Project : Enhancing Solar Energy Forecasting with Machine Learning for Sustainable Energy Planning

Introduction Solar energy plays a pivotal role in the transition to sustainable and green energy sources. By harnessing the power of the sun, we can reduce our reliance on fossil fuels and mitigate the impact of climate change. Predicting solar yields accurately is crucial for optimizing energy production and planning. This project utilizes machine learning algorithms to predict solar yields, contributing to the efficient utilization of solar energy resources and promoting a greener, more sustainable energy future.

Importance of Solar Energy Solar energy is a clean, renewable energy source that can significantly reduce carbon emissions and combat climate change. By investing in solar energy technologies, we can reduce our dependence on fossil fuels and move towards a more sustainable energy mix. Accurate prediction of solar yields is essential for maximizing energy production efficiency, reducing costs, and minimizing environmental impact.

Project Benefits

- **Optimized Energy Production:** Accurate prediction of solar yields enables better planning and management of solar energy systems, leading to optimized energy production and reduced wastage.
- **Cost Reduction:** By predicting solar yields, energy companies can reduce operational costs and improve the overall efficiency of solar energy systems, making green energy more economically viable.
- **Environmental Impact:** Utilizing solar energy helps reduce greenhouse gas emissions, contributing to a cleaner and healthier environment and combating climate change.
- **Sustainability:** Solar energy is a sustainable energy source that can help meet the world's growing energy demands without depleting finite resources, ensuring a more sustainable future for generations to come.

This project demonstrates the potential of machine learning in optimizing solar energy production for a greener and more sustainable future. By accurately predicting solar yields, we can enhance the efficiency and sustainability of solar energy systems, paving the way for a cleaner, greener, and more sustainable energy future.

Project Objectives:

- **Develop machine learning models**: Create robust machine learning models for predicting solar yields based on historical data and weather patterns.
- **Enhance forecasting accuracy:** Improve the accuracy of solar energy forecasting by incorporating advanced machine learning algorithms and feature engineering techniques.
- **Optimize energy planning:**Provide insights and tools for energy planners to optimize the integration of solar energy into the power grid, leading to more efficient and sustainable energy use.

Project Methodology:

- **Data collection and preprocessing**:Gather historical solar energy production data, weather data, and other relevant variables for model training and testing.
- **Feature engineering**:Extract and engineer features from the data to improve the predictive power of the machine learning models.
- **Model development:** Develop machine learning models, such as Linear Regression, LSTM, and Deep Learning Regression, to predict solar energy yields based on the input features.
- **Model evaluation:** Evaluate the performance of the models using metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) on a test dataset.
- **Optimization:** Fine-tune the models and parameters to achieve the best possible forecasting accuracy.
- **Integration:**Integrate the machine learning models into existing solar energy forecasting systems or develop a standalone tool for energy planners.

EXPECTED OUTCOMES

Here are the expected outcomes for the project :

- **Improved accuracy:** The project aims to significantly improve the accuracy of solar energy forecasting compared to traditional methods.
- **Enhanced sustainability:** By providing more accurate forecasts, the project contributes to the efficient use of solar energy and supports sustainable energy planning.
- **Practical tools:** The project aims to deliver practical tools and insights that can be used by energy planners and policymakers to optimize energy planning and grid integration.

MACHINE LEARNING ALGORITHMS USED

Introduction

- The project aimed to predict solar yields using machine learning algorithms. Three algorithms were utilized: sklearn Linear Regression, Deep Learning Linear Regression using PyTorch, and LSTM Deep Learning using PyTorch. The analysis included examining the trend of the yields over time and exploring the correlation between AC power, DC power, daily yields, and total yields.

Description of Algorithms used :

Linear Regression:

- Sklearn's Linear Regression model is a simple linear approach to modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the dependent and independent variables.

Linear Regression Equation:

$Y = \beta 0 + \beta 1X + \beta 2X2 + ... + \beta nXn + \epsilon Y = \beta 0 + \beta 1\, X1 + \beta 2\, X2 + ... + \beta n\, Xn + \epsilon$

Deep Learning Linear Regression:

- This approach uses a deep learning model for linear regression, implemented using PyTorch. It involves a neural network with multiple layers to learn complex patterns in the data.It is Similar to linear regression, but with additional hidden layers and activation functions.

LSTM (Long Short-Term Memory):

- LSTM is a type of recurrent neural network (RNN) that is capable of learning long-term dependencies. It is particularly useful for sequential data like time series. LSTM has complex gating mechanisms to control the flow of information, allowing it to remember or forget information over long periods.

Preprocessing

- Standard Scaler was used to scale the data.
- Train Test Split was performed to split the data into training and testing sets.
- Y values were reshaped for deep learning models.
- Data was converted into tensors for processing with PyTorch.

Model Training

- ADAM optimizer was used for training the models.
- Mean Squared Error (MSE) was used as the loss function for model evaluation.

PROJECT WORKSPACE ::

In [ ]:
```
# importing the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
```

```python
import matplotlib.pyplot as plt
import torch
from torch import nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
```

In [3]:
```python
df = pd.read_csv("/Plant_1_Generation_Data.csv")
df_original = df.copy()
```

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 68778 entries, 0 to 68777
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   DATE_TIME    68778 non-null  object 
 1   PLANT_ID     68778 non-null  int64  
 2   SOURCE_KEY   68778 non-null  object 
 3   DC_POWER     68778 non-null  float64
 4   AC_POWER     68778 non-null  float64
 5   DAILY_YIELD  68778 non-null  float64
 6   TOTAL_YIELD  68778 non-null  float64
dtypes: float64(4), int64(1), object(2)
memory usage: 3.7+ MB
```

In [5]:
```python
df.shape
```

Out[5]:
```
(68778, 7)
```

In [6]:
```python
df.describe()
```

Out[6]:

	PLANT_ID	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD
count	68778.0	68778.000000	68778.000000	68778.000000	6.877800e+04
mean	4135001.0	3147.426211	307.802752	3295.968737	6.978712e+06
std	0.0	4036.457169	394.396439	3145.178309	4.162720e+05
min	4135001.0	0.000000	0.000000	0.000000	6.183645e+06
25%	4135001.0	0.000000	0.000000	0.000000	6.512003e+06
50%	4135001.0	429.000000	41.493750	2658.714286	7.146685e+06
75%	4135001.0	6366.964286	623.618750	6274.000000	7.268706e+06
max	4135001.0	14471.125000	1410.950000	9163.000000	7.846821e+06

In [7]:
```python
df.isnull().sum()
```

Out[7]:
```
DATE_TIME      0
PLANT_ID       0
SOURCE_KEY     0
DC_POWER       0
AC_POWER       0
DAILY_YIELD    0
TOTAL_YIELD    0
dtype: int64
```

In [ ]:
```
import datetime
df['DATE_TIME'] = pd.to_datetime(df['DATE_TIME'])
```

In [9]:
```
df.dtypes
```

Out[9]:
```
DATE_TIME      datetime64[ns]
PLANT_ID                int64
SOURCE_KEY             object
DC_POWER              float64
AC_POWER              float64
DAILY_YIELD           float64
TOTAL_YIELD           float64
dtype: object
```

In [10]:
```
df.drop(['SOURCE_KEY','PLANT_ID'],axis=1,inplace=True)
```

In [11]:
```
# renaming the columns to lower case letters
df= df.rename(columns={'DATE_TIME':'year','SOURCE_KEY':'source_key','DC_POWER':'dc_pow
                        'AC_POWER':'ac_power','DAILY_YIELD':'daily_yield','TOTAL_YIELD
            })
```

In [12]:
```
for column in df.columns:
    if column in ['year']:
        pass
    else :
        df[column] = np.round(df[column])
```

In [ ]:
```
df['dc_power'].head(20)
```

In [14]:
```
df.columns
```

Out[14]:
```
Index(['year', 'dc_power', 'ac_power', 'daily_yield', 'total_yield'], dtype='object')
```

In [15]:
```
df['year'].head()
```

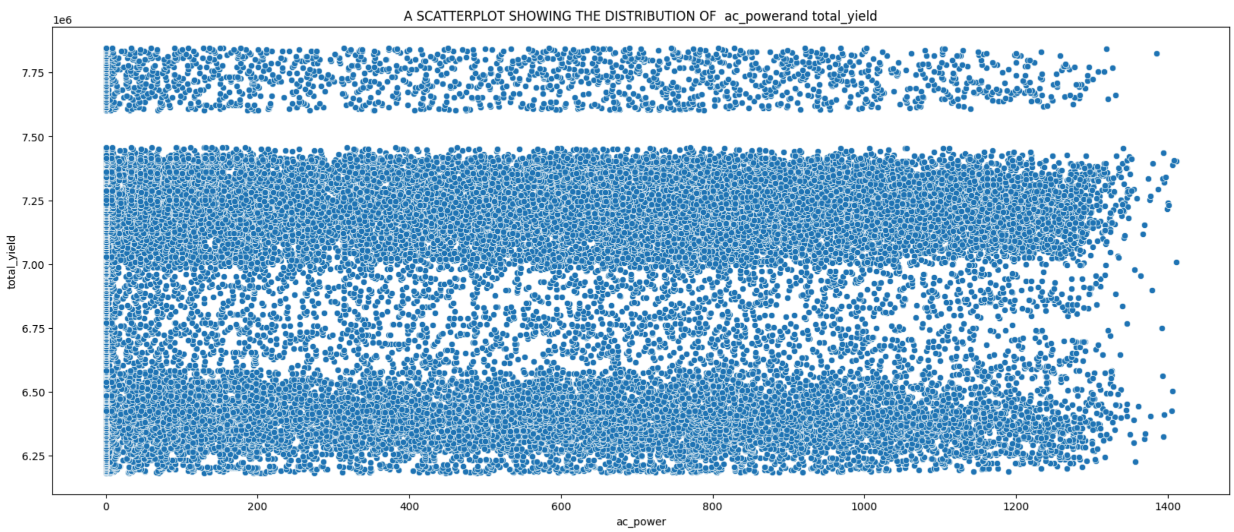
Out[15]:
```
0   2020-05-15
1   2020-05-15
2   2020-05-15
3   2020-05-15
4   2020-05-15
Name: year, dtype: datetime64[ns]
```

BASIC EDA

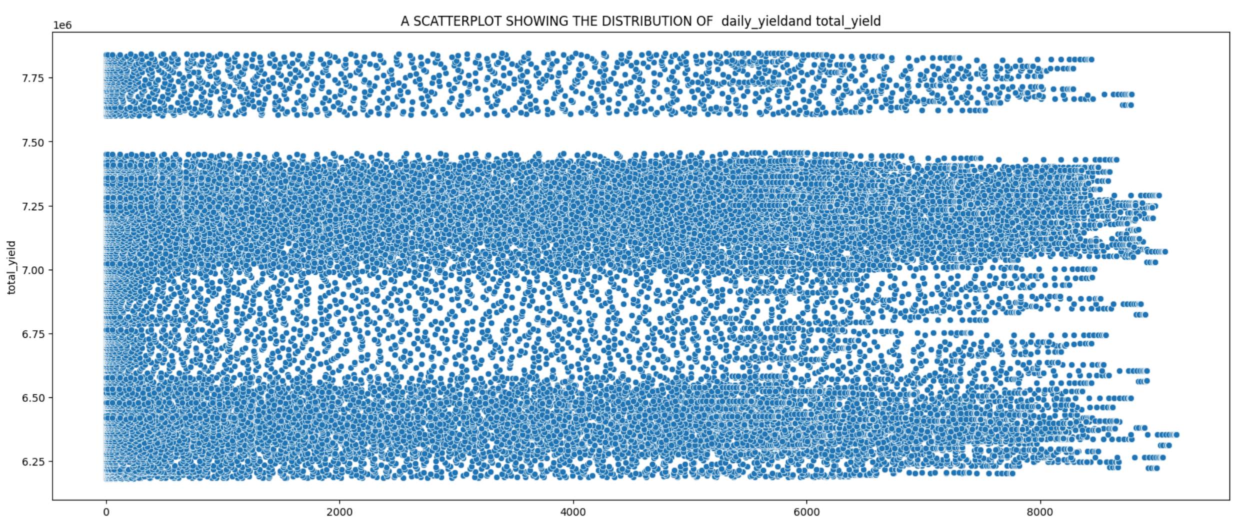
In [16]:
```
def  trend_overtime(column,color,date=df['year'],df=df):
    """This function aims at plotting the relationships between variable over time,the
    insight over the yields ,and it involves data visualization using seaborn and matp
```

```
    Args:
    column1 - the column we wish to analyse
    df : DataFrame that we are studying
    Returns :
    plots and visualizations]
    """
    fig = plt.figure(figsize=(20,8))
    sns.lineplot(y=column,x=date,color=color,data=df)
    plt.title(f'The distribution of {column} overtime')
    plt.show()
def compare_bivariately(column1,column2,data=df):
    fig=plt.figure(figsize=(20,8))
    sns.scatterplot(x=column1,y=column2,data=data)
    plt.xlabel(f"{column1}")
    plt.ylabel(f"{column2}")
    plt.title(f"A SCATTERPLOT SHOWING THE DISTRIBUTION OF  {column1}and {column2}")
    plt.show()
```

In [17]:
```
compare_bivariately("ac_power",'total_yield')
```

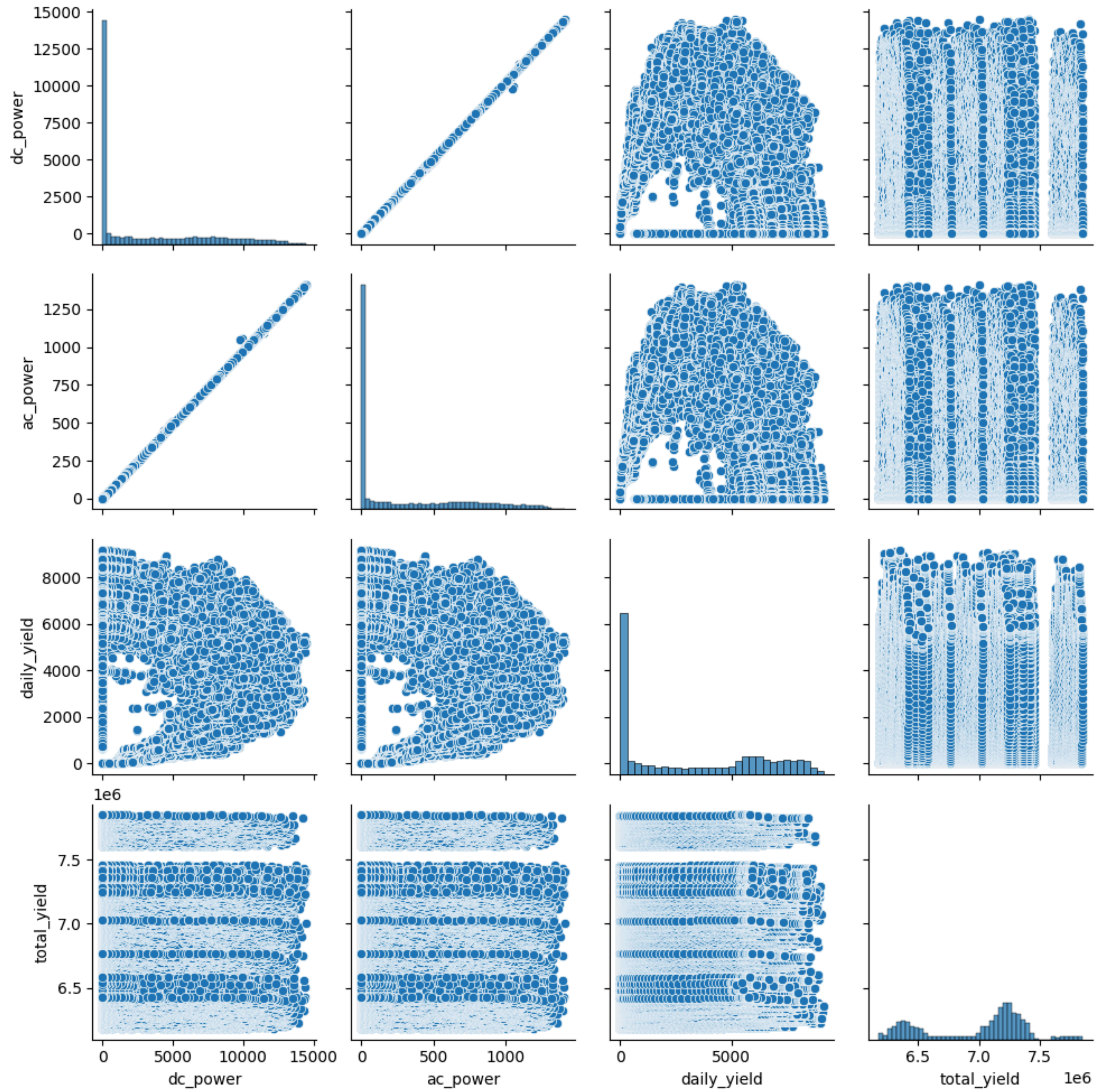


In [18]:
```
compare_bivariately("daily_yield",'total_yield')
```



In [19]:
```
sns.pairplot(df)
plt.show()
```

```
# The pairplot indicates that a linear relationship cannot be easily concluded
```



In [20]:

```
# Checking for  correlation between our variables
correlation = df.corr()
correlation # The total yield has correlations with the ac_power ,dc_power and so we c
```

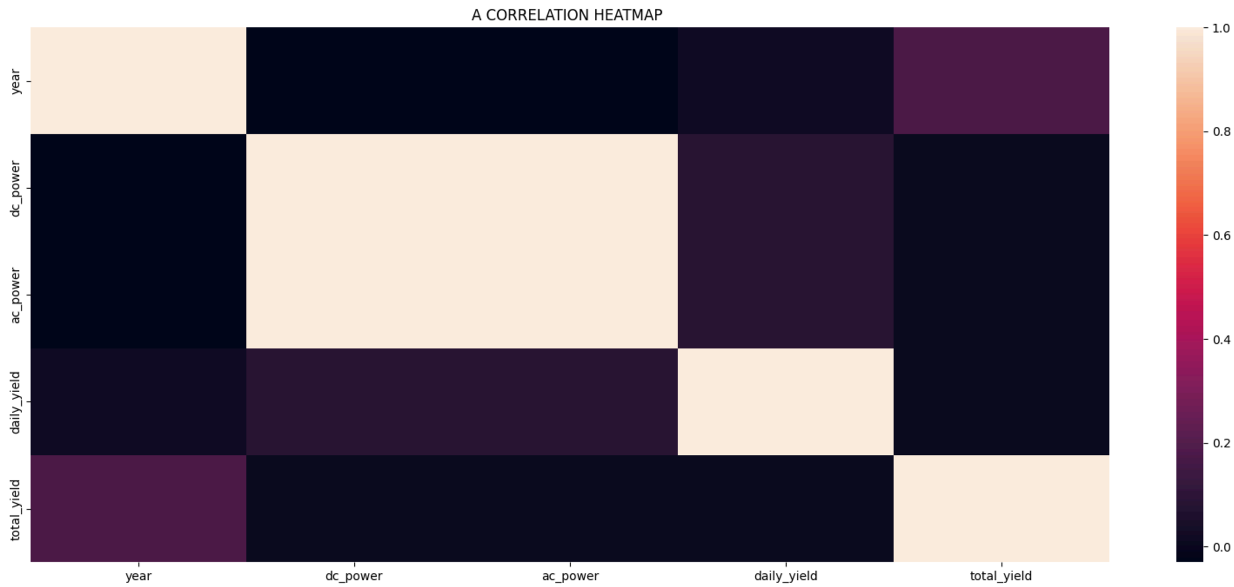
Out[20]:

	year	dc_power	ac_power	daily_yield	total_yield
year	1.000000	-0.029830	-0.029763	0.021383	0.172984
dc_power	-0.029830	1.000000	0.999996	0.082285	0.003815
ac_power	-0.029763	0.999996	1.000000	0.082235	0.003805
daily_yield	0.021383	0.082285	0.082235	1.000000	0.009867
total_yield	0.172984	0.003815	0.003805	0.009867	1.000000

In [21]:

```
plt.figure(figsize=(20,8))
sns.heatmap(correlation)
```

```
plt.title('A CORRELATION HEATMAP ')
plt.show()
```



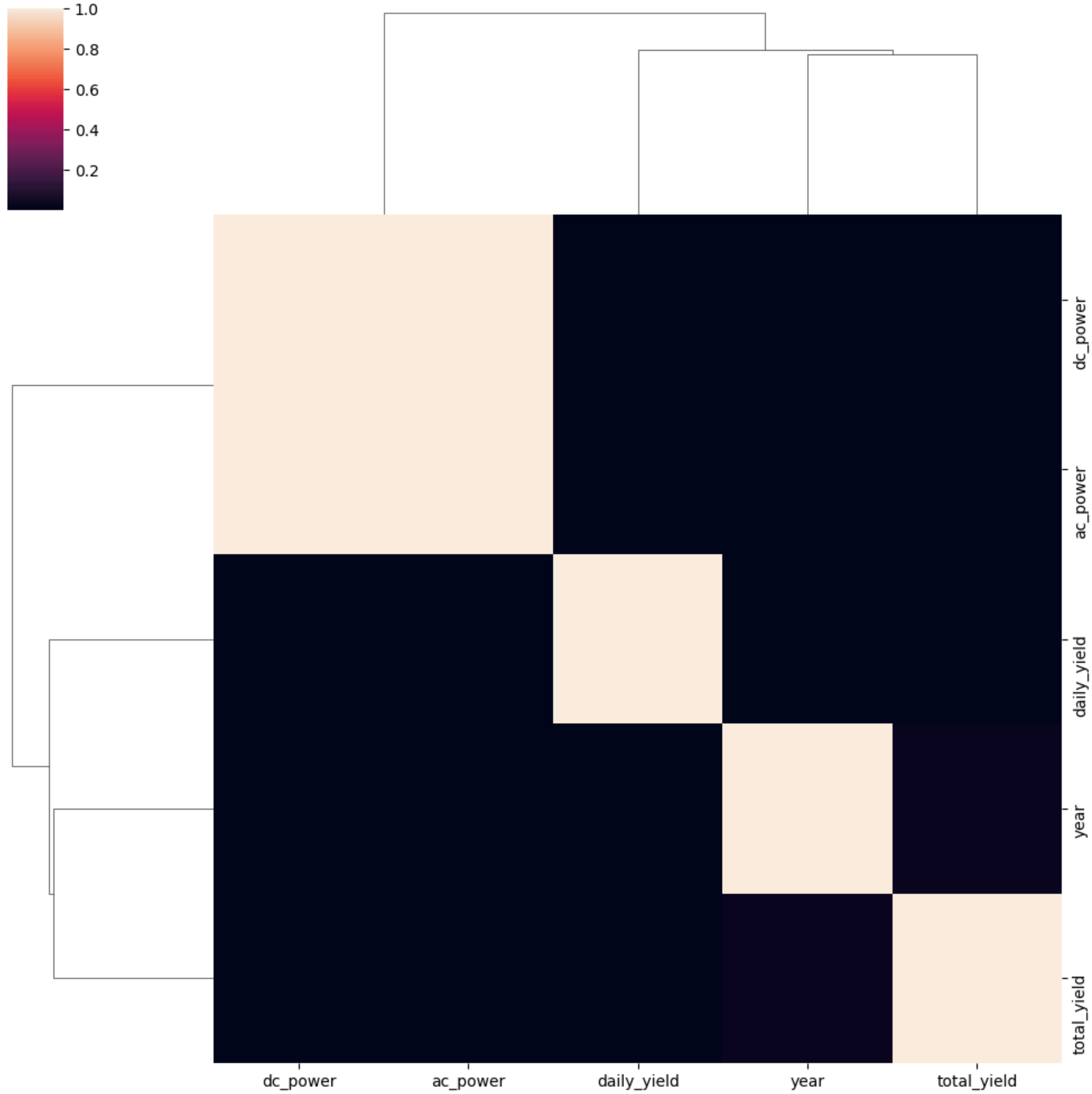
In [22]:
```
# Building on the intuition of CORRELATION how about R Squared
r_squared = correlation**2
r_squared # The R squared isnt that high for any of the variables
```

Out[22]:

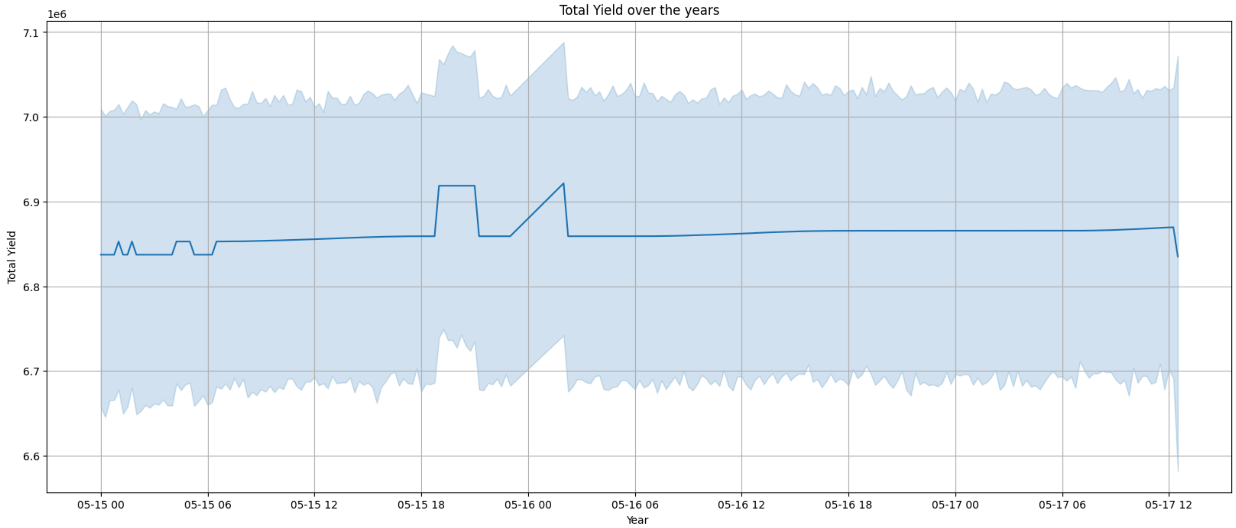
	year	dc_power	ac_power	daily_yield	total_yield
year	1.000000	0.000890	0.000886	0.000457	0.029923
dc_power	0.000890	1.000000	0.999992	0.006771	0.000015
ac_power	0.000886	0.999992	1.000000	0.006763	0.000014
daily_yield	0.000457	0.006771	0.006763	1.000000	0.000097
total_yield	0.029923	0.000015	0.000014	0.000097	1.000000

In [25]:
```
sns.clustermap(r_squared)
```

Out[25]:
```
<seaborn.matrix.ClusterGrid at 0x7b2da6be6c20>
```



In [26]:
```
# Learning the trajectories of total yield
fig = plt.figure(figsize=(20,8))
sns.lineplot(x='year',y = 'total_yield',data=df.head(5000))
plt.xlabel('Year ')
plt.ylabel('Total Yield')
plt.title('Total Yield over the years')
plt.grid()
plt.show()
```



In [27]:
```
# We can use the variance to explain whats happening in the above plot
total_yield =df['total_yield']
total_yield.describe() # The standard deviation is so small
```

Out[27]:
```
count    6.877800e+04
mean     6.978712e+06
std      4.162720e+05
min      6.183645e+06
25%      6.512002e+06
50%      7.146685e+06
75%      7.268706e+06
max      7.846821e+06
Name: total_yield, dtype: float64
```

In [28]:
```
df.columns
```

Out[28]:
```
Index(['year', 'dc_power', 'ac_power', 'daily_yield', 'total_yield'], dtype='object')
```

STANDARD SCALER : TEXT PREPROCESSING

In [29]:
```
scaler = MinMaxScaler()
X = df.drop(['year','total_yield'],axis=1).values
y = df['total_yield'].values.reshape(-1,1)
print(f"The shape of our labels is{X.shape}")
print(f"The shape of our target features is {y.shape}")
```

```
The shape of our labels is(68778, 3)
The shape of our target features is (68778, 1)
```

In [30]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [31]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
y_test = scaler.fit_transform(y_test)
y_train = scaler.fit_transform(y_train)
```

In [32]:
```
X_train.shape,y_train.shape
```

Out[32]:
```
((55022, 3), (55022, 1))
```

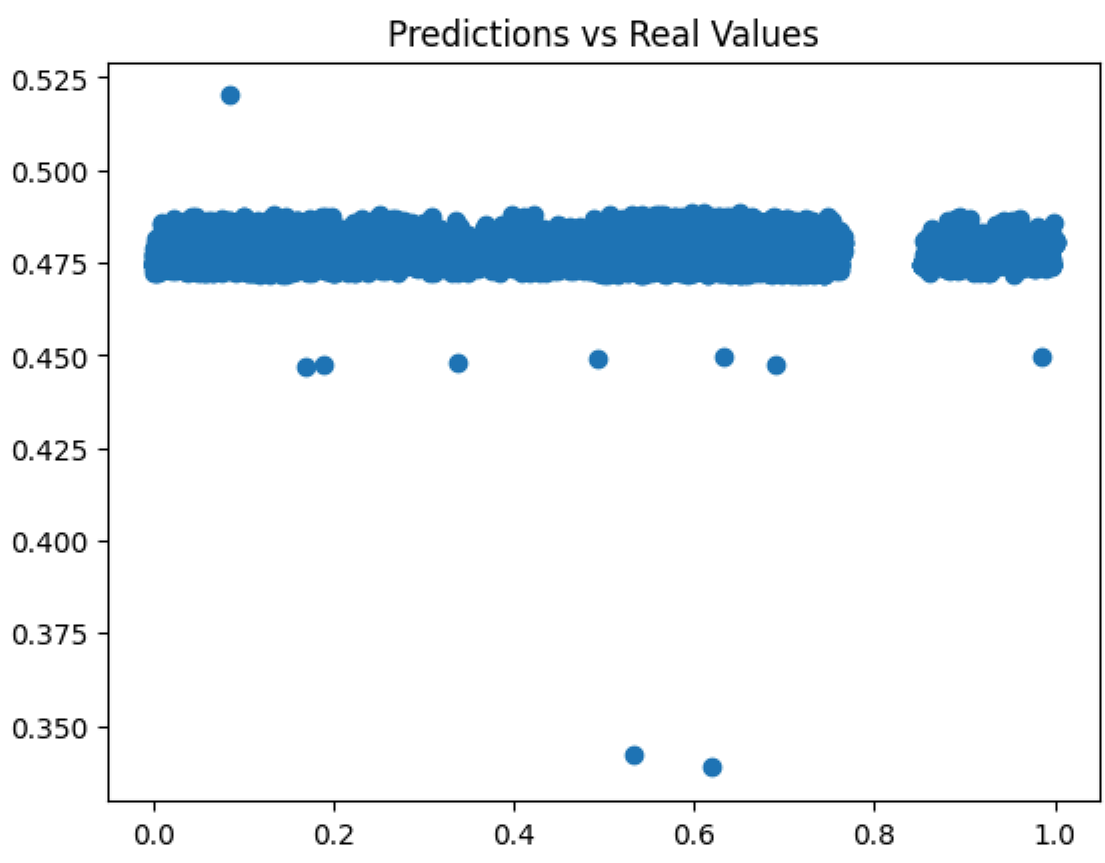
In [33]:
```
assert len(X_train) + len(X_test) == len(df)
```

Linear Regression : MACHINE LEARNING

In [34]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error
lr = LinearRegression()
lr.fit(X_train,y_train)
lr_prediction  = lr.predict(X_test)
print(f"The Mean Absolute Error for Linear Model is {mean_absolute_error(y_test,lr_pre
print(f"The Mean Squared Error for Linear Model is {mean_squared_error(y_test,lr_predi
```

```
The Mean Absolute Error for Linear Model is 0.21745585379369614
The Mean Squared Error for Linear Model is 0.06264275217419242
```

In [35]:
```
# plotting the residuals vs the real values
#fig = plt.figure(figsize=(15,8))
plt.scatter(y_test,lr_prediction)
plt.title('Predictions vs Real Values')
plt.show()
```



In [36]:
```
print(scaler.inverse_transform(lr_prediction))
```

```
[[6983524.54234887]
 [6973149.05263772]
 [6982520.64589477]
 ...
 [6968871.6479323 ]
 [6981329.5518485 ]
 [6973149.05263772]]
```

DEEP LEARNING LINEAR_REGRESSION

In [37]:
```
# We convert the numpy arrays into tensors
X_train = torch.from_numpy(X_train).type(torch.Tensor)
X_test = torch.from_numpy(X_test).type(torch.Tensor)
y_train = torch.from_numpy(y_train).type(torch.Tensor)
y_test = torch.from_numpy(y_test).type(torch.Tensor)
```

In [38]:
```
type(X_train)
```

Out[38]:
```
torch.Tensor
```

In [39]:
```
X_train.shape,y_train.shape
```

Out[39]:
```
(torch.Size([55022, 3]), torch.Size([55022, 1]))
```

In [40]:
```
# Creating our Model using Pytorch
class SolarYields(nn.Module):
    def __init__(self,input_dim=3,hidden_dim=27,output_dim=1,p=0.4):
        super().__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        # Create 2 Linear layers and a dropout layer
        self.linear1 = nn.Linear(input_dim,hidden_dim)
        self.linear2 = nn.Linear(hidden_dim,hidden_dim)
        self.fc = nn.Linear(hidden_dim,output_dim)
        self.dropout = nn.Dropout(p) # a dropout layer takes care of  overfitting prob
    def forward(self,x):
        x = self.linear1(x)
        #x = F.relu(x) # passing a linear activation function
        x = self.linear2(x)
        x = self.dropout(self.linear2(x))
        x = self.fc(x)
        return x
```

In [41]:
```
model = SolarYields()
```

In [42]:
```
scaler.inverse_transform(model.forward(X_train).detach().numpy())
```

Out[42]:
```
array([[5820944.5],
       [6178944.5],
       [5805814. ],
       ...,
       [6034201. ],
       [5862963. ],
       [5843696.5]], dtype=float32)
```

In [43]:
```
from sklearn.metrics import accuracy_score
```

Deep Linear Regression Model Training

In [46]:
```
# Optimizer and Loss Function
optimizer = optim.SGD(model.parameters(),lr = 0.001)
criterion = nn.MSELoss()
#accuracy = accuracy_score()
```

In [47]:
```
epochs = 100
for epoch in range(epochs):
    model.train()
    y_pred = model(X_train)
    loss = criterion(y_pred,y_train)
    #train_accuracy = accuracy_score(y_pred.detach().numpy(),y_train.detach().numpy())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    model.eval()
    with torch.no_grad():
        y_pred = model(X_test)
        test_loss = criterion(y_pred,y_test)
        #test_accuracy = accuracy_score(y_pred.detach().numpy(),y_test.detach().numpy(
        if epoch % 10 == 0:
            print(f"Epoch{epoch}|train loss{loss}||test loss{test_loss}")
```

```
Epoch0|train loss0.5020686388015747||test loss0.48668527603149414
Epoch10|train loss0.4400651454925537||test loss0.4250389337539673
Epoch20|train loss0.38638603687286377||test loss0.3723895251750946
Epoch30|train loss0.34100577235221863||test loss0.3272905647754669
Epoch40|train loss0.3014690577983856||test loss0.2885824143886566
Epoch50|train loss0.26860177516937256||test loss0.25535574555397034
Epoch60|train loss0.2398761510848999||test loss0.22683997452259064
Epoch70|train loss0.21458013355731964||test loss0.20237454771995544
Epoch80|train loss0.19429287314414978||test loss0.18142545223236084
Epoch90|train loss0.1765342354774475||test loss0.16353215277194977
```

In [48]:
```
torch.manual_seed(42)
deep_regression_preds = scaler.inverse_transform(model.forward(X_train).detach().numpy
deep_regression_preds
```

Out[48]:
```
array([[6461812.5],
       [6460682. ],
       [6540890.5],
       ...,
       [6501273. ],
       [6510020.5],
       [6521758.5]], dtype=float32)
```

In [49]:
```
deep_regression_preds.squeeze(1)
```

Out[49]:
```
array([6461812.5, 6460682. , 6540890.5, ..., 6501273. , 6510020.5,
       6521758.5], dtype=float32)
```

LSTM : USING RNNS TO PREDICT THE FUTURE YIELD

In [50]:

```
class SolarLSTM(nn.Module):
    def __init__(self,input_dim=1,hidden_dim=64,n_layers=2,output_dim=1,p=0.4):
        super().__init__()
        #self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        #self.output_dim = output_dim
        self.n_layers = n_layers
        self.rnn = nn.LSTM(input_dim,hidden_dim,n_layers,dropout=p)
        self.dropout = nn.Dropout(p)
        self.fc = nn.Linear(hidden_dim,output_dim)
    def forward(self,x:torch.Tensor):
        # instantiate the current and hidden cell states
        h0 = torch.zeros(self.n_layers,x.size(1),self.hidden_dim).requires_grad_()
        c0 = torch.zeros(self.n_layers,x.size(1),self.hidden_dim).requires_grad_()
        output,(hn,cn) = self.rnn(x,(h0.detach().squeeze(1),c0.detach().squeeze(1)))
        output = self.fc(output)

        return output
```

In [51]:

```
lstm_model = SolarLSTM()
```

In [52]:

```
torch.manual_seed(42)
scaler.inverse_transform(lstm_model.forward(y_train).detach().numpy())
```

Out[52]:

```
array([[5998880. ],
       [5995208. ],
       [6009558.5],
       ...,
       [6015136. ],
       [6005220.5],
       [6005518.5]], dtype=float32)
```

LSTM MODEL TRAINING

In [53]:

```
epochs =100
for epoch in range(epochs):
    lstm_model.train()
    y_pred = lstm_model(y_train)
    loss = criterion(y_pred,y_train)
    #train_accuracy = accuracy_score(y_pred.detach().numpy(),y_train.detach().numpy())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    lstm_model.eval()
    with torch.no_grad():
        y_pred = lstm_model(y_test)
        lstm_loss = criterion(y_pred,y_test)
        #test_accuracy = accuracy_score(y_pred.detach().numpy(),y_test.detach().numpy(
        if epoch % 10 == 0:
            print(f"Epoch{epoch}|train loss{loss}||test loss{lstm_loss}")
```

```
Epoch0|train loss0.3987686038017273||test loss0.3988194465637207
Epoch10|train loss0.39868029952049255||test loss0.3988194465637207
Epoch20|train loss0.39873918890953064||test loss0.3988194465637207
Epoch30|train loss0.398694783449173||test loss0.3988194465637207
Epoch40|train loss0.39873671531677246||test loss0.3988194465637207
Epoch50|train loss0.39874500036239624||test loss0.3988194465637207
Epoch60|train loss0.3987261950969696||test loss0.3988194465637207
Epoch70|train loss0.39868220686912537||test loss0.3988194465637207
Epoch80|train loss0.3987462520599365||test loss0.3988194465637207
Epoch90|train loss0.3986998200416565||test loss0.3988194465637207
```

Findings

- Linear Regression had a mean squared error of 0.0626.
- Deep Learning Linear Regression had a training loss of 0.1765 and a test loss of 0.1635.
- LSTM had a training loss of 0.3987 and a test loss of 0.3988.

Conclusion

- The best overall model for predicting solar yields was the sklearn Linear Regression model, which had the lowest mean squared error compared to the other models. However, further optimization and tuning may be required to improve the performance of the deep learning models.This project showcases the potential of machine learning in optimizing solar energy production.
- By accurately predicting solar yields, we can enhance the efficiency and sustainability of solar energy systems, contributing to a greener and more sustainable future.

Feel free to review ,fork and contribute