A Major Project Report on

CROP YIELD PREDICTION USING MACHINE LEARNING

Submitted In partial fulfillment of the Requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

BY

ROHIT KUMAR	(17UJ1A0541)
VENNELA AISHWARAYA	(17UJ1A0525)
PONDHARA LOKNATH	(18UJ5A0509)
MAKKALA CHANDRAJA	(17UJ1A0510)
PALATA RAJASHEKAR REDDY	(17UJ1A0540)

Under the Esteemed Guidance of
Mrs. D.MAMATHA, M.Tech
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MALLA REDDY COLLEGE OF ENGINEERING AND MANAGEMENT SCIENCE
(Approved by AICTE New Delhi & Affiliated to JNTU Hyderabad)
Kistapur, Medchal, Medchal Dist- 501401.
2017-2021

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
HYDERABAD,KUKATPALLY, HYDERABAD-85.

MALLA REDDY ENGINEERING COLLEGE AND MANAGEMENT SCIENCES
(Approved by AICTE New Delhi & Affiliated to JNTU Hyderabad)
Kistapur, Medchal, Medchal Dist- 501401.

CERTIFICATE

This is to certify that the major project report entitled " **CROP YIELD PREDICTION USING MACHINE LEARNING**" being submitted by

ROHIT KUMAR	(**17UJ1A0541**)
VENNELA AISHWARYA	(**17UJ1A0525**)
PONDHARA LOKNATH	(**18UJ5A0509**)
MAKKALA CHANDRAJA	(**17UJ1A0510**)
PALATA RAJASHEKAR REDDY	(**17UJ1A0540**)

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Jawaharlal Nehru Technological University , Hyderabad, during the academic year 2017-2021 is a record of bonafied work carried out under my guidance and supervision.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Internal Guide	**Head of The Department**
Mrs. D.MAMATHA **M.Tech.**	**Dr. D . MAHAMMAD RAFI, M.Tech, Ph.D**
Assistant Professor	Professor & Head of CSE
Department.	Department of CSE

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First and foremost, I express my deep debt of gratitude to the Chairperson and Managing Trustee **Mr.V. Malla Reddy** for their immense contribution in making this organization grow and providing me the state of the art facilities to do this project work.

My heartfelt gratefulness to Director of MREM **Mr.L. Venugopal Reddy** for their deep commitment and dedication, to bring this institution to the peak in terms of discipline and values.

I owe my full satisfaction in my project work to the extensive hands of cooperation of Principal Academics **Dr.CNV.SRIDHAR , M.Tech,Ph.D.**

I would like to express my deep appreciation and sincere gratitude to my project internal guide , **Mrs,D.MAMATHA** Assistant Professor,Department of Computer Science & Engineering, Malla Reddy Engineering College and Management Sciences, for her guidance, support, encouragement, understanding and patience. I have been honored to work under her supervision and learn from her advice and useful insights throughout my project work.

I extend my gratitude to **Dr. D.Mahammad Rafi**, Professor and Head, Department of Computer Science and Engineering for his constant support to complete the project work.

I express my sincere thanks to all the teaching and non-teaching staffs of Malla Reddy Engineering College and Management Sciences who cooperated with me, which helped me to realize my dream.We would like to thank our internal project mates and department faculties for their full-fledged guidance and giving courage to carry out the project.I am very much thankful to one and all that helped me for the successful completion of my project.

ROHIT KUMAR	**(17UJ1A0541)**
VENNALA AISHWARYA	**(17UJ1A0525)**
PONDHARA LOKNATH	**(18UJ5A0509)**
MAKKALA CHANDRAJA	**(17UJ1A0510)**
PALATA RAJASHEKAR REDDY	**(17UJ1A0540)**

ABSTRACT

The classification and recognition of agricultural crop types is an important application of remote sensing. New machine learning algorithms have emerged in the last years, but so far, few studies only have compared their performance and usability. Therefore, we compared three different state-of-the-art machine learning classifiers, namely Support Vector Machine (SVM)) and Random Forest (RF) . For this purpose we classified a dataset of crop fields located a stratified randomized sampling approach. We compared the mean overall classification accuracies as well as standard deviations. Furthermore, the classification accuracy of single crops was analyses.

Support Vector Machine classifiers using radial basis function or polynomial kernels exhibited superior results to SVM and RF in terms of overall accuracy and robustness, Grassland exhibited the best results for early-season mono-temporal analysis. With a multi-temporal approach, the highest accuracies were achieved for Rapeseed and Field Peas. Other crops, such as Wheat, Flax and Lentils were also successfully classified. The user’s and producer’s accuracies were higher than 85 %.

The main objective of this proposal is to build a Machine Learning model that can accurately predict the rice crop yield prediction. Over 97% of the population in India depends on rice for food and is the second-highest in overall agriculture productions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

Anthropological climate change will affect the agricultural sector more directly than many others because of its direct dependence on weather (Porter *et al* 2013). The nature and magnitude of these impacts depends both on the evolution of the climate system, as well as the relationship between crop yields and weather. This paper focuses on the latter—yield prediction from weather. Accurate models mapping weather to crop yields are important not only for projecting impacts to agriculture, but also for projecting the impact of climate change on linked economic and environmental outcomes, and in turn for mitigation and adaptation policy.

A substantial portion of the work of modelling yield for the purpose of climate change impact assessment relies on deterministic, biophysical crop models (e.g. Rosenzweig *et al* 2013). These models are based on detailed representations of plant physiology and remain important, particularly for assessing response mechanisms and adaptation options (Ciscar *et al* 2018). However, they are generally outperformed by statistical models in prediction over larger spatial scales (Lo-bell and Burke 2010, Lo-bell and Ass eng 2017). In particular, a large literature following Schlenker and Roberts (2009) has used statistical models to demonstrate a strong linkage between extreme heat and poor crop performance. These approaches have relied on classical econometric methods. Recent work has sought to fuse crop models with statistical models, variously by including crop model output within statistical models (Roberts *et al* 2017), and by using insights from crop models in the parameterize of statistical models (Roberts *et al* 2012, Urban *et al* 2015).

In parallel, machine learning (ML) techniques have advanced considerably over the past several decades. ML is philosophically distinct from much of classical statistics, largely because its goals are different—it is largely focused on *prediction* of outcomes, as opposed to inference into the nature of the mechanistic processes generating those outcomes. (We focus on supervised ML—used for prediction—rather than *unsupervised* ML, which is used to discover structure in unlabelled data.)

We are using two different algorithm, Artificial Neural Network and Random Forest to find out the best applicable one. Also providing an option to predict the future production based on user input.

1.2 STATE-OF-THE-ART

Since the dawn of remote sensing, numerous studies on crop type classification have been published. Either optical or the combination of Radar and optical data were used as primary data sources.In recent studies (Yang et al., 2011; Dixon & Canada, 2008)the superiority of non-parametric machine-learning algorithms to parametric classifiers, such as nearest neighbour or ML, has been described. Classification ac-curacies of Decision Trees such as RF, Artificial Neural Networks and Support Vector Machine were found to be similar. Artificial-Neural-Network and SVM achieved similar results in a land-cover classification study on Landsat TM data carried out by Dixon & Canada (2008),whereas ML performed significantly worse.

In a comparison of Decision Tree (DT), ANN and ML, Pal & Mather (2003)reported non-significant differences in classification accuracy between the former two, whereas the manual work- and computational time effort turned out to be much more intensive for ANN. A land-cover classification with ANN, SVM, DT and ML published by Huang et al. (2002) resulted in higher ac curacies of ANN and SVM as compared to DT. Nonetheless DT performed much faster with a calculation time of minutes compared to hours and days respectively for SVM and ANN. Hence, the major disadvantage of the machine learning classifiers is that their computational complexity is higher compared to traditional supervised methods, such as ML or Nearest Neighbor, but they also differ strongly among each other. However, as to our knowledge so far no crop classification study compared the main machine-learning algorithms RF, ANN and SVM altogether.

1.3 METHODOLOGY

Accurate yield estimation is essential in agriculture. This Project develops machine learning based techniques for accurate crop yield prediction. The paper concludes that the rapid advances in sensing technologies and ML techniques will provide cost-effective and comprehensive solutions for better crop and environment state estimation and decision making. We are using two different algorithm, Artificial Neural Network and Random Forest to find out the best applicable one. Also providing an option to predict the future production based on user input.

CHAPTER-2

LITREATURE SURVEY

2. LIREATURE SURVEY

Grajales et al have proposed a web application that utilizes open dataset like historical production, land cover, local climate conditions and integrates them to provide easy access to the farmers. The proposed architecture mainly focuses on open source tools for the development of the application. The user can select location from map for which the details are available at one click

Bendre et al collects data from GIS (Global Information System), GPS (Global Positioning System), VRT (Variable Rate Fertilizer) and RS (Remote sensing) and are manipulated using Map Reduce algorithm and linear regression algorithm to forecast the weather data that can be used in precision agriculture. The purpose of this study was to investigate the effective model to improve the accuracy of rainfall forecasting.

Hemageetha mainly focuses on using the soil parameters like pH, Nitrogen, moisture etc for crop yield prediction. Naive Bayes algorithm is used to classify the soil and a77% accuracy is achieved. Appriori algorithm is used to assosciate the soil with the crops that could provide maximum yield in them. A comparison of accuracy achieved during classification using Naïve Bayes, J48 and JRIP is also presented.

S.No	Title of Paper	Authors	Year	Technology Used	Advantages	Disadvantages
1	Crop-Planning, Making Smarter Agriculture With Climate Data	Grajales	2015	A web application is developed using Open source tools.	The details of the selected location from map are available at one look to the user.	More focused on UI
2	Big Data in Precision Agriculture :Weather Forecasting for Future Farming	Bendre	2015	Map Reduce and Linear Regression algorithm are used for weather forecasting.	The effective model to improve the accuracy of rainfall forecasting is investigated	The forecasting is done based on only a weather data.
3	A survey on application of data mining techniques to analyze the soil for agricultural purpose	Hemage ethaa	2016	Naïve Bayes, Appriori algorithm are used for yield prediction.	Focuses mainly on various soil parameters like pH, Nitrogen, moisture etc and comparison accuracy is	Only 77% of accuracy is achieved.

Table 2.1 : Literature Survey

2.1 DISADVANTAGES OF EXISTING SYSTEM

RS based approaches require processing of enormous amounts of remotely sensed data from different platforms and, therefore, greater attention is currently being devoted to machine learning (ML) methods. This is due to the capability of machine learning based systems to process a large number of inputs and handle nonlinear tasks.

- Remote sensing is a fairly expensive menthod.
- Remote sensing requires a special kind of training to analyze the images. .
- It analyze different aspects of the photography features.
- The instruments used in uncallbrated which may lead to remote sensing data.
- It may look the same during measurement which lead to classification erroe.

2.2 EXISTING SYSTEM

Remote sensing (RS) systems are being more widely used in building decision support tools for contemporary farming systems to improve yield production and nitrogen management while reducing operating costs and environmental impact. for contemporary farming systems to improve yield production and nitrogen management while reducing operating costs and environmental impact.

However, RS based approaches require processing of enormous amounts of remotely sensed data from different platforms and, therefore, greater attention is currently being devoted to machine learning (ML) methods.This is due to the capability of machine learning based systems to process a large number of inputs and handle non-linear tasks.

CHAPTER-3

SYSTEM ANALYSIS

3.1 PROBLEM STATEMENT

The Objective of the project is to develop a Machine Learning based system which can predict yield of the crop in advance so that the farmer can take better decision in crop selection. This will also help our country to become independent on food resources.

Agriculture is a business with risk and realible crop yield prediction is vital for decision related to agriculture risk management. To overcome these above issues we need to develop a system which will able to find the fact can or results, patterns and insights. The farmer can predict which crop should sow so that he/she can get more benefit

3.2 PROPOSED SYSTEM

Proposed system provides a modern approach to the problem using machine learning techniques with better accuracy and outcome.In the proposed system, we use supervised learning to form a model Here We can Use two technologies in this project, i.e Support vector machine and random forest So by Comparing these two algorithm we have choose best one.

The proposed system is described in following stages such as dataset collection(agriculture.csv file has been taken from government website), preprocessing step, feature selections(state name,district name,season name,crop,year,area) and applying machine learning modules.In this after comparing both the algorithm Support vector machine and Random forest , Random forest is more suitable than Support vector machine,so here we use in this project to implement Random forest algorithm. Proposed System provides a modern approach to the problem using machine learning techniques with better accuracy and outcome.

Proposed System architecture diagram

In This architecture we have give the input in two ways and than it will go for pre-processing the data means here excel data convert in proper format which understand the algorithm and than it move for next step for regression it shows statistical method used in finance,investing and other disciplines that attempts to determine the strength and character of relation-ship between one dependent variable and a series of other variables,than it move for next step it predicted the yield and suggest to the farmer

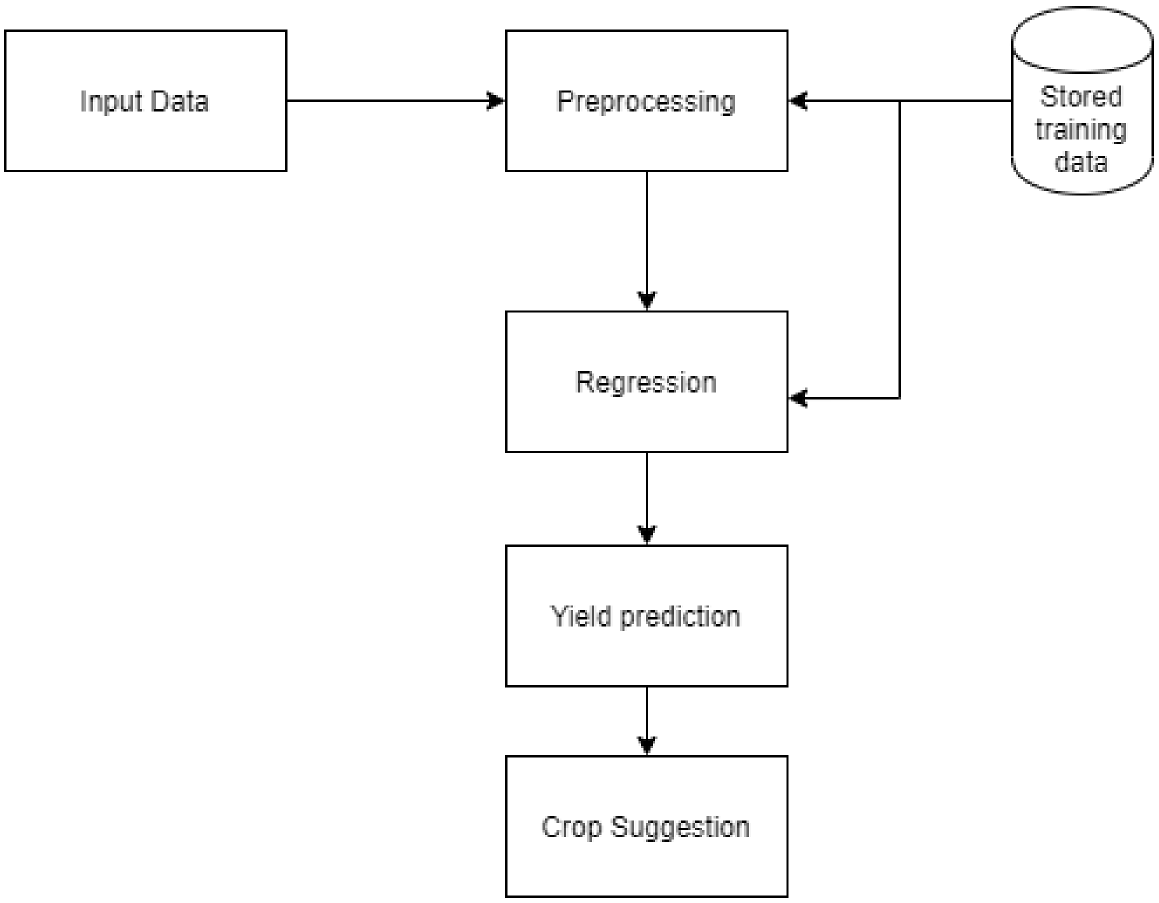


Figure 3.2: **Proposed System architecture**

3.3 PROPOSED SYSTEM ADVANTAGES

Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

No human intervention needed (automation)

With ML, you don’t need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

Continuous Improvement

As **ML algorithms** gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

Handling multi-dimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

Wide Applications

You could be an e-trailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

3.4 ALGORITHMS

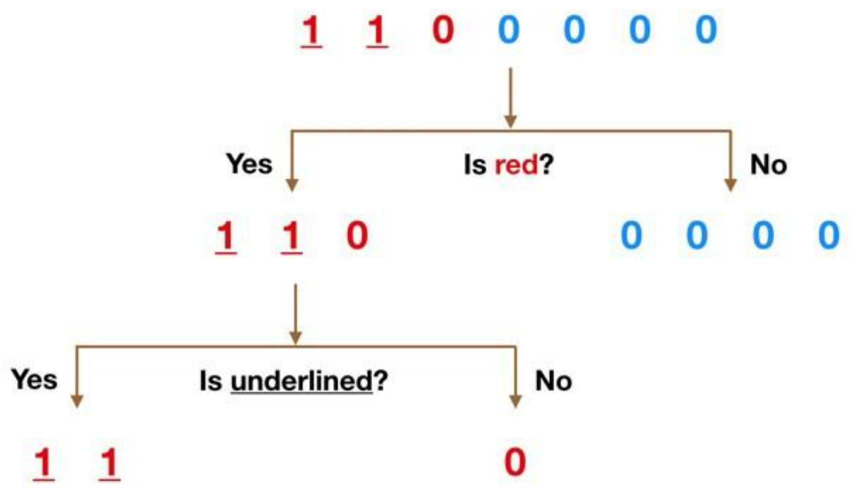
3.4.1 Random Forest Algorithm:

Data science provides a plethora of classification algorithms such as logistic regression, support vector machine, naive Bayes classifier, and decision trees. But near the top of the classifier hierarchy is the random forest classifier (there is also the random forest regressor but that is a topic for another day).

In this post, we will examine how basic decision trees work, how individual decisions trees are combined to make a random forest, and ultimately discover why random forests are so good at what they do.

Decision Trees

Let’s quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I’d be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.



Simple Decision Tree Example

It’s probably much easier to understand how a decision tree works through an example.Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features. The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this?

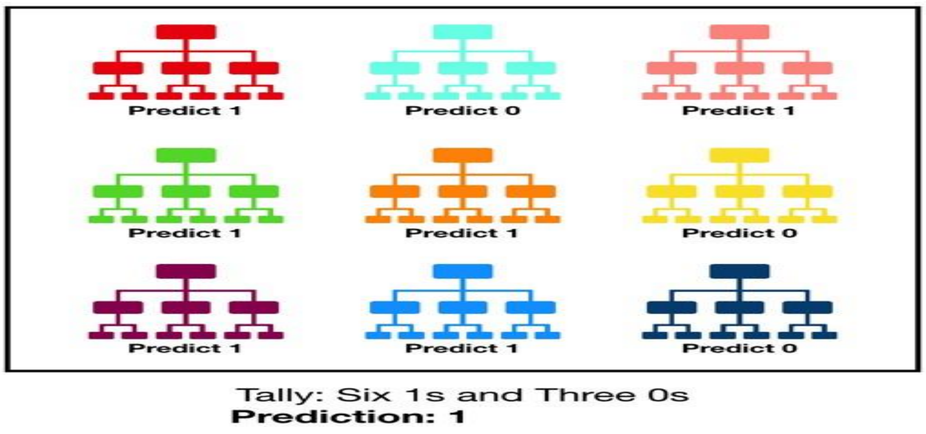
Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, “Is it red?” to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don’t go down the No branch.

The No branch (the blues) is all 0s now so we are done there, but our Yes branch can still be split further. Now we can use the second feature and ask, “Is it underlined?” to make a second split.The two 1s that are underlined go down the Yes subbranch and the 0 that is not underlined goes down the right subbranch and we are all done. Our decision tree was able to use the two features to split up the data perfectly. Victory!

Obviously in real life our data will not be this clean but the logic that a decision tree employs remains the same. At each node, it will ask What feature will allow me to split the observations at hand in a way that the resulting groups are as different from each other as possible (and the members of each resulting subgroup are as similar to each other as possible)?

The Random Forest Classifier

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below). Visualization of a Random Forest Model Making a Prediction



The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

- There needs to be some actual signal in our features so that models built using those features do better than random guessing.

- The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

An Example of Why Uncorrelated Outcomes are So Great

The wonderful effects of having many uncorrelated models is such a critical concept that I want to show you an example to help it really sink in. Imagine that we are playing the following game:

- I use a uniformly distributed random number generator to produce a number.
- If the number I generate is greater than or equal to 40, you win (so you have a 60% chance of victory) and I pay you some money. If it is below 40, I win and you pay me the same amount.
- Now I offer you the the following choices. We can either:

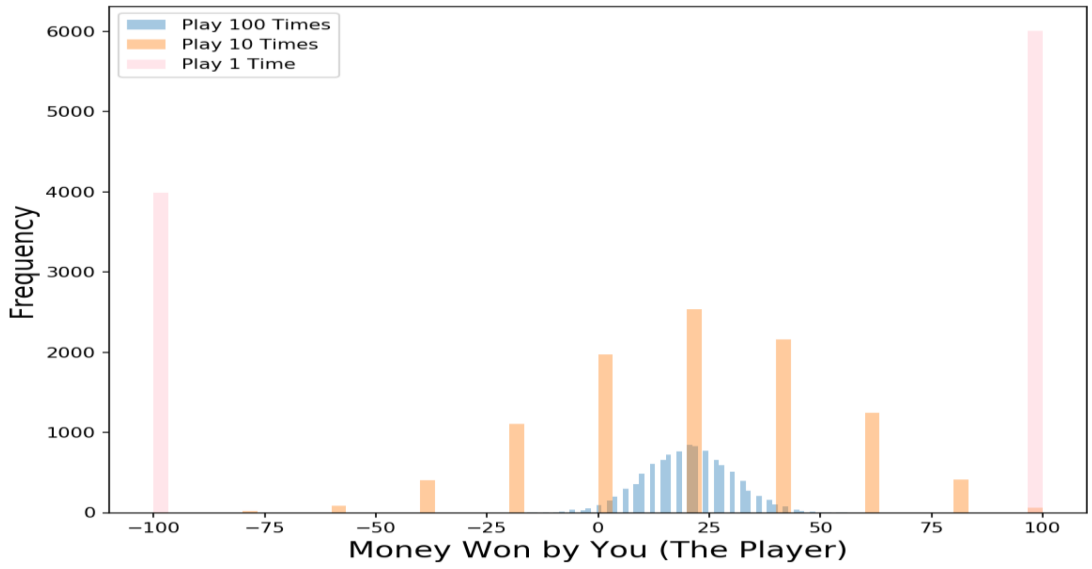
1. **Game 1** — play 100 times, betting $1 each time.
2. **Game 2**— play 10 times, betting $10 each time.
3. **Game 3**— play one time, betting $100.

Which would you pick? The expected value of each game is the same:

Expected Value Game 1 = (0.60*1 + 0.40*-1)*100 = 20

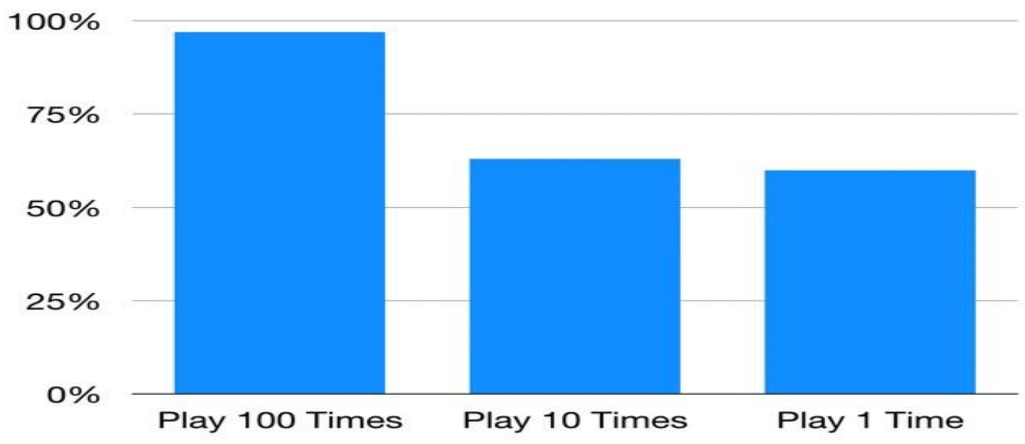
Expected Value Game 2= (0.60*10 + 0.40*-10)*10 = 20

Expected Value Game 3= 0.60*100 + 0.40*-100 = 20



Outcome Distribution of 10,000 Simulations for each Game

What about the distributions? Let's visualize the results with a Monte Carlo simulation (we will run 10,000 simulations of each game type; for example, we will simulate 10,000 times the 100 plays of Game 1). Take a look at the chart on the left — now which game would you pick? Even though the expected values are the same, the outcome distributions are vastly different going from positive and narrow (blue) to binary (pink).Game 1 (where we play 100 times) offers up the best chance of making some money — out of the 10,000 simulations that I ran, you make money in 97% of them! For Game 2 (where we play 10 times) you make money in 63% of the simulations, a drastic decline (and a drastic increase in your probability of losing money). And Game 3 that we only play once, you make money in 60% of the simulations, as expected.



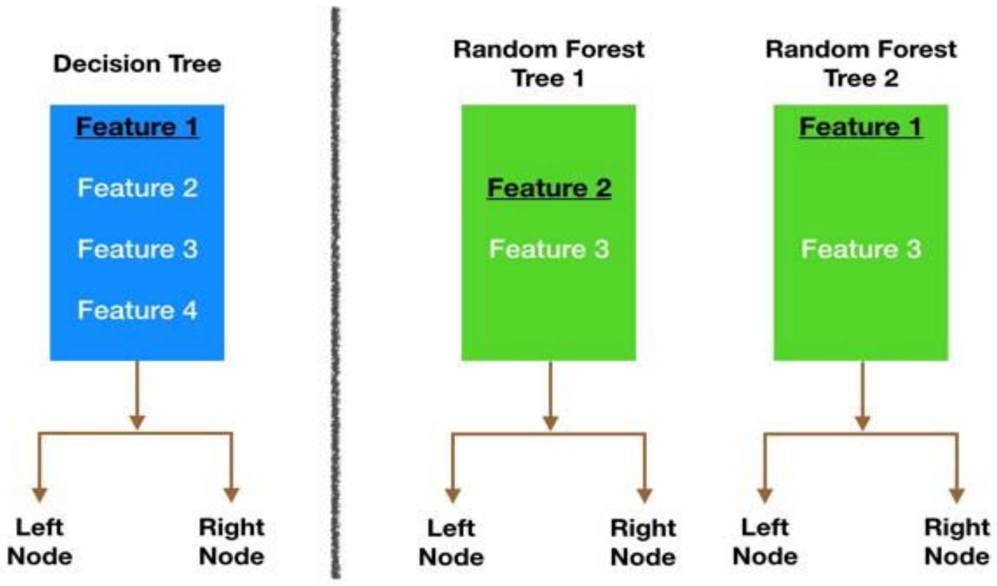
Probability of Making Money for Each Game

Random forest is the same — each tree is like one play in our game earlier. We just saw how our chances of making money increased the more times we played. Similarly, with a random forest model, our chances of making correct predictions increase with the number of uncorrelated trees in our model.If you would like to run the code for simulating the game yourself you can find it on my GitHub here.

Ensuring that the Models Diversify Each Other

So how does random forest ensure that the behavior of each individual tree is not too correlated with the behavior of any of the other trees in the model? It uses the following two methods:

Bagging (Bootstrap Aggregation) — Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.



Feature Randomness — In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.Let’s go through a visual example — in the picture above, the traditional decision tree (in blue) can select from all four features when deciding how to split the node. It decides to go with

3.4.2 SUPPORT VECTOR MACHINE ALGORITHM:

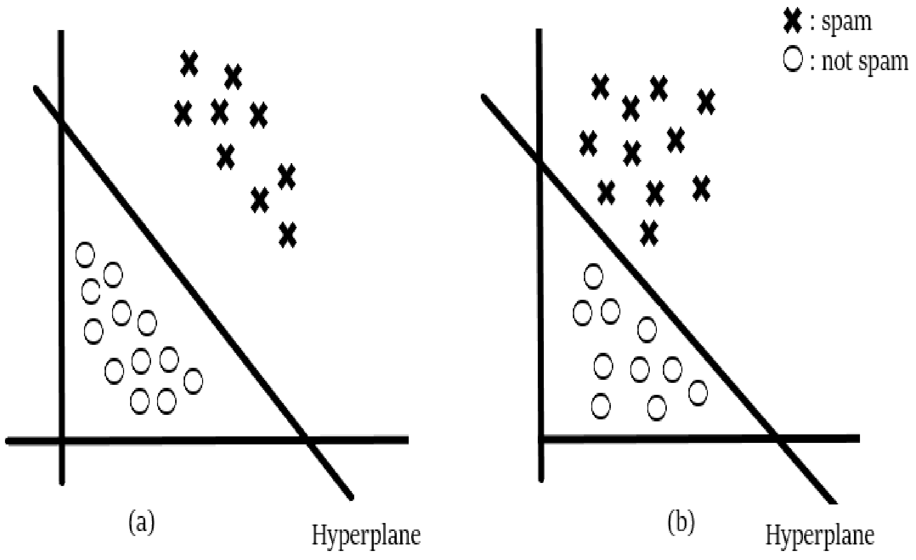
Support vector machines so called as SVM is a supervised learning algorithm which can be used for classification and regression problems as support vector classification (SVC) and support vector regression (SVR). It is used for smaller dataset as it takes too long to process. In this set, we will be focusing on SVC.

3.4.2.1 The ideology behind SVM:

SVM is based on the idea of finding a hyperplane that best separates the features into different domains.

3.4.2.2 Intuition development:

Consider a situation following situation: There is a stalker who is sending you emails and now you want to design a function(hyperplane) which will clearly differentiate the two cases, such that whenever you received an email from the stalker it will be classified as a spam. The following are the figure of two cases in which the hyperplane are drawn, which one will you pick and why? take a moment to analyze the situation …

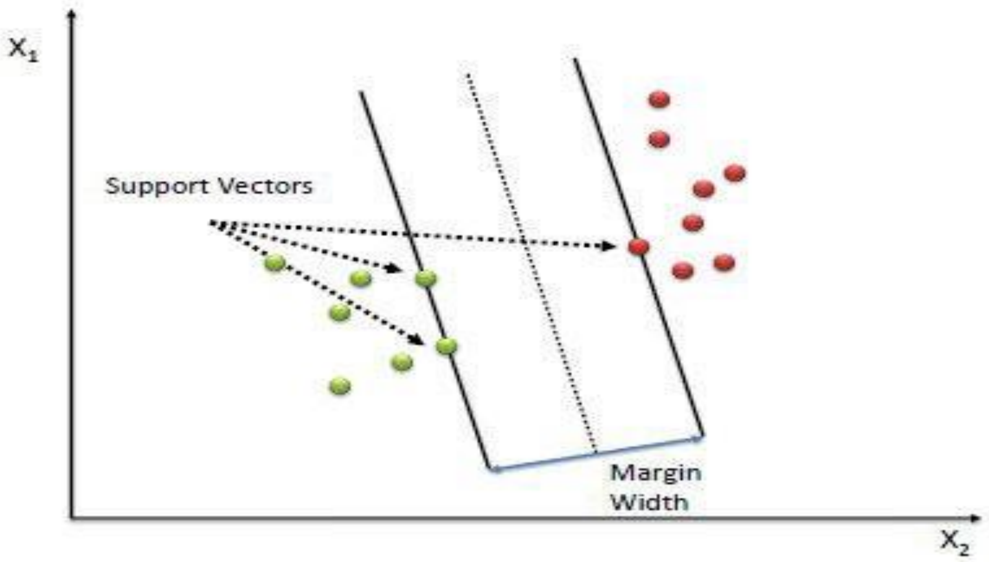


…

I guess you would have picked the fig(a). Did you think why have you picked the fig(a)? Because the emails in fig(a) are clearly classified and you are more confident about that as compared to fig(b). Basically, SVM is composed of the idea of coming up with an ***Optimal hyperplane*** which will clearly classify the different classes(in this case they are binary classes).

3.4.2.3. Terminologies used in SVM:

The points closest to the hyperplane are called as the ***support vector points*** and the distance of the vectors from the hyperplane are called the ***margins***.



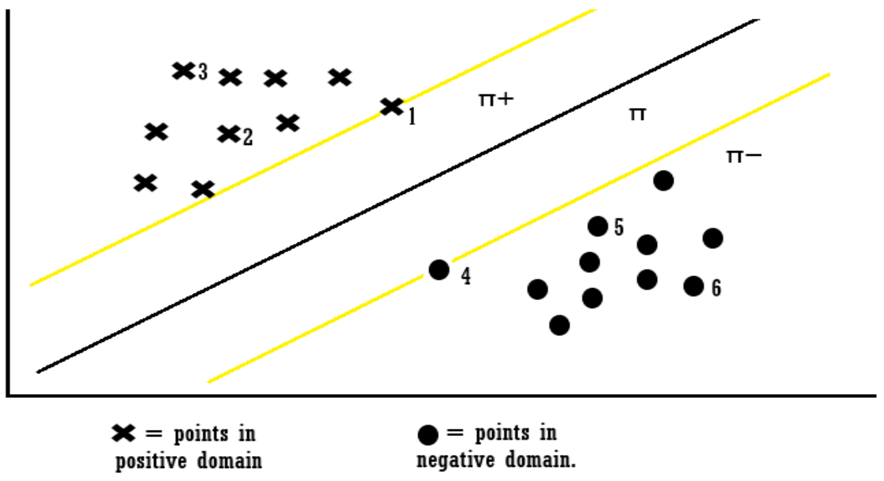
The basic intuition to develop over here is that more the farther SV points, from the hyperplane, more is the probability of correctly classifying the points in their respective region or classes. SV points are very critical in determining the hyperplane because if the position of the vectors changes the hyperplane's position is altered. Technically this hyperplane can also be called as **margin maximizing hyperplane**.

3.4.2.4. Hyperplane(Decision surface):

For so long in this post we have been discussing the hyperplane, let's justify its meaning before moving forward. The hyperplane is a function which is used to differentiate between features. In 2-D, the function used to classify between features is a line whereas, the function used to classify the features in a 3-D is called as a plane similarly the function which classifies the point in higher dimension is called as a hyperplane. Now since you know about the hyperplane lets move back to SVM.

3.4.2.5. Hard margin SVM:

Now,



Assume 3 hyperplanes namely (π, π+, π−) such that 'π+' is parallel to 'π' passing through the support vectors on the positive side and 'π−' is parallel to 'π' passing through the support vectors on the negative side.The equations of each hyperplane can be considered as:

$$\pi = b + w^T X = 0$$
$$\pi^+ = b + w^T X = 1$$
$$\pi^- = b + w^T X = -1$$

for the point X1 :

$$y_1 = 1$$
$$y_1(w^T x_1 + b) = 1$$

Explanation: when the point X1 we can say that point lies on the hyperplane and the equation determines that the product of our actual output and the hyperplane equation is 1 which means the point is correctly classified in the positive domain.

for the point X3:

$$y_1 = 1$$
$$y_1(w^T x_1 + b) > 1$$

Explanation: when the point X3 we can say that point lies away from the hyperplane and the equation determines that the product of our actual output and the hyperplane equation is greater 1 which means the point is correctly classified in the positive domain.

for the point X4:

$$y_1 = -1$$
$$y_1(w^T x_1 + b) = 1$$

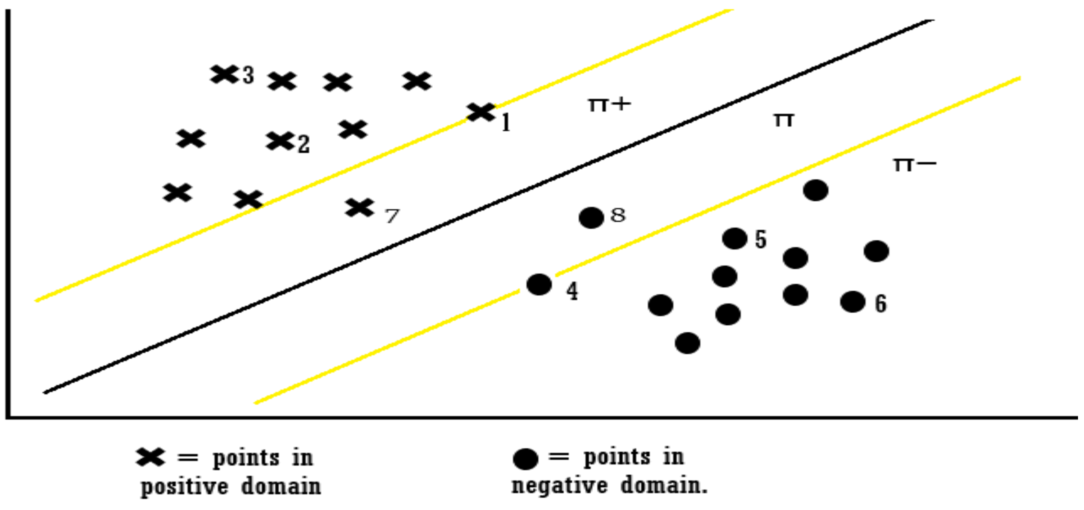
Explanation: when the point X4 we can say that point lies on the hyperplane in the negative region and the equation determines that the product of our actual output and the hyperplane equation is equal to 1 which means the point is correctly classified in the negative domain.

for the point X6 :

$$y_1 = -1$$
$$y_1(w^T x_1 + b) > 1$$

Explanation: when the point X6 we can say that point lies away from the hyperplane in the negative region and the equation determines that the product of our actual output and the hyperplane equation is greater 1 which means the point is correctly classified in the negative domain.

Let’s look into the constraints which are not classified:



$$y_1 = 1$$
$$y_1(w^T x_1 + b)$$
$$(1)(1 <)$$

for point X7:

Explanation: When Xi = 7 the point is classified incorrectly because for point 7 the wT + b will be smaller than one and this violates the constraints. So we found the misclassification because of constraint violation. Similarly, we can also say for points Xi = 8.

Thus from the above examples, we can conclude that for any point Xi,

if Yi(WT*Xi +b) ≥ 1:

then Xi is correctly classified

else:

Xi is incorrectly classified.

So we can see that if the points are linearly separable then only our hyperplane is able to distinguish between them and if any outlier is introduced then it is not able to separate them. So these type of SVM is called **as hard margin SVM** (since we have very strict constraints to correctly classify each and every datapoint).

3.4.2.6. Soft margin SVM:

We basically consider that the data is linearly separable and this might not be the case in real life scenario. We need an update so that our function may skip few outliers and be able to classify almost linearly separable points. For this reason, we introduce a new **Slack variable (** ξ) which is called Xi.

if we introduce ξ it into our previous equation we can rewrite it as

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

Introduction of Xi

if ξi= 0,

the points can be considered as correctly classified.

else:

ξi> 0 , Incorrectly classified points.

so if ξi> 0 it means that Xi(variables)lies in incorrect dimension, thus we can think of ξi as an error term associated with Xi(variable). The average error can be given as;

$$\frac{1}{n}\sum_{i=1}^{n}\xi_i$$

average error

thus our objective, mathematically can be described as;

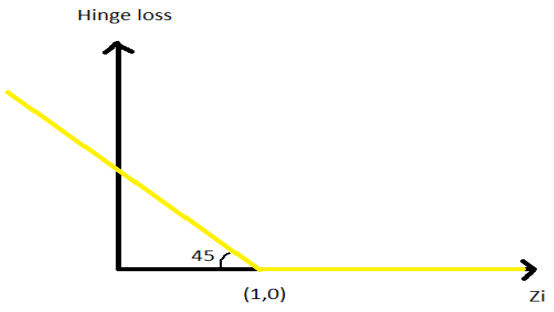
$$\underset{w,b}{minimize}\ \frac{1}{2}\,||w||^2+\sum_{i=1}^{n}\zeta_i$$

$$such\ that\ y_i\left(w^T.X_i+b\right)\geq 1-\zeta_i\quad For\ all\ i=1,2,.....,n$$

where ξi = ςi

READING: To find the vector w and the scalar b such that the hyperplane represented by w and b maximizes the margin distance and minimizes the loss term subjected to the condition that all points are correctly classified.This formulation is called the Soft margin technique.

3.4.2.7. Loss Function Interpretation of SVM:



$$Z_i = y_i(w^T x_i + b)$$
$$Z_i \geq 1$$

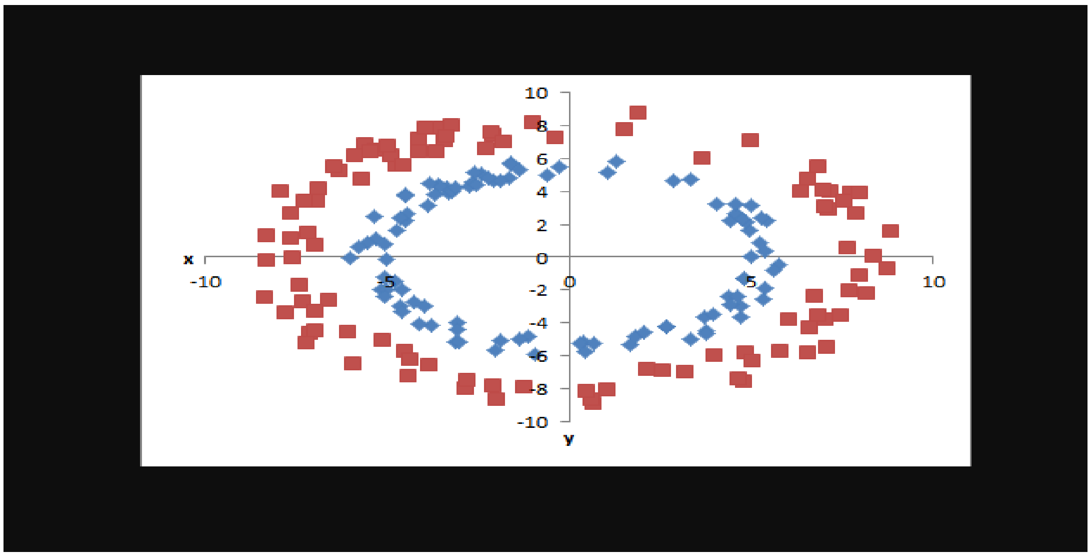
when Zi is ≥ 1 then the loss is 0

$$Z_i < 1$$

when Zi < 1 then loss increases.thus it can be interpreted that hinge loss is max(0,1-Zi).

3.4.2.8. Dual form of SVM:

Now, let’s consider the case when our data set is not at all linearly separable.



basically, we can separate each data point by projecting it into the higher dimension by adding relevant features to it as we do in logistic regression. But with SVM there is a powerful way to achieve this task of projecting the data into a higher dimension. The above-discussed formulation was the **primal form of SVM** . The alternative method is dual form of SVM which uses **Lagrange's multiplier** to solve the constraints optimization problem.

$$\underset{\alpha}{maximize} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(X_i^T . X_j \right)$$

$$subject\ to\ \alpha_i \geq 0\ \ For\, all\, i = 1, 2, \cdots, n\ and \sum_{i=1}^{n} \alpha_i y_i = 0$$

Note:

If αi>0 then Xi is a Support vector and when αi=0 then Xi is not a support vector.

Observation:

1. To solve the actual problem we do not require the actual data point instead only the dot product between every pair of a vector may suffice.
2. To calculate the "b" biased constant we only require dot product.
3. The major advantage of dual form of SVM over Lagrange formulation is that it only depends on the **α**.

3.4.2.9. What is Kernel trick?

Coming to the major part of the SVM for which it is most famous, the **kernel trick**. The kernel is a way of computing the dot product of two vectors **x** and **y** in some (very high dimensional) feature space, which is why kernel functions are sometimes called "generalized dot product.

$$\underset{\alpha}{maximize} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j\, K \left(X_i^T . X_j \right)$$

try reading this equation…

$$s.t \quad 0 \leq \alpha_i \leq C \quad For\ all\ i = 1, 2, \ldots.., n \quad and \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

s.t = subjected to

Applying kernel trick means just to the replace dot product of two vectors by the kernel function.

11. Types of kernels:

1. linear kernel
2. polynomial kernel
3. Radial basis function kernel (RBF)/ Gaussian Kernel

We will be focusing on the polynomial and Gaussian kernel since its most commonly used.

Polynomial kernel:

In general, the polynomial kernel is defined as ;

$$K(X_1, X_2) = (a + X_1^T X_2)^b$$

b = degree of kernel & a = constant term.

in the polynomial kernel, we simply calculate the dot product by increasing the power of the kernel.

Example:

using kernel trick:

$$Z_a^T Z_b = k(X_a, X_b) = (1 + X_a^T X_b)^2$$
$$Z_a^T Z_b = 1 + a_1 b_1 + a_2 b_2 + a_1^2 b_1^2 + a_2^2 b_2^2 + a_1 b_1 a_2 b_2$$

In this method, we can simply calculate the dot product by increasing the value of power. Simple isn’t it?

Radial basis function kernel (RBF)/ Gaussian Kernel:

Gaussian RBF(Radial Basis Function) is another popular Kernel method used in SVM models for more. RBF kernel is a function whose value depends on the distance from the origin or from some point. Gaussian Kernel is of the following format;

$$K(X_1, X_2) = exponent(-\gamma\|X_1 - X_2\|^2)$$

||X1 — X2 || = Euclidean distance between X1 & X2

Using the distance in the original space we calculate the dot product (similarity) of X1 & X2.

Note: similarity is the angular distance between two points.

Parameters:

1. C: Inverse of the strength of regularization.

Behavior: As the value of ‘c’ increases the model gets overfits.

As the value of ‘c’ decreases the model underfits.

3.5 SOFTWARE MODULES

3.5.1 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 with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric,[4] also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called Numarray was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.

There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then.In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.[7] This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.

In 2011, Py started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with 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. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[16] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas

NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as

universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Traits

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. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[16] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are

very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

The ndarray data structure

The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure. These arrays are strided views on memory.[7] In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.[7]

Limitations

Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave, numexpr and Numba. Cython and Pythran are static-compiling alternatives to these.

The Basics

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called *axes*.

For example, the coordinates of a point in 3D space [1, 2, 1] has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

```
[[ 1., 0., 0.], [ 0., 1., 2.]]
```

NumPy's array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

ndarray.shape

The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n*rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

ndarray.size

The total number of elements of the array. This is equal to the product of the elements of shape.

ndarray.dtype

An object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

ndarray.itemsize

The size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

ndarray.data

The buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

An example

```
>>>
```

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)

>>> a

array([[ 0,  1,  2,  3,  4],

[ 5,  6,  7,  8,  9],

[10, 11, 12, 13, 14]])

>>> a.shape

(3, 5)

>>> a.ndim

2

>>> a.dtype.name

'int64'

>>> a.itemsize

8

>>> a.size

15

>>> type(a)

<type 'numpy.ndarray'>

>>> b = np.array([6, 7, 8])

>>> b

array([6, 7, 8])

>>> type(b)

<type 'numpy.ndarray'>
```

Array Creation

There are several ways to create arrays.

For example, you can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.

```
>>>
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

A frequent error consists in calling array with multiple numeric arguments, rather than providing a single list of numbers as an argument.

```
>>>
>>> a = np.array(1,2,3,4)    # WRONG
>>> a = np.array([1,2,3,4])  # RIGHT
```

3.5.2 PANDAS:

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.**Pandas** is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real**

world data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- ➢ Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- ➢ Ordered and unordered (not necessarily fixed-frequency) time series data.
- ➢ Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- ➢ Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, **DataFrame** provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.Here are just a few of the things that pandas does well:

- ➢ Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating point data

- Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional objects
- Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations
- Powerful, flexible **group by** functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets
- Intuitive **merging** and **joining** data sets
- Flexible **reshaping** and pivoting of data sets
- **Hierarchical** labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from **flat files** (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast **HDF5 format**
- **Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists,

working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.Some other notes

- pandas is **fast**. Many of the low-level algorithmic bits have been extensively tweaked in Cythoncode. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.
- pandas is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python.
- pandas has been used extensively in production in financial applications.

3.5.2.1 DATA STRUCTURE

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

Why more than one data structure?

The best way to think about the pandas data structures is as flexible containers for lower dimensional data. For example, DataFrame is a container for Series, and Series is a container for scalars. We would like to be able to insert and remove objects from these containers in a dictionary-like fashion.

Also, we would like sensible default behaviors for the common API functions which take into account the typical orientation of time series and cross-sectional data sets. When using ndarrays to store 2- and 3-dimensional data, a burden is placed on the user to consider the orientation of the data set when writing functions; axes are considered more

or less equivalent (except when C- or Fortran-contiguousness matters for performance). In pandas, the axes are intended to lend more semantic meaning to the data; i.e., for a particular data set there is likely to be a “right” way to orient the data. The goal, then, is to reduce the amount of mental effort required to code up data transformations in downstream functions.

For example, with tabular data (DataFrame) it is more semantically helpful to think of the **index** (the rows) and the **columns** rather than axis 0 and axis 1. Iterating through the columns of the DataFrame thus results in more readable code:

```
for col in df.columns:

    series = df[col]

    # do something with series
```

3.5.2.2 MUTABILITY AND COPYING OF DATA

All pandas data structures are value-mutable (the values they contain can be altered) but not always size-mutable. The length of a Series cannot be changed, but, for example, columns can be inserted into a DataFrame. However, the vast majority of methods produce new objects and leave the input data untouched. In general we like to **favor immutability** where sensible.

3.5.2.3 INTRO TO DATA STRUCTURES

We'll start with a quick, non-comprehensive overview of the fundamental data structures in pandas to get you started. The fundamental behavior about data types, indexing, and axis labeling / alignment apply across all of the objects. To get started, import NumPy and load pandas into your namespace:

```
In [1]: import numpy as np



In [2]: import pandas as pd
```

Here is a basic tenet to keep in mind: **data alignment is intrinsic**. The link between labels and data will not be broken unless done so explicitly by you.We'll give a brief intro to the data structures, then consider all of the broad categories of functionality and methods in separate sections.

3.5.2.4 SERIES

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the **index**. The basic method to create a Series is to call:

```
>>> s = pd.Series(data, index=index)
```

Here, data can be many different things:

- a Python dict
- an ndarray
- a scalar value (like 5)

The passed **index** is a list of axis labels. Thus, this separates into a few cases depending on what **data is**:

From ndarray

If data is an ndarray, **index** must be the same length as **data**. If no index is passed, one will be created having values [0, ..., len(data) - 1].

```
In [3]: s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])

In [4]: s

Out[4]:

a    0.469112

b   -0.282863

c   -1.509059

d   -1.135632

e    1.212112

dtype: float64
```

From dict

Series can be instantiated from dicts:

```
In [7]: d = {'b': 1, 'a': 0, 'c': 2}

In [8]: pd.Series(d)

Out[8]:

b    1

a    0

c    2

dtype: int64
```

Note

When the data is a dict, and an index is not passed, the Series index will be ordered by the dict’s insertion order, if you’re using Python version >= 3.6 and Pandas version >= 0.23.If you’re using Python < 3.6 or Pandas < 0.23, and an index is not passed, the Series index will be the lexically ordered list of dict keys.

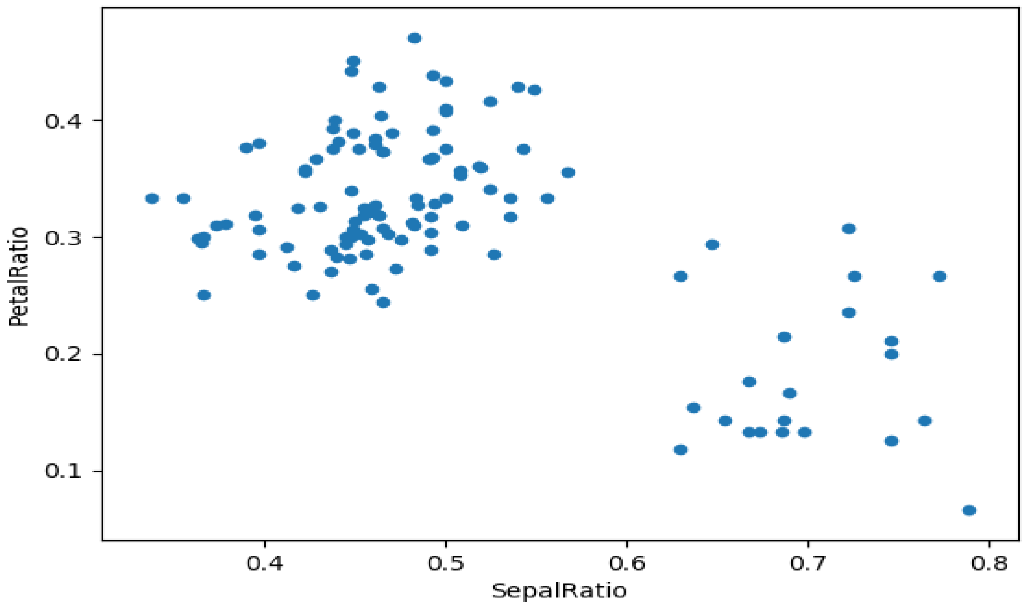
Assigning New Columns in Method Chains

Inspired by dplyr’s mutate verb, DataFrame has an **assign()** method that allows you to easily create new columns that are potentially derived from existing columns.In the example above, we inserted a precomputed value. We can also pass in a function of one argument to be evaluated on the DataFrame being assigned to.

```
In [77]: iris.assign(sepal_ratio=lambda x: (x['SepalWidth'] / x['SepalLength'])).head()

Out[77]:

   SepalLength  SepalWidth  PetalLength  PetalWidth        Name  sepal_ratio

0        5.1        3.5        1.4        0.2  Iris-setosa    0.686275

1        4.9        3.0        1.4        0.2  Iris-setosa    0.612245

2        4.7        3.2        1.3        0.2  Iris-setosa    0.680851

3        4.6        3.1        1.5        0.2  Iris-setosa    0.673913

4        5.0        3.6        1.4        0.2  Iris-setosa    0.720000
```

Assign **always** returns a copy of the data, leaving the original DataFrame untouched.Passing a callable, as opposed to an actual value to be inserted, is useful when you don't have a reference to the DataFrame at hand. This is common when using assign in a chain of operations. For example, we can limit the DataFrame to just those observations with a Sepal Length greater than 5, calculate the ratio, and plot:

```
In [78]: (iris.query('SepalLength > 5')

   ....:      .assign(SepalRatio=lambda x: x.SepalWidth / x.SepalLength,

   ....:              PetalRatio=lambda x: x.PetalWidth / x.PetalLength)

   ....:      .plot(kind='scatter', x='SepalRatio', y='PetalRatio'))

   ....:

Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2b527b1a58>
```

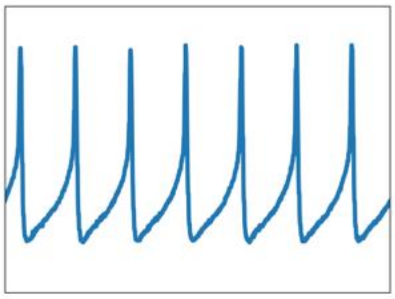


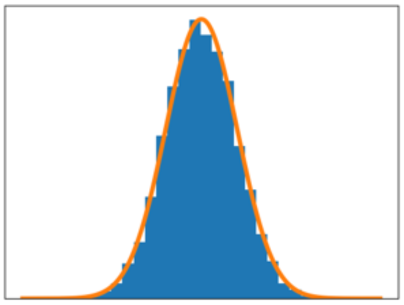
Since a function is passed in, the function is computed on the DataFrame being assigned to. Importantly, this is the DataFrame that's been filtered to those rows with sepal length greater than 5. The filtering happens first, and then the ratio calculations. This is an example where we didn't have a reference to the *filtered* DataFrame available.

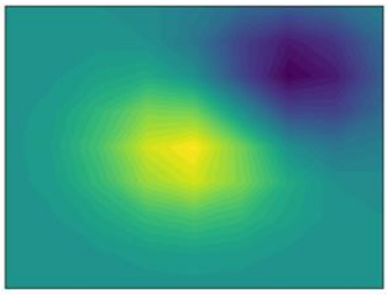
The function signature for assign is simply **kwargs. The keys are the column names for the new fields, and the values are either a value to be inserted (for example, Series or NumPy array), or a function of one argument to be called on the DataFrame. A *copy* of the original DataFrame is returned, with the new values inserted.

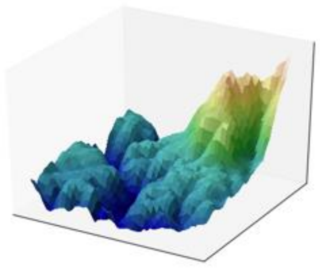
3.5.3 MATPLOTLIB

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.









Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font

properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Installing an official release

Matplotlib and its dependencies are available as wheel packages for macOS, Windows and Linux distributions:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

Although not required, we suggest also installing IPython for interactive use. To easily install a complete Scientific Python stack, see Scientific Python Distributions below.

macOS

To use the native OSX backend you will need a framework build build of Python.

Test data

The wheels (*.whl) on the PyPI download page do not contain test data or example code.If you want to try the many demos that come in the Matplotlib source distribution, download the *.tar.gz file and look in the examples subdirectory.

To run the test suite:

- extract the lib/matplotlib/tests or lib/mpl_toolkits/tests directories from the source distribution;
- install test dependencies: pytest, Pillow, MiKTeX, GhostScript, ffmpeg, avconv, ImageMagick, and Inkscape;
- run python -mpytest.

Third-party distributions of Matplotlib

Scientific Python Distributions

Anaconda and Canopy and ActiveState are excellent choices that "just work" out of the box for Windows, macOS and common Linux platforms. WinPython is an option for Windows users. All of these distributions include Matplotlib and *lots* of other useful (data) science tools.

Linux: using your package manager

If you are on Linux, you might prefer to use your package manager. Matplotlib is packaged for almost every major Linux distribution.

- Debian / Ubuntu: sudo apt-get install python3-matplotlib
- Fedora: sudo dnf install python3-matplotlib
- Red Hat: sudo yum install python3-matplotlib
- Arch: sudo pacman -S python-matplotlib

Installing from source

If you are interested in contributing to Matplotlib development, running the latest source code, or just like to build everything yourself, it is not difficult to build Matplotlib from source. Grab the latest *tar.gz* release file from the PyPI files page, or if you want to develop Matplotlib or just need the latest bugfixed version, grab the latest git version Install from source.

The standard environment variables CC, CXX, PKG_CONFIG are respected. This means you can set them if your toolchain is prefixed. This may be used for cross compiling.

```
export CC=x86_64-pc-linux-gnu-gcc
export CXX=x86_64-pc-linux-gnu-g++
export PKG_CONFIG=x86_64-pc-linux-gnu-pkg-config
```

Once you have satisfied the requirements detailed below (mainly Python, NumPy, libpng and FreeType), you can build Matplotlib.

```
cd matplotlib
python -mpip install .
```

We provide a setup.cfg file which you can use to customize the build process. For example, which default backend to use, whether some of the optional libraries that Matplotlib ships with are installed, and so on. This file will be particularly useful to those packaging Matplotlib.

If you have installed prerequisites to nonstandard places and need to inform Matplotlib where they are, edit setupext.py and add the base dirs to the basedir dictionary entry for your sys.platform; e.g., if the header of some required library is in /some/path/include/someheader.h, put /some/path in the basedir list for your platform.

Dependencies

Matplotlib requires the following dependencies:

- Python (>= 3.5)
- FreeType (>= 2.3)
- libpng (>= 1.2)
- NumPy (>= 1.10.0)
- setuptools
- cycler (>= 0.10.0)
- dateutil (>= 2.1)
- kiwisolver (>= 1.0.0)
- pyparsing

Optionally, you can also install a number of packages to enable better user interface toolkits. See What is a backend? for more details on the optional Matplotlib backends and the capabilities they provide.

- tk (>= 8.3, != 8.6.0 or 8.6.1): for the Tk-based backends;
- PyQt4 (>= 4.6) or PySide (>= 1.0.3): for the Qt4-based backends;
- PyQt5: for the Qt5-based backends;
- PyGObject or pgi (>= 0.0.11.2): for the GTK3-based backends;
- wxpython (>= 4): for the WX-based backends;
- cairocffi (>= 0.8) or pycairo: for the cairo-based backends;
- Tornado: for the WebAgg backend;

For better support of animation output format and image file formats, LaTeX, etc., you can install the following:

- ffmpeg/avconv: for saving movies;
- ImageMagick: for saving animated gifs;
- Pillow (>= 3.4): for a larger selection of image file formats: JPEG, BMP, and TIFF image files;

- LaTeX and GhostScript (>=9.0) : for rendering text with LaTeX.

Building on Linux

It is easiest to use your system package manager to install the dependencies.If you are on Debian/Ubuntu, you can get all the dependencies required to build Matplotlib with:

```
sudo apt-get build-dep python-matplotlib
```

If you are on Fedora, you can get all the dependencies required to build Matplotlib with:

```
sudo dnf builddep python-matplotlib
```

If you are on RedHat, you can get all the dependencies required to build Matplotlib by first installing yum-builddep and then running:

```
su -c "yum-builddep python-matplotlib"
```

These commands do not build Matplotlib, but instead get and install the build dependencies, which will make building from source easier.

Building on macOS

The build situation on macOS is complicated by the various places one can get the libpng and FreeType requirements (MacPorts, Fink, /usr/X11R6), the different architectures (e.g., x86, ppc, universal), and the different macOS versions (e.g., 10.4 and 10.5). We recommend that you build the way we do for the macOS release: get the source from the tarball or the git repository and install the required dependencies through a third-party package manager. Two widely used package managers are Homebrew, and MacPorts. The following example illustrates how to install libpng and FreeType using brew:

```
brew install libpng freetype pkg-config
```

If you are using MacPorts, execute the following instead:

```
port install libpng freetype pkgconfig
```

After installing the above requirements, install Matplotlib from source by executing:

```
python -mpip install .
```

Note that your environment is somewhat important. Some conda users have found that, to run the tests, their PYTHONPATH must include /path/to/anaconda/.../site-packages and their DYLD_FALLBACK_LIBRARY_PATH must include /path/to/anaconda/lib.

Building on Windows

The Python shipped from https://www.python.org is compiled with Visual Studio 2015 for 3.5+. Python extensions should be compiled with the same compiler, see e.g. https://packaging.python.org/guides/packaging-binary-extensions/#setting-up-a-build-environment-on-windows for how to set up a build environment.

Since there is no canonical Windows package manager, the methods for building FreeType, zlib, and libpng from source code are documented as a build script at matplotlib-winbuild.

There are a few possibilities to build Matplotlib on Windows:

- Wheels via matplotlib-winbuild
- Wheels by using conda packages (see below)
- Conda packages (see below)

Wheel builds using conda packages

This is a wheel build, but we use conda packages to get all the requirements. The binary requirements (png, FreeType,...) are statically linked and therefore not needed during the wheel install.Set up the conda environment. Note, if you want a qt backend, add pyqt to the list of conda packages.

```
conda create -n "matplotlib_build" python=3.7 numpy python-dateutil pyparsing tornado cycler tk libpng zlib freetype msinttypes
conda activate matplotlib_build
```

For building, call the script build_alllocal.cmd in the root folder of the repository:

```
build_alllocal.cmd
```

General Concepts

matplotlib has an extensive codebase that can be daunting to many new users. However, most of matplotlib can be understood with a fairly simple conceptual framework and knowledge of a few important points.Plotting requires action on a range of levels, from the most general (e.g., 'contour this 2-D array') to the most specific (e.g., 'color this screen pixel red'). The purpose of a plotting package is to assist you in visualizing your data as easily as possible, with all the necessary control -- that is, by

using relatively high-level commands most of the time, and still have the ability to use the low-level commands when needed.

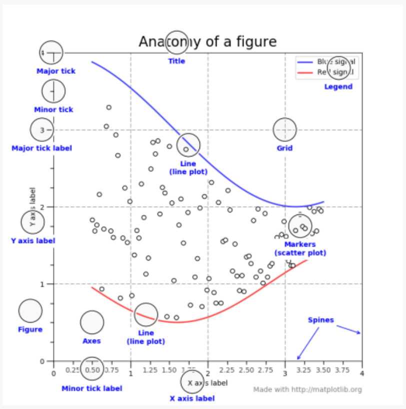
Therefore, everything in matplotlib is organized in a hierarchy. At the top of the hierarchy is the matplotlib "state-machine environment" which is provided by the matplotlib.pyplot module. At this level, simple functions are used to add plot elements (lines, images, text, etc.) to the current axes in the current figure.

Note

Pyplot's state-machine environment behaves similarly to MATLAB and should be most familiar to users with MATLAB experience.

The next level down in the hierarchy is the first level of the object-oriented interface, in which pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.

For even more control -- which is essential for things like embedding matplotlib plots in GUI applications -- the pyplot level may be dropped completely, leaving a purely object-oriented approac



```
import matplotlib.pyplot as plt
import numpy as np
```

3.5.4 SEABORN:

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

An introduction to seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

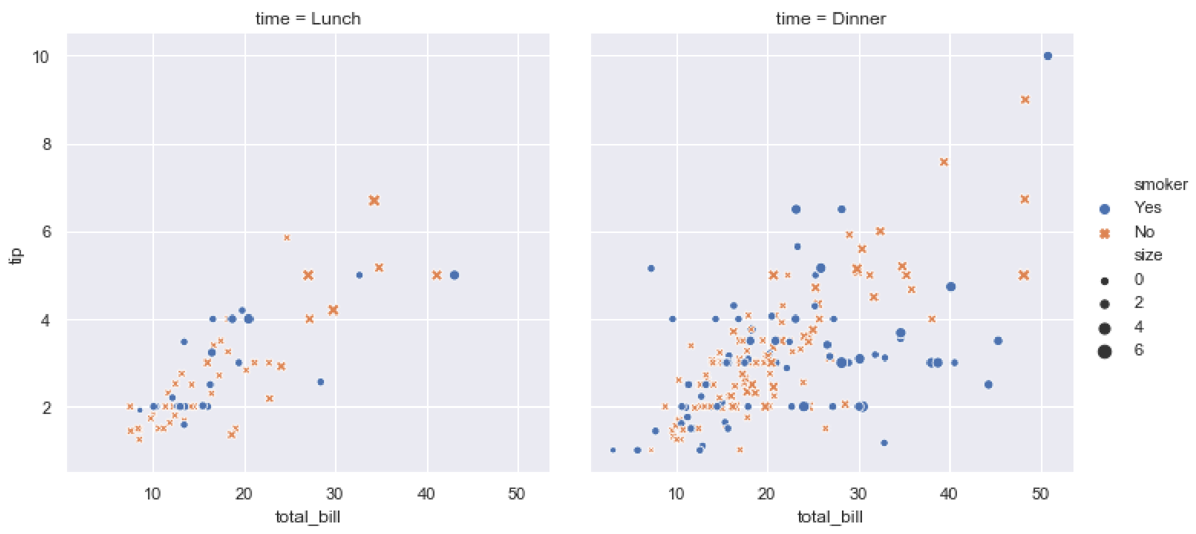
Here is some of the functionality that seaborn offers:A dataset-oriented API for examining relationships between multiple variables Specialized support for using categorical variables to show observations or aggregate statistics Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data

- Automatic estimation and plotting of linear regression models for different kinds dependent variables
- Convenient views onto the overall structure of complex datasets
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Here’s an example of what this means:

```
import seaborn as sns

sns.set()

tips = sns.load_dataset("tips")

sns.relplot(x="total_bill", y="tip", col="time",

            hue="smoker", style="smoker", size="size",

            data=tips);
```



A few things have happened here. Let’s go through them one by one:

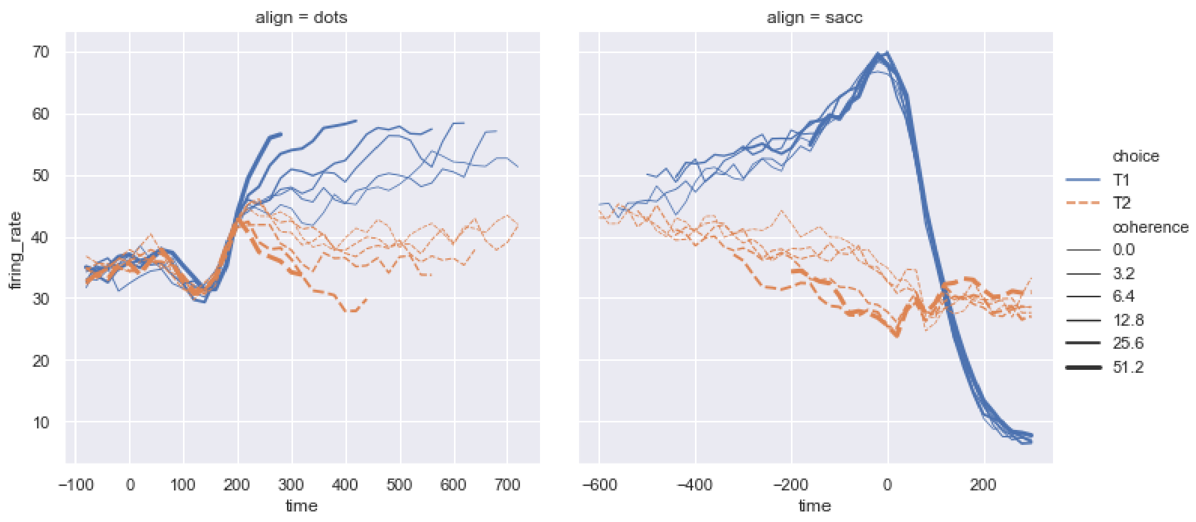
1. We import seaborn, which is the only library necessary for this simple example.

API abstraction across visualizations

There is no universal best way to visualize data. Different questions are best answered by different kinds of visualizations. Seaborn tries to make it easy to switch between different visual representations that can be parameterized with the same dataset-oriented API.

The function **relplot()** is named that way because it is designed to visualize many different statistical *relationships*. While scatter plots are a highly effective way of doing this, relationships where one variable represents a measure of time are better represented by a line. The **relplot()** function has a convenient kind parameter to let you easily switch to this alternate representation:

```
dots = sns.load_dataset("dots")

sns.relplot(x="time", y="firing_rate", col="align",

            hue="choice", size="coherence", style="choice",

              facet_kws=dict(sharex=False),

              kind="line", legend="full", data=dots);
```

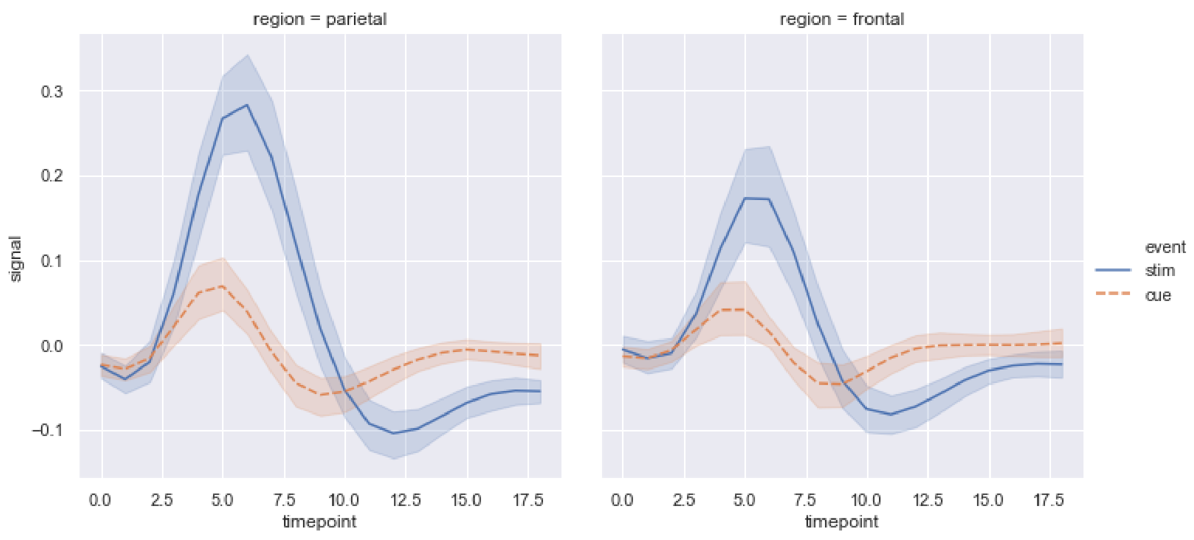


Notice how the `size` and `style` parameters are shared across the scatter and line plots, but they affect the two visualizations differently (changing marker area and symbol vs line width and dashing). We did not need to keep those details in mind, letting us focus on the overall structure of the plot and the information we want it to convey.

Statistical estimation and error bars

Often we are interested in the average value of one variable as a function of other variables. Many seaborn functions can automatically perform the statistical estimation that is neccesary to answer these questions:

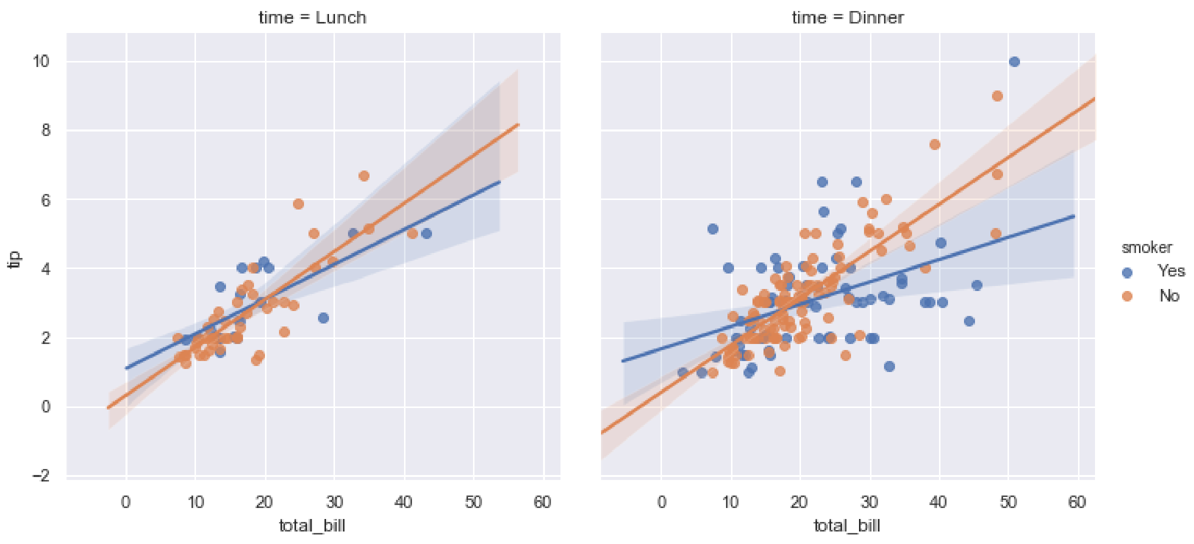
```
fmri = sns.load_dataset("fmri")

sns.relplot(x="timepoint", y="signal", col="region",

            hue="event", style="event",

            kind="line", data=fmri);
```



When statistical values are estimated, seaborn will use bootstrapping to compute confidence intervals and draw error bars representing the uncertainty of the estimate.

Statistical estimation in seaborn goes beyond descriptive statisitics. For example, it is also possible to enhance a scatterplot to include a linear regression model (and its uncertainty) using **`lmplot()`**:

```
sns.lmplot(x="total_bill", y="tip", col="time", hue="smoker",

           data=tips);
```

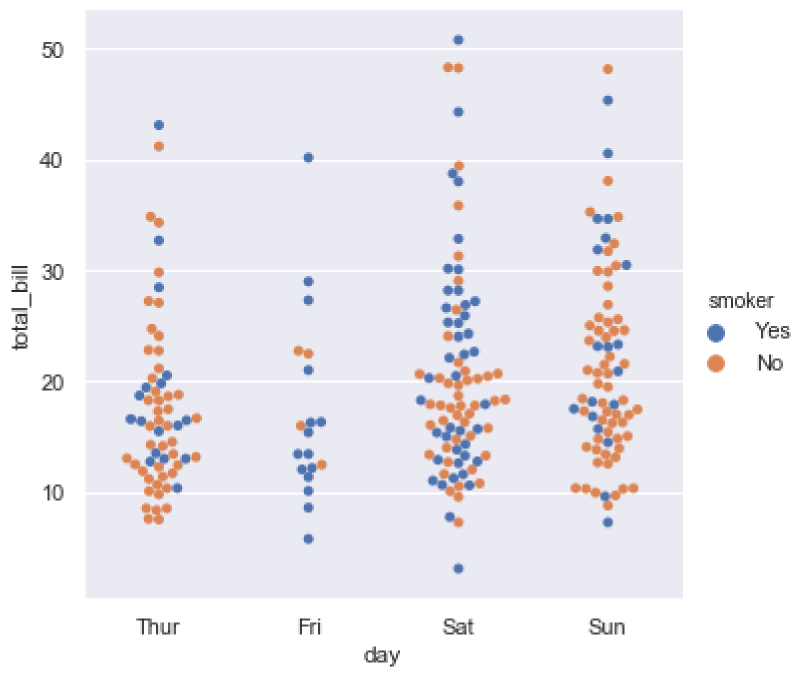


Specialized categorical plots

Standard scatter and line plots visualize relationships between numerical variables, but many data analyses involve categorical variables. There are several specialized plot types in seaborn that are optimized for visualizing this kind of data. They can be accessed through **catplot()**. Similar to **relplot()**, the idea of **catplot()** is that it exposes a common dataset-oriented API that generalizes over different representations of the relationship between one numeric variable and one (or more) categorical variables.

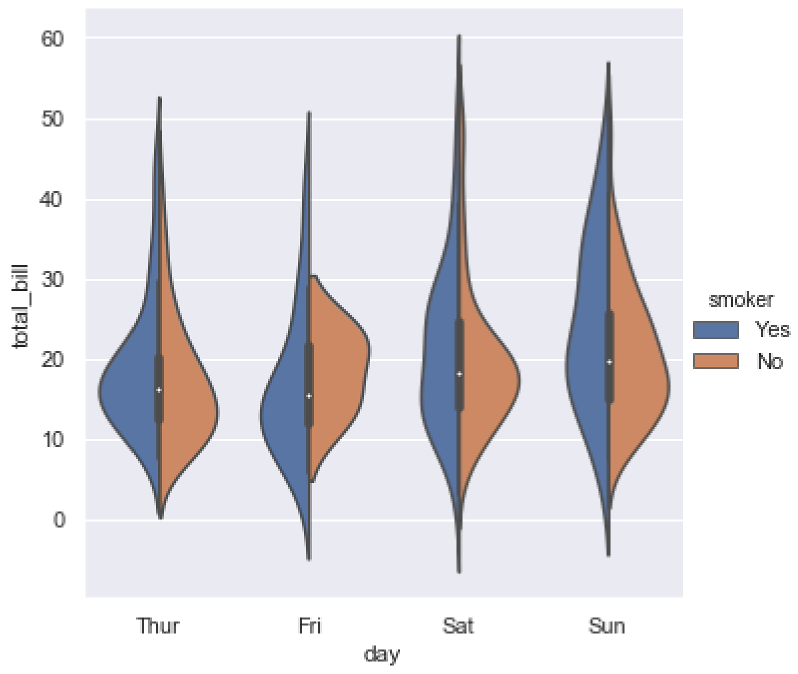
These representations offer different levels of granularity in their presentation of the underlying data. At the finest level, you may wish to see every observation by drawing a scatter plot that adjusts the positions of the points along the categorical axis so that they don't overlap:

```
sns.catplot(x="day", y="total_bill", hue="smoker",

            kind="swarm", data=tips);
```



Alternately, you could use kernel density estimation to represent the underlying distribution that the points are sampled from:

```
sns.catplot(x="day", y="total_bill", hue="smoker",

        kind="violin", split=True, data=tips);
```



Or you could show the only mean value and its confidence interval within each nested category:

```
sns.catplot(x="day", y="total_bill", hue="smoker",

            kind="bar", data=tips);
```

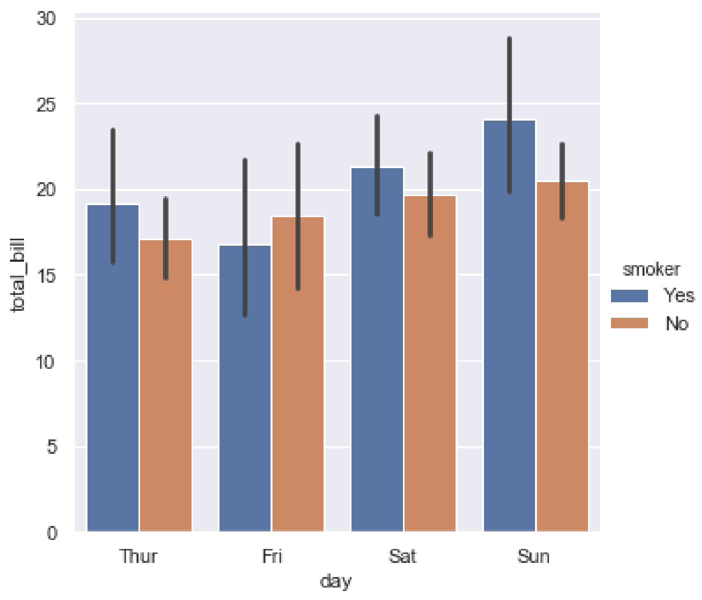


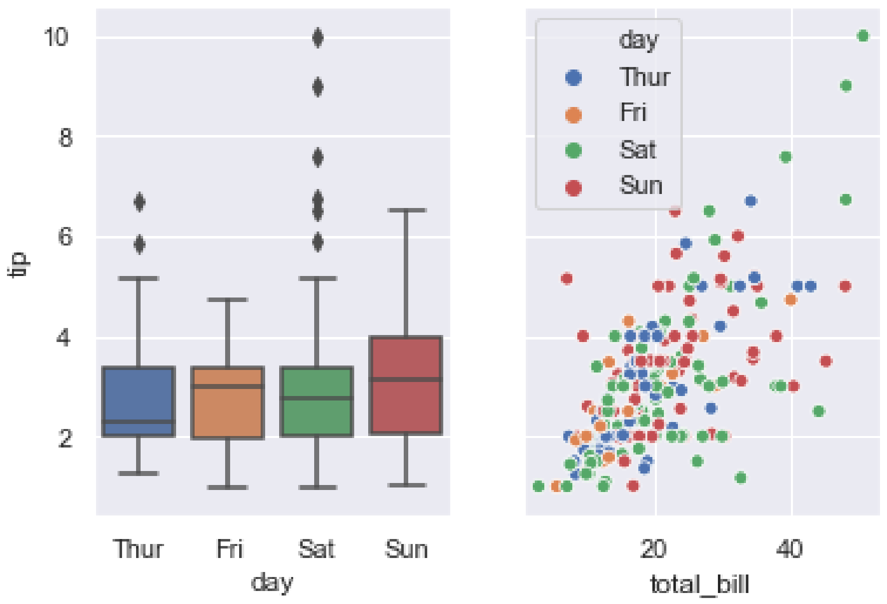
Figure-level and axes-level functions

How do these tools work? It’s important to know about a major distinction between seaborn plotting functions. All of the plots shown so far have been made with “figure-level” functions. These are optimized for exploratory analysis because they set up the matplotlib figure containing the plot(s) and make it easy to spread out the visualization across multiple axes. They also handle some tricky business like putting the legend outside the axes. To do these things, they use a seaborn **FacetGrid**.

Each different figure-level plot kind combines a particular “axes-level” function with the **FacetGrid** object. For example, the scatter plots are drawn using the **scatterplot()** function, and the bar plots are drawn using the **barplot()** function. These functions are called “axes-level” because they draw onto a single matplotlib axes and don’t otherwise affect the rest of the figure.

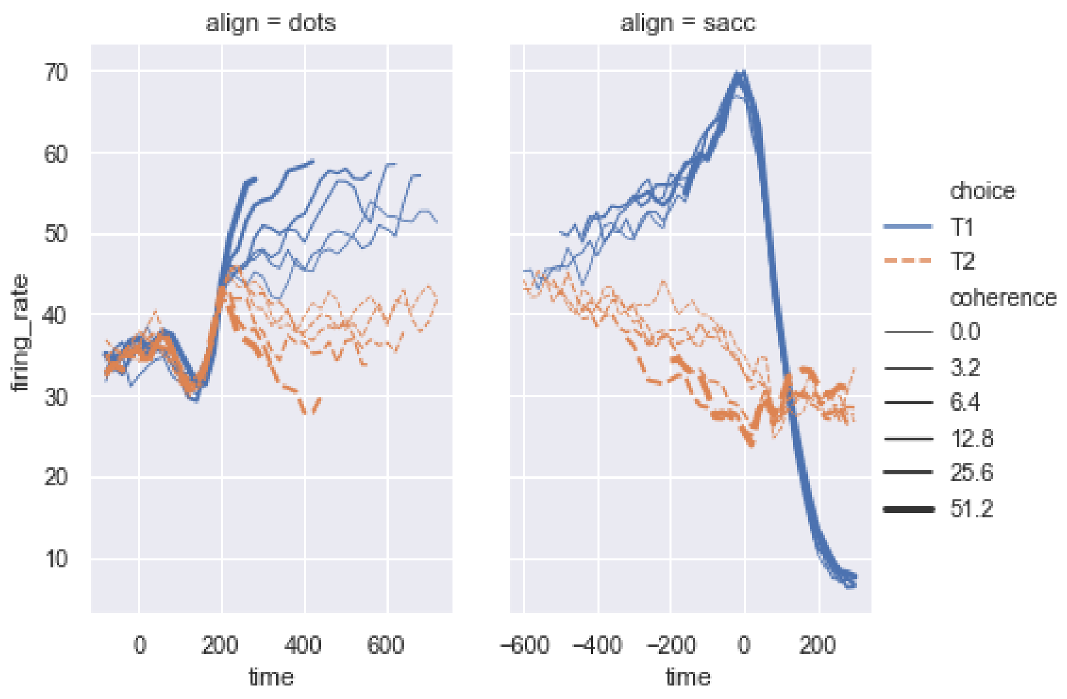
The upshot is that the figure-level function needs to control the figure it lives in, while axes-level functions can be combined into a more complex matplotlib figure with other axes that may or may not have seaborn plots on them:

```
import matplotlib.pyplot as plt

f, axes = plt.subplots(1, 2, sharey=True, figsize=(6, 4))

sns.boxplot(x="day", y="tip", data=tips, ax=axes[0])

sns.scatterplot(x="total_bill", y="tip", hue="day", data=tips, ax=axes[1]);
```



Controling the size of the figure-level functions works a little bit differently than it does for other matplotlib figures. Instead of setting the overall figure size, the figure-level functions are parameterized by the size of each facet. And instead of setting the height and width of each facet, you control the height and *aspect* ratio (ratio of width to height). This parameterization makes it easy to control the size of the graphic without thinking about exactly how many rows and columns it will have, although it can be a source of confusion:

```
sns.relplot(x="time", y="firing_rate", col="align",

            hue="choice", size="coherence", style="choice",

            height=4.5, aspect=2 / 3,

            facet_kws=dict(sharex=False),

            kind="line", legend="full", data=dots);
```

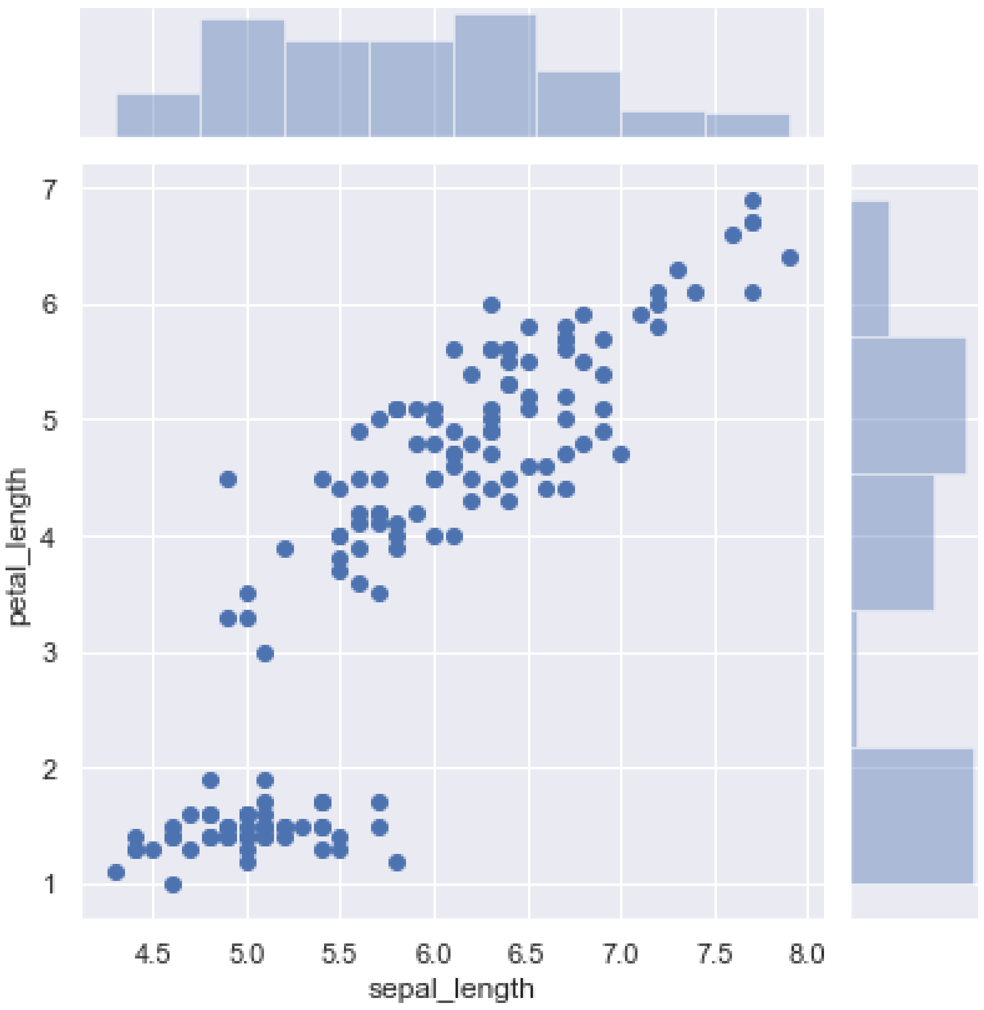


The way you can tell whether a function is "figure-level" or "axes-level" is whether it takes an ax= parameter. You can also distinguish the two classes by their output type: axes-level functions return the matplotlib axes, while figure-level functions return the **FacetGrid**.

Visualizing dataset structure

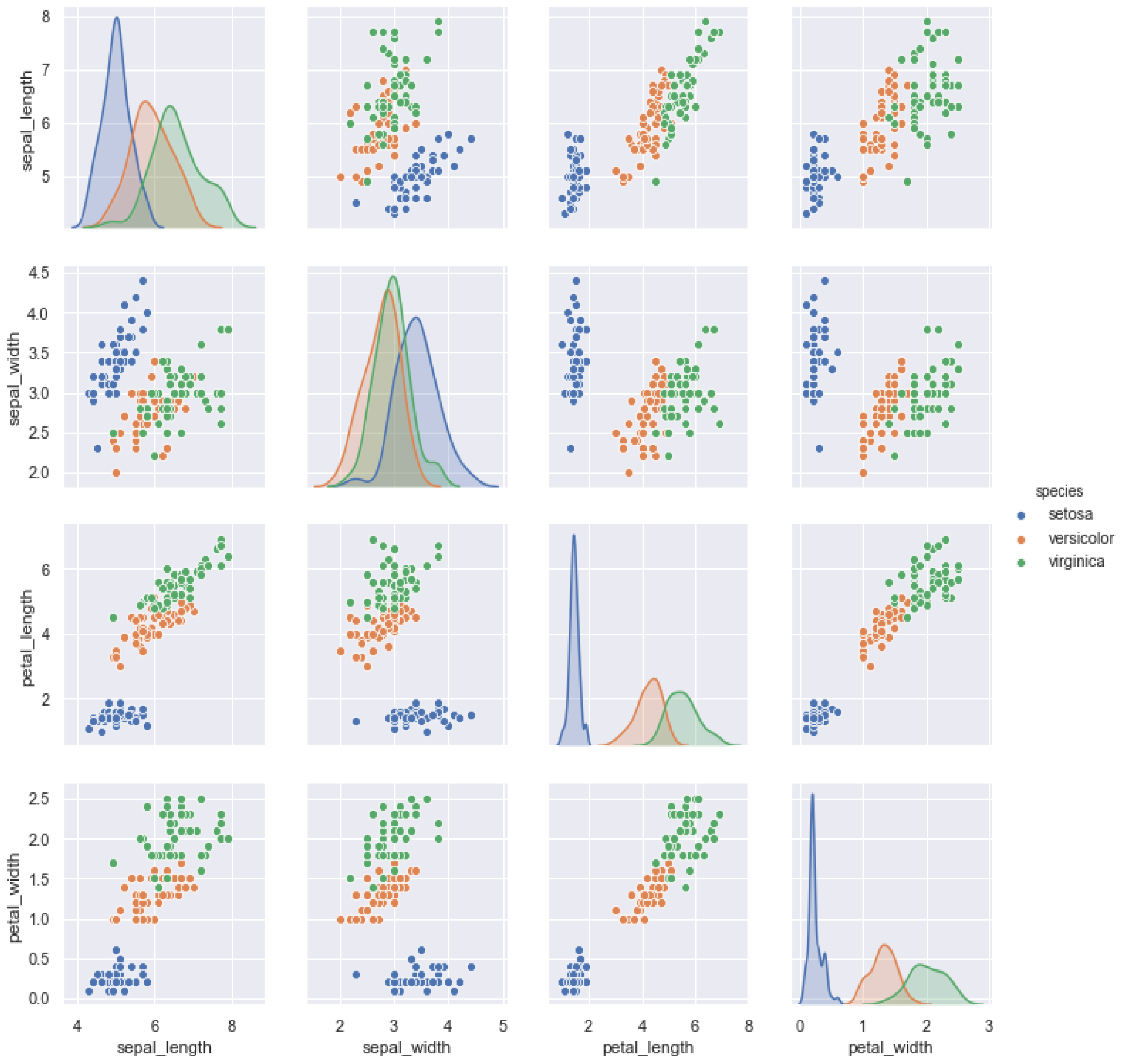
There are two other kinds of figure-level functions in seaborn that can be used to make visualizations with multiple plots. They are each oriented towards illuminating the structure of a dataset. One, **jointplot()**, focuses on a single relationship:

```
iris = sns.load_dataset("iris")

sns.jointplot(x="sepal_length", y="petal_length", data=iris);
```



The other, **pairplot()**, takes a broader view, showing all pairwise relationships and the marginal distributions, optionally conditioned on a categorical variable :

```
sns.pairplot(data=iris, hue="species");

sns.relplot(x="time", y="firing_rate", col="align",
```



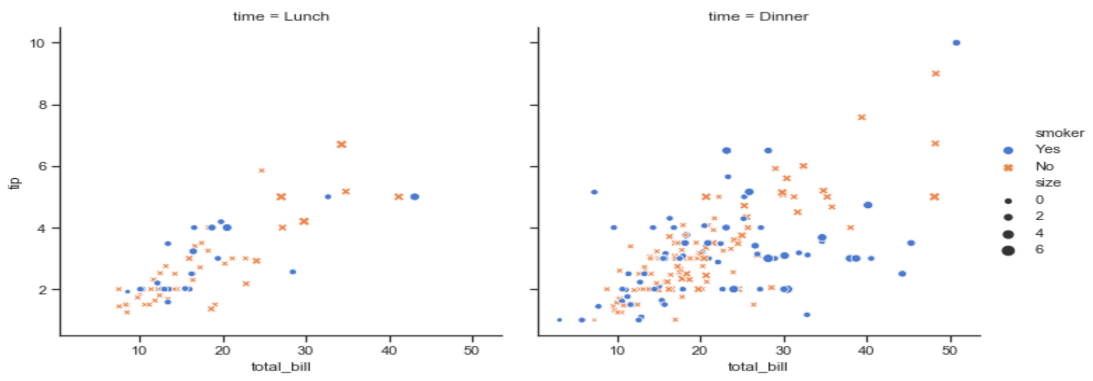
Both **jointplot()** and **pairplot()** have a few different options for visual representation, and they are built on top of classes that allow more thoroughly customized multi-plot figures (**JointGrid** and **PairGrid**, respectively).

Customizing plot appearance

The plotting functions try to use good default aesthetics and add informative labels so that their output is immediately useful. But defaults can only go so far, and creating a fully-polished custom plot will require additional steps. Several levels of additional customization are possible.

The first way is to use one of the alternate seaborn themes to give your plots a different look. Setting a different theme or color palette will make it take effect for all plots:

```
sns.set(style="ticks", palette="muted")

sns.relplot(x="total_bill", y="tip", col="time",

            hue="smoker", style="smoker", size="size",

            data=tips);
```



For figure-specific customization, all seaborn functions accept a number of optional parameters for switching to non-default semantic mappings, such as different colors. (Appropriate use of color is critical for effective data visualization, and seaborn has extensive support for customizing color palettes).

Organizing datasets

As mentioned above, seaborn will be most powerful when your datasets have a particular organization. This format ia alternately called “long-form” or “tidy” data and is described in detail by Hadley Wickham in this academic paper. The rules can be simply stated:

1. Each variable is a column
2. Each observation is a row

A helpful mindset for determining whether your data are tidy is to think backwards from the plot you want to draw. From this perspective, a “variable” is something that will be assigned a role in the plot. It may be useful to look at the example datasets and see how they are structured. For example, the first five rows of the “tips” dataset look like this:

tips.head()

	total_bill	**tip**	**sex**	**smoker**	**day**	**time**	**size**
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Table: 3.5.4.1 Organizing dataset**s**

In some domains, the tidy format might feel awkward at first. Tim series data, for example, are sometimes stored with every timepoint as part of the same observational unit and appearing in the columns. The “fmri” dataset that we used above illustrates how a tidy timeseries dataset has each timepoint in a different row:

fmri.head()

	subject	**timepoint**	**event**	**region**	**signal**
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

Table: 3.5.4.2 Organizing datasets

3.5.5 INSTALIZATION OF PYTHON IN WINDOWS

When you visit the Python for Windows download page, you'll immediately see the division. Right at the top, square and center, the repository asks if you want the latest release of Python 2 or Python 3 (2.7.13 and 3.6.1, respectively, as of this tutorial)

You can download just Python 2 or Python 3 if you're sure you only need a particular version. We're going the distance today and will be installing both of them, so we recommend you download both versions and do the same. Under the main entry for both versions you'll see an "x86-64" installer, as seen below.

Python 3 Installation on Windows

Step 1: Select Version of Python to Install
Step 2: Download Python Executable Installer
Step 3: Run Executable Installer
Step 4: Verify Python Was Installed On Windows
Step 5: Verify Pip Was Installed
Step 6: Add Python Path to Environment Variables (Optional)
Step 7: By using pip We have to download Numpy , Pandas &Matplotlib.
Step 8: Install virtualnv (Optional)

Chapter-4

SYSTEM DESIGN

4.1 SYSTEM USE CASE DIAGRAM

Use case diagrams are a way to capture the system's functionality and requirements in UML diagrams. It captures the dynamic behavior of a live system. A use case diagram consists of a use case and an actor. A use case represents a distinct functionality of a system, a component, a package, or a class

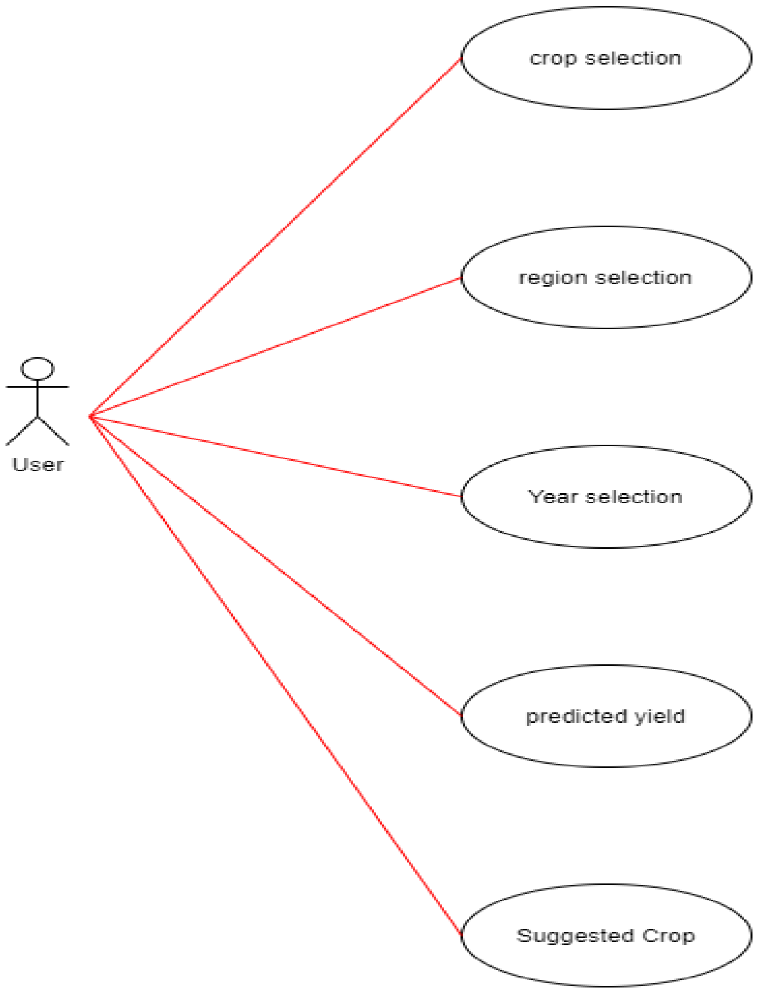


Figure 4.1: **Use Case Diagram**

4.2 SYSTEM DESIGN FLOW CHART

In this above flow chart collecting the data and performing a operation like pre-processing which is used to convert the data and than trained the data and than we evaluate all the data and than finally it shows the prediction of the crop.

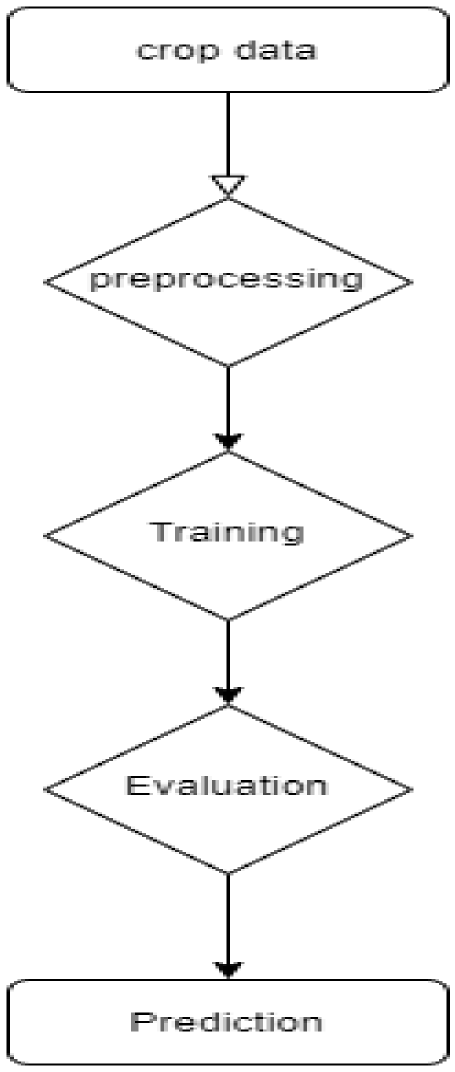


Figure 4.2 : **system Flow chart**

4.3 SYSTEM DESIGN ACTIVITY DIAGRAM

Activity diagrams are an essential part of the modeling process. They are used to clarify complicated use cases, illustrate control among objects, or to show the logic of an algorithm.

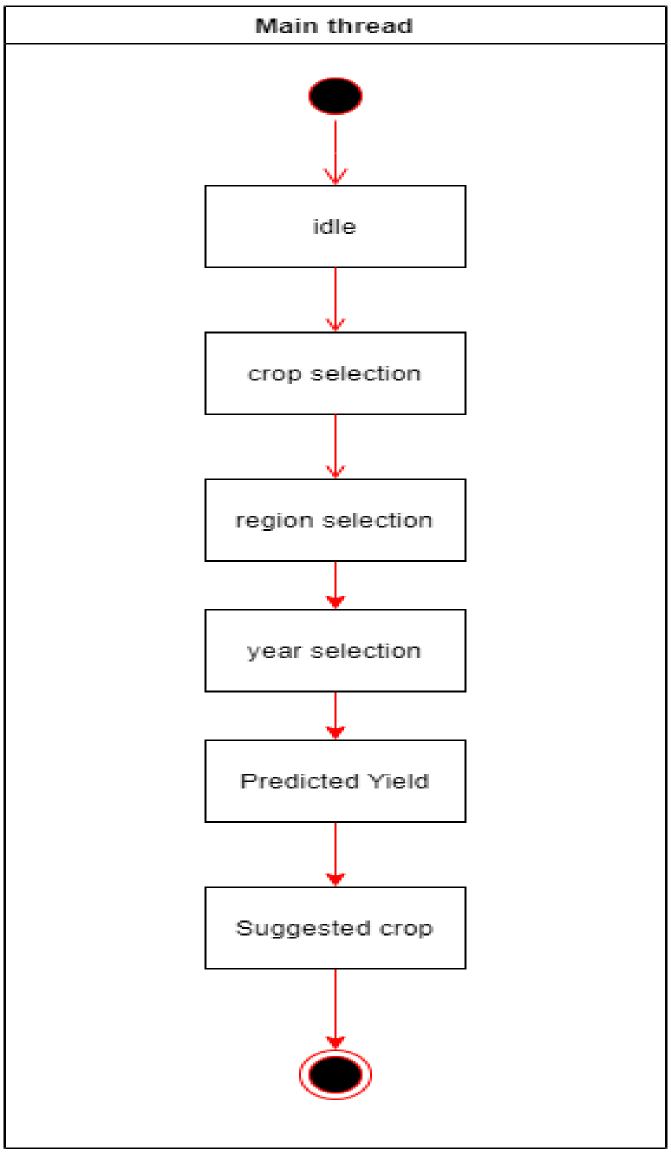


Figure 4.3 : **Activity Diagram**

4.4 SYSTEM DESIGN SEQUENCE DIAGRAM

The **sequence diagram** is a good **diagram** to **use** to document a system's requirements and to flush out a system's design. The reason the **sequence diagram** is so useful is because it shows the interaction logic between the objects in the system in the time order that the interactions take place.

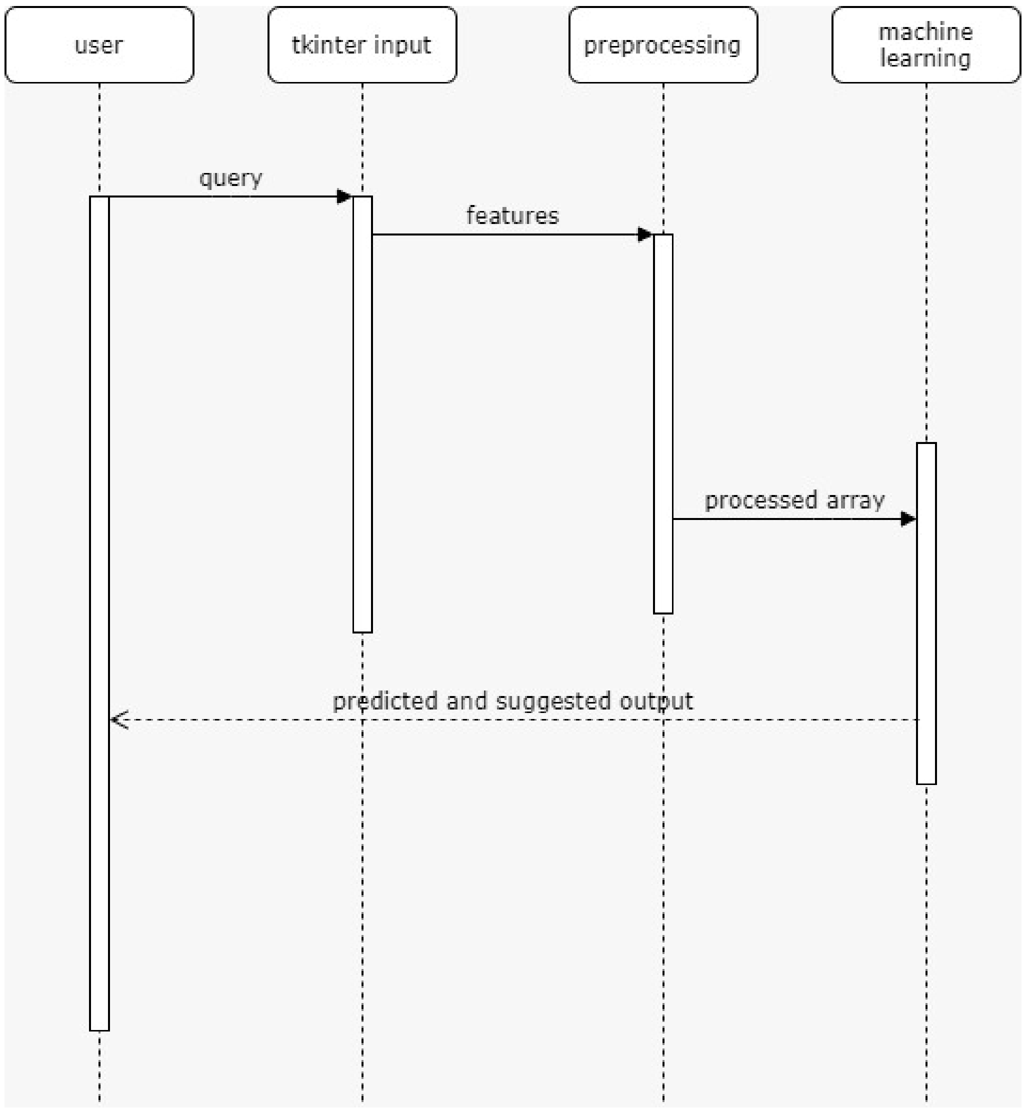


Figure 4.4 : **Sequence diagram**

4.5 CLASS DIAGRAM OF SYSTEM DESIGN

Class diagrams are the main building block in object-oriented modeling. They are used to show the different objects in a system, their attributes, their operations and the relationships among them.

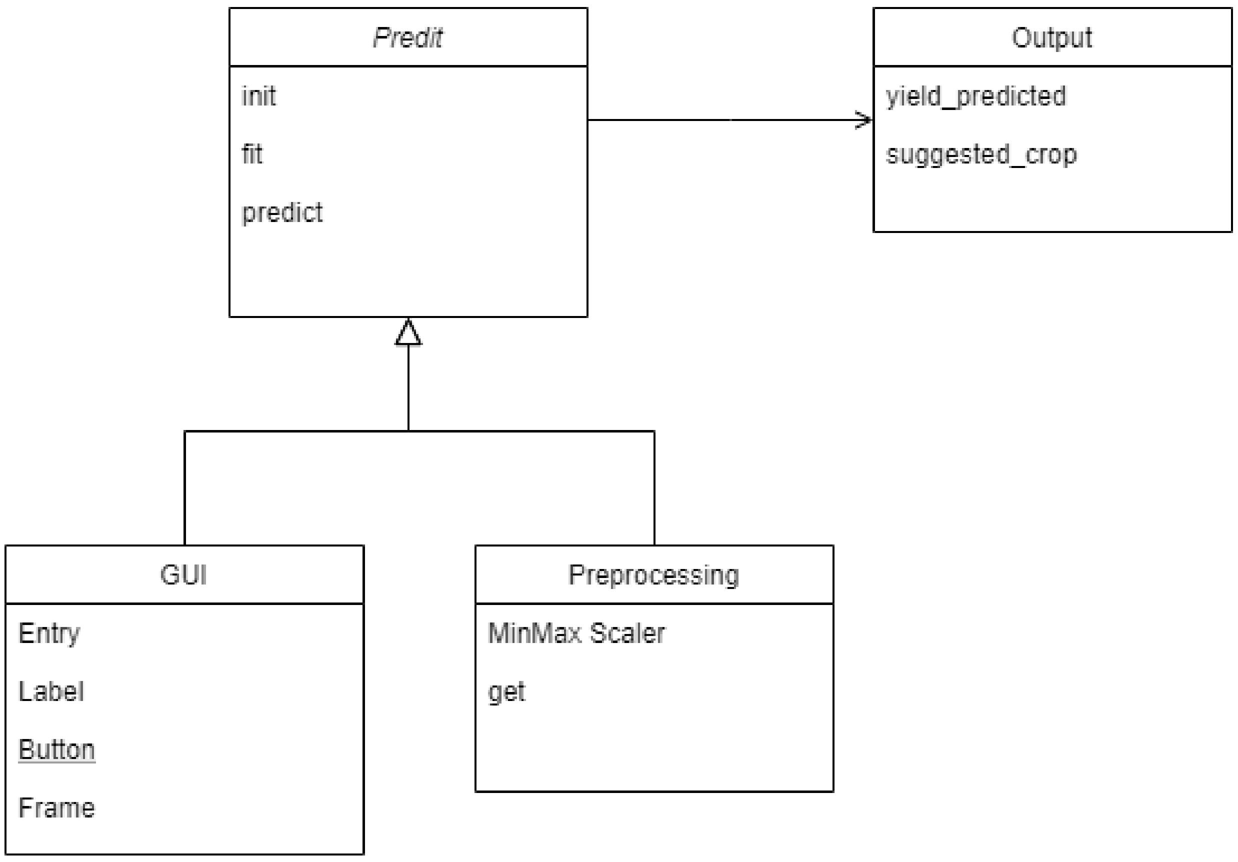


Figure 4.5 : **Class diagram**

CHAPTER-5

SYSTEM REQUIREMENTS

5.1 SOFTWARE REQUIREMENTS

- OS: Windows XP + / Linux/ MacOS
- Python 3
- Scikit-learn
- Pandas
- Numpy
- Matplotlib

5.2 HARDWARE REQUIREMENTS

Component	**Minimum requirement**
Processor	64-bit, i3 processor, 2.5 GHz minimum per core
RAM	8 GB for developer or evaluation use 16 GB for single server and multiple server farm installation for production use
Hard disk	120 GB

CHAPTER-6

SOURCE CODE

6.1 SOURCE CODE:

- HERE WE USING JOBLIB FOR AUTO SAVING. THE TK NTER IS AN GUI TOOL.

```
import joblib as jb

import pandas as pd

import tkinter as tk

from PIL import ImageTk,Image

import numpy as np

import matplotlib.pyplot as plt

from tkinter import ttk

import systemcheck
```

- HERE WE DID DATA LOADING WHICH IS STORE IN AGRICULTURE.CSV FILE.

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

- HERE WE CREATE DICTIONARY AND STORE UNIQUE NAME INDIVIDUAL.

```
d={}

for i in data["State_Name"].unique():
    d[i]=list(data[data["State_Name"]==i]["District_Name"].unique())

    s_n=list(data["State_Name"].unique())

d_n=list(data["District_Name"].unique())

s=list(data["Season"].unique())

c=list(data["Crop"].unique())
```

- HERE WE USE THIS FOR UPLOADING THE DATA

```
rfr1=jb.load("3054rfr.sav")
```

- HERE TKINTER IS GUI NAME AND IT WINDOW SIZE IS SPECIFIED

```
r=tk.Tk()

r.geometry("845x440")
```

- HERE WE USE FUNCTION FOR THE GUI AND WE CALL THEM

```
def fun6():

    f3.destroy()

    f1=tk.Frame(r)

    f1.place(x=0,y=0,width=845,height=440)

    fun1()
```

- HERE IT SAME CREATING FUNCTION AND CALL THEM FRAME.

```
def fun3():

    global a1
    global a2

    global a3

    global a4

    global a5

    global a6,f3

    f2.destroy()

    f3=tk.Frame(r)

    f3.place(x=0,y=0,width=850,height=480)
```

- HERE WE USE FOR LOADING THE IMAGE WHICH YOU CAN SEE IN GUI PAGE.

```
background_image=ImageTk.PhotoImage(file='crop.jpeg')

background_label = tk.Label(f3,image=background_image)

background_label.image=background_image

background_label.place(x=0, y=0, relwidth=1, relheight=1)

a1=s_n.index(a1)

a2=d_n.index(a2)

a4=s.index(a4)

a5=c.index(a5)

global rfr1
```

- HERE WE CREATE LIST AND USE ITERATION AND ITERATING THE DATA.

```
    l=[]

    for i in range(len(c)):

        a={}

a={'State_Name':pd.Series(a1),'District_Name':pd.Series(a2),'Crop_Year':pd.Series(a3),'S
eason':pd.Series(a4),'Crop':pd.Series(a5),'Area':pd.Series(a6)}

        a["Crop"]=i

        x_input=pd.DataFrame(a)

        p=rfr1.predict(x_input)

        l.append(p)

    gh=max(l)

   gh=l.index(gh)

    ghc=c[gh]

    print("\n\n\n",l)

    a={}

a={'State_Name':pd.Series(a1),'District_Name':pd.Series(a2),'Crop_Year':pd.Series(a3),'S
eason':pd.Series(a4),'Crop':pd.Series(a5),'Area':pd.Series(a6)}

    x_input=pd.DataFrame(a)

    print("\n\n\n",x_input)
    pre=rfr1.predict(x_input)

    pre=pre[0]

    print("\n\n\nyour output",pre)

    l14=tk.Label(f3,text="",font=("monaco",30,"bold"),bg="seagreen1")

     l14.place(x=120,y=65)

    l14.pack(pady=20)

    l14.config(text=f"The predited production value\nfor given details is given below\n{pre}\n\nAnd the best suggested crop\nin the given region is:\n{ghc}")

    b5=tk.Button(f3,text="predict again",font=("monaco",25,"bold"),fg="green",command=fun6)

    b5.place(x=320,y=370)
```

- HERE IF YOU PUT ANYTHING WRONG THE FUNCTION IS EXIT

```
def fun5():

    f4.destroy()

   f1=tk.Frame(r)

    f1.place(x=0,y=0,width=845,height=440)

    fun1()
```

- HERE WE USE IF YOU ENTER WRONG VALUE HE WILL REMAIND YOU APPROPRIATE INPUT TO RE-ENTER.

```
def fun4():
    f2.destroy()

    global f4

    f4=tk.Frame(r,bg="salmon",width=845,height=440)

    f4.place(x=0,y=0)

    l_f4=tk.Label(text="Please enter the correct values\nand\n please do not leave anything
empty",font=("monaco",30,"bold"),fg="firebrick1",bg="bisque")

    # l_f4.place(x=75,y=170)

    l_f4.pack(pady=70)

    b_f4=tk.Button(text="<--
back",font=("monaco",30,"bold"),fg="firebrick1",command=fun5)

    b_f4.place(x=40,y=340)

def fun2():

    c=check()

    if(c==1):

        fun3()

    else:

        fun4()

def check():

    global a1,a2,a3,a4,a5,a6

    a1=str(e1.get())
    a2=str(e2.get())

    a4=str(e4.get())

    a5=str(e5.get())

    try:

        if a1 not in s_n:

            raise

        if a2 not in d[a1]:

            raise

        if a4 not in s:

            raise

        if a5 not in c:

            raise

        a3=float(e3.get())

        a6=float(e6.get())

        return 1

    except:

        return 0

def select_dis(a):

    global a1
    print(a)

   a1=e1.get()

    e2.delete(0,"end")

    e2.configure(values=d[a1])
```

- HERE WE USE SOME BUTTON , AND GUI FRAME WORK DONE IN THIS LIKE STYLING

```
def fun1():

    global e1,e2,e3,e4,e5,e6,e7,e8,f2

    f1.destroy()

    f2=tk.Frame(r)

    f2.place(x=0,y=0,width=850,height=480)

    background_image=ImageTk.PhotoImage(file='crop.jpeg')

    background_label = tk.Label(f2,image=background_image)

    background_label.image=background_image

    background_label.place(x=0, y=0, relwidth=1, relheight=1)

    xref=100

    yref=50

    x1ref=480

    l1=tk.Label(f2,text="State Name",fg="green",font=("monaco",20,"bold"))
```

```
 l1.place(x=xref,y=yref)

e1=ttk.Combobox(f2,values=s_n)

 e1.place(x=xref,y=yref+35)

 e1.bind("<<ComboboxSelected>>",select_dis)

 l2=tk.Label(f2,text="District Name",fg="green",font=("monaco",20,"bold"))

 l2.place(x=xref,y=yref+90)

 e2=ttk.Combobox(f2,values=["select state name"])

 e2.place(x=xref,y=yref+125)

 l3=tk.Label(f2,text="Crop Year",fg="green",font=("monaco",20,"bold"))

 l3.place(x=xref,y=yref+180)

 e3=tk.Entry(f2,font=("monaco",20,"bold"))

 e3.place(x=xref,y=yref+215)

 l4=tk.Label(f2,text="Season",fg="green",font=("monaco",20,"bold"))

 l4.place(x=x1ref,y=yref)

 e4=ttk.Combobox(f2,values=s)

 e4.place(x=x1ref,y=yref+35)

 l5=tk.Label(f2,text="Crop",fg="green",font=("monaco",20,"bold"))

 l5.place(x=x1ref,y=yref+90)

 e5=ttk.Combobox(f2,values=c)
```

```
   e5.place(x=x1ref,y=yref+125)

  l6=tk.Label(f2,text="Area",fg="green",font=("monaco",20,"bold"))

   l6.place(x=x1ref,y=yref+180)

   e6=tk.Entry(f2,font=("monaco",20,"bold"))

   e6.place(x=x1ref,y=yref+215)

  b2=tk.Button(f2,text="predict",font=("monaco",25,"bold"),fg="green",command=fun2)

   b2.place(x=360,y=360)

       f1=tk.Frame(r)

f1.place(x=0,y=0,width=850,height=480)

background_image=ImageTk.PhotoImage(file='crop.jpeg')

background_label = tk.Label(f1,image=background_image)

background_label.image=background_image

background_label.place(x=0, y=0, relwidth=1, relheight=1)

b1=tk.Button(f1,text="Predict\ncrop
production",font=("monaco",25,"bold"),command=fun1)

b1.config(fg="green",justify="center",relief="groove")

 b1.place(x=308,y=182)

b1.pack(expand=True)

r.mainloop()
```

CHAPTER-7

TESTING

7.1 TEST CASES

Test cases can be divided in to two types. First one is Positive test cases and second one is negative test cases. In positive test cases are conducted by the developer intention is to get the output. In negative test cases are conducted by the developer intention is to don't get the output.

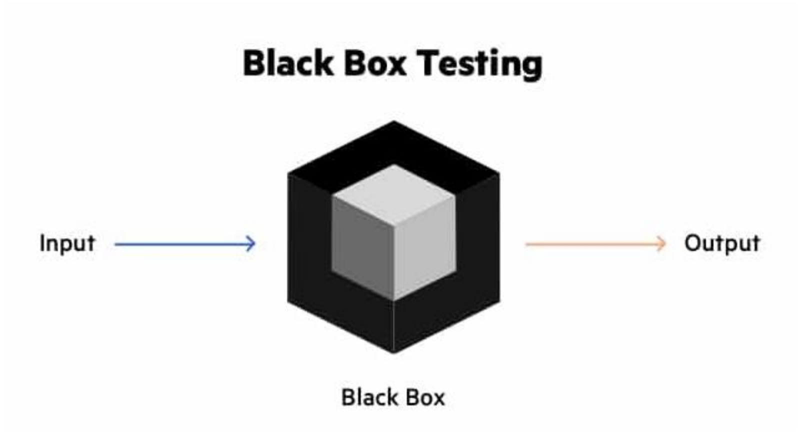
7.2 BLACK BOX TESTING

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing

Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

Black box testing is a powerful testing technique because it exercises a system end-to-end. Just like end-users "don't care" how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system delivers on its promises. Along the way, a black box test evaluates all relevant subsystems, including UI/UX, web server or application server, database, dependencies, and integrated systems.

An example of a security technology that performs black box testing is Dynamic Application Security Testing (DAST), which tests products in staging or production and provides feedback on compliance and security issues.



7.2.1 Types Of Black Box Testing

Black box testing can be applied to three main types of tests: functional, non-functional, and regression testing.

i) Functional Testing

Black box testing can test specific functions or features of the software under test. For example, checking that it is possible to log in using correct user credentials, and not possible to log in using wrong credentials.Functional testing can focus on the most critical aspects of the software (smoke testing/sanity testing), on integration between key components (integration testing), or on the system as a whole (system testing).

ii) Non-Functional Testing

Black box testing can check additional aspects of the software, beyond features and functionality. A non-functional test does not check “if” the software can perform a specific action but “how” it performs that action.Black box tests can uncover if software is:Usable and easy to understand for its users

- Performant under expected or peak loads
- Compatible with relevant devices, screen sizes, browsers or operating systems
- Exposed to security vulnerabilities or common security threats

7.3 WHITE BOX TESTING

WhiteBox Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing

7.4 UNIT TESTING

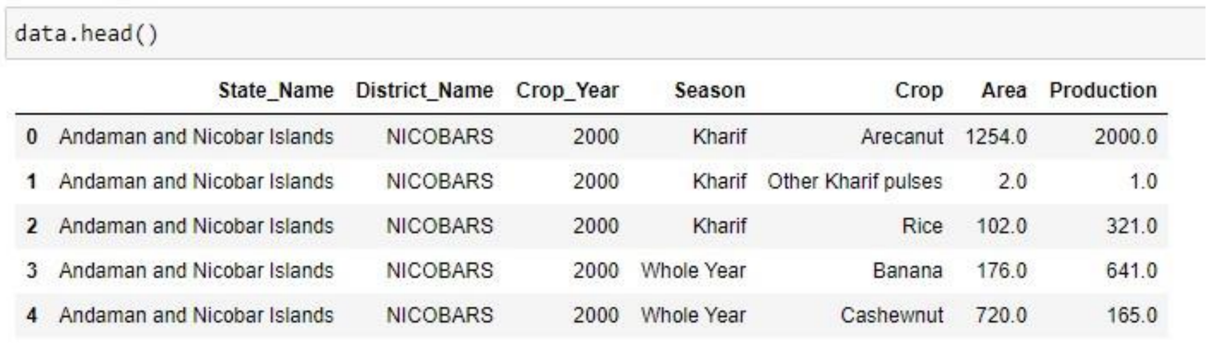
Unit Testing is a software Testing Technique in which parts of the projects are tested individually

CHAPTER-8

OUTPUT SNAPSHOT

8.1 JUYPTER NOTEBOOK CODE CHECKING

8.1.1 Head file data : Here we collecting first five data list into the file

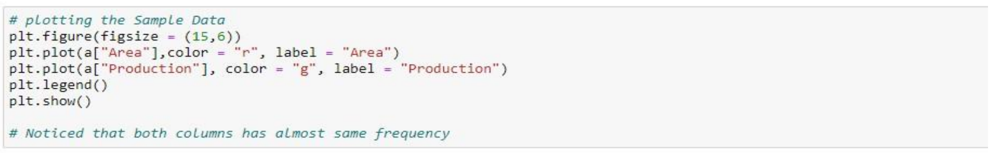


```
data.head()
```

	State_Name	District_Name	Crop_Year	Season	Crop	Area	Production
0	Andaman and Nicobar Islands	NICOBARS	2000	Kharif	Arecanut	1254.0	2000.0
1	Andaman and Nicobar Islands	NICOBARS	2000	Kharif	Other Kharif pulses	2.0	1.0
2	Andaman and Nicobar Islands	NICOBARS	2000	Kharif	Rice	102.0	321.0
3	Andaman and Nicobar Islands	NICOBARS	2000	Whole Year	Banana	176.0	641.0
4	Andaman and Nicobar Islands	NICOBARS	2000	Whole Year	Cashewnut	720.0	165.0

Table 8.1.1 : Head file data

8.1.2 Plotting the sample data: Here we plotting above data set into the graph, Here visible y-axis area &x-axis production.



```
# plotting the Sample Data
plt.figure(figsize = (15,6))
plt.plot(a["Area"],color = "r", label = "Area")
plt.plot(a["Production"], color = "g", label = "Production")
plt.legend()
plt.show()

# Noticed that both columns has almost same frequency
```

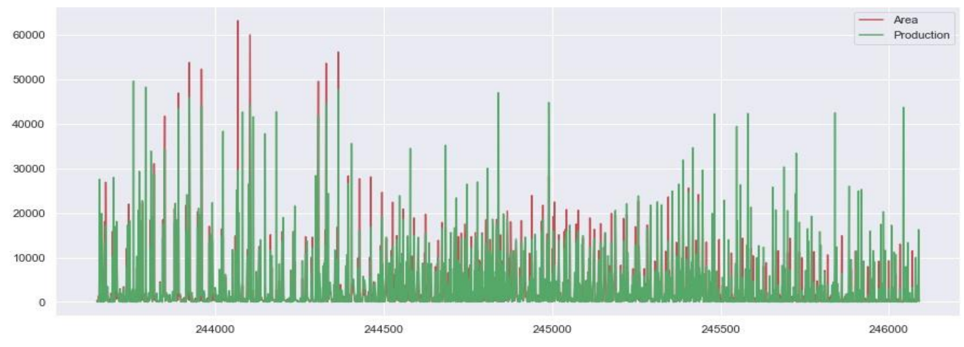
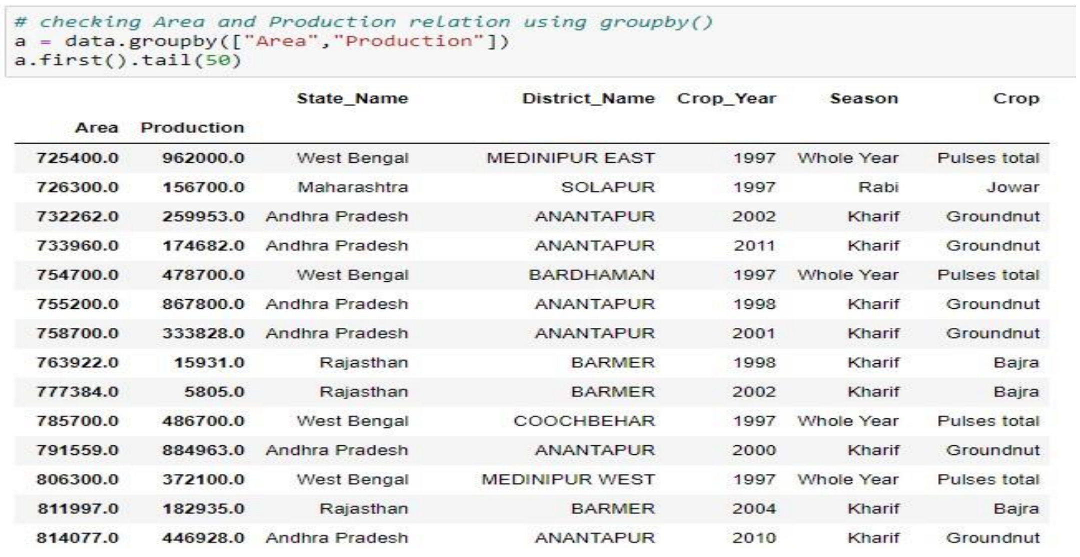


Figure 8.1.2 :Plotting the sample data

8.1.3 Area & Production Relation Data: The above graph and data table is represent ,area & production relation and graph shows



```
# checking Area and Production relation using groupby()
a = data.groupby(["Area","Production"])
a.first().tail(50)
```

Area	Production	State_Name	District_Name	Crop_Year	Season	Crop
725400.0	962000.0	West Bengal	MEDINIPUR EAST	1997	Whole Year	Pulses total
726300.0	156700.0	Maharashtra	SOLAPUR	1997	Rabi	Jowar
732262.0	259953.0	Andhra Pradesh	ANANTAPUR	2002	Kharif	Groundnut
733960.0	174682.0	Andhra Pradesh	ANANTAPUR	2011	Kharif	Groundnut
754700.0	478700.0	West Bengal	BARDHAMAN	1997	Whole Year	Pulses total
755200.0	867800.0	Andhra Pradesh	ANANTAPUR	1998	Kharif	Groundnut
758700.0	333828.0	Andhra Pradesh	ANANTAPUR	2001	Kharif	Groundnut
763922.0	15931.0	Rajasthan	BARMER	1998	Kharif	Bajra
777384.0	5805.0	Rajasthan	BARMER	2002	Kharif	Bajra
785700.0	486700.0	West Bengal	COOCHBEHAR	1997	Whole Year	Pulses total
791559.0	884963.0	Andhra Pradesh	ANANTAPUR	2000	Kharif	Groundnut
806300.0	372100.0	West Bengal	MEDINIPUR WEST	1997	Whole Year	Pulses total
811997.0	182935.0	Rajasthan	BARMER	2004	Kharif	Bajra
814077.0	446928.0	Andhra Pradesh	ANANTAPUR	2010	Kharif	Groundnut

Table:8.1.3 : Area & Production Relation Data

8.1.4 Crop production by season.

```
data.groupby(["Crop","Season"])["Production"].count().plot(figsize = (15,6), color = "g")
plt.title("Crop Production By Season", size = 20, color = "r")
plt.show()
```

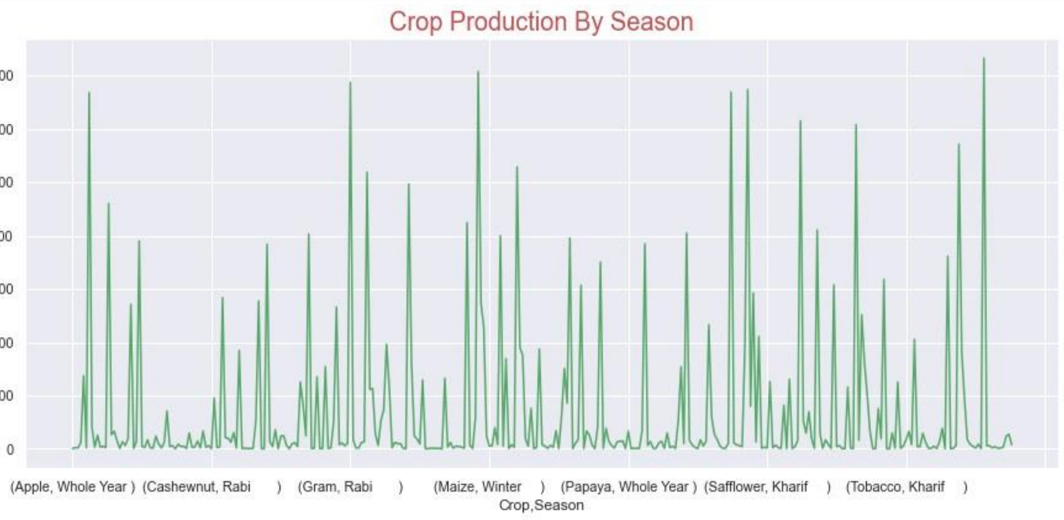
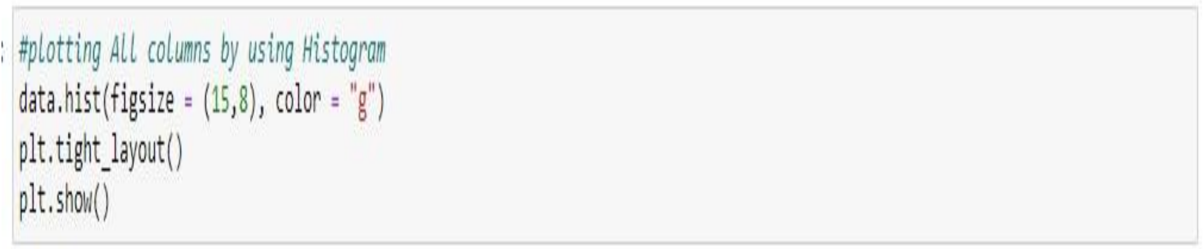


Figure:8.1.4 : Crop Production By Season

8.1.5 Plotting all columns by using Histogram: In this histogram it shows the all the columns individually showing the all the data & productions increases or decreases.



```
#plotting All columns by using Histogram
data.hist(figsize = (15,8), color = "g")
plt.tight_layout()
plt.show()
```

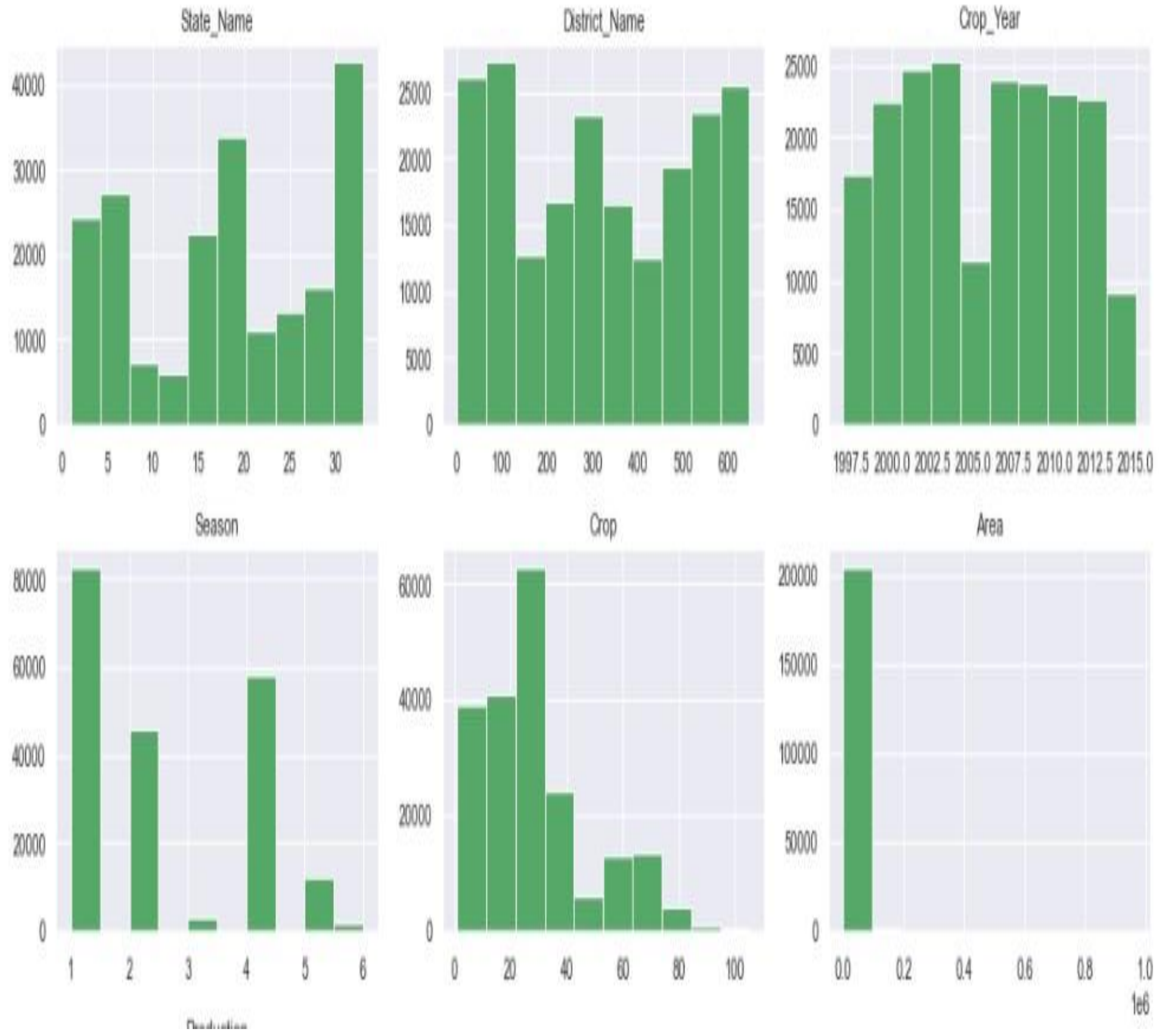


Figure:8.1.5 : Plotting all columns by using Histogram

8.1.6 Correlation of the Predicted data

Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statement about cause and effect.

Here Correlation is showing how the data highly co-related to other one here positive meaning is if one increases other one also increases ,negative means if one increases other one decrease and zero means it is not that much related.

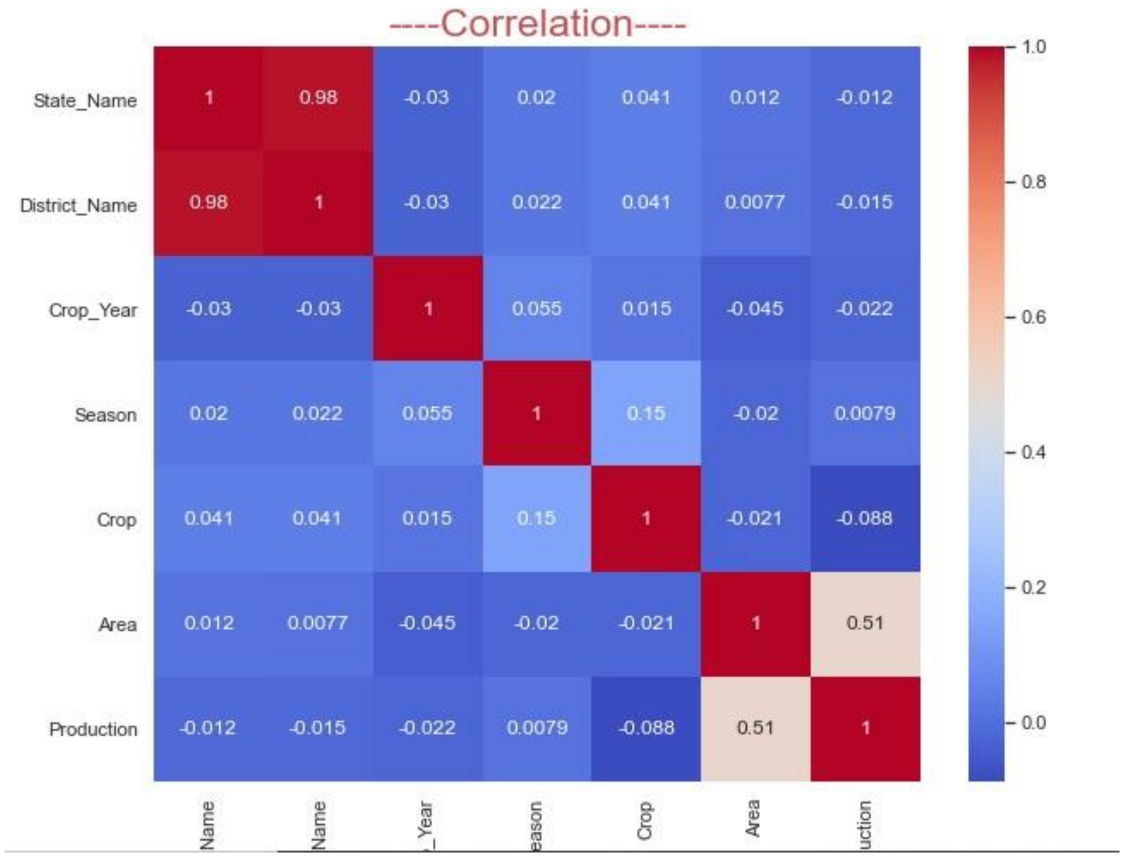


Figure:8.1.6 : Correlation of the Predicted data

8.2 Gui outputs

8.2.1 Crop Predict Application Button : After executing the code the GUI Window will be opened when we have click on the predict crop production then another tab will be opened.

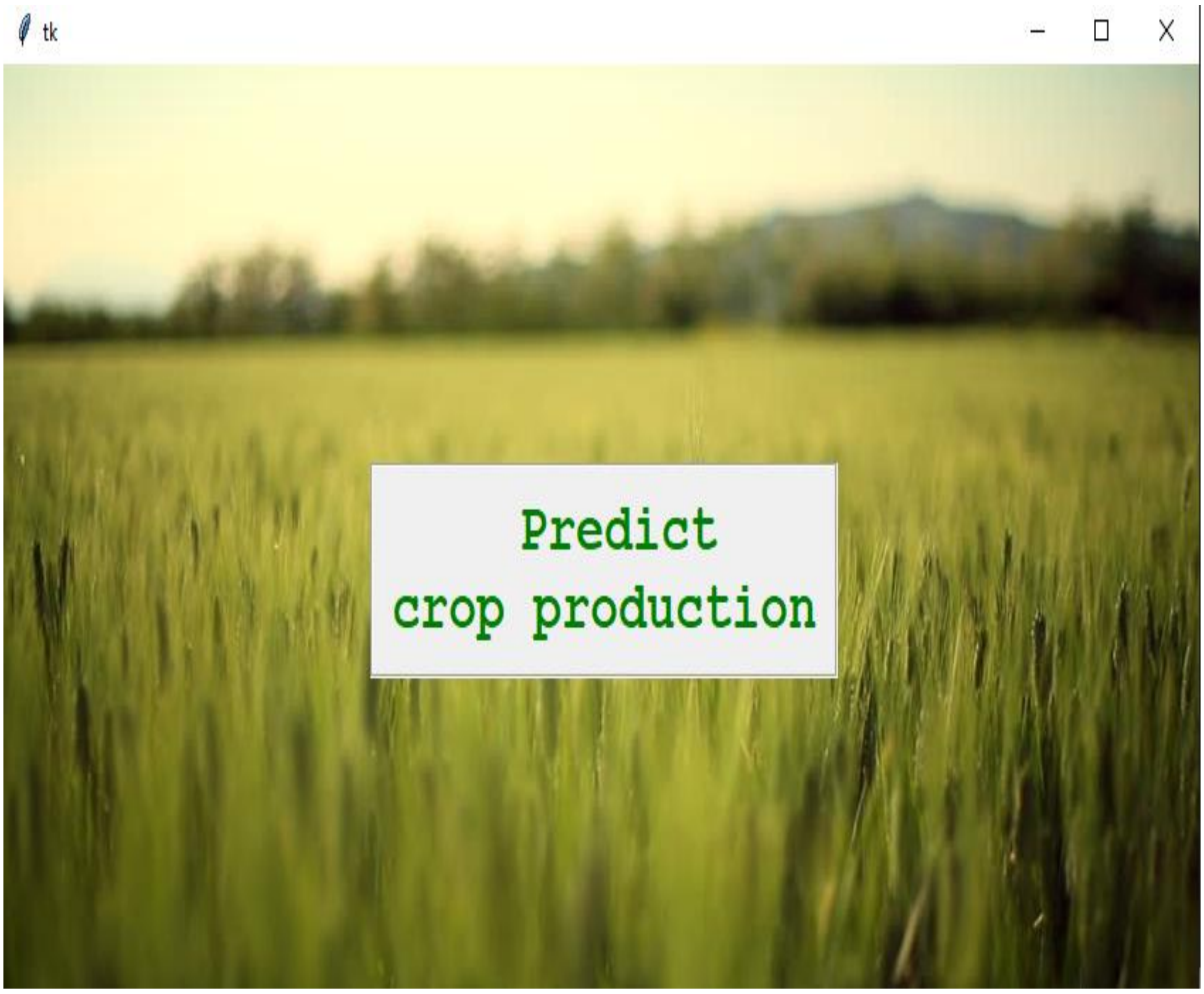
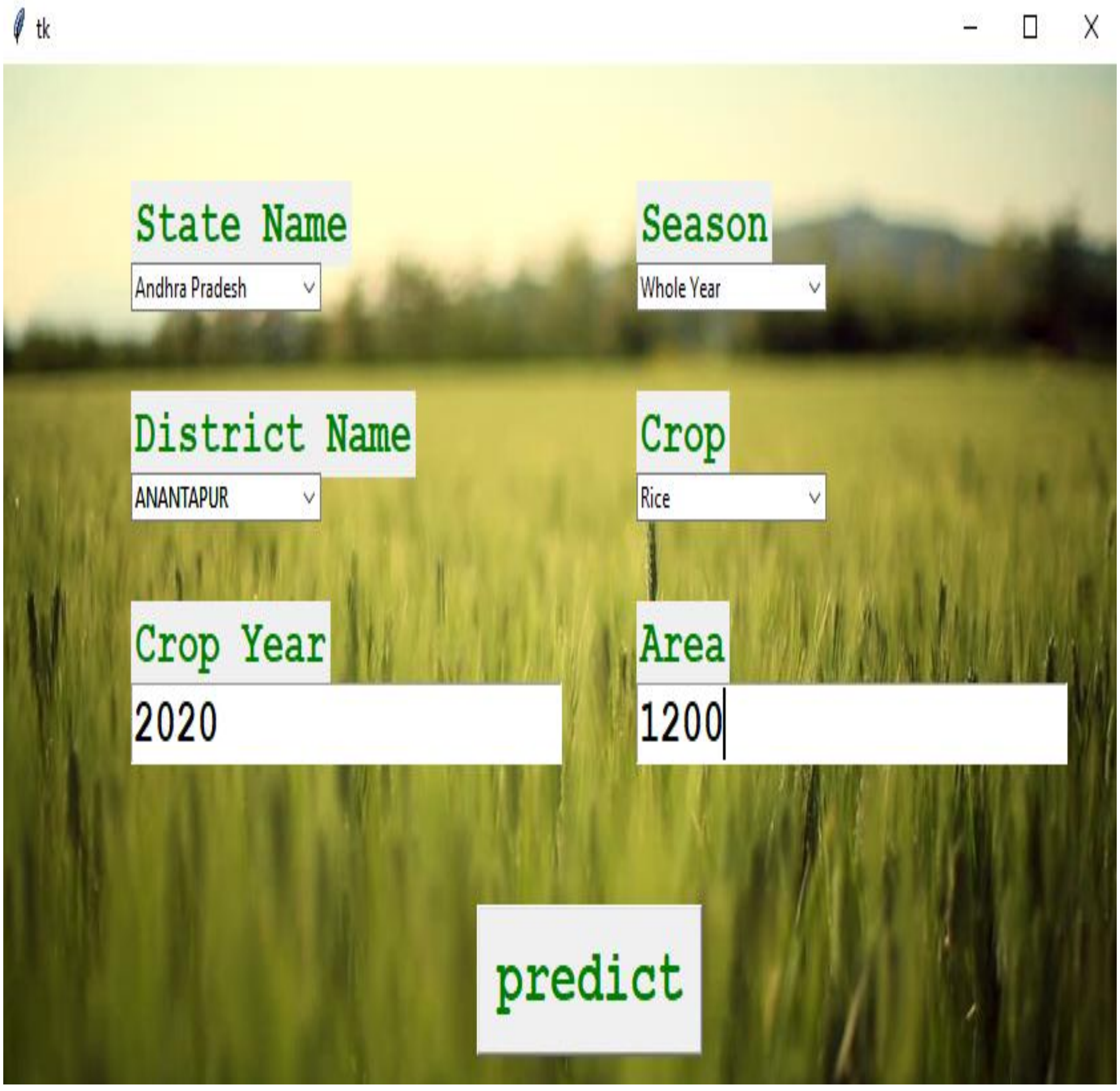


Figure:8.2.1 : **Crop Predict Application Button**

8.2.2 Predict Searching: Here is the next Window we need to select all Above details then we have click predict crop and Area is square feet.


tk

State Name: Andhra Pradesh

Season: Whole Year

District Name: ANANTAPUR

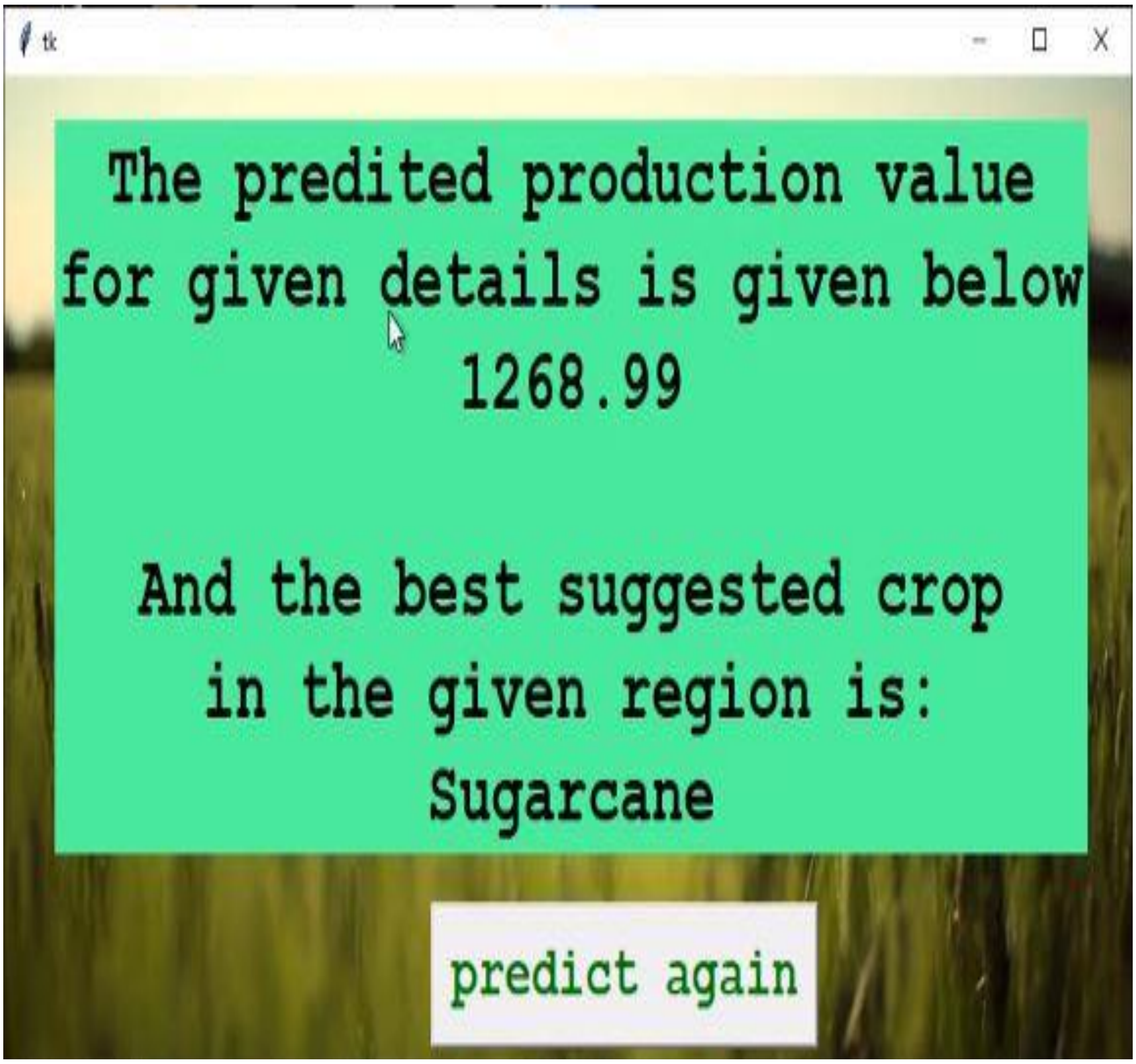
Crop: Rice

Crop Year: 2020

Area: 1200

Figure:8.2 : **Predict Searching**

8.2.3 Output Of Crop Predict : Here is last Window, In this Window predicted crop will be displayed And also suggest the better crop for this region.



tk

The predited production value
for given details is given below
1268.99

And the best suggested crop
in the given region is:
Sugarcane

Figure:8.3 : **Output Of Crop Predict**

CHAPTER-9

CONCLUSION

Crop yield prediction is still remaining as a challenging issue for farmers.The aim of this research is to propose and implement a rule based system to predict the crop yield production from the collection of past data.

The paper concludes that the rapid advances in sensing technologies and ML techniques will provide cost-effective and comprehensive solutions for better crop and environment state estimation and decision making.

We had tried to show that ML models have been applied in multiple applications of agriculture management like crop yield management. This trend in the application distribution reflects the data-intense applications within the crop and high use of Previous year data set images and other data values related to crop yield like rainfall, temp etc. Various ML algorithm like RF, Cubist, soil samples was tested and prediction is done accordingly by various researchers.

It is also evident from the analysis that most of the studies used Random Forest and SVM ML models. More specifically, Random Forest were used mostly for implementations for crop yield management and prediction. This leads the future work for us to take a particular area of the Indian subcontinent and use the available dataset of a region and apply Machine learning and Support Vector Machine to predict the crop yield by selecting multiple features so that results will be more appropriate for crop yield prediction.

CHAPTER-10

FUTURE SCOPE

The future scope of this project is to develop a Machine Learning based system which can predict yield of the crop in advance. Also providing an option to predict the future production based on user inputs

Linear regression techniques can be implemented in future in the various fields like the stock marketing, bitcoin value prediction and the many other varies business, this technique may be applied to multiple regression and logistic regression.the future scope can be very much accurate and error can be reduced and model can be made more robust.

the farmer can take better decision in crop selection. This will also help our country to become independent on food resources.

CHAPTER-11

REFERENCES

1. Grajales, D.F.P., Mosquera, G.J.A, Mejia, F., Piedrahita, L.C., Basurto, C., "Crop-Planning, Making Smarter Agriculture With Climate Data",Fourth International Conference on Agro-GeoInformatics, pp.240-244, 2015.
2. Bendre, M. R., Thool, R.C., Thool, V. R., "Big Data in Precision Agriculture : Weather Forecasting for Future Farming", 1 st Internationa
3. Conference on Next Generation Computing Technologies, pp.744-750, 2015.Hemageetha, N., "A survey on application of data mining techniques to analyze the soil for agricultural purpose", 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp.3112-3117, 2016.