Name: Bhavesh Waghela

Student Number: N01639685

In [1]:
```
import pandas as pd
import tensorflow as tf
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Normalization
```

In [2]:
```
dataset = pd.read_csv('WEC_Sydney_100.csv')
dataset
```

Out[2]:

	X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5	...	Power93	Powe
0	1.0	1.0	1.00	51.00	1.00	101.00	1.00	151.0	398.0	0.0	...	74018.52	71727
1	198.0	0.0	197.18	80.53	193.59	150.00	77.58	198.0	598.0	0.0	...	63702.46	67776
2	198.0	0.0	197.07	76.64	192.74	155.74	84.67	198.0	798.0	0.0	...	55788.34	59593
3	1.0	1.0	1.00	51.00	1.00	101.00	1.00	151.0	398.0	0.0	...	66961.48	65716
4	198.0	0.0	197.46	75.07	197.18	149.14	149.00	198.0	598.0	0.0	...	51814.27	59556
...	...	...	...	...	...	...	...	...	...	...	...	...	
2313	198.0	0.0	197.60	72.52	193.04	150.00	84.79	198.0	398.0	0.0	...	52910.73	58836
2314	198.0	0.0	96.79	74.77	192.57	150.00	86.00	183.0	398.0	0.0	...	61488.75	61218
2315	598.0	0.0	597.72	67.53	593.10	146.65	549.00	198.0	198.0	200.0	...	52752.08	54870
2316	398.0	0.0	397.18	80.53	393.59	150.00	277.58	198.0	1398.0	0.0	...	58755.78	64815
2317	48.0	1.0	98.00	50.00	148.00	100.00	198.00	150.0	398.0	0.0	...	68332.23	64935

2318 rows × 302 columns

In [3]:
```
dataset.tail()
```

Out[3]:

	X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5	...	Power93	Powe
2313	198.0	0.0	197.60	72.52	193.04	150.00	84.79	198.0	398.0	0.0	...	52910.73	58836
2314	198.0	0.0	96.79	74.77	192.57	150.00	86.00	183.0	398.0	0.0	...	61488.75	61218
2315	598.0	0.0	597.72	67.53	593.10	146.65	549.00	198.0	198.0	200.0	...	52752.08	54870
2316	398.0	0.0	397.18	80.53	393.59	150.00	277.58	198.0	1398.0	0.0	...	58755.78	64815
2317	48.0	1.0	98.00	50.00	148.00	100.00	198.00	150.0	398.0	0.0	...	68332.23	64935

5 rows × 302 columns

In [4]:
```
dataset.isna().sum()
```

Out[4]:
```
X1             0
Y1             0
X2             0
Y2             0
X3             0
              ..
Power98        0
Power99        0
Power100       0
qW             0
Total_Power    0
Length: 302, dtype: int64
```

In [5]:
```
num_wecs = 100

# Generate column names dynamically for X and Y coordinates
column_names_x = [f'X{i}' for i in range(1, num_wecs + 1)]
column_names_y = [f'Y{i}' for i in range(1, num_wecs + 1)]


# Combine X and Y columns
column_names = column_names_x + column_names_y + ['Total_Power']

# Extract features (X) and target variable (y)
dataset = dataset[column_names]

dataset
```

Out[5]:

X2	X3	X4	X5	X6	X7	X8	X9	X10	...	Y92	Y93	Y94
.00	1.00	1.00	398.0	397.46	397.18	349.00	598.0	597.46	...	1398.0	1200.0	1275.07
.18	193.59	77.58	598.0	597.18	593.59	477.58	798.0	797.18	...	1398.0	1200.0	1280.53
.07	192.74	84.67	798.0	797.07	792.74	684.67	998.0	997.07	...	1398.0	1200.0	1276.64
.00	1.00	1.00	398.0	397.07	392.74	349.00	798.0	797.07	...	1398.0	1200.0	1276.56
.46	197.18	149.00	598.0	597.46	597.18	549.00	998.0	997.46	...	1198.0	1200.0	1275.07
...	...	...	...	...	...	...	...	...	...	...	...	...
.60	193.04	84.79	398.0	397.60	393.04	284.79	598.0	597.60	...	1198.0	1200.0	1272.52
.79	192.57	86.00	398.0	346.79	392.57	255.00	618.0	596.79	...	1398.0	1200.0	1274.77
.72	593.10	549.00	198.0	197.72	193.10	149.00	398.0	397.72	...	1398.0	1200.0	1267.53
.18	393.59	277.58	1398.0	1397.18	1393.59	1277.58	398.0	397.18	...	1398.0	1200.0	1280.53
.00	148.00	198.00	398.0	397.60	393.36	349.00	598.0	597.60	...	1398.0	1200.0	1271.65

