Loading Necessary Modules

In [1]:

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

from sklearn import model_selection

from keras.wrappers.scikit_learn import KerasRegressor
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import BatchNormalization
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, Learning

import tensorflow as tf

import helper
```

```
Using TensorFlow backend.
```

Styling Tables

In [2]:

```
%%HTML
<style type='text/css'>
table.dataframe th, table.dataframe td{
    border: 3px solid purple !important;
    color: solid black !important;
}
</style>
```

Loading Data

In [3]:

```
# Loading dataset
filename = "Clean_Akosombo_data.csv"
akosombo = helper.load_csv_data(filename)
```

```
Successfully loaded!
```

Splitting Data

In [4]:

```
# Splitting dataset
target_variable = "generation"
X, y, X_train, X_test, X_val, y_train, y_test, y_val = helper.split_data(akosombo, targ
```

```
Data is splitted into X, y, X_train, X_test, X_val, y_train, y_test, y_val.

Shape Info of Features Training Set:
Number of datapoints (rows): 7500
Number of features (columns): 2

Shape Info of Features Test Set:
Number of datapoints (rows): 2501
Number of features (columns): 2

Shape Info of Features Validation Set:
Number of datapoints (rows): 2501
Number of features (columns): 2
```

Scaling Data

In [5]:

```
# Data Scaling
X_train, X_test, X_val = helper.scale(X_train, X_test, X_val, scale_validation=True)
```

Model Creation

In [9]:

```
# Creating Sequential Model
neural_network_model = Sequential()

# Input Layer
neural_network_model.add(Dense(20, input_dim=X_train.shape[1], kernel_initializer='norm

# Hidden Layers
neural_network_model.add(Dense(40, kernel_initializer='normal', activation='relu'))
neural_network_model.add(Dense(40, kernel_initializer='normal', activation='relu'))
neural_network_model.add(Dropout(0.01))

# Output Layer
neural_network_model.add(Dense(1, kernel_initializer='normal', activation='linear'))

# Compiling the network
neural_network_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mea
neural_network_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 20)                60
_________________________________________________________________
dense_6 (Dense)              (None, 40)                840
_________________________________________________________________
dense_7 (Dense)              (None, 40)                1640
_________________________________________________________________
dropout_2 (Dropout)          (None, 40)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 41
=================================================================
Total params: 2,581
Trainable params: 2,581
Non-trainable params: 0
_________________________________________________________________
```

Callbacks

In [10]:

```
checkpoint_name = 'Weights-{epoch:03d}--{val_loss:.5f}.h5'


checkpoint = ModelCheckpoint(checkpoint_name,
                             monitor="val_loss",
                             save_best_only=True,
                             verbose=1,
                            mode='auto')


earlystop = EarlyStopping(monitor='val_loss',
                          min_delta=0,
                          patience=3,
                          verbose=1,
                          mode='auto',
                          restore_best_weights=True)


reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=3,
                              verbose=1,
                              mode='auto',
                              min_delta=0.00001)

# Putting call backs into a callback list
callbacks = [earlystop, checkpoint, reduce_lr]
```

Training Model

In [11]:

```
epochs = 150
batch_size = 5

history = neural_network_model.fit(
    X_train, y_train,
    batch_size=batch_size,
    validation_data=(X_test, y_test), # Overides validation_split argument.
#     validation_split=0.25,
    epochs=epochs,
    verbose=2,
    callbacks=callbacks,
)
```

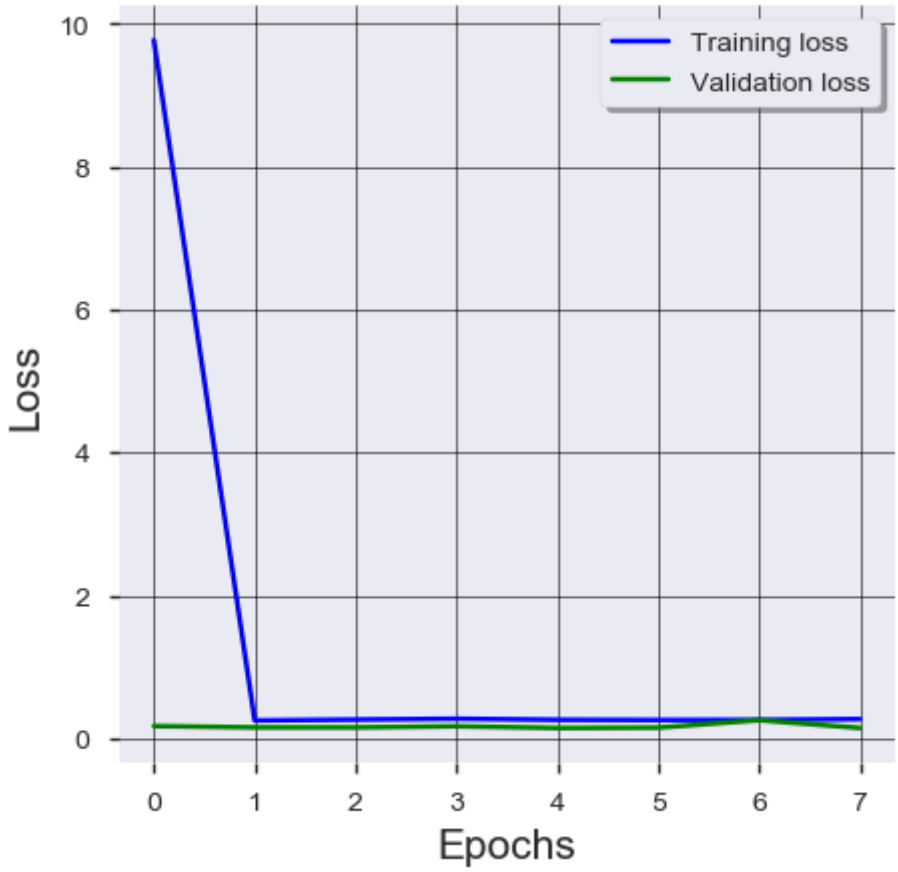
```
Train on 7500 samples, validate on 2501 samples
Epoch 1/150
 - 2s - loss: 9.7737 - mean_squared_error: 9.7738 - mean_absolute_error: 1.2
441 - val_loss: 0.1738 - val_mean_squared_error: 0.1738 - val_mean_absolute_
error: 0.3295

Epoch 00001: val_loss improved from inf to 0.17377, saving model to Weights-
001--0.17377.h5
Epoch 2/150
 - 1s - loss: 0.2522 - mean_squared_error: 0.2522 - mean_absolute_error: 0.3
883 - val_loss: 0.1546 - val_mean_squared_error: 0.1546 - val_mean_absolute_
error: 0.2972

Epoch 00002: val_loss improved from 0.17377 to 0.15459, saving model to Weig
hts-002--0.15459.h5
Epoch 3/150
 - 2s - loss: 0.2640 - mean_squared_error: 0.2640 - mean_absolute_error: 0.3
929 - val_loss: 0.1541 - val_mean_squared_error: 0.1541 - val_mean_absolute_
error: 0.2962

Epoch 00003: val_loss improved from 0.15459 to 0.15405, saving model to Weig
hts-003--0.15405.h5
Epoch 4/150
 - 1s - loss: 0.2759 - mean_squared_error: 0.2759 - mean_absolute_error: 0.3
996 - val_loss: 0.1708 - val_mean_squared_error: 0.1708 - val_mean_absolute_
error: 0.3255

Epoch 00004: val_loss did not improve from 0.15405
Epoch 5/150
 - 1s - loss: 0.2613 - mean_squared_error: 0.2613 - mean_absolute_error: 0.3
945 - val_loss: 0.1462 - val_mean_squared_error: 0.1462 - val_mean_absolute_
error: 0.2946

