23]:
```
<torch.optim.lr_scheduler.LambdaLR at 0x2391264ab90>
```

In [ ]:
```

```

In [24]:
```
import numpy as np

# Function to calculate the accuracy of our predictions vs labels
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)
```

In [25]:

```
#Creating the helper function to have a watch on elapsed time

import time
import datetime

def format_time(elapsed):
    '''
    Takes a time in seconds and returns a string hh:mm:ss
    '''
    # Round to the nearest second.
    elapsed_rounded = int(round((elapsed)))

    # Format as hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

In [26]:

```
import random
import numpy as np
import torch
from transformers import BertForSequenceClassification, AdamW, BertToke
from torch.utils.data import DataLoader, RandomSampler, SequentialSampl

# Set the seed value all over the place to make this reproducible.
seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

# Store the average loss after each epoch so we can plot them.
loss_values = []

# For each epoch...
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epoch
    print('Training...')

    # Measure how long the training epoch takes.
    t0 = time.time()

    # Reset the total loss for this epoch.
    total_loss = 0

    # Put the model into training mode. Don't be mislead--the call to
    # `train` just changes the *mode*, it doesn't *perform* the trainin
    # `dropout` and `batchnorm` layers behave differently during traini
    # vs. test (source: https://stackoverflow.com/questions/51433378/wh
    model.train()

    # For each batch of training data...
    for step, batch in enumerate(train_dataloader):

        # Progress update every 40 batches.
        if step % 40 == 0 and not step == 0:
            # Calculate elapsed time in minutes.
            elapsed = format_time(time.time() - t0)

            # Report progress.
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format

        # Unpack this training batch from our dataloader.
        b_input_ids = batch[0]
        b_input_mask = batch[1]
        b_labels = batch[2]

        # Always clear any previously calculated gradients before perfo
        # backward pass. PyTorch doesn't do this automatically because
        # accumulating the gradients is "convenient while training RNNs
```

```
        # (source: https://stackoverflow.com/questions/48001598/why-do-
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training b
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask,
                        labels=b_labels)

        # The call to `model` always returns a tuple, so we need to pul
        # loss value out of the tuple.
        loss = outputs.loss

        # Accumulate the training loss over all of the batches so that
        # calculate the average loss at the end.
        total_loss += loss.item()

        # Perform a backward pass to calculate the gradients.
        loss.backward()

        # Clip the norm of the gradients to 1.0.
        # This is to help prevent the "exploding gradients" problem.
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        # Update parameters and take a step using the computed gradient
        # The optimizer dictates the "update rule"--how the parameters
        # modified based on their gradients, the learning rate, etc.
        optimizer.step()

        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over the training data.
    avg_train_loss = total_loss / len(train_dataloader)

    # Store the loss value for plotting the learning curve.
    loss_values.append(avg_train_loss)

    print("")
    print("  Average training loss: {0:.2f}".format(avg_train_loss))
    print("  Training epoch took: {:}".format(format_time(time.time() -

    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performa
    # our validation set.

    print("")
    print("Running Validation...")

    t0 = time.time()

    # Put the model in evaluation mode--the dropout layers behave diffe
    # during evaluation.
    model.eval()

    # Tracking variables
    eval_loss, eval_accuracy = 0, 0
    nb_eval_steps, nb_eval_examples = 0, 0
```

```
    # Evaluate data for one epoch
    for batch in validation_dataloader:

        # Add batch to GPU (if available)
        b_input_ids = batch[0]
        b_input_mask = batch[1]
        b_labels = batch[2]

        # Telling the model not to compute or store gradients, saving m
        # speeding up validation
        with torch.no_grad():

            # Forward pass, calculate logit predictions.
            outputs = model(b_input_ids,
                            token_type_ids=None,
                            attention_mask=b_input_mask)

        # Get the "logits" output by the model. The "logits" are the ou
        # values prior to applying an activation function like the soft
        logits = outputs.logits

        # Move


        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Calculate the accuracy for this batch of test sentences.
        tmp_eval_accuracy = flat_accuracy(logits, label_ids)

        # Accumulate the total accuracy.
        eval_accuracy += tmp_eval_accuracy

        # Track the number of batches
        nb_eval_steps += 1

    # Report the final accuracy for this validation run.
    print("  Accuracy: {0:.2f}".format(eval_accuracy/nb_eval_steps))
    print("  Validation took: {:}".format(format_time(time.time() - t0)

print("")
print("Training complete!")
```

```
======== Epoch 1 / 5 ========
Training...

