Importing required libraries

In [ ]:
```
import pandas as pd
import numpy as np


from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN,LSTM
from keras.layers import Dropout
from keras.regularizers import L1L2

import matplotlib.pyplot as plt



import warnings
warnings.filterwarnings("ignore")
```

Paths for data import and result storage

In [ ]:
```
dpath = r"/Users/bashirulazambiswas/Documents/Sadia/SUNY courses/2022_
path = r"/Users/bashirulazambiswas/Documents/Sadia/SUNY courses/2022_F
```

Specification Selection

In [ ]:
```
station_name = 'QUEE'
target_variable = 'temperature'
window = 5
train_method = 'prediction with target only'
model = "AR"
```

Data preparation function

In [ ]:
```
def data_prep(path, station_name,target_variable,train_method):
    data = deep_learning_data(path,station_name,target_variable,train_
    data.time = time_coversion(data.time)
    return data

def deep_learning_data(path,station_name,dependent_variable_label,trai

    data_preprocessed = preprocess(path,station_name,dependent_variabl
    if dependent_variable_label == 'temperature':
        dependent_variable = 'temp_2m_avg [degF]'
    elif dependent_variable_label == 'humidity':
        dependent_variable = 'relative_humidity_avg [percent]'
    elif dependent_variable_label == 'precipitation':
        dependent_variable = 'precip_total [inch]'
    if train_method == 'prediction with target only':
        data = data_preprocessed[0][['station','time',dependent_variab
    elif train_method == 'prediction with whole dataset':
        data = data_preprocessed[0]
    return data

def time_coversion(time_col): #Converts time from string to datetime
    time_list = [time_col[i].split()[0] for i in range(len(time_col))]
    return pd.to_datetime(time_list)
```

Preprocessing function

In [ ]:
```
def preprocess(path,station_name,dependent_variable_label):
    data = pd.read_csv(path+"/"+station_name+".csv").drop("Unnamed: 0"
    correlation = data.corr().round(2)
    index = correlation.index
    for i in correlation.index:
        if i in index:
            temp = correlation.loc[i]
            temp = temp[temp==1]
            for j in temp.index:
                if j != i:
                    if j not in index:
                        continue
                    index = index.drop(j)

    preprocessed_data = pd.DataFrame()
    preprocessed_data['station'] = data['station']
    preprocessed_data['time'] = data['time']
    preprocessed_data[index] = data[index]

    if dependent_variable_label == 'temperature':
        dependent_variable = 'temp_2m_avg [degF]'
        independent_feature_label = preprocessed_data.columns.drop(['t
    elif dependent_variable_label == 'humidity':
        dependent_variable = 'relative_humidity_avg [percent]'
        independent_feature_label = preprocessed_data.columns.drop(['r
    elif dependent_variable_label == 'precipitation':
        dependent_variable = 'precip_total [inch]'
        independent_feature_label = preprocessed_data.columns.drop(['p
    x = preprocessed_data[independent_feature_label]
    y = preprocessed_data[dependent_variable]

    return preprocessed_data,x,y
```

Data generated after preprocessing

In [ ]:
```
data = data_prep(dpath, station_name,target_variable,train_method)
data.head(5)
```

Train-test split function

In [ ]:
```
def normalize_data(df):
    scaler = StandardScaler()
    df_ = df.copy()
    for col in df.columns:
        if col == 'station':
            continue
```

```
            continue
        elif col == 'time':
            continue
        scaler = scaler.fit(df_[col].values.reshape(-1,1))
        df_[col] = scaler.transform(df_[col].values.reshape(-1,1))
    return df_

def load_data(data_normalized, window,target):
    train_year = [2018,2019,2020]
    test_year = [2021,2022]
    if target == 'temperature':
        x = data_normalized.drop(['temp_2m_avg [degF]'],axis=1)
        y = data_normalized['temp_2m_avg [degF]']
    elif target == 'humidity':
        x = data_normalized.drop('relative_humidity_avg [percent]',axi
        y = data_normalized['relative_humidity_avg [percent]']
    elif target == 'precipitation':
        x = data_normalized.drop('precip_total [inch]',axis=1)
        y = data_normalized['precip_total [inch]']

    data = pd.concat([x,y],axis=1)
    data = data.dropna(how='any',axis=0)
    sequence_length = window + 1

    train = pd.DataFrame()
    for year in train_year:
         train = train.append(data[(data['time'].dt.year == year)])
    train = train.drop(['station','time'],axis=1)
    amount_of_features = len(train.columns)
    train = train.values
    train_result = []
    for index in range(len(train) - sequence_length):
        train_result.append(train[index: index + sequence_length])
    train_result = np.array(train_result)
    x_train = train_result[:, :-1]
    y_train = train_result[:, -1][:,-1]
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],

    test = pd.DataFrame()
    for year in test_year:
         test = test.append(data[(data['time'].dt.year == year)])
    test = test.drop(['station','time'],axis=1)
    test = test.values
    test_result = []
    for index in range(len(test) - sequence_length):
        test_result.append(test[index: index + sequence_length])
    test_result = np.array(test_result)
    x_test = test_result[:, :-1]
    y_test = test_result[:, -1][:,-1]

    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amo
```

```
    return [x_train, y_train, x_test, y_test,amount_of_features]
```

Train and test set after splitting

In [ ]:
```
data_normalized = normalize_data(data)
X_train, y_train, X_test, y_test,amount_of_features = load_data(data_n

print("Train:",train.shape)
print("Test:",test.shape)
```

Building LSTM model and predicting

In [ ]:
```
def build_model(layers, model, target_variable):
    model = Sequential()
    model.add(LSTM(64,activation = 'relu', return_sequences=True, inpu
    model.add(LSTM(32,activation = 'relu', return_sequences=True))
    model.add(LSTM(32,activation = 'relu', return_sequences=False))
    model.add(Dense(1))
    model.compile(loss='mse',optimizer='adam', metrics=['accuracy'])
    return model

def denormalize_data(df,pred,target):
    if target == 'temperature':
        target_variable = 'temp_2m_avg [degF]'
    elif target == 'humidity':
        target_variable = 'relative_humidity_avg [percent]'
    elif target == 'precipitation':
        target_variable = 'precip_total [inch]'
    scaler = StandardScaler()
    x = scaler.fit_transform(df[target_variable].values.reshape(-1,1))
    pred_denormalized = scaler.inverse_transform(pred.reshape(-1,1))
    return pred_denormalized
```

In [ ]:
```
model = build_model([amount_of_features,window,1],model_name,target_va
model.fit(X_train,y_train,batch_size=64,epochs=epoch,validation_split=
pred = model.predict(X_test)
y_test_denormalized = denormalize_data(data,y_test,target_variable)
pred_denormalized = denormalize_data(data,pred,target_variable)
```

Plotting function

In [ ]:
```
plt.figure(figsize=(20,10))
plt.plot(y_test_denormalized,'--',label='actual')
plt.plot(pred_denormalized,'-',label='predicted')
plt.legend()
plt.xlabel('Time')
plt.ylabel(target_variable)
```

In [ ]:
```

```