In [1]:

```
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the s
econd argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
C:\Users\Admin\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:523:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Admin\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:524:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Admin\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Admin\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:526:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Admin\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:527:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Admin\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:532:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [2]:

```
import pandas as pd
df = pd.read_csv('C:/Users/Admin/Desktop/Datasets/IoT/Beach_1.csv')
print(df)
```

```
             Beach Name  Water Temperature  Turbidity  Transducer Depth  \
0         Montrose Beach               20.3       1.18             0.891
1      Ohio Street Beach               14.4       1.23             0.800
2          Calumet Beach               16.2       1.26             1.514
3         Montrose Beach               14.4       3.36             1.388
4         Montrose Beach               14.5       2.72             1.395
5          Calumet Beach               16.3       1.28             1.524
6         Montrose Beach               14.8       2.97             1.386
7          Calumet Beach               16.5       1.32             1.537
8          Calumet Beach               16.8       1.31             1.568
9         Montrose Beach               14.5       4.30             1.377
10         Calumet Beach               17.1       1.37             1.520
11        Montrose Beach               14.4       4.87             1.366
12         Calumet Beach               17.2       1.48             1.525
13        Montrose Beach               14.1       5.06             1.382
14        Montrose Beach               14.2       5.76             1.415
15         Calumet Beach               17.1       1.80             1.501
16         Calumet Beach               17.0       1.82             1.487
17        Montrose Beach               14.2       6.32             1.386
18        Montrose Beach               14.4       6.89             1.401
19         Calumet Beach               17.0       1.83             1.488
20        Montrose Beach               14.5       7.11             1.374
21         Calumet Beach               16.8       1.90             1.494
22        Montrose Beach               14.5       6.88             1.413
23         Calumet Beach               16.7       1.83             1.467
```

24	Montrose Beach	14.3	7.32	1.406
25	Calumet Beach	16.5	1.69	1.548
26	Calumet Beach	16.3	1.62	1.519
27	Montrose Beach	14.2	7.18	1.460
28	Montrose Beach	14.2	6.35	1.450
29	Calumet Beach	16.2	1.57	1.535
...	...	...	...	...
34892	Ohio Street Beach	19.3	2.47	NaN
34893	Ohio Street Beach	19.7	2.54	NaN
34894	Ohio Street Beach	19.9	2.57	NaN
34895	Ohio Street Beach	20.3	2.60	NaN
34896	Ohio Street Beach	20.3	2.58	NaN
34897	Ohio Street Beach	20.4	2.45	NaN
34898	Ohio Street Beach	20.5	2.47	NaN
34899	Ohio Street Beach	20.2	2.61	NaN
34900	Ohio Street Beach	20.2	2.60	NaN
34901	Ohio Street Beach	20.1	2.64	NaN
34902	Ohio Street Beach	20.0	2.54	NaN
34903	Ohio Street Beach	19.9	2.56	NaN
34904	Ohio Street Beach	19.8	2.43	NaN
34905	Ohio Street Beach	19.7	2.48	NaN
34906	Ohio Street Beach	19.6	2.57	NaN
34907	Ohio Street Beach	19.5	2.54	NaN
34908	Ohio Street Beach	19.5	2.75	NaN
34909	Ohio Street Beach	19.4	2.84	NaN
34910	Ohio Street Beach	19.4	2.83	NaN
34911	Ohio Street Beach	19.3	2.81	NaN
34912	Ohio Street Beach	19.3	2.85	NaN
34913	Ohio Street Beach	19.3	2.76	NaN
34914	Ohio Street Beach	19.3	2.58	NaN
34915	Ohio Street Beach	19.5	2.47	NaN
34916	Ohio Street Beach	19.8	2.39	NaN
34917	Ohio Street Beach	19.9	2.61	NaN
34918	Ohio Street Beach	19.8	0.00	NaN
34919	Ohio Street Beach	22.3	0.00	NaN
34920	Ohio Street Beach	21.1	26.97	NaN
34921	Ohio Street Beach	21.3	27.55	NaN

	Wave Height	Wave Period	Battery Life
0	0.080	3.0	9.4
1	0.111	4.0	12.4
2	0.147	4.0	11.7
3	0.298	4.0	11.9
4	0.306	3.0	11.9
5	0.162	4.0	11.7
6	0.328	3.0	11.9
7	0.185	4.0	11.7
8	0.196	4.0	11.7
9	0.328	3.0	11.9
10	0.194	4.0	11.7
11	0.341	3.0	11.9
12	0.203	4.0	11.7
13	0.340	4.0	11.9
14	0.356	3.0	11.9
15	0.188	4.0	11.7
16	0.194	4.0	11.7
17	0.346	3.0	11.9
18	0.380	4.0	11.9
19	0.196	4.0	11.7
20	0.361	5.0	11.9
21	0.181	4.0	11.7
22	0.345	4.0	11.9
23	0.180	4.0	11.7
24	0.331	4.0	11.9
25	0.177	4.0	11.7
26	0.159	4.0	11.7
27	0.305	4.0	11.9
28	0.321	3.0	11.9
29	0.154	4.0	11.7
...	...	...	...
34892	0.187	3.0	10.6
34893	0.187	3.0	10.6
34894	0.187	3.0	10.6
34895	0.187	3.0	10.6
34896	0.187	3.0	10.6
34897	0.187	3.0	10.6
34898	0.187	3.0	10.6

```
34899        0.187          3.0          10.6
34900        0.187          3.0          10.6
34901        0.187          3.0          10.5
34902        0.187          3.0          10.6
34903        0.187          3.0          10.5
34904        0.187          3.0          10.5
34905        0.187          3.0          10.5
34906        0.187          3.0          10.5
34907        0.187          3.0          10.5
34908        0.187          3.0          10.5
34909        0.187          3.0          10.5
34910        0.187          3.0          10.5
34911        0.187          3.0          10.5
34912        0.187          3.0          10.5
34913        0.187          3.0          10.5
34914        0.187          3.0          10.5
34915        0.187          3.0          10.5
34916        0.187          3.0          10.5
34917        0.187          3.0          10.5
34918        0.187          3.0          10.5
34919        0.187          3.0          10.5
34920        0.187          3.0           9.4
34921        0.187          3.0           9.4

[34922 rows x 7 columns]
```

In [3]:

```
df.fillna(df.mean(), inplace=True)
X = df.iloc[:, :-1].values
print(X)
```

```
[['Montrose Beach' 20.3 1.18 0.8909999999999999 0.08 3.0]
 ['Ohio Street Beach' 14.4 1.23 0.8 0.111 4.0]
 ['Calumet Beach' 16.2 1.26 1.514 0.147 4.0]
 ...
 ['Ohio Street Beach' 22.3 0.0 1.570197608370705 0.187 3.0]
 ['Ohio Street Beach' 21.1 26.97 1.570197608370705 0.187 3.0]
 ['Ohio Street Beach' 21.3 27.55 1.570197608370705 0.187 3.0]]
```

In [4]:

```
y = df.iloc[:, -1].values
print(y)
```

```
[ 9.4 12.4 11.7 ... 10.5  9.4  9.4]
```

In [5]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:,0] = labelencoder_X.fit_transform(X[:,0])
X[:,1] = labelencoder_X.fit_transform(X[:,1])
onehotencoder = OneHotEncoder(categorical_features=[0,1])
X = onehotencoder.fit_transform(X).toarray()
print(X)
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:415: FutureWarning:
The handling of integer data will change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, t
hen you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:451:
DeprecationWarning: The 'categorical_features' keyword is deprecated in version 0.20 and will be r
emoved in 0.22. You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

```
[[0.         0.         1.         ... 0.891      0.08       3.        ]
 [0.         0.         0.         ... 0.8        0.111      4.        ]
 [0.         1.         0.         ... 1.514      0.147      4.        ]
 ...
 [0.         0.         0.         ... 1.57019761 0.187      3.        ]
 [0.         0.         0.         ... 1.57019761 0.187      3.        ]
 [0.         0.         0.         ... 1.57019761 0.187      3.        ]]
```

In [6]:

```
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
print(y)
```

```
[46 77 70 ... 57 46 46]
```

In [28]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = X_train.shape[1], activation='relu'
))

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(128, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(64, kernel_initializer='normal',activation='relu'))


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

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_error'])
NN_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 128)               26496
_________________________________________________________________
dense_7 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_8 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
dense_11 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_12 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 65
=================================================================
Total params: 298,113
Trainable params: 298,113
Non-trainable params: 0
_________________________________________________________________
```

In [29]:

```
NN_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split = 0.2)
```

```
Train on 22349 samples, validate on 5588 samples
Epoch 1/10
```

```
22349/22349 [==============================] - 6s 251us/step - loss: 11.0254 -
mean_absolute_error: 11.0254 - val_loss: 6.5209 - val_mean_absolute_error: 6.5209
Epoch 2/10
22349/22349 [==============================] - 5s 235us/step - loss: 6.2733 - mean_absolute_error:
6.2733 - val_loss: 6.0383 - val_mean_absolute_error: 6.0383
Epoch 3/10
22349/22349 [==============================] - 5s 229us/step - loss: 6.1896 - mean_absolute_error:
6.1896 - val_loss: 5.9775 - val_mean_absolute_error: 5.9775
Epoch 4/10
22349/22349 [==============================] - 5s 219us/step - loss: 6.1083 - mean_absolute_error:
6.1083 - val_loss: 6.4070 - val_mean_absolute_error: 6.4070
Epoch 5/10
22349/22349 [==============================] - 5s 239us/step - loss: 5.9265 - mean_absolute_error:
5.9265 - val_loss: 6.4361 - val_mean_absolute_error: 6.4361
Epoch 6/10
22349/22349 [==============================] - 6s 248us/step - loss: 5.7987 - mean_absolute_error:
5.7987 - val_loss: 5.6761 - val_mean_absolute_error: 5.6761
Epoch 7/10
22349/22349 [==============================] - 5s 230us/step - loss: 5.7047 - mean_absolute_error:
5.7047 - val_loss: 5.4867 - val_mean_absolute_error: 5.4867
Epoch 8/10
22349/22349 [==============================] - 5s 231us/step - loss: 5.6993 - mean_absolute_error:
5.6993 - val_loss: 5.7559 - val_mean_absolute_error: 5.7559
Epoch 9/10
22349/22349 [==============================] - 5s 235us/step - loss: 5.5399 - mean_absolute_error:
5.5399 - val_loss: 5.6191 - val_mean_absolute_error: 5.6191
Epoch 10/10
22349/22349 [==============================] - 5s 222us/step - loss: 5.4199 - mean_absolute_error:
5.4199 - val_loss: 5.4349 - val_mean_absolute_error: 5.4349
```

Out[29]:

```
<keras.callbacks.History at 0x1f95b897c88>
```

In [21]:

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

In [23]:

```
import numpy as np
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 5.364148267337741
Mean Squared Error: 70.70993673055644
Root Mean Squared Error: 8.40892006922152
```

In [27]:

```
# The mean squared error
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_pred))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
Mean squared error: 70.71
Test Variance score: -0.09
```

In [30]:

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
df
```

Out[30]:

	Actual	Predicted
0	60	58.515812

	Actual	Predicted
1	53	59.010757
2	80	70.428925
3	70	68.356750
4	61	60.561039
5	74	65.453171
6	69	68.435074
7	61	63.985359
8	68	63.930935
9	54	94.429367
10	60	62.069714
11	66	67.260498
12	65	67.502502
13	67	67.525757
14	58	65.121918
15	56	69.256012
16	73	66.309341
17	60	62.495434
18	62	63.536339
19	55	54.241169
20	60	68.007812
21	56	61.522530
22	75	63.542027
23	71	65.920059
24	57	61.510189
25	60	58.566593
26	68	61.484989
27	72	64.651718
28	67	66.188828
29	69	60.236370
...	...	...
6955	54	64.398697
6956	73	62.567959
6957	55	62.917847
6958	58	62.115528
6959	66	60.006577
6960	59	62.962963
6961	65	60.035976
6962	60	63.045200
6963	62	59.587864
6964	66	62.554928
6965	70	65.524567
6966	59	64.628632
6967	71	63.319477
6968	64	64.743759
6969	72	64.828217
6970	65	62.099205
6971	70	68.847870
6972	75	59.618610
6973	68	66.648834
6974	68	67.375153
6975	76	65.390030

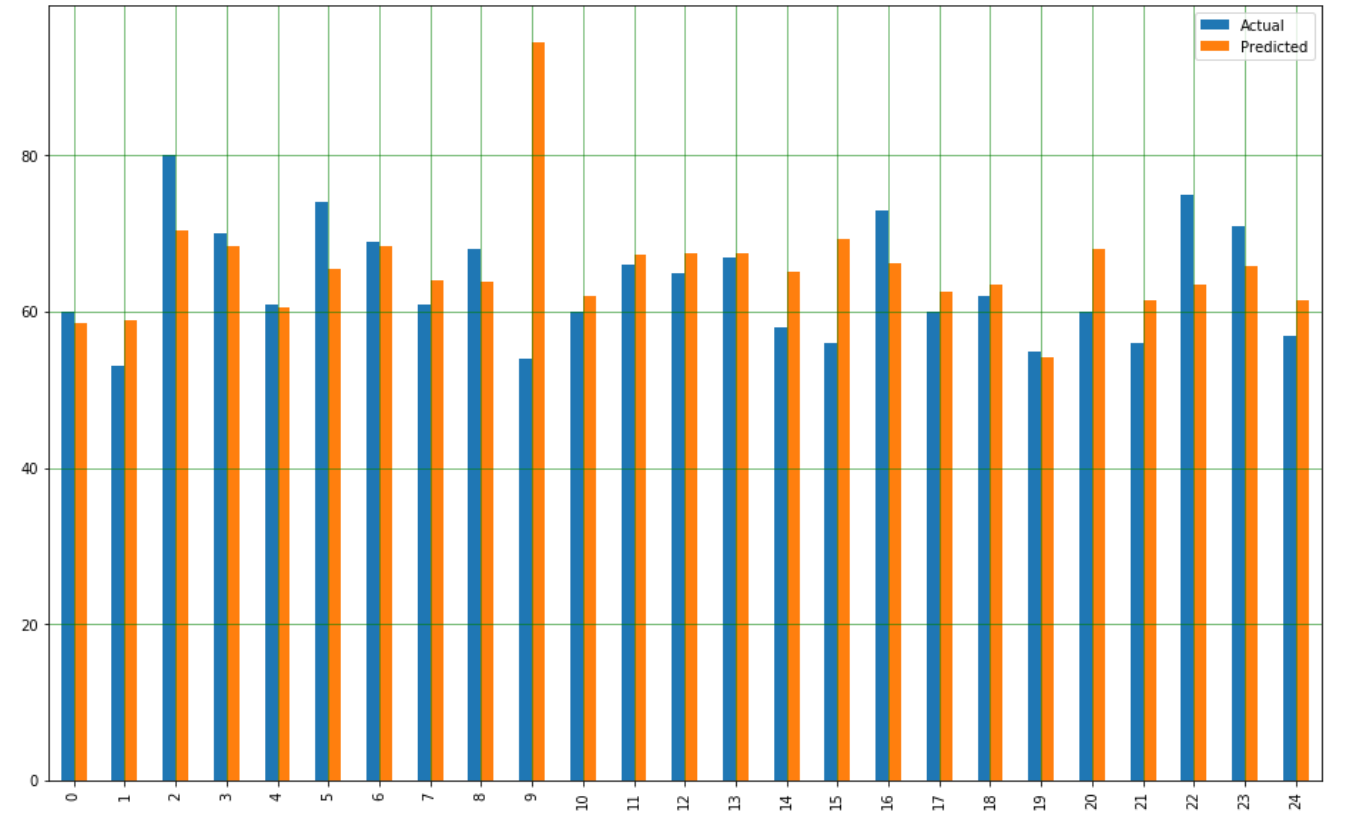
	Actual	Predicted
6976	53	66.092728
6977	67	64.986969
6978	60	60.440258
6979	66	68.853500
6980	68	64.012817
6981	60	62.329510
6982	68	65.546402
6983	66	63.958569
6984	60	61.615685

6985 rows × 2 columns

In [32]:

```
import matplotlib.pyplot as plt

df1 = df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



In [ ]: