**ELECTRICITY PRICE PREDICTION**

**MINIPROJECT REPORT**

***Submitted by***

**VISHNU A (953621104060)**

**SUMESHPATHI S (9536211040303)**

**MOHANKUMAR(953621104028)**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**RAMCO INSTITUTE OF TECHNOLOGY,**

**RAJAPALAYAM**

**JANUARY 2023**

**BONAFIDE CERTIFICATE**

Certified that this miniproject report **“ Electricity Price Prediction ”** is the bonafide work of **“ VISHNU A (953621104060), SUMESHPATHI S (9536211040303), MOHANKUMAR(953621104028) ”** who carried out the miniproject work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE**  Dr.K.Vijayalakshmi M.E., Ph.D.  **HEAD OF THE DEPARTMENT**  Department of Computer Science  and Engineering  Ramco Institute of Technology  North Venganallur Village  Rajapalayam – 626117 | **SIGNATURE**  Mr.K.VigneshSaravanan B.E.,M.Tech.  **Faculty In-charge**  Assistant Professor  Department of Computer Science and Engineering  Ramco Institute of Technology  North Venganallur Village  Rajapalayam– 626117 |

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ABSTRACT**

Predicting household electric power consumption is a task that involves forecasting the future energy consumption of a household based on historical data. The goal of this task is to enable households and energy providers to make informed decisions about energy usage and reduce overall energy consumption.

There are various approaches to predicting household electric power consumption, including statistical methods, machine learning techniques, and hybrid methods that combine both. Traditional statistical methods such as time series analysis, linear regression, and exponential smoothing can be used to model historical data and make predictions based on trends and patterns.

Machine learning techniques, such as neural networks, decision trees, and Random Forest can also be used to model the data and make predictions. These techniques can be used to learn patterns from historical data, and make predictions based on the learned patterns.

Hybrid methods that combine both statistical and machine learning techniques can also be used for prediction. These methods can be used to incorporate additional information such as weather data, and other external factors that may impact consumption.

Recently, the increasing amount of data available due to IoT devices and smart meter could be used to improve the prediction of consumption, also with the use of distributed systems, such as Hadoop and Spark, have made it possible to process big data and improve the accuracy of prediction.

Predicting household electric power consumption is a critical task for the energy industry, as it can provide insights into consumer behavior, and enables households and energy providers to make more informed decisions about energy usage. Ongoing research in this area will focus on developing more accurate and robust prediction models, incorporating new data sources, and leveraging advances in big data and IoT technologies.

**TABLE OF CONTENT:**

|  |  |  |
| --- | --- | --- |
|  | CONTENT | PAGE NO |
|  | 1.INTRODUCTION | 5 |
|  | 1.1 Electricity Price Prediction | 5 |
|  | 1.2 Project objectives | 5 |
|  | 1.3 Project specification | 6 |
|  | 2.SYSTEM SPECIFICATION | 6 |
|  | 2.1Hardware specification | 6 |
|  | 2.2 Software specification | 6 |
|  | 3. PACKAGES | 6 |
|  | 3.1 Numpy | 7 |
|  | 3.2 Pandas | 8 |
|  | 3.3 Matplotlib | 9 |
|  | 3.3.1 Matplotlib bar plot | 10 |
|  | 3.3.2 Matplotlib histogram | 11 |
|  | 4.APPENDIX | 12 |
|  | 4.1 Source code  4.1.1 LINEAR REGRESSION  4.1.2 MULTIPLE REGRESSION | 12  12  17 |
|  | 4.2.Screenshots | 18 |
|  | 5. CONCLUSION | 24 |
|  | 6.FUTURE WORK | 24 |
|  | 7. REFERENCE | 25 |
|  |  |  |

**1.INTRODUCTION**

**1.1 ELECTRICITY PRICE PREDICTION**

They were measurements of electric power consumption in one household with a one-minute sampling rate over almost 4 years. Different electrical quantities and some sub-metering values were available.

The data was collected between December 2006 and November 2010 and observations of power consumption within the household were collected every minute.

It is a multivariate series comprised of seven variables (besides the date and time); they are:

• Global\_active\_power: The total active power consumed by the household (kilowatts).

• Global\_reactive\_power: The total reactive power consumed by the household (kilowatts).

• voltage: Average voltage (volts).

• Global\_intensity: Average current intensity (amps).

• sub\_metering\_1: Active energy for the kitchen (watt-hours of active energy).

• sub\_metering\_2: Active energy for laundry (watt-hours of active energy).

• sub\_metering\_3: Active energy for climate control systems (watt-hours of active energy).

LSTM model was built to predict household electric power consumption. Dropout layers were added to improve the model.

The first year of data (resampled over an hour) was used to train the model and the rest of the data to test the model to reduce the computation time and get some results quickly.

**1.2 PROJECT OBJECTIVE**

The objective of the "Predict Household Electric Power Consumption" project is to use historical data on electric power consumption to train a model that can predict future consumption levels for a specific household. This could potentially be used to help households reduce their energy usage and costs, or for energy companies to more accurately forecast power demand.

