**Symbiosis Skills and Professional University Kiwale, Pune**



**PROJECT REPORT**

**On**

“Solar Radiation Prediction Analysis”



A PROJECT REPORT SUBMITTED BY,

Mr. Swapnil Taware. Mr. Anup Yelpale. Mr. Amit Yadav. Mr. Nishant Gaikwad.

FST 2.0 ML 2

Under The Guidance of

Trainers’s Name: Mrs. Amrita Helwade

**STUDENT DECLARATION AND ATTESTATION BY TRAINER**

This is to declare that this report has been written by me. No part of the report is plagiarized from other sources. All information included from other sources have been duly acknowledged. I aver that if any part of the report is found to be plagiarized, I shall take full responsibility for it.

Signature of student

**Name of student:**

1. Swapnil Taware.
2. Anup Yelpale.
3. Amit Yadav.
4. Nishant Gaikwad.

Signature of trainer

**Name of trainer:** Mrs. Amrita Helwade

**CERTIFICATE**

This is to certify that the report entitled, “” submitted by “Swapnil Taware, Anup Yelpale Amit Yadav, Nishant Gaikwad to Symbiosis Skills and Professional University, Pune, Maharashtra, India, is a record of bonafide Project work carried out by him under my supervision and guidance and is worthy of consideration for the completion of certificate course in ‘Machine Learning Engineer”.

Signature of Trainer

Name of Trainer: Mrs. Amrita Helwade

Date: 18/09 / 2022

Supervisor Supervisor

+

**ACKNOWLEDGEMENTS**

We have satisfaction upon completion of this project work entitled “Solar Radiation Prediction Analysis” to Symbiosis Skills and Professional University, Pune, Maharashtra, India.

We take this opportunity to express the sense of gratitude to Mrs. Amrita Helwade for valuable guidance and help provided us during course of completion of project.

We would like to thank Mrs. Amrita Helwade for giving us such a wonderful opportunity to expand our knowledge for our own creolized of what we study for.

We would like to thank our friends who helped us to make our work more organized and well stacked till the end.

Last but not least, we would thank the **Symbiosis Skills and Professional University, Kiwale and JPM** for giving us this great opportunity and help us to complete our report on time.

Our thanks and appreciation also go to the all my supporters.

**INDEX**

|  |  |  |
| --- | --- | --- |
| Sr no | Title | Pg no |
| 1 | Introduction | 6 |
| 2 | Objective | 6 |
| 3 | Methodology   1. Descriptive Analysis | 7 |
| 4 | Graphs/Plots   1. Scatter 2. Pairsplots 3. Heatmap | 9  9  10  11-12 |
| 5 | Regression And Classification Analysis   1. Linear Regression 2. Time Series 3. Clustering | 13  13  14  14 |
| 6 | Conclusion | 15 |
| 7 | References | 15 |

**Introduction**

An electrical operator should ensure a precise balance between the electricity production and consumption at any moment. This is often very difficult to maintain with conventional and controllable energy production system, mainly in small or not interconnected (isolated) electrical grid

(as found in islands). Many countries nowadays consider using renewable energy sources into their electricity grid. This creates even more problems as the resource (solar radiation, wind, etc.) is not steady. It is therefore very important to be able to predict the solar radiation effectively especially in case of high energy integration [1].

**Objective**

The aim is to create a model for the prediction of solar radiation that uses past weather data including temperatures, wind speed, air and moisture, and forecasts potential solar irradiance. This model can help the grid operators for improved supply and demand management

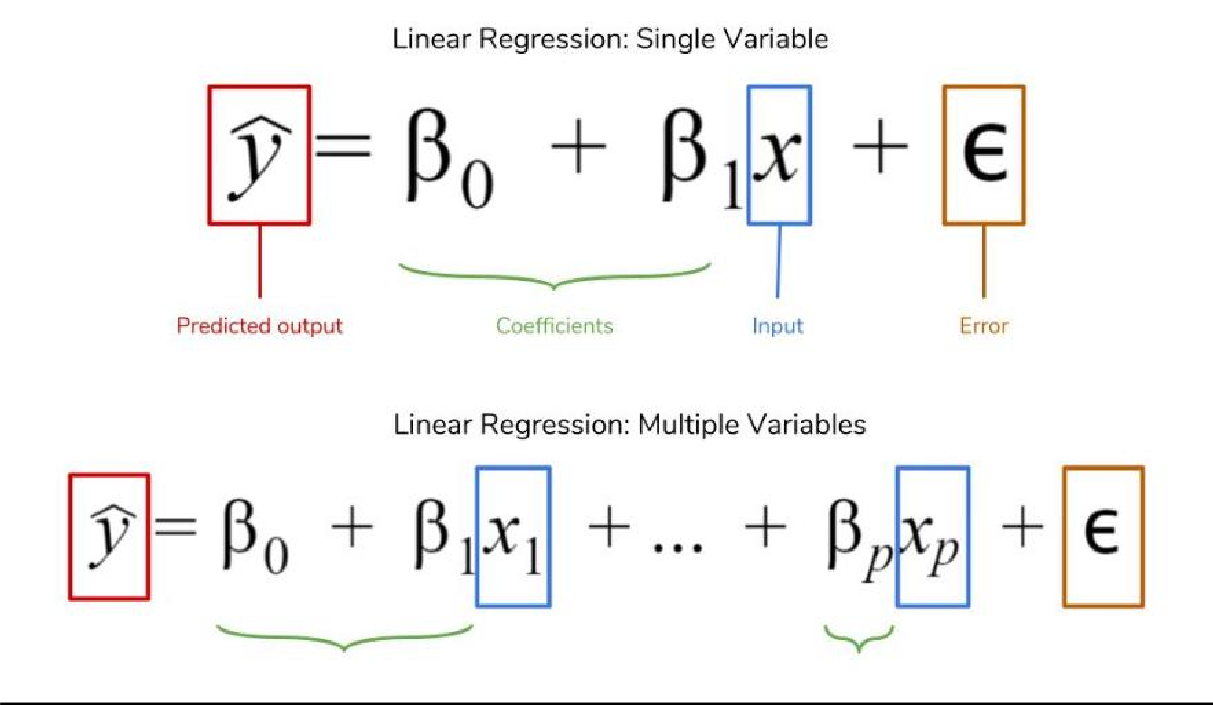
**Description:-**

Solar Prediction Analysis : "Analyzing Solar radition based on Temperature,Pressure,Humidity,WindDirection,Speed,SunRise & SunSet Time

1. Cleaning and Preprocessing the Solar Prediction Data.
2. Done Feature Engineering on data and drop unnecessary column. Plot the Pairplot w.r.t Radiation vs Temperature,Pressure,Huminity.
3. To Plot Heat map we splitting and scaling the data and we plot the Box plot graph for finding out the outliers in the dataset.
4. Implementation of ML algorithm for classification of data done Train\_test split & prediction( linear Regression).
5. Implementation of Boosting Technique (XGBOOST)
6. Time series analysis on Radiation and Datetime ,Month.Implementation of Seasonal Decomposition, SARIMAX Model.
7. Clustering(K-Means and Hierarchical Clustering).
8. Evaluation of the models.

**Linear Regression:-**

A linear regression is a data plot that graphs the linear relationship between an independent and a dependent variable(s). It is typically used to visually show the strength of the relationship, and the dispersion of results.



**Time Series Algoritm:-**

Time series is a machine learning technique that forecasts target value based solely on a known history of target values. It is a specialized form of regression, known in the literature as auto-regressive modeling.

The input to time series analysis is a sequence of target values. A case id column specifies the order of the sequence. The case id can be of

type NUMBER or a date type (date, datetime, timestamp with timezone, or timestamp with local timezone). Regardless of case id type, the user can request that the model include trend, seasonal effects or both in its forecast computation. When the case id is a date type, the user must specify a time interval (for example, month) over which the target values are to be aggregated, along with an aggregation procedure (for example, sum). Aggregation is performed by the algorithm prior to constructing the model.

The time series model provide estimates of the target value for each step of a time window that can include up to 30 steps beyond the historical data. Like other regression models, time series models compute various statistics that measure the goodness of fit to historical data.

Forecasting is a critical component of business and governmental decision making. It has applications at the strategic, tactical and operation level. The following are the applications of forecasting:

* Projecting return on investment, including growth and the strategic effect of innovations
* Addressing tactical issues such as projecting costs, inventory requirements and customer satisfaction
* Setting operational targets and predicting quality and conformance with standards

**Clustering**

Clustering is a classic data mining technique based on machine learning that divides groups

of abstract objects into classes of similar objects.

Clustering helps to split data into several subsets. Each of these clusters consists of data

objects with high inter-similarity and low intra-similarity

**Source of Data**

We collected this data from website: Kaggle

**Technology Used**

* 1. Python
  2. Preprocessing
  3. Numpy
  4. Pandas
  5. Matplotlib, Seaborn
  6. Sklearn
  7. XGBoost
  8. Time Series
  9. Clustering
  10. Linear Regression

**Conclusion:-**

* After implement the model we have seen that it gives very less accuracy up to 63.82
* We used XGBoost to boost the model and after that it gives 94.06 % accuracy which is very good
* In Time series analysis we found that data is stationary.

**ML Model:-**

Firstly we have imported all the Libraries and packages we'll require for this project.

In [1]:

import pandas as pd import numpy as np import pylab as pl import seaborn as sns

import matplotlib.pyplot as plt from matplotlib.pyplot import \* import time

from sklearn.preprocessing import StandardScaler from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error from sklearn import metrics

import xgboost as xgb import optuna

import sklearn.cluster as cluster from sklearn.cluster import KMeans from sklearn.decomposition import PCA

from scipy.cluster.hierarchy import dendrogram, linkage import scipy.cluster.hierarchy as shc

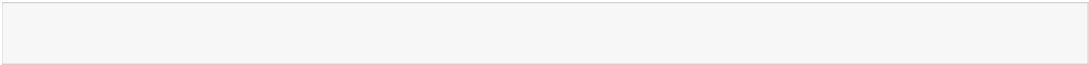
from sklearn.cluster import AgglomerativeClustering import statsmodels.api as sm

from statsmodels.tsa.arima.model import ARIMA



We are importing our data to be analysed.

In [2]:



solar\_radiation = pd.read\_csv(r"C:\Users\Swapnil\Downloads\downloaded project datasets\So larPrediction.csv")

In [3]:



data = solar\_radiation

In [4]:



df = data

In [5]:



pd.options.display.max\_rows=15000

The units of each dataset are:

Solar radiation: watts per meter^2 Temperature: degrees Fahrenheit Humidity: percent

Barometric pressure: Hg Wind direction: degrees Wind speed: miles per hour

Sunrise/sunset: "Hawaii" time

In [6]:



data.head()

Out[6]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **UNIXTime** | **Data** | **Time** | **Radiation** | **Temperature** | **Pressure** | **Humidity** | **WindDirection(Degrees)** | **Speed** | **TimeSunRise** |
|  | 9/29/2016 |  |  |  |  |  |  |  |  |
| **0** 1475229326 | 12:00:00 | 23:55:26 | 1.21 | 48 | 30.46 | 59 | 177.39 | 5.62 | 06:13:00 |
|  | AM |  |  |  |  |  |  |  |  |
|  | 9/29/2016 |  |  |  |  |  |  |  |  |
| **1** 1475229023 | 12:00:00 | 23:50:23 | 1.21 | 48 | 30.46 | 58 | 176.78 | 3.37 | 06:13:00 |
|  | AM |  |  |  |  |  |  |  |  |
|  | 9/29/2016 |  |  |  |  |  |  |  |  |
| **2** 1475228726 | 12:00:00 | 23:45:26 | 1.23 | 48 | 30.46 | 57 | 158.75 | 3.37 | 06:13:00 |
|  | AM |  |  |  |  |  |  |  |  |
|  | 9/29/2016 |  |  |  |  |  |  |  |  |
| **3** 1475228421 | 12:00:00 | 23:40:21 | 1.21 | 48 | 30.46 | 60 | 137.71 | 3.37 | 06:13:00 |
|  | AM |  |  |  |  |  |  |  |  |
|  | 9/29/2016 |  |  |  |  |  |  |  |  |
| **4** 1475228124 | 12:00:00 | 23:35:24 | 1.17 | 48 | 30.46 | 62 | 104.95 | 5.62 | 06:13:00 |
|  | AM |  |  |  |  |  |  |  |  |







We are using shape function for checking attributes and features of data.