columns

In [6]:
```
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

In [ ]:
```

```

In [7]:
```
train_dataset.describe().transpose()
```

Out[7]:

	count	mean	std	min	25%	50%	
X1	1854.0	1.750434e+02	172.849509	0.00	48.00	198.000	198
X2	1854.0	2.039537e+02	172.815809	0.00	100.00	197.070	201
X3	1854.0	2.251456e+02	182.153753	0.00	192.57	193.700	250
X4	1854.0	2.133537e+02	226.177273	0.00	78.60	95.560	279
X5	1854.0	5.465644e+02	204.426957	0.00	398.00	401.000	636
...	...	...	...	...	...	...	
Y97	1854.0	1.230488e+03	98.620077	76.79	1200.00	1200.000	1200
Y98	1854.0	1.290511e+03	78.889248	125.54	1274.75	1277.610	1278
Y99	1854.0	1.355698e+03	72.891211	167.60	1350.00	1350.000	1354
Y100	1854.0	1.397207e+03	73.578608	132.73	1398.00	1398.000	1398
Total_Power	1854.0	7.169366e+06	105215.266481	6506096.43	7126146.81	7189008.015	7240335

201 rows × 8 columns

In [8]:
```
train_features = train_dataset.copy()
test_features = test_dataset.copy()

train_labels = train_features.pop('Total_Power')
test_labels = test_features.pop('Total_Power')
```

In [9]:
```
# Normalize the target variable (Total_Power)
y_mean = train_labels.mean()
y_std = train_labels.std()
y_normalized = (train_labels - y_mean) / y_std
```

In [10]:
```
train_dataset.describe().transpose()[['mean', 'std']]
```

Out[10]:

	mean	std
X1	1.750434e+02	172.849509
X2	2.039537e+02	172.815809
X3	2.251456e+02	182.153753
X4	2.133537e+02	226.177273
X5	5.465644e+02	204.426957
...	...	...
Y97	1.230488e+03	98.620077
Y98	1.290511e+03	78.889248
Y99	1.355698e+03	72.891211
Y100	1.397207e+03	73.578608
Total_Power	7.169366e+06	105215.266481

201 rows × 2 columns

In [11]:
```
normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(np.array(train_features))
print(normalizer.mean.numpy())
```

```
[[ 175.04344   203.9537    225.14561   213.35367   546.56445   561.9887
   581.1926    553.16113   897.1456    922.8767    701.11926   659.61597
   967.60675   994.49365  1016.9813    961.2696    911.49506   938.77264
   960.73376   921.7028    599.69684   632.1351    647.31824   611.62897
   730.62476   753.0667    775.7173    734.52435   967.73267   988.5247
   767.6233    717.3717    647.7961    673.37006   698.508     657.12036
   881.70355   897.23676   919.811     868.45447   737.5966    763.18115
   781.93744   744.1683    636.8003    658.0222    689.8543    647.0149
   870.89667   896.3337    686.3995    651.9861    675.8017    706.81464
   731.26166   682.41956   731.9375    752.5213    768.9279    731.369
   693.88666   719.0513    737.12854   703.5057    752.60693   784.20123
   812.9041    765.94714   848.8072    862.7247    654.3628    612.0771
   737.8328    763.97076   790.03827   751.8416    868.13776   895.0089
   915.2414    884.2763    431.25665   444.3438    479.34988   448.9854
   523.76196   549.7741    577.7505    551.2831    822.17535   833.3479
   639.746     586.47614   916.4948    940.59314   963.83887   908.5899
  1239.4163   1272.7504   1292.1105   1249.8145      7.334337   64.057884
   125.10537   163.4653     17.450077   66.79009   135.52457   175.01083
    24.437126   78.885185  163.16762   199.74446    55.448467  117.47841
   179.581     217.1055    119.20764   179.54913   229.80626   266.8557
   212.71902   266.61328   320.94394   359.13855   236.6895    303.40564
   357.20267   398.64014   268.8677    333.64572   410.3715    458.90176
   376.26154   447.91525   493.4879    539.7501    427.43588   480.61737
   538.7827    579.9563    486.12625   548.37616   601.10913   655.2039
   571.0805    622.09265   680.42474   720.7297    602.0719    660.2369
   742.0554    785.1771    674.89294   742.1525    800.213     844.95624
   746.6084    803.2405    864.0488    899.46295   814.08795   871.52356
   931.16736   977.02716   877.06805   938.16846   996.03125  1047.1691
   925.6685    985.2493   1076.99     1114.2832   1000.09296  1061.6635
  1118.337    1158.903    1067.0514   1119.9867   1174.5525   1207.0701
  1166.3704   1219.457    1279.8569   1310.6727   1209.7631   1263.9667
  1312.3579   1352.0931   1204.3351   1264.4742   1349.285    1389.9479
  1227.3536   1290.0052   1348.4249   1391.9336   1230.4884   1290.5114
  1355.6976   1397.2069  ]]
```

In [12]:
```
first = np.array(train_features[:1])

