Internet of Things (IoT): Final Project
Plotting and Analysis

Name: Ajay Choudhury **Roll No.: 18018** **Date: 14th November 2021**

Component 1: Plotting

Here the **weather_data_2sites** dataset is used to plot the following five plots using Pandas in Python:

Plot 1:

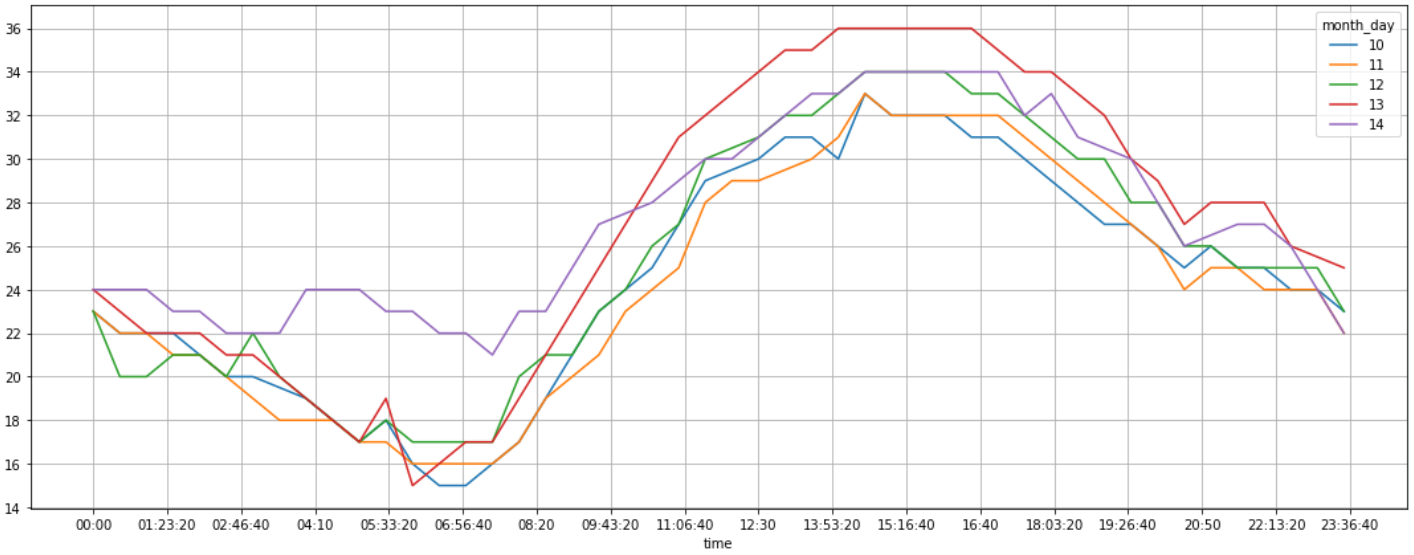
This plot shows the change of temperature at different times of a day across 5 different days. Each day's data is depicted with a different colour.

Script:

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

# my roll no.: 18018
# starting date = (1+8+1) = 10 to 14 (5 days)
# chosen month = March(3)
df1 = df1.set_index(['timestamp'])
df1 = df1.loc['2018-3-10':'2018-3-14']

df=pd.read_csv("weather_data_2sites-1.csv") # read the .csv file
df['timestamp'] = pd.to_datetime(df['timestamp'])

df.sort_values(by='timestamp', inplace=True)
df1 = df.copy()
df1 = df1.dropna(axis = 0, how ='any')

# my roll no.: 18018
# starting date = (1+8+1) = 10 to 14 (5 days)
# chosen month = March(3)
df1 = df1.set_index(['timestamp'])
df1 = df1.loc['2018-3-10':'2018-3-14']

df1.drop(['temperature_site2', 'humidity_site2'], axis=1, inplace= True)
df1.drop(df1.columns[df1.columns.str.contains('unnamed',case =
False)],axis = 1, inplace = True)
```

```
df2 = df1.copy()
df2.reset_index(inplace = True)
df2['time'] = df2['timestamp'].dt.time
df2['month_day'] = df2['timestamp'].dt.day

df2.drop(['timestamp'], axis=1, inplace= True)

plt.rcParams["figure.figsize"] = [15.00, 6.00]

(df2.pivot(index='time', columns='month_day',
values='temperature_site1')).plot()

plt.locator_params(axis='x', nbins=24)
plt.locator_params(axis='y', nbins=15)
plt.grid()
plt.show()
```

Output:



Plot 2:

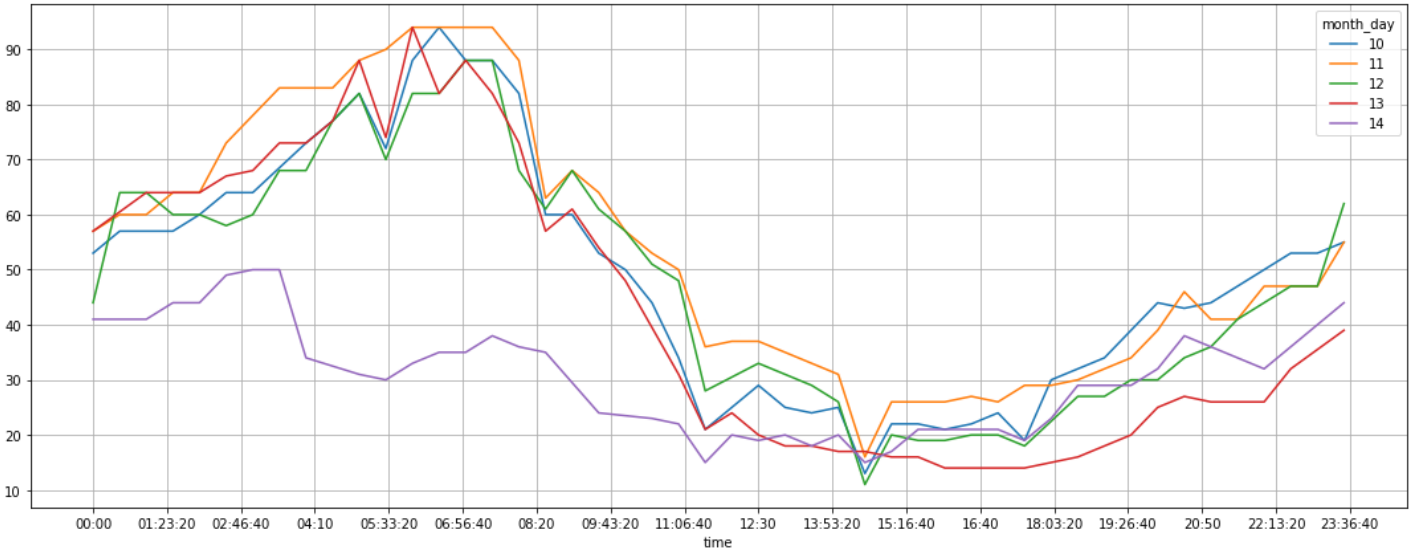
This plot shows the change of temperature at different times of a day across 5 different days. Each day's data is depicted with a different colour.

Script:

```
plt.rcParams["figure.figsize"] = [15.00, 6.00]

(df2.pivot(index='time', columns='month_day',
values='humidity_site1')).plot()
plt.locator_params(axis='x', nbins=24)
```

```
plt.locator_params(axis='y', nbins=15)
plt.grid()
plt.show()
```

Output:



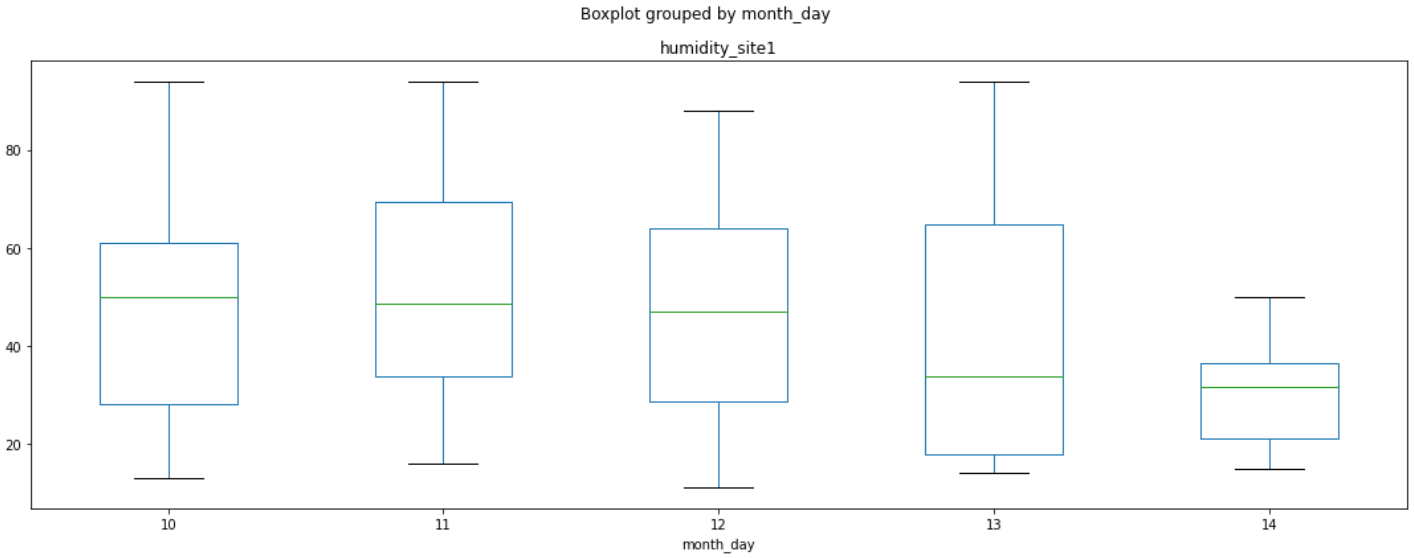
Plot 3:

This is a box plot for the humidity of each day in the 5 selected days.

Script:

```
df2.boxplot(by ='month_day', column =['humidity_site1'], grid = False)
plt.show()
```

Output:



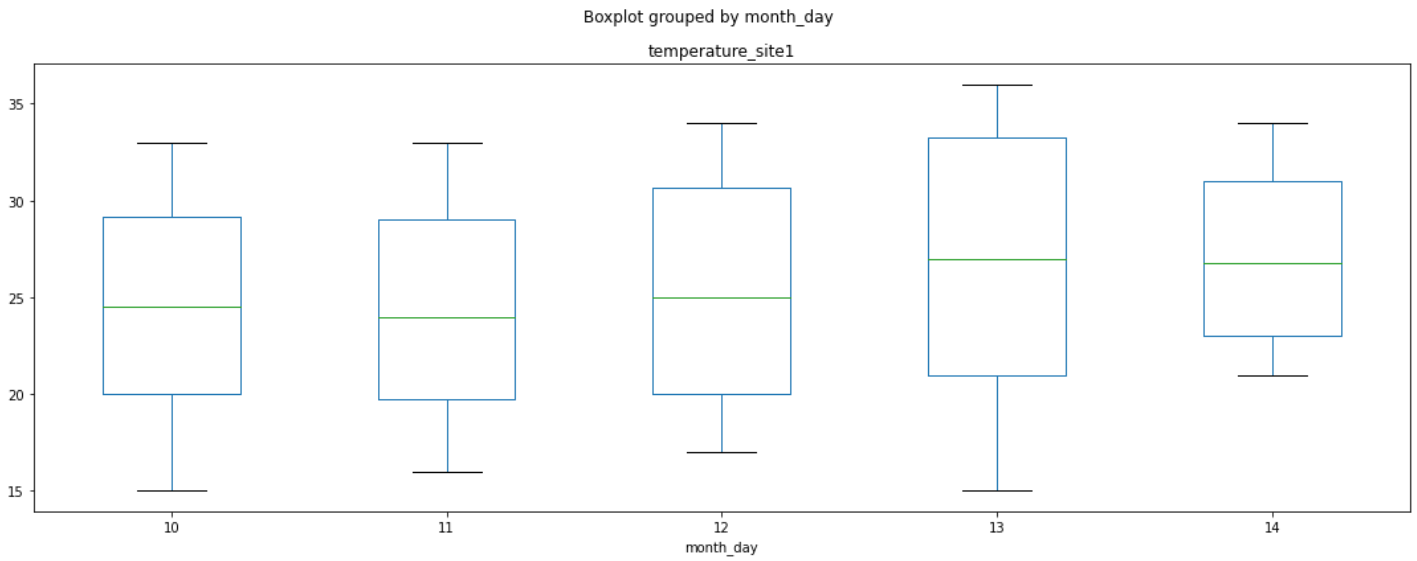
Plot 4:

This is a box plot for the temperature of each day in the 5 selected days.

Script:

```
df2.boxplot(by ='month_day', column =['temperature_site1'], grid =
False)
plt.show()
```

Output:



Plot 5:

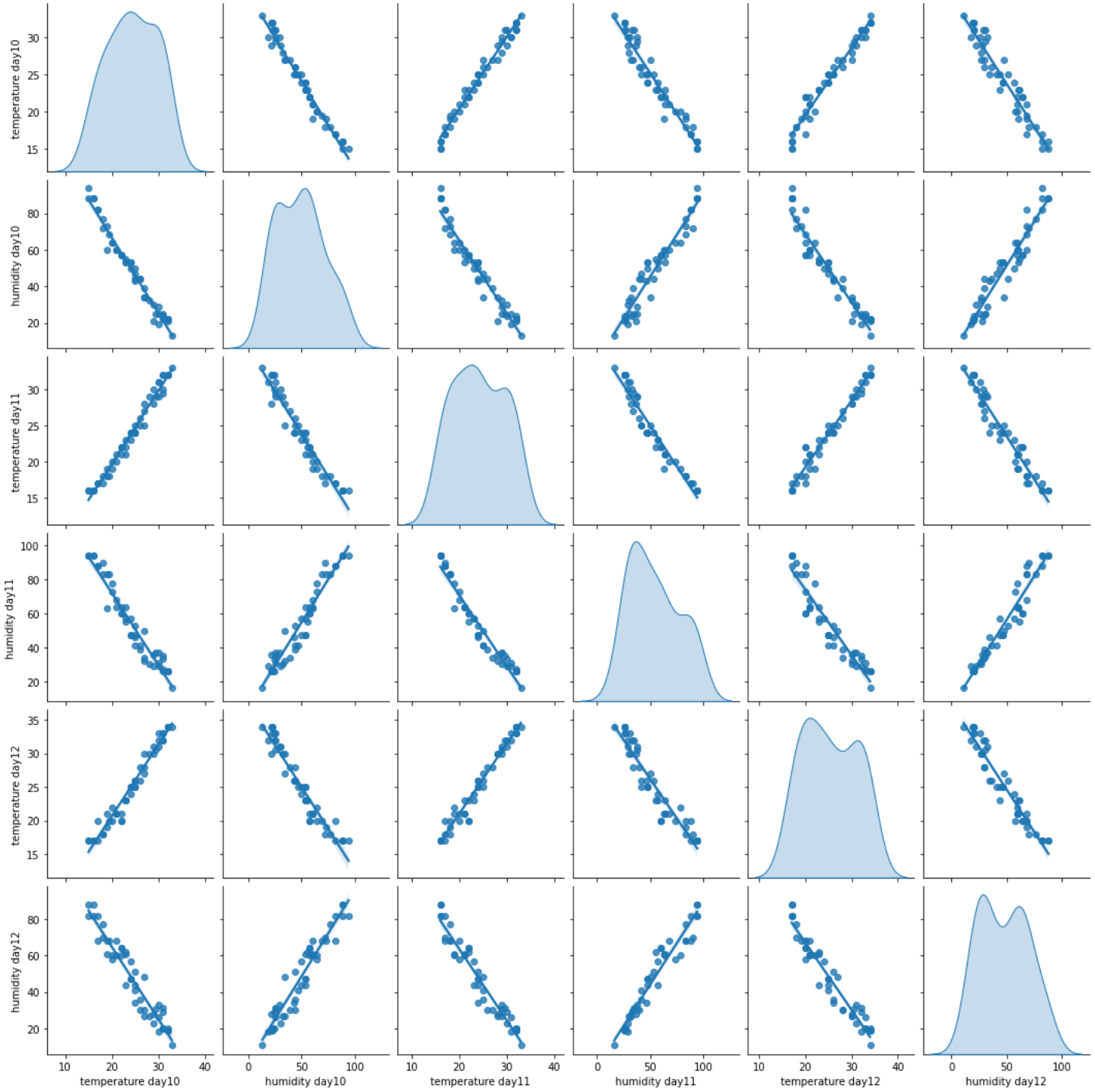
This is a grid of plots amongst each day's temperature and humidity.

Script:

```
df3 = df2.copy()
df3 = df3.set_index(['month_day'])
df3 = df3.loc['10':'12']

df3.reset_index(inplace = True)

df3.loc[df3['month_day'] == 10, 'temperature day10'] =
df3['temperature_site1']
df3.loc[df3['month_day'] == 10, 'humidity day10'] =
df3['humidity_site1']
df3.loc[df3['month_day'] == 11, 'temperature day11'] =
df3['temperature_site1']
df3.loc[df3['month_day'] == 11, 'humidity day11'] =
df3['humidity_site1']
df3.loc[df3['month_day'] == 12, 'temperature day12'] =
df3['temperature_site1']
```

```
df3.loc[df3['month_day'] == 12, 'humidity day12'] =
df3['humidity_site1']

df3.drop(['month_day', 'temperature_site1', 'humidity_site1', 'time'],
axis=1, inplace= True)

def justify(a, invalid_val=0, axis=1, side='left'):
    if invalid_val is np.nan:
        mask = ~np.isnan(a)
    else:
        mask = a!=invalid_val
    justified_mask = np.sort(mask,axis=axis)
    if (side=='up') | (side=='left'):
        justified_mask = np.flip(justified_mask,axis=axis)
    out = np.full(a.shape, invalid_val)
    if axis==1:
        out[justified_mask] = a[mask]
    else:
        out.T[justified_mask.T] = a.T[mask.T]
    return out

arr = justify(df3.to_numpy(), invalid_val=np.NaN,axis=0)

df3 = pd.DataFrame(arr, columns=df3.columns, index=df3.index)
df3.dropna(how='all', inplace=True)

import seaborn as sns
sns.pairplot(df3, kind="reg", diag_kind="kde")
```

Output:



Component 2: Analysis

Regression 1: RMSE = 0.9754396323594324

Script:

```
df.dropna(axis=0, inplace=True)

df.drop(df.columns[df.columns.str.contains('unnamed',case = False)],axis
= 1, inplace = True)
df['timestamp'] =  pd.to_datetime(df['timestamp'])
```

```
df['temperature'] = df['temperature_site1']
df['humidity'] = df['humidity_site1']
df['day-minutes'] = df['timestamp'].dt.minute
df['day-of-the-week'] = df['timestamp'].dt.dayofweek

df.loc[0, 'previous-temperature'] = 0
for i in range(1, len(df)):
    df.loc[i, 'previous-temperature'] = df.loc[i-1, 'temperature']

df.drop(['timestamp','temperature_site1', 'humidity_site1',
'temperature_site2', 'humidity_site2'], axis=1, inplace= True)

df['previous-temperature'] = df['previous-temperature'].fillna(0)

syn_weather = df.copy()
syn_weather.dropna(inplace=True)

from sklearn.model_selection import train_test_split
train_set, test_set  = train_test_split(syn_weather, test_size=0.2,
random_state=42)
print(f"Rows in train set: {len(train_set)}\nRows in test set:
{len(test_set)}\n")

trainer = train_set.drop("temperature", axis=1)
trainer_labels = train_set["temperature"].copy()

from sklearn.linear_model import LinearRegression
M1 = LinearRegression()
M1.fit(trainer, trainer_labels)

some_data = trainer.iloc[:5]
some_labels = trainer_labels.iloc[:5]

M1.predict(some_data)

from sklearn.metrics import mean_squared_error
predictions = M1.predict(trainer)
mse = mean_squared_error(trainer_labels, predictions)
rmse = np.sqrt(mse)
```

Regression 2: Mean RMSE = 1.2500364748789576

Script:

```
from sklearn.tree import DecisionTreeRegressor
M2 = DecisionTreeRegressor()
M2.fit(trainer, trainer_labels)

M2.predict(some_data)

# 10-fold
from sklearn.model_selection import cross_val_score
values = cross_val_score(M2, trainer, trainer_labels,
scoring="neg_mean_squared_error", cv=10)
rmse_values = np.sqrt(-values)

def print_values(values):
    print("Values:", values)
    print("RMSE Mean: ", values.mean())
    print("Standard deviation: ", values.std())

print_values(rmse_values)
```

Correlations:

Script:

```
# correlations
corr_matrix = syn_weather.corr()
corr_matrix['temperature'].sort_values(ascending=False)
```

Explanation: More extreme the correlation value (more positive or more negative) more it affects the predicted value.

Final Plot:

```
# on test dataset
X_test = test_set.drop("temperature", axis=1)
Y_test = test_set["temperature"].copy()

M1_predictions = M1.predict(X_test)
M2_predictions = M2.predict(X_test)

M1_mse = mean_squared_error(Y_test, M1_predictions)
M2_mse = mean_squared_error(Y_test, M2_predictions)
```

```
M1_rmse = np.sqrt(M1_mse)
M2_rmse = np.sqrt(M2_mse)
# print(final_predictions, list(Y_test))

new_df = pd.DataFrame()
new_df['M1'] = M1_predictions
new_df['M2'] = M2_predictions
new_df['Actual'] = list(Y_test)

x1 = new_df['M1']
x2 = new_df['M2']
y = new_df['Actual']

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Model Predictions vs Actual temperature')
ax1.scatter(x1,y)
ax2.scatter(x2,y)

ax1.set(xlabel='M1 Predicted Temperature', ylabel='Actual temperature')
ax2.set(xlabel='M2 Predicted Temperature', ylabel='Actual temperature')
```