**1.3 PROJECT SPECIFICATION**

This "Predict Household Electric Power Consumption" project would depend on the data and resources available, as well as the specific goals and constraints of the project. However, some possible specifications for such a project could include:

Data: The project would require historical data on electric power consumption for a specific household, including information on usage levels, dates and times, and possibly other factors such as weather conditions or appliance usage. This data would be used to train and evaluate the predictive model.

Evaluation: The model's performance would be evaluated using metrics such as mean absolute error or mean squared error, comparing the predicted consumption levels to the actual consumption levels in the test data set.

**2.SYSTEM SPECIFICATION**

**2.1Hardware specification**

* Processor : Intel dual core
* Processor speed: 1.04GHZ
* Ram : 1GB
* Moniter
* Keyboard
* Mouse

**2.2** **Software** **specification**

* OS
* Language : Python
* Compiler : Googlecolab

**3.PACKAGES**

**3.1 NUMPY**

* NumPy is a Python library used for working with arrays.
* It also has functions for working in domain of linear algebra, fourier transform, and matrices.
* NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
* NumPy stands for Numerical Python.

**INSTALLING NUMPY PACKAGE**

pip install numpy

## WHY USE NUMPY?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

**IMPORT NUMPY**

Once NumPy is installed, import it in your applications by adding the import keyword:

import numpy

## NUMPY AS np:

NumPy is usually imported under the np.

Create an np with the as keyword while importing:

import numpy as np

Now the NumPy package can be referred to as np instead of numpy.

**Example:**

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

## 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

**3.2 PANDAS**

* Pandas is a Python library used for working with data sets.
* It has functions for analyzing, cleaning, exploring, and manipulating data.
* The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

## Why Use Pandas

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Pandas gives you answers about the data. Like:

* Is there a correlation between two or more columns?
* What is average value?
* Max value?
* Min value?
* Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

**INSTALLING PANDAS PACKAGE**

pip install pandas

## Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas

Now Pandas is imported and ready to use

**Example:**

Importpandas

mydataset={'cars':["BMW","Volvo","Ford"],'passings':[3,7,2]}  
myvar=pandas.DataFrame(mydataset)  
print(myvar)

## Pandas as pd

Pandas is usually imported under the pd

Create an pd with the as keyword while importing:

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas.

**3.3 MATPLOTLIB**

* Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.
* As such, it offers a viable open source alternative to **MATLAB.** Developers can also use matplotlib’s APIs(Application Programming Interfaces) to embed plots inGUI applications.

A Python matplotlib script is structured so that a fewlines of code are all that is required in most instancesto generate a visual data plot.

The matplotlib scripting layer overlays two APIs:

* The pyplot API is a hierarchy of Python codeobjects topped by matplotlib.pyplot
* An OO (Object-Oriented) API collection of objectsthat can be assembled with greater flexibility thanpyplot. This API provides direct access to Matplotlib’sbackend layers.

**Matplotlib and Pyplot in Python :**

The pyplot API has a convenient MATLAB-style statefulinterface. In fact, matplotlib was originally written as an open source alternative for MATLAB. The OO API and its interface is more customizable and powerful than pyplot, but considered more difficult to use. As a result, the pyplot interface is more commonly used, and is referred to by default in this article.

Understanding matplotlib’s pyplot API is key to understanding how to work with plots:

* **matplotlib.pyplot.figure**: Figure is the top-level container. It includes everything visualized in a plot including one or more Axes.
* **matplotlib.pyplot.axes**: Axes contain most of the elements in a plot: Axis, Tick, Line2D, Text, etc., and sets the coordinates. It is the area in which data is plotted. Axes include the X-Axis, Y-Axis, and possibly a Z-Axis, as well.

**Installing Matplotlib :**

pip install matplotlib

**3.3.1 MATPLOTLIB BAR PLOT:**

A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. One of the axis of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

**Creating a bar plot:**

The matplotlib API in Python provides the bar() function which can be used in MATLAB style use or as an object-oriented API. The syntax of the bar() function to be used with the axes is as follows:- plt.bar(x, height, width, bottom, align).The function creates a bar plot bounded with a rectangle depending on the given parameters. Following is a simple example of the bar plot, which represents the number of students enrolled in different courses of an institute.

**EXAMPLE:**

**import numpy as np**

**import matplotlib.pyplot as plt**

**data = {'C':30, 'C++':15, 'Java':20,'Python':5}**

**courses = list(data.keys())**

**values = list(data.values())**

**fig = plt.figure(figsize = (10, 5))**

**plt.bar(courses, values, color ='darkblue',width = 0.4)**

**plt.xlabel("Courses offered")**

**plt.ylabel("No. of students enrolled")**

**plt.title("Students enrolled in different courses")**

**plt.show()**

**Output:**

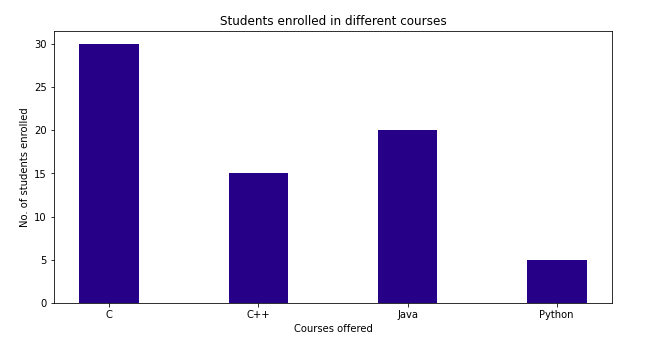


FIGURE:1-BAR CHART

**3.3.2 MATPLOTLIB HISTOGRAM:**

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph.

To construct a histogram, follow these steps −

* Bin the range of values.
* Divide the entire range of values into a series of intervals.
* Count how many values fall into each interval.

The bins are usually specified as consecutive, non-overlapping intervals of a variable.

The **matplotlib.pyplot.hist()** function plots a histogram. It computes and draws the histogram of x.

**4.APPENDIX**

**4.1 SOURCE CODE**

**4.1.1 LINEAR REGRESSION**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

The Pandas library to read in a csv file called "power.txt" and store it in a dataframe called "df".

Here is a breakdown of the various options being passed to the pd.read\_csv() function:

sep=';': The file is separated by semicolons, indicating that each value in the file is separated by a semicolon.

parse\_dates={'Date\_Time' : ['Date', 'Time']}: The 'Date' and 'Time' columns in the file are being combined and parsed as a single 'Date\_Time' column.

infer\_datetime\_format=True: This option tells pandas to try to infer the format of the datetime strings in the 'Date' and 'Time' columns.

low\_memory=False: This option tells pandas to not use a lower-memory data structure when reading the file, which could be useful if the file is large.

na\_values=['nan','?']: This option tells pandas to recognize 'nan' and '?' as missing values and fill them with NaN.

index\_col='Date\_Time': This option tells pandas to set the 'Date\_Time' column as the index for the dataframe.

The resulting dataframe "df" will have a DatetimeIndex and the 'Date' and 'Time' columns are combined and parsed as a single 'Date\_Time' column and it will have the date time column as index.

df=pd.read\_csv('power.txt',sep=';',parse\_dates={'Date\_Time':['Date','Time']}, infer\_datetime\_format=True,low\_memory=False,na\_values=['nan','?'],index\_col='Date\_Time)

Now let’s check the head of the data to see the data we are dealing with.

df.head()

Now let’s check the column of the data to see the data we are dealing with.

df.columns()

df.info()

df.describe()

Using the describe or info commands we can get a brief description of our dataset. This is important in order to enable us to understand the dataset we are working with.

The Pandas library to extract the 'Global\_active\_power' and 'Global\_reactive\_power' columns from the dataframe 'df' that was created in the previous step.

Here is a breakdown of the code:

X=df['Global\_active\_power']: This code is creating a new variable X, which is a subset of the dataframe 'df' and contains only the values in the 'Global\_active\_power' column.

y=df['Global\_reactive\_power']: This code is creating a new variable y, which is a subset of the dataframe 'df' and contains only the values in the 'Global\_reactive\_power' column.

X.head(): This code is displaying the first 5 rows of the X variable, which is a subset of the dataframe 'df' containing only the values in the 'Global\_active\_power' column

X=df['Global\_active\_power']

y=df['Global\_reactive\_power']

X.head()

y.head(): This code is displaying the first 5 rows of the y variable, which is a subset of the dataframe 'df' containing only the values in the 'Global\_active\_power' column

y.head()

'X' variable and displaying the shape of the transformed data. reshaping it into a 2-dimensional array with -1 rows and 1 column. The -1 tells numpy to infer the number of rows based on the number of items in the 1-dimensional array.

X = X.values.reshape(-1,1)

X.shape

The train\_test\_split() function from the sklearn.model\_selection module to split the 'X' and 'y' variables into training and testing sets. In this case, 70% of the data is being used for training and 30% of the data is being used for testing.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size = 0.7, test\_size = 0.3, random\_state = 100)

Printing the shape of the 'X', 'y', 'X\_train', 'X\_test', 'y\_train', and 'y\_test' variables.

print(X.shape): This line of code is printing the shape of the 'X' variable, which is the independent variable that represents the 'Global\_active\_power' column in the original dataframe 'df'. The shape of the variable is a tuple that shows the number of rows and columns in the data.

print(y.shape): This line of code is printing the shape of the 'y' variable, which is the dependent variable that represents the 'Global\_reactive\_power' column in the original dataframe 'df'. The shape of the variable is a tuple that shows the number of rows and columns in the data.

print(X\_train.shape): This line of code is printing the shape of the 'X\_train' variable, which is the independent variable that represents the 'Global\_active\_power' column in the training data. The shape of the variable is a tuple that shows the number of rows and columns in the data.

print(X\_test.shape): This line of code is printing the shape of the 'X\_test' variable, which is the independent variable that represents the 'Global\_active\_power' column in the testing data. The shape of the variable is a tuple that shows the number of rows and columns in the data.

print(y\_train.shape): This line of code is printing the shape of the 'y\_train' variable, which is the dependent variable that represents the 'Global\_reactive\_power' column in the training data. The shape of the variable is a tuple that shows the number of rows and columns in the data.

print(y\_test.shape): This line of code is printing the shape of the 'y\_test' variable, which is the dependent variable that represents the 'Global\_reactive\_power' column in the testing data. The shape of the variable is a tuple that shows the number of rows and columns in the data.

print(X.shape)

print(y.shape)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

Creating an instance of a linear regression model and fitting it to the training data.

regr = linear\_model.LinearRegression()

regr.fit(X\_train,y\_train)

The coef\_ attribute of the linear regression model to get the coefficients of the independent variable.

regr.coef\_

The intercept\_ attribute of the linear regression model to get the y-intercept of the linear regression line.

regr.intercept\_

The Matplotlib library to create a scatter plot of the training data, with the linear regression line plotted on top of the data.

import matplotlib.pyplot as plt

plt.scatter (X\_train, y\_train)

plt.plot(X\_train,0.05930 + (0.038187\*X\_train), 'r')

plt.show()

The linear regression model to make predictions on the testing data and creating residuals from the predicted and actual values of the dependent variable.

y\_pred = regr.predict(X\_test): This line of code is using the predict() method of the linear regression model to make predictions on the independent variable 'X\_test'. The predicted values of the dependent variable are stored in the variable 'y\_pred'.

res = (y\_test - y\_pred): This line of code is creating a new variable 'res' that stores the difference between the actual values of the dependent variable 'y\_test' and the predicted values 'y\_pred'. This difference is also called residuals and it represents the error or deviation of the predicted values from the actual values.#

Predicting y\_value using teting data of X

y\_pred = regr.predict(X\_test)

# Creating residuals from the y\_train and y\_pred

res = (y\_test - y\_pred)

r\_squared = r2\_score(y\_test, y\_pred): This line of code is using the r2\_score() function to calculate the R-squared value for the linear regression model. The function takes two arguments: 'y\_test' and 'y\_pred', which are the actual and predicted values of the dependent variable, respectively. The R-squared value ranges from 0 to 1, with a higher value indicating a better fit of the model to the data.

r\_squared: This line of code is printing the R-squared value.

r\_squared = r2\_score(y\_test, y\_pred)

r\_squared

The mean\_squared\_error() and mean\_absolute\_error() functions from the sklearn.metrics module to calculate the mean squared error (MSE) and mean absolute error (MAE) of the linear regression model's predictions.

print('Mean squared error: %.2f'% mean\_squared\_error(y\_test, y\_pred)): This line of code is using the mean\_squared\_error() function to calculate the MSE of the model's predictions. The function takes two arguments: 'y\_test' and 'y\_pred', which are the actual and predicted values of the dependent variable, respectively. MSE is a measure of the average squared difference between the predicted and actual values.

print('Mean Absolute Error: %.2f'% mean\_absolute\_error(y\_test, y\_pred)): This line of code is using the mean\_absolute\_error() function to calculate the MAE of the model's predictions. The function takes two arguments: 'y\_test' and 'y\_pred', which are the actual and predicted values of the dependent variable, respectively. MAE is a measure of the average difference between the predicted and actual values.

Both MSE and MAE are common performance metrics for evaluating the accuracy of a regression model's predictions

print('Mean squared error: %.2f'% mean\_squared\_error(y\_test, y\_pred))

print('Mean Absolute Error: %.2f'% mean\_absolute\_error(y\_test, y\_pred))

**4.1.2 MULTIPLE REGRESSION**

SIMLARLY TO LINEAR REGRESSION

Multiple regression is a statistical technique that is used to model the relationship between multiple independent variables and a single dependent variable. It is an extension of simple linear regression, which is used to model the relationship between one independent variable and one dependent variable. In multiple regression, the goal is to find the best linear combination of the independent variables that can predict the value of the dependent variable.

Multiple regression is used when there are multiple independent variables that could affect the dependent variable. It allows for the analysis of how each independent variable is associated with the dependent variable, and how the independent variables are associated with one another. The multiple regression model is represented by an equation of the form:

y = b0 + b1x1 + b2x2 + ... + bn\*xn

where y is the dependent variable, x1, x2, ... xn are the independent variables, b0 is the y-intercept, and b1, b2, ... bn are the coefficients of the independent variables.

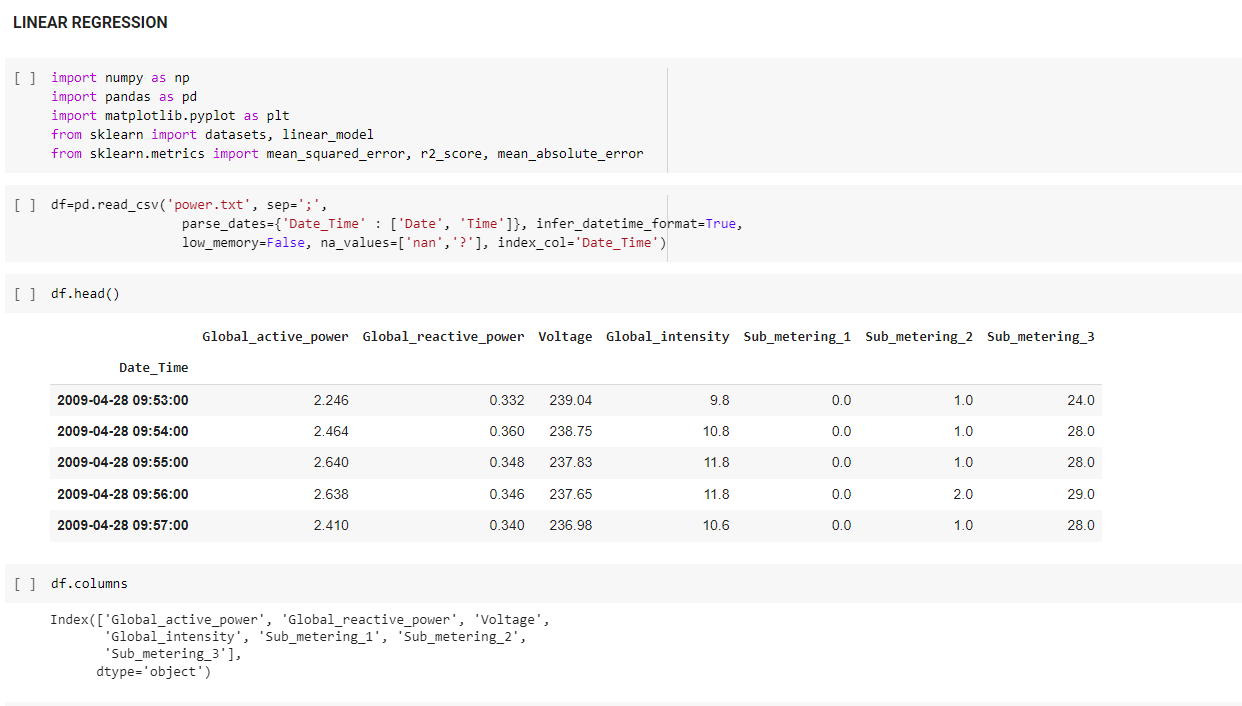
Multiple regression is a powerful tool that can be used to analyze complex relationships between variables and make predictions. It is widely used in fields such as economics, finance, marketing, and social sciences.

X=df[['Global\_active\_power','Global\_reactive\_power','Voltage']]

y=df['Global\_intensity']

X.head()

**4.2 SCREENSHOT**

****

FIFURE 2

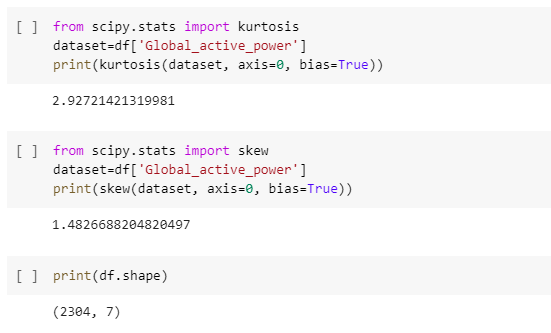


FIGURE 3

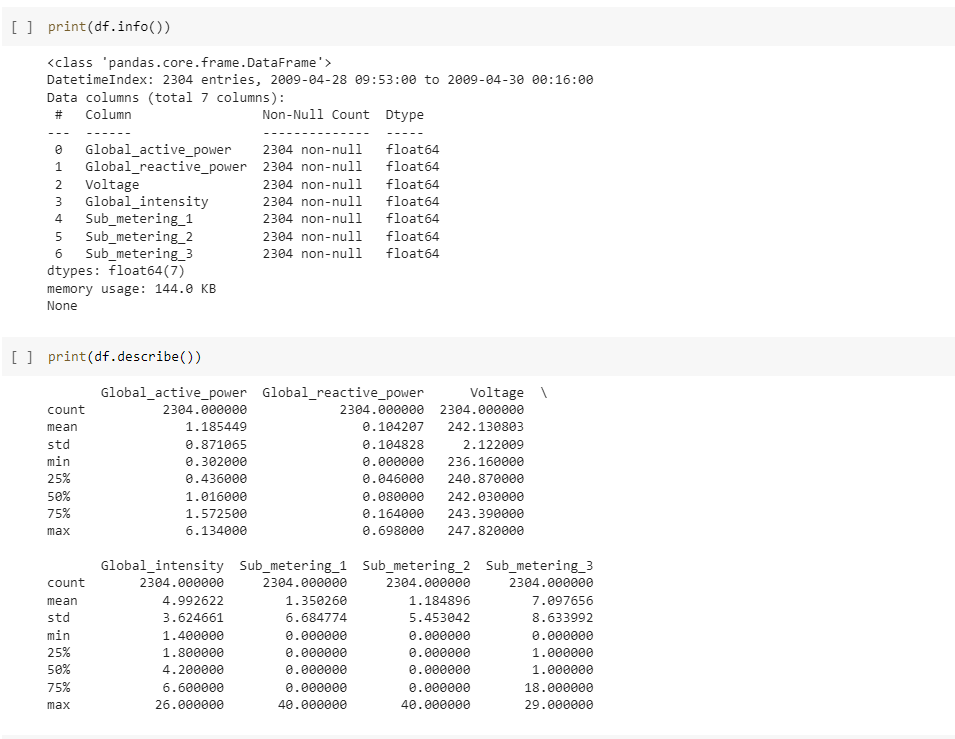


FIGURE 3

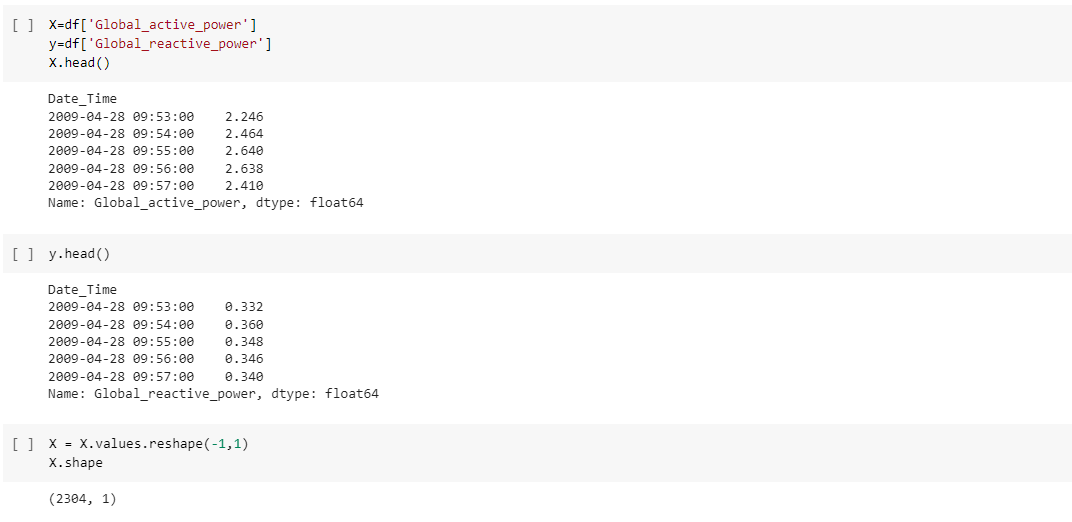


FIGURE 4



FIGURE 5

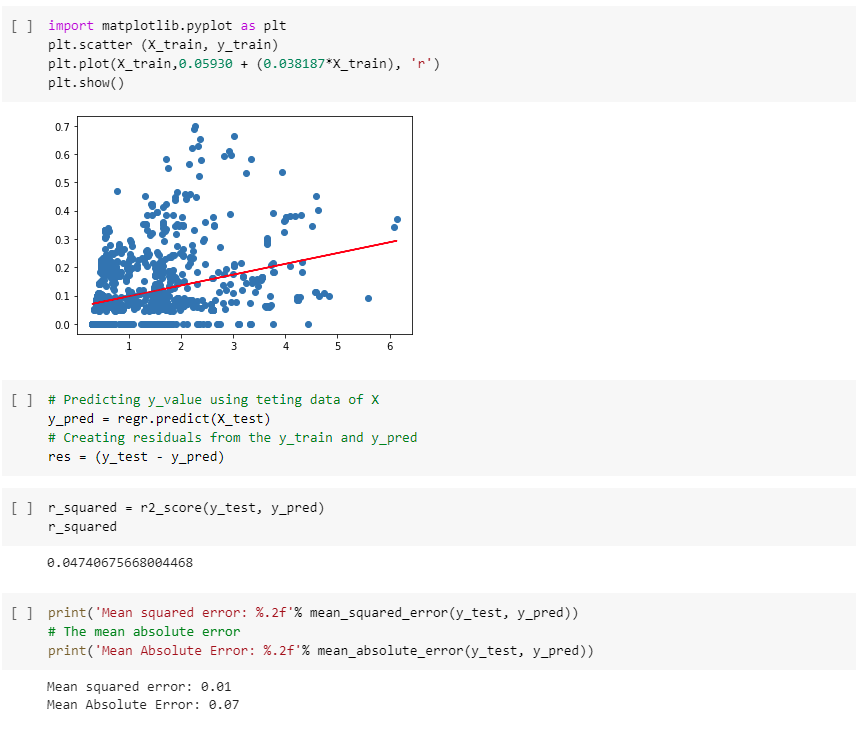


FIGURE 6

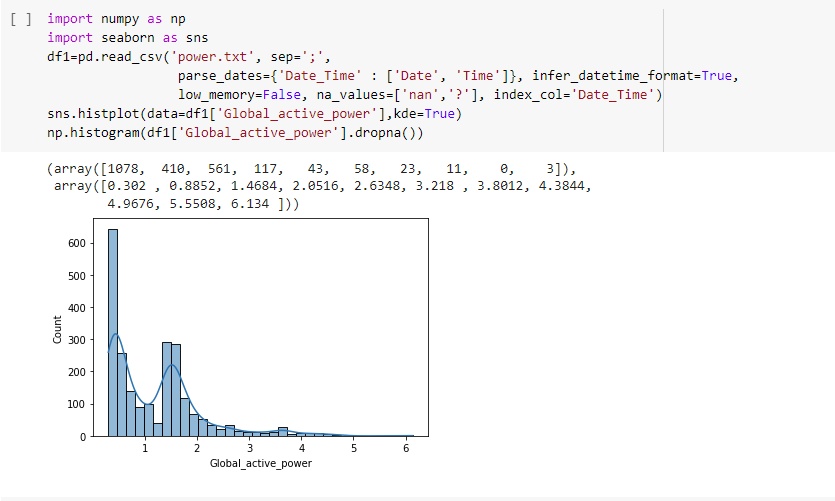


FIGURE 7

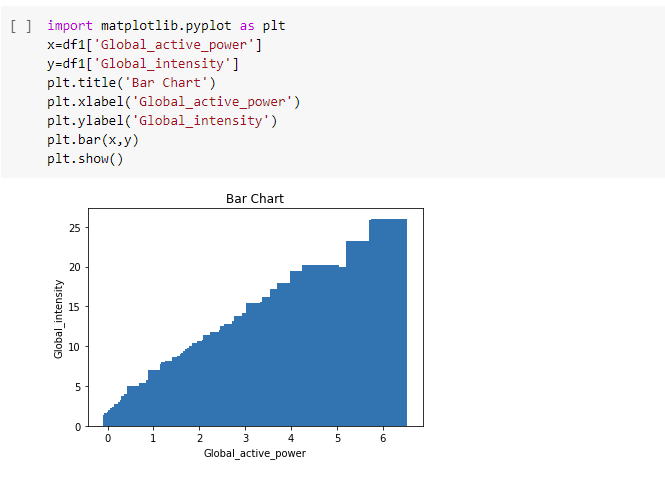


FIGURE 8

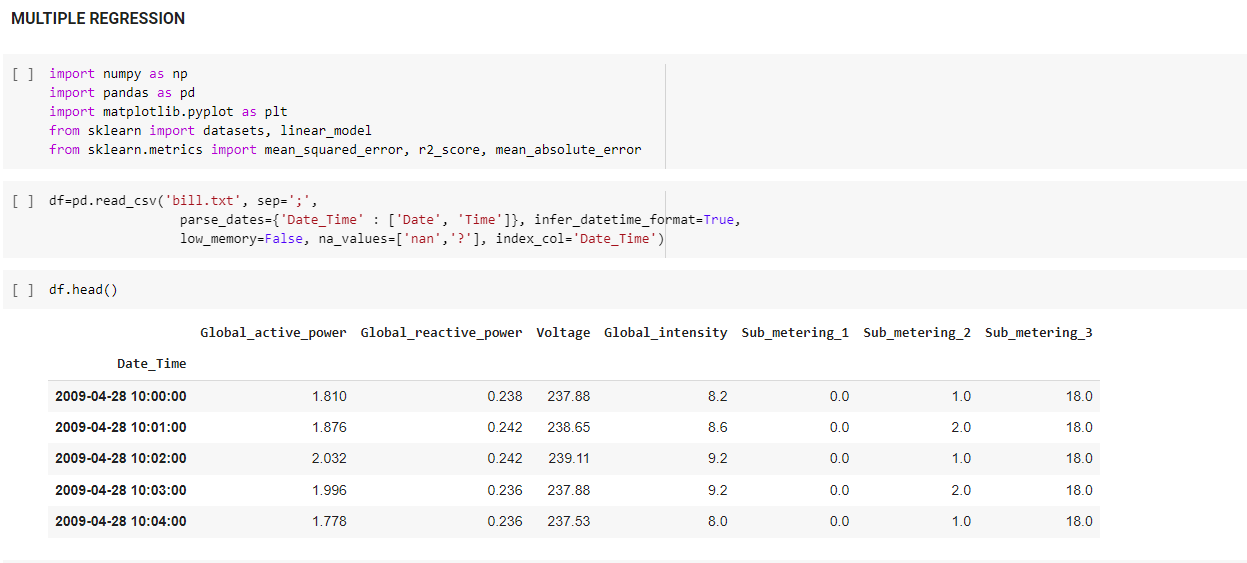


FIGURE 9

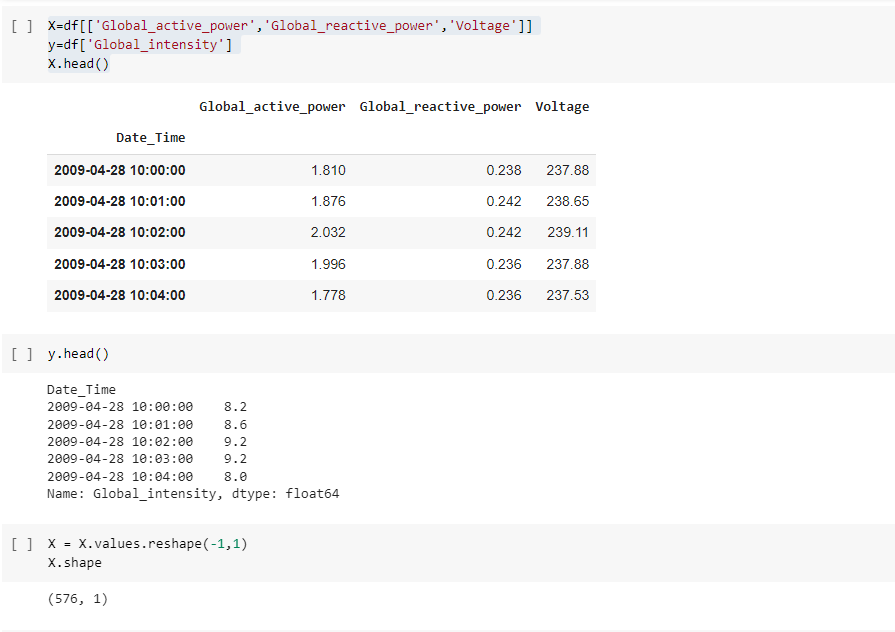


FIGURE 10

****

FIGURE 11

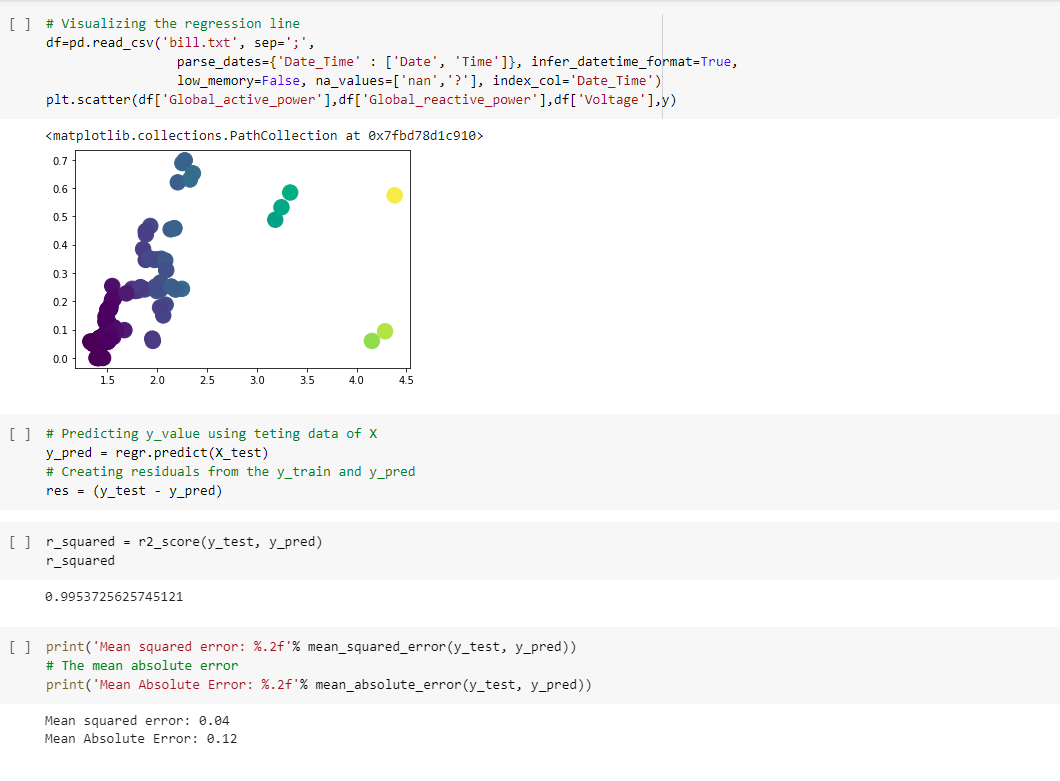


FIGURE 12

**5. CONCLUSION**

In conclusion, the data used for this project was a multivariate time series of electric power consumption in one household over a period of almost 4 years. The data was collected at a one-minute sampling rate and included various electrical quantities and sub-metering values. A Long Short-Term Memory (LSTM) model was built to predict household electric power consumption, with dropout layers added to improve the model's performance. The model was trained using data from the first year and tested on the remaining data. The results of this study could be used to gain insight into household energy consumption patterns and potentially aid in developing strategies for reducing energy use.

**6.FUTURE WORK**

Future work on the project of predicting household electric power consumption could include:

* Expanding the dataset to include multiple households, to improve the generalizability of the model.
* Incorporating external factors, such as weather data, to improve the accuracy of the predictions.
* Investigating the use of other machine learning models, such as Random Forest or XGBoost, to see if they can improve the performance of the model.
* Using more recent data to train and test the model, to see how well the model performs on more recent data.
* Investigate the use of more advanced techniques in deep learning such as attention mechanism, transformer and fine-tuning pre-trained models
* Study the impact of different parameter tuning on the model's performance.
* Use the model to develop an energy management system that can be used to control and optimize energy consumption in a household.
* Building a real-time prediction model using the data stream of household electricity consumption.
* Evaluating the model's performance using more comprehensive evaluation metrics, such as mean absolute error or mean squared error.

**7.REFERENCE**

<https://www.kaggle.com/code/itzmevishnua/notebook64e9fa6d88/edit/run/114625831>

<https://colab.research.google.com/drive/1UoxJXF--rFakVhDQXgD3mSoLYYTuwdGh#scrollTo=wp5cqtIrxUER>

<https://github.com/chandimap/LSTM-To-Predict-Household-Electric-Power-Consumption/blob/master/Electric_Power.ipynb>

|  |  |
| --- | --- |
| **PERFORMANCE** |  |
| **VIVAVOCE** |  |
| **MINI PROJECT** |  |
| **TOTAL** |  |