In [7]:



data.shape

Out[7]:

(32686, 11)

our data has 32686 attributes(rows) and 11 features(columns)

In [8]:



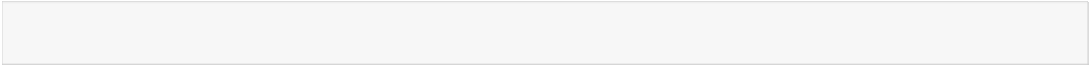
data.columns

Out[8]:

Index(['UNIXTime', 'Data', 'Time', 'Radiation', 'Temperature', 'Pressure', 'Humidity', 'WindDirection(Degrees)', 'Speed', 'TimeSunRise', 'TimeSunSet'],

dtype='object')

In [7]:



#data.dtypes

data.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 32686 entries, 0 to 32685 Data columns (total 11 columns):

# Column Non-Null Count Dtype

1. UNIXTime 32686 non-null int64
2. Data 32686 non-null object
3. Time 32686 non-null object
4. Radiation 32686 non-null float64
5. Temperature 32686 non-null int64
6. Pressure 32686 non-null float64
7. Humidity 32686 non-null int64
8. WindDirection(Degrees) 32686 non-null float64
9. Speed 32686 non-null float64
10. TimeSunRise 32686 non-null object
11. TimeSunSet 32686 non-null object dtypes: float64(4), int64(3), object(4)

memory usage: 2.7+ MB

In [8]:



data.describe()

Out[8]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **UNIXTime** | **Radiation** | **Temperature** | **Pressure** | **Humidity** | **WindDirection(Degrees)** | **Speed** |
| **count** | 3.268600e+04 | 32686.000000 | 32686.000000 | 32686.000000 | 32686.000000 | 32686.000000 | 32686.000000 |
| **mean** | 1.478047e+09 | 207.124697 | 51.103255 | 30.422879 | 75.016307 | 143.489821 | 6.243869 |
| **std** | 3.005037e+06 | 315.916387 | 6.201157 | 0.054673 | 25.990219 | 83.167500 | 3.490474 |
| **min** | 1.472724e+09 | 1.110000 | 34.000000 | 30.190000 | 8.000000 | 0.090000 | 0.000000 |
| **25%** | 1.475546e+09 | 1.230000 | 46.000000 | 30.400000 | 56.000000 | 82.227500 | 3.370000 |
| **50%** | 1.478026e+09 | 2.660000 | 50.000000 | 30.430000 | 85.000000 | 147.700000 | 5.620000 |
| **75%** | 1.480480e+09 | 354.235000 | 55.000000 | 30.460000 | 97.000000 | 179.310000 | 7.870000 |
| **max** | 1.483265e+09 | 1601.260000 | 71.000000 | 30.560000 | 103.000000 | 359.950000 | 40.500000 |

Checking null values in the data set

In [11]:



print("Total missing values:", data.isna().sum())

Total missing values: UNIXTime 0

Data 0

Time 0

Radiation 0

Temperature 0

Pressure 0

Humidity 0

WindDirection(Degrees) 0

Speed 0

TimeSunRise 0

TimeSunSet 0

dtype: int64

## Feature Engineering

In [14]:



data['Month'] = pd.to\_datetime(data.Data).dt.month.astype(np.int)

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/3275581446.py:1: DeprecationWarning: ` np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` b y itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, y ou may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['Month'] = pd.to\_datetime(data.Data).dt.month.astype(np.int)

In [15]:



data['Day'] = pd.to\_datetime(data.Data).dt.day.astype(np.int)

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/234670999.py:1: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['Day'] = pd.to\_datetime(data.Data).dt.day.astype(np.int)

In [16]:



data['Year'] = pd.to\_datetime(data.Data).dt.year.astype(np.int)

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/1511827949.py:1: DeprecationWarning: ` np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` b y itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, y ou may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['Year'] = pd.to\_datetime(data.Data).dt.year.astype(np.int)

In [17]:



data = data.drop('Data', axis=1)

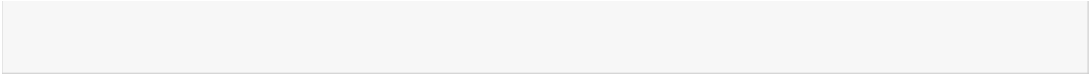
In [18]:



##data.drop(['day'],axis = 1, inplace = True)

In [19]:

data['Hour'] = pd.to\_datetime(data.Time).dt.hour.astype(np.int) data['Minute'] = pd.to\_datetime(data.Time).dt.minute.astype(np.int) data['Second'] = pd.to\_datetime(data.Time).dt.second.astype(np.int)



C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/360697051.py:1: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['Hour'] = pd.to\_datetime(data.Time).dt.hour.astype(np.int) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/360697051.py:2: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['Minute'] = pd.to\_datetime(data.Time).dt.minute.astype(np.int) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/360697051.py:3: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

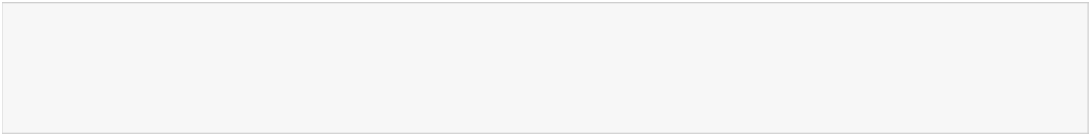
data['Second'] = pd.to\_datetime(data.Time).dt.second.astype(np.int)

In [20]:



data = data.drop('Time', axis=1)

In [21]:



data['SunriseHour'] = pd.to\_datetime(data.TimeSunRise).dt.hour.astype(np.int) data['SunriseMinute'] = pd.to\_datetime(data.TimeSunRise).dt.minute.astype(np.int)

data['SunsetHour'] = pd.to\_datetime(data.TimeSunSet).dt.hour.astype(np.int) data['SunsetMinute'] = pd.to\_datetime(data.TimeSunSet).dt.minute.astype(np.int)

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/549316017.py:1: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['SunriseHour'] = pd.to\_datetime(data.TimeSunRise).dt.hour.astype(np.int) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/549316017.py:2: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['SunriseMinute'] = pd.to\_datetime(data.TimeSunRise).dt.minute.astype(np.int) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/549316017.py:4: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

data['SunsetHour'] = pd.to\_datetime(data.TimeSunSet).dt.hour.astype(np.int) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_7456/549316017.py:5: DeprecationWarning: `n p.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re view your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

 data['SunsetMinute'] = pd.to\_datetime(data.TimeSunSet).dt.minute.astype(np.int)

In [22]:



data = data.drop(['TimeSunRise','TimeSunSet'], axis=1)

Now we'll look for the names of the columns of the dataset using pd.columns

In [23]:



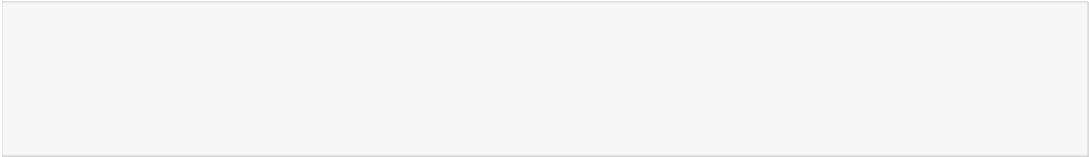
data.columns

Out[23]:

Index(['UNIXTime', 'Radiation', 'Temperature', 'Pressure', 'Humidity', 'WindDirection(Degrees)', 'Speed', 'Month', 'Day', 'Year', 'Hour', 'Minute', 'Second', 'SunriseHour', 'SunriseMinute', 'SunsetHour', 'SunsetMinute'],

dtype='object')

In [24]:



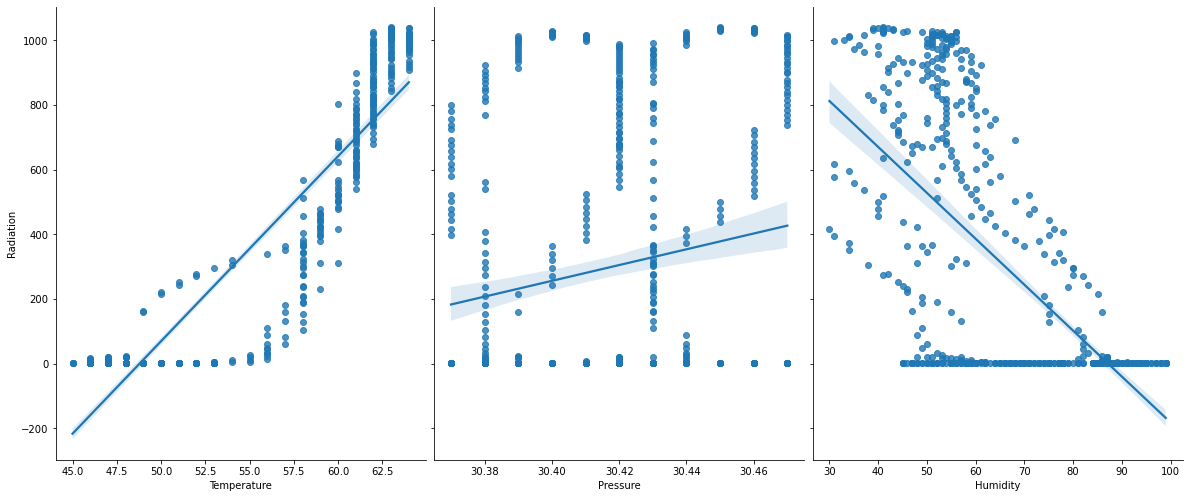
sns.pairplot(data.head(600),

x\_vars=["Temperature","Pressure","Humidity"], y\_vars="Radiation",

height=7, aspect=0.8, kind='reg')

Out[24]:

<seaborn.axisgrid.PairGrid at 0x1e399391a90>



In Temperature vs Radiation graph - we found that both the parameters are corresponding to each other, increasing in temperature Radiation is also increasing.

But in Pressure vs Radiation Graph No such relation is formed.

In Humidity vs Radiation - Radiation is maximum where Humidity is low.

In [25]:



data.corr()

Out[25]:

**UNIXTime Radiation Temperature Pressure Humidity WindDirection(Degrees) Speed Month**

**UNIXTime** 1.000000 -0.081286

-0.369169

-

-

0.332016 0.063117

0.152613 0.173860 0.968235

**Radiation** -0.081286 1.000000 0.734955 0.119016 - -0.230324 0.0**S**7**p**3**e**6**e**2**d**7 -

**UNIXTime Radiation Temperature Pressure Humidity WindDirection(Degrees)**

0.226171

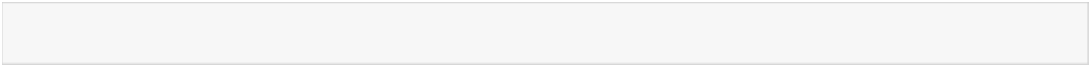
**Month** 0.095450

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| **Temperature** | -0.369169 | 0.734955 | 1.000000 | 0.311173 | - 0.285055 | -0.259421 | - 0.031458 | - 0.354560 |
| **Pressure** | -0.332016 | 0.119016 | 0.311173 | 1.000000 | - 0.223973 | -0.229010 | - 0.083639 | - 0.341759 |
| **Humidity** | -0.063117 | -0.226171 | -0.285055 | - 0.223973 | 1.000000 | -0.001833 | - 0.211624 | - 0.068854 |
| **WindDirection(Degrees)** | 0.152613 | -0.230324 | -0.259421 | - 0.229010 | - 0.001833 | 1.000000 | 0.073092 | 0.181485 |
| **Speed** | 0.173860 | 0.073627 | -0.031458 | - 0.083639 | - 0.211624 | 0.073092 | 1.000000 | 0.150822 |
| **Month** | 0.968235 | -0.095450 | -0.354560 | - 0.341759 | - 0.068854 | 0.181485 | 0.150822 | 1.000000 |
| **Day** | 0.286457 | 0.039978 | -0.123705 | - 0.024633 | 0.014637 | -0.082354 | 0.117337 | 0.038027 |
| **Year** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **Hour** | 0.001056 | 0.004398 | 0.197464 | 0.091069 | 0.077899 | -0.077969 | - 0.057939 | - 0.005396 |
| **Minute** | 0.000406 | -0.000730 | -0.001934 | 0.001860 | 0.000499 | -0.000602 | 0.000192 | 0.000168 |
| **Second** | 0.231002 | -0.031270 | -0.036147 | - 0.031102 | - 0.027682 | -0.032568 | - 0.032934 | 0.220563 |
| **SunriseHour** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **SunriseMinute** | 0.981908 | -0.092850 | -0.380968 | - 0.380399 | - 0.023955 | 0.176929 | 0.167075 | 0.952472 |
| **SunsetHour** | -0.818884 | 0.048719 | 0.300920 | 0.151939 | 0.145143 | -0.078540 | - 0.159384 | - 0.784783 |
| **SunsetMinute** | 0.586612 | -0.039816 | -0.242881 | - 0.119599 | - 0.119526 | 0.070030 | 0.119926 | 0.541883 |



|  |  |  |
| --- | --- | --- |
|  |  |  |

In [26]:



## In year column there is only one value 2016 so it gives mean standard daviation 0

data=data.drop('Year', axis=1)

In [27]:



data['SunriseHour'].unique()

Out[27]:

array([6])

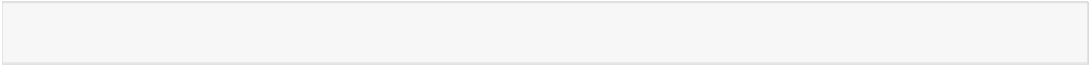
In [28]:



data = data.drop('SunriseHour', axis=1)

## Splitting/Scaling

In [29]:



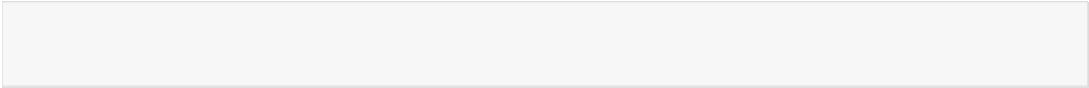
y = data['Radiation'].copy()

x = data.drop('Radiation',axis = 1).copy()

Our data is having high variation so StandardScaler is used to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1

In [30]:





scaler = StandardScaler()

scale\_data = scaler.fit\_transform(x)

In [31]:



scale\_data = pd.DataFrame(scale\_data)

