

Climate Impact on Crop Production

A Project Report

In partial fulfilment of the requirements for the award of the degree

Bachelor of Computer Application

in

Computer Application

Under the guidance of

Joyjit Guha Biswas

by

Himangsu Sen

Anirban Banerjee

Susanta Gorai

Asansol Engineering College,

Asansol

In association with



PX82+66Q, Vivekananda Sarani, Opp. IndianOil, Kanyapur, Asansol, West Bengal 713305,

Asansol, West Bengal 713305

Title of the Project: Climate Impact on Crop Production

Project Members: Himangsu Sen, Anirban Banerjee, Susanta Gorai

Name of the guide: Mr. Joyjit Guha Biswas

Address: PX82+66Q, Vivekananda Sarani, Opp. IndianOil,
Kanyapur, Asansol, West Bengal 713305

Project Version Control History:

Version	Primary Author	Description of Version	Date Completed
Final	Himangsu Sen Anirban Banerjee Susanta Gorai	Project Report	16 th October, 2025

Signature of Team Member

Signature of Approver

Date:

Date:

For Office Use Only

Mr. Joyjit Guha Biswas
Project Proposal Evaluator

Declaration

We hereby declare that the project work being presented in the project proposal entitled “**Climate Impact on Crop Production**” in partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Application at Asansol Engineering College, Asansol, West Bengal, is an authentic work carried out under the guidance of Mr. Joyjit Guha Biswas. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date: 17/10/2025

Name of the Students:

Anirban Banerjee
Himangsu Sen
Susanta Gorai

Signature of the student:



PX82+66Q, Vivekananda Sarani, Opp. IndianOil, Kanyapur,
Asansol, West Bengal 713305

Certificate

This is to certify that this proposal of minor project entitled “**Climate Impact on Crop Production**” is a record of Bonafide work, carried out by Anirban Banerjee, Himangsu Sen & Susanta Gorai under my guidance at Ardent Computech PVT. LTD. In my opinion, the report in its present form is in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Application and as per regulations of the Ardent®. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide / Supervisor

Mr. Joyjit Guha Biswas

Project Engineer

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V,
Kolkata, West Bengal 700091

Contents:

- Overview
- History of Python
- Environment Setup
- Basic Syntax
- Variable Types
- Functions
- Modules
- Packages
- Artificial Intelligence
 - Deep Learning
 - Machine Learning
- Machine Learning
 - Supervised and Unsupervised Learning
 - NumPy
 - Scikit-learn
 - Pandas
 - Regression Analysis
 - Matplotlib
 - Clustering
- Climate Impact on Crop Production

Overview:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to Perl and PHP.

Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python:

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Features of Python:

Easy-to-learn: Python has few Keywords, simple structure and clearly

defined syntax. This allows a student to pick up the language quickly.

Easy-to-Read: Python code is more clearly defined and visible to the eyes. **Easy -to-Maintain:** Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on the wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low level modules to the python interpreter. These modules enables programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It support functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte code for building large applications.
- It provides very high level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collections.
- It can be easily integrated with C, C++, COM, Active X, CORBA and JAVA.

Environment Setup:

- 64-Bit OS
- Install Python 3
- Setup virtual environment
- Install Packages
- Install Flask

Basic Syntax of Python Program:

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in `print ("Hello, Python!");`.

However in Python version 2.4.3, this produces the following result –

Hello, Python!

Python Identifiers:

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (`_`) followed by zero or more letters, underscores and digits (0 to 9). Python does not allow punctuation characters such as `@`, `$`, and `%` within identifiers. Python is a case sensitive programming language.

Python Keywords:

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

For example:

And, exec, not, Assert, finally, orBreak, for, pass, Class, from, print, continue, global, raisedef, if, return, del, import, tryelif, in, while else, is, with, except, lambda, yield.

Lines & Indentation:

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True: print "True"
```

```
else:
```

```
print "False"
```

Command Line Arguments:

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h –

```
$ python-h
```

```
usage: python [option]...[-c cmd|-m mod | file |-][arg]...
```

Options and arguments (and corresponding environment variables):

-c cmd: program passed in as string(terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit [etc.]

Variable Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
counter=10          # An integer assignment
weight=10.60        # A floating point
name="Ardent" # A string
```

Multiple Assignment:

Python allows you to assign a single value to several variables simultaneously.

For example –

```
a = b = c = 1
```

```
a,b,c      =      1,2,"hello"
```

Standard Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types –

- String

- List

- Tuple

- Dictionary

- Number

Data Type Conversion:

Sometimes, you may need to perform conversions between the built-in types.

To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another.

Sr.No.	Function & Description
1	int(x [,base]) Converts x to an integer. base specifies the base if x is a string
2	long(x [,base]) Converts x to a long integer. base specifies the base if x is a string.
3	float(x) Converts x to a floating-point number.
4	complex(real [,imag]) Creates a complex number.
5	str(x) Converts object x to a string representation.
6	repr(x) Converts object x to an expression string.
7	eval(str) Evaluates a string and returns an object.
8	tuple(s) Converts s to a tuple.
9	list(s) Converts s to a list.

Functions:

Defining a Function:

- `def functionname(parameters):`
 `"function_docstring"`
 `function_suite`
 `return [expression]`

Pass by reference vs Pass by value:

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

Function definition is here

```
def changeme(mylist):  
    "This changes a passed list into this function"  
    mylist.append([1,2,3,4]);  
    print"Values inside the function: ",mylist  
    return
```

Now you can call changeme function

```
mylist=[10,20,30];  
changeme(mylist);  
print"Values outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result –

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope . For Example-

```
total=0;           # This is global variable.
```

Function definition is here

```
def sum( arg1, arg2 ):
```

```
# Add both the parameters and return them."
```

```
total= arg1 + arg2;          # Here total is local variable.  
print"Inside the function local total : ", total  
return total;
```

```
# Now you can call sum functionsum(10,20);  
print"Outside the function global total : ", total
```

When the above code is executed, it produces the following result –

```
Inside the function local total : 30  
Outside the function global total : 0
```

Modules:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference .

The Python code for a module named `aname` normally resides in a file named `aname.py`. Here's an example of a simple module, `support.py`

```
def print_func( par ):  
    print"Hello : ",  
    par return
```

The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax –

```
import module1[, module2[,... moduleN]
```

Packages:

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-subpackages, and so on.

Consider a file Pots.py available in Phone directory. This file has following line of source code –

```
def Pots():  
    print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above –

Phone/Isdn.py file having function Isdn()

Phone/G3.py file having function G3()

Now, create one more file _init_.py in Phone directory –

Phone/_init_.py

To make all of your functions available when you've imported Phone, you need to put explicit import statements in _init_.py as follows –

```
from Pots import Pots  
from Isdn import Isdn  
from G3 import
```

Numpy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

Pandas:

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

Library features:

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.

- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation.

Python Speech Features:

This library provides common speech features for ASR including MFCCs and filterbank energies. If you are not sure what MFCCs are, and would like to know more have a look at this MFCC

tutorial: <http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>.

You will need numpy and scipy to run these files. The code for this project is available at https://github.com/jameslyons/python_speech_features .

Supported features:

- `python_speech_features.mfcc()` - Mel Frequency Cepstral Coefficients
- `python_speech_features.fbank()` - Filterbank Energies
- `python_speech_features.logfbank()` - Log Filterbank Energies
- `python_speech_features.ssc()` - Spectral Subband Centroids

To use MFCC features:

```
from python_speech_features import mfcc
from python_speech_features import logfbank
import scipy.io.wavfile as wav
```

```
(rate,sig) = wav.read("file.wav")
```

```
mfcc_feat = mfcc(sig,rate)
fbank_feat = logfbank(sig,rate)
```

```
print(fbank_feat[1:3,:])
```


OS: Miscellaneous operating system interfaces

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see `open()`, if you want to manipulate paths, see the `os.path` module, and if you want to read all the lines in all the files on the command line see the `fileinput` module. For creating temporary files and directories see the `tempfile` module, and for high-level file and directory handling see the `shutil` module.

Notes on the availability of these functions:

The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function `os.stat(path)` returns stat information about `path` in the same format (which happens to have originated with the POSIX interface).

Extensions peculiar to a particular operating system are also available through the `os` module, but using them is of course a threat to portability.

All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

On VxWorks, `os.fork`, `os.execv` and `os.spawn*p*` are not supported.

Pickle: Python object serialization

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. “*Pickling*” is the process whereby a Python object hierarchy is converted into a byte stream, and “*unpickling*” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” 1 or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

Operator: Standard operators as functions

The operator module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, `operator.add(x, y)` is equivalent to the expression `x+y`. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The variants without the double underscores are preferred for clarity.

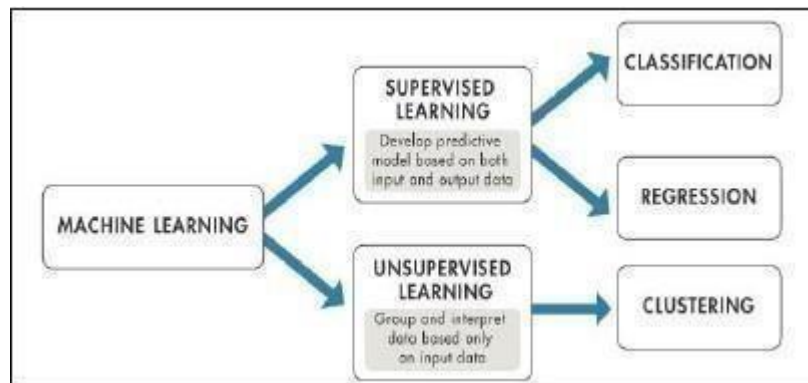
The functions fall into categories that perform object comparisons, logical operations, mathematical operations and sequence operations.

Tempfile: Generate temporary files and directories

This module creates temporary files and directories. It works on all supported platforms. `TemporaryFile`, `NamedTemporaryFile`, `TemporaryDirectory`, and `SpooledTemporaryFile` are high-level interfaces which provide automatic cleanup and can be used as context managers. `mkstemp()` and `mkdtemp()` are lower-level functions which require manual cleanup.

All the user-callable functions and constructors take additional arguments which allow direct control over the location and name of temporary files and directories. Files names used by this module include a string of random characters which allows those files to be securely created in shared temporary directories. To maintain backward compatibility, the argument order is somewhat odd; it is recommended to use keyword arguments for clarity.

Introduction to Machine Learning:



Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:-

Supervised Learning:

Supervised learning is the machine learning task of inferring a function from labeled training data.^[1] The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value.

A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal

scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

Unsupervised Learning:

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabelled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining key features of the data.

Linear Regression Algorithm:

Linear regression is a simple and widely used algorithm in machine learning and statistics for predicting continuous numerical values based on input features. It fits a linear equation to the data, where the relationship between the dependent variable (target) and one or more independent variables (features) is modeled as a straight line. The main goal of linear regression is to find the best-fitting line that minimizes the difference between the predicted values and the actual values of the target variable.

The equation of a linear regression model can be represented as:

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$$

where:

y is the predicted value (the dependent variable).

b₀ is the intercept term, representing the value of y when all input features (x₁, x₂, ..., x_n) are zero.

b_1, b_2, \dots, b_n are the coefficients (also known as slopes) of the respective input features (x_1, x_2, \dots, x_n).

The goal of training a linear regression model is to find the best values for the coefficients ($b_0, b_1, b_2, \dots, b_n$) that minimize the error between the predicted values and the actual target values in the training data. The most common method used to find these coefficients is called "Ordinary Least Squares" (OLS), where the sum of the squared differences between the predicted and actual values is minimized.

Once the coefficients are determined, the model can be used to make predictions on new data by simply plugging the feature values into the linear equation.

Linear regression is suitable for problems where the relationship between the target variable and the features is approximately linear. However, it may not perform well when the relationship is highly non-linear or if there are complex interactions between the features.

Algorithm:

- Data Collection
- Data Formatting
- Model Selection
- Training
- Testing

Data Collection: We have collected data sets of movies from online website. We have downloaded the .csv files in which information was present.

Data Formatting: The collected data is formatted into suitable data sets.

Model Selection: We have selected different models to minimize the error of the predicted value. The different models used are Linear Regression Linear Model.

Training: The data sets was divided such that x_{train} is used to train the model with corresponding x_{test} values and some y_{train} kept reserved for testing.

Testing: The model was tested with y_{train} and stored in $y_{predict}$. Both y_{train} and $y_{predict}$ was compared.

Climate Impact on Crop Production

Climate plays a crucial role in determining crop growth and productivity. Factors such as temperature, rainfall, humidity, and extreme weather events directly affect soil health, plant development, and yield. Understanding these impacts helps farmers plan sowing and harvesting schedules, optimize irrigation and fertilization, and adopt sustainable practices. Data-driven analysis and predictive modeling can provide insights into crop performance under varying climatic conditions, enabling better decision-making and improving overall agricultural productivity.

Weather prediction is a complex task that involves numerous variables and non-linear interactions. While linear regression can be a useful tool for certain simple weather forecasting scenarios, it may not be sufficient for accurate predictions over more extended periods or complex crop production.

Here are the steps to create a simple linear regression model for crop production:

Define the Problem: Clearly identify what you want to predict—in this case, crop yield. Decide which factor(s) such as rainfall, temperature, or fertilizer usage will be used as input features to model the relationship with yield.

Collect Data: Gather historical data on crop production along with relevant climate and environmental factors. Reliable data sources include government agricultural databases, research studies, or IoT sensors installed on farms.

Prepare Data: Clean the dataset by handling missing values, removing outliers, and converting categorical variables into numerical formats if needed. Then, split the dataset into training and testing sets to validate the model's performance.

Visualize Data: Use plots such as scatter plots or line graphs to explore the relationship between the selected features and crop yield. Visualization helps identify trends and ensures that a linear regression model is appropriate.

Build Model: Use a machine learning library (like scikit-learn in Python) to train a linear regression model on the training dataset. The model learns the relationship between input features and crop yield.

Make Predictions: Apply the trained model to the test dataset to predict crop yields. This step checks how well the model can generalize to new, unseen data.

Evaluate Model: Measure the model's performance using evaluation metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or R^2 score. These metrics indicate the accuracy and reliability of the predictions.

Actual codes for Crop Production:

Imports:

```
import os
import joblib
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Load the dataset:

	crop_type	region	temperature_c	rainfall_mm	humidity_percent	soil_type	production_tonnes_per_hectare
0	Millets	Uttarakhand	34.4	500	51	Black Soil	1.77
1	Cotton	Maharashtra	31.1	926	47	Black Soil	2.20
2	Cotton	Telangana	27.8	793	51	Red Soil	2.56
3	Tea	Gujarat	14.6	1768	73	Mountain	2.90
4	Sugarcane	Maharashtra	26.7	1192	69	Alluvial	117.70
...
995	Rice	Haryana	20.7	2512	69	Clay	3.81
996	Sugarcane	Uttar Pradesh	23.0	1302	68	Loam	95.78
997	Sugarcane	Jammu and Kashmir	33.0	1532	63	Alluvial	95.93
998	Soybean	Assam	31.7	945	60	Clay Loam	2.84
999	Pulses	Uttar Pradesh	25.4	431	51	Loam	1.58

1000 rows × 7 columns

Checking information about the dataset:

```
✓ 0s weather.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16743 entries, 0 to 16742
Data columns (total 14 columns):
 #   Column                                Non-Null Count  Dtype  
---  -
 0   Data.Precipitation                    16743 non-null  float64
 1   Date.Full                             16743 non-null  object  
 2   Date.Month                            16743 non-null  int64   
 3   Date.Week of                          16743 non-null  int64   
 4   Date.Year                             16743 non-null  int64   
 5   Station.City                          16743 non-null  object  
 6   Station.Code                          16743 non-null  object  
 7   Station.Location                      16743 non-null  object  
 8   Station.State                        16743 non-null  object  
 9   Data.Temperature.Avg Temp             16743 non-null  int64   
10   Data.Temperature.Max Temp             16743 non-null  int64   
11   Data.Temperature.Min Temp             16743 non-null  int64   
12   Data.Wind.Direction                   16743 non-null  int64   
13   Data.Wind.Speed                       16743 non-null  float64
dtypes: float64(2), int64(7), object(5)
memory usage: 1.8+ MB
```

Checking null values:

```
crop.isnull().sum()
```

```
crop_type          0
region             0
temperature_c      0
rainfall_mm        0
humidity_percent   0
soil_type          0
production_tonnes_per_hectare  0
dtype: int64
```






Describe the dataset:

```
crop.describe()
```

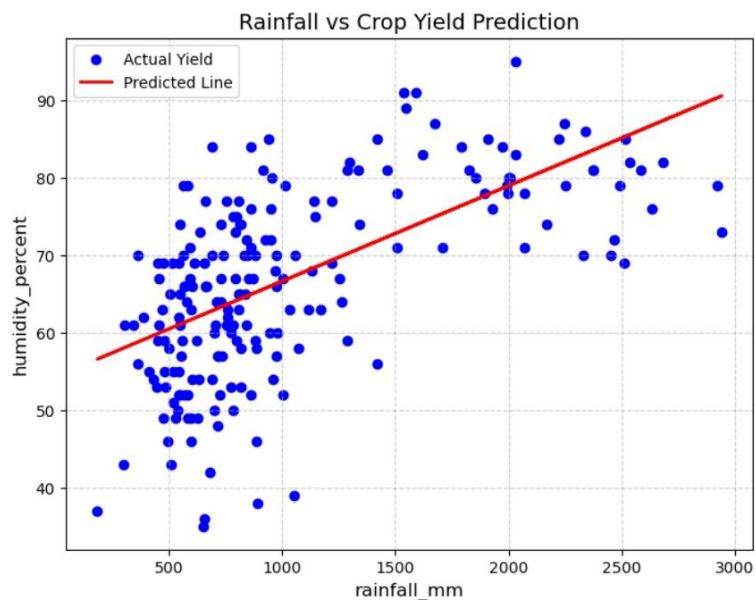
	temperature_c	rainfall_mm	humidity_percent	production_tonnes_per_hectare
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	25.490000	1027.579000	66.796000	10.402980
std	4.973928	619.037611	12.090019	24.885987
min	11.700000	121.000000	30.000000	0.500000
25%	21.800000	608.000000	59.000000	2.020000
50%	25.700000	798.000000	67.000000	2.780000
75%	29.200000	1270.000000	76.000000	4.325000
max	37.200000	3043.000000	95.000000	119.630000

Variation of data:

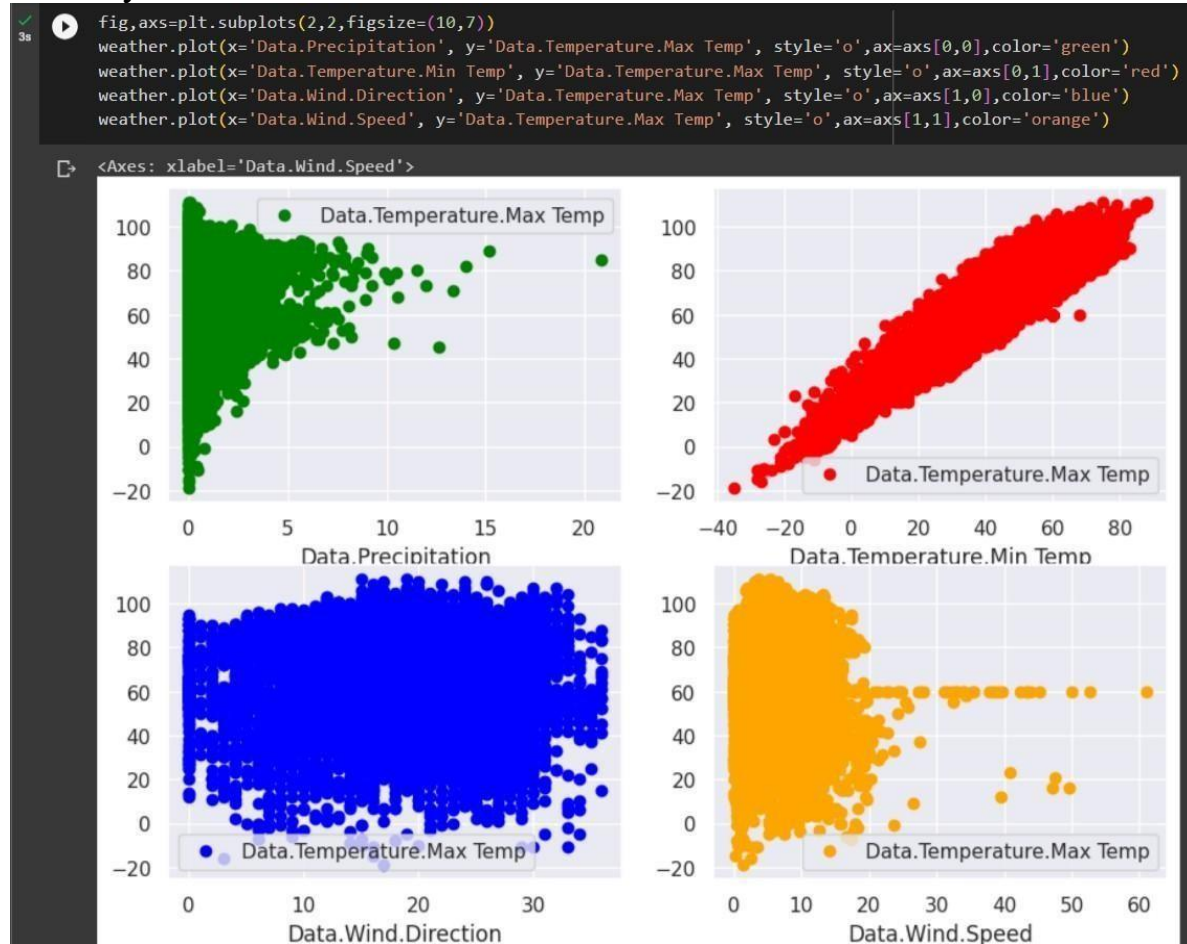
```
crop = LinearRegression()
crop.fit(X_train, y_train)
```

LinearRegression		
Parameters		
	fit_intercept	True
	copy_X	True
	tol	1e-06
	n_jobs	None
	positive	False


```
plt.figure(figsize=(8,6))
plt.scatter(X_test, y_test, color='blue', label='Actual Yield') # Actual data points
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted Line') # Regression Line
plt.title('Rainfall vs Crop Yield Prediction', fontsize=14)
plt.xlabel('rainfall_mm', fontsize=12)
plt.ylabel('humidity_percent', fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
y_pred = crop.predict(X_test)
```



Data analysis:



Data preprocessing:

```
0s x1,x2,x3,x4,y= weather['Data.Precipitation'],weather['Data.Temperature.Min Temp'],weather['Data.Wind.Direction'],weather['Data.Wind.Speed'], weather['Data.Temperature.Max Temp']
x1,x2,x3,x4,y=np.array(x1),np.array(x2),np.array(x3),np.array(x4),np.array(y)
x1,x2,x3,x4,y=x1.reshape(-1,1),x2.reshape(-1,1),x3.reshape(-1,1),x4.reshape(-1,1),y.reshape(-1,1)
X=np.concatenate((x1,x2,x3,x4),axis=1)
print(X)
```

```
[[ 0.  32.  33.  4.33]
 [ 0.  31.  32.  3.86]
 [ 0.16 41.  35.  9.73]
 ...
 [ 0.   4.  26.  1.65]
 [ 0.06 13.  24. 18.16]
 [ 0.1   8.  23.  7.51]]
```

Train Test Split:

```
0s X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
print("training data",X_train)
print("testing data",X_train)
print("training data",y_train)
print("testing data",y_train)
```

Training the model:

```
0s [34] regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

▼ LinearRegression
LinearRegression()

Intercept and Coefficient values:

```
0s [96] print("Intercept:",regressor.intercept_)

Intercept: [20.01204131]
```

```
0s print("Coefficients:",regressor.coef_)

Coefficients: [[-1.64831368  1.00162956  0.24264557 -0.52407733]]
```

Model Evaluation:

```
0s y_pred = regressor.predict(X_test)
    print(y_pred)
    df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
    df
```

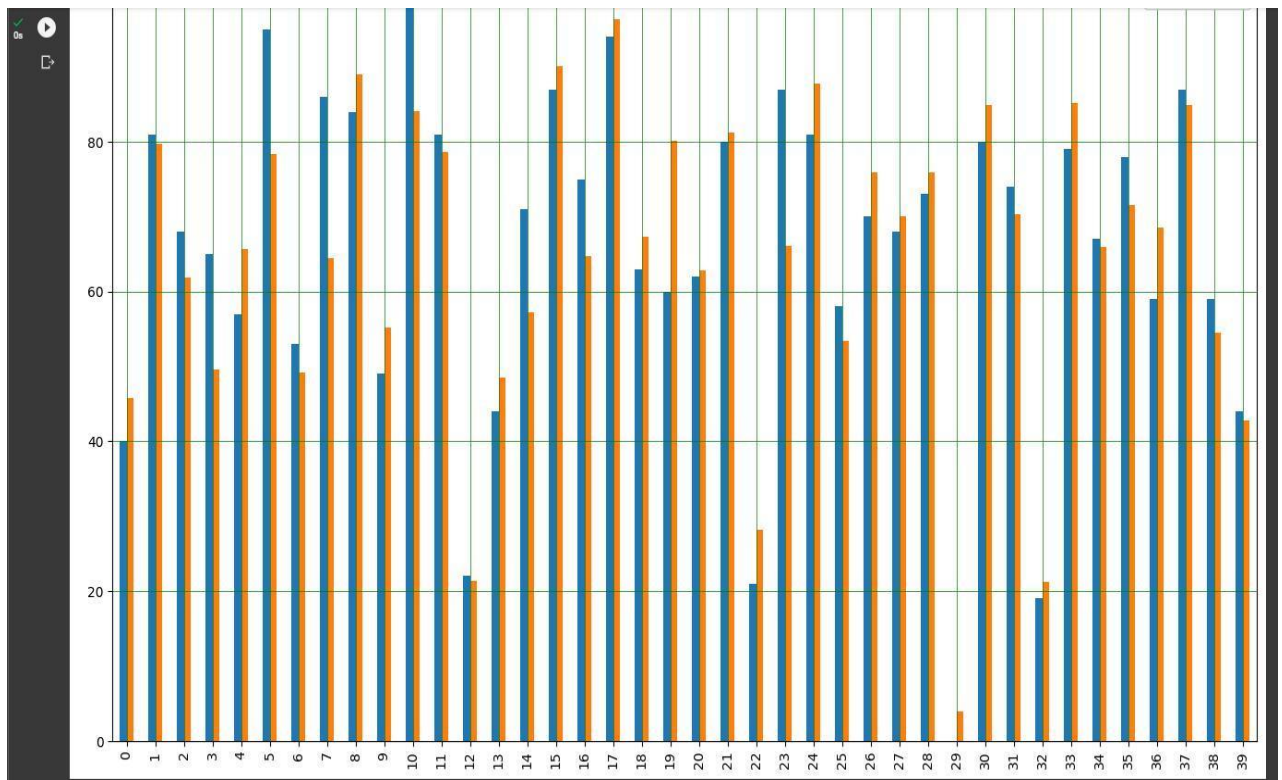
```
[[45.72768559]
 [79.77617955]
 [61.89882985]
 ...
 [68.24199195]
 [91.31446275]
 [56.67611227]]
```

	Actual	Predicted
0	40	45.727686
1	81	79.776180
2	68	61.898830
3	65	49.610616
4	57	65.676848
...
3344	68	57.550240
3345	84	75.552554
3346	70	68.241992
3347	86	91.314463
3348	58	56.676112

3349 rows x 2 columns

Plotting Actual value vs Predicted value:

```
df1 = df.head(30)
df1.plot(kind='bar', figsize=(12,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



Checking errors and accuracy:

```
[40] print("Mean Squared Error:",mean_squared_error(y_test,y_pred))
      print("Root Mean Squared Error:",np.sqrt(mean_squared_error(y_test,y_pred)))
      print('r_2 statistic:%.2f' % r2_score(y_test,y_pred) )
```

Mean Squared Error: 51.44210247954479
Root Mean Squared Error: 7.1723150014165435
r_2 statistic:0.87

Conclusion:

The project “Climate Impact on Crop Production” demonstrates how data analysis and machine learning—specifically linear regression—can be used to understand and predict the relationship between climate factors and crop yield. By analyzing variables such as rainfall, temperature, and humidity, the model helps identify patterns that influence agricultural productivity.

The results show that predictive modeling can assist farmers in making informed decisions regarding crop planning, irrigation, and resource management. This approach promotes sustainable farming by optimizing inputs and reducing risks associated with unpredictable climate changes. Overall, the project highlights the potential of data-driven technologies in improving agricultural efficiency and resilience.

Future Scope:

The future scope of this project lies in expanding its capabilities to include multiple climate parameters and more diverse crop varieties for improved prediction accuracy. Integration of real-time weather data and IoT-based sensors can enable live monitoring and forecasting of crop conditions. Advanced machine learning and deep learning techniques can be employed to handle complex, non-linear relationships between environmental factors and yield outcomes. Developing a mobile-friendly interface will make the system more accessible to farmers, allowing them to receive instant recommendations. Additionally, incorporating AI-based advisory systems for irrigation, fertilization, and pest control can transform this project into a comprehensive smart farming solution that supports sustainable and climate-resilient agriculture.

THANK YOU