Epoch 00005: val_loss improved from 0.15405 to 0.14619, saving model to Weig
hts-005--0.14619.h5
Epoch 6/150
 - 1s - loss: 0.2584 - mean_squared_error: 0.2584 - mean_absolute_error: 0.3
859 - val_loss: 0.1515 - val_mean_squared_error: 0.1515 - val_mean_absolute_
error: 0.2979

Epoch 00006: val_loss did not improve from 0.14619
Epoch 7/150
 - 1s - loss: 0.2597 - mean_squared_error: 0.2597 - mean_absolute_error: 0.3
920 - val_loss: 0.2564 - val_mean_squared_error: 0.2564 - val_mean_absolute_
error: 0.4028
```

```
Epoch 00007: val_loss did not improve from 0.14619
Epoch 8/150
 - 1s - loss: 0.2711 - mean_squared_error: 0.2711 - mean_absolute_error: 0.3
987 - val_loss: 0.1469 - val_mean_squared_error: 0.1469 - val_mean_absolute_
error: 0.2909
Restoring model weights from the end of the best epoch

Epoch 00008: val_loss did not improve from 0.14619

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.00020000000949949
026.
Epoch 00008: early stopping
```

Visualizing Losses

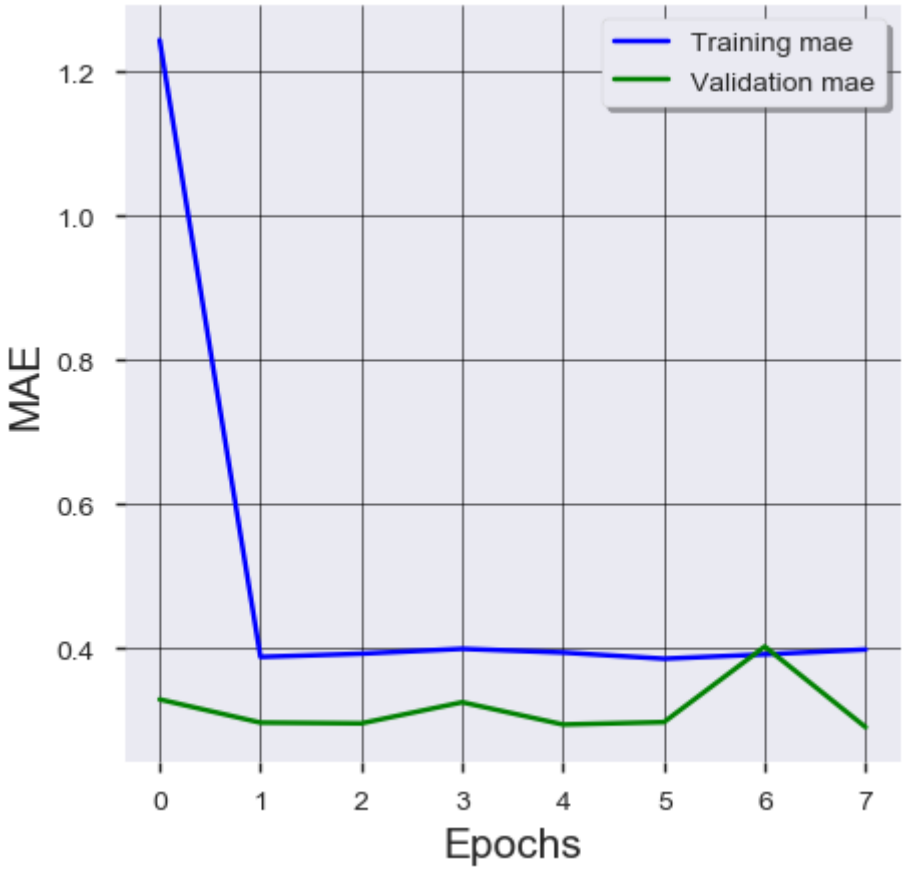
In [12]:

```
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
epochs = history.epoch

fig = plt.figure(figsize=(5,5), dpi=100)

plt.plot(epochs, training_loss, label='Training loss', color='blue')
plt.plot(epochs, validation_loss, label='Validation loss', color='green')

plt.xlabel('Epochs', fontsize=15)
plt.ylabel('Loss', fontsize=15)

plt.grid(b=True, which="both", axis="both", color="black", linewidth=0.4)
plt.legend(loc='best', fontsize='medium', numpoints=1, frameon=True, shadow=True, fancy

plt.savefig('Neural_Network_Loss.png', dpi=300, transparent=True)

plt.show()
```



In [13]:

```
training_accuracy = history.history['mean_absolute_error']
validation_loss = history.history['val_mean_absolute_error']
epochs = history.epoch


fig = plt.figure(figsize=(5,5), dpi=100)

plt.plot(epochs, training_accuracy, label='Training mae', color='blue')
plt.plot(epochs, validation_loss, label='Validation mae', color='green')

plt.xlabel('Epochs', fontsize=15)
plt.ylabel('MAE', fontsize=15)

plt.grid(b=True, which="both", axis="both", color="black", linewidth=0.4)
plt.legend(loc='best', fontsize='medium', numpoints=1, frameon=True, shadow=True, fancy

plt.savefig('Neural_Network_mean_absolute_error.png', dpi=300, transparent=True)

plt.show()
```



Model History Data

In [39]:

```
history_data = pd.DataFrame(history.history)
history_data['epochs'] = history.epoch
history_data.to_csv("neural_network_history.csv", index=True)
history_data
```

Out[39]:

	val_loss	val_mean_squared_error	val_mean_absolute_error	loss	mean_squared_erro
0	0.173769	0.173769	0.329457	9.773736	9.77375
1	0.154588	0.154588	0.297226	0.252188	0.25218
2	0.154051	0.154051	0.296220	0.263980	0.26398
3	0.170786	0.170786	0.325498	0.275920	0.27592
4	0.146188	0.146188	0.294601	0.261313	0.26131
5	0.151516	0.151516	0.297905	0.258412	0.25841
6	0.256357	0.256357	0.402799	0.259739	0.25973
7	0.146919	0.146919	0.290868	0.271056	0.27105

In [40]:

```
model_parameters = pd.DataFrame(history.params)
model_parameters.to_csv("neural_network_parameters.csv", index=True)
model_parameters
```

Out[40]:

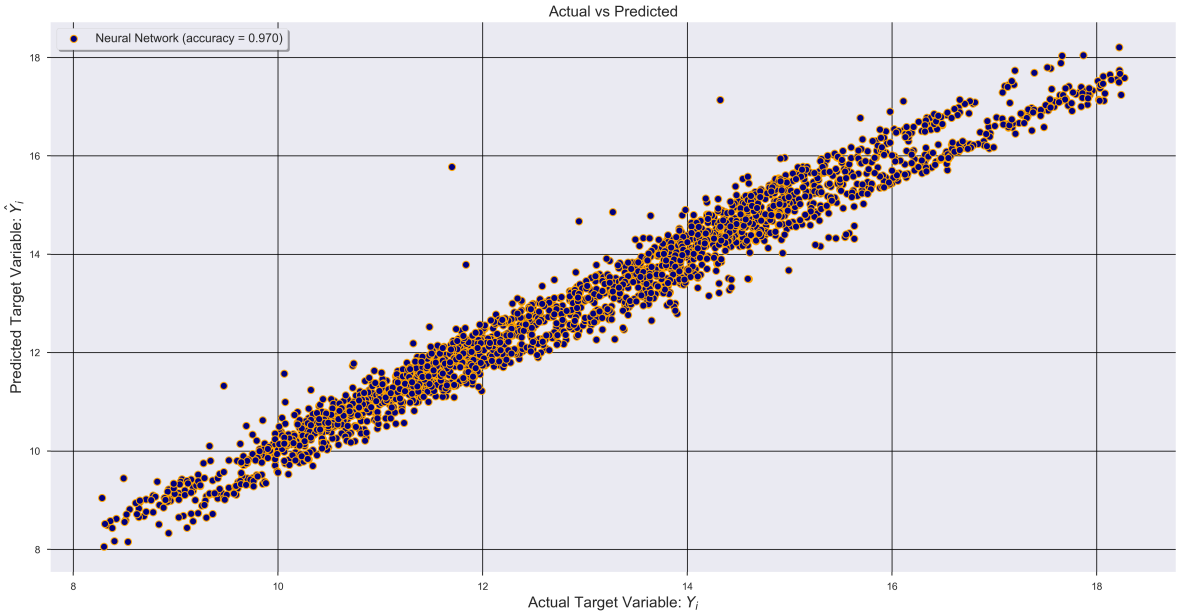
	batch_size	epochs	steps	samples	verbose	do_validation	metrics
0	5	150	None	7500	2	True	loss
1	5	150	None	7500	2	True	mean_squared_error
2	5	150	None	7500	2	True	mean_absolute_error
3	5	150	None	7500	2	True	val_loss
4	5	150	None	7500	2	True	val_mean_squared_error
5	5	150	None	7500	2	True	val_mean_absolute_error

Model Evaluation

In [16]:

```
model_name = 'Neural Network'
helper.evaluate(X_test, y_test, model_name, neural_network_model)
```

```
Neural Network Mean Squared Error: 0.14618774985909064
Neural Network Root Mean Squared Error: 0.38234506647672417
Neural Network R2 Score: 0.9697627857246928
Neural Network Explained Variance Score: 0.9698258542620452
Neural Network Mean Absolute Error: 0.29460089848033
Neural Network Meadian Abosulute Error: 0.2666202926635748
Neural Network Mean Squared Log Error: 0.0007157302181291301
```



Prediction

In [17]:

```
y_pred = neural_network_model.predict(X_test)
```

In [18]:

```
model_prediction_results = pd.DataFrame({
    'actual_generation' : list(y_test),
    'predicted_generation' : list(y_pred),
})
```

In [19]:

```
1 model_prediction_results.head()
```

Out[19]:

	actual_generation	predicted_generation
0	17.596	[17.10944]
1	15.630	[15.234886]
2	10.850	[11.42171]
3	14.520	[14.833691]
4	8.420	[8.61916]

In [20]:

```
1 model_prediction_results.dtypes
```

Out[20]:

```
actual_generation       float64
predicted_generation     object
dtype: object
```

In [21]:

```
1 # data['y_pred_generation'].str.replace('[\[\]]', '')
```

In [22]:

```
1 model_prediction_results = model_prediction_results.astype({'predicted_generation':'flo
```

In [23]:

```
model_prediction_results.head(10)
```

Out[23]:

	actual_generation	predicted_generation
0	17.596	17.109440
1	15.630	15.234886
2	10.850	11.421710
3	14.520	14.833691
4	8.420	8.619160
5	13.640	14.780686
6	14.010	13.918062
7	9.699	9.447412
8	13.900	12.787080
9	11.150	11.148086

In [24]:

```
model_prediction_results.to_csv("neural_network_predicted_values.csv", index=True)
```

Saving the Model

In [25]:

```
#Saving model
neural_network_model.save('nueral_network_model.h5')
```

Optimizing the Hyperparameter of the Neural Network with `GridSearchCV`

In [26]:

```
# Setting Random Seed
np.random.seed(82)
```

In [27]:

```
def neural_network_regressor_model(optimizer, activation):
    model = Sequential()

    # Input Layer
    model.add(Dense(20, input_dim=X_train.shape[1], kernel_initializer='normal', activa

    # Hidden Layers
    model.add(Dense(40, kernel_initializer='normal', activation=activation))
    model.add(Dense(40, kernel_initializer='normal', activation=activation))

    # Output Layer
    model.add(Dense(1, kernel_initializer='normal', activation='linear'))

    # Compiling the network :
    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mean_square

    print(model.summary())
    return model
```

In [28]:

```
epochs = 150
batch_size = 5

nn_model = KerasRegressor(
    build_fn=neural_network_regressor_model,
    epochs=epochs,
    batch_size=batch_size,
    verbose=2
)
```

In [ ]:

```

```

In [29]:

```
# Kfold with with n_splits = 5 to split the Dataset into 5-folds
kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=23)

# Dictionary of parameters to optimize
parameters = {
    "activation" : ['tanh', 'relu', 'elu', 'selu'],
    "optimizer" : ['adam', 'rmsprop'],
    "batch_size" : [5, 10, 15, 20],
    "epochs" : [50, 100, 150],
}

# Scoring Metric
scorer = "r2"

# Instantiating Search object
grid = model_selection.RandomizedSearchCV(
    estimator=nn_model,
    param_distributions=parameters,
    scoring=scorer,
    cv=kfold,
    n_jobs=1,
    verbose=2,
)

# Fit the grid object on Training Dataset
grid.fit(X_train, y_train)
```

```
 - 0s - loss: 0.1532 - mean_squared_error: 0.1532 - mean_absolute_error:
0.3095
Epoch 134/150
 - 0s - loss: 0.1537 - mean_squared_error: 0.1537 - mean_absolute_error:
0.3112
Epoch 135/150
 - 0s - loss: 0.1553 - mean_squared_error: 0.1553 - mean_absolute_error:
0.3103
Epoch 136/150
 - 0s - loss: 0.1533 - mean_squared_error: 0.1533 - mean_absolute_error:
0.3101
Epoch 137/150
 - 0s - loss: 0.1531 - mean_squared_error: 0.1531 - mean_absolute_error:
0.3098
Epoch 138/150
 - 0s - loss: 0.1541 - mean_squared_error: 0.1541 - mean_absolute_error:
0.3105
Epoch 139/150
 - 0s - loss: 0.1528 - mean_squared_error: 0.1528 - mean_absolute_error:
0.3079
```

In [30]:

```
# Saving Hyperparameter optimization results as a DataFrame
results = pd.DataFrame(grid.cv_results_)[["params", "mean_test_score", "std_test_score"
results.sort_values("rank_test_score", inplace=True)
results.to_csv("neural_network_hyperparameter_optimization_results.csv", index=True)
results
```

Out[30]:

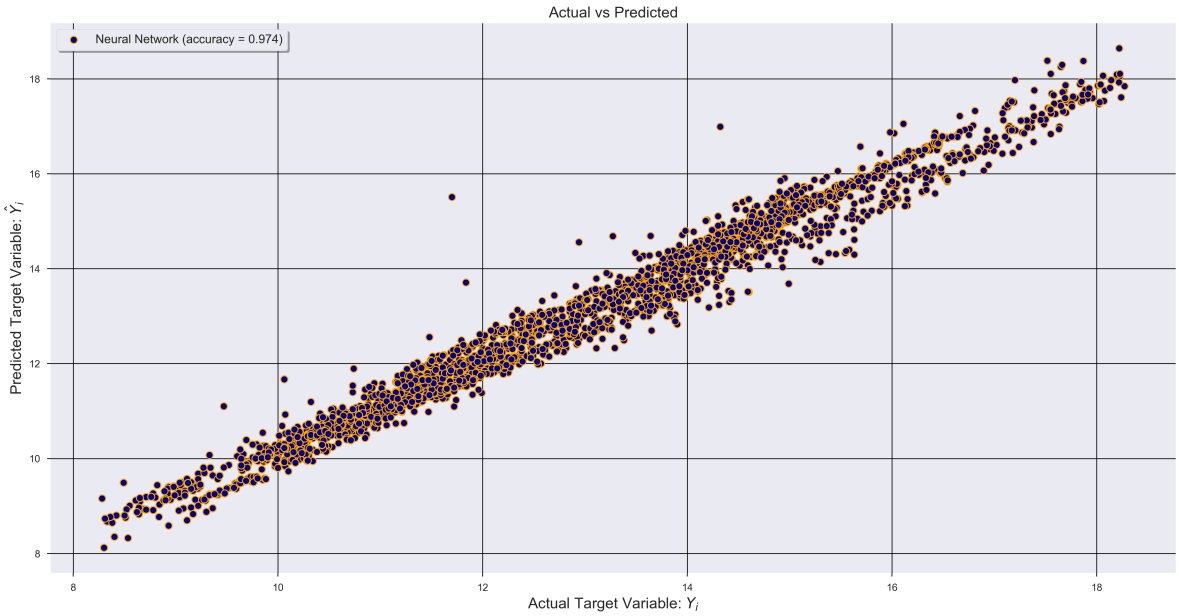
	params	mean_test_score	std_test_score	rank_test_score
2	{'optimizer': 'adam', 'epochs': 50, 'batch_siz...	0.975144	0.001759	1
4	{'optimizer': 'adam', 'epochs': 150, 'batch_si...	0.975027	0.002148	2
5	{'optimizer': 'adam', 'epochs': 50, 'batch_siz...	0.973465	0.001229	3
6	{'optimizer': 'adam', 'epochs': 100, 'batch_si...	0.972549	0.001911	4
3	{'optimizer': 'adam', 'epochs': 100, 'batch_si...	0.972254	0.001825	5
8	{'optimizer': 'rmsprop', 'epochs': 150, 'batch...	0.971457	0.001375	6
7	{'optimizer': 'adam', 'epochs': 100, 'batch_si...	0.969546	0.004080	7
9	{'optimizer': 'rmsprop', 'epochs': 50, 'batch_...	0.968880	0.007177	8
1	{'optimizer': 'rmsprop', 'epochs': 50, 'batch_...	0.968604	0.004121	9
0	{'optimizer': 'adam', 'epochs': 50, 'batch_siz...	0.966260	0.003683	10

Evaluating Best Estimator

In [31]:

```
1 best_estimator = grid.best_estimator_
2 helper.evaluate(X_test, y_test, "Neural Network", best_estimator)
```

```
Neural Network Mean Squared Error: 0.1242707793972232
Neural Network Root Mean Squared Error: 0.3525206084716512
Neural Network R2 Score: 0.9742960529290915
Neural Network Explained Variance Score: 0.9749624975053861
Neural Network Mean Absolute Error: 0.27229552823035874
Neural Network Meadian Abosulute Error: 0.22310390472412145
Neural Network Mean Squared Log Error: 0.0006259880454793604
```



Predicting with the Best Estimator and Saving Predicted Results as `csv`

In [34]:

```
tune_y_pred = best_estimator.predict(X_test)

hyp_tune_data = pd.DataFrame(
    {"actual_generation": list(y_test), "predicted_generation": list(tune_y_pred),}
)

hyp_tune_data.to_csv("best_estimator_predicted_values.csv", index=True)

hyp_tune_data.head(10)
```

Out[34]:

	actual_generation	predicted_generation
0	17.596	17.467146
1	15.630	15.372869
2	10.850	11.452371
3	14.520	14.715870
4	8.420	8.799006
5	13.640	14.690557
6	14.010	13.920854
7	9.699	9.571536
8	13.900	12.830315
9	11.150	11.287091

In [ ]:

```

```

Saving the Model

In [35]:

```
import joblib

joblib.dump(best_estimator, "optimized_neural_network_model.joblib")
```

Out[35]:

```
['optimized_neural_network_model.joblib']
```

In [ ]:

```

```

In [38]:

```
1 best_estimator.get_params()
```

Out[38]:

```
{'epochs': 50,
 'batch_size': 5,
 'verbose': 2,
 'optimizer': 'adam',
 'activation': 'selu',
 'build_fn': <function __main__.neural_network_regressor_model(optimizer, ac
tivation)>}
```

In [ ]:

```
1
```