Introduction:

This project is a part of IIT Dh's Tech fest 'PARSEC'. This project is supposed to predict the solar radiatio n and goemagnetic storms in the near future if past data is given.
This report delves into the intricate task of determining Disturbance Storm Index (DST) values through a c omprehensive data analysis approach.
It also aims to provide the reader with all the information such as why we took some particular features to scale, preprocess data, scale it and our thoughtprocess while choosing our Model.

Our objective for the Quest 1 of AURORA is to predict Dst (Disturbance Storm-TimeIndex), which is a me asure of Solar Storm Intensity, using Level 2 Data
refined from DSCOVR's Raw Data that
○ Push the prediction performance envelope.
○ Is within operationally feasible restrictions
○ Is making use of specific real-time solar-wind data sources.

Data Preprocessing:

1. Importing required libraries for smooth functioning of our code.
2. Reading each csv file, checking data for prediction.

->Start of preprocessing
1. Timedelta columns and indices readjusted/periods were grouped,max-min values,mean and standard d eviations observed.

2. Checking for null values & Visualising the different parameters through line plots, We infer the following :
-> The proportion varied by feature, but every one required some kind of imputation[a huge number of na values encountered for each of the feature].
-> The na values were imputed using the forward fill method(ffill).
-> The features 'bx_gse' and bx_gsm are closely related. Their respective plots showed near resemb lance.
-> The source column was dropped to find the correlation of other data.
-> A correlation matrix was plotted against all the features.
=>speed and temp. were highly anti-correlated with correlation values ranging from [-0.6,-0.4].

Correlation And Data Visualisation:

1. Highly correlated features were shortlisted for festure scaling and next procedures...
-> "bt"
-> "temperature"
-> "bx_gse"
-> "by_gse"
-> "bz_gse"
-> "speed"
-> "density"
-> "sunspots numbers"

2. Subsets of these data were made and monthly/daily data was made hourly so as to aggregate with oth er features
Missing values when made hourly were filled using 'mean'.
'smoothed_ssn' was imputed using forward fill.
'solar_wind' was imputed through interpolation.

3. Standard Scaler was used for Feature Scaling: to merged and fitted the solar_wind and sunspots data

4. Then the correlated features were simply preprocessed using the preprocess_features function within S tandard Scaler.

5. Modifying the shape of labels dst by data shifting as we want to predict dst of 1 hour ahead also by cre ating variables t0 and t1(we are basically creating two columns t0 and t1 which will predict the dst values with 1 hour gap)
Hence we can do multi-step prediction by providing it both steps.
-List of column names for the labels joining the t0 and t1 columns with the dst dataframe

6. The given data is large in size. So, the data is splitted in three different periods namely, train, test and v alidation
we have created a new dataframe interim which includes all the rows from the original dataframe "data" except for those whose index values matches with the index of the dataframe "test".

7. Finally, Data splits: 3 parts(width=0.75)--> "train_a", "train_b" and "train_c" were created from the existi ng preprocessed dataset so as to monitor the train, validation and test split counts over hourly timesteps.
using plots.

MODEL TRAINING-->

1. We chose the Tensorflow neural network library for our model training:
-> Tensorflow provides functionality for working with timeseries data.
-> In this case, the code is utilizing the timeseries_dataset_from_array function from tensorflow.keras.p reprocessing to create time series datasets.
-[import tensorflow as tf]
-[from tensorflow.keras.preprocessing import timeseries_dataset_from_arr]
-> The data_config dictionary holds configuration parameters such as the number of time steps (timest eps) and batch size (batch_size).
-> The timeseries_dataset_from_df function takes a DataFrame (df) and batch size as input and create s a time series dataset. It does this by iterating over periods in the DataFrame, extracting input and outpu t sequences, and using timeseries_dataset_from_array to create the dataset.
-> The train_ds and val_ds are created by calling the timeseries_dataset_from_df function for the traini ng and validation data, respectively.
-> Finally, the code prints the number of batches in the training and validation datasets.

2. We have used LSTM (Long Short-Term Memory) Neural network architecture for using Sequential Mod eling. Since our task is a type of time series prediction task, this RNN is the optimal one.
-The Sequential model is created, representing a linear stack of layers.
-> We have added the Dense Layer with number of neurons equal to len(YCOLS)
-> We have configured the model after several trials for n_epochs=16, n_neurons=512, (an epoch refer s to one complete pass through the entire training dataset during the training phase)
For the compilation of the model, we have used the Adam Algorithm and the loss is calculated by mea n_squared error.

3. Prints only the shape of the first batch.

4. The fit method trains the model on the provided training data (train_ds) for the specified number of epo chs, using the specified batch size. The training progress is monitored, and if a validation dataset is prov ided (val_ds), the model's performance on the validation set is evaluated after each epoch.

5. The training history, including metrics like loss and accuracy on both the training and validation sets, is returned and stored in the history variable. This information can be useful for later analysis and visualizati

on of the training process.

6. Finishing ===>
   -> Plotting Training History: it includes metrics like loss and other metrics specified during training, over epochs.
   -> Preparing Test Dataset: A time series dataset for testing (test_ds) is created using the timeseries_dataset_from_df function, similar to what was done for the training and validation datasets.
   -> Model Evaluation on Test Dataset: The trained model is evaluated on the test dataset (test_ds) using the evaluate method. The resulting Mean Squared Error (MSE) is then printed. Additionally, the Root Mean Squared Error (RMSE) is calculated and printed, which is a common metric for regression problems.
   -> Saving Model and Scaler: The trained model is saved to a file named "model".The scaler used for normalization during training (scaler) is saved to a file named "scaler.pck".e
   -> Updating and Saving Configuration: The data_config dictionary is updated by adding a new key, "solar_wind_subset", and its value (SOLAR_WIND_FEATURES).

The updated data_config is then saved to a JSON file named "config.json". This file stores configuration information that might be needed when deploying or using the model in different contexts.