Thermal Estimation of Power Electronic Converters

```
from google.colab import files
import pandas as pd
import io
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import BayesianRidge
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.preprocessing import normalize, minmax_scale
from sklearn.neural_network import MLPRegressor
import sklearn

import seaborn as sns
```

```
!pip install rainflow
import rainflow
# https://pypi.org/project/rainflow/
```

```
Collecting rainflow
  Downloading https://files.pythonhosted.org/packages/82/40/a7c674e534d4b364ee2e7e5cfc1a5118c8628
Requirement already satisfied: importlib_metadata in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from importli
Building wheels for collected packages: rainflow
  Building wheel for rainflow (setup.py) ... done
  Created wheel for rainflow: filename=rainflow-3.0.0-cp36-none-any.whl size=5124 sha256=c3287046
  Stored in directory: /root/.cache/pip/wheels/47/03/88/dd5203e5a812e55ec7d261342f4a549607f004ca3
Successfully built rainflow
Installing collected packages: rainflow
Successfully installed rainflow-3.0.0
```

```
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data375.xlsx to data375.xlsx
```

Model Training

Data Pre Processing

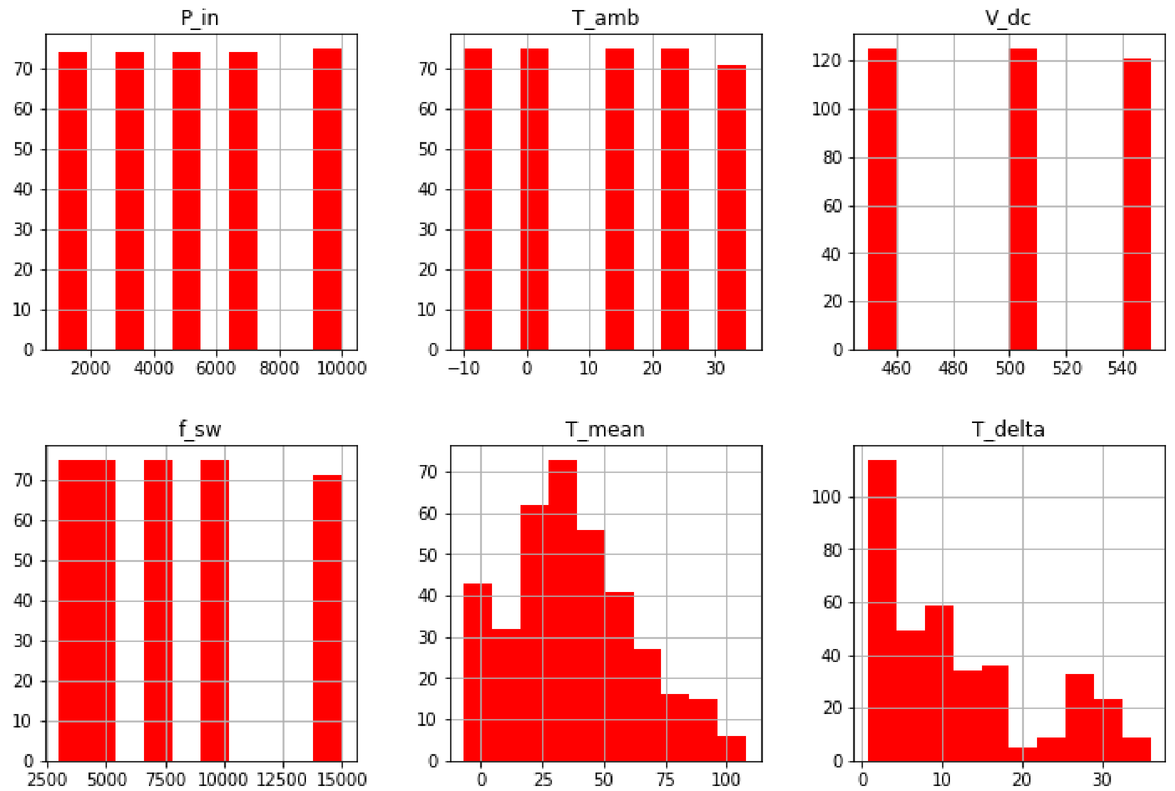
```
# Load df and drop rows with missing values
df = pd.read_excel(io.BytesIO(uploaded['data375.xlsx']))
```

```
df = pd.read_excel(io.BytesIO(uploaded['data375.xlsx']))
df = df.dropna()

# seperate response and predictors
x = df[['P_in','T_amb','V_dc','f_sw']]
y_mean = df['T_mean']
y_delta = df['T_delta']
print(x.shape)

# Normalize the data using Min-Max Scaling for feature range of [0,1]
x = minmax_scale(x, (0,1), axis=0)
#y_mean = minmax_scale(y_mean, (0,1), axis=0)
#y_delta = minmax_scale(y_delta, (0,1), axis=0)

x_train, x_test, y_mean_train, y_mean_test = train_test_split(x, y_mean, test_size=0.3)
x_train, x_test, y_delta_train, y_delta_test = train_test_split(x, y_mean, test_size=0.3)

model_output = {}
model_mse = []

print('Train Sets:', len(x_train))
print('Test Sets:', len(x_test))
```

