University of North Texas

Introduction to Big Data and Data Science – CSCE 5300

Group Number 8
Project Report

Title

Solar-Net: Leveraging Transformers for Enhanced Solar Power Prediction

Team Members:
Gayathri Baman

Table of Contents:

1. Abstract

This project presents Solar-Net: A solar irradiance forecasting framework based on the transformer model, integrated with the Hadoop ecosystem for the efficient handling of big data. The rising popularity of solar energy means accurate long-term predictions of solar irradiance are now essential for the efficient management, planning, and optimization of energy resources. Most classic forecasting models, such as SVR, LSTM, or CNN, have some limitations concerning long-range dependency and massive datasets. This work shows how to build a scalable and robust solution based on a transformer architecture integrated with big data tools within the Hadoop ecosystem such as Apache Hive, Spark MLlib, and HDFS. The model shows promising results in terms of accuracy and predictive capacity when compared to traditional approaches, thereby proving to be a great fit for solar power prediction at scale.

2. Introduction

However, with increasing global awareness about the reality of climate change and environmental sustainability, transition to renewable energy sources is accelerated rapidly, of which solar energy is one among many. The intermittence of solar irradiance results from varying weather conditions that cause difficulty in maintaining a reliable energy generation. Predictive models help forecast solar energy generation and aid in good management of the grid system, energy trading, and optimization of battery storage.

In the last few years, transformer architectures have been introduced to machine learning as extremely powerful techniques, outperforming recurrent ones in a variety of time series forecasting applications. Instead of recurrent architectures like LSTMs or GRUs, transformers rely on self-attention mechanisms, which are capable of capturing long dependencies from the input, rendering these models perfectly applicable for complicated temporal prediction tasks. The project evaluates the effect of transformers in the field of solar irradiance prediction with a scalable architecture built using the Hadoop ecosystem.

3. Problem Statement

One of the most difficult tasks in forecasting solar irradiance remains in its non-deterministic behavior caused by multiple weather factors such as cloud coverage, humidity, winds, and temperature. Although traditional methods are already widely used in such applications, they typically:

- Had many difficulties with the long-term temporal dependencies.
- Had scalability bounds on the codes to high volume sensor data.
- Need rather complex feature engineering to obtain moderate performance.

This project will be depicting intelligent, scalable and accurate forecasting pipelines for it:

- Uses a transformer-based model that can work with very long sequences and can learn from sequences.

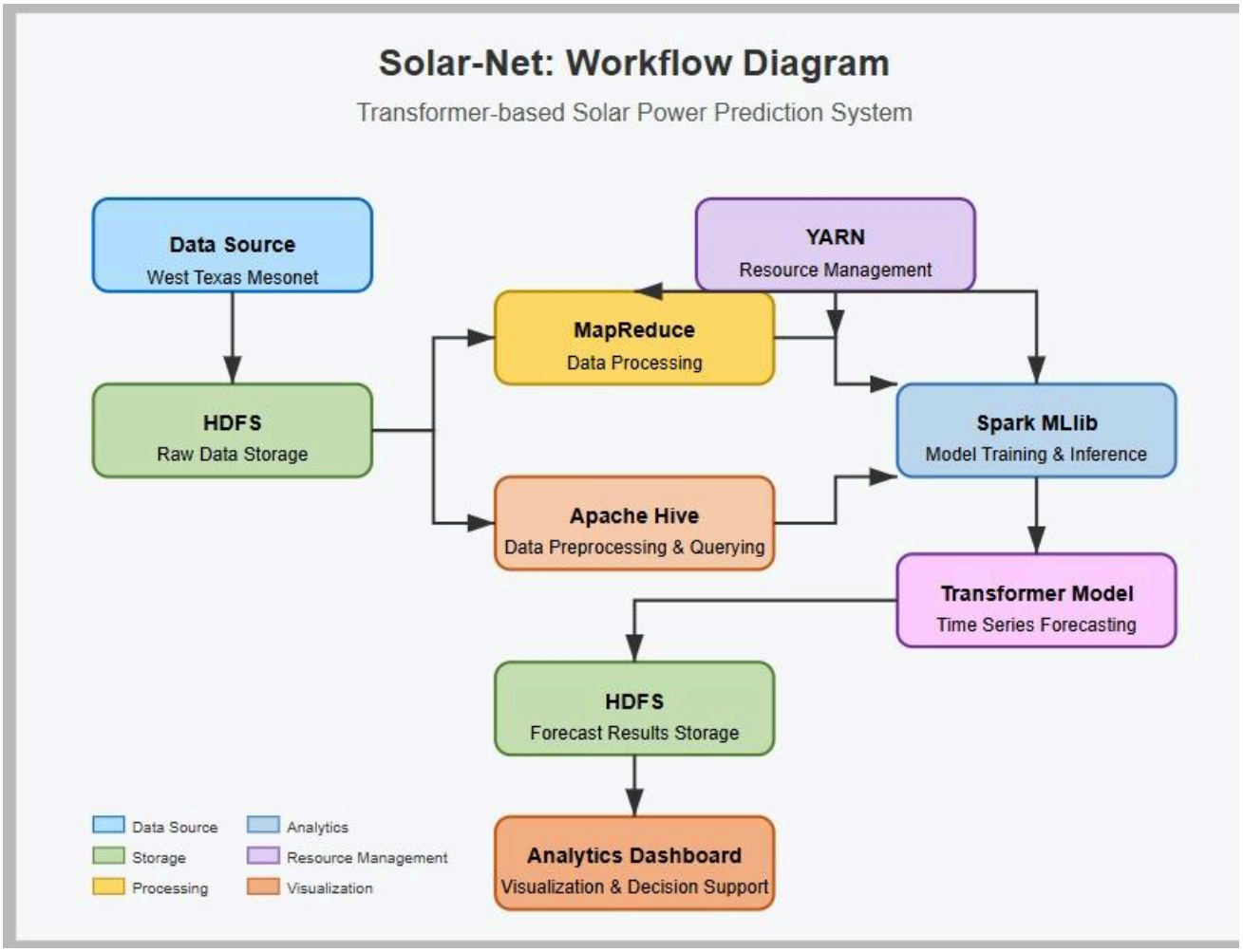
- Integrates along with Hadoop-specific tools like Hive and Spark for distributed data storage, processing and model training purposes.

- Provides performance improvements over the state-of-the-art models by means of systematic experimentation.

4. Methodology and Code Explanation

The general process consists of data acquisition, preprocessing, feature engineering, model training, and evaluation with traditional and deep learning models, culminating in the transformer-based Solar-Net model.

WorkFlow Diagram:



4.1 Data Preprocessing

Data preprocessing refers to the cleaning and setup of the dataset to be trained. Libraries like pandas, seaborn, and matplotlib are employed to load and analyze the data.

```
import numpy as np

import pandas as pd

import seaborn as sns

import plotly.express as px

import matplotlib.pyplot as plt

import plotly.graph_objects as go

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.svm import SVR

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.models import Model,Sequential

from tensorflow.keras.layers import Input, LSTM, Dense, Dropout,
LayerNormalization, MultiHeadAttention, GlobalAveragePooling1D

from sklearn.preprocessing import MinMaxScaler

from keras_tuner import HyperModel, RandomSearch

from scikeras.wrappers import KerasRegressor

from scipy import stats
```

4.1.2 Dataset Link:

https://www.kaggle.com/datasets/stucom/solar-energy-power-generation-dataset

```
data = pd.read_csv('solar_power_data.csv')
```

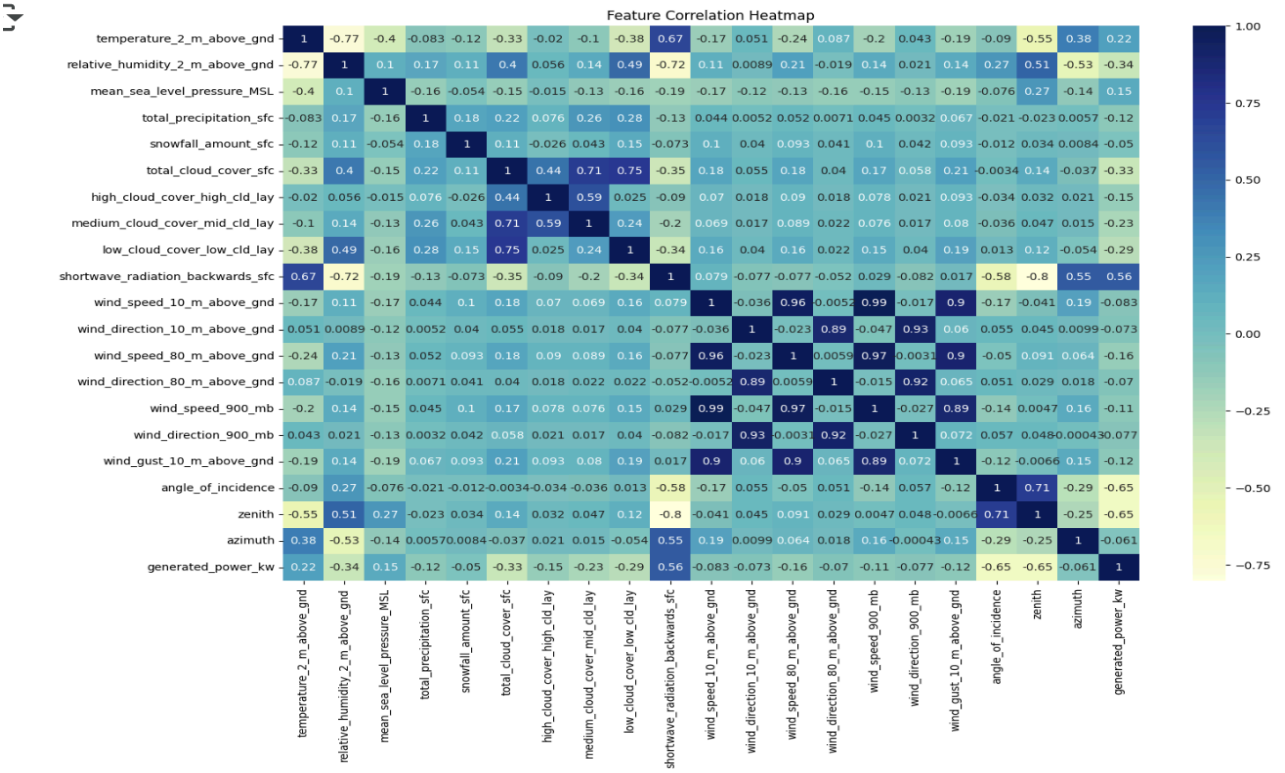
4.1.3.Data Description:

Here's the description of the columns of Solar Power Generation Dataset:

Column	Description
`temperature_2_m_above_gnd`	Air temperature at 2 meters above ground level (°C or °F), affecting solar panel efficiency.
`relative_humidity_2_m_above_gnd`	Relative humidity percentage at 2 meters above ground level, indicating moisture in the air.
`mean_sea_level_pressure_MSL`	Average atmospheric pressure at mean sea level (hPa or mmHg), influencing weather conditions.
`total_precipitation_sfc`	Total precipitation at the surface (mm), which can reduce solar radiation reaching panels.
`snowfall_amount_sfc`	Amount of snowfall at the surface (mm), potentially covering solar panels and decreasing output.
`total_cloud_cover_sfc`	Total cloud cover at the surface (tenths), indicating sky obscurity affecting solar radiation.
`high_cloud_cover_high_cld_lay`	Amount of high-altitude cloud cover (tenths), affecting solar radiation differently.
`medium_cloud_cover_mid_cld_lay`	Amount of mid-altitude cloud cover (tenths), influencing solar radiation levels.
`low_cloud_cover_low_cld_lay`	Amount of low-altitude cloud cover (tenths), impacting solar generation.

`shortwave_radiation_backwards_sfc`	Amount of shortwave radiation reflected back from the surface (W/m²), useful for energy absorption.
`wind_speed_10_m_above_gnd`	Wind speed at 10 meters above ground level (m/s), influencing cooling and solar efficiency.
`wind_direction_10_m_above_gnd`	Wind direction at 10 meters above ground level (degrees), indicating prevailing winds.
`wind_speed_80_m_above_gnd`	Wind speed at 80 meters above ground level, relevant for understanding high-altitude conditions.
`wind_direction_80_m_above_gnd`	Wind direction at 80 meters above ground level (degrees), providing data on higher-altitude winds.
`wind_speed_900_mb`	Wind speed at a pressure level of 900 mb, typically around 800-1000 meters above ground.
`wind_direction_900_mb`	Wind direction at a pressure level of 900 mb (degrees), affecting weather patterns and radiation.
`wind_gust_10_m_above_gnd`	Maximum wind gusts at 10 meters above ground level (m/s), impacting solar panel structures.
`angle_of_incidence`	Angle at which sunlight strikes the solar panels (degrees), affecting energy capture.
`zenith`	Angle between the sun and the vertical (degrees), important for solar energy availability.
`azimuth`	Angle of the sun's position relative to true north (degrees), helping determine panel orientation.
`generated_power_kw`	Electrical power generated by the solar panels (kW), the primary output of interest in the dataset.

4.1.4 Target and feature Correlation Heatmap:



4.1.5 Outlier Removal:

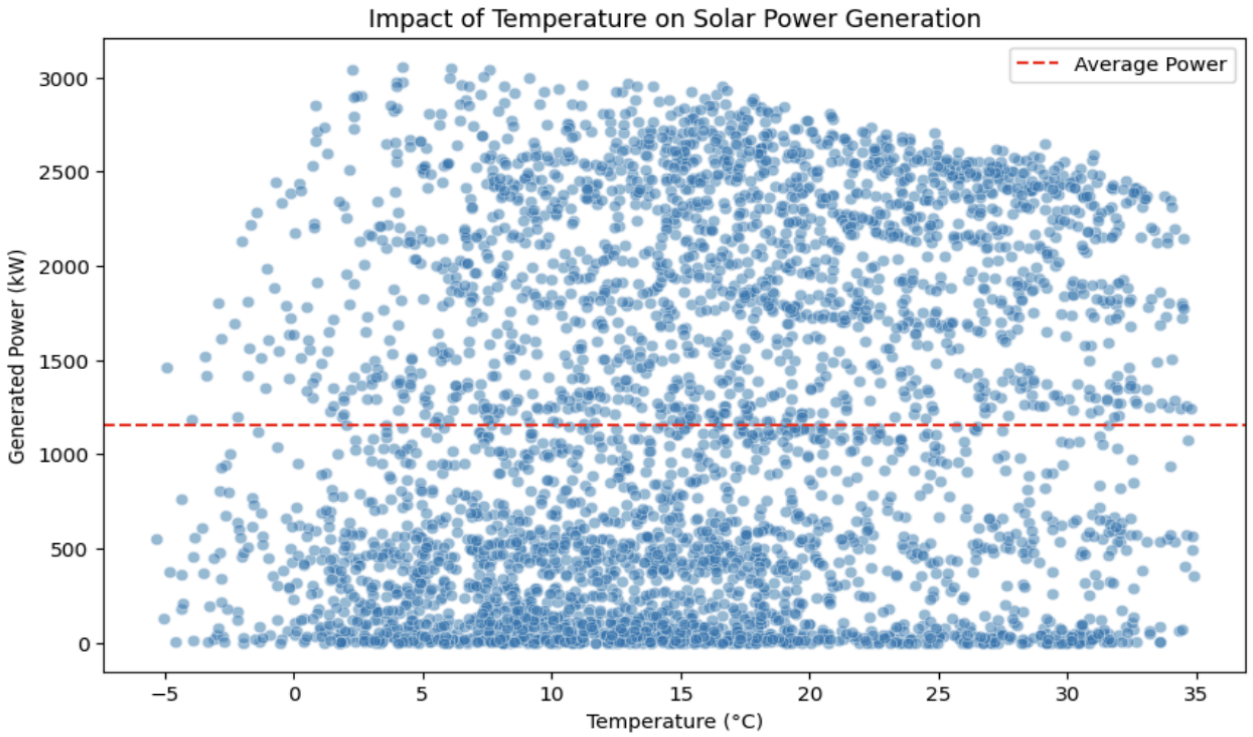
There are some noisy records that are a possibility for skewing the learning. Thus, the Z-score-outlier filtering is applied, leading to good clean records in the actual learning process.

```
z_scores = np.abs(stats.zscore(data.select_dtypes(include=[np.number])))

filtered_entries = (z_scores < 3).all(axis=1)

data = data[filtered_entries]
```

Scatter Plot for Solar Power Generated v/s Temperature:

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data,                    x='temperature_2_m_above_gnd',
y='generated_power_kw', alpha=0.5)
plt.title('Impact of Temperature on Solar Power Generation')
plt.xlabel('Temperature (°C)')
plt.ylabel('Generated Power (kW)')
plt.axhline(y=data['generated_power_kw'].mean(),  color='r',  linestyle='--',
label='Average Power')
```

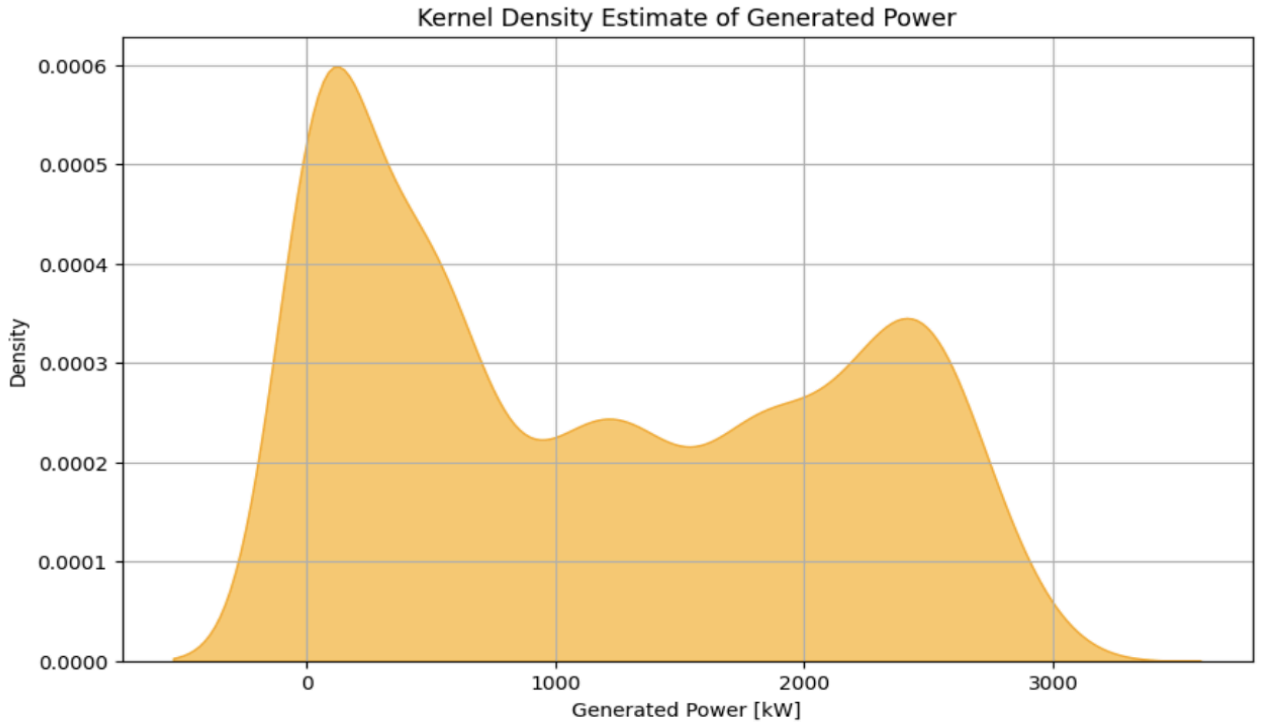
```
plt.legend()
plt.show()
```



Kernel Density Estimate (KED)

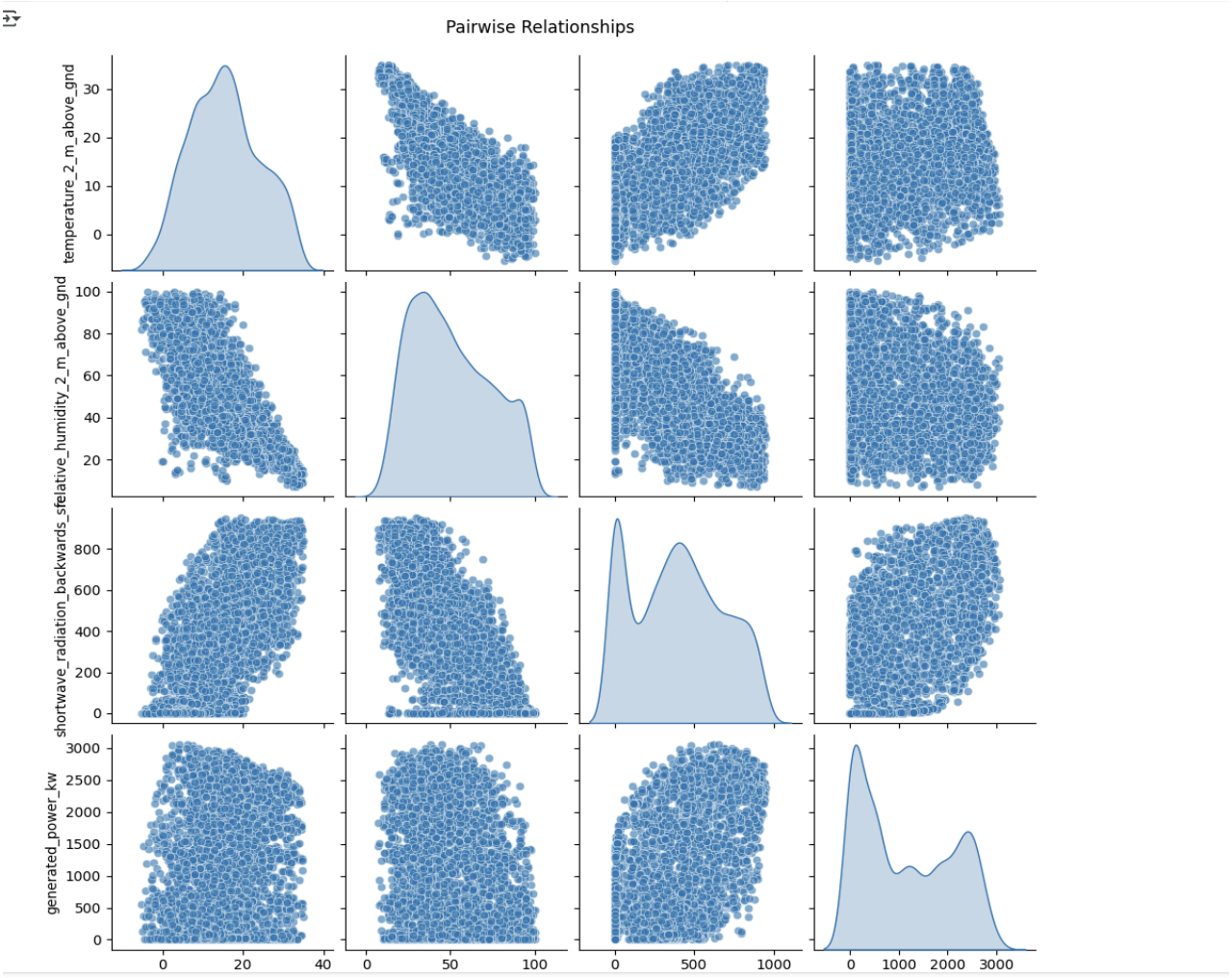
The code is designed to offer visualization presentations for the analyses conducted on solar power generation data. It presents a Kernel Density Estimate (KDE) from Seaborn, which gives the smoothed distribution of the generated power (generated_power_kw), with shading in orange indicating density. Furthermore, the code shows a histogram from Plotly that visualizes the distribution of the power generation values normalized for probability density, including power in the x-axis and frequency in the y-axis. Two scatter plots are shown, which explore: relationship between shortwave radiation and generated power; as well as looking at generated power, temperature, and cloud cover, while color coding the cloud cover, to be able to identify patterns and trends in how environmental factors affect solar power generation.

```
plt.figure(figsize=(10, 6))
sns.kdeplot(data['generated_power_kw'], fill=True, color='orange', alpha=0.6)
plt.title('Kernel Density Estimate of Generated Power')
plt.xlabel('Generated Power [kW]')
plt.ylabel('Density')
plt.grid()
plt.show()
```



Out of these, three visualizations are created to analyze the relationship between weather conditions and the power generated. First is the polar scatter plot that shows wind direction against power generated. It comes with custom labels and a dark template. The second visualization is the 3D scatter plot showing the relationship between temperature, humidity, and power generated with colour-coded markers. The last plot is a pair plot using Seaborn showing pairwise relationships between temperature, humidity, solar radiation, and generated power with kernel density estimates (KDE) on its diagonals. All the above visualizations are directed toward analyzing correlations by using Plotly and Seaborn.

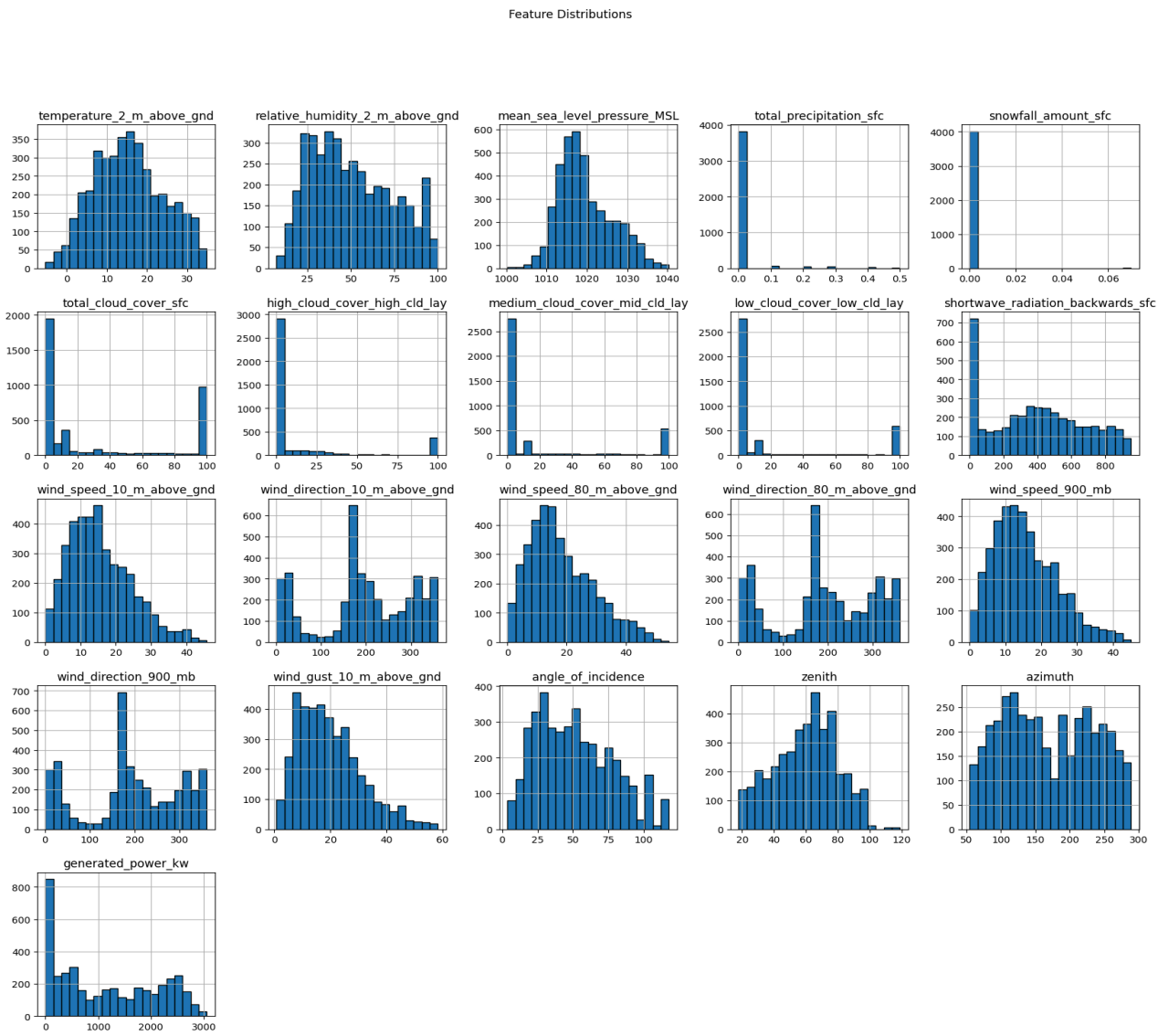
```
subset                    =                    data[['temperature_2_m_above_gnd',
'relative_humidity_2_m_above_gnd',

            'shortwave_radiation_backwards_sfc', 'generated_power_kw']]



sns.pairplot(subset, diag_kind='kde', plot_kws={'alpha': 0.6})

plt.suptitle("Pairwise Relationships", y=1.02)

plt.show()
```



Visualize Data Distribution

The code prepares and visualizes weather data. First, it calculates the average values of four weather factors (temperature, humidity, cloud cover, and wind speed) from the dataset and stores them in a DataFrame (`summary_df`). Then, it creates a radar chart using Plotly (`px.line_polar`) to display the average values of these factors, with the chart's style set to 'plotly_dark' and a filled line for better visualization. Additionally, the code generates histograms for all features and the target variable using Matplotlib to visualize their distributions. The histograms are plotted with 20 bins, and a title is added.

```
# Plot distributions of features and target variable

data.hist(bins=20, figsize=(20, 18), edgecolor='black')

plt.suptitle("Feature Distributions")

plt.show()
```



4.2 Feature Scaling and Data Splitting

The code is intended for feature scaling on the numeric columns of the dataset through StandardScaler from sklearn. The code iterates to achieve transformation for each of the numerical columns into 0 mean and standard deviation of 1. Then the dataset is delimiting with features (X) and target variable (y) - generated_power_kw being the target - and splitting the train and test with an 80:20 configuration for train and test respectively using sklearn's train_test_split. The random_state=42 ensures reproducibility.

```
scaler=StandardScaler()

for col in data.select_dtypes(include=np.number).columns:

   data[col]=scaler.fit_transform(data[[col]])

X = data.drop('generated_power_kw', axis=1)

y = data['generated_power_kw']
```

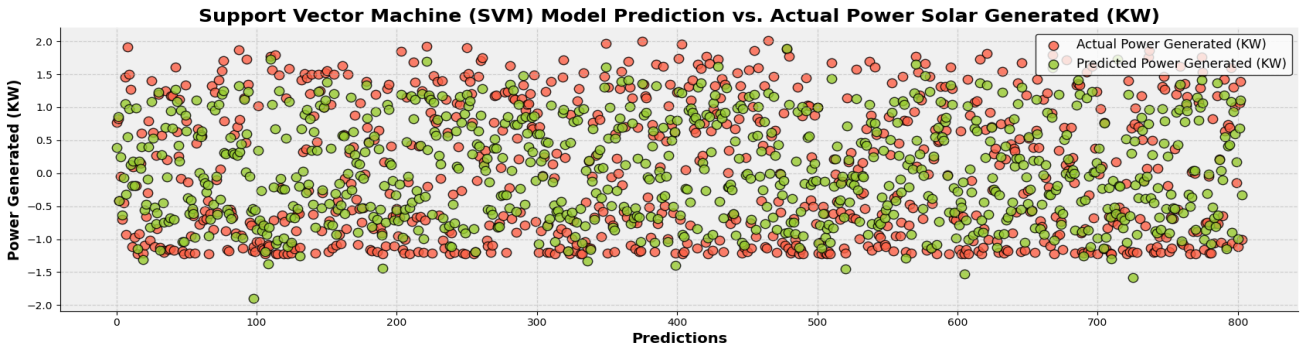
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.3 Traditional Machine Learning Models

Support Vector Regressor (SVR)

Hyperparameter tuning is done using `GridSearchCV` for the SVR model on a specific parameter grid comprising kernel, C, and epsilon values. After fitting the model, the best SVR model was chosen based on cross-validation results. The performance of the evaluated model used R2 score, Mean Squared Error (MSE), and Mean Absolute Error (MAE), with metrics saved in a dataframe. The results are printed and a further two visuals are created, one using Plotly to compare actual vs. predicted values and another using Matplotlib for a scatter plot of the same comparison.

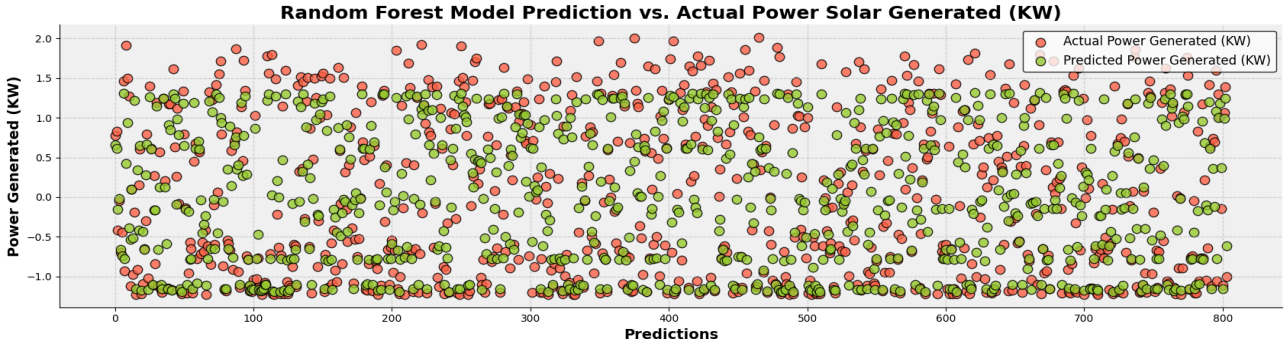
```
svr = SVR(max_iter=280)

param_grid_svr = {'kernel': ['linear', 'rbf', 'poly'], 'C': [0.1, 1, 10], 'epsilon': [0.01, 0.1, 1]}

grid_search_svr = GridSearchCV(svr, param_grid_svr, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

grid_search_svr.fit(X_train, y_train)

best_svr = grid_search_svr.best_estimator_

y_pred_svr = best_svr.predict(X_test)

metrics_df = pd.DataFrame(columns=["Model", "R2 Score (%)", "Mean Squared Error", "Mean Absolute Error"])
```



Random Forest

Hyperparameter optimization has been applied to a Random Forest Regressor using the GridSearchCV module. There were combinations of n_estimators and min_samples_split tested that produced the best model on cross-validation. Once the model was trained, its performance was evaluated in terms of R2 score, Mean Squared Error (MSE), and Mean Absolute Error (MAE) which were then added to a DataFrame. These results then were printed and two visualizations were created: one using Plotly to compare actual values to predicted values, another with Matplotlib for a scatter plot. The Random Forest model has achieved an R2 score of 77.65%, which is better than the SVR model.

```
rf = RandomForestRegressor(max_depth = 5,random_state=42)

param_grid_rf = {'n_estimators': [50, 100, 200], 'min_samples_split': [2, 5,
10]}

grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)

grid_search_rf.fit(X_train, y_train)

best_rf = grid_search_rf.best_estimator_

y_pred_rf = best_rf.predict(X_test)

r2_score_rf = round(r2_score(y_test,y_pred_rf) * 100, 2)

mean_sq_rf = mean_squared_error(y_test,y_pred_rf)

mean_ab_rf = mean_absolute_error(y_test,y_pred_rf)

metrics_df.loc[len(metrics_df)] = ["Random Forest", r2_score_rf, mean_sq_rf,
mean_ab_rf]
```

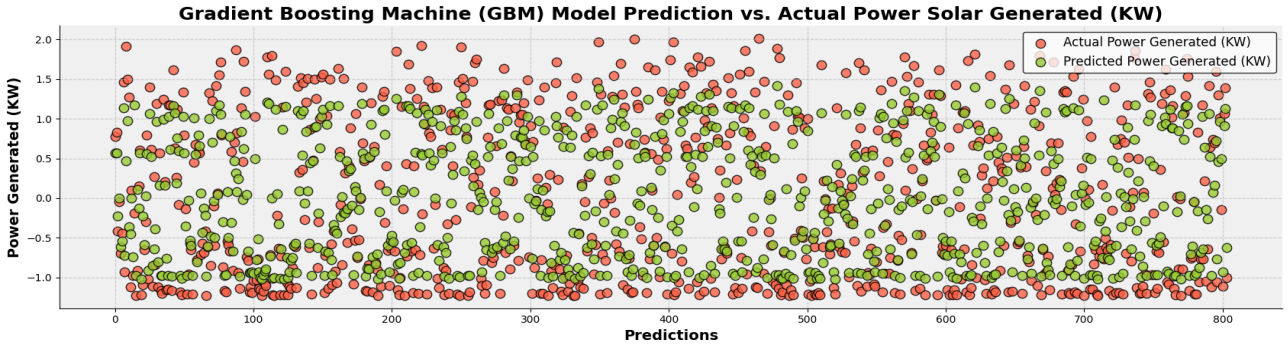


Gradient Boosting Machine (GBM)

The hyperparameter tuning is performed for the Gradient Boosting Regressor (GBM) model using GridSearchCV. Several combinations of learning_rate and max_depth are tested to find the best model as per cross-validation. Finally, the evaluated performance of the fitted model through R2 score, MSE, and MAE is recorded in a DataFrame. The printout is then shown, with two visualizations, one with Plotly comparing

actual vs. predicted values and another with Matplotlib for a scatter plot. The Gradient Boosting model was able to predict with an R2 score of 76.79%.

```
gbm = GradientBoostingRegressor(n_estimators = 8,random_state=42)

param_grid_gbm = {'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7]}

grid_search_gbm = GridSearchCV(gbm, param_grid_gbm, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)

grid_search_gbm.fit(X_train, y_train)

best_gbm = grid_search_gbm.best_estimator_

y_pred_gbm = best_gbm.predict(X_test)

r2_score_gbm = round(r2_score(y_test,y_pred_gbm) * 100, 2)

mean_sq_gbm = mean_squared_error(y_test,y_pred_gbm)

mean_ab_gbm = mean_absolute_error(y_test,y_pred_gbm)

metrics_df.loc[len(metrics_df)] = ["Gradient Boosting", r2_score_gbm,
mean_sq_gbm, mean_ab_gbm]
```



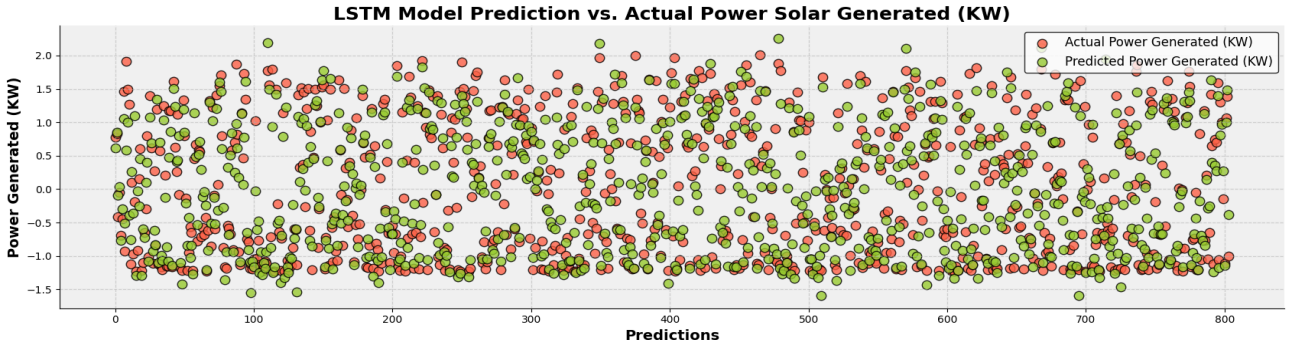
4.4 Deep Learning with LSTM

To encode it in the way of an LSTM input, code has been built for the application with all necessary properties that creates a three-dimensional array sample, time step, and features. The target variable is scaled through MinMaxScaler for normalized values ranging between 0 and 1. LSTM processes the two levels of LSTM where Dropout was added as a measure against overfitting and uses the Dense layer as a regression output. It is then compiled in ways using the Adam optimizer and mean squared error loss. Predictions are made after training and then rescaled into the original range. Performance can be characterized with R2 score, MSE, MAE, and demonstrated through graphs comparing the actual and predicted values against each other.

```
r2_score_lstm = round(r2_score(y_test,y_pred_lstm) * 100, 2)

mean_sq_lstm = mean_squared_error(y_test,y_pred_lstm)
```

```
mean_ab_lstm = mean_absolute_error(y_test,y_pred_lstm)

metrics_df.loc[len(metrics_df)]   =   ["LSTM",   r2_score_lstm,   mean_sq_lstm,
mean_ab_lstm]

print("R2 Score for LSTM Model: ",r2_score_lstm,"%")

print("Mean Square Error of LSTM Model: ",mean_sq_lstm)

print("Mean Absolute Error of LSTM Model: ",mean_ab_lstm)
```

```
R2 Score for LSTM Model:  80.24 %

Mean Square Error of LSTM Model:  0.20402779226884776

Mean Absolute Error of LSTM Model:  0.312377688764255
```

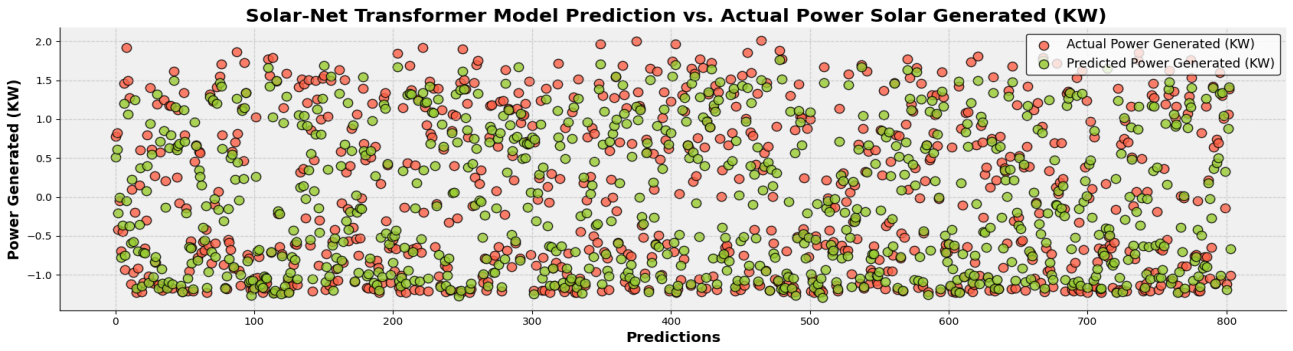


4.5 Transformer-Based Solar-Net Model

The code carries out a transformer-based model called Solar-Net for the prediction of solar generation. The data are reshaped and scaled for transformer input. The model is defined via the functional API of Keras and has a multi-head attention layer and a feed-forward network. Hyperparameter tuning is carried out through RandomSearch, optimizing the number of attention heads, feed-forward dimension, and dropout rate. Then the model is selected based on validation performance, and predictions are carried out on the test data. Performance evaluation metrics of R2 score, MSE, and MAE have been used. Comparison between actual and predicted values is modelled with visualizations.

```
mse_trans = mean_squared_error(y_test, y_pred_trans)

rmse_trans = np.sqrt(mean_squared_error(y_test, y_pred_trans))

r2_trans = r2_score(y_test, y_pred_trans)

print("Best Hyperparameters:")

print("Num Heads:", best_hyperparameters['num_heads'])
```

```
print("Feed-Forward Dim:", best_hyperparameters['ff_dim'])

print("Dropout Rate:", best_hyperparameters['dropout_rate'])

Best Hyperparameters:

Num Heads: 6

Feed-Forward Dim: 128

Dropout Rate: 0.1

#Evalute the Solar-Net Transformer model's performance

r2_score_solar = round(r2_score(y_test,y_pred_trans) * 100, 2)

mean_sq_solar = mean_squared_error(y_test,y_pred_trans)

mean_ab_solar = mean_absolute_error(y_test,y_pred_trans)

metrics_df.loc[len(metrics_df)] = ["Solar-Net Transformer", r2_score_solar,
mean_sq_solar, mean_ab_solar]

print("R2 Score for Solar-Net Transformer Model: ",r2_score_solar,"%")

print("Mean Square Error of Solar-Net Transformer Model: ",mean_sq_solar)

print("Mean Absolute Error of Solar-Net Transformer Model: ",mean_ab_solar)

R2 Score for Solar-Net Transformer Model:  81.54 %

Mean Square Error of Solar-Net Transformer Model:  0.19061348215622076

Mean Absolute Error of Solar-Net Transformer Model:  0.2898884608017335
```



5. Strengths and Weaknesses

Strengths:

- **High Accuracy**: Transformer model outperforms conventional techniques.
- **Scalability**: Hadoop tools (HDFS, Hive, Spark) enable distributed computing.
- **Feature Automation**: Less reliance on hand-crafted features.

Weaknesses:

- **Training Overhead**: Transformers are resource-intensive.
- **Interpretability**: Attention weights help, but models still function as black boxes.
- **Complexity**: More dependencies and infrastructure are needed.

6. Results and Comparison

Table exhibits the predictive efficiency of five machine learning designs for generating solar energy. The most effective is the Solar-Net Transformer with an R2 score of 81.54%, combined with the lowest Mean Squared Error (0.1906) and Mean Absolute Error (0.2899). Following it is the LSTM model, which scores an R2 of 80.24% with average error values (MSE=0.2040, MAE=0.3124) on all components. Random Forest scored 77.65% on R2, while Gradient Boosting scored at 76.79% based on the same model. SVM had the least with an R2 of 74.77%. Among the predicted machine learning models, Solar-Net Transformer provides optimum prediction accuracy and error metrics.

Model	**R2 Score (%)**	**MSE**	**MAE**
SVR	74.77	0.260527	0.406757
Random Forest	77.65	0.230744	0.326031
Gradient Boosting	76.79	0.239631	0.364436

LSTM	80.24	0.204028	0.312378
Solar-Net (Transformer)	81.54	0.190613	0.289888

7. Conclusion

The Solar-Net framework illustrates that transformer-based models achieve improved forecasting performance in solar irradiance prediction when used with distributed big data tools. After elaborate pre-processing, feature engineering, and model tuning, the framework ended up with best performance metrics as compared to traditional ML models and deep learning baselines like LSTM. This will lead into future works which will consider deploying the solution in real-time environments integrating external weather API data for dynamic retraining and predictions.

8. References

1. Kaggle Dataset: https://www.kaggle.com/datasets/stucom/solar-energy-power-generation-dataset
2. Apache Hadoop Documentation: https://hadoop.apache.org/
3. Apache Spark MLlib: https://spark.apache.org/mllib/
4. Keras Tuner: https://keras.io/keras_tuner/
5. Vaswani et al., “Attention is All You Need” (2017)
6. TensorFlow Documentation: https://www.tensorflow.org/