with np.printoptions(precision=2, suppress=True):
    print('First example:', first)
    print()
    print('Normalized:', normalizer(first).numpy())
```

```
First example: [[ 198.    197.07  193.7    84.98  398.    397.07  393.7   28
4.98  848.
   898.    948.    998.   1198.   1197.07 1193.7  1084.98 1398.   1397.07
  1393.7  1284.98  798.    797.07  793.7   684.98 1198.   1197.07 1193.7
  1084.98 1398.   1397.07 1393.7  1284.98  398.    397.07  393.7   284.98
   998.    997.07  993.7   884.98   48.     98.    148.    198.    798.
   797.07  793.7   684.98 1198.   1197.07 1193.7  1084.98  398.    397.07
   393.7   284.98 1198.   1197.07 1193.7  1084.98  198.    197.07  193.7
    84.98  398.    397.07  393.7   284.98  598.    597.07  593.7   484.98
  1198.   1197.07 1193.7  1084.98 1398.   1397.07 1393.7  1284.98    1.
     1.      1.     50.    398.    397.07  393.7   284.98  798.    797.07
   793.7   684.98 1198.   1197.07 1193.7  1084.98 1398.   1397.07 1393.7
  1284.98    0.     77.44  150.    198.       0.     77.44  150.    198.
     1.     50.    100.    150.       0.     77.44  150.    198.       0.
    77.44  150.    198.    200.    277.44  350.    398.    200.    277.44
   350.    398.    200.    277.44  350.    398.    400.    477.44  550.
   598.    400.    477.44  550.    598.    601.    650.    700.    750.
   600.    677.44  750.    798.    600.    677.44  750.    798.    800.
   877.44  950.    998.    800.    877.44  950.    998.   1000.   1077.44
  1150.   1198.   1000.   1077.44 1150.   1198.   1000.   1077.44 1150.
  1198.   1000.   1077.44 1150.   1198.   1000.   1077.44 1150.   1198.
  1201.   1270.   1340.   1398.   1200.   1277.44 1350.   1398.   1200.
  1277.44 1350.   1398.   1200.   1277.44 1350.   1398.   1200.   1277.44
  1350.   1398.  ]]

