MAJOR PROJECT REPORT

On

Time Series Mining Approach for Agriculture Area Detection using Machine Learning

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Mr. M.Vamshi Krishna	**17J41A05F4**
Mr. Y.Venu Gopal Reddy	**17J41A05J0**
Mr. M.Kumar Aravind	**17J45A05F5**
Mr.B.Ajay Naik	**17J41A05C6**

Under the guidance of,

Dr. B.HARI KRISHNA
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MALLA REDDY ENGINEERING COLLEGE

(An UGC Institution, Approved by AICTE and Affiliated to JNTUH Hyderabad,
Recognized under section 2(f) &12(B) of UGC Act 1956,
Accredited by NAAC with 'A' Grade (II Cycle) and NBA Maisammaguda,
Dhulapally (Post Via Kompally), Secunderabad-500 100

Website: www.mrec.ac.in E-mail: principal@mrec.ac.in

2017-2021

MALLA REDDY ENGINEERING COLLEGE

Maisammaguda, Dhulapally (Post Via Kompally), Secunderabad – 500 100 Telangana State

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work titled " **TIME SERIES MINING APPROACH FOR AGRICULTURE AREA DETECTION USING MACHINE LEARNING** " is a bonafide workdone by **Mr. M.VamshiKrishna(17J41A05F5), Mr. Y.VenuGopal Reddy(17J41A05J0),Mr.M.KumarAravind17J45A05F5),Mr.B.AjayNaik(17J41A05 C6),** in partial fulfillment of the requirements for the award of **Bachelor of Technology** in **Computer Science and Engineering** of the **Malla Reddy Engineering College** affiliated to JNTUH, Hyderabad and that this has not submitted for the award of any other degree of any Institution/University.

Internal Guide
Dr.B.HARI KRISHNA
Associate Professor

Head of the Department
Dr. N. Lakhmipathi Anantha
Professor

External Examiner

DECLARATION

We hereby declare that this project work dissertation titled "**TIME SERIES MINING APPROACH FOR AGRICULTURE AREA DETECTION USING MACHINE LEARNING "** is original and bonafide work of our own in partial fulfillment ofthe requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** at **Malla Reddy Engineering College**, affiliated to **JNTUH, Hyderabad** under the guidance of **Dr.B.Hari Krishna**, **Associate Professor,** Department of CSE and has not been copied from any earlier reports.

ROLL NUMBER	NAME	SIGNATURE
17J41A05F4	Mr.M.Vamshi Krishna	
17J41A05J0	Mr.Y.Venu Gopal Reddy	
17J45A05F5	Mr. M. Kumar Aravind	
17J45A05C6	Mr.B.Ajay Naik	

ACKNOWLEDGEMENT

We are extremely thankful to our beloved Chairman and Founder of Malla Reddy Group of Institutions **Sri.Ch.Malla Reddy,** for providing necessary infrastructure facilities throughout the project work.

We express our sincere thanks to **Director Dr. A. Ramaswamy Reddy**, who took keen interest and encouraged us in every effort during the project work.

We owe our gratitude to **Dr. A. Ravindra**, **Principal**, for his encouragement to accomplish the project work successfully.

We express our heartfelt thanks to **Dr. N. Lakshmipathi Anantha** , **Professor and Head**, Department of Computer Science and Engineering, for his kind attention and valuable guidance throughout the project work.

We are thankful to our Project Coordinator **Dr. Ch GVN Prasad, Professor** of CSE for his valuable suggestions and guidance throughout the project work.

We are extremely thankful to our Project Guide **Mr.B.Hari Krishna,Associate Professor** for his constant guidance and support to complete the project work.

We also thank all the teaching and non-teaching staff of Computer Science and Engineering Department for their cooperation during the project work.

Mr.M.Vamshi Krishna	**17J41A05F4**
Mr.Y.Venu Gopal Reddy	**17J41A05J0**
Mr.M.Kumar Aravind	**17J45A05F5**
Mr.B.Ajay Naik	**17J45A05C6**

ABSTRACT

Acquiring meaningful data to be employed in building training sets for classification models is a costly task, both in terms of difficulty to find suitable samples as well as their quantity. In this sense, Active Learning (AL) improves the training set building by providing an efficient way to select only essential data to be attached to the training set, consequently reducing its size and even enhancing model's accuracy, when compared to random sample selection. In this paper, we proposed a framework for time series classification in order to monitor sugarcane areas in São Paulo, Brazil. The AL approach consisted of selecting seasonal time series information from less than 1% of each class' pixels to build the training set and evaluate this selection by an expert user supported by distance measurements, repeating this process until both distance measurement thresholds were satisfied. In most years, the classification results presented about 90% of correlation with official estimates based on both traditional and satellite image analysis methods. This framework can then help Land Use Change (LUC) monitoring as it produces similar results compared to other methods that demand more human and financial resources to be adopted.

CONTENTS

Chapter-V: SOFTWARE DEVELOPMENT

5.1. Installing Anaconda on windows

5.2. Add Anaconda to Path

5.3 Anaconda conclusion

5.4. What is Jupyter Notebook ?

5.5. How to Install Jupyter Environment ?

5.6. Installing Jupyter Notebook using Anaconda

5.7. Installing Jupyter Notebook using Pip

5.8. How to run or open Jupyter Notebook

5.9. What are the cells in Jupyter Notebook

Chapter-VI: CONCLUSION & REFERENCES

6.1 Conclusion

6.2 References

Chapter-1

INTRODUCTION

1.1 Introduction to Machine Learning:

Machine learning is about extracting knowledge from data. It is a research field at the intersection of statistics, artificial intelligence, and computer science and is also known as predictive analytics or statistical learning. The application of machine learning methods has in recent years become ubiquitous in everyday life. From automatic recommendations of which movies to watch, to what food to order or which products to buy, to personalized online radio and recognizing your friends in your photos, many modern websites and devices have machine learning algorithms at their core. When you look at a complex website like Facebook, Amazon, or Netflix, it is very likely that every part of the site contains multiple machine learning models. Outside of commercial applications, machine learning has had a tremendous influence on the way data-driven research is done today. The tools introduced in this book have been applied to diverse scientific problems such as understanding stars, finding distant planets, discovering new particles, analyzing DNA sequences, and providing personalized cancer treatments. Your application doesn't need to be as large-scale or world-changing as these examples in order to benefit from machine learning, though. In this chapter, we will explain why machine learning has become so popular and discuss what kinds of problems can be solved using machine learning. Then, we will show you how to build your first machine learning model, introducing important concepts along the way.

1.2 Why Machine Learning?

In the early days of "intelligent" applications, many systems used hand coded rules of "if " and "else" decisions to process data or adjust to user input. Think of a spam filter whose job is to move the appropriate incoming email messages to a spam folder. You could make up a blacklist of words that would result in an email being marked as spam. This would be an example of using an expert-designed rule system to design an "intelligent" application. Manually crafting decision rules is feasible for some applications, particularly those in which humans have a good understanding of the process to model. However, using hand coded rules to make decisions has two major disadvantages: • The logic required to make a decision is specific to a single domain and task.

Changing the task even slightly might require a rewrite of the whole system. • Designing rules requires a deep understanding of how a decision should be made by a human expert. One example of where this hand coded approach will fail is in detecting faces in images. Today, every smart phone can detect a face in an image. However, face detection was an unsolved problem until as recently as 2001. The main problem is that the way in which pixels (which make up an image in a computer) are "perceived" by the computer is very different from how humans perceive a face. This difference in representation makes it basically impossible for a human to come up with a good set of rules to describe what constitutes a face in a digital image. Using machine learning, however, simply presenting a program with a large collection of images of faces is enough for an algorithm to determine what characteristics are needed to identify a face.

1.3 Problem Machine Learning can Solve

The most successful kinds of machine learning algorithms are those that automate decision-making processes by generalizing from known examples. In this setting, which is known as supervised learning, the user provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired out- put given an input. In particular, the algorithm is able to create an output for an input it has never seen before without any help from a human. Going back to our example of spam classification, using machine learning, the user provides the algorithm with a large number of emails (which are the input), together with information about whether any of these emails are spam (which is the desired output). Given a new email, the algorithm will then produce a prediction as to whether the new email is spam. Machine learning algorithms that learn from input/output pairs are called supervised learning algorithms because a "teacher" provides supervision to the algorithms in the form of the desired outputs for each example that they learn from. While creating a dataset of inputs and outputs is often a laborious manual process, supervised learning algorithms are well understood and their performance is easy to measure.

If your application can be formulated as a supervised learning problem, and you are able to create a dataset that includes the desired outcome, machine learning will likely be able to solve your problem. Examples of supervised machine learning tasks include: Identifying the zip code

from handwritten digits on an envelope Here the input is a scan of the handwriting, and the desired output is the actual digits in the zip code. To create a dataset for building a machine learning model,

you need to collect many envelopes. Then you can read the zip codes yourself and store the digits as your desired outcomes. Determining whether a tumor is benign based on a medical image Here the input is the image, and the output is whether the tumor is benign. To create a dataset for building a model, you need a database of medical images. You also need an expert opinion, so a doctor needs to look at all of the images and decide which tumors are benign and which are not. It might even be necessary to do additional diagnosis beyond the content of the image to determine whether the tumor in the image is cancerous or not. Detecting fraudulent activity in credit card transactions Here the input is a record of the credit card transaction, and the output is whether it is likely to be fraudulent or not. Assuming that you are the entity distributing the credit cards, collecting a dataset means storing all transactions and recording if a user reports any transaction as fraudulent.

An interesting thing to note about these examples is that although the inputs and outputs look fairly straightforward, the data collection process for these three tasks is vastly different. While reading envelopes is laborious, it is easy and cheap. Obtaining medical imaging and diagnoses, on the other hand, requires not only expensive machinery but also rare and expensive expert knowledge, not to mention the ethical concerns and privacy issues. In the example of detecting credit card fraud, data collection is much simpler. Your customers will provide you with the desired output, as they will report fraud. All you have to do to obtain the input/output pairs of fraudulent and non fraudulent activity is waiting. Unsupervised algorithms are the other type of algorithm that we will cover in this book. In unsupervised learning, only the input data is known, and no known output data is given to the algorithm. While there are many successful applications of these methods, they are usually harder to understand and evaluate. Examples of unsupervised learning include: Identifying topics in a set of blog posts If you have a large collection of text data, you might want to summarize it and find prevalent themes in it. You might not know beforehand what these topics are, or how many topics there might be. Therefore, there are no known outputs segmenting customers into groups with similar preferences given a set of customer records, you

might want to identify which customers are similar, and whether there are groups of customers with similar preferences.

For a shopping site, these might be “parents,” “bookworms,” or “gamers.” Because you don’t know in advance what these groups might be, or even how many there are, you have no

known outputs. Detecting abnormal access patterns to a website To identify abuse or bugs, it is often helpful to find access patterns that are different from the norm. Each abnormal pattern might be very different, and you might not have any recorded instances of abnormal behavior. Because in this example you only observe traffic, and you don’t know what constitutes normal and abnormal behavior, this is an unsupervised problem. For both supervised and unsupervised learning tasks, it is important to have a representation of your input data that a computer can understand. Often it is helpful to think of your data as a table.

Each data point that you want to reason about (each email, each customer, each transaction) is a row, and each property that describes that data point (say, the age of a customer or the amount or location of a transaction) is a column. You might describe users by their age, their gender, when they created an account, and how often they have bought from your online shop. You might describe the image of a tumor by the grayscale values of each pixel, or maybe by using the size, shape, and color of the tumor. Each entity or row here is known as a sample (or data point) in machine learning, while the columns—the properties that describe these entities—are called features. Later in this book we will go into more detail on the topic of building a good representation of your data, which is called feature extraction or feature engineering. You should keep in mind, however, that no machine learning algorithm will be able to make a prediction on data for which it has no information. For example, if the only feature that you have for a patient is their last name, no algorithm will be able to predict their gender. This information is simply not contained in your data. If you add another feature that contains the patient’s first name, you will have much better luck, as it is often possible to tell the gender by a person’s first name.

1.4 Knowing your Task and Knowing Your Data

Quite possibly the most important part in the machine learning process is under- standing the data you are working with and how it relates to the task you want to solve. It will not be effective to randomly choose an algorithm and throw your data at it. It is necessary to understand what is going on in your dataset before you begin building a model. Each algorithm is different in terms of what kind of data and what problem setting it works best for. While you are building a machine learning solution, you should answer, or at least keep in mind, the following questions:

What question(s) am I trying to answer?

Do I think the data collected can answer that question?

What is the best way to phrase my question(s) as a machine learning problem?

Have I collected enough data to represent the problem I want to solve?

What features of the data did I extract, and will these enable the right predictions?

How will I measure success in my application?

How will the machine learning solution interact with other parts of my research or business product?

In a larger context, the algorithms and methods in machine learning are only one part of a greater process to solve a particular problem, and it is good to keep the big picture in mind at all times. Many people spend a lot of time building complex machine learning solutions, only to find out they don't solve the right problem. When going deep into the technical aspects of machine learning (as we will in this book), it is easy to lose sight of the ultimate goals. While we will not discuss the questions listed here in detail, we still encourage you to keep in mind all the assumptions that you might be making, explicitly or implicitly, when you start building machine learning models.

1.5 What is Computer Vision?

Computer vision is simply the process of perceiving the images and videos available in the digital formats. In Machine Learning (ML) and AI – Computer vision is used to train the model to recognize certain patterns and store the data into their artificial memory to utilize the same for predicting the results in real-life use.

The main purpose of using computer vision technology in ML and AI is to create a model that can work itself without human intervention. The whole process involves methods of acquiring the data, processing, analyzing, and understanding the digital images to utilize the same in the real-world scenario.

1.6 How Computer Vision System Works?

You can say computer vision is used for deep learning to analyze the different types of data sets through annotated images showing the object of interest in an image. It can recognize the patterns to understand the visual data feeding thousands or millions of images that have been labeled for supervised machine learning algorithms training.

This process depends subject to use of various software techniques and algorithms, that are allowing the computers to recognize the patterns in all the elements that relate to those labels and make the model predictions accurately in the future. Computer vision can be only utilized only with image processing through machine learning.

1.7 What is difference between Computer Vision and Image Processing?

Both are the part of AI technology used while processing the data and creating a model. The **difference between computer vision and image processing** in Computer vision helps to gain high-level of understanding from images or videos.

For instance, object recognition, which is the process of identifying the type of objects in an image, is a computer vision problem. In computer vision, you receive an image as input, and you can produce an image as output or some other type of information.

Whereas, image processing doesn't need such a high level of understanding of image. In fact, it is the sub-field of signal processing but also applied to images. For example, if you have

noisy or blurred images, then under image processing the deblurring or denoising is done to make the object in the image clearly visible to machines.

Image process task involves filtering, noise removal, edge detection, and color processing. In entire processing, you receive an image as input and produce another image as an output that can be used to train the machine through computer vision.

The main difference between computer vision and image processing are the goals (not the methods used). For example, if the goal is to enhance the image quality for later use, which is called image processing. If the goal is to visualize like humans, like object recognition, defect detection or automatic driving, then it is called computer vision.

Chapter-2

PROECT DESCRIPTION

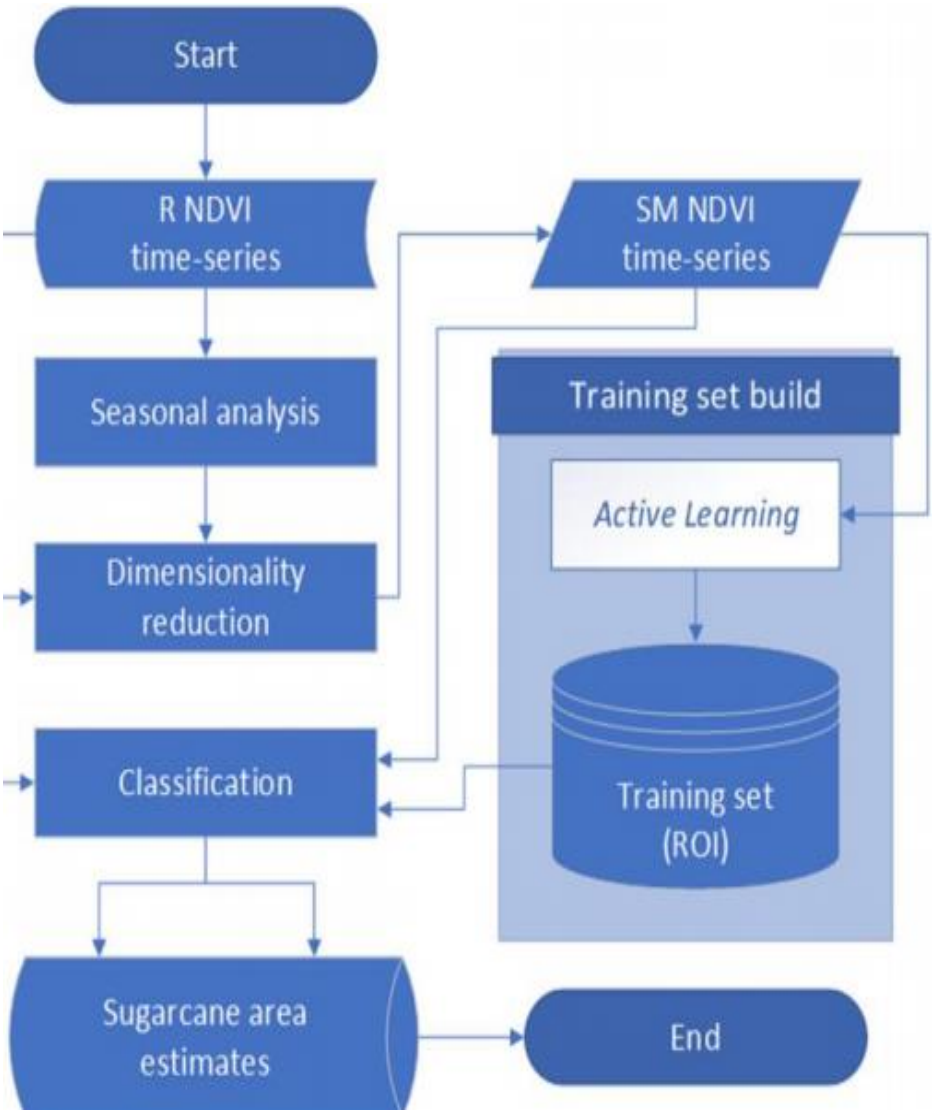
2.1 Abstract

Acquiring meaningful data to be employed in building training sets for classification models is a costly task, both in terms of difficulty to find suitable samples as well as their quantity. In this sense, Active Learning (AL) improves the training set building by providing an efficient way to select only essential data to be attached to the training set, consequently reducing its size and even

enhancing model's accuracy, when compared to random sample selection. In this paper, we proposed a framework for time series classification in order to monitor sugarcane areas in São Paulo, Brazil. The AL approach consisted of selecting seasonal time series information from less than 1% of each class' pixels to build the training set and evaluate this selection by an expert user supported by distance measurements, repeating this process until both distance measurement thresholds were satisfied. In most years, the classification results presented about 90% of correlation with official estimates based on both traditional and satellite image analysis methods. This framework can then help Land Use Change (LUC) monitoring as it produces similar results compared to other methods that demand more human and financial resources to be adopted.

Index Terms -Environment, Pixel classification, Remote sensing, Time series analysis

2.2 Block Diagram



2.3 Project Introduction

Agriculture is the authority of India. Only one-third of cropped part is only inundated in India, in spite of large areas. Since the agriculture data occurred everyday the capacity of data has been enlarged rapidly mostly on last five years. Farmers, researchers, government and agricultural scientists are still searching and extracting for fresh techniques for farming to increase the better production. At present new methods are present in agriculture are used by a very few farmers. For predicting future trends of agriculture processes "data mining" can be used. The process of examining data by summarizing in different perspective and converting it into an beneficial information in large datasets is called Data mining. Data mining has no restriction for analyzing the type of data.

Data Mining In Agriculture

In large data sets, data mining is the computational process for discovering new patterns. Data mining provides major advantage in agriculture for disease detection, problem prediction and for optimizing the pesticides. In recent technologies agriculture related activities provide lot of information. Hence this data mining techniques in agriculture are used for pattern reorganization and disease detection. Data's of agriculture in data mining can be presented in form of data marts. Crop production for reliable and timely requirement for various decisions for marketing, pricing, storage distribution and import-export. The yield of agriculture primarily depends on diseases, pests, climatic conditions, planning of different crops for the harvest productivity are the results. So by these predictions are very useful for agriculture domains. Data mining techniques are used for pre-harvest forecasting. For example by applying data mining technique government can fully benefit data about farmers buying patterns and also to gain a superior understanding of their land to protect them in order to gain more profit on farmer's part.

Data mining is also called as knowledge discovery database (KDD). Data mining tasks can be classified into two categories: Descriptive data mining.¬ Predictive data mining.¬ Descriptive data mining tasks characterize the general properties of the data in the database while predictive data mining is used to predict the direct values based on patterns determined from known results. Prediction involves using some variables or fields in the database to predict unknown or future values of other variables of interest. As far as data mining technique is concern, in the most of cases predictive data mining approach is used. Predictive data mining technique is used to predict

future crop, weather forecasting, pesticides and fertilizers to be used, revenue to be generated and so on.

Importance of Data Mining: Data mining is the major technique for collection of data's in various forms among the data collected in the process of data mining includes research data, survey data, organization data, competitive data and social media such as whatsapp, Facebook. Several steps are involved in analyzes on selected set of data where the process involves of filtering, transformation, testing, modelling, visualization and documentation is prepared and the result is outputted (or) the data is stored accordingly in data warehouse or databases. To propose a smart agriculture we must predict the yield of crop based on the water, texture of soil and climate. It is essential for our country to build a large production of organic crops. So by applying data mining techniques for agriculture we can reduce the cost of food production and improves productivity which encounters in greater decision making process in business world. i.e. agriculture.

Five Major Elements in Data Mining:

Fetch the data and load the data to transform onto the warehouse system. Store and use the data in the database system.

Make available to access data for researchers,

IT professionals and for various organizational analytics.

Examine the required data using suitable software's. Formulate the data's inform of table or graph to represent data in an useful format.

2.4 Literature Review

Zhong Yinetal [1] presented a novel mental workload (MWL) detection framework based on a combination of unsupervised and supervised learning strategies to improve the prediction accuracy of mental workload. The use of EEG recording for the data acquisition causes the problems of high dimensionality of the candidate EEG features vectors and also reduces the ability to determine the MWL variations and the target class labels. In the presented approach, the locally linear embed-ding (LLE), support vector clustering (SVC) and support vector data description (SVDD) techniques are combined to overcome the problems of using EEG recording. The LLE

technique is employed to find the low-dimensional MWL features in the high-dimensional EEG feature space in order to extract the representative EEG markers from different cortical regions. Then the SVC-SVDD hybrid framework is employed in which the SVC technique is used to find the data clusters in EEG data space and SVDD technique is used to distinguish the blurred and overlapped cluster into two classes. Thus the presented approach improves the prediction accuracy in the three class MWL temporal data classification. Still the time delays effects, non-automatic execution of SVDD and the real-time MWL assessment problems reduce the efficiency of the approach.

Ranganatha Sitarametal [2] presented an approach for determining the feasibility of using a multi-channel Near-infrared spectroscopy (NIRS) in the development of brain–computer interface (BCI) system. The presented approach employs temporal classification of the multi-channel NIRS signals of the motor images to improve the prediction accuracy. Initially the signal acquisition is performed and the signals are analyzed to test the presence of significant patterns in the hemodynamic response to motor imagery. Then the classification of the NIRS signals is performed offline using two pattern recognition techniques, Support Vector Machines (SVM) and Hidden Markov Model (HMM). Thus the classification problem can be resolved and the development of NIRS-BCI system can be visualized. The major drawback is the slow process of the long time constants of the hemodynamic response making NIRS–BCI system. The inability to cope with the fast NIRS signal is also a major concern.

Mohamed F Ghalwashetal [3] proposed a new early detection method called Multivariate Shapelets Detection (MSD) for early and patient-specific classification of multivariate time series clinical data. The presented approach extends the concept of univariate shapelets to multivariate shapelets to improve the prediction accuracy. The approach utilized the information gain-based distance threshold and the weighted information-gain based utility score of a shapelet to incorporate the earliness and assigns high utility score to the shapelet to improve the early detection of disease pattern change. Thus the approach can improve the early classification of the multivariate time series data. The drawback in the presented approach is that all the multivariate time series shapelets have the same starting positions which cannot be possible at all situations due to the increasing number of shapelets.

Iyad Bataletal [4] suggested an approach called the minimal predictive temporal patterns (MPTP) framework by integrating classification and pattern mining techniques for classifying Multivariate temporal data to predict the patients with disease developing risks. The presented framework resolves the problem of excessive irrelevant pattern generation caused in temporal pattern mining techniques. The electronic health records consisting of multivariate time series data are collected from the patients and the temporal domain is incorporated to determine the temporal patterns by the temporal abstractions and temporal logic to construct the classification features. The MPTP framework, which effectively filters the non-predictive and spurious temporal patterns, is utilized to automatically mine the predicted temporal patterns by integrating the pattern selection and frequent pattern mining.

Rainer Schmidtetal [5] presented a prognostic model for temporal courses for early detection of the disease risks using the multiple time series data. The prognostic model was presented by combining the temporal abstractions with case-based reasoning (CBR) for efficiently classifying the time series data. The Temporal courses are characterized by domain-dependent trend descriptions to detect the early risks of kidney courses and the influenza diseases. The prognostic model maintains the functioning details of the normal kidney courses and compares it with the current functioning of the kidney to find out the dissimilarities for detecting the abnormal functioning. Thus the multiple time series data can be utilized for effective and early detection of diseases using the prognostic model.

Riccardo Bellazzietal [6] presented a temporal mining approach for the assessment of the clinical performance of hemodialysis (HD) services based on the automatically collected time series data. The presented approach uses two new methods for association rule discovery and temporal rule discovery are applied to the time series for executing the pre-processing techniques such as data reduction, multi-scale filtering and temporal abstractions. Initially, the time series data are represented using the temporal abstractions. Then multi-scale filtering methods are utilized to preprocess the median time series data. Then the association between the temporal abstractions and the dialysis outcomes are searched using a search algorithm based on the association rules and finally the temporal rules are utilized to classify the patterns. Thus the dialysis services can be assessed effectively. But the slow processing of the time series data is a major drawback.

2.5 Existing Method

Different techniques were proposed for mining data over the years and the most used general Data Mining techniques in the field of agriculture are The Artificial Neural Networks but do not appear among the aforementioned Data Mining techniques because there are few applications of this technique in agriculture. By using the Multilayer Perceptron model of Neural Networks the researchers trained to predict wheat yield by considering sensor input and fertilizers as parameters. MLPs were used successfully in the work by the researchers Two different neural networks are one network with a Multilayered Perceptron, another one with a Radial Basis Function, as well as a Support Vector Regression and a Decision Regression Tree. The comparison of these four techniques showed that the Support Vector Regression technique is the most suitable for this kind of problem. Recently spatial AutoCorrelation has improved the quality of the prediction.

2.6 Proposed Method

In this paper it proposed a framework that uses AL, an iterative resembling approach based on query strategies with the training set building process so that it can result in a 90% smaller set without accuracy loss In supervised classification, which depends on the quality of referenced (training) data to reach good results that uses AL to optimize the training set based on class labels transferred from a domain (source) to n target images, such that it could reach up to 90% overall accuracy on image classification This study was divided into steps, as illustrated in Fig 1. First, time series were organized and divided into two sets representing the same area and period one submitted through seasonal modeling and another one used to test its effectiveness. Then, both seasonally modeled (SM) and raw (R) time series were submitted through dimensionality reduction to reduce its size and eliminate remaining noise that could harm final classification. Using the AL approach, we built the training set by sampling regions of interest (ROIs), or pixels, from SM time series images and used it to train classifiers for both SM and R time series. Finally, we compared sugarcane area estimates for every harvest cycle

Chapter- 3

REQUIREMENTS

3.1 Hardware Requirements

- **System** : Intel i3 2.4 GHz.
- **Hard Disk** : 100 GB above.
- **Monitor** : 14 Colour Monitor or any size.
- **Mouse** : Optical Mouse.
- **Ram** : 2 GB.

3.2 Software Requirements

- **Operating system** : Windows 10 (any windows operating system).
- **Coding Language** : Python.
- **Front-End** : Python
- **Designing** : tkinter module.
- **Data Base** : sqlite3, Mysqlclient.
- **IDE** : Anaconda IDE – Jupyter Note Book

Chapter-4

SOURCE CODE , OUTPUT RESULTS AND EXECUTION

4.1 Source Code & Results , Execution

SocialCops Data Science Intern Challenge
Challenge: Agriculture Commodities, Prices & Seasons

Aim: Your team is working on building a variety of insight packs to measure key trends in the Agriculture sector in India. You are presented with a data set around Agriculture and your aim is to understand trends in APMC (Agricultural produce market committee)/mandi price & quantity arrival data for different commodities in Maharashtra.

Objectives:

- Test and filter outliers.
- Understand price fluctuations accounting the seasonal effect
    - Detect seasonality type (multiplicative or additive) for each cluster of APMC and commodities
    - De-seasonalise prices for each commodity and APMC according to the detected seasonality type
- Compare prices in APMC/Mandi with MSP(Minimum Support Price)- raw and deseasonalised
- Flag set of APMC/mandis and commodities with highest price fluctuation across different commodities in each relevant season, and year.

Dataset Description

The agriculture data contains two files namely:

* 'CMO_MSP_Mandi.csv' that contains Min. Support Prices (MSPs) and Crop Type for each commodity.

* 'Monthly_data_cmo.csv' contains the APMC-wise monthly prices (min., max. and average) for each and every commodity.

Variable description:

- MSPrice- Minimum Support Price
- arrivals_in_qtl- Quantity arrival in market (in quintal)
- min_price- Minimum price charged per quintal
- max_price- Maximum price charged per quintal
- modal_price- Mode (Average) price charged per quintal

Section 1: Exploratory Data Analysis

In [26]:
```
# Importing necessary modules.

import datetime
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [71]:
```
# Reading the dataset.

date_parse = lambda dates: pd.datetime.strptime(dates, '%Y')
df_mandi = pd.read_csv("C:\\Users\\mvk\\Documents\\C5\\Mandi_Data\\CMO_MSP_Mandi.csv", parse_dates=['year'], index_col='year', date_parser=date_parse)

print('Length of dataset: ', len(df_mandi))
print('Crops types:', set(df_mandi['Type']))
#df_mandi.loc[df_mandi['commodity'] == 'BAJRI']
```

```
Length of dataset:  155
Crops types: {'Kharif Crops', 'Other Crops', 'Rabi Crops'}
C:\Users\mvk\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: The pandas.datetime class is deprecated and will be removed from pandas
in a future version. Import from datetime module instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [72]:
```
# Checking missing data values.

df_mandi.isna().sum()
```

Out[72]:
```
commodity      0
Type           0
msprice       10
msp_filter     0
dtype: int64
```

The above missing values count demonstrates that MSP Prices are missing for 10 commodities in the Mandi dataset.

In [73]:
```
# Reading the dataset as a timeseries data.

date_parse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
df_monthly = pd.read_csv('./Mandi_Data/Monthly_data_cmo.csv', parse_dates=['date'], index_col='date', date_parser=date_parse)

print('Length of dataset:', len(df_monthly), '\n')
#print(type(df_monthly['date']))
print(df_monthly.dtypes)

print('\nSet of unique district names:', set(df_monthly['district_name']))

df_monthly.head()
```

```
C:\Users\mvk\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: The pandas.datetime class is deprecated and will be removed from pandas
in a future version. Import from datetime module instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
Length of dataset: 62429

APMC               object
Commodity          object
Year                int64
Month              object
arrivals_in_qtl     int64
min_price           int64
max_price           int64
modal_price         int64
district_name      object
state_name         object
dtype: object

Set of unique district names: {'Kolhapur', 'Thane', 'Nandurbar', 'Gadchiroli', 'Satara', 'Amaravathi', 'Wasim', 'Hingoli', 'Latur', 'Nasik', 'Akola', 'D
hule', 'Ratnagiri', 'Aurangabad', 'Bhandara', 'Pune', 'Mumbai', 'Ahmadnagar', 'Jalgaon', 'Gondiya', 'Sangli', 'Chandrapur', 'Solapur', 'Jalna', 'Osmanab
ad', 'Raigad', 'Wardha', 'Buldhana', 'Beed', 'Yewatmal', 'Nagpur', 'Parbhani', 'Nanded'}
```

Out[73]:

date	APMC	Commodity	Year	Month	arrivals_in_qtl	min_price	max_price	modal_price	district_name	state_name
2015-04-01	Ahmednagar	Bajri	2015	April	79	1406	1538	1463	Ahmadnagar	Maharashtra
2016-04-01	Ahmednagar	Bajri	2016	April	106	1788	1925	1875	Ahmadnagar	Maharashtra
2015-04-01	Ahmednagar	Wheat(Husked)	2015	April	1253	1572	1890	1731	Ahmadnagar	Maharashtra
2016-04-01	Ahmednagar	Wheat(Husked)	2016	April	387	1750	2220	1999	Ahmadnagar	Maharashtra
2015-04-01	Ahmednagar	Sorgum(Jawar)	2015	April	3825	1600	2200	1900	Ahmadnagar	Maharashtra

In [74]:
```
# Checking missing values.
print(df_monthly.isna().sum())
```

```
APMC              0
Commodity         0
Year              0
Month             0
arrivals_in_qtl   0
min_price         0
max_price         0
modal_price       0
district_name     0
state_name        0
dtype: int64
```

The above counts are all zeros. It demonstrates that there are no missing values in the monthly dataset for the commodities.

In [75]:
```
# Index values for df_monthly dataset.
df_monthly.index
```

Out[75]:
```
DatetimeIndex(['2015-04-01', '2016-04-01', '2015-04-01', '2016-04-01',
               '2015-04-01', '2016-04-01', '2015-04-01', '2016-04-01',
               '2015-04-01', '2016-04-01',
               ...
               '2016-11-01', '2016-11-01', '2016-11-01', '2016-11-01',
               '2016-11-01', '2016-11-01', '2016-11-01', '2016-11-01',
               '2016-11-01', '2016-11-01'],
              dtype='datetime64[ns]', name='date', length=62429, freq=None)
```

In [76]:
```
print('Total number of Commodities:', len(set(df_monthly['Commodity'])))
print('Total number of APMCs:', len(set(df_monthly['APMC'])))

df_monthly.loc[(df_monthly['Commodity'] == 'Onion') & (df_monthly['APMC'] == 'Ahmednagar')]
df_monthly.head()
```

```
Total number of Commodities: 352
Total number of APMCs: 349
```

Out[76]:

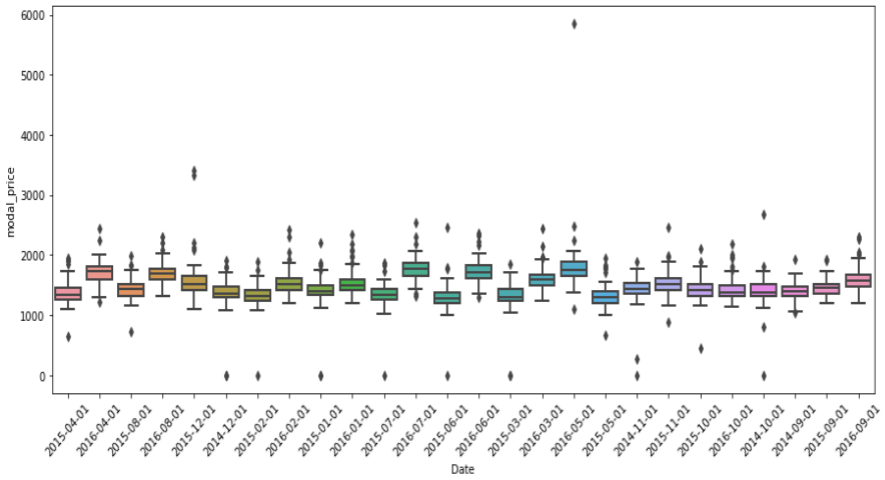
date	APMC	Commodity	Year	Month	arrivals_in_qtl	min_price	max_price	modal_price	district_name	state_name
2015-04-01	Ahmednagar	Bajri	2015	April	79	1406	1538	1463	Ahmadnagar	Maharashtra
2016-04-01	Ahmednagar	Bajri	2016	April	106	1788	1925	1875	Ahmadnagar	Maharashtra
2015-04-01	Ahmednagar	Wheat(Husked)	2015	April	1253	1572	1890	1731	Ahmadnagar	Maharashtra
2016-04-01	Ahmednagar	Wheat(Husked)	2016	April	387	1750	2220	1999	Ahmadnagar	Maharashtra
2015-04-01	Ahmednagar	Sorgum(Jawar)	2015	April	3825	1600	2200	1900	Ahmadnagar	Maharashtra

Section 2: Visualizations & Outlier Detection

In [77]:
```
# Method to plot the boxplot for prices of the commodity and detect the outliers.

def plot_boxplot(df_commodity):
    plt.figure(figsize=(15, 6))
    ax = sns.boxplot(x=df_commodity.index.date, y=df_commodity['modal_price'], linewidth=2)
    #ax = sns.boxplot(x=df_commodity.index, y=df_commodity['max_price'], linewidth=2)
    #ax = sns.boxplot(x=df_commodity.index, y=df_commodity['min_price'], linewidth=2)

    plt.xticks(rotation=45)
    plt.xlabel('Date')
    plt.show()
```

In [78]:
```
df_bajri = df_monthly.loc[df_monthly['Commodity'] == 'Bajri']
plot_boxplot(df_bajri)
```

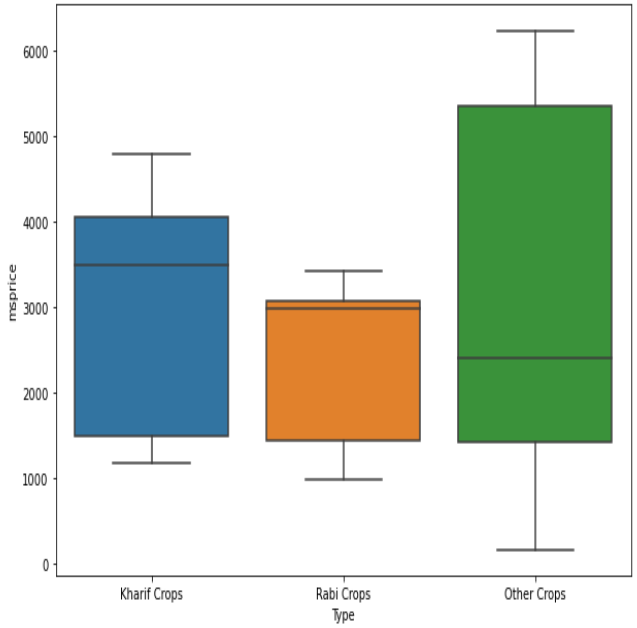
In [78]:
```
df_bajri = df_monthly.loc[df_monthly['Commodity'] == 'Bajri']
plot_boxplot(df_bajri)
```



The above **Box Plot** is for 'Bajri' commodity. It represents the average monthly prices for years: 2014, 2015 and 2016. The values in the rectangular box represent 1st, 2nd and 3rd Quartiles from bottom to up for every timestamp. The dark points away from the whiskers are outliers. For ex- In Mar 2016, the average price value (per quintal) is close to Rs. 6000 which is clearly an outlier.

Box Plot provides a very good visualization to find out the presence of outliers in the dataset.

In [79]:
```
plt.figure(figsize=(10, 7))
ax = sns.boxplot(x='Type', y='msprice', data=df_mandi)
```



The above box plot for MSP (Min. Support Price) for various commodities evidents that there are no points outside the whiskers and hence, **no outliers** in the dataset: 'CMO_MSP_Mandi'.

Section 3: Understanding price fluctuations accounting the seasonal effect

Methodology:

Since there are total **352 commodities** and **349 APMCs** in the dataset, we have taken a sample of 3 APMCs: Ahmednagar, Rahata and Kamkhed and again, sample of 3 commodities: Bajri, Onion and Capsicum. But all the methods are written in a generic way so that the analysis can be easily extended to more number of APMCs and commodities just by calling these methods.

For each pair of commodity and APMC, we have done the following:

- Plot the monthly line graph for minimum, maximum and average prices of the commodity belonging to a given APMC to analyse the seasonality in the monthly prices.
- **Stationarity Test:** A time series is said to be stationary if its **statistical properties** such as mean and variance remain **constant over time**.
    - **Dickey Fuler Test:** It is one of the statistical tests for checking stationarity. Here, the null hypothesis is that the TS is non-stationary and alternate hypothesis is that the TS is stationary. The test results comprise of a Test Statistic and some Critical Values for difference confidence levels. If the **'Test Statistic' is less than the 'Critical Value'**, we can reject the null hypothesis and say that the time series is stationary.
- De-seasonalizing the time series (making the TS stationary):
    - There are 2 major reasons behind non-stationarity of a TS:
        1. **Trend** – varying mean over time.
        2. **Seasonality** – variations at specific time-frames.
    - So, we need to remove both trends and seasonality to make TS stationary.
- Removing the seasonality from the time series:
    - **Differencing with lag 1:** One of the most common methods of dealing with both trend and seasonality is differencing. In this technique, we take the difference of the observation at a particular instant with that at the previous instant.
    - **STR Decomposition:** STR refers to Seasonality, Trend and Residual. In this approach, trend and seasonality are modeled separately from TS data and residual part of the series becomes stationary.
- Comparison of APMC-wise commodity prices (max., average and min.) with MSPs: Line plots are plotted to clearly demonstrate how MSPs deviate largely from the average prices.

APMCs and their Commodities

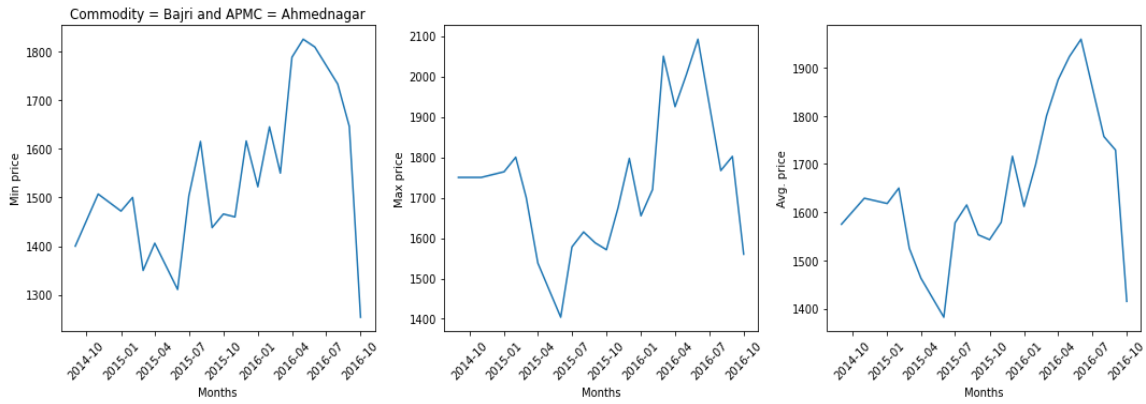
In [80]:

```
# Method to plot prices (max, min and average prices) corresponding to the given commodity and APMC.

def plot_prices(commodity, apmc):
    global df_monthly
    plt.figure(figsize=(15,5))

    plt.subplot(1, 3, 1)
    plt.plot(df_monthly.loc[(df_monthly['Commodity'] == commodity) & (df_monthly['APMC'] == apmc)]['min_price'])
    plt.ylabel('Min price')
    plt.xlabel('Months')
    plt.xticks(rotation=45)
    plt.title('Commodity = %s and APMC = %s' % (commodity, apmc))

    plt.subplot(1, 3, 2)
    plt.plot(df_monthly.loc[(df_monthly['Commodity'] == commodity) & (df_monthly['APMC'] == apmc)]['max_price'])
    plt.ylabel('Max price')
    plt.xlabel('Months')
    plt.xticks(rotation=45)

    plt.subplot(1, 3, 3)
    plt.plot(df_monthly.loc[(df_monthly['Commodity'] == commodity) & (df_monthly['APMC'] == apmc)]['modal_price'])
    plt.ylabel('Avg. price')
    plt.xlabel('Months')
    plt.xticks(rotation=45)

    plt.tight_layout()
```

APMC: Ahmednagar and Commodities: Bajri, Onion and Capsicum

In [81]:

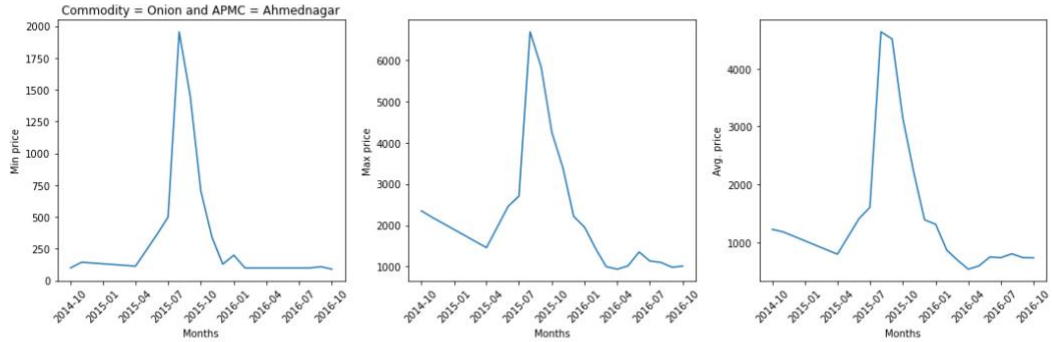
```
# Sort by index values (i.e. by dates).
df_monthly.sort_index(axis=0, inplace=True)

# Plot prices for different commodities in 'Ahmednagar' APMC.
plot_prices('Bajri', 'Ahmednagar')
```



The above plots clearly demonstrate rapid variations at specific time-frames. It is evident that the seasonality is yearly. However, it shows very mixed kind of seasonality (additive + multiplication).

In [82]:

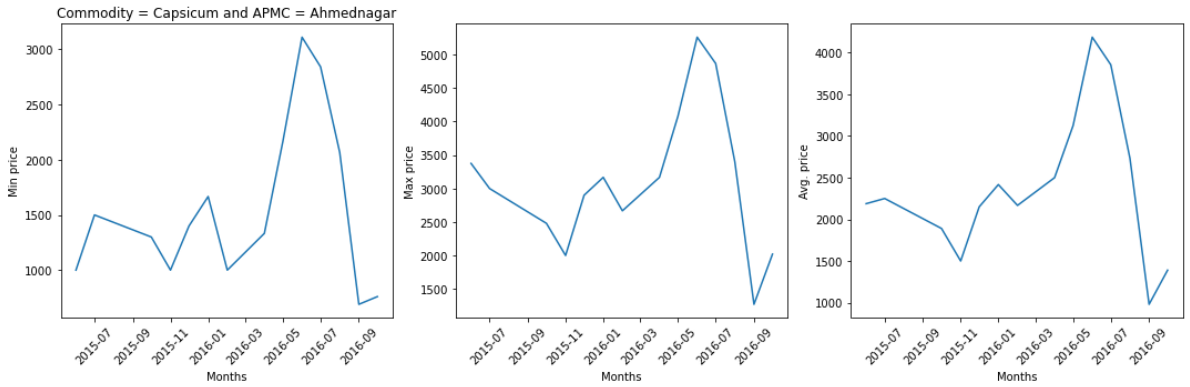
```
plot_prices('Onion', 'Ahmednagar')
```



The above plots clearly demonstrate a peak-shaped curve (surge) with peak around September' 2015 and rapid variations at specific time-frames. It is evident that the seasonality is yearly.

Reason: The main reason for this surge is **decreased supply**. Back in 2015, due to considerable **lower production of Onions**, supply had largely dropped. Consequently, the retail prices jumped.

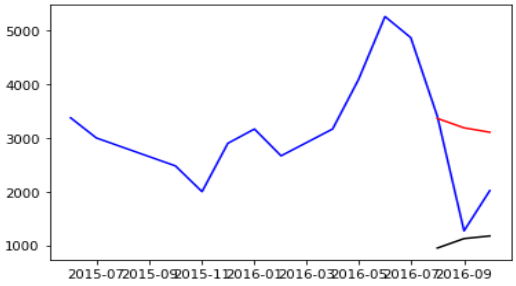
In [83]:
```
plot_prices('Capsicum', 'Ahmednagar')
```



The above plots clearly demonstrate rapid variations at specific time-frames. It is evident that the seasonality is yearly and seasonality type is **multiplicative**.

In [112…
```
def test_stationarity(timeseries):

    # Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()
    orig = plt.plot(timeseries, color='blue',label="Original")
    mean = plt.plot(rolmean, color='red',label='Rolling Mean')
    std = plt.plot(rolstd, color='black',label = 'Rolling Std')
 # Method to test stationarity of the time series data using 'Dickey-Fuller Test'.
# Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
        print(value)

# Testing stationarity for APMC: Ahmednagar and Commodity: Bajri
test_stationarity(df_monthly.loc[(df_monthly['Commodity']=='Capsicum') & (df_monthly['APMC']=='Ahmednagar')]['max_price'])
```

```
Results of Dickey-Fuller Test:
-4.137829282407408
-3.1549724074074077
-2.7144769444444443
```



Analysis: For the commodity 'Bajri' and APMC 'Ahmednagar', we applied 'Dickey Fuller Test' to test stationarity of the time series data for 'Modal (Average) price', 'Max price' and 'Min price' of the commodity. The full description of this test is above just under Section 3.

As **Test Stats < Critical Values**, we reject the Null Hypothesis and accept the Alternate Hypothesis to say that the time series is stationary for 'Avg' and 'Min' prices. Whereas for 'Max' prices, **Test Stats > Critical Values** (-1.05 >-2.77) and hence, we accept Null Hypothesis to say that the time series is not stationary. Here, the seasonality type is Multiplicative.

Eliminating Seasonality using Differencing method

In [17]:

```
# Method to eliminate seasonality from the time series data via 'Differencing' method with lag 1.

def eliminate_seasonality(ts_log):
    ts_log_diff = ts_log - ts_log.shift()
    #plt.plot(ts_log_diff)

    ts_log_diff.dropna(inplace=True)
    test_stationarity(ts_log_diff)
    return ts_log_diff
```

Using Decomposition to separate out trend, seasonality and residual

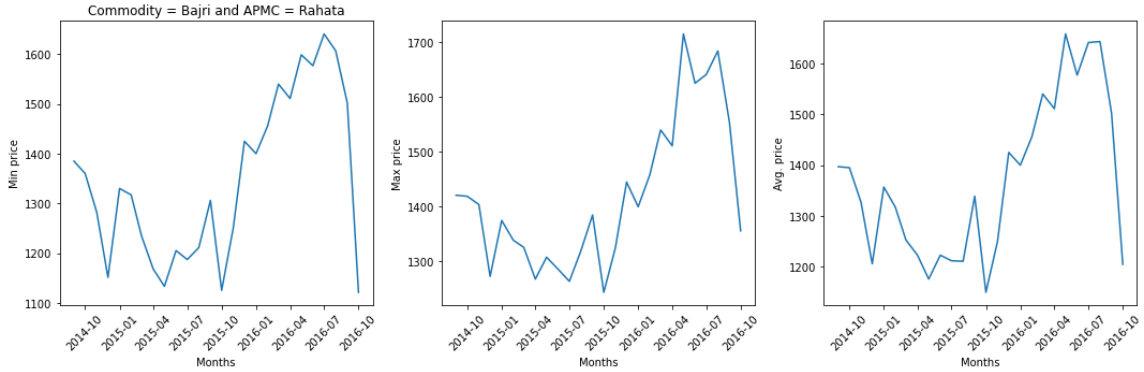
In [113…

```
# Method for applying 'Decomposition' on the original time series data to separate out trends and seasonality.

def decompose(ts_log):
    decomposition = seasonal_decompose(ts_log)

    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    plt.figure(figsize=(10, 5))
    plt.subplot(411)
    plt.plot(ts_log, label='Original')
    plt.legend(loc='best')
    plt.subplot(412)
    plt.plot(trend, label='Trend')
    plt.legend(loc='best')
    plt.subplot(413)
    plt.plot(seasonal,label='Seasonality')
    plt.legend(loc='best')
    plt.subplot(414)
    plt.plot(residual, label='Residuals')
    plt.legend(loc='best')
    plt.tight_layout()
```

APMC: Rahata and Commodities: Bajri, Onion and Capsicum

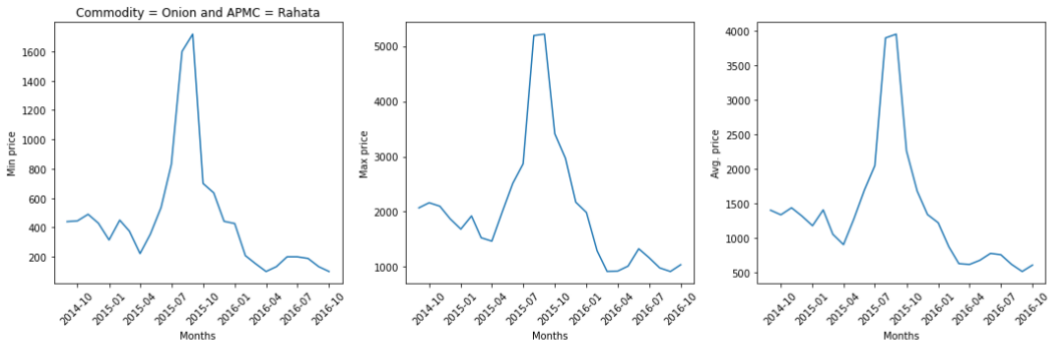
In [114…

```
plot_prices('Bajri', 'Rahata')
```



The above plots clearly demonstrate rapid variations at specific time-frames. It is evident that the seasonality is yearly and seasonality type is **multiplicative**.

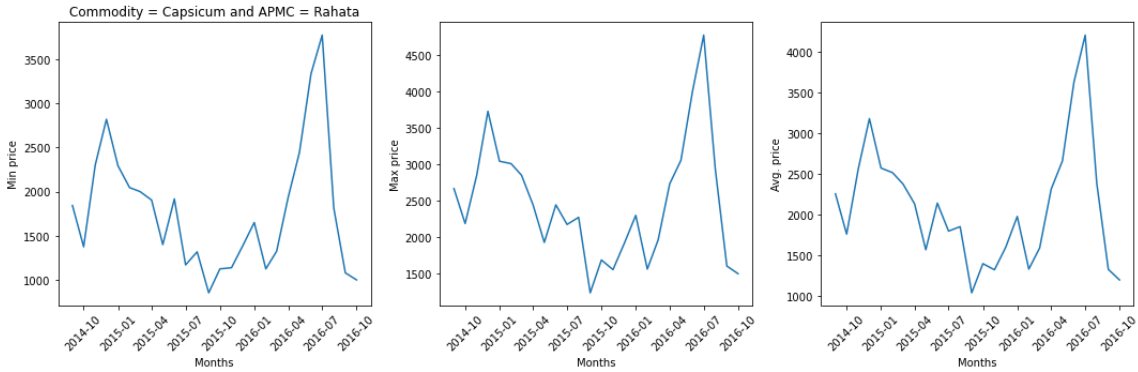
In [115…
```
plot_prices('Onion', 'Rahata')
```



The above plots clearly demonstrate a peak-shaped curve (surge) with peak around September' 2015 and rapid variations at specific time-frames. It is evident that the seasonality is yearly.

Reason: The main reason for this surge is **decreased supply**. Back in 2015, due to considerable **lower production of Onions**, supply had largely dropped. Consequently, the retail prices jumped. It had affected Onion prices in all the APMCs not only in Maharashtra but also in India as a whole.

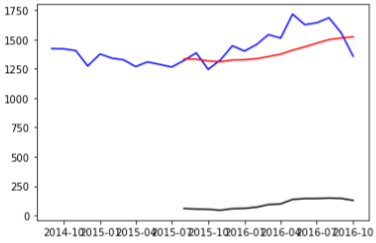
In [116…
```
plot_prices('Capsicum', 'Rahata')
```



The above plots clearly demonstrate rapid variations at specific time-frames and hence, the TS has seasonal effects.

In [117…
```
test_stationarity(df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Rahata')]['max_price'])
```

```
Results of Dickey-Fuller Test:
-3.889265672705068
-3.0543579727254224
-2.66698384083045
```



Analysis: For the commodity 'Bajri' and APMC 'Rahata', we applied 'Dickey Fuller Test' to test stationarity of the time series data for 'Modal (Average) price', 'Max price' and 'Min price' of the commodity.

As **Test Stats** < **Critical Values**, we reject the Null Hypothesis and accept the Alternate Hypothesis to say that the time series is **stationary** for 'Avg' and 'Min' prices. Whereas for 'Max' prices, **Test Stats** > **Critical Values** and hence, we accept Null Hypothesis to say that the time series is **not stationary**. Here, the seasonality type is **Multiplicative**.

Analysis: For the commodity 'Bajri' and APMC 'Rahata', we applied 'Dickey Fuller Test' to test stationarity of the time series data for 'Modal (Average) price', 'Max price' and 'Min price' of the commodity.

As **Test Stats < Critical Values**, we reject the Null Hypothesis and accept the Alternate Hypothesis to say that the time series is stationary for 'Avg' and 'Min' prices. Whereas for 'Max' prices, **Test**

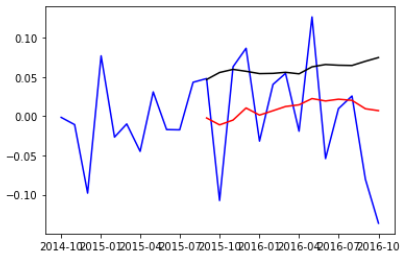
Stats > Critical Values and hence, we accept Null Hypothesis to say that the time series is not stationary. Here, the seasonality type is Multiplicative.

Eliminating Seasonality using Differencing method

In [118…

```
# Making time series for 'Max' prices stationary.
# Eliminating seasonality via Differencing method with lag 1.

# Take log of max prices since seasonality type is 'Multiplicative'.
ts_log = np.log(df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Rahata')]['max_price'])
ts_log_diff = eliminate_seasonality(ts_log)
```

```
Results of Dickey-Fuller Test:
-3.9644434814814815
-3.0849081481481484
-2.6818144444444445
```



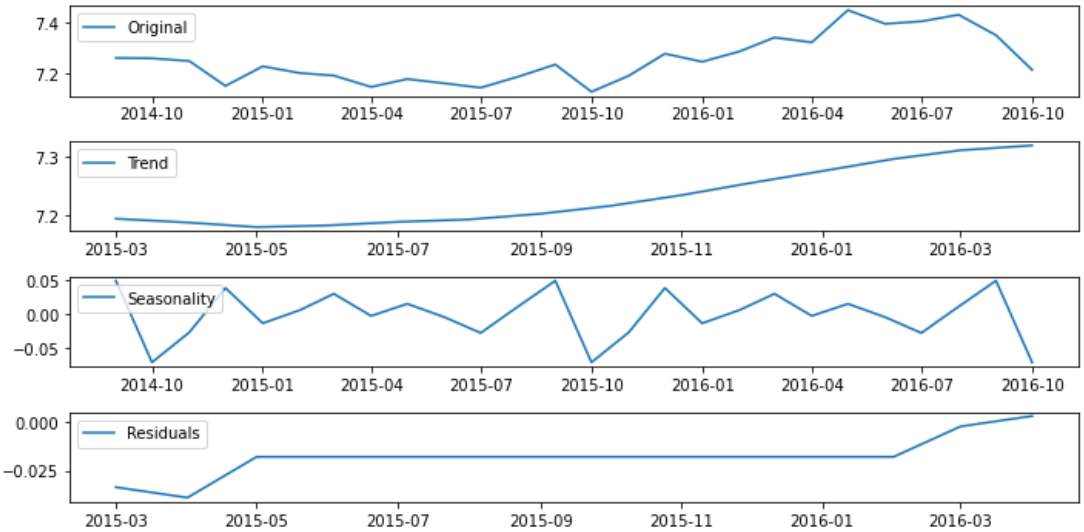
Analysis: For max price value of the commodity 'Bajri', **Test Stats value is smaller than 10% Critical value**. It means the time series is **stationary with 90% confidence**.

Since the seasonality type was 'Multiplicative', we took **log transformation** of all the three prices to make them additive [since **log(u.v.w)** = **log(u)** + **log(v)** + **log(w)**] and then, applied **Differencing method** to de-seasonalize the prices.

Applying STR Decomposition to separate out trends, seasonality and residuals.

In [119…

```
# STR Decomposition on APMC: Rahata and Commodity: Bajri
decompose(ts_log)
```



From STR Decomposition, trends and seasonality can be easily removed to get the residual de-seasonalized series.

Comparison of APMC prices with Min. Support Prices of the commodities

In [121…

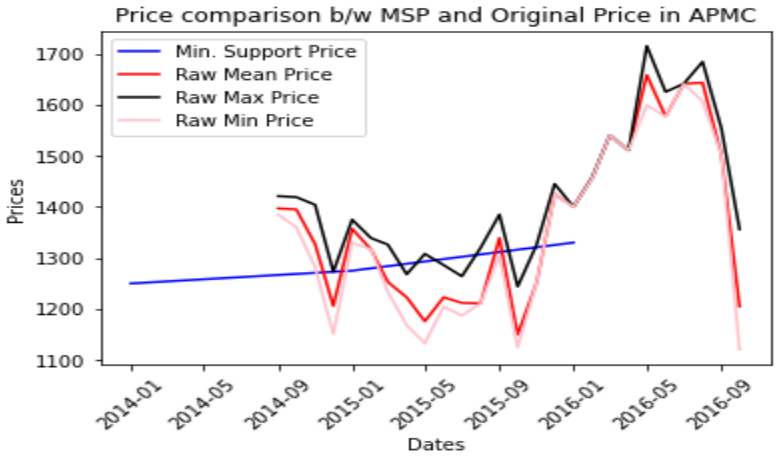
```
ts = df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Rahata')]
ts_mandi = df_mandi.loc[(df_mandi['commodity'] == 'BAJRI') & (df_mandi.index > datetime.datetime(2013, 12, 1))]
```

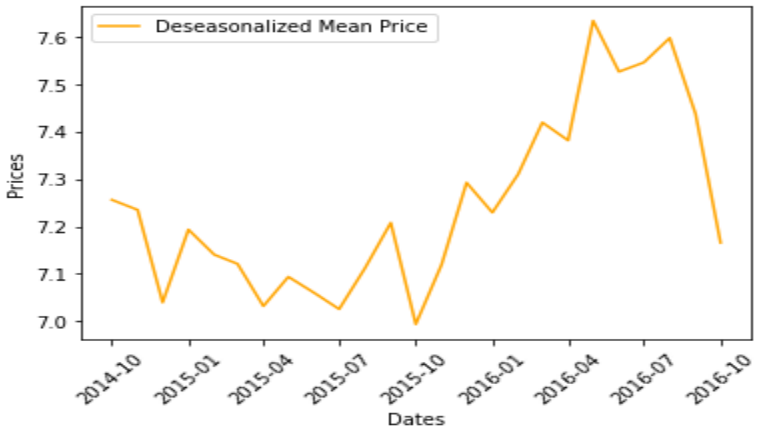
In [122…

```
ts_log_diff_cumsum = ts_log_diff.cumsum()

ts_pred_log = pd.Series(ts_log.ix[0], index=ts_log_diff.index)
ts_pred_log = ts_pred_log.add(ts_log_diff_cumsum, fill_value=0)
```

In [123…

```
# Compare prices in APMC: Rahata with Minimum Support Price for the Commodity: Bajri

ts = df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Rahata')]
ts_mandi = df_mandi.loc[(df_mandi['commodity'] == 'BAJRI') & (df_mandi.index > datetime.datetime(2013, 12, 1))]

# Code to bring back the TS into its original domain.

ts_log_diff_cumsum = ts_log_diff.cumsum()

ts_pred_log = pd.Series(ts_log, index=ts_log_diff.index)
#print(ts_pred_log.head())
ts_pred_log = ts_pred_log.add(ts_log_diff_cumsum, fill_value=0)

msp = plt.plot(ts_mandi['msprice'], color='blue', label='Min. Support Price')
orig_mean_price = plt.plot(ts['modal_price'], color='red', label='Raw Mean Price')
orig_max_price = plt.plot(ts['max_price'], color='black', label = 'Raw Max Price')
orig_min_price = plt.plot(ts['min_price'], color='pink', label = 'Raw Min Price')
#deseas_mean_price = plt.plot(ts_pred_log, color='orange', label = 'Deseas Mean Price')

plt.legend(loc='best')
plt.title('Price comparison b/w MSP and Original Price in APMC')
plt.xticks(rotation=45)
plt.ylabel('Prices')
plt.xlabel('Dates')
plt.show(block=False)


deseasonalized_ts_mean = (ts_pred_log)
plt.figure()
plt.plot(deseasonalized_ts_mean, color='orange', label = 'Deseasonalized Mean Price')
plt.legend(loc='best')
plt.xticks(rotation=45)
plt.ylabel('Prices')
plt.xlabel('Dates')
plt.show()
```





