VIT®
BHOPAL
www.vitbhopal.ac.in

CLIMATE CHANGE PREDICTOR USING TIME SERIES FORECASTING

EPICS
Project Review – 2

Project Members

Uday Agarwal
19BAI10147

Aryan Tandon
19BAI10148

Parv Bhargava
19BAI10116

Aabir Dutta
19BCE10062

Mriganka Das
19BOE10053

Abdul Raziq Khan
19BCE10135

Govit Kharsare
19MIM10094

Vimal Tiwari
19BCG10054

Project Guide

Dr. Pavan Kumar

ABSTRACT

Climate change is undoubtedly one of the biggest problems in the 21st century. And with the sudden advent and breakout of disastrous climate hazards in India and many such countries, it has become a growing public concern. Hence we take steps in understanding climate change and make certain vital predictions which are both effective and efficient. We explore the performance of a phenomenological, top-down model constructed using a neural network and big data of global mean monthly temperature. By generating graphical images using the monthly temperature data of 30 years, the neural network system successfully predicts the rise and fall of temperatures for the next 10 years. Moreover, the prediction accuracy differs among climatic zones and temporal ranges. We shall be suggesting that using artificial intelligence-based forecasting methods along with conventional physics-based models because these two approaches can work together in a complementary manner.


OBJECTIVE

- The latest IPCC report has devoted a separate chapter to extreme weather events, emphasizing compound events i.e. the combination of multiple drivers and/or hazards that contribute to societal or environmental risk.
- Examples are concurrent heatwaves and droughts, compound flooding (a storm surge in combination with extreme rainfall and/or river flow), compound fire weather conditions (a combination of hot, dry, and windy conditions), or concurrent extremes at different locations.
- Thus to produce a better understanding of the weather in the current scenario and also in the near future a climate change predictor shall indeed help in setting up precautionary measures allowing to have a clear picture about the depreciations.

PROPOSED WORK

- The precipitation and temperatures (maximum and minimum) data are considered for the study and prepared for the analysis as monthly means.
- Once the data files are prepared, the ARIMA model needs to be identified. In this study, we have tried to fit separate SARIMA models to precipitation and temperature (minimum and maximum) time series.
- So, we have one SARIMA model that fits the precipitation time series and one that fits the temperature time series (minimum and maximum).

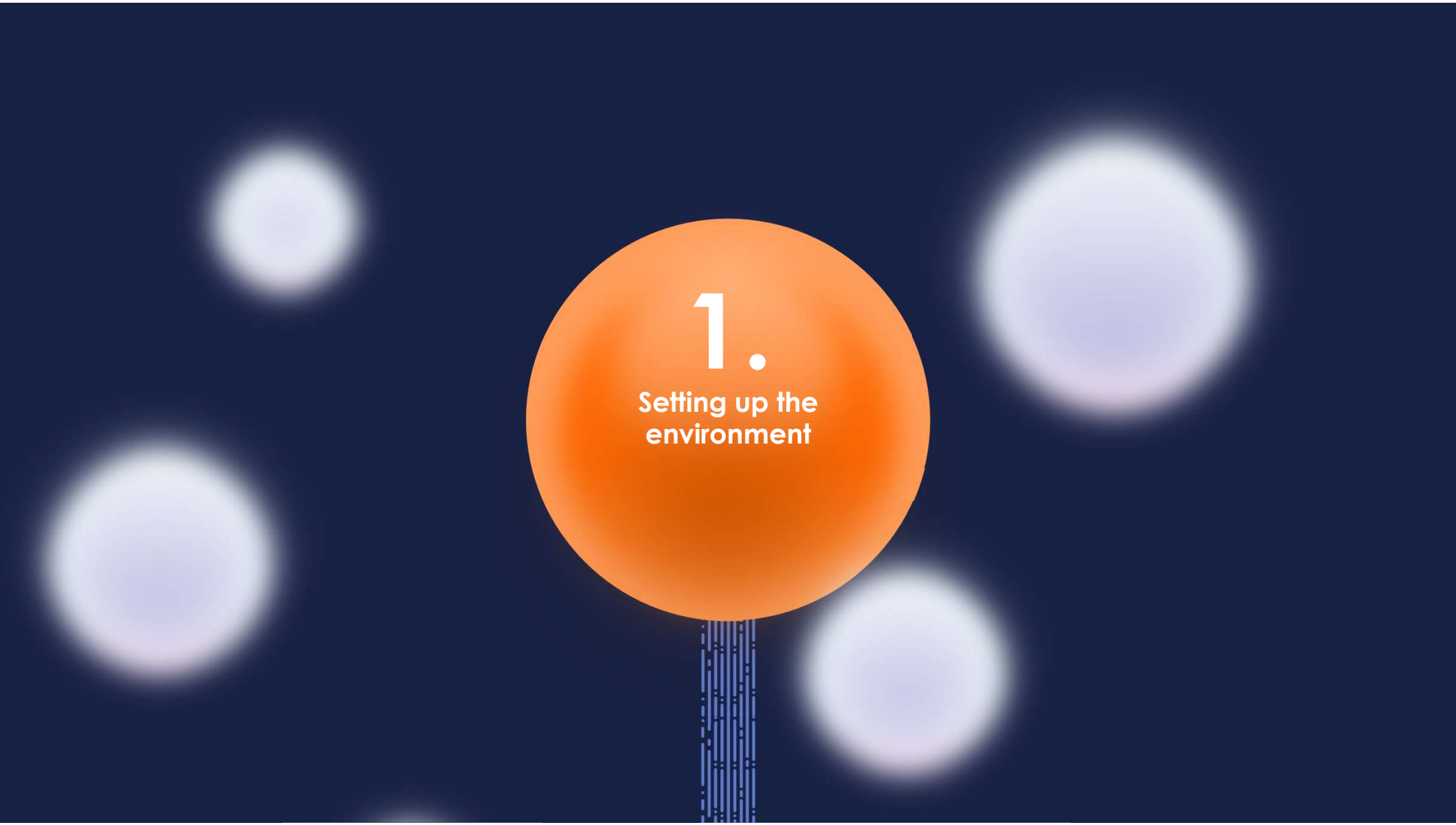
ABOUT SARIMA MODEL

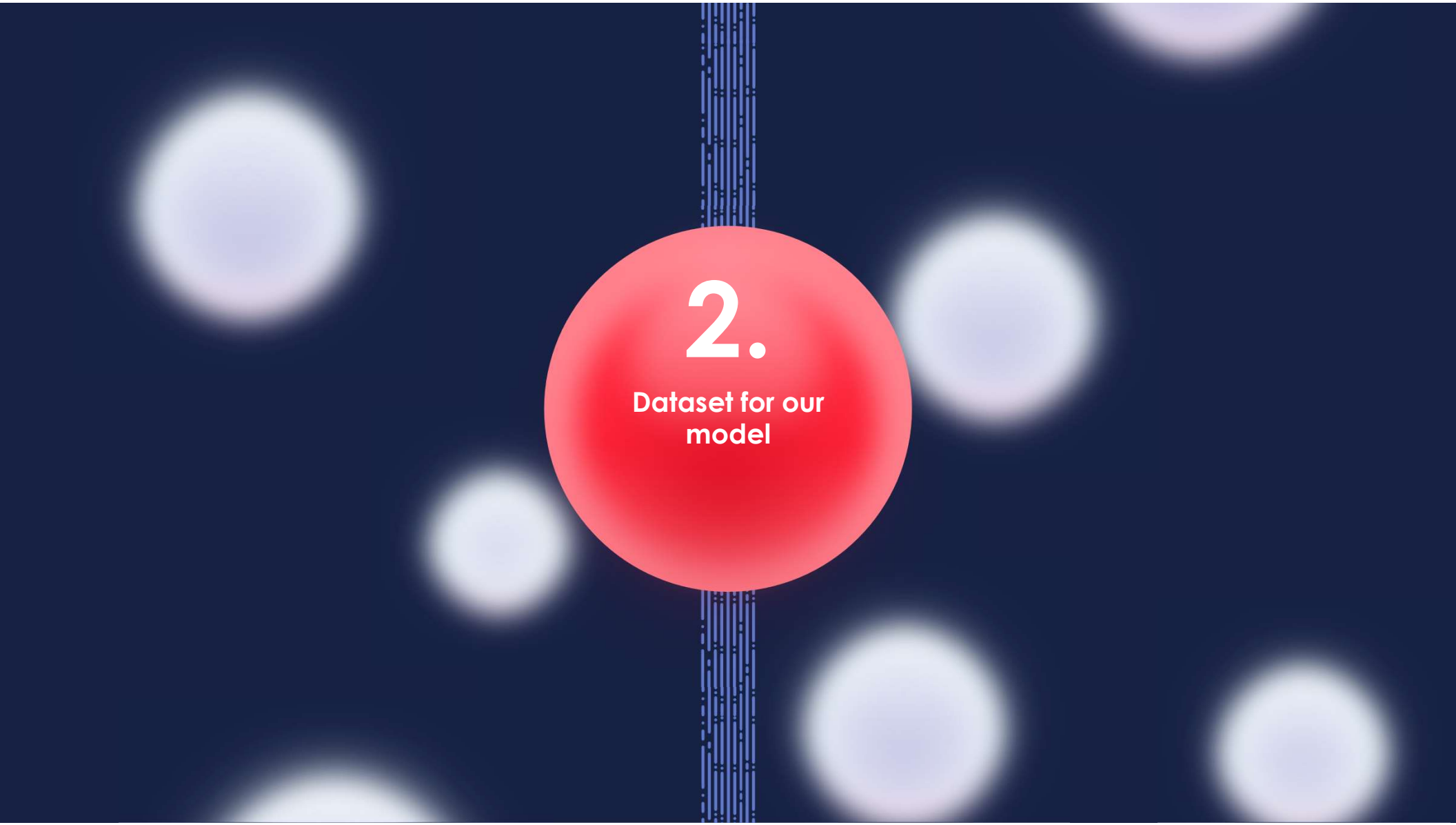
Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component.

It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality.

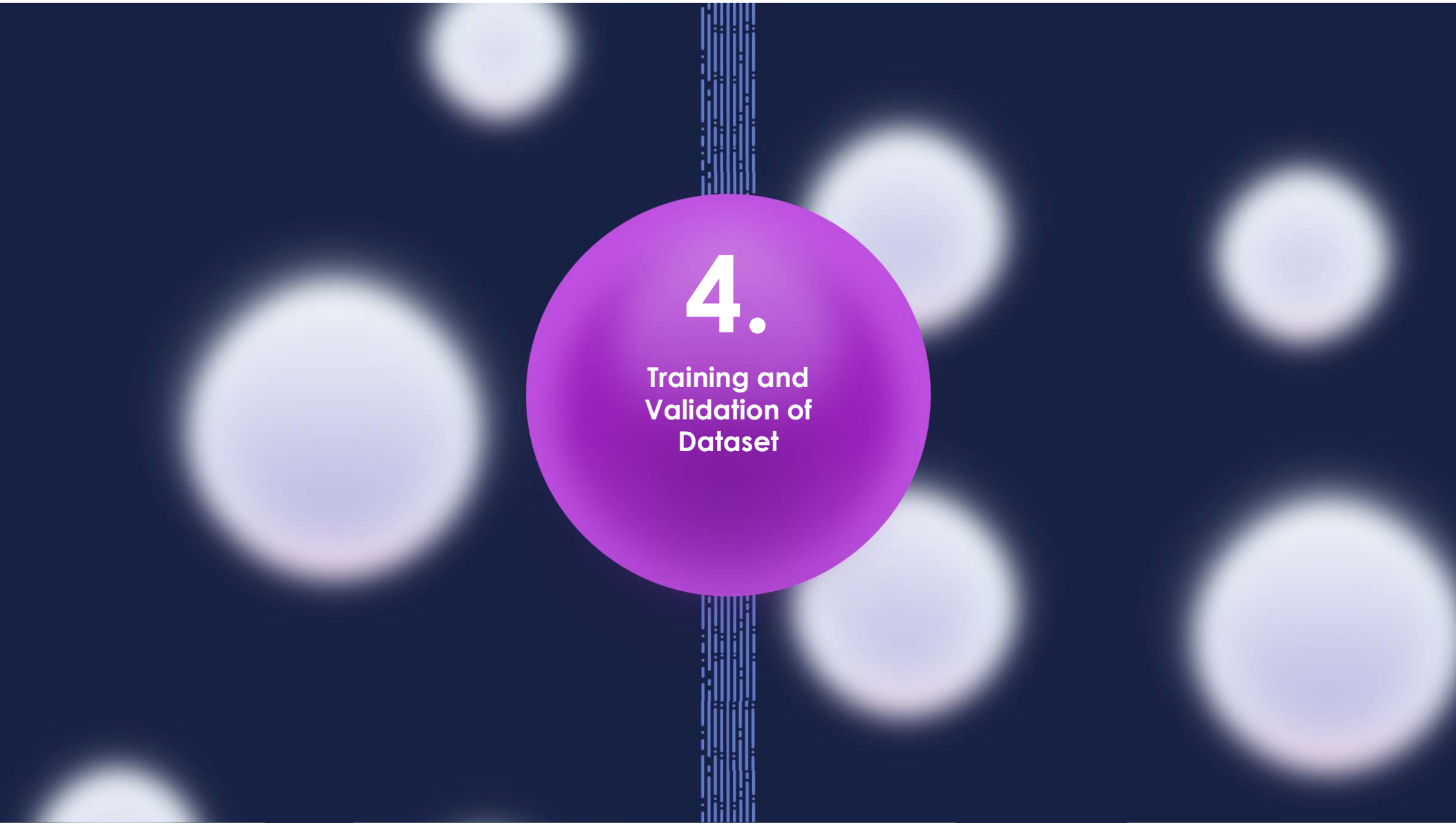
The difference between ARIMA and SARIMA is about the seasonality of the dataset. If the data used is seasonal, like it happens after a certain period of time, then we will use SARIMA. Just as we know the weather datasets are seasonal in nature.

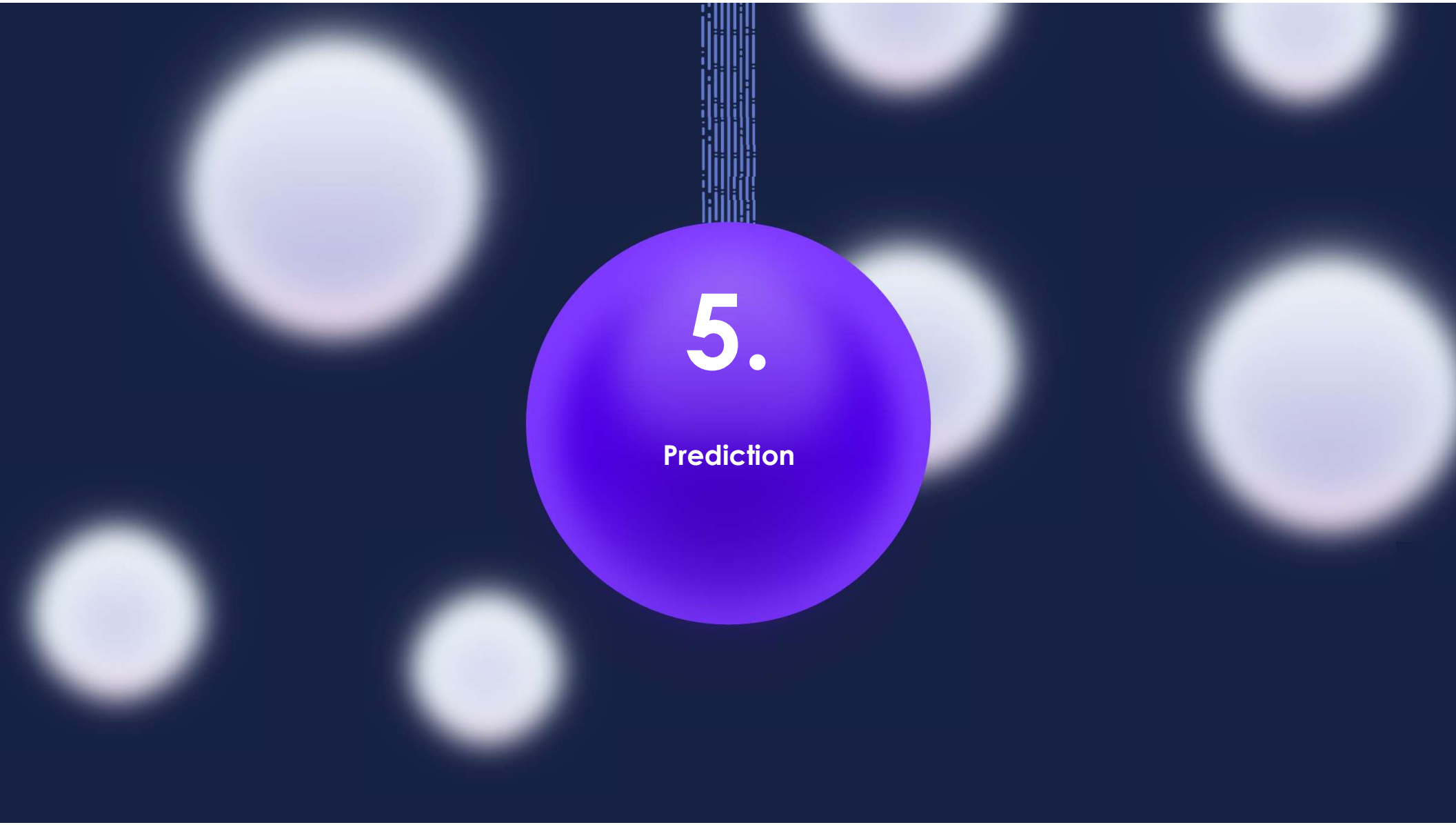
METHODOLOGY


1.
Setting up the
environment


2.
Dataset for our model


3.
Data visualization


4.
Training and Validation of Dataset


5.
Prediction

SETTING UP THE ENVIRONMENT

The environment of our project is set using the following libraries of python and using these will give us the ideal base for advancing with our time series forecasting model.

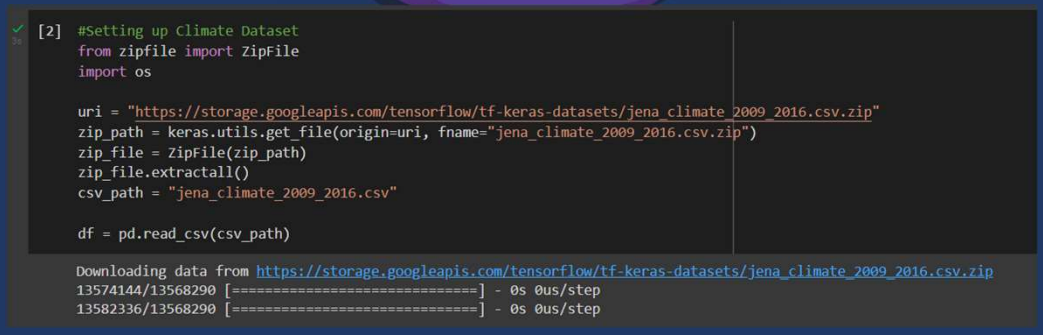
```
[1]  #Importing libraries and setting up the environment
     import pandas as pd
     import matplotlib.pyplot as plt
     import tensorflow as tf
     from tensorflow import keras
```

DATASET FOR OUR MODEL

The environment of our project is set using the following libraries of python and using these will give us the ideal base for advancing with our time series forecasting model.

We will be using the Jena Climate dataset recorded by the Max Planck Institute for Biogeochemistry. The dataset consists of 14 features such as temperature, pressure, humidity etc, recorded once per 10 minutes.

Time-frame Considered: Jan 10, 2009 - December 31, 2016



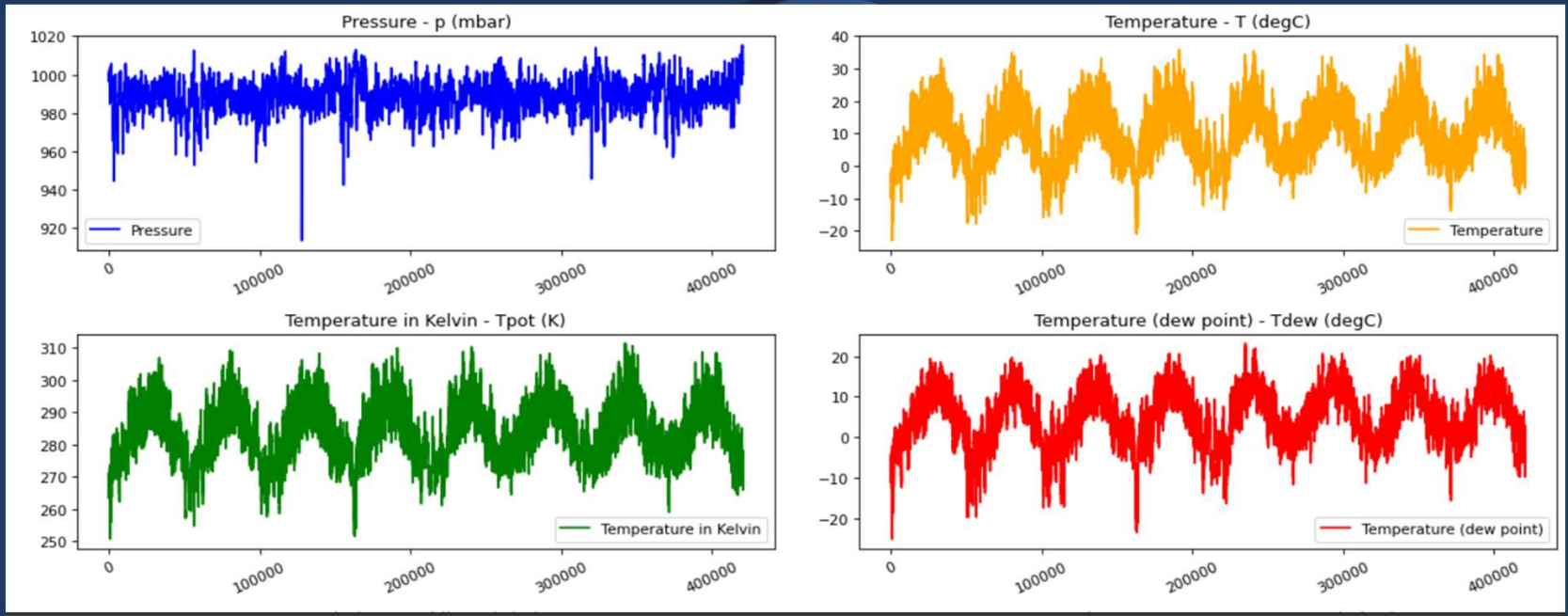
DATA VISUALIZATION

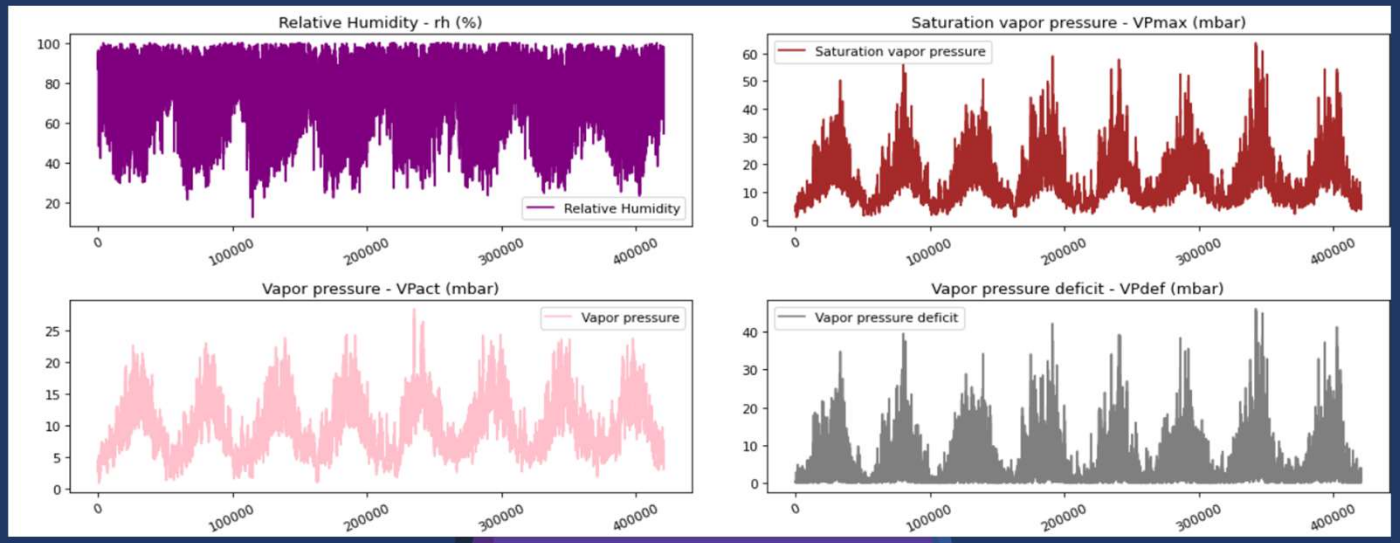
Raw Data Visualization:

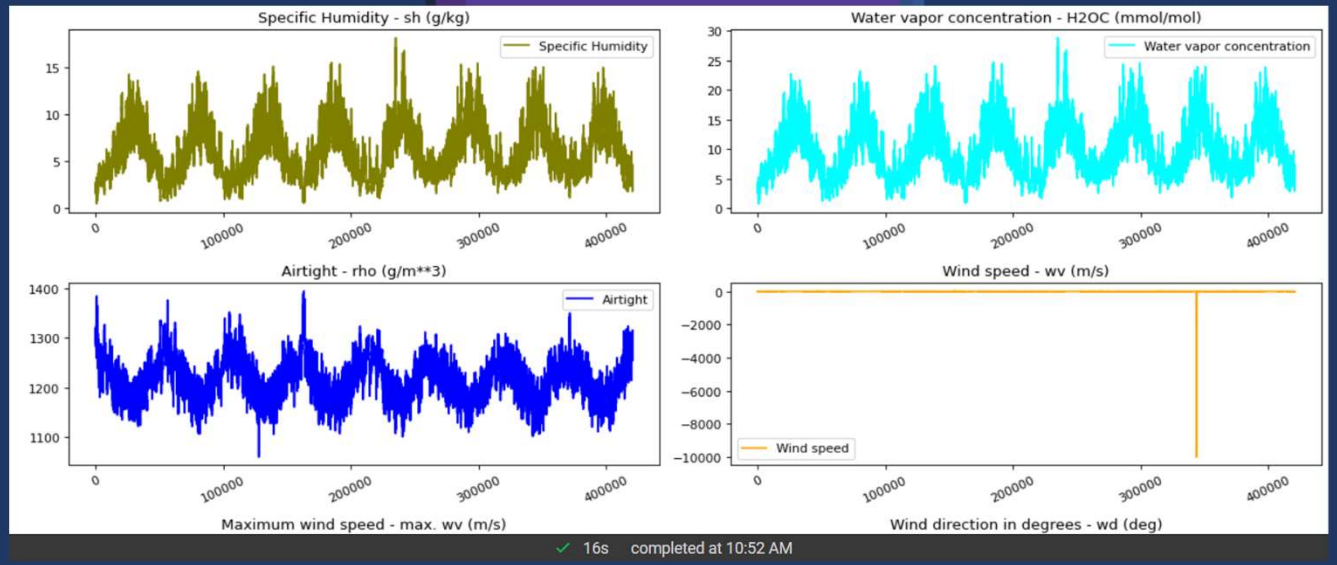
To give us a sense of the data we are working with, each feature has been plotted below. This shows the distinct pattern of each feature over the time period from 2009 to 2016. It also shows where anomalies are present, which will be addressed during normalization.

```
#Raw Data Visualization
[6] titles = [
          "Pressure",
          "Temperature",
          "Temperature in Kelvin",
          "Temperature (dew point)",
          "Relative Humidity",
          "Saturation vapor pressure",
          "Vapor pressure",
          "Vapor pressure deficit",
          "Specific Humidity",
          "Water vapor concentration",
          "Airtight",
          "Wind speed",
          "Maximum wind speed",
          "Wind direction in degrees",
    ]

    feature_keys = [
                    "p (mbar)",
                    "T (degC)",
                    "Tpot (K)",
                    "Tdew (degC)",
                    "rh (%)",
                    "VPmax (mbar)",
                    "VPact (mbar)",
                    "VPdef (mbar)",
                    "sh (g/kg)",
                    "H2OC (mmol/mol)",
                    "rho (g/m**3)",
                    "wv (m/s)",
                    "max. wv (m/s)",
                    "wd (deg)",
    ]
```

Results for raw visualization:

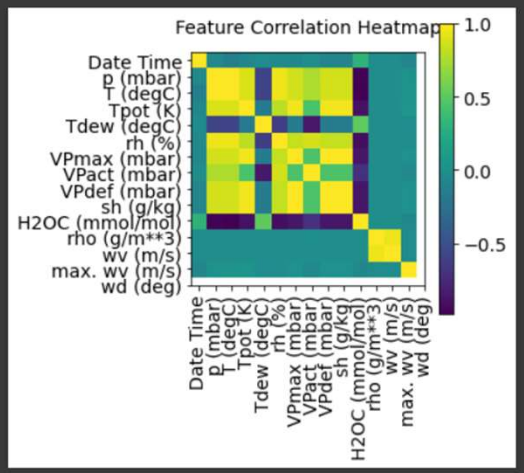



Relative Humidity - rh (%)
Relative Humidity
Saturation vapor pressure - VPmax (mbar)
Saturation vapor pressure
Vapor pressure - VPact (mbar)
Vapor pressure
Vapor pressure deficit - VPdef (mbar)
Vapor pressure deficit


Specific Humidity - sh (g/kg)
Specific Humidity
Water vapor concentration - H2OC (mmol/mol)
Water vapor concentration
Airtight - rho (g/m**3)
Airtight
Wind speed - wv (m/s)
Wind speed
Maximum wind speed - max. wv (m/s)
Wind direction in degrees - wd (deg)
16s completed at 10:52 AM

This heat map shows the correlation between different features.

```
#Heat map for the dataset
def show_heatmap(data):
  plt.matshow(data.corr())
  plt.xticks(range(data.shape[1]), data.columns, fontsize=14, rotation=90)
  plt.gca().xaxis.tick_bottom()
  plt.yticks(range(data.shape[1]), data.columns, fontsize=14)

  cb = plt.colorbar()
  cb.ax.tick_params(labelsize=14)
  plt.title("Feature Correlation Heatmap", fontsize=14)
  plt.show()

show_heatmap(df)
```



✓ 1s completed at 11:10 AM

Data preprocessing

We can see from the correlation heatmap, few parameters like Relative Humidity and Specific Humidity are redundant. Hence we will be using select features, not all.

```
[8]  #Data Preprocessing
     split_fraction = 0.715
     train_split = int(split_fraction * int(df.shape[0]))

     past = 720
     future = 72
     learning_rate = 0.001
     batch_size = 256
     epochs = 10

     def normalize(data, train_split):
       data_mean = data[:train_split].mean(axis=0)
       data_std = data[:train_split].std(axis=0)
       return(data - data_mean) / data_std
```

```
print(
    "The selected parameters are:",
    ", ".join([titles[i] for i in [0,1,5,7,8,10,11]]),
)
selected_features = [feature_keys[i] for i in [0,1,5,7,8,10,11]]
features = df[selected_features]
features.index = df[date_time_key]
features.head()

features = normalize(features.values, train_split)
features = pd.DataFrame(features)
features.head()

train_data = features.loc[0 : train_split - 1]
val_data = features.loc[train_split:]
```

```
The selected parameters are: Pressure, Temperature, Saturation vapor pressure, Vapor pressure deficit, Specific Humidity, Airtight, Wind speed
```

TRAINING AND VALIDATION OF DATASET

Training and Validation of Dataset

The training dataset labels start from the 792nd observation (720 + 72).

The timeseries_dataset_from_array function takes in a sequence of data-points gathered at equal intervals, along with time series parameters such as length of the sequences/windows, spacing between two sequence/windows, etc., to produce batches of sub-timeseries inputs and targets sampled from the main timeseries.

```
#Training Dataset
start = past + future
end = start + train_split

x_train = train_data[[i for i in range(7)]].values
y_train = features.iloc[start:end][[1]]

sequence_length = int(past / step)

dataset_train = keras.preprocessing.timeseries_dataset_from_array(
    x_train,
    y_train,
    sequence_length=sequence_length,
    sampling_rate=step,
    batch_size=batch_size,
)
```

The validation dataset must not contain the last 792 rows as we won't have label data for those records, hence 792 must be subtracted from the end of the data.

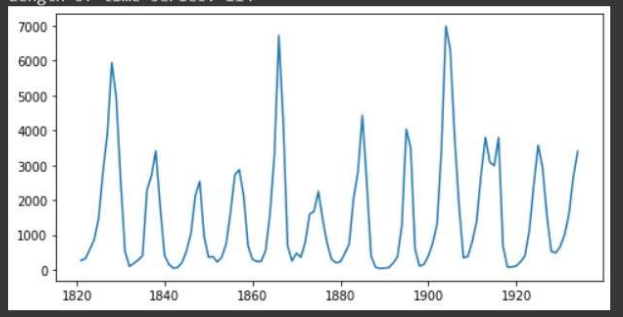
The validation label dataset must start from 792 after train_split, hence we must add past + future (792) to label_start.

```
x_end = len(val_data) - past - future

label_start = train_split + past + future

x_val = val_data.iloc[:x_end][[i for i in range(7)]].values
y_val = features.iloc[label_start:][[1]]

dataset_val = keras.preprocessing.timeseries_dataset_from_array(
    x_val,
    y_val,
    sequence_length=sequence_length,
    sampling_rate=step,
    batch_size=batch_size,
)


for batch in dataset_train.take(1):
    inputs, targets = batch

print("Input shape:", inputs.numpy().shape)
print("Target shape:", targets.numpy().shape)
```

```
Input shape: (256, 120, 7)
Target shape: (256, 1)
```

```
# Importing the Dataset for the model
URL = "https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/lynx.csv"
lynx = pd.read_csv(URL, index_col = 0)
data = lynx["value"].values
print('Length of time series:', data.shape[0])
plt.figure(figsize=(8,4))
plt.plot(lynx["time"], data)
plt.show()
```

```
Length of time series: 114
```



Results of training the Data

The following results show a detailed output of the trained data.

```
#Training
inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = keras.layers.LSTM(32)(inputs)
outputs = keras.layers.Dense(1)(lstm_out)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=learning_rate), loss="mse")
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 120, 7)]          0

 lstm (LSTM)                 (None, 32)                5120

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 5,153
Trainable params: 5,153
Non-trainable params: 0
_________________________________________________________________
```

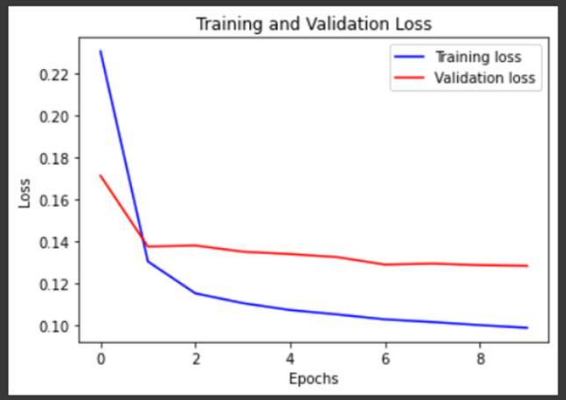
We'll use the ModelCheckpoint callback to regularly save checkpoints, and the EarlyStopping callback to interrupt training when the validation loss is no longer improving.

```
path_checkpoint = "model_checkpoint.h5"
es_callback = keras.callbacks.EarlyStopping(monitor="val_loss", min_delta=0, patience=5)

modelckpt_callback = keras.callbacks.ModelCheckpoint(
    monitor="val_loss",
    filepath=path_checkpoint,
    verbose=1,
    save_weights_only=True,
    save_best_only=True,
)

history = model.fit(
    dataset_train,
    epochs=epochs,
    validation_data=dataset_val,
    callbacks=[es_callback, modelckpt_callback],
)
```

```
Epoch 1/10
1172/1172 [==============================] - ETA: 0s - loss: 0.2305
Epoch 1: val_loss improved from inf to 0.17133, saving model to model_checkpoint.h5
1172/1172 [==============================] - 197s 165ms/step - loss: 0.2305 - val_loss: 0.1713
Epoch 2/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1305
Epoch 2: val_loss improved from 0.17133 to 0.13761, saving model to model_checkpoint.h5
1172/1172 [==============================] - 184s 157ms/step - loss: 0.1305 - val_loss: 0.1376
Epoch 3/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1154
Epoch 3: val_loss did not improve from 0.13761
1172/1172 [==============================] - 188s 160ms/step - loss: 0.1154 - val_loss: 0.1381
Epoch 4/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1107
Epoch 4: val_loss improved from 0.13761 to 0.13518, saving model to model_checkpoint.h5
```

We can visualize the loss with the function below. After one point, the loss stops decreasing.

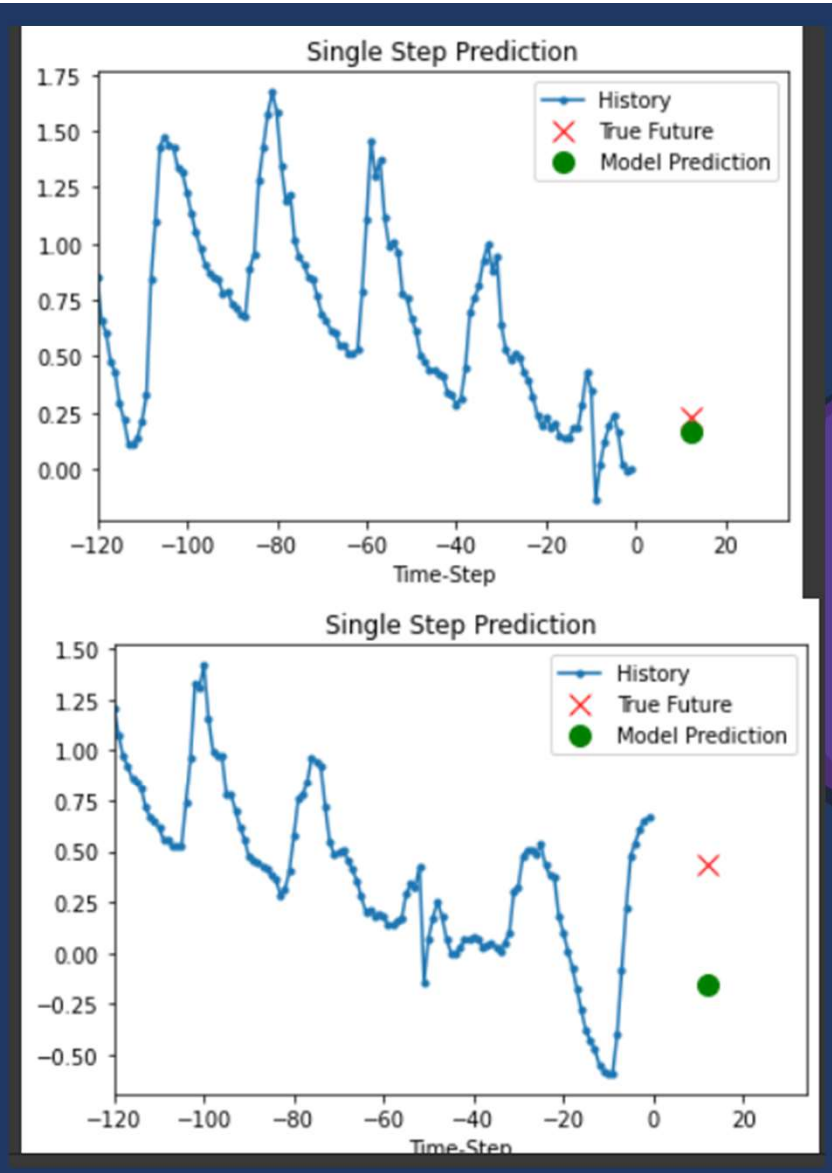
```python
def visualize_loss(history, title):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Training loss")
    plt.plot(epochs, val_loss, "r", label="Validation loss")
    plt.title(title)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()


visualize_loss(history, "Training and Validation Loss")
```

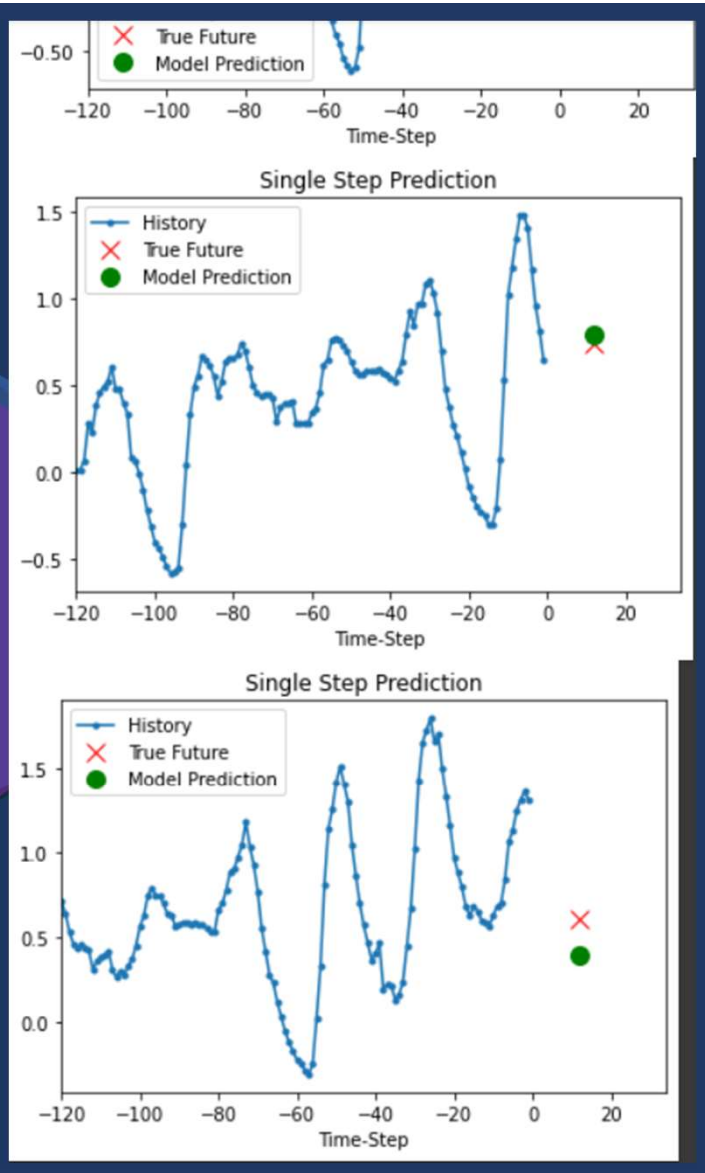


PREDICTION

The trained model above is now able to make predictions for 5 sets of values from the validation set.

```python
#Prediction
def show_plot(plot_data, delta, title):
    labels = ["History", "True Future", "Model Prediction"]
    marker = [".-", "rx", "go"]
    time_steps = list(range(-(plot_data[0].shape[0]), 0))
    if delta:
        future = delta
    else:
        future = 0

    plt.title(title)
    for i, val in enumerate(plot_data):
        if i:
            plt.plot(future, plot_data[i], marker[i], markersize=10, label=labels[i])
        else:
            plt.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels[i])
    plt.legend()
    plt.xlim([time_steps[0], (future + 5) * 2])
    plt.xlabel("Time-Step")
    plt.show()
    return


for x, y in dataset_val.take(5):
    show_plot(
        [x[0][:, 1].numpy(), y[0].numpy(), model.predict(x)[0]],
        12,
        "Single Step Prediction",
    )
```


Single Step Prediction
History
True Future
Model Prediction
Time-Step
Single Step Prediction
History
True Future
Model Prediction
Time-Step


True Future
Model Prediction
Time-Step
Single Step Prediction
History
True Future
Model Prediction
Time-Step
Single Step Prediction
History
True Future
Model Prediction
Time-Step

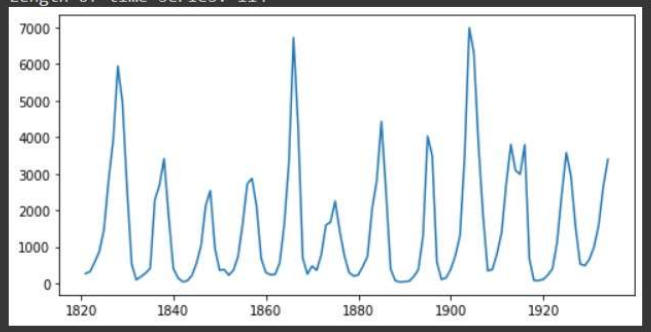
JAX

JAX is an automatic differentiation (AD) toolbox developed by a group of people at Google Brain and the open source community. It aims to bring differentiable programming in NumPy-style onto TPUs. On the highest level JAX combines the previous projects XLA & Autograd to accelerate your favorite linear algebra-based projects.

NEW MODEL

```
# Importing the Dataset for the model
URL = "https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/lynx.csv"
lynx = pd.read_csv(URL, index_col = 0)
data = lynx["value"].values
print('Length of time series:', data.shape[0])
plt.figure(figsize=(8,4))
plt.plot(lynx["time"], data)
plt.show()
```

```
Length of time series: 114
```




```
%%time
kernel = NUTS(sgt)
mcmc = MCMC(kernel, num_warmup=5000, num_samples=5000, num_chains=4)
mcmc.run(random.PRNGKey(0), y_train, seasonality=38)
mcmc.print_summary()
samples = mcmc.get_samples()
```

```
Running chain 0: 100% 10000/10000 [01:13<00:00, 228.86it/s]
Running chain 1: 100% 10000/10000 [01:13<00:00, 285.91it/s]
Running chain 2: 100% 10000/10000 [01:13<00:00, 236.81it/s]
Running chain 3: 100% 10000/10000 [01:13<00:00, 249.27it/s]
```

	mean	std	median	5.0%	95.0%	n_eff	r_hat
coef_trend	33.59	122.06	14.10	-99.64	152.78	1904.42	1.00
init_s[0]	84.84	104.31	63.35	-63.06	241.40	5796.14	1.00
init_s[1]	-20.07	70.21	-24.19	-132.36	86.35	7629.30	1.00
init_s[2]	29.53	94.25	17.99	-118.37	164.41	6180.99	1.00
init_s[3]	126.35	126.54	106.84	-54.97	328.29	5000.04	1.00
init_s[4]	457.55	263.35	413.21	85.61	851.33	4543.72	1.00
init_s[5]	1183.05	468.12	1106.27	468.69	1865.87	3179.71	1.00
init_s[6]	2008.15	676.61	1905.43	975.97	3026.53	2285.37	1.00
init_s[7]	3702.58	1104.04	3556.11	1915.62	5337.32	2070.79	1.00
init_s[8]	2615.86	840.02	2501.46	1322.06	3889.29	2113.34	1.00
init_s[9]	967.28	455.56	892.93	292.60	1632.05	3361.29	1.00
init_s[10]	47.23	104.45	32.75	-109.30	198.34	7192.65	1.00
init_s[11]	1.24	55.16	-1.32	-78.31	74.30	3711.73	1.00
init_s[12]	-9.41	66.67	-10.99	-120.09	83.57	6606.77	1.00
init_s[13]	66.38	105.14	46.75	-82.97	210.01	4522.81	1.00
init_s[14]	339.33	259.99	286.95	-33.47	689.60	4653.25	1.00

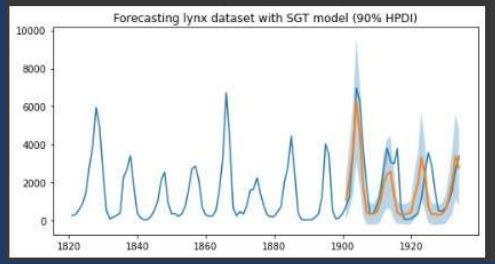
NEW PREDICTION

```
# Forecasting

predictive = Predictive(sgt, samples, return_sites=["y_forecast"])
forecast_marginal = predictive(random.PRNGKey(1), y_train, seasonality=38, future=34)[
    "y_forecast"
]
```

```
y_pred = jnp.mean(forecast_marginal, axis=0)
sMAPE = jnp.mean(jnp.abs(y_pred - y_test) / (y_pred + y_test)) * 200
msqrt = jnp.sqrt(jnp.mean((y_pred - y_test) ** 2))
print("sMAPE: {:.2f}, rmse: {:.2f}".format(sMAPE, msqrt))
```

```
sMAPE: 64.16, rmse: 1252.14
```

```
plt.figure(figsize=(8, 4))
plt.plot(lynx["time"], data)
t_future = lynx["time"][80:]
hpd_low, hpd_high = hpdi(forecast_marginal)
plt.plot(t_future, y_pred, lw=2)
plt.fill_between(t_future, hpd_low, hpd_high, alpha=0.3)
plt.title("Forecasting lynx dataset with SGT model (90% HPDI)")
plt.show()
```



CONCLUSION

Thus, we could implement a seasonal SARIMA model in Python. We made extensive use of the pandas and statsmodels libraries and showed how to run model diagnostics, as well as how to produce forecasts of the temperature.

Here are a few other things we can try additionally to our coded model:

- Change the start date of our dynamic forecasts to see how this affects the overall quality of your forecasts.
- Try more combinations of parameters to see if we can improve the goodness-of-fit of the model.
- Select a different metric to select the best model. For example, we used the AIC measure to find the best model, but we could seek to optimize the out-of-sample mean square error instead.

CONTRIBUTION

Uday Agarwal Data Visualization	**Aryan Tandon** Data Visualization	**Parv Bhargava** Core Code	**Aabir Dutta** Core Code
Mriganka Das Finding real world applications of our project	**Abdul Raziq Khan** Dataset Description	**Govit Kharsare** Literature Review	**Vimal Tiwari** Literature Review


THANK YOU!