Normalized: [[ 0.13 -0.04 -0.17 -0.57 -0.73 -0.74 -0.75 -0.79 -0.18 -0.08  0.
67  1.12
   0.58  0.57  0.54  0.42  1.03  0.97  0.92  0.75  0.39  0.34  0.31  0.17
   1.22  1.15  1.08  0.87  1.07  0.97  1.3   1.26 -0.62 -0.7  -0.79 -0.97
   0.35  0.3   0.21  0.04 -1.3  -1.35 -1.35 -1.3   0.38  0.33  0.24  0.09
   0.76  0.67  1.06  1.   -0.66 -0.77 -0.85 -1.03  1.16  1.07  0.98  0.75
  -1.07 -1.19 -1.29 -1.6  -0.81 -0.89 -0.95 -1.06 -0.57 -0.58 -0.13 -0.3
   1.22  1.2   1.15  0.98  1.05  0.98  0.94  0.78 -0.91 -0.95 -1.07 -0.94
  -0.44 -0.52 -0.58 -0.74 -0.08 -0.11  0.47  0.34  0.83  0.86  0.88  0.77
   0.64  0.6   0.55  0.18 -0.14  0.17  0.26  0.32 -0.2   0.13  0.18  0.25
  -0.27 -0.32 -0.76 -0.61 -0.43 -0.39 -0.31 -0.2  -0.85 -0.79 -0.67 -0.55
  -0.09  0.1   0.29  0.41 -0.37 -0.3  -0.08 -0.01 -0.66 -0.57 -0.58 -0.61
   0.21  0.26  0.57  0.57 -0.25 -0.04  0.12  0.19  0.91  0.91  1.02  0.96
   0.24  0.52  0.65  0.68 -0.02  0.18  0.08  0.12  1.08  1.23  1.44  1.48
   0.45  0.71  0.79  0.84  1.68  2.11  2.12  2.27  1.1   1.29  1.38  1.28
   0.64  0.77  0.64  0.72 -0.    0.18  0.34  0.38 -0.45 -0.32 -0.19 -0.07
   0.27  0.41  0.52  0.72 -0.08  0.13  0.36  0.4  -0.05  0.14  0.01  0.1
  -0.27 -0.15  0.02  0.08 -0.31 -0.17 -0.08  0.01]]
```

In [29]:
```
def build_and_compile_model(norm):
    model = keras.Sequential([
        norm,
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)
    ])

    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.01))
    return model
```

In [30]:
```
dnn_model = build_and_compile_model(normalizer)
dnn_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizati  (None, 200)               401
 on)

 dense_6 (Dense)             (None, 64)                12864

 dense_7 (Dense)             (None, 64)                4160

 dense_8 (Dense)             (None, 32)                2080

 dense_9 (Dense)             (None, 1)                 33

=================================================================
Total params: 19538 (76.32 KB)
Trainable params: 19137 (74.75 KB)
Non-trainable params: 401 (1.57 KB)
_________________________________________________________________
```

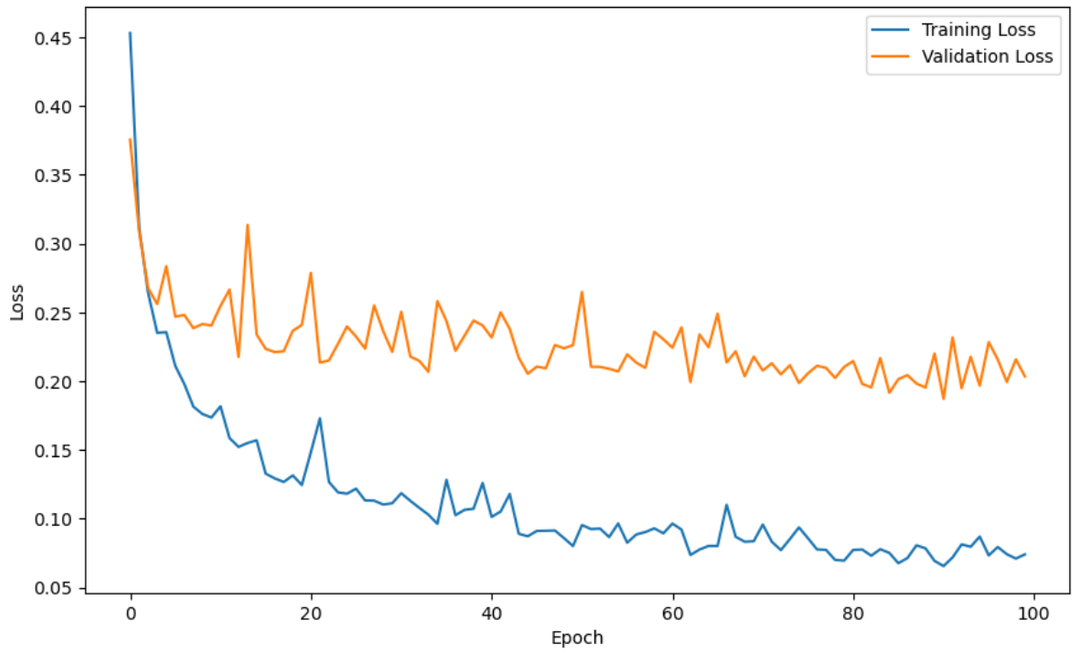
In [31]:
```
import matplotlib.pyplot as plt
from IPython.display import clear_output