In [32]:



scale\_data

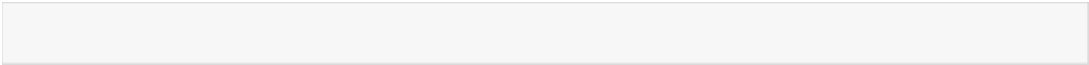
Out[32]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **1** |
| **0** | - 0.937753 | - 0.500439 | 0.678974 | - 0.616253 | 0.407620 | - 0.178738 | - 1.391540 | 1.512248 | 1.655482 | 1.589236 | 0.660811 | 0.94342 |
| **1** | - 0.937854 | - 0.500439 | 0.678974 | - 0.654730 | 0.400285 | - 0.823359 | - 1.391540 | 1.512248 | 1.655482 | 1.299687 | 0.429167 | 0.94342 |
| **2** | - 0.937953 | - 0.500439 | 0.678974 | - 0.693206 | 0.183490 | - 0.823359 | - 1.391540 | 1.512248 | 1.655482 | 1.010138 | 0.660811 | 0.94342 |
| **3** | - 0.938054 | - 0.500439 | 0.678974 | - 0.577776 | - 0.069497 | - 0.823359 | - 1.391540 | 1.512248 | 1.655482 | 0.720589 | 0.274737 | 0.94342 |
| **4** | - 0.938153 | - 0.500439 | 0.678974 | - 0.500823 | - 0.463407 | - 0.178738 | - 1.391540 | 1.512248 | 1.655482 | 0.431040 | 0.506381 | 0.94342 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **32681** | 0.845373 | - 1.145490 | 0.130250 | 1.038241 | 0.023209 | 0.145006 | 1.344003 | - 1.701824 | - 1.672098 | - 0.437606 | - 1.037912 | 0.86087 |
| **32682** | 0.845273 | - 1.145490 | - 0.052658 | 1.038241 | - 0.309138 | 0.145006 | 1.344003 | - 1.701824 | - 1.672098 | - 0.727155 | - 1.269556 | 0.86087 |
| **32683** | 0.845173 | - 1.145490 | - 0.052658 | 1.038241 | 0.020443 | 0.789627 | 1.344003 | - 1.701824 | - 1.672098 | - 1.016704 | - 1.269556 | 0.86087 |
| **32684** | 0.845073 | - 1.145490 | - 0.052658 | 0.999764 | 0.248901 | 0.465884 | 1.344003 | - 1.701824 | - 1.672098 | - 1.306253 | - 1.192341 | 0.86087 |
| **32685** | 0.844973 | - 1.145490 | 0.130250 | 0.999764 | - 0.720242 | - 0.823359 | 1.344003 | - 1.701824 | - 1.672098 | - 1.595802 | - 1.192341 | 0.86087 |

32686 rows × 14 columns

|  |  |  |
| --- | --- | --- |
|  |  |  |

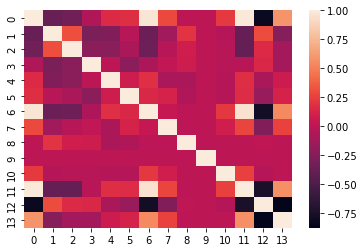
In [33]:



plt.figure(figsize=(6,4)) sns.heatmap(scale\_data.corr())

Out[33]:

<AxesSubplot:>



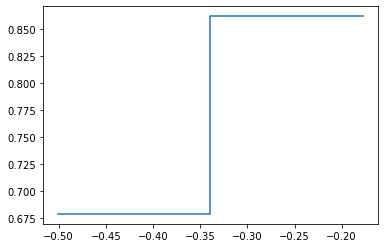
In [34]:



plt.plot(scale\_data[1].head(20), scale\_data[2].head(20))

Out[34]:

[<matplotlib.lines.Line2D at 0x1e39b077ee0>]



In [35]:

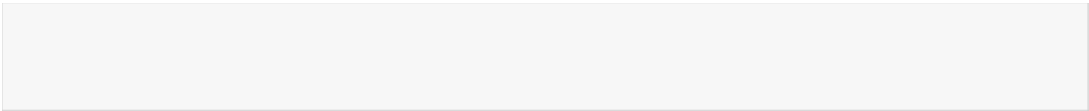
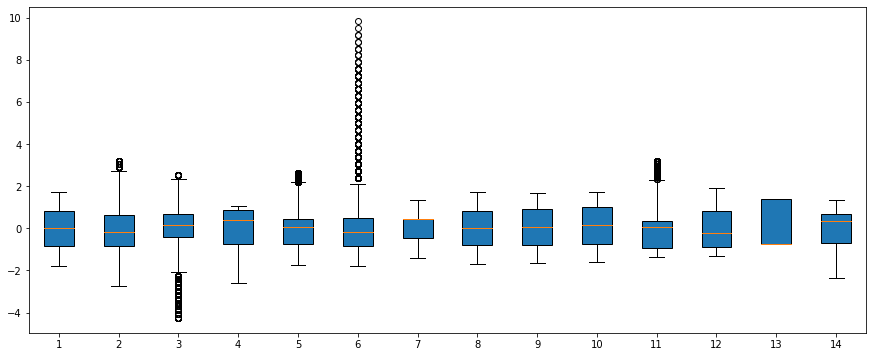


fig = plt.figure(1, figsize=(15,6))

ax = fig.add\_subplot(111)

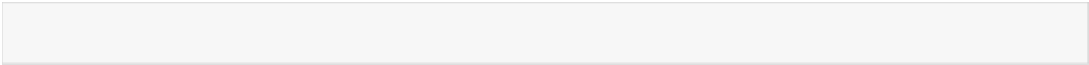
bp = ax.boxplot(scale\_data, patch\_artist=True)



Above Boxplot shows the outliers present in the data

For finding outliers we've used IQR(Inter Quartile Range) method.

In [36]:



l1 = scale\_data.quantile(0.25) l1

Out[36]:

|  |  |
| --- | --- |
| 0 | -0.832205 |
| 1 | -0.822965 |
| 2 | -0.418473 |
| 3 | -0.731683 |
| 4 | -0.736625 |
| 5 | -0.823359 |
| 6 | -0.479692 |
| 7 | -0.783518 |
| 8 | -0.804034 |

10 -0.960697

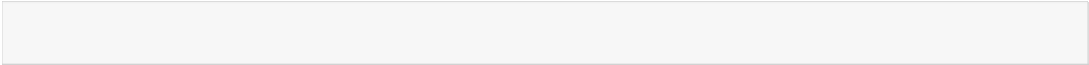
11 -0.878986

12 -0.730393

13 -0.710113

Name: 0.25, dtype: float64

In [37]:

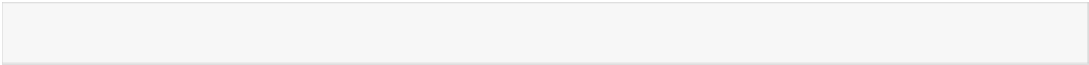


h1 = scale\_data.quantile(0.75) h1

Out[37]:

|  |  |  |
| --- | --- | --- |
| 0 | 0.809608 |  |
| 1 | 0.628400 |  |
| 2 | 0.678974 |  |
| 3 | 0.845858 |  |
| 4 | 0.430706 |  |
| 5 | 0.465884 |  |
| 6 | 0.432155 |  |
| 7 | 0.823518 |  |
| 8 | 0.932095 |  |
| 9 | 1.010138 |  |
| 10 | 0.351952 |  |
| 11 | 0.796437 |  |
| 12 | 1.369126 |  |
| 13 | 0.670221 |  |
| Name: | 0.75, dtype: | float64 |

In [38]:



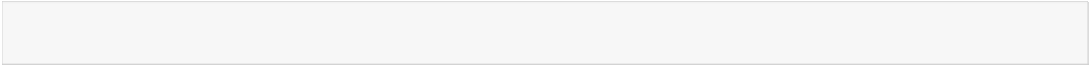
IQR = h1-l1 IQR

Out[38]:

|  |  |
| --- | --- |
| 0 | 1.641812 |
| 1 | 1.451364 |
| 2 | 1.097448 |
| 3 | 1.577541 |
| 4 | 1.167331 |
| 5 | 1.289243 |
| 6 | 0.911848 |
| 7 | 1.607036 |
| 8 | 1.736129 |
| 9 | 1.737293 |
| 10 | 1.312649 |
| 11 | 1.675422 |
| 12 | 2.099519 |
| 13 | 1.380334 |

dtype: float64

In [39]:



LB = l1 - 1.5\*IQR LB

Out[39]:

0 -3.294923

1 -3.000011

2 -2.064645

3 -3.097994

4 -2.487621

5 -2.757224

6 -1.847463

7 -3.194072

8 -3.408228

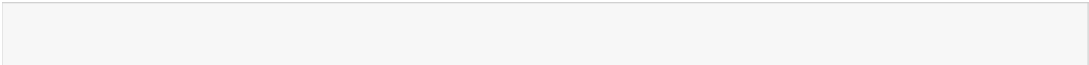
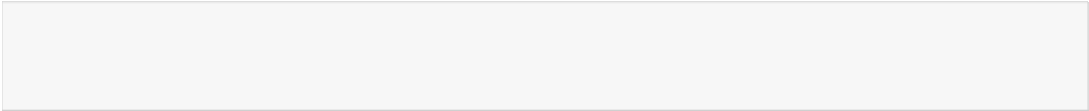
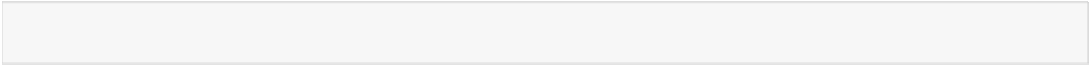
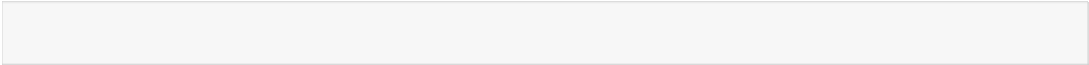
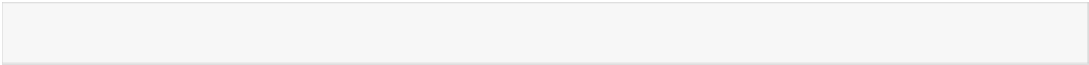
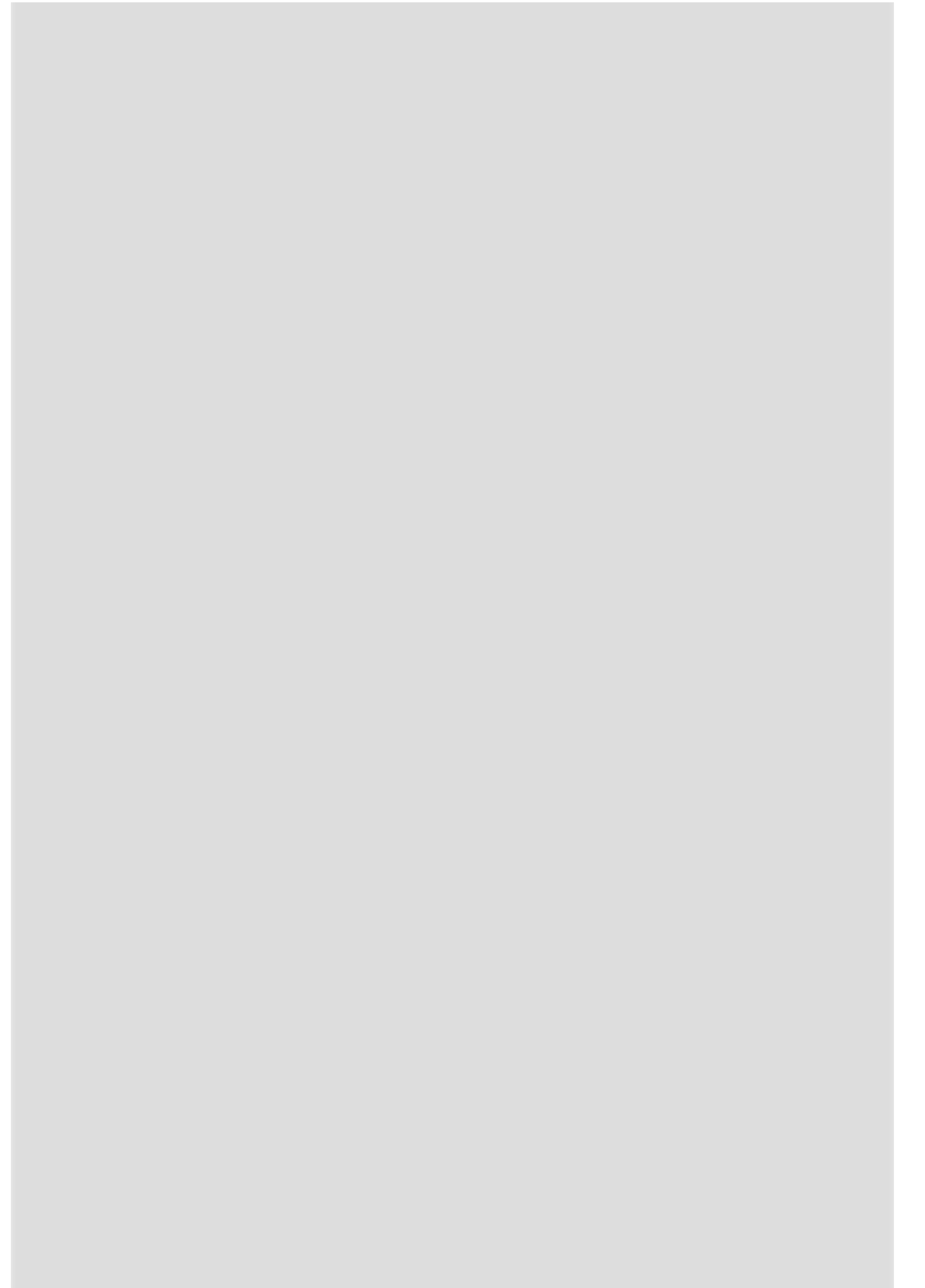
9 -3.333095

10 -2.929670

11 -3.392119

12 -3.879671

dtype: float64



In [40]:

UB = h1 + 1.5\*IQR UB

Out[40]:

|  |  |
| --- | --- |
| 0 | 3.272326 |
| 1 | 2.805446 |
| 2 | 2.325145 |
| 3 | 3.212169 |
| 4 | 2.181702 |
| 5 | 2.399749 |
| 6 | 1.799927 |
| 7 | 3.234072 |
| 8 | 3.536289 |
| 9 | 3.616078 |
| 10 | 2.320925 |
| 11 | 3.309570 |
| 12 | 4.518404 |
| 13 | 2.740722 |

dtype: float64

In [48]:

### #scale\_data[scale\_data < LB]

In [49]:

### #scale\_data[scale\_data > UB]

In [41]:

var = scale\_data[(scale\_data < LB) | (scale\_data > UB)].index var

Out[41]:

RangeIndex(start=0, stop=32686, step=1)

In [42]:

without\_outlier = scale\_data.drop(index=var) without\_outlier

Out[42]:

**0 1 2 3 4 5 6 7 8 9 10 11 12 13**

## train\_test\_split

splitting x and y into training and testing sets

random\_state select data randomely and freez/fix it to entire analysis

In [44]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, random\_state=100, train\_size = 0.7)

x\_train, x\_val, y\_train, y\_val = train\_test\_split(x\_train, y\_train, random\_state=200, tr ain\_size = 0.8)

In [45]:

print('Size of x\_train:',x\_train.shape[0]) print('Size of x\_val:',x\_val.shape[0])





print('Size of x\_test:',x\_test.shape[0])

Size of x\_train: 18304 Size of x\_val: 4576 Size of x\_test: 9806

# LinearRegression

Linear Regression is a machine learning algorithm based on supervised learning Reason for choosing Linear regression

1. Linear relationship and data is continuous.
2. No or little multicollinearity.
3. No autocorrelation

In [46]:



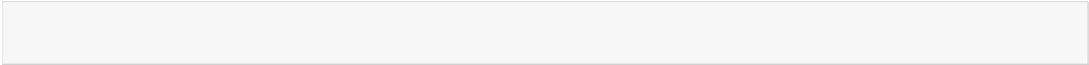
linreg = LinearRegression()

In [47]:



linreg = linreg.fit(x\_train, y\_train)

In [48]:



y\_pred = linreg.predict(x\_test) y\_pred

Out[48]:

array([ 72.13372087, -228.14066218, 117.6368593 , ..., 137.49005509,

-44.0111195 , 216.75232338])

r2\_score = 1- (RSS / TSS) RSS- residual sum of squares TSS- total sum of squares

In [49]:



metrics.r2\_score(y\_test, y\_pred)

Out[49]: 0.6313621786547534

In [50]:



metrics.mean\_absolute\_error(y\_test, y\_pred)

Out[50]: 146.18566616387497

In [51]:



metrics.mean\_squared\_error(y\_test, y\_pred)

Out[51]: 36866.21322174484

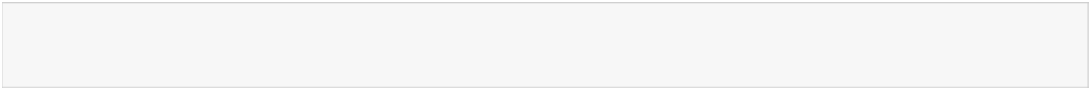
# XGBoost Training

Our r2\_score for linear regression is 0.6313.



So we have to use boosting technique on this data.

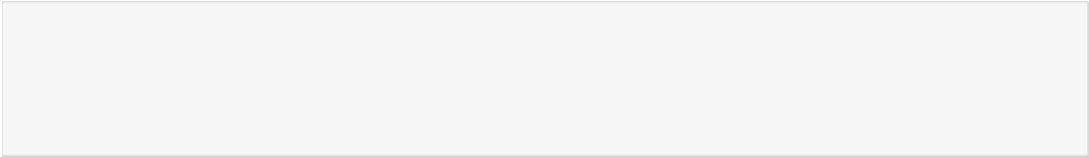
In [59]:



dtrain = xgb.DMatrix(x\_train, label=y\_train) dval = xgb.DMatrix(x\_val, label=y\_val)

dtest = xgb.DMatrix(x\_test, label=y\_test)

In [60]:



def get\_model\_rmse(params):

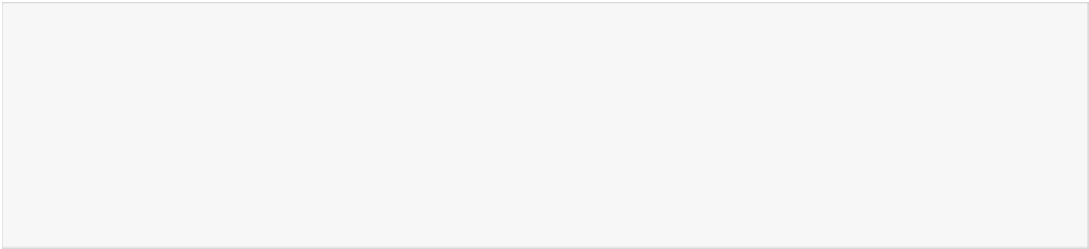
model = xgb.train(params, dtrain, num\_boost\_round=100, evals=[(dval, 'eval')], early

\_stopping\_rounds=10)

results = model.eval(dtest)

rmse = float(re.search(r'[\d.]+$', results).group(0)) return rmse

In [61]:



def objective(trial):

learning\_rate = trial.suggest\_loguniform('learning\_rate', 0.00001, 10.0)

max\_depth = trial.suggest\_int('max\_depth', 4, 8)

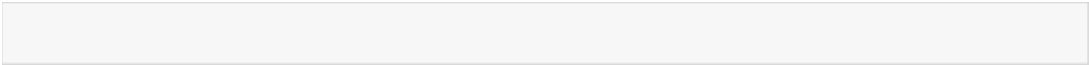
l1\_reg = trial.suggest\_loguniform('l1\_reg', 0.00001, 10.0)

l2\_reg = trial.suggest\_loguniform('l2\_reg', 0.00001, 10.0)

params = {'learning\_rate':learning\_rate, 'max\_depth':max\_depth, 'alpha':l1\_reg, 'lam bda':l2\_reg}

return get\_model\_rmse(params)

In [62]:



study = optuna.create\_study()

study.optimize(objective, n\_trials=100, show\_progress\_bar=True)

[I 2022-09-16 00:08:22,891] A new study created in memory with name: no-name-457e2248-df1 f-480f-91a1-6150d174f586

C:\Users\Swapnil\anaconda3\lib\site-packages\optuna\progress\_bar.py:49: ExperimentalWarni ng: Progress bar is experimental (supported from v1.2.0). The interface can change in the future.

self.\_init\_valid()

[0] eval-rmse:376.06470

[1] eval-rmse:370.75150

[2] eval-rmse:365.68145

[3] eval-rmse:360.56027

[4] eval-rmse:355.54584

[5] eval-rmse:350.72919

[6] eval-rmse:345.89422

[7] eval-rmse:341.13401

[8] eval-rmse:336.48748

[9] eval-rmse:331.92892

[10] eval-rmse:327.48300

[11] eval-rmse:323.20884

[12] eval-rmse:318.92756

[13] eval-rmse:314.61478

[14] eval-rmse:310.45021

[15] eval-rmse:306.26982

[16] eval-rmse:302.33925

[17] eval-rmse:298.37979

[18] eval-rmse:294.58901

[19] eval-rmse:290.82275

[20] eval-rmse:287.03360

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/3826600089.py:2: FutureWarning: sugge st\_loguniform has been deprecated in v3.0.0. This feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use :func:`~optuna.trial.Trial.sugg est\_float` instead.

learning\_rate = trial.suggest\_loguniform('learning\_rate', 0.00001, 10.0) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/3826600089.py:4: FutureWarning: sugge



st\_loguniform has been deprecated in v3.0.0. This feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use :func:`~optuna.trial.Trial.sugg est\_float` instead.

l1\_reg = trial.suggest\_loguniform('l1\_reg', 0.00001, 10.0) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/3826600089.py:5: FutureWarning: sugge st\_loguniform has been deprecated in v3.0.0. This feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use :func:`~optuna.trial.Trial.sugg est\_float` instead.

l2\_reg = trial.suggest\_loguniform('l2\_reg', 0.00001, 10.0)

[21] eval-rmse:283.55253

[22] eval-rmse:280.09835

[23] eval-rmse:276.51593

[24] eval-rmse:273.18075

[25] eval-rmse:269.74573

[26] eval-rmse:266.44758

[27] eval-rmse:263.16363

[28] eval-rmse:260.07800

[29] eval-rmse:256.97613

[30] eval-rmse:253.89749

[31] eval-rmse:251.00011

[32] eval-rmse:248.05970

[33] eval-rmse:245.27835

[34] eval-rmse:242.48963

[35] eval-rmse:239.70969

[36] eval-rmse:236.96404

[37] eval-rmse:234.26564

[38] eval-rmse:231.63007

[39] eval-rmse:229.04032

[40] eval-rmse:226.51093

[41] eval-rmse:224.11788

[42] eval-rmse:221.61022

[43] eval-rmse:219.31174

[44] eval-rmse:216.90811

[45] eval-rmse:214.71147

[46] eval-rmse:212.43754

[47] eval-rmse:210.33632

[48] eval-rmse:208.13255

[49] eval-rmse:206.14205

[50] eval-rmse:204.03540

[51] eval-rmse:202.06407

[52] eval-rmse:200.04863

[53] eval-rmse:198.16719

[54] eval-rmse:196.29431

[55] eval-rmse:194.49880

[56] eval-rmse:192.72943

[57] eval-rmse:190.98812

[58] eval-rmse:189.29369

[59] eval-rmse:187.66021

[60] eval-rmse:186.01440

[61] eval-rmse:184.49781

[62] eval-rmse:182.95777

[63] eval-rmse:181.46351

[64] eval-rmse:179.97451

[65] eval-rmse:178.59711

[66] eval-rmse:177.17186

[67] eval-rmse:175.78880

[68] eval-rmse:174.39662

[69] eval-rmse:173.08132

[70] eval-rmse:171.78653

[71] eval-rmse:170.51844

[72] eval-rmse:169.27912

[73] eval-rmse:168.14178

[74] eval-rmse:166.96512

[75] eval-rmse:165.81255

[76] eval-rmse:164.74455

[77] eval-rmse:163.72081

[78] eval-rmse:162.72600

[79] eval-rmse:161.74254

[80] eval-rmse:160.70132

[81] eval-rmse:159.77932

[83] eval-rmse:157.92662

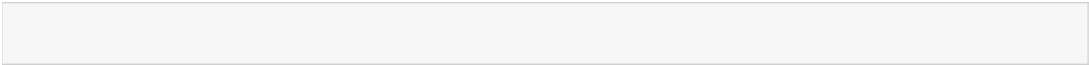
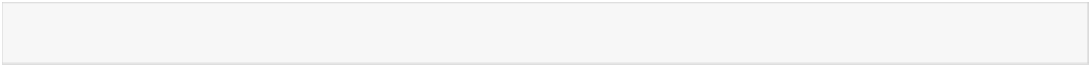
[84] eval-rmse:156.99067

[85] eval-rmse:156.14064

96]

[90] eval-rmse:85.76193

[91] eval-rmse:85.76279



[92] eval-rmse:85.73992

[93] eval-rmse:85.73423

[94] eval-rmse:85.69192

[95] eval-rmse:85.74615

[96] eval-rmse:85.72670

[97] eval-rmse:85.66380

[98] eval-rmse:85.64125

[99] eval-rmse:85.59240

[I 2022-09-16 00:10:22,759] Trial 99 finished with value: 76.95296748473557 and parameter

s: {'learning\_rate': 0.08434873539279487, 'max\_depth': 8, 'l1\_reg': 0.010667925015782835, 'l2\_reg': 9.745186089331475e-05}. Best is trial 73 with value: 76.54032596367466.

In [65]:

best\_params = study.best\_params best\_params

Out[65]:

{'learning\_rate': 0.08272902885234269,

'max\_depth': 8,

'l1\_reg': 0.03518319856284036,

'l2\_reg': 0.01981267541444342}

In [66]:

model = xgb.train(best\_params, dtrain, num\_boost\_round=10000, evals=[(dval, 'eval')], ea rly\_stopping\_rounds=10)

[00:15:15] WARNING: C:/Users/administrator/workspace/xgboost-win64\_release\_1.6.0/src/lear ner.cc:627:

Parameters: { "l1\_reg", "l2\_reg" } might not be used.

This could be a false alarm, with some parameters getting used by language bindings but then being mistakenly passed down to XGBoost core, or some parameter actually being use

d

but getting flagged wrongly here. Please open an issue if you find any such cases.