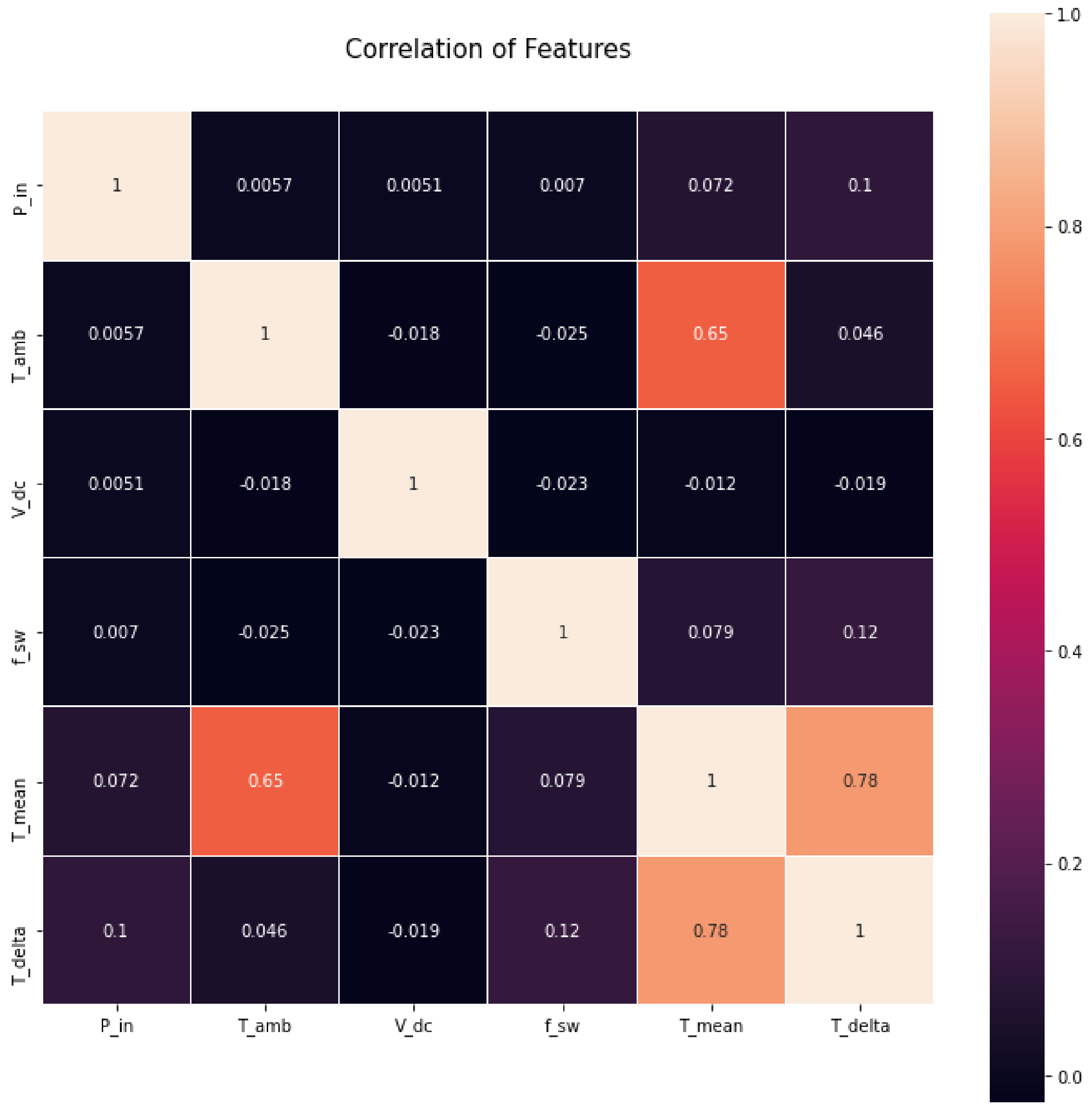
```
(371, 4)
Train Sets: 259
Test Sets: 112
```

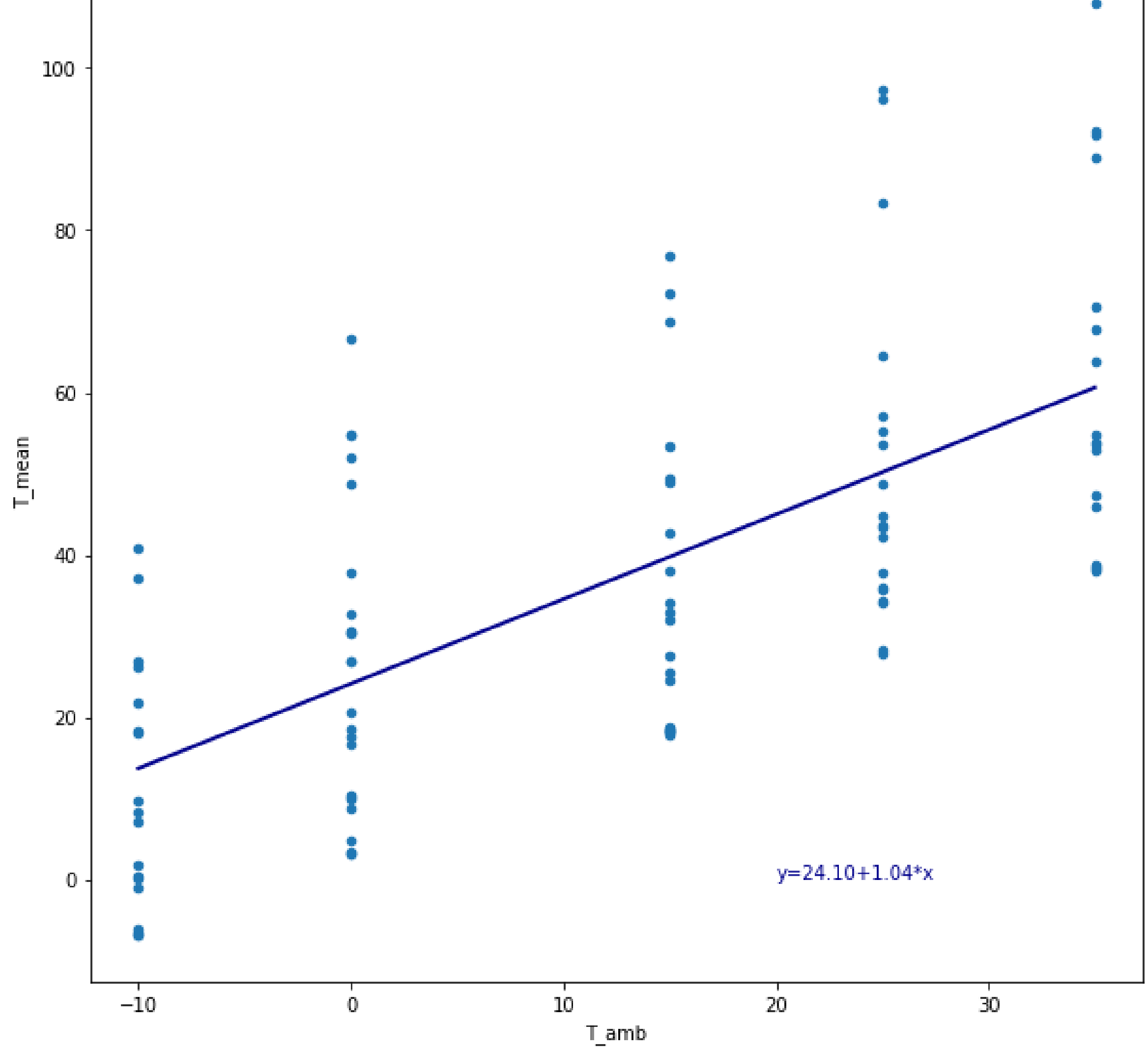
Describe the data

```
df.head()
df.describe()
# df.shape

df.hist(figsize=(12,8), color='red', layout=(2,3))
plt.show()

colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Correlation of Features', y=1.05, size=15)
sns.heatmap(df.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True,
            linecolor='white', annot=True)

fit = np.polyfit(df.T_amb, df.T_mean, 1)
df.sample(100).plot.scatter(x='T_amb', y='T_mean', figsize=(10,10))
plt.plot(df.T_amb, fit[0] * df.T_amb + fit[1], color='darkblue', linewidth=2)
plt.text(20,  0, 'y={:.2f}+{:.2f}*x'.format(fit[1], fit[0]), color='darkblue')
```



Text(20, 0, 'y=24.10+1.04*x')





Linear Regression

```
model_mean = LinearRegression()
model_delta = LinearRegression()

model_mean.fit(x_train,y_mean_train)
model_delta.fit(x_train,y_delta_train)

lin_reg_mean = model_mean
lin_reg_delta = model_delta

y_mean_pred = model_mean.predict(x_test)
y_delta_pred = model_delta.predict(x_test)

# Reporting
model_output["linear_regression_mean"] = y_mean_pred
model_output["linear_regression_delta"] = y_delta_pred
model_mse.append(['linear_regression',mean_squared_error(y_mean_test, y_mean_pred), mean_squared_error(

print('Mean squared error Mean Model: %.4f'
      % mean_squared_error(y_mean_test, y_mean_pred))
print('Mean squared error Delta Model: %.4f'
```

```
      % mean_squared_error(y_delta_test, y_delta_pred))
```

```
Mean squared error Mean Model: 595.7785
Mean squared error Delta Model: 421.7810
```

Bayesian Ridge

```
model_mean = BayesianRidge()
model_delta = BayesianRidge()

model_mean.fit(x_train,y_mean_train)
model_delta.fit(x_train,y_delta_train)

bayes_mean = model_mean
bayes_delta = model_delta

y_mean_pred = model_mean.predict(x_test)
y_delta_pred = model_delta.predict(x_test)

# Reporting
model_output["bayesian_ridge_mean"] = y_mean_pred
model_output["bayesian_ridge_delta"] = y_delta_pred
model_mse.append(['bayesian_ridge',mean_squared_error(y_mean_test, y_mean_pred), mean_squared_error(y_d

print('Mean squared error Mean Model: %.4f'
      % mean_squared_error(y_mean_test, y_mean_pred))
print('Mean squared error Delta Model: %.4f'
      % mean_squared_error(y_delta_test, y_delta_pred))
```

```
Mean squared error Mean Model: 595.2432
Mean squared error Delta Model: 420.7333
```

Decision Tree

```
model_mean = DecisionTreeRegressor()
model_delta = DecisionTreeRegressor()

model_mean.fit(x_train,y_mean_train)
model_delta.fit(x_train,y_delta_train)

dec_mean = model_mean
dec_delta = model_delta

y_mean_pred = model_mean.predict(x_test)
y_delta_pred = model_delta.predict(x_test)

# Reporting
model_output["decision_tree_mean"] = y_mean_pred
model_output["decision_tree_delta"] = y_delta_pred
model_mse.append(['decision_tree',mean_squared_error(y_mean_test, y_mean_pred), mean_squared_error(y_de
```

```
print('Mean squared error Mean Model: %.4f'
      % mean_squared_error(y_mean_test, y_mean_pred))
print('Mean squared error Delta Model: %.4f'
      % mean_squared_error(y_delta_test, y_delta_pred))

text_representation = sklearn.tree.export_text(model_mean)
print(text_representation)
```

```
Mean squared error Mean Model: 1506.5686
Mean squared error Delta Model: 1.7721
|--- feature_2 <= 0.25
|   |--- feature_3 <= 0.79
|   |   |--- feature_1 <= 0.11
|   |   |   |--- feature_0 <= 0.56
|   |   |   |   |--- feature_3 <= 0.46
|   |   |   |   |   |--- feature_3 <= 0.17
|   |   |   |   |   |   |--- feature_0 <= 0.33
|   |   |   |   |   |   |   |--- feature_0 <= 0.11
|   |   |   |   |   |   |   |   |--- value: [28.73]
|   |   |   |   |   |   |   |--- feature_0 >  0.11
|   |   |   |   |   |   |   |   |--- value: [24.19]
|   |   |   |   |   |   |--- feature_0 >  0.33
|   |   |   |   |   |   |   |--- value: [11.94]
|   |   |   |   |   |--- feature_3 >  0.17
|   |   |   |   |   |   |--- feature_0 <= 0.33
|   |   |   |   |   |   |   |--- value: [53.35]
|   |   |   |   |   |   |--- feature_0 >  0.33
|   |   |   |   |   |   |   |--- value: [25.96]
|   |   |   |   |--- feature_3 >  0.46
|   |   |   |   |   |--- feature_0 <= 0.11
|   |   |   |   |   |   |--- value: [31.99]
|   |   |   |   |   |--- feature_0 >  0.11
|   |   |   |   |   |   |--- feature_0 <= 0.33
|   |   |   |   |   |   |   |--- value: [-1.08]
|   |   |   |   |   |   |--- feature_0 >  0.33
|   |   |   |   |   |   |   |--- value: [28.46]
|   |   |   |--- feature_0 >  0.56
|   |   |   |   |--- feature_3 <= 0.08
|   |   |   |   |   |--- feature_0 <= 0.83
|   |   |   |   |   |   |--- value: [-0.97]
|   |   |   |   |   |--- feature_0 >  0.83
|   |   |   |   |   |   |--- value: [-6.44]
|   |   |   |   |--- feature_3 >  0.08
|   |   |   |   |   |--- feature_3 <= 0.37
|   |   |   |   |   |   |--- value: [30.53]
|   |   |   |   |   |--- feature_3 >  0.37
|   |   |   |   |   |   |--- feature_0 <= 0.83
|   |   |   |   |   |   |   |--- value: [-5.93]
|   |   |   |   |   |   |--- feature_0 >  0.83
|   |   |   |   |   |   |   |--- value: [9.48]
|   |   |--- feature_1 >  0.11
|   |   |   |--- feature_3 <= 0.08
|   |   |   |   |--- feature_1 <= 0.89
|   |   |   |   |   |--- feature_0 <= 0.83
|   |   |   |   |   |   |--- feature_0 <= 0.33
|   |   |   |   |   |   |   |--- feature_0 <= 0.11
|   |   |   |   |   |   |   |   |--- feature_1 <= 0.67
|   |   |   |   |   |   |   |   |   |--- value: [52.59]
```

```
 |   |   |   |   |   |   |   |   |--- feature_1 >  0.67
 |   |   |   |   |   |   |   |   |   |--- value: [37.17]
 |   |   |   |   |   |   |   |--- feature_0 >  0.11
 |   |   |   |   |   |   |   |   |--- value: [95.23]
 |   |   |   |   |   |   |--- feature_0 >  0.33
 |   |   |   |   |   |   |   |--- feature_1 <= 0.67
 |   |   |   |   |   |   |   |   |--- feature_0 <= 0.56
 |   |   |   |   |   |   |   |   |   |--- feature_1 <= 0.39
 |   |   |   |   |   |   |   |   |   |   |--- value: [59.79]
```

SVM

```
model_mean = SVR(kernel='rbf',C=100,gamma=0.1,epsilon=.1)
model_delta = SVR(kernel='rbf',C=100,gamma=0.1,epsilon=.1)

model_mean.fit(x_train,y_mean_train)
model_delta.fit(x_train,y_delta_train)

svm_mean = model_mean
svm_delta = model_delta

y_mean_pred = model_mean.predict(x_test)
y_delta_pred = model_delta.predict(x_test)

print(y_mean_pred)

# Reporting
model_output["svr_mean"] = y_mean_pred
model_output["svr_delta"] = y_delta_pred
model_mse.append(['svr',mean_squared_error(y_mean_test, y_mean_pred), mean_squared_error(y_delta_test,


print('Mean squared error Mean Model: %.4f'
      % mean_squared_error(y_mean_test, y_mean_pred))
print('Mean squared error Delta Model: %.4f'
      % mean_squared_error(y_delta_test, y_delta_pred))
```

```
[26.36741512 30.78771738 36.65257666 41.80126388 32.59734411 33.71165112
 33.50079289 30.30903763 34.38739509 34.01381374 30.34647726 31.63786159
 34.98554893 40.76163085 35.06284409 30.51380244 33.12193409 34.93291445
 37.57329787 37.8006809  34.4863157  35.90851485 35.15652808 30.87498987
 32.13787991 34.76960067 28.92407615 40.71802028 27.33298483 28.94667409
 32.5603031  32.29839809 39.53361687 38.92508448 31.56083869 33.40722047
 37.81802344 32.21865758 29.39046223 31.81967209 35.79377266 34.05849725
 37.41019257 37.20404906 33.86625914 36.8736635  34.27642099 40.53135273
 38.75526153 32.59933325 37.90965242 35.15812523 35.22153993 33.4433907
 37.14365787 36.24456981 41.54339972 41.28655497 39.45984593 37.51314107
 40.25014124 39.33486437 36.68007421 32.43867396 32.73551382 36.11046867
 38.91964907 35.53714249 31.66392731 38.53044451 41.2490046  37.57964378
 41.30536884 36.63173571 35.98248564 34.17409022 37.91876768 37.85202437
 39.24935492 37.94274382 34.46730871 38.44518704 34.01623241 30.88975843
 29.53075001 29.49957384 35.18032253 38.9079222  32.07937258 36.59697823
 36.0982563  38.15048171 32.56587099 30.95494515 36.79822509 35.13232357
 40.98541204 35.20901357 40.964123   43.38193788 34.28399497 36.02553733
 46.50037363 34.25146778 37.15322434 30.33810257 38.46580994 36.82187021
 34.1236785  37.89566016 31.32376336 34.12284002]
```

```
Mean squared error Mean Model: 549.9495
Mean squared error Delta Model: 271.1750
```

Neural Network

```
model_mean = MLPRegressor(hidden_layer_sizes=(4, 5, 3, 1), activation='tanh',  solver='lbfgs')
model_delta = MLPRegressor(hidden_layer_sizes=(4, 5, 3, 1), activation='tanh',  solver='lbfgs')

model_mean.fit(x_train, y_mean_train)
model_delta.fit(x_train, y_delta_train)

dnn_mean = model_mean
dnn_delta = model_delta

y_mean_pred = model_mean.predict(x_test)
y_delta_pred = model_delta.predict(x_test)

print(y_mean_pred)
print(model_mean.score(x_test,y_mean_test))
# Reporting
model_output["dnn_mean"] = y_mean_pred
model_output["dnn_delta"] = y_delta_pred
model_mse.append(['dnn',mean_squared_error(y_mean_test, y_mean_pred), mean_squared_error(y_delta_test,

print('Mean squared error Mean Model: %.4f'
      % mean_squared_error(y_mean_test, y_mean_pred))
print('Mean squared error Delta Model: %.4f'
      % mean_squared_error(y_delta_test, y_delta_pred))
```

```
[38.08928203 38.08928165 38.08928113 38.08927691 38.08928205 38.0892819
 38.08928136 38.08928204 38.08928125 38.08928146 38.08928182 38.08928103
 38.08928102 38.08927744 38.08928107 38.08928192 38.08928203 38.08928183
 38.0892803  38.08928002 38.08928158 38.08927882 38.08928018 38.08928145
 38.08928173 38.08928017 38.08928171 38.08927894 38.08928205 38.08928157
 38.08928175 38.08928198 38.08927399 38.08928011 38.0892818  38.0892811
 38.08928158 38.08928144 38.08928206 38.08928155 38.08928035 38.08928004
 38.08928091 38.08928144 38.08928018 38.0892806  38.08928184 38.08927742
 38.08928106 38.08928158 38.08927952 38.08928176 38.0892807  38.08928159
 38.08928143 38.08928002 38.08927662 38.08928    38.08927946 38.08928155
 38.08927595 38.0892786  38.08928117 38.08928127 38.08928175 38.0892812
 38.08927823 38.08928017 38.089282   38.08927852 38.08927502 38.08928017
 38.08927364 38.08928048 38.08928143 38.08928064 38.08928007 38.0892805
 38.08927936 38.08928047 38.08928182 38.08927948 38.08928204 38.08928169
 38.08928199 38.08928202 38.08928184 38.08927488 38.08928184 38.0892791
 38.08928169 38.08927931 38.08928149 38.08928198 38.08927911 38.08928117
 38.08927757 38.08928018 38.08927751 38.08927482 38.08928199 38.08928061
 38.08927086 38.08928187 38.08927985 38.08928185 38.08928065 38.0892809
 38.08928173 38.08928055 38.08928187 38.08928187]
-0.008818567937652455
Mean squared error Mean Model: 580.8759
Mean squared error Delta Model: 2.1110
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:470: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

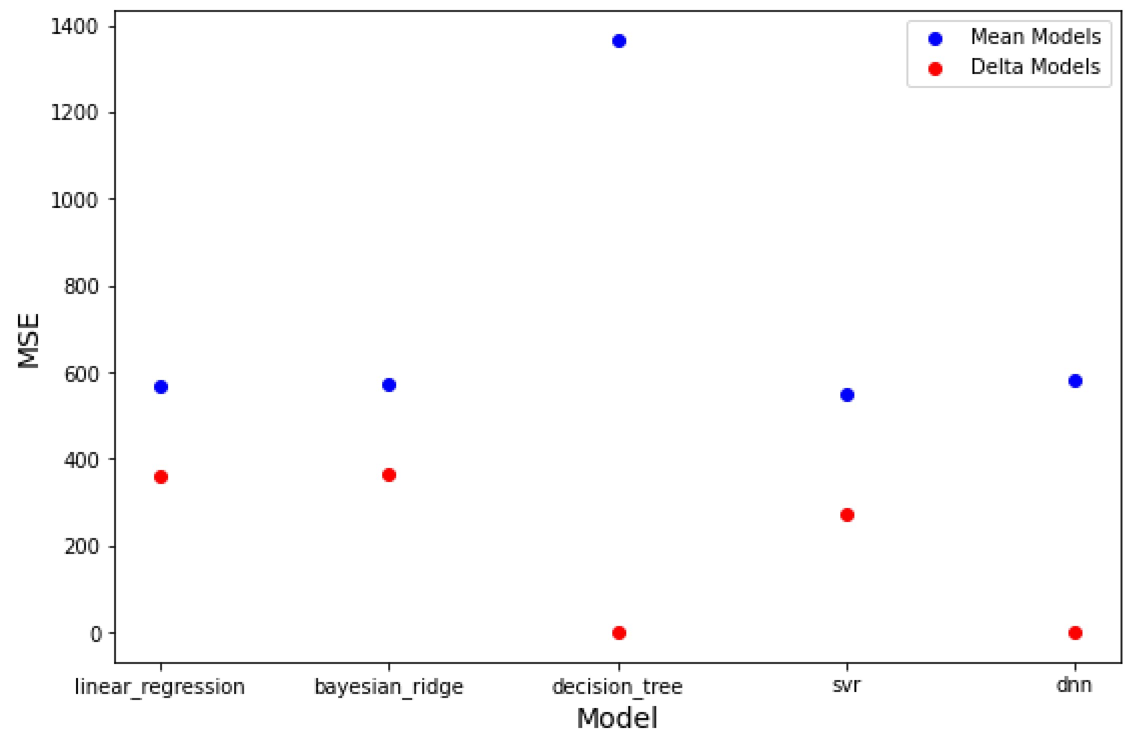
```
https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

Data Post Processing

```
model_mse = np.array(model_mse).T
```

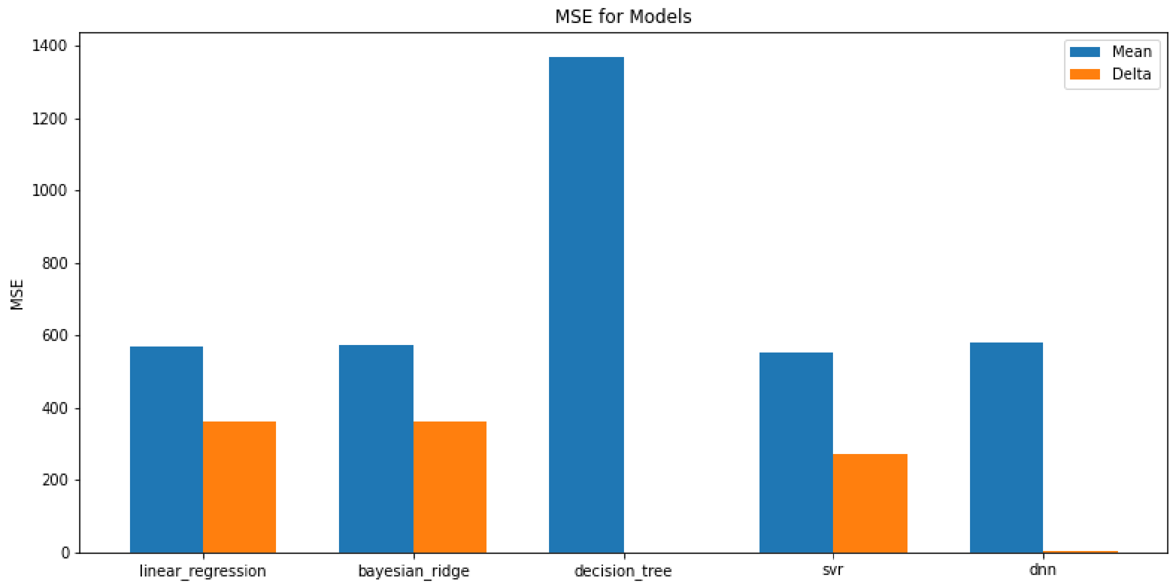
```
plot_data = pd.DataFrame(model_mse[1:3], columns=model_mse[0]).astype(float)
print(plot_data)
fig, ax = plt.subplots(figsize=(9, 6))  # a figure with a single Axes
ax.set_xlabel('Model', fontdict={"size": 14})  # Add an x-label to the axes.
ax.set_ylabel('MSE', fontdict={"size": 14})  # Add a y-label to the axes.
plt.xticks(range(1,6), list(plot_data.columns))
ax.scatter(x=range(1,6), y=plot_data.loc[0], c="blue")
ax.scatter(x=range(1,6), y=plot_data.loc[1], c="red")
ax.legend(["Mean Models", "Delta Models"])
```

```
   linear_regression  bayesian_ridge  decision_tree         svr         dnn
0         569.258095      572.695857    1368.111953  549.949523  580.875881
1         362.793881      363.801384       0.575493  271.175000    2.110970
<matplotlib.legend.Legend at 0x7fc1e3ef2d30>
```




```
x = np.arange(len(model_mse[0]))
width = 0.35

labels = model_mse[0]
means = model_mse[1].astype(np.float)
deltas = model_mse[2].astype(np.float)

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, means, width, label='Mean')
```

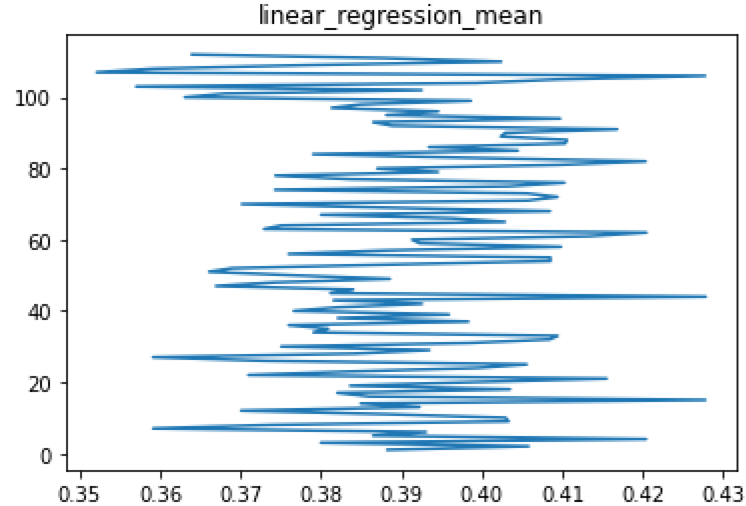
```
rects2 = ax.bar(x + width/2, deltas, width, label='Delta')
ax.set_xticks(np.arange(len(x)))
ax.set_xticklabels(labels)
ax.legend()
plt.title("MSE for Models")
ax.set_ylabel('MSE')  # Add a y-label to the axes.
fig.tight_layout()
fig.set_size_inches(12, 6)
```



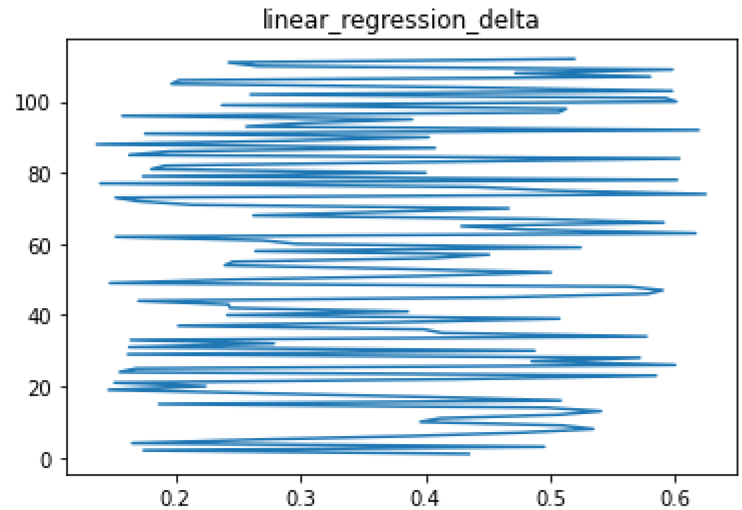
Rainflow-Counting Algorithm

```
# Print keys of stored output data
# Access values for array of model output values
for key, values in model_output.items():
  print(key)
  plt.title(key)
  plt.plot(values, range(1,len(values)+1))
  print(rainflow.count_cycles(values))
  plt.show()
```

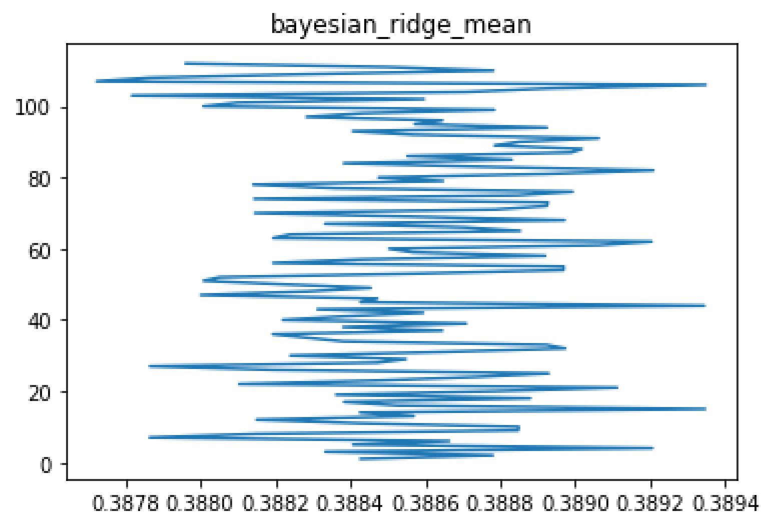
linear_regression_mean
[(0.001826989518227351, 1.0), (0.0028451529199547965, 1.0), (0.006479478628486246, 1.0), (0.00659



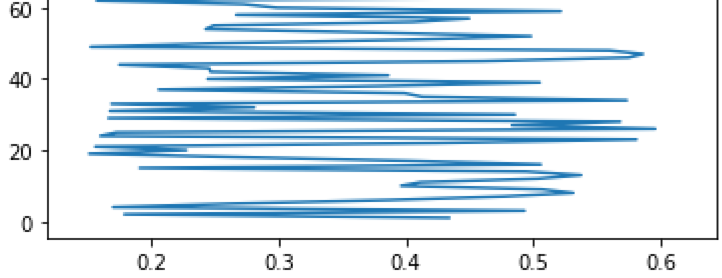
linear_regression_delta
[(0.07295063044104616, 1.0), (0.08646834683697907, 1.0), (0.10795810808486983, 1.0), (0.114597813



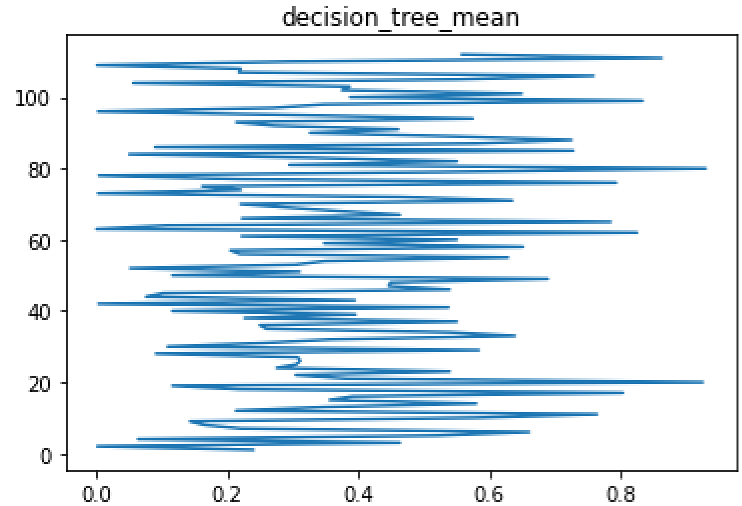
bayesian_ridge_mean
[(4.490728447342773e-05, 1.0), (7.373886628309068e-05, 1.0), (0.00014355693540213377, 1.0), (0.00



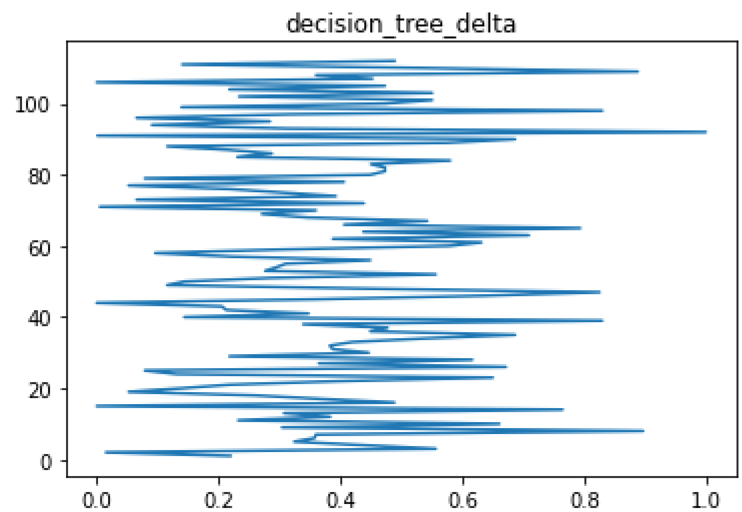
bayesian_ridge_delta
[(0.07163294571620882, 1.0), (0.08519424747274978, 1.0), (0.10591064817823614, 1.0), (0.112333035



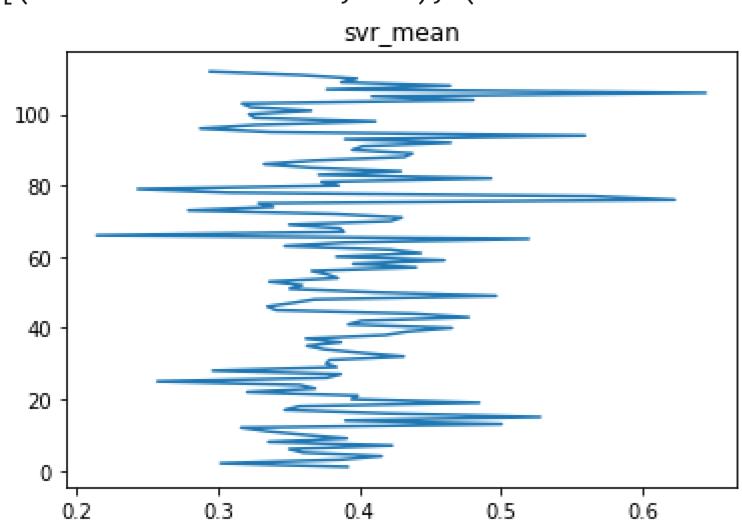
decision_tree_mean
[(0.0024438254870472464, 1.0), (0.011681257291391745, 1.0), (0.03592636757465978, 1.0), (0.059261



decision_tree_delta
[(0.023131142317646514, 1.0), (0.028356084396850056, 1.0), (0.056521383945986936, 1.0), (0.063726

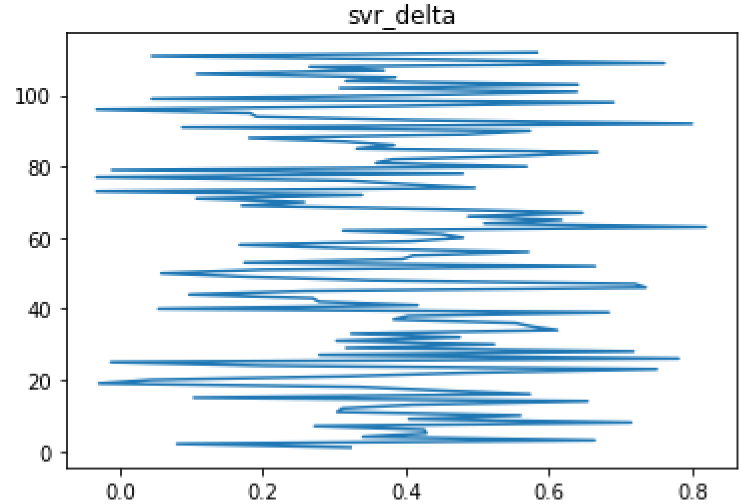


svr_mean
[(0.004346866716090858, 1.0), (0.007549331712922935, 1.0), (0.008882035503035013, 1.0), (0.010497

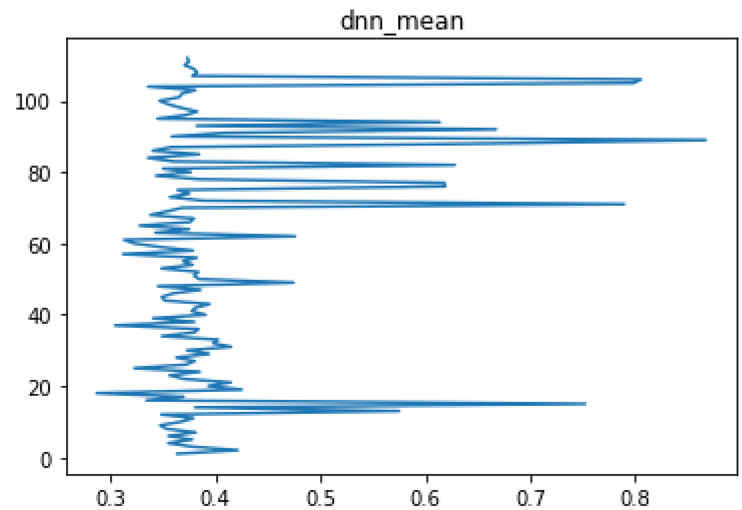


svr_delta

```
[(0.05215104728945619, 1.0), (0.06916119788247954, 1.0), (0.08745238498742935, 1.0), (0.088588893
```




```
dnn_mean
[(0.001471141922214425, 0.5), (0.004194558688744965, 1.0), (0.004230249961911 07, 0.5), (0.0046985
```




```
dnn_delta
[(0.06242893840703195, 1.0), (0.06998203129479752, 1.0), (0.07394942487702444, 1.0), (0.084227807
```

Import Temp/Irradiance Data

```
uploaded_aalborg_solaryear = files.upload()
uploaded_aalborg_ambienttemp = files.upload()
aalborg_solaryear = pd.read_csv(io.BytesIO(uploaded_aalborg_solaryear['aalborg_solaryear.csv']), encod
aalborg_ambienttemp = pd.read_csv(io.BytesIO(uploaded_aalborg_ambienttemp['aalborg_ambienttemp.csv']),
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

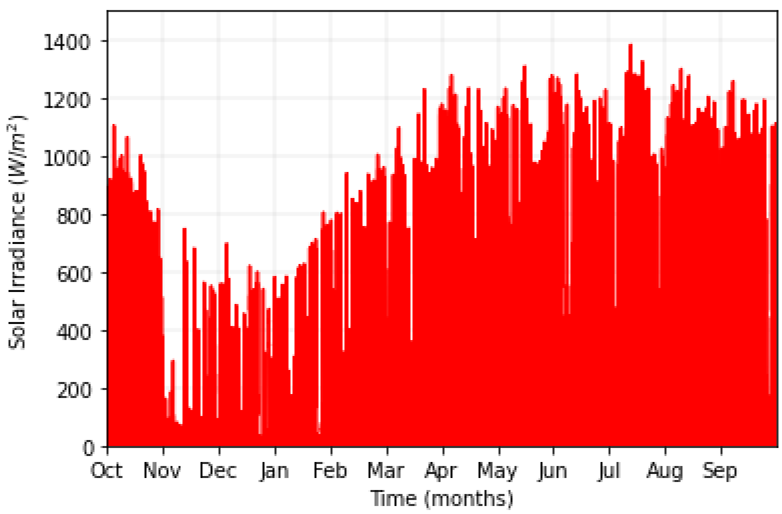
```
Saving aalborg_solaryear.csv to aalborg_solaryear.csv
```

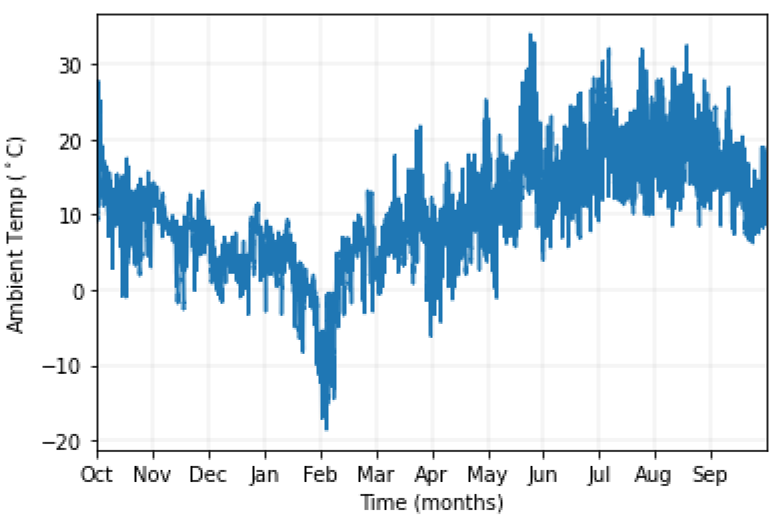
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
labels = ['Oct', 'Nov', 'Dec', 'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep']
loc = [0, 9175.1, 18350, 27525,36700, 45875, 55051, 64226, 73401, 82576, 91751, 100930]
```

```
ax = aalborg_solaryear.plot(c="red")
ax.get_legend().remove()
plt.grid(which='major',axis='both',linewidth=0.25)
plt.ylabel('Solar Irradiance ($W/m^2$)')
plt.xticks(loc,labels)
plt.xlabel('Time (months)')
plt.xlim(0,110101)
plt.ylim(0,1500)
ax = aalborg_ambienttemp.plot()
plt.grid(which='major',axis='both', linewidth=0.25)
plt.xticks(loc,labels)
plt.xlim(0,110101)
plt.xlabel('Time (months)')
ax.get_legend().remove()
plt.ylabel('Ambient Temp ($^\circ$C)')
```

```
Text(0, 0.5, 'Ambient Temp ($^\\circ$C)')
```






```
P_in = (aalborg_solaryear['Solar Irradiance'])
T_amb = aalborg_ambienttemp['Ambient Temp']

P_in = minmax_scale(P_in, (0,1), axis=0)
T_amb = minmax_scale(T_amb,(0,1), axis=0)

V_dc = np.ones((110101,))     # gives 1 for the normalized voltages
f_sw = np.ones((110101,))     # gives 1 for normalized switching frequencies
```

```
x_final = pd.DataFrame(data=[P_in,T_amb,V_dc,f_sw]).T

T_mean_pred = dnn_mean.predict(x_final)
T_delta_pred = dnn_delta.predict(x_final)

mean_cycles = rainflow.count_cycles(T_mean_pred)
delta_cycles = 50*60*5*len(P_in)

cycle_total = 0
for i in range(len(mean_cycles)):
  cycle_total += mean_cycles[i][1]

print(cycle_total)
```

```
19156.5
```

```
# Linear Regression Cycle Counting
T_mean_pred = lin_reg_mean.predict(x_final)

mean_cycles = rainflow.count_cycles(T_mean_pred)

l_cycle_total = 0
for i in range(len(mean_cycles)):
  l_cycle_total += mean_cycles[i][1]

print(l_cycle_total)
```

```
19964.5
```

```
# Bayesian Ridge Cycle Counting
T_mean_pred = bayes_mean.predict(x_final)

mean_cycles = rainflow.count_cycles(T_mean_pred)

b_cycle_total = 0
for i in range(len(mean_cycles)):
  b_cycle_total += mean_cycles[i][1]

print(b_cycle_total)
```

```
20041.5
```

```
# Decision Tree Cycle Counting
T_mean_pred = dec_mean.predict(x_final)

mean_cycles = rainflow.count_cycles(T_mean_pred)

d_cycle_total = 0
for i in range(len(mean_cycles)):
  d_cycle_total += mean_cycles[i][1]

print(d_cycle_total)
```

```
3639.0
```

```python
# SVM Cycle Counting
T_mean_pred = svm_mean.predict(x_final)

mean_cycles = rainflow.count_cycles(T_mean_pred)

s_cycle_total = 0
for i in range(len(mean_cycles)):
  s_cycle_total += mean_cycles[i][1]

print(s_cycle_total)
```

```
18552.5
```

```python
# Neural Net Cycle Counting
T_mean_pred = dnn_mean.predict(x_final)

mean_cycles = rainflow.count_cycles(T_mean_pred)

n_cycle_total = 0
for i in range(len(mean_cycles)):
  n_cycle_total += mean_cycles[i][1]

print(n_cycle_total)
```

```
19156.5
```