  Average training loss: 0.70
  Training epoch took: 0:02:13

Running Validation...
  Accuracy: 0.43
  Validation took: 0:00:06

======== Epoch 2 / 5 ========
Training...

  Average training loss: 0.61
  Training epoch took: 0:02:16

Running Validation...
  Accuracy: 0.73
  Validation took: 0:00:06

======== Epoch 3 / 5 ========
Training...

  Average training loss: 0.46
  Training epoch took: 0:02:16

Running Validation...
  Accuracy: 0.83
  Validation took: 0:00:06

======== Epoch 4 / 5 ========
Training...

  Average training loss: 0.33
  Training epoch took: 0:02:15

Running Validation...
  Accuracy: 0.87
  Validation took: 0:00:06

======== Epoch 5 / 5 ========
Training...

  Average training loss: 0.25
  Training epoch took: 0:02:13

Running Validation...
  Accuracy: 0.87
  Validation took: 0:00:06

Training complete!
```

In [27]:

```
print(loss_values) #Having a view of stored loss values in the list
```

```
[0.7013650983572006, 0.6121422350406647, 0.4569171257317066, 0.3274149
8574614525, 0.2469380870461464]
```

In [37]:
```
# Report the number of sentences.
print('Number of test sentences: {:,}\n'.format(test_inputs.shape[0]))

# Convert to tensors.
prediction_inputs = torch.tensor(test_inputs)
prediction_masks = torch.tensor(test_masks)
prediction_labels = torch.tensor(test_labels)

# Set the batch size.
batch_size = 32

# Create the DataLoader.
prediction_data = TensorDataset(prediction_inputs, prediction_masks, pr
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_
```

```
Number of test sentences: 30
```

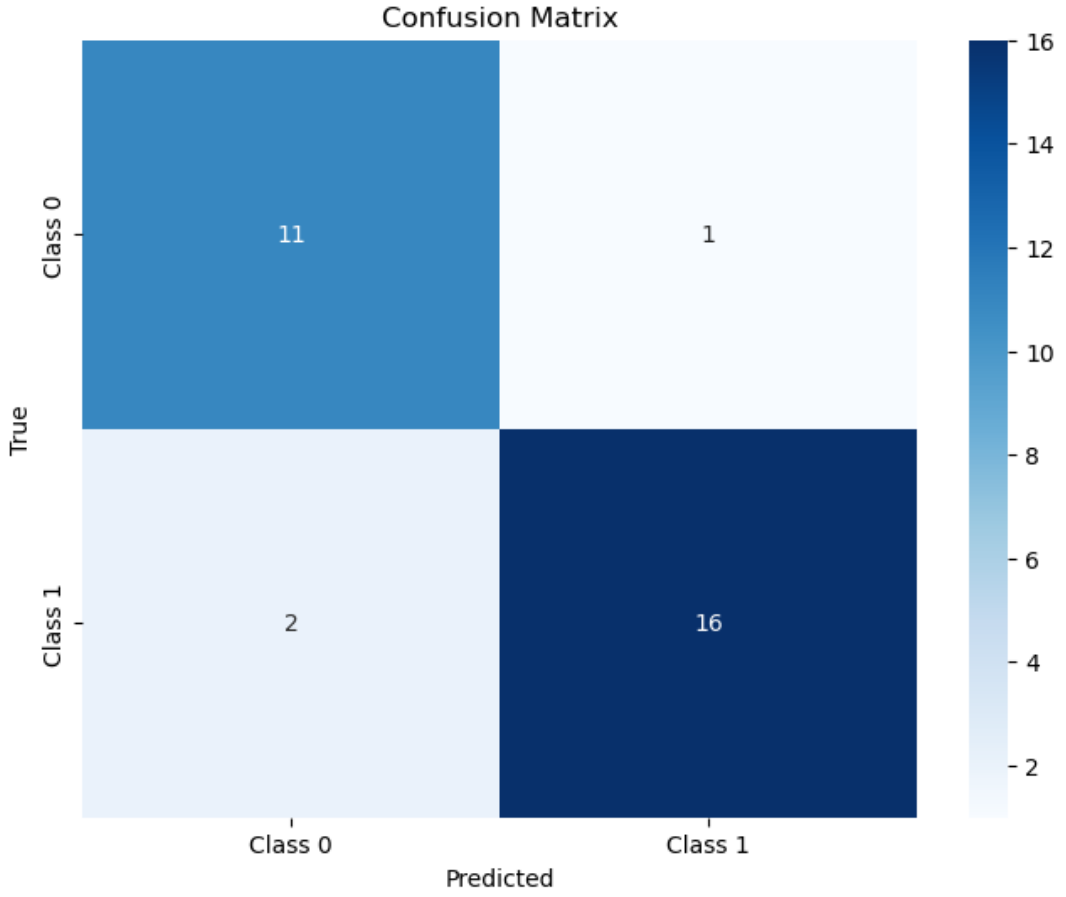
In [54]:
```
# Report the number of sentences.
print('Predicting labels for {:,} test sentences...'.format(len(test_ir