class LivePlotCallback(tf.keras.callbacks.Callback):
    def __init__(self):
        self.epochs = []
        self.losses = []
        self.val_losses = []

    def on_epoch_end(self, epoch, logs=None):
        self.epochs.append(epoch)
        self.losses.append(logs['loss'])
        self.val_losses.append(logs['val_loss'])
        clear_output(wait=True)
        plt.figure(figsize=(10, 6))
        plt.plot(self.epochs, self.losses, label='Training Loss')
        plt.plot(self.epochs, self.val_losses, label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

live_plot = LivePlotCallback()

history = dnn_model.fit(
    train_features,
    y_normalized,
    validation_split=0.2,
    verbose=0,
    epochs=100,
    callbacks=[live_plot])
```



In [32]:
```
loss = dnn_model.evaluate(train_features, y_normalized, verbose=0)
print(f"Mean Absolute Error on Training Set: {loss}")
```

```
Mean Absolute Error on Training Set: 0.09486076235771179
```

In [33]:
```
predictions_normalized = dnn_model.predict(test_features)
```

```
15/15 [==============================] - 0s 2ms/step
```

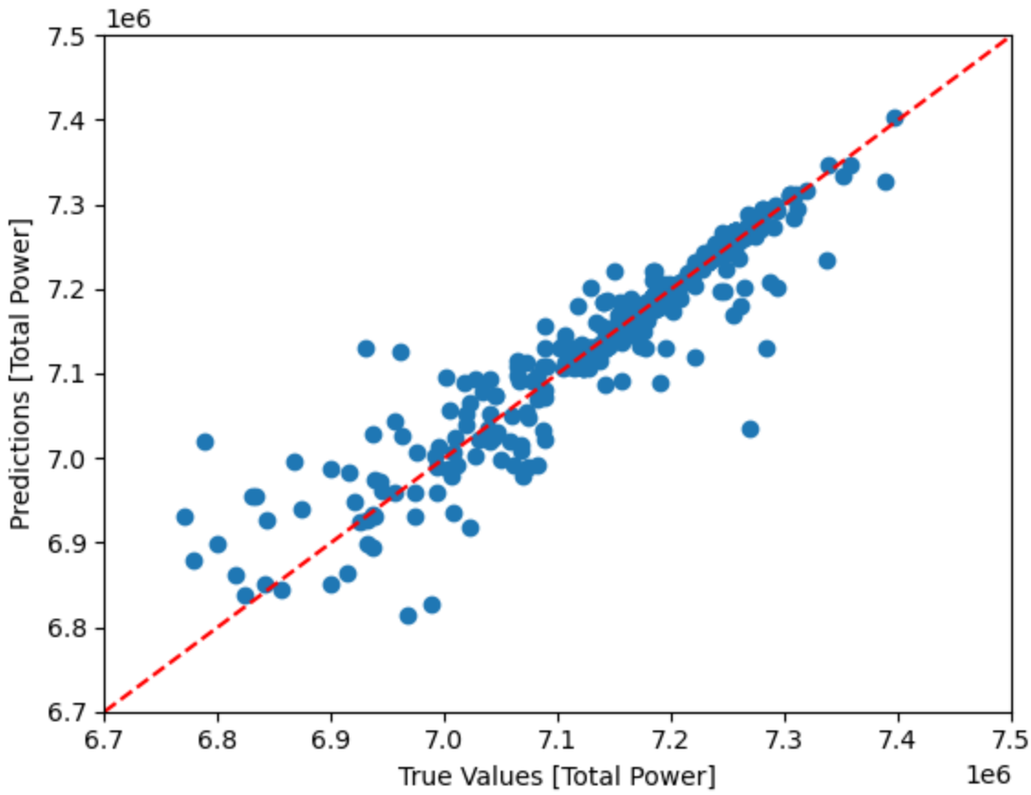
In [34]:
```
predictions = predictions_normalized * y_std + y_mean
```

In [35]:
```
# Plot the scatter plot
plt.scatter(test_labels, predictions)
plt.xlabel('True Values [Total Power]')
plt.ylabel('Predictions [Total Power]')

# Set the limits for better visualization
lims = [6700000, 7500000]
plt.xlim(lims)
plt.ylim(lims)

# Plot a diagonal line for reference
_ = plt.plot(lims, lims, color='red', linestyle='--')

plt.show()
```



In [37]:
```
def build_and_compile_model_2(norm):
    model = keras.Sequential([
        norm,
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
         layers.Dense(32, activation='relu'),
        layers.Dense(1)
    ])

    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.SGD(0.001))
    return model
```

In [38]:
```
dnn_model_sgd = build_and_compile_model(normalizer)
dnn_model_sgd.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizati  (None, 200)               401
 on)

 dense_10 (Dense)            (None, 64)                12864

 dense_11 (Dense)            (None, 64)                4160

 dense_12 (Dense)            (None, 32)                2080

 dense_13 (Dense)            (None, 1)                 33

=================================================================
Total params: 19538 (76.32 KB)
Trainable params: 19137 (74.75 KB)
Non-trainable params: 401 (1.57 KB)
_________________________________________________________________
```

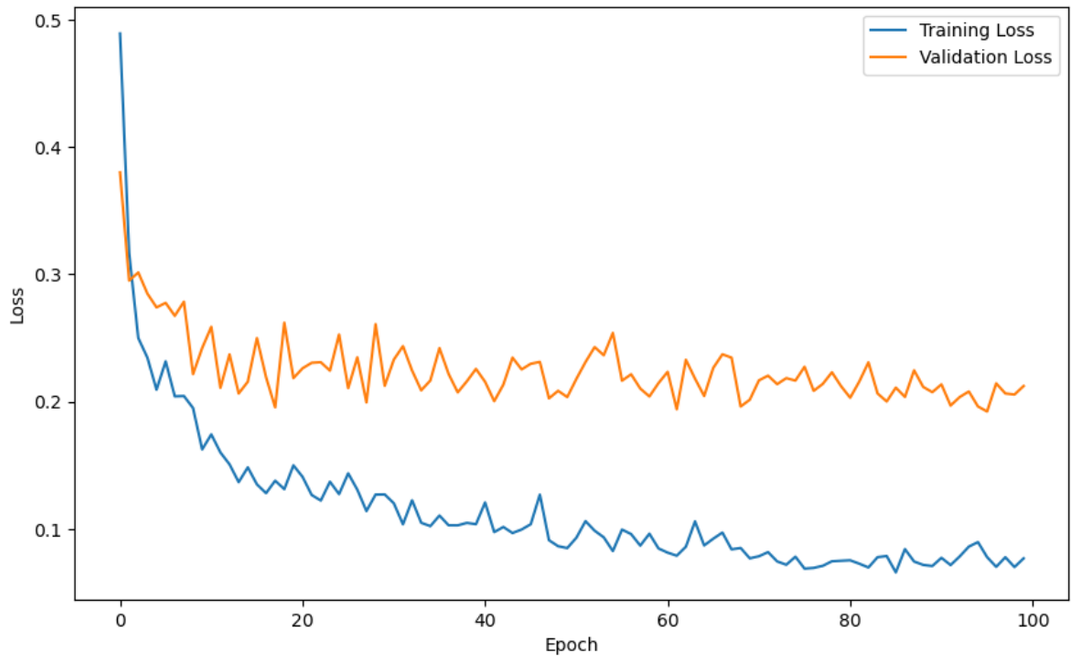
In [39]:
```
import matplotlib.pyplot as plt
from IPython.display import clear_output

class LivePlotCallback(tf.keras.callbacks.Callback):
    def __init__(self):
        self.epochs = []
        self.losses = []
        self.val_losses = []

    def on_epoch_end(self, epoch, logs=None):
        self.epochs.append(epoch)
        self.losses.append(logs['loss'])
        self.val_losses.append(logs['val_loss'])
        clear_output(wait=True)
        plt.figure(figsize=(10, 6))
        plt.plot(self.epochs, self.losses, label='Training Loss')
        plt.plot(self.epochs, self.val_losses, label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

live_plot = LivePlotCallback()

history = dnn_model_sgd.fit(
    train_features,
    y_normalized,
    validation_split=0.2,
    verbose=0,
    epochs=100,
    callbacks=[live_plot])
```



In [40]:
```
loss = dnn_model_sgd.evaluate(train_features, y_normalized, verbose=0)
print(f"Mean Absolute Error on Training Set: {loss}")
```

```
Mean Absolute Error on Training Set: 0.10546934604644775
```

In [41]:
```
predictions_normalized = dnn_model_sgd.predict(test_features)

predictions = predictions_normalized * y_std + y_mean
```

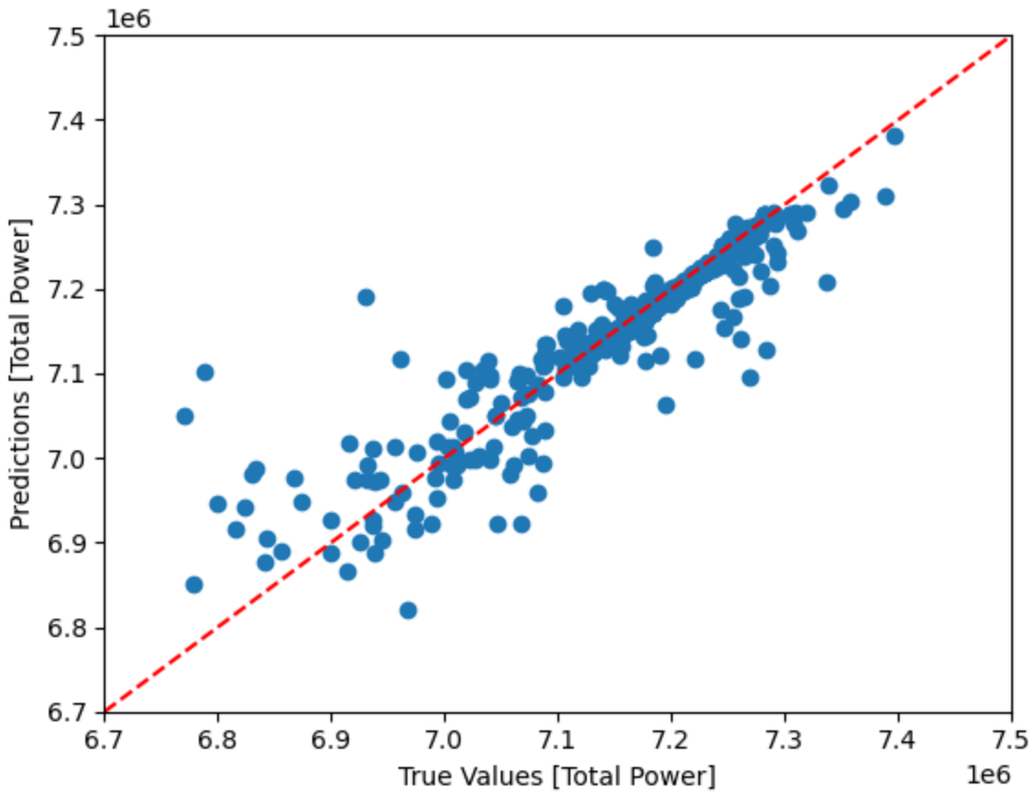
```
15/15 [==============================] - 0s 2ms/step
```

In [42]:
```
# Plot the scatter plot
plt.scatter(test_labels, predictions)
plt.xlabel('True Values [Total Power]')
plt.ylabel('Predictions [Total Power]')

# Set the limits for better visualization
lims = [6700000, 7500000]
plt.xlim(lims)
plt.ylim(lims)

# Plot a diagonal line for reference
_ = plt.plot(lims, lims, color='red', linestyle='--')

plt.show()
```



Observation:

Model 1: Optimizer = Adam Learning Rate = 0.01 Mean Absolute Error = 0.095

Model 2: Optimizer = SGD Learning Rate = 0.001 Mean Absolute Error = 0.105

Model 1 performed better with the following hyperparameters and model 2 was also good in predecting the Total Power attribute.