[0] eval-rmse:352.88760

[1] eval-rmse:326.90826

[2] eval-rmse:303.15570

[3] eval-rmse:281.44770

[4] eval-rmse:261.69322

[5] eval-rmse:243.73023

[6] eval-rmse:227.41662

[7] eval-rmse:212.56383

[8] eval-rmse:199.25476

[9] eval-rmse:187.15267

[10] eval-rmse:176.26835

[11] eval-rmse:166.39981

[12] eval-rmse:157.64394

[13] eval-rmse:149.72642

[14] eval-rmse:142.49751

[15] eval-rmse:136.02838

[16] eval-rmse:130.23729

[17] eval-rmse:125.22660

[18] eval-rmse:120.70218

[19] eval-rmse:116.62178

[20] eval-rmse:113.08506

[21] eval-rmse:110.07661

[22] eval-rmse:107.24425

[23] eval-rmse:104.74737

[24] eval-rmse:102.67941

[25] eval-rmse:100.83175

[26] eval-rmse:99.14647

[27] eval-rmse:97.73427

[28] eval-rmse:96.51934

[29] eval-rmse:95.25603

[30] eval-rmse:94.34875

[31] eval-rmse:93.41476

[32] eval-rmse:92.39979



[33] eval-rmse:91.53877

[34] eval-rmse:90.83755

[35] eval-rmse:90.15850

[36] eval-rmse:89.52864

[37] eval-rmse:89.02843

[38] eval-rmse:88.61707

[39] eval-rmse:88.31465

[40] eval-rmse:87.91944

[41] eval-rmse:87.70886

[42] eval-rmse:87.37963

[43] eval-rmse:87.21475

[44] eval-rmse:86.94630

[45] eval-rmse:86.70167

[46] eval-rmse:86.54051

[47] eval-rmse:86.38329

[48] eval-rmse:86.22928

[49] eval-rmse:86.13061

[50] eval-rmse:86.00594

[51] eval-rmse:85.86457

[52] eval-rmse:85.76407

[53] eval-rmse:85.68528

[54] eval-rmse:85.58573

[55] eval-rmse:85.50618

[56] eval-rmse:85.46428

[57] eval-rmse:85.30382

[58] eval-rmse:85.26144

[59] eval-rmse:85.23413

[60] eval-rmse:85.26489

[61] eval-rmse:85.31166

[62] eval-rmse:85.32186

[63] eval-rmse:85.36007

[64] eval-rmse:85.24602

[65] eval-rmse:85.22171

[66] eval-rmse:85.14470

[67] eval-rmse:85.04538

[68] eval-rmse:85.00447

[69] eval-rmse:84.94234

[70] eval-rmse:84.85963

[71] eval-rmse:84.83882

[72] eval-rmse:84.78299

[73] eval-rmse:84.77024

[74] eval-rmse:84.72946

[75] eval-rmse:84.69680

[76] eval-rmse:84.62223

[77] eval-rmse:84.49719

[78] eval-rmse:84.43283

[79] eval-rmse:84.40773

[80] eval-rmse:84.38907

[81] eval-rmse:84.35920

[82] eval-rmse:84.34260

[83] eval-rmse:84.30712

[84] eval-rmse:84.30393

[85] eval-rmse:84.29501

[86] eval-rmse:84.31352

[87] eval-rmse:84.19825

[88] eval-rmse:84.19675

[89] eval-rmse:84.16610

[90] eval-rmse:84.17392

[91] eval-rmse:84.18383

[92] eval-rmse:84.17032

[93] eval-rmse:84.13712

[94] eval-rmse:84.14029

[95] eval-rmse:84.14219

[96] eval-rmse:84.14405

[97] eval-rmse:84.15393

[98] eval-rmse:84.09681

[99] eval-rmse:84.10973

[100] eval-rmse:84.08137

[101] eval-rmse:84.03909

[102] eval-rmse:84.03767

[103] eval-rmse:84.03557

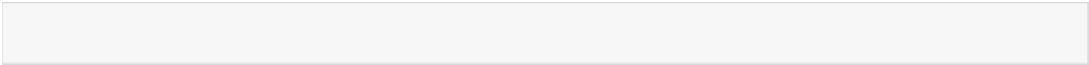
|  |  |
| --- | --- |
| [104] | eval-rmse:84.00764 |
| [105] | eval-rmse:84.02096 |
| [106] | eval-rmse:83.95675 |
| [107] | eval-rmse:83.93339 |
| [108] | eval-rmse:83.90768 |
| [109] | eval-rmse:83.89651 |
| [110] | eval-rmse:83.83146 |
| [111] | eval-rmse:83.81469 |
| [112] | eval-rmse:83.77579 |
| [113] | eval-rmse:83.76965 |
| [114] | eval-rmse:83.78330 |
| [115] | eval-rmse:83.78632 |
| [116] | eval-rmse:83.77710 |
| [117] | eval-rmse:83.75767 |
| [118] | eval-rmse:83.76397 |
| [119] | eval-rmse:83.73186 |
| [120] | eval-rmse:83.71698 |
| [121] | eval-rmse:83.70540 |
| [122] | eval-rmse:83.67351 |
| [123] | eval-rmse:83.65809 |
| [124] | eval-rmse:83.65483 |
| [125] | eval-rmse:83.69870 |
| [126] | eval-rmse:83.69157 |
| [127] | eval-rmse:83.65400 |
| [128] | eval-rmse:83.64164 |
| [129] | eval-rmse:83.64045 |
| [130] | eval-rmse:83.63196 |
| [131] | eval-rmse:83.59038 |
| [132] | eval-rmse:83.61219 |
| [133] | eval-rmse:83.62579 |
| [134] | eval-rmse:83.63362 |
| [135] | eval-rmse:83.58632 |
| [136] | eval-rmse:83.54230 |
| [137] | eval-rmse:83.52484 |
| [138] | eval-rmse:83.52928 |
| [139] | eval-rmse:83.52253 |
| [140] | eval-rmse:83.46877 |
| [141] | eval-rmse:83.44141 |
| [142] | eval-rmse:83.46708 |
| [143] | eval-rmse:83.45596 |
| [144] | eval-rmse:83.44183 |
| [145] | eval-rmse:83.44797 |
| [146] | eval-rmse:83.35696 |
| [147] | eval-rmse:83.32711 |
| [148] | eval-rmse:83.26224 |
| [149] | eval-rmse:83.22326 |
| [150] | eval-rmse:83.23888 |
| [151] | eval-rmse:83.24528 |
| [152] | eval-rmse:83.27615 |
| [153] | eval-rmse:83.16626 |
| [154] | eval-rmse:83.15607 |
| [155] | eval-rmse:83.18432 |
| [156] | eval-rmse:83.22531 |
| [157] | eval-rmse:83.24613 |
| [158] | eval-rmse:83.22637 |
| [159] | eval-rmse:83.19401 |
| [160] | eval-rmse:83.12803 |
| [161] | eval-rmse:83.09228 |
| [162] | eval-rmse:83.05152 |
| [163] | eval-rmse:83.04138 |
| [164] | eval-rmse:83.03502 |
| [165] | eval-rmse:83.03645 |
| [166] | eval-rmse:83.02382 |
| [167] | eval-rmse:83.00582 |
| [168] | eval-rmse:82.99891 |
| [169] | eval-rmse:82.98363 |
| [170] | eval-rmse:82.96276 |
| [171] | eval-rmse:82.95374 |
| [172] | eval-rmse:82.93993 |
| [173] | eval-rmse:82.94420 |
| [174] | eval-rmse:82.95664 |
| [175] | eval-rmse:82.92749 |



|  |  |
| --- | --- |
| [176] | eval-rmse:82.95917 |
| [177] | eval-rmse:82.96050 |
| [178] | eval-rmse:82.95131 |
| [179] | eval-rmse:82.95147 |
| [180] | eval-rmse:82.95150 |
| [181] | eval-rmse:82.96250 |
| [182] | eval-rmse:82.93556 |
| [183] | eval-rmse:82.93313 |
| [184] | eval-rmse:82.94228 |
| [185] | eval-rmse:82.93279 |



In [67]:



y\_true = np.array(y\_test, dtype=np.float)

y\_pred = np.array(model.predict(dtest), dtype=np.float)

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/2899363029.py:1: DeprecationWarning:

`np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `f loat` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

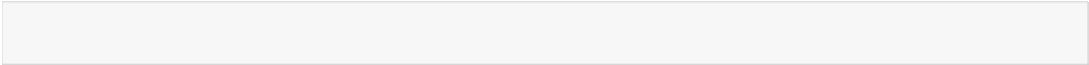
y\_true = np.array(y\_test, dtype=np.float) C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/2899363029.py:2: DeprecationWarning:

`np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `f loat` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.

Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas e/1.20.0-notes.html#deprecations

y\_pred = np.array(model.predict(dtest), dtype=np.float)

In [128]:



r2 = r2\_score(y\_true, y\_pred) print("R^2 Score: {:.4f}".format(r2))

R^2 Score: 0.9406 In [ ]:

# Time series

In [69]:



df[['date', 't', 'ap']] = df['Data'].str.split(' ', expand=True)

In [70]:



df['Datetime'] = df['date'].map(str) +' '+ df['Time'].map(str)

In [71]:



df['Datetime'] = pd.to\_datetime(df['Datetime'])

In [72]:



df\_time = df[['Datetime', 'Radiation']]

In [73]:



df\_time.set\_index('Datetime', inplace=True)

In [74]:



df\_time.resample(rule = 'A').mean()

Out[74]:



**Radiation**

**Datetime**

**2016-12-31** 207.124697

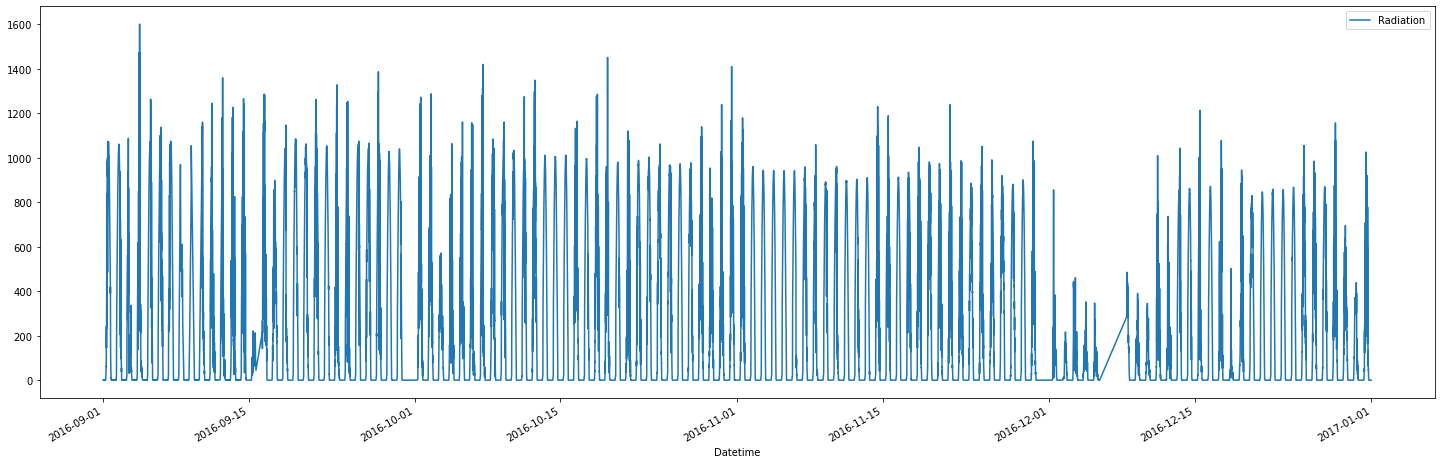
In [75]:



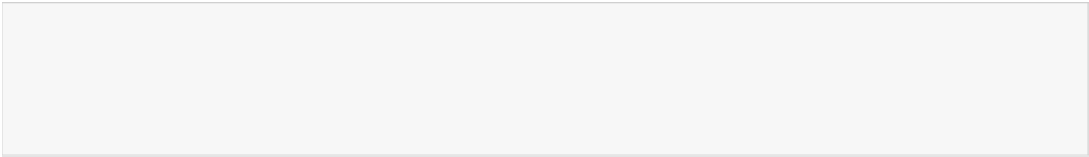
df\_time.plot(figsize=(25,8))

Out[75]:

<AxesSubplot:xlabel='Datetime'>



In [76]:



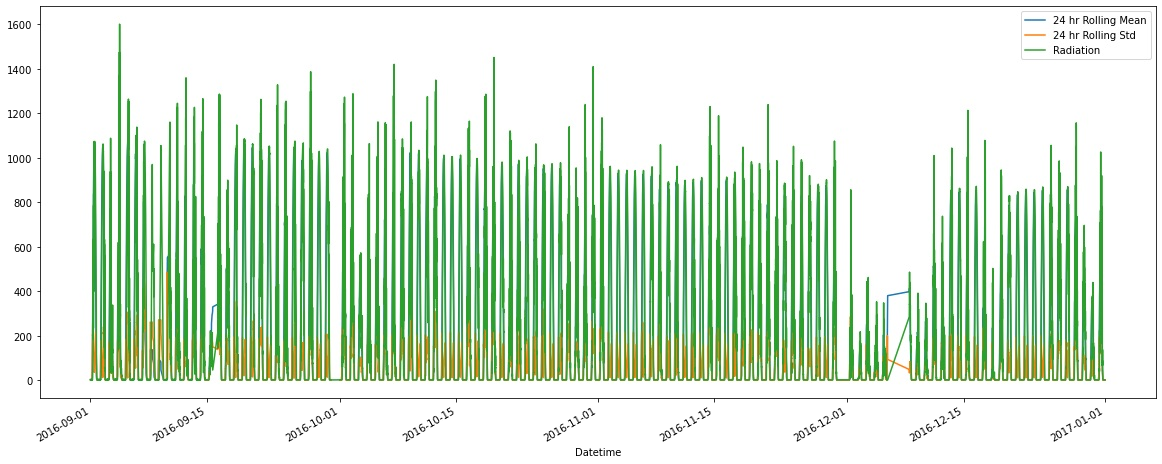
plt.figure(figsize=(20,8)) timeseries = df\_time['Radiation']

timeseries.rolling(24).mean().plot(label='24 hr Rolling Mean') timeseries.rolling(24).std().plot(label='24 hr Rolling Std') timeseries.plot()

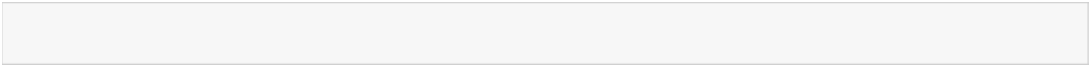
plt.legend(loc='best')

Out[76]:

<matplotlib.legend.Legend at 0x18ddafc7f70>



In [77]:



from statsmodels.tsa.seasonal import seasonal\_decompose

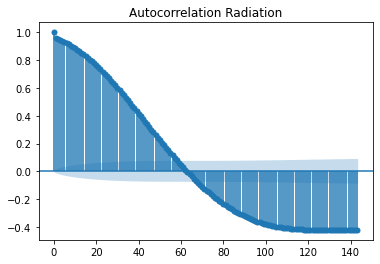
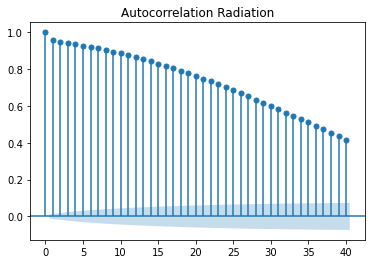
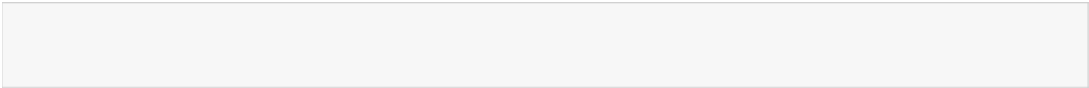
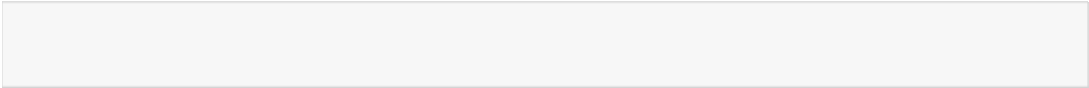
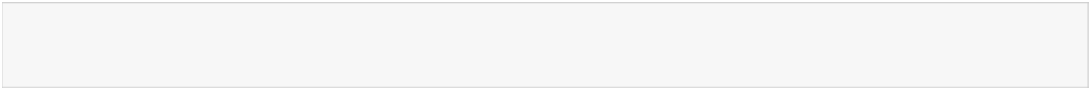
from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

In [78]:



df\_sort = df\_time

In [79]:



df\_sort.sort\_index(inplace=True)

C:\Users\Swapnil\anaconda3\lib\site-packages\pandas\core\frame.py:6393: SettingWithCopyWa rning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_g uide/indexing.html#returning-a-view-versus-a-copy

return super().sort\_index(

In [80]:

title = "Autocorrelation Radiation" lags = 40

plot\_acf(df\_sort['Radiation'], title=title, lags = lags);

In [81]:

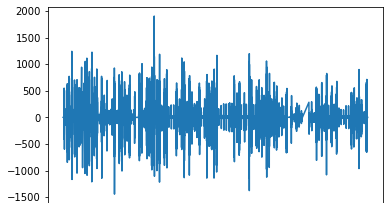
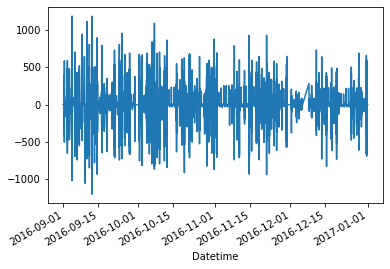
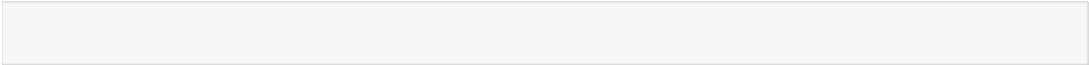
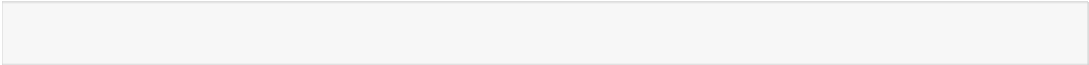
title = "Autocorrelation Radiation" lags = 40

plot\_acf(df\_sort['Radiation'], title=title, lags = 143);

In [82]:

title = "Auto-correlation Radiation" lags = 40

plot\_pacf(df\_sort['Radiation'], title=title, lags = lags);



In [83]:

df\_sort['first\_Diff'] = df\_sort['Radiation'] - df\_sort['Radiation'].shift(1) df\_sort['first\_Diff'].plot()

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/3187470166.py:1: SettingWithCopyWarni ng:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_g uide/indexing.html#returning-a-view-versus-a-copy

df\_sort['first\_Diff'] = df\_sort['Radiation'] - df\_sort['Radiation'].shift(1)

Out[83]:

<AxesSubplot:xlabel='Datetime'>

In [84]:

df\_sort['Seasonal\_first\_diff'] = df\_sort['first\_Diff'] - df\_sort['first\_Diff'].shift(24) df\_sort['Seasonal\_first\_diff'].plot()

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/854232961.py:1: SettingWithCopyWarnin g:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_g uide/indexing.html#returning-a-view-versus-a-copy

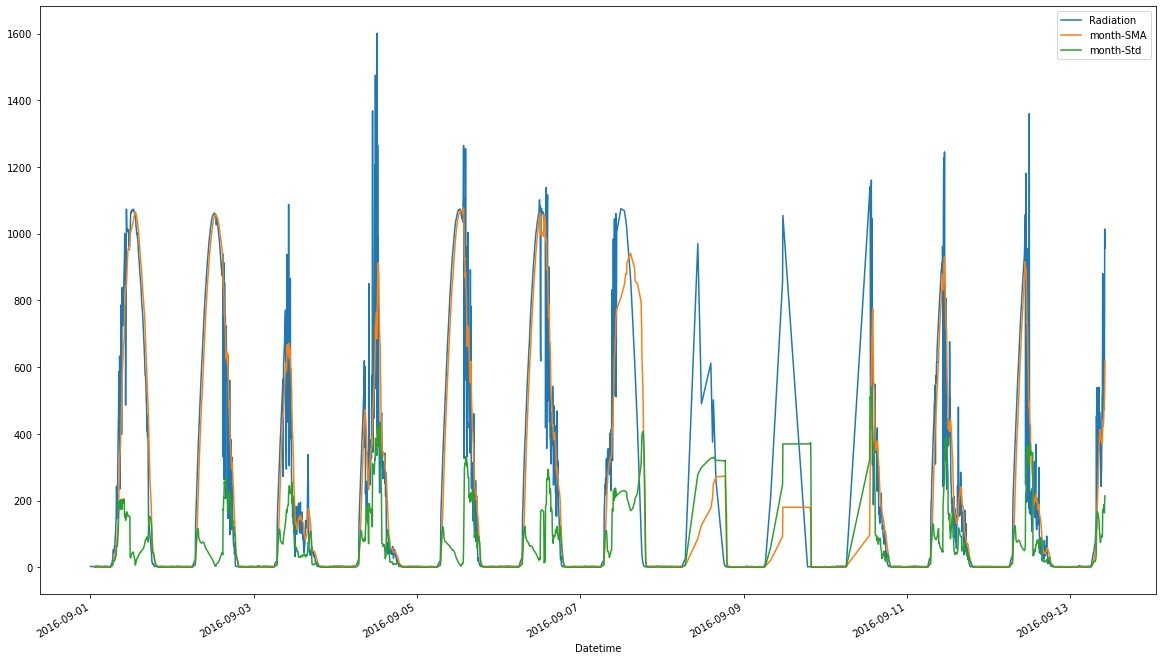
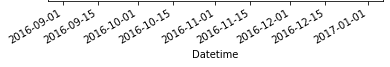
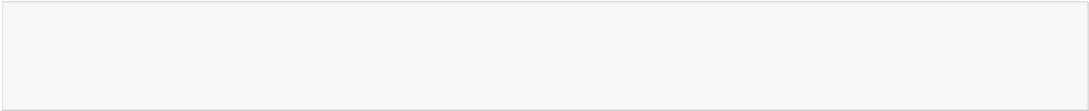
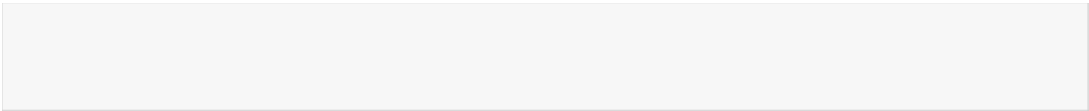
df\_sort['Seasonal\_first\_diff'] = df\_sort['first\_Diff'] - df\_sort['first\_Diff'].shift(24

)

Out[84]:

<AxesSubplot:xlabel='Datetime'>

In [85]:



df\_sort['month-SMA'] = df\_sort['Radiation'].rolling(window=12).mean() df\_sort['month-Std'] = df\_sort['Radiation'].rolling(window=12).std()

df\_sort[['Radiation','month-SMA','month-Std']].head(3000).plot(figsize=(20,12));

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/1945535349.py:1: SettingWithCopyWarni ng:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_g uide/indexing.html#returning-a-view-versus-a-copy

df\_sort['month-SMA'] = df\_sort['Radiation'].rolling(window=12).mean() C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/1945535349.py:2: SettingWithCopyWarni ng:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_g uide/indexing.html#returning-a-view-versus-a-copy

df\_sort['month-Std'] = df\_sort['Radiation'].rolling(window=12).std()

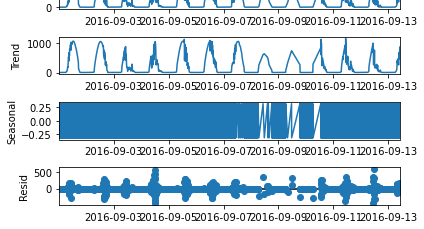
In [ ]:

In [86]:

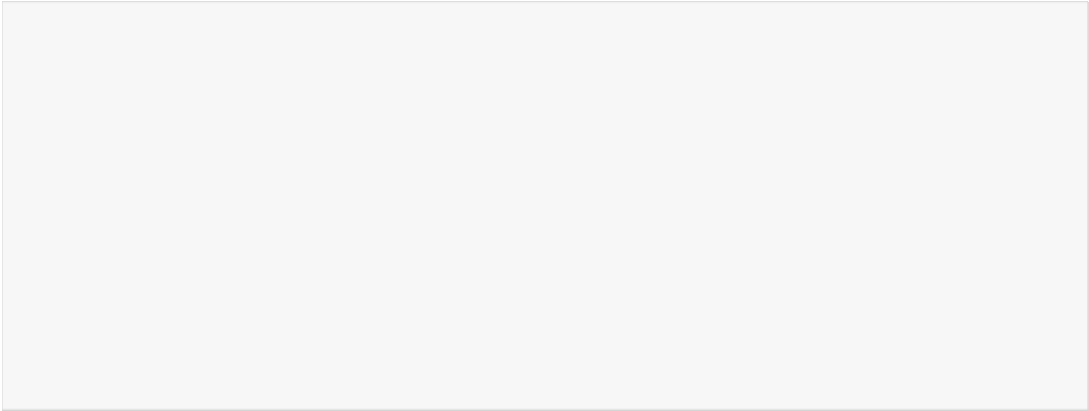
decomposition = seasonal\_decompose(df\_sort['Radiation'].head(3000), extrapolate\_trend='fr eq', period=2)

plt.figure(figsize=(20,12)) decomposition.plot();

<Figure size 1440x864 with 0 Axes>



In [87]:



from statsmodels.tsa.stattools import adfuller

def adf\_check(time\_series):

result = adfuller(time\_series) print("Augmented Dickey-Fuller Test")

labels = ['ADF Test Statistic ', 'p-value', '#Lags Used', 'Number of Observations Us

ed']

for value, label in zip(result, labels): print(label+' : ' +str(value))

if result[1] <= 0.05:

print("Strong evidence against the null hypothesis, reject the null hypothesis. data is stationary")

else:

print("weak evidence against null hypothesis, time series is non-stationary")

In [88]:



adf\_check(df\_sort['Radiation'])

Augmented Dickey-Fuller Test

ADF Test Statistic : -23.773592416137298 p-value : 0.0

#Lags Used : 45

Number of Observations Used : 32640

Strong evidence against the null hypothesis, reject the null hypothesis. data is stationa ry

In [89]:



adf\_check(df\_sort['Seasonal\_first\_diff'].dropna())

Augmented Dickey-Fuller Test

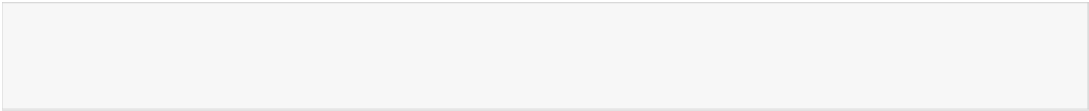
ADF Test Statistic : -33.012520323416155 p-value : 0.0

#Lags Used : 52

Number of Observations Used : 32608

Strong evidence against the null hypothesis, reject the null hypothesis. data is stationa ry

In [90]:



model = sm.tsa.statespace.SARIMAX(df\_sort['Radiation'],order=(1,1,0), seasonal\_decompose

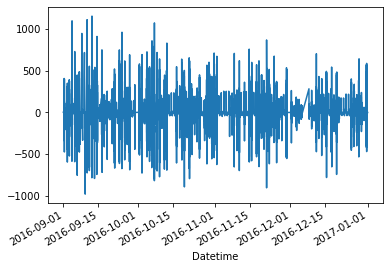
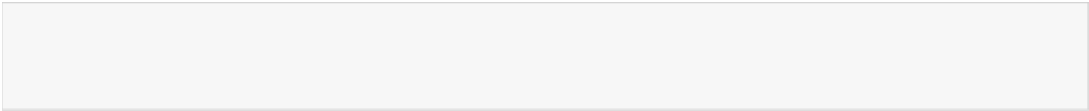
=(1,0,0,24))

result = model.fit() print(result.summary())

C:\Users\Swapnil\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:581: Value Warning: A date index has been provided, but it has no associated frequency information a nd so will be ignored when e.g. forecasting.

warnings.warn('A date index has been provided, but it has no' C:\Users\Swapnil\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:581: Value Warning: A date index has been provided, but it has no associated frequency information a

nd so will be ignored when e.g. forecasting.



warnings.warn('A date index has been provided, but it has no'

SARIMAX Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ============================================================================== | | | | | | | |
| Dep. Variable: | | Radiation | | No. Observations: | | 32686 | |
| Model: | | SARIMAX(1, 1, 0) | | Log Likelihood | | -190912.081 | |
| Date: | | Fri, 16 Sep 2022 | | AIC | | 381828.163 | |
| Time: | | 00:15:47 | | BIC | | 381844.952 | |
| Sample: | | 0 | | HQIC | | 381833.528 | |
| Covariance Type: | | - 32686  opg | |  | |  | |
| ============================================================================== | | | | | | | |
|  | coef | std err | z | | P>|z| | [0.025 | 0.975] |
| ar.L1 | -0.3688 | 0.002 | -239.160 | | 0.000 | -0.372 | -0.366 |
| sigma2 | 6933.2758 | 13.295 | 521.501 | | 0.000 | 6907.218 | 6959.333 |
| =================================================================================== | | | | | | | |
| Ljung-Box (L1) (Q): | | | 179.26 | | Jarque-Bera | (JB): | 1384605.64 |
| Prob(Q): | | | 0.00 | | Prob(JB): |  | 0.00 |
| Heteroskedasticity (H): | | | 0.43 | | Skew: |  | 0.07 |
| Prob(H) (two-sided): | | | 0.00 | | Kurtosis: |  | 34.89 |

===================================================================================

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [91]:

result.resid.plot() Out[91]:

<AxesSubplot:xlabel='Datetime'>

In [94]:

df\_sort['predict'] = result.predict(start=25000, end=30000) df\_sort[['Radiation', 'predict']].plot(figsize=(12,6))

plt.title('RMSE: %.4f'% np.sqrt(sum((result.predict(start=0) - df\_sort['Radiation']).dro pna()\*\*2/len(df\_sort))))

C:\Users\Swapnil\AppData\Local\Temp/ipykernel\_22496/3754496875.py:1: SettingWithCopyWarni ng:

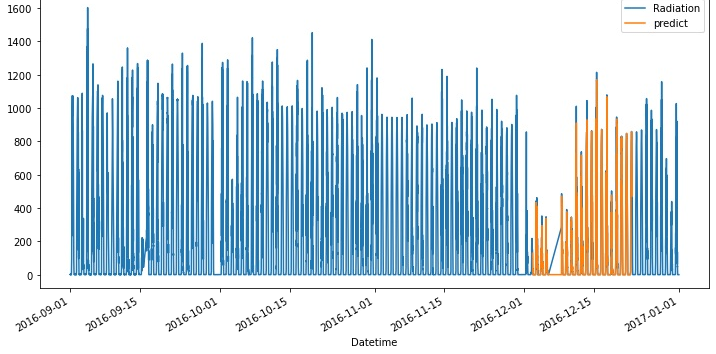
A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_g uide/indexing.html#returning-a-view-versus-a-copy

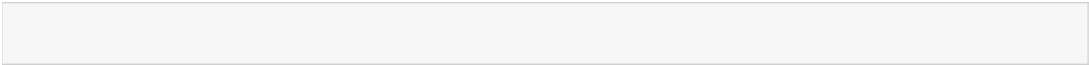
df\_sort['predict'] = result.predict(start=25000, end=30000)

Out[94]:

Text(0.5, 1.0, 'RMSE: 83.2637')



In [95]:



predictions = result.predict(start=df\_sort.shape[0],end=(df\_sort.shape[0]+10), dynamic=F alse)

C:\Users\Swapnil\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:376: Value Warning: No supported index is available. Prediction results will be given with an intege r index beginning at `start`.

warnings.warn('No supported index is available.'

In [96]:



predictions

Out[96]:

|  |  |
| --- | --- |
| 32686 | 1.202625 |
| 32687 | 1.205345 |
| 32688 | 1.204342 |
| 32689 | 1.204711 |
| 32690 | 1.204575 |
| 32691 | 1.204625 |
| 32692 | 1.204607 |
| 32693 | 1.204614 |
| 32694 | 1.204611 |
| 32695 | 1.204612 |
| 32696 | 1.204612 |
| Name: | predicted\_mean, dtype: float64 |

In [97]:



result.forecast(2016-12-31)

C:\Users\Swapnil\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:376: Value Warning: No supported index is available. Prediction results will be given with an intege r index beginning at `start`.

warnings.warn('No supported index is available.'

|  |  |
| --- | --- |
| Out[97]: |  |
| 32686 | 1.202625 |
| 32687 | 1.205345 |
| 32688 | 1.204342 |
| 32689 | 1.204711 |
| 32690 | 1.204575 |
| 32691 | 1.204625 |
| 32692 | 1.204607 |
| 32693 | 1.204614 |
| 32694 | 1.204611 |
| 32695 | 1.204612 |



|  |  |
| --- | --- |
| 32697 | 1.204612 |
| 32698 | 1.204612 |
| 32699 | 1.204612 |
| 32700 | 1.204612 |
| 32701 | 1.204612 |
| 32702 | 1.204612 |
| 32703 | 1.204612 |
| 32704 | 1.204612 |
| 32705 | 1.204612 |
| 32706 | 1.204612 |
| 32707 | 1.204612 |
| 32708 | 1.204612 |
| 32709 | 1.204612 |
| 32710 | 1.204612 |
| 32711 | 1.204612 |
| 32712 | 1.204612 |
| 32713 | 1.204612 |
| 32714 | 1.204612 |
| 32715 | 1.204612 |
| 32716 | 1.204612 |
| 32717 | 1.204612 |
| 32718 | 1.204612 |
| 32719 | 1.204612 |
| 32720 | 1.204612 |
| 32721 | 1.204612 |
| 32722 | 1.204612 |
| 32723 | 1.204612 |
| 32724 | 1.204612 |
| 32725 | 1.204612 |
| 32726 | 1.204612 |
| 32727 | 1.204612 |
| 32728 | 1.204612 |
| 32729 | 1.204612 |
| 32730 | 1.204612 |
| 32731 | 1.204612 |
| 32732 | 1.204612 |
| 32733 | 1.204612 |
| 32734 | 1.204612 |
| 32735 | 1.204612 |
| 32736 | 1.204612 |
| 32737 | 1.204612 |
| 32738 | 1.204612 |
| 32739 | 1.204612 |
| 32740 | 1.204612 |
| 32741 | 1.204612 |
| 32742 | 1.204612 |
| 32743 | 1.204612 |
| 32744 | 1.204612 |
| 32745 | 1.204612 |
| 32746 | 1.204612 |
| 32747 | 1.204612 |
| 32748 | 1.204612 |
| 32749 | 1.204612 |
| 32750 | 1.204612 |
| 32751 | 1.204612 |
| 32752 | 1.204612 |
| 32753 | 1.204612 |
| 32754 | 1.204612 |
| 32755 | 1.204612 |
| 32756 | 1.204612 |
| 32757 | 1.204612 |
| 32758 | 1.204612 |
| 32759 | 1.204612 |
| 32760 | 1.204612 |
| 32761 | 1.204612 |
| 32762 | 1.204612 |
| 32763 | 1.204612 |
| 32764 | 1.204612 |
| 32765 | 1.204612 |
| 32766 | 1.204612 |
| 32767 | 1.204612 |



|  |  |
| --- | --- |
| 32769 | 1.204612 |
| 32770 | 1.204612 |
| 32771 | 1.204612 |
| 32772 | 1.204612 |



|  |  |  |
| --- | --- | --- |
| 34641 | 1.204612 |  |
| 34642 | 1.204612 |  |
| 34643 | 1.204612 |  |
| 34644 | 1.204612 |  |
| 34645 | 1.204612 |  |
| 34646 | 1.204612 |  |
| 34647 | 1.204612 |  |
| 34648 | 1.204612 |  |
| 34649 | 1.204612 |  |
| 34650 | 1.204612 |  |
| 34651 | 1.204612 |  |
| 34652 | 1.204612 |  |
| 34653 | 1.204612 |  |
| 34654 | 1.204612 |  |
| 34655 | 1.204612 |  |
| 34656 | 1.204612 |  |
| 34657 | 1.204612 |  |
| 34658 | 1.204612 |  |
| Name: | predicted\_mean, | dtype: float64 |

In [ ]:

# Clustering

In [98]:



data['Datetime'] = '2016'+ '-' + data['Month'].map(str) +'-'+ data['Day'].map(str)

In [99]:



data['Datetime'] = pd.to\_datetime(data['Datetime'])

In [100]:



time\_y = y.values.reshape(-1,1)

In [101]:



time\_x = data.Datetime.values.reshape(-1,1)

In [102]:



time\_x

Out[102]:

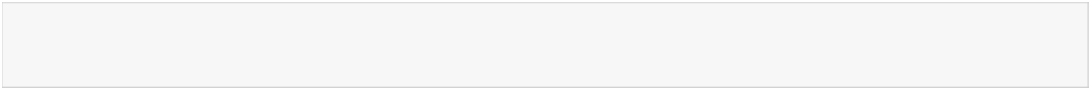
array([['2016-09-29T00:00:00.000000000'], ['2016-09-29T00:00:00.000000000'], ['2016-09-29T00:00:00.000000000'],

...,

['2016-12-01T00:00:00.000000000'], ['2016-12-01T00:00:00.000000000'],

['2016-12-01T00:00:00.000000000']], dtype='datetime64[ns]')

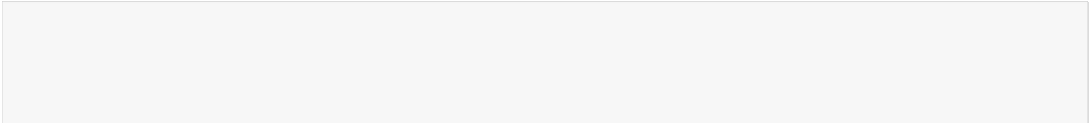
In [103]:



pca = PCA(n\_components=1).fit(time\_y) pca\_d = pca.transform(time\_y)

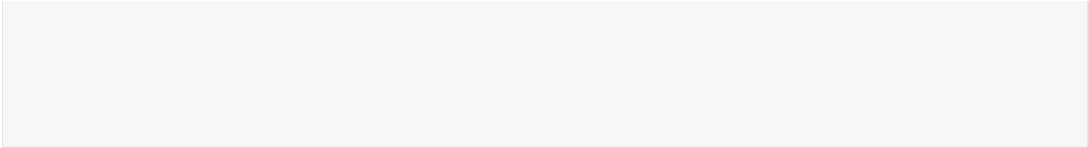
pca\_c = pca.transform(time\_x)

In [113]:



kmeans = KMeans(n\_clusters=3) kmeansoutput = kmeans.fit(time\_y) print(kmeansoutput)

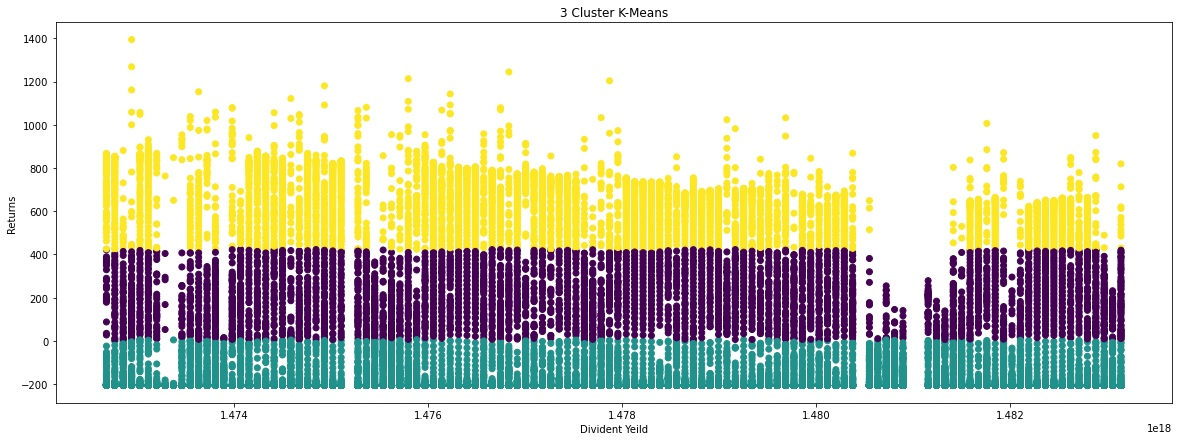




plt.figure('3 Cluster K-Means',figsize=(20,7)) plt.scatter(pca\_c, pca\_d, c=kmeansoutput.labels\_) plt.xlabel('Divident Yeild') plt.ylabel('Returns')

plt.title('3 Cluster K-Means') plt.show()

KMeans(n\_clusters=3)



In [105]:



time\_data = data.values

In [114]:



kmeans = KMeans(n\_clusters=5)

In [115]:



kmeans.fit(time\_y)

Out[115]:

KMeans(n\_clusters=5)

In [116]:



kmeans.cluster\_centers\_

Out[116]:

array([[935.89145074],

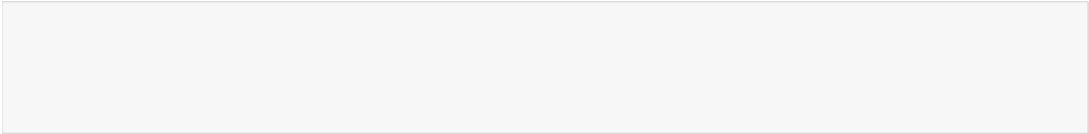
[ 8.92978152],

[442.37679795],

[220.20285 ],

[698.8432125 ]])

In [117]:



fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6)) ax1.set\_title("K Means")

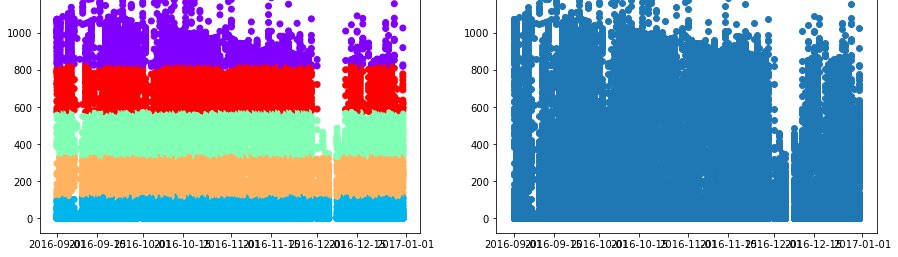
ax1.scatter(time\_data[:,15],time\_data[:,1], c=kmeans.labels\_, cmap='rainbow') ax2.set\_title("Original")

ax2.scatter(time\_data[:,15],time\_data[:,1], cmap='rainbow')

Out[117]:

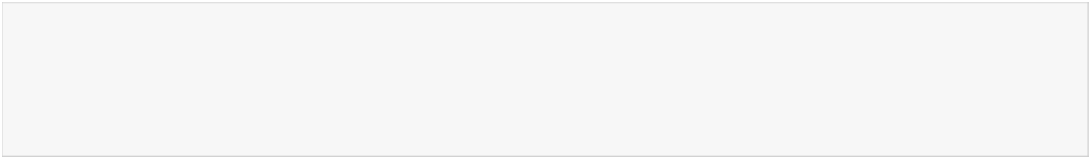
<matplotlib.collections.PathCollection at 0x18d85ea57c0>





In [ ]:

In [118]:



sum\_of\_squared\_distances = [] K = range(1, 15)

for k in K:

km = KMeans(n\_clusters=k) km = km.fit(time\_y)

sum\_of\_squared\_distances.append(km.inertia\_)

In [119]:



sum\_of\_squared\_distances

Out[119]: [3262066405.482444,

538723014.931757,

196673314.37805504,

113358937.02055618,

73496661.27774778,

51109958.52444067,

37478702.53404634,

29030945.217398193,

23737063.943424996,

19511299.467934847,

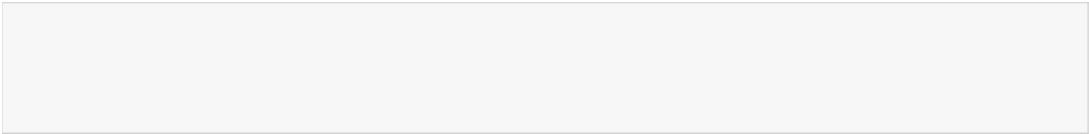
16254546.001732085,

13295978.177111834,

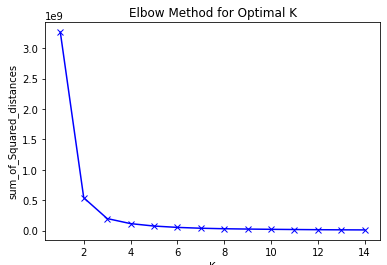
11183768.257167483,

9709391.515487079]

In [120]:

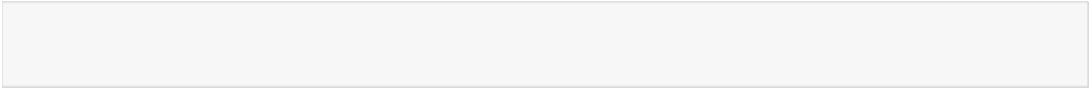


plt.plot(K, sum\_of\_squared\_distances, 'bx-') plt.xlabel("K") plt.ylabel("sum\_of\_Squared\_distances") plt.title("Elbow Method for Optimal K") plt.show()



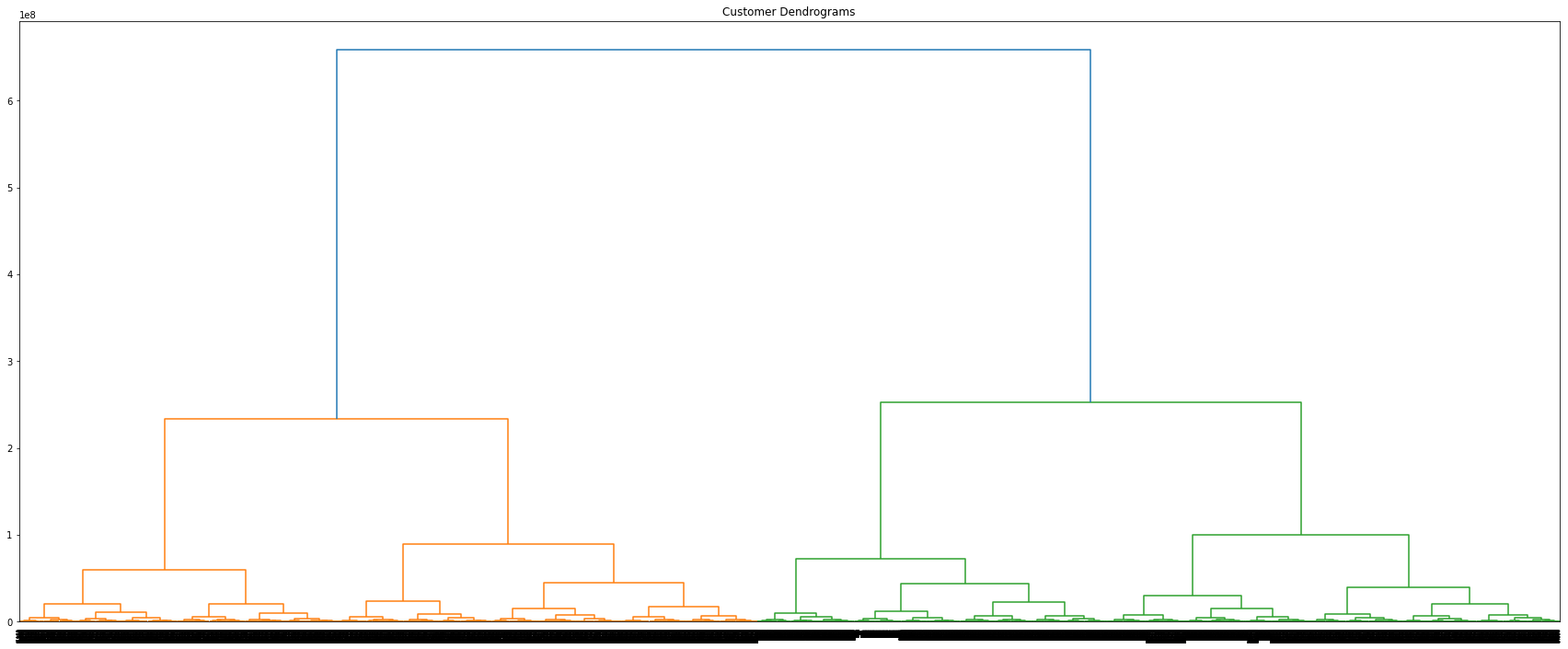


In [121]:

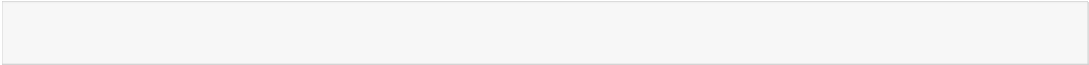


plt.figure(figsize=(30,12)) plt.title("Customer Dendrograms")

dend = shc.dendrogram(shc.linkage(x, method='ward'))



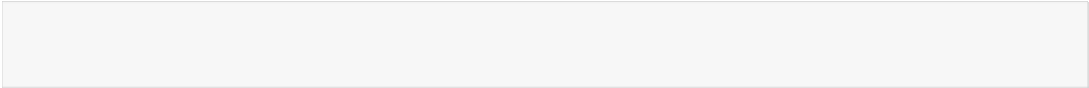
In [122]:



samples = x.values

mergings = linkage(samples, method='complete')

In [123]:

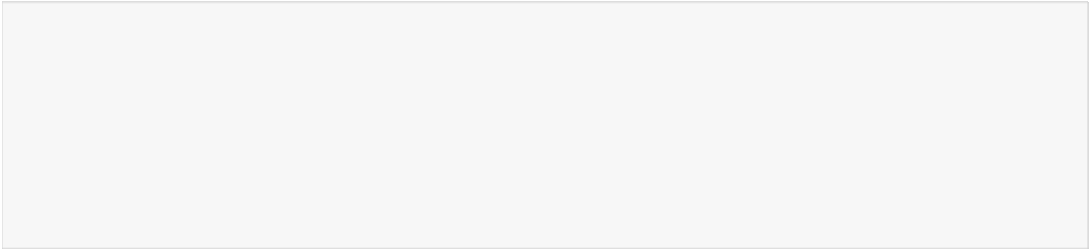


# Remove the Temperature from the DataFrame, save for later

d = data

varieties = list(d.pop('Temperature'))

In [124]:



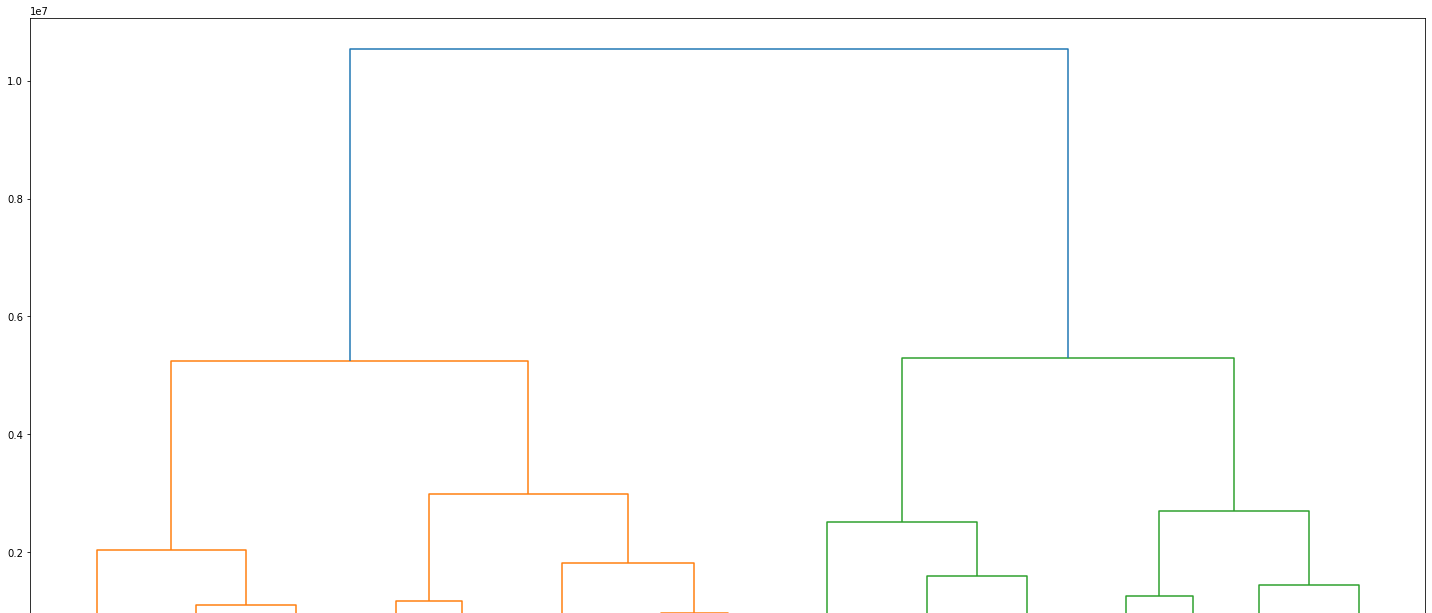
plt.figure(figsize=(25, 12)) dendrogram(mergings,

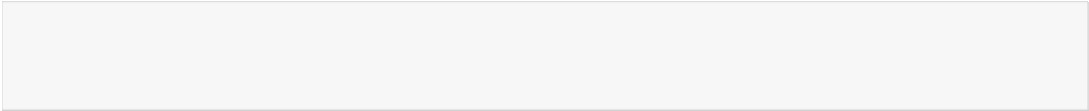
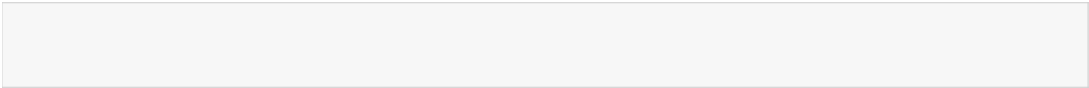
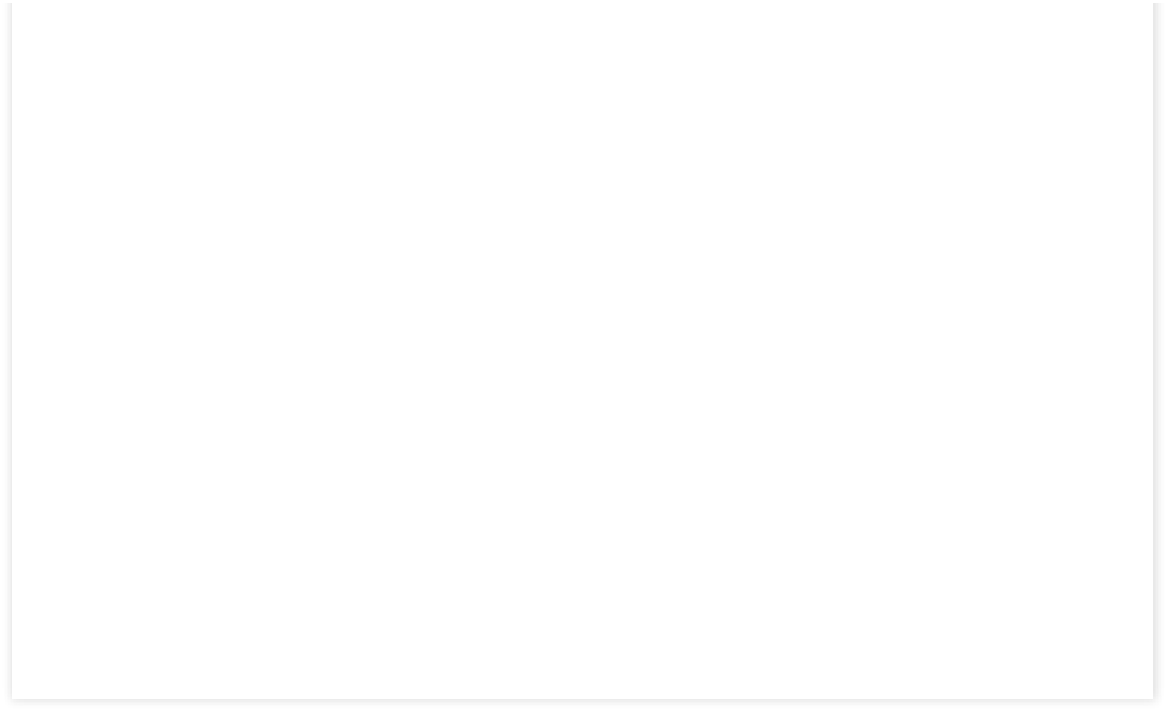
p=21,

truncate\_mode='lastp', labels=varieties, leaf\_rotation=90, leaf\_font\_size=12,

)

plt.show()





**Evaluation of the model**

In [ ]:

from sklearn.metrics import mean\_squared\_error error = mean\_squared\_error(test\_data, predictions) print('MSE error',error)

In [ ]:

from statsmodels.tools.eval\_measures import rmse

error = rmse(test\_data, predictions)

print(f'SARIMAX(4,1,3)&(1, 0, [], 12) preictions RMSE Error: {error:11.10}')

In [ ]:

**THA**N**K YOU**