# Put model in evaluation mode
model.eval()

# Tracking variables
predictions, true_labels = [], []

# Create the DataLoader.
test_data = TensorDataset(torch.tensor(test_inputs), torch.tensor(test_
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_siz

# Predict
for batch in test_dataloader:
    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask, b_labels = batch

    # Telling the model not to compute or store gradients, saving memor
    # speeding up prediction
    with torch.no_grad():
        # Forward pass, calculate logit predictions
        outputs = model(b_input_ids, token_type_ids=None, attention_mas

    logits = outputs.logits.detach().numpy()

    # Move labels to CPU
    label_ids = b_labels.numpy()

    # Store predictions and true labels
    predictions.append(logits)
    true_labels.append(label_ids)
```

```
Predicting labels for 30 test sentences...
```

In [69]:

```
# Flatten the nested lists of predictions and true labels
flat_predictions = np.concatenate(predictions, axis=0)
flat_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
predicted_labels = np.argmax(flat_predictions, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(flat_true_labels, predicted_labels)

# Calculate metrics
accuracy = accuracy_score(flat_true_labels, predicted_labels)
precision = precision_score(flat_true_labels, predicted_labels, average
recall = recall_score(flat_true_labels, predicted_labels, average='weig
f1 = f1_score(flat_true_labels, predicted_labels, average='weighted')

# Print metrics
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')

# Plot the confusion matrix with seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.9000
Precision: 0.9032
Recall: 0.9000
F1 Score: 0.9006
```


Confusion Matrix
11
1
2
16
Class 0
Class 1
True
Class 0
Class 1
Predicted
16
14
12
10
8
6
4
2

In [76]:

```
print("Length of flat_true_labels:", len(flat_true_labels))
print("Length of predicted_labels:", len(predicted_labels))
print("Length of sentences:", len(sentences))
print("Misclassified Indices:", misclassified_indices)

# Print the contents of the arrays
print("Contents of flat_true_labels:", flat_true_labels)
print("Contents of predicted_labels:", predicted_labels)
print("Contents of sentences:", sentences)
```

```
Length of flat_true_labels: 30
Length of predicted_labels: 30
Length of sentences: 18
Misclassified Indices: [15 25 28]
Contents of flat_true_labels: [0 0 1 0 1 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1
1 1 1 0 1 1 1 0 1 1]
Contents of predicted_labels: [0 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 1
1 1 1 0 1 0 1 0 0 1]
Contents of sentences: ['Given the working-set MCVs in both conditions
did not differ greatly, it is possible the current subjects were not s
trong enough to exhibit a PAP response during the down sets or a longe
r rest interval was needed after the working sets to dissipate fatigue
before performing the down set.', 'The results for the plankhold follo
wed a similar pattern but were not statistically significant.', 'Rows
that share the same subscript letter, do not differ significantly.',
'Aerobic fitness groups did not differ significantly by  ethnicity, ra
ce, sex, estimated IQ, education, pastyear cannabis use,  pastyear alc
ohol use, recent nicotine exposure (cotinine level), or amount of sede
ntary behavior (see Table 1).', 'Repeatedmeasures ANOVA results indica
ted a significant effect between each 5 s interval (p < 0.001), but no
differences  were observed between trials ( p > 0.05).', 'LT assessedb
y VO 2(Figure 2 (E)) did also not differ significantly between BC supp
lementation and placebo ( p>.', 'Although we did note a trend toward i
ncreased D’ and time to exhaustion in a cohort of our subjects, but th
ese were weak trends that did not reach statistical significance and t
he effect sizes were small to medium, but variable.', 'In this respec
t, recent reviews have suggested that the use of compression garments
after running  has little or no effect on muscle damage and inflammato
ry markers (Brown et al., 2017; Engel et  al., 2016).', 'Similarly, th
ere was a decrease in PLB (2.16±0.34  m.s1) and MC (2.17±0.33 m.s1) dr
op jump performance from 24h prematch to 48h postmatch  (PLB: 2.05±0.4
0 m.s1 and MC: 2.06±0.41 m.s1) although this was not statistically sig
nificant  (P=0.228 and P =0.893, respectively).', 'Nonetheless, they w
ere significantly higher in DELAY  (+ 3.6 ± 3.5 mU · L1; p = 0.003) an
d CHO (+ 4.7 ± 3.0; mU · L1 p < 0.001) as compared with PLA , whereas
there was no difference between DELAY and CHO (p > 0.999) at 60min.',
'5.2 ± 0.6 mmol · L1) as compared to PLA (4.2 ± 0.6  and 3.3 ± 0.6 mmo
l ·  L1) at 60min and post TT time points (p < 0.05) with no differenc
e between CHO and DELAY (p >  0.999) conditions.', 'AUC for glucose wa
s significantly higher in CHO as compared with PLA (p =  0.006), where
as there was no difference between CHO and DELAY (p = 0.189) or PLA an
d  DELAY (p = 0.228 ).', 'AUC for lactate was significantly higher in
CHO as compared with PLA (p = 0.029 ) and  DELAY (p = 0.019), whereas
there was no difference between PLA and DELAY (p = 0.974).', 'However,
in their study, the preservation of muscular force could  not be attri
buted to changes in the central factor because voluntary activation (V
A) d oes not differ  between the maltodextrin and placebo mouth rinse
groups.', 'There were no differences in T lim between conditions (BR =
22.8 ± 8.1 min;  Placebo = 20.7 ± 7.9 min) ( P = 0.184), despite incre
ases in plasma', 'Dietary NO 3 supplementation had no effect on exerci
se tolerance or thermoregulation in hot, dry  conditions, despite redu
ctions in resting MAP and increases in plasma', 'There were no differe
nces ( t(10)= 1.4, P = 0.184) in T lim between the BR and PLA  conditi
ons, despite seven out of the eleven participants extending their perf
ormance  after BR supplementation (BR = 22.8 ± 8 .1min; Placebo = 20.7
± 7.9min).', 'There were no differences between  PLA and BR for H prod
(t(10) = 0.103, P = 0.920), H dry (t(10) = 1.913, P = 0.085), E req (t
(10)  = 0.789, P = 0.448), heat storage ( t(10) = 0.941, P = 0.369), E
max (t(10) = 1.919, P =  0.084) or W ( t(10) = 0.101, P = 0.337)']
```

In [75]:
```
print("Misclassified Sentences:")
for idx in misclassified_indices:
    if 0 <= idx < len(sentences):
        print(f'True Label: {flat_true_labels[idx]}, Predicted Label: {
    else:
        print(f'Index {idx} is out of range for sentences.')
```

```
Misclassified Sentences:
True Label: 0, Predicted Label: 1, Sentence: Dietary NO 3 supplementat
ion had no effect on exercise tolerance or thermoregulation in hot, dr
y  conditions, despite reductions in resting MAP and increases in plas
ma
Index 25 is out of range for sentences.
Index 28 is out of range for sentences.
```

In [ ]:
```
```

In [78]:
```
# Ensure that the lengths match
min_length = min(len(flat_true_labels), len(predicted_labels), len(sent
flat_true_labels = flat_true_labels[:min_length]
predicted_labels = predicted_labels[:min_length]
sentences = sentences[:min_length]

# Create a DataFrame for misclassified sentences
misclassified_df1 = pd.DataFrame({
    'True Label': flat_true_labels,
    'Predicted Label': predicted_labels,
    'Sentence': sentences
})

# Filter the DataFrame to include only misclassified sentences
misclassified_df1 = misclassified_df1[misclassified_df1['True Label'] !

# Set pandas options for better display
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)

# Assuming you have the misclassified DataFrame named misclassified_df

# Change labels for better readability
misclassified_df1['True Label'] = np.where(misclassified_df1['True Labe
misclassified_df1['Predicted Label'] = np.where(misclassified_df1['Pred

# Save the misclassified DataFrame to an Excel file
misclassified_df1.to_excel("misclassified_sentences1.xlsx", index=False

# Now, read the Excel file into a new DataFrame
misclassified_df_from_excel = pd.read_excel("misclassified_sentences1.x

misclassified_df_from_excel
```

Out[78]:

	True Label	Predicted Label	Sentence
0	Incorrect	Correct	In comparison to a previous study examining the RHE, similar peak EMG values were seen in the ES because this was also above 100% of MVIC; in contrast, however, the HE also elicited $100% MVIC in the same study and with no significant difference present between the two (21).
1	Correct	Incorrect	Given the working-set MCVs in both conditions did not differ greatly, it is possible the current subjects were not strong enough to exhibit a PAP response during the down sets or a longer rest interval was needed after the working sets to dissipate fatigue before performing the down set.
2	Correct	Incorrect	This hypothesis, however, was not supported when investigating ingame performance.

In [44]:
```
import os
# Specify the directory where you want to save the model
output_dir = './trained_model1/'

# Create the directory if it doesn't exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Save model to directory
model.save_pretrained(output_dir)

# Save tokenizer to directory
tokenizer.save_pretrained(output_dir)

# Save configuration to directory
model.config.save_pretrained(output_dir)
```

In [45]:
```
from transformers import BertForSequenceClassification, BertTokenizer

# Load the saved model and tokenizer
model = BertForSequenceClassification.from_pretrained(output_dir)
tokenizer = BertTokenizer.from_pretrained(output_dir)
```

In [46]:

```
from transformers import BertTokenizer, BertForSequenceClassification
import torch

# Load the saved model and tokenizer
output_dir = './trained_model1/'
model = BertForSequenceClassification.from_pretrained(output_dir)
tokenizer = BertTokenizer.from_pretrained(output_dir)

# Input sentences for classification
sentencess = [
  "Given the working-set MCVs in both conditions did not differ greatly
  "The results for the plankhold followed a similar pattern but were no
  "Rows that share the same subscript letter, do not differ significant
  "Aerobic fitness groups did not differ significantly by  ethnicity, r
  "Repeatedmeasures ANOVA results indicated a significant effect betwee
  "LT assessedby VO 2(Figure 2 (E)) did also not differ significantly b
  "Although we did note a trend toward increased D' and time to exhaust
  "In this respect, recent reviews have suggested that the use of compr
  "Similarly, there was a decrease in PLB (2.16±0.34  m.s1) and MC (2.1
  "Nonetheless, they were significantly higher in DELAY  (+ 3.6 ± 3.5 m
  "5.2 ± 0.6 mmol · L1) as compared to PLA (4.2 ± 0.6  and 3.3 ± 0.6 mm
  "AUC for glucose was significantly higher in CHO as compared with PLA
  "AUC for lactate was significantly higher in CHO as compared with PLA
  "However, in their study, the preservation of muscular force could  n
  "There were no differences in T lim between conditions (BR = 22.8 ± 8
  "Dietary NO 3 supplementation had no effect on exercise tolerance or
  "There were no differences ( t(10)= 1.4, P = 0.184) in T lim between
  "There were no differences between  PLA and BR for H prod (t(10) = 0.
]

# Tokenize input sentences
tokenized_input = tokenizer(sentences, padding=True, truncation=True, r

# Ensure the model is in evaluation mode
model.eval()

# Make predictions
with torch.no_grad():
    # Forward pass
    outputs = model(**tokenized_input)

# Get the predicted probabilities
probs = torch.nn.functional.softmax(outputs.logits, dim=-1)

# Get the predicted class (0 or 1 in binary classification)
predicted_class = torch.argmax(probs, dim=1).tolist()

# Display results
for sentence, label in zip(sentences, predicted_class):
    print(f"Sentence: {sentence}")
    print(f"Predicted Label: {label}")
    print()


#1 is correct 0 is incorrect
```

Sentence: Given the working-set MCVs in both conditions did not differ greatly, it is possible the current subjects were not strong enough to exhibit a PAP response during the down sets or a longer rest interval was needed after the working sets to dissipate fatigue before performi ng the down set.
Predicted Label: 1

Sentence: The results for the plankhold followed a similar pattern but were not statistically significant.
Predicted Label: 1

Sentence: Rows that share the same subscript letter, do not differ sig nificantly.
Predicted Label: 0

Sentence: Aerobic fitness groups did not differ significantly by ethn icity, race, sex, estimated IQ, education, pastyear cannabis use, pas tyear alcohol use, recent nicotine exposure (cotinine level), or amoun t of sedentary behavior (see Table 1).
Predicted Label: 0

Sentence: Repeatedmeasures ANOVA results indicated a significant effec t between each 5 s interval (p < 0.001), but no differences were obse rved between trials (p > 0.05).
Predicted Label: 1

Sentence: LT assessedby VO 2(Figure 2 (E)) did also not differ signifi cantly between BC supplementation and placebo (p>.
Predicted Label: 0

Sentence: Although we did note a trend toward increased D’ and time to exhaustion in a cohort of our subjects, but these were weak trends tha t did not reach statistical significance and the effect sizes were sma ll to medium, but variable.
Predicted Label: 0

Sentence: In this respect, recent reviews have suggested that the use of compression garments after running has little or no effect on musc le damage and inflammatory markers (Brown et al., 2017; Engel et al., 2016).
Predicted Label: 0

Sentence: Similarly, there was a decrease in PLB (2.16±0.34 m.s1) and MC (2.17±0.33 m.s1) drop jump performance from 24h prematch to 48h pos tmatch (PLB: 2.05±0.40 m.s1 and MC: 2.06±0.41 m.s1) although this was not statistically significant (P=0.228 and P =0.893, respectively).
Predicted Label: 1

Sentence: Nonetheless, they were significantly higher in DELAY (+ 3.6 ± 3.5 mU · L1; p = 0.003) and CHO (+ 4.7 ± 3.0; mU · L1 p < 0.001) as compared with PLA , whereas there was no difference between DELAY and CHO (p > 0.999) at 60min.
Predicted Label: 1

Sentence: 5.2 ± 0.6 mmol · L1) as compared to PLA (4.2 ± 0.6 and 3.3 ± 0.6 mmol · L1) at 60min and post TT time points (p < 0.05) with no difference between CHO and DELAY (p > 0.999) conditions.
Predicted Label: 0

Sentence: AUC for glucose was significantly higher in CHO as compared with PLA (p = 0.006), whereas there was no difference between CHO and

```
DELAY (p = 0.189) or PLA and  DELAY (p = 0.228 ).
Predicted Label: 0

Sentence: AUC for lactate was significantly higher in CHO as compared
with PLA (p = 0.029 ) and  DELAY (p = 0.019), whereas there was no dif
ference between PLA and DELAY (p = 0.974).
Predicted Label: 0

Sentence: However, in their study, the preservation of muscular force
could  not be attributed to changes in the central factor because volu
ntary activation (VA) d oes not differ  between the maltodextrin and p
lacebo mouth rinse groups.
Predicted Label: 0

Sentence: There were no differences in T lim between conditions (BR =
22.8 ± 8.1 min;  Placebo = 20.7 ± 7.9 min) ( P = 0.184), despite incre
ases in plasma
Predicted Label: 0

Sentence: Dietary NO 3 supplementation had no effect on exercise toler
ance or thermoregulation in hot, dry  conditions, despite reductions i
n resting MAP and increases in plasma
Predicted Label: 0

Sentence: There were no differences ( t(10)= 1.4, P = 0.184) in T lim
between the BR and PLA  conditions, despite seven out of the eleven pa
rticipants extending their performance  after BR supplementation (BR =
22.8 ± 8 .1min; Placebo = 20.7 ± 7.9min).
Predicted Label: 0

Sentence: There were no differences between  PLA and BR for H prod (t
(10) = 0.103, P = 0.920), H dry (t(10) = 1.913, P = 0.085), E req (t(1
0)  = 0.789, P = 0.448), heat storage ( t(10) = 0.941, P = 0.369), E m
ax (t(10) = 1.919, P =  0.084) or W ( t(10) = 0.101, P = 0.337)
Predicted Label: 1
```

In [ ]:

In [ ]:

In [ ]: