Configuration Manual

MSc in Data Analytics

Saranya Varshni Roshan Karthikha
Student ID: x20154801

School of Computing
National College of Ireland

Supervisor: Dr. Bharathi Chakravarthi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saranya Varshni Roshan Karthikha
Student ID:	x20154801
Programme:	MSc in Data Analytics
Year:	2021
Module:	MSc in Data Analytics
Supervisor:	Dr. Bharathi Chakravarthi
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	704
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Saranya Varshni Roshan Karthikha
Date:	16th December 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Saranya Varshni Roshan Karthikha
x20154801

1 Introduction

The aim of this configuration manual is to demonstrate different steps involved in the research project implementation. The research project is based on forecasting energy generation from different renewable energy sources using ARIMA and neural network models. This project will require Python packages that are to be installed in the local machine. A jupyter environment created using Anaconda is required.

2 System Configuration

The project has been performed on the below specified Hardware configuration Figure 1.

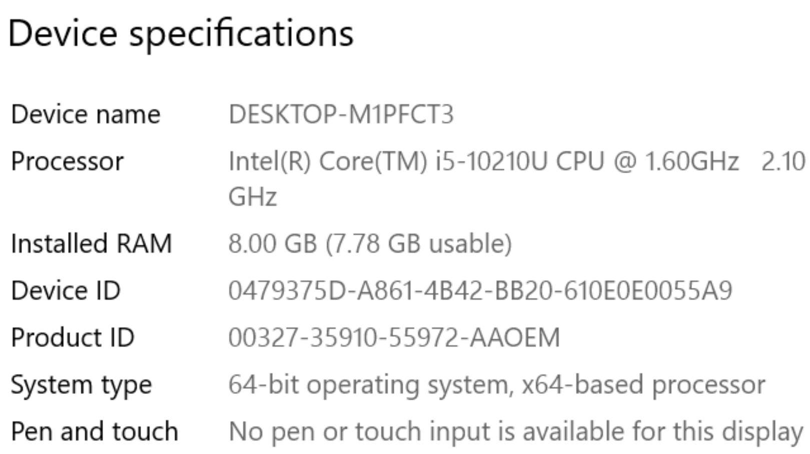


Figure 1: Device Configuration

3 Software Specification

This project was implemented using 'Anaconda Jupyter Notebook'

1. **Anaconda** - It is an open source free to use software. It had python 3.8.8 by default. The Jupyter notebook was used to execute the project code.

Hardware	Specification
Operating System	Windows 10
Processor	Intel(R) Core(TM) i5-10210U
RAM	8 GB
Hard Disk	1 TB
Software	**Versions**
Anaconda	1.7.2
Python	3.9.5
Numpy	1.19.4
Matplotlib	3.3.4
Sklearn	0.24.1

Figure 2: Versions

4 Installation

4.1 Anaconda

- Anaconda software can be downloaded from **Link**
- Basic installation instructions are adequate for complete installation
- Once the environment is up, Python is to be installed from **Click Here**

5 Package required

Figure 3 shows the python libraries required for data cleaning

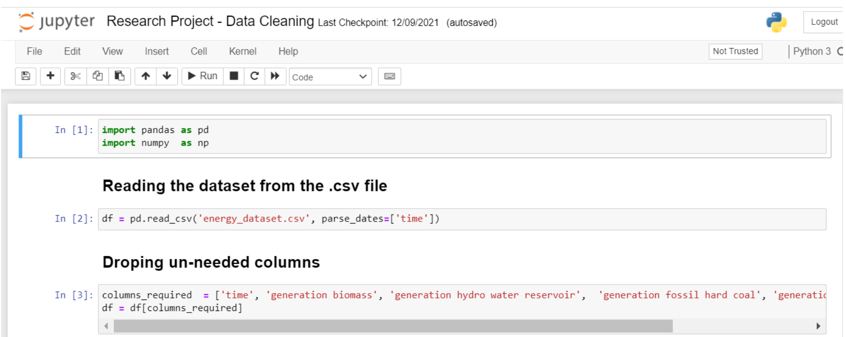


Figure 3: Packages for Data Clean

Figure 4 shows the python libraries required for model implementation

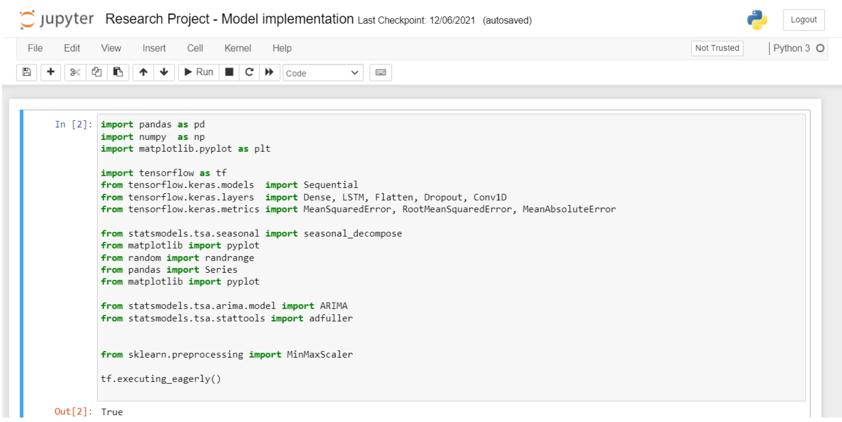


Figure 4: Importing the necessary Libraries

6 Project Development

- All the standard libraries and packages were installed using pip install command in the Anaconda Library including numpy, pandas, matplotlib, sklearn, statsmodel etc.,

- **Data Gathering:**

  The primary step with data aggregation was to find on the granularity of the data. It was found to be an hourly granularity. The initial data pre processing and implementation required Anaconda environment. With the help of Pandas library, the CSV file was loaded as a data frame.

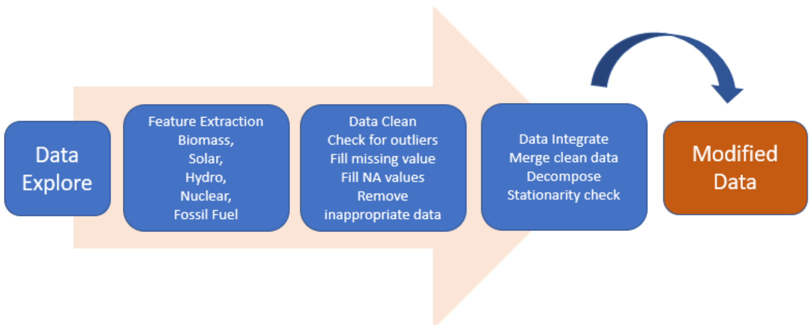


Figure 5: Data processing Flow

- **Data Preprocessing:**

  Once the dataset was uploaded to the anaconda environment, the csv file was imported into python jupyter notebook. Figure 6 shows that the time feature in the dataset was set as index as it is associated in temporal order with other features.

  There were 64 features in total, but only solar, hydro water reservoir, nuclear, and fossil hard coal were required for the research implementation. Hence, the remaining other columns were dropped as show in Figure 7

Making the 'time' column as index

```
df['time'] = pd.to_datetime(df['time'], utc=True, infer_datetime_format=True)
df.set_index('time', inplace=True)
```

Figure 6: Setting time feature as index

Droping un-needed columns

```
columns_required  = ['time', 'generation biomass', 'generation hydro water reservoir',  'generation fossil hard coal', 'generatic
df = df[columns_required]
```

Figure 7: Removing unnecessary features

As the first step in data cleaning, the dataset is examined for missing values as show in Figure 8 . In total record count, 90 missing values were identified.

Now that the dataset in required format, we will start cleaning the data

```
print(
        f'Number of missing values in our dataset: {df.isnull().values.sum()}'
)
```

```
Number of missing values in our dataset: 90
```

Figure 8: Examine for missing values

All the missing values, almost all belongs to similar record. Hence, data manipulation is carried out.

one of the following can be performed:

- fill these missing values with average values of the column
- drop these entire rows
- find a better way of filling them

The first method of filling entire rows with averages will not generate meaningful data, but will only create outliers. The second way will create a discrepancy in the time differences of the dataset. So, fill the missing values using time-based interpolation Figure 9 which pandas already provides us.

Figure 10 shows that after data manipulation there are no missing values.

Figure 11 depicts that the cleaned dataset is extracted for further modelling.

7 Model Implementation

- Figure 12 Total data size is measured
- Basic visualization is done on all 5 renewable energy sources Figure 13
- Observed seasonality is regulated for all 5 sources Figure 14

In [8]:
```
df.interpolate(method='time', limit_direction='forward', inplace=True, axis=0)
```

Figure 9: Using interpolate function for data manipulation

Lets check for missing values in the dataset now

```
df.isnull().sum(axis=0)
```

```
generation biomass                0
generation hydro water reservoir  0
generation fossil hard coal       0
generation nuclear                0
generation solar                  0
dtype: int64
```

Figure 10: Checking on the count of missing values once again

As it can be seen above, the dataset is clean and now it can be saved.

In [11]:
```
df.to_csv('energy_dataset_cleaned.csv', index_label='time')
```

Figure 11: Cleaned dataset extracted

Reading the data

In [10]:
```
df = pd.read_csv('energy_dataset_cleaned.csv', parse_dates=['time'], index_col='time')
```

First of all lets see how many days worth of data do we have

In [11]:
```
df.index[-1] - df.index[0]
```

Out[11]:
```
Timedelta('1460 days 23:00:00')
```

Figure 12: Check on the data Size

Lets plot visualize every generation type

```
In [13]: figure, axes = plt.subplots(2, 3, figsize=(20, 10))


         df['generation biomass'].plot(ax=axes[0][0], title='generation biomass')
         df['generation hydro water reservoir'].plot(ax=axes[0][1], title='generation hydro water reservoir')
         df['generation fossil hard coal'].plot(ax=axes[0][2], title='generation fossil hard coal')
         df['generation nuclear'].plot(ax=axes[1][0], title='generation nuclear')
         df['generation solar'].plot(ax=axes[1][1], title='generation solar')

Out[13]: <AxesSubplot:title={'center':'generation solar'}, xlabel='time'>
```

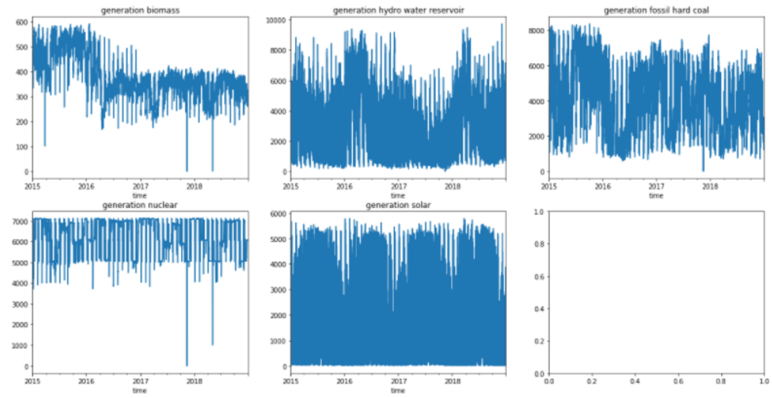
- 

Figure 13: Plots of different generation type

```
In [10]: # Season Periods for every generation type

         biomass_season_period              = 30 * 24    # 30 days
         nuclear_season_period              = 30 * 24    # 30 days
         solar_season_period                = 1  * 24    # 1 day
         fossilHardCoal_season_period       = 30 * 24    # 30 days
         hydroWaterResorvoir_season_period  = 1  * 24    # 1 day
```

-

Figure 14: Observed Seasonality

- Correlation heat map to analyse the relation between each other of the columns Figure 15

Finally lets check the correlation between different generation types

```
In [11]: corr = df.corr()
         plt.imshow(corr)
         corr
```

Out[11]:

	generation biomass	generation hydro water reservoir	generation fossil hard coal	generation nuclear	generation solar
generation biomass	1.000000	-0.033307	0.433734	-0.021053	-0.004730
generation hydro water reservoir	-0.033307	1.000000	-0.157031	-0.049237	0.091661
generation fossil hard coal	0.433734	-0.157031	1.000000	-0.023150	0.045906
generation nuclear	-0.021053	-0.049237	-0.023150	1.000000	0.003911
generation solar	-0.004730	0.091661	0.045906	0.003911	1.000000

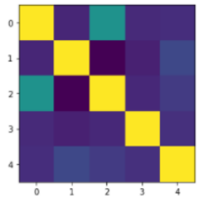


Figure 15: Correlation heat map

- Augmented Dickey Fuller test performed to check on the stationary behaviour of the data Figure 16.

```
In [42]: result = adfuller(df['generation nuclear'][:24*30*12])
         print('ADF Statistic: %f' % result[0])
         print('p-value: %f' % result[1])
         print('Critical Values:')
         for key, value in result[4].items():
             print('\t%s: %.3f' % (key, value))

         ADF Statistic: -8.680778
         p-value: 0.000000
         Critical Values:
                 1%: -3.431
                 5%: -2.862
                 10%: -2.567
```

Figure 16: Augmented Dickey Fuller test

8 LSTM implementation

- Takes in a 2d array like object and returns a windowed feature (3d) array and it's corresponding label array (3d) Figure ??
- Figure 18 Dataset is split into train, validation and Test set.
- Figure 19 used to visualize the train and validation set.
- Figure 20 LSTM model creation and model architecture.
- Figure 21 LSTM model training.

Data Preprocessing

```
In [12]: def window(arr, nPast, nFuture):
             '''
             arr: 2D array like object
             nPast: window size of the required feature array
             nFuture: how far in the future the required label is

             takes in a 2d array like object and returns a windowed feature (3d) array and it's corresponding label array (3d)
             '''
             winSize = nFuture + nPast

             data = np.stack([
                 arr[i:i-winSize]
                 for i in range(winSize)
             ]
             ).transpose([1,0,2])

             data = data[
                 np.random.permutation(data.shape[0])
             ]

             X = data[:, :nPast,    :].copy()
             y = data[:, -nFuture:, :].copy()

             return X, y
```

```
In [13]: colNames        = df.columns
         generationTypes = [ 'biomass', 'hwr', 'fhc', 'nuclear', 'solar' ]

         nPast   = 24
         nFuture = 1
         winSize = nPast + nFuture

         num_attr    = 1
         input_shape = (nPast, num_attr)
```

Figure 17: Data Processing

Splitting Data into Train, Validation and Test sets

```
In [14]: train_size = 24 * 365 * 3          # first 3 years of data
         val_size   = int(24 * 365 * 0.5)   # first 6 months of last year of data
         test_size  = int(24 * 365 * 0.5)   # last 6 months of last year of data
```

```
In [15]: train   = df[:train_size]
         val     = df[train_size: train_size + val_size]
         test    = df[train_size + val_size:]
```

```
In [16]: print(f'''
             Total data is {len(df)    // (24*365)} years
             Training is   {len(train) // (24*365)} years of data
             Validation is {len(val)   /  (24*365)} years of data
             Testing is    {len(test)  /  (24*365)} years of data
         ''')

             Total data is 4 years
             Training is   3 years of data
             Validation is 0.5 years of data
             Testing is    0.5027397260273972 years of data
```

Figure 18: Dataset Split

Neural Networks

```
In [54]: loss    = tf.keras.losses.MeanSquaredError()
         metrics = [RootMeanSquaredError(), MeanAbsoluteError()]

         # lr_schedule = tf.keras.callbacks.LearningRateScheduler(
         #                   lambda epoch: 1e-3 * 10**(epoch / 10)
                        #   )
         early_stopping = tf.keras.callbacks.EarlyStopping(patience=10)
```

```
In [31]: def plot_save_history(history, path):

             # Evaluate train and validation accuracies and losses

             train_rmse = history.history['root_mean_squared_error'][1:]
             val_rmse = history.history['val_root_mean_squared_error'][1:]

             train_mae = history.history['mean_absolute_error'][1:]
             val_mae = history.history['val_mean_absolute_error'][1:]

             train_loss = history.history['loss'][1:]
             val_loss = history.history['val_loss'][1:]

             # Visualize epochs vs. train and validation accuracies and losses

             plt.figure(figsize=(30, 10))
             plt.subplot(1, 3, 1)
             plt.plot(train_rmse, label='Training RMSE')
             plt.plot(val_rmse, label='Validation RMSE')
             plt.legend()
             plt.title('Epochs vs. Training and Validation RMSE')

             plt.subplot(1, 3, 2)
             plt.plot(train_loss, label='Training Loss')
             plt.plot(val_loss, label='Validation Loss')
             plt.legend()
             plt.title('Epochs vs. Training and Validation Loss')

             plt.subplot(1, 3, 3)
             plt.plot(train_mae, label='Training MAE')
             plt.plot(val_mae, label='Validation MAE')
             plt.legend()
             plt.title('Epochs vs. Training and Validation MAE')

             plt.savefig(path)
             plt.show()
```

Figure 19: Code for visualization

LSTM

Creating models

```
In [35]: lstm_models = {}

         for type in generationTypes:

             model = tf.keras.Sequential([
                 LSTM(100, input_shape=input_shape, return_sequences=True),
                 Flatten(),
                 Dense(200, activation='relu'),
                 Dropout(0.1),
                 Dense(num_attr)
             ],
                 name= f'lstm_{type}'
             )

             model_checkpoint = tf.keras.callbacks.ModelCheckpoint(f'models/lstm_{type}.h5', monitor=('val_loss'), save_best_only=True)
             optimizer        = tf.keras.optimizers.Adam(learning_rate=6e-2, amsgrad=True)

             model.compile(loss=loss, optimizer=optimizer, metrics=metrics)

             lstm_models[type] = model
```

Displaying models

```
In [36]: for type, model in lstm_models.items():
             print()
             print('='*100)
             print(f'the model for {type} generation: ')
             model.summary()
```

```
====================================================================================================
the model for biomass generation:
Model: "lstm_biomass"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 24, 100)           40800
_________________________________________________________________
flatten_1 (Flatten)          (None, 2400)              0
_________________________________________________________________
dense_2 (Dense)              (None, 200)               480200
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 201
=================================================================
Total params: 521,201
Trainable params: 521,201
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: LSTM model

training models

```
In [37]: lstm_models_history = {}

for type in generationTypes:
    tf.keras.backend.clear_session()            ## Clearing session to avoid any discrepency


    model   = lstm_models[type]

    x_train = datasets[type]['train']['x']
    y_train = datasets[type]['train']['y']

    x_val   = datasets[type]['val']['x']
    y_val   = datasets[type]['val']['y']

    history = model.fit(
        x_train, y_train,
        epochs          = 5,
        validation_data = ( x_val, y_val ),
        callbacks       = [early_stopping, model_checkpoint]
    )
    lstm_models_history[type] = history
```

Figure 21: LSTM train

9 LSTM-CNN implementation

- Figure 21 CNN- LSTM model creation and model Architecture.

10 Stacked LSTM implementation

- Figure 23 Stacked- LSTM model creation and model Architecture.

11 ARIMA implementation

- Figure 24 ARIMA model creation.

12 Evaluation

- Figure 25 Evaluation of all the 4 models implemented.

CNN-LSTM

Creating models

In [62]:
```
cnn_lstm_models = {}

for type in generationTypes:

    model = tf.keras.models.Sequential([
        Conv1D(filters=100, kernel_size=2, strides=1, padding='causal', activation='relu', input_shape=input_shape),
        LSTM(100, return_sequences=True),
        Flatten(),
        Dense(50, activation='relu'),
        Dense(num_attr)
    ])

    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(f'models/cnn_lstm_{type}.h5', monitor=('val_loss'), save_best_only=Tru
    optimizer        = tf.keras.optimizers.Adam(learning_rate=6e-2, amsgrad=True)

    model.compile(loss=loss, optimizer=optimizer, metrics=metrics)

    cnn_lstm_models[type] = model
```

Displaying models

In [47]:
```
for type, model in cnn_lstm_models.items():
    print()
    print('='*100)
    print(f'the model for {type} generation: ')
    model.summary()
```

```
====================================================================================================
the model for biomass generation:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 24, 100)           300
_________________________________________________________________
lstm (LSTM)                  (None, 24, 100)           80400
_________________________________________________________________
flatten (Flatten)            (None, 2400)              0
_________________________________________________________________
dense (Dense)                (None, 50)                120050
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 200,801
Trainable params: 200,801
Non-trainable params: 0
_________________________________________________________________
```

Figure 22: CNN- LSTM model creation and model Architecture.

Stacked-LSTMs

Creating models

```
In [34]: stacked_lstm_models = {}

for type in generationTypes:

    model = tf.keras.models.Sequential([
                    LSTM(100, input_shape=input_shape, return_sequences=True),
                    LSTM(50, return_sequences=True),
                    Flatten(),
                    Dense(150, activation='relu'),
                    Dropout(0.1),
                    Dense(num_attr)
])

    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(f'models/stacked_lstm_{type}.h5', monitor=('val_loss'), save_best_only
    optimizer        = tf.keras.optimizers.Adam(learning_rate=6e-2, amsgrad=True)

    model.compile(loss=loss, optimizer=optimizer, metrics=metrics)

    stacked_lstm_models[type] = model
```

Displaying models

```
In [35]: for type, model in stacked_lstm_models.items():
    print()
    print('='*100)
    print(f'the model for {type} generation: ')
    model.summary()
```

```
====================================================================================================
the model for biomass generation:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 24, 100)           40800
_________________________________________________________________
lstm_1 (LSTM)                (None, 24, 50)            30200
_________________________________________________________________
flatten (Flatten)            (None, 1200)              0
_________________________________________________________________
dense (Dense)                (None, 150)               180150
_________________________________________________________________
dropout (Dropout)            (None, 150)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 151
=================================================================
Total params: 251,301
```

Figure 23: Stacked LSTM model

Creating Dataset and Making Train Test Split

```
In [18]: arima_df    = df[:24*365]

size        = int(arima_df.shape[0] * (10/12))
train, test = arima_df[:size], arima_df[size:]
```

Arima

training and evaluating

```
In [21]: arima_preds = {}

for X_train, X_test, type in zip(train.values.T, test.values.T, generationTypes):

    history     = list(X_train)
    predictions = []

    # walk-forward validation
    for row in X_test:
        model         = ARIMA(history, order= (5,1,0))
        model_fit     = model.fit()
        pred         ,= model_fit.forecast()

        predictions.append(pred)
        history.append(row)
        history.pop(0)


    # evaluate forecasts
    mse  = mean_squared_error(X_test, predictions)
    mae  = mean_absolute_error(X_test, predictions)
    rmse = mse ** .5

    evals[f'ARIMA_{type}']  = [ mse, rmse, mae ]
    arima_preds[type]       = predictions
    # print('here')
```

Figure 24: ARIMA model

analysing and saving the evaluations

In [49]:
```
evaluations = pd.DataFrame(evals, index= errors).T
# evaluations.to_csv('model_evaluations.csv')
```

In [51]:
```
evaluations = pd.read_csv('model_evaluations.csv', index_col=0)
evaluations
```

Out[51]:

	MSE	RMSE	MAE
cnn_lstm_biomass	9.427014e-04	0.030703	0.018231
cnn_lstm_hwr	1.768821e-02	0.132997	0.104852
cnn_lstm_fhc	4.796336e-02	0.219005	0.185880
cnn_lstm_nuclear	3.286812e-04	0.018130	0.005645
cnn_lstm_solar	8.201557e-02	0.286384	0.244694
lstm_biomass	1.415877e-03	0.037628	0.024368
lstm_hwr	1.796482e-02	0.134033	0.109201
lstm_fhc	5.362865e-02	0.231579	0.195228
lstm_nuclear	1.244893e-02	0.111575	0.096310
lstm_solar	1.421944e-03	0.037709	0.029226
stacked_lstm_biomass	3.375910e-03	0.058103	0.039278
stacked_lstm_hwr	1.728150e-02	0.131459	0.099326
stacked_lstm_fhc	3.384789e-02	0.183978	0.152968
stacked_lstm_nuclear	6.713348e-04	0.025910	0.016577
stacked_lstm_solar	8.217511e-02	0.286662	0.245897
ARIMA_biomass	3.019281e+07	5494.798182	4472.078787
ARIMA_hwr	5.086833e+07	7132.203994	5507.973425
ARIMA_fhc	5.389466e+07	7341.298123	6157.616897
ARIMA_nuclear	2.660342e+07	5157.850542	3877.560486
ARIMA_solar	2.563690e+07	5063.289383	3705.724438

Figure 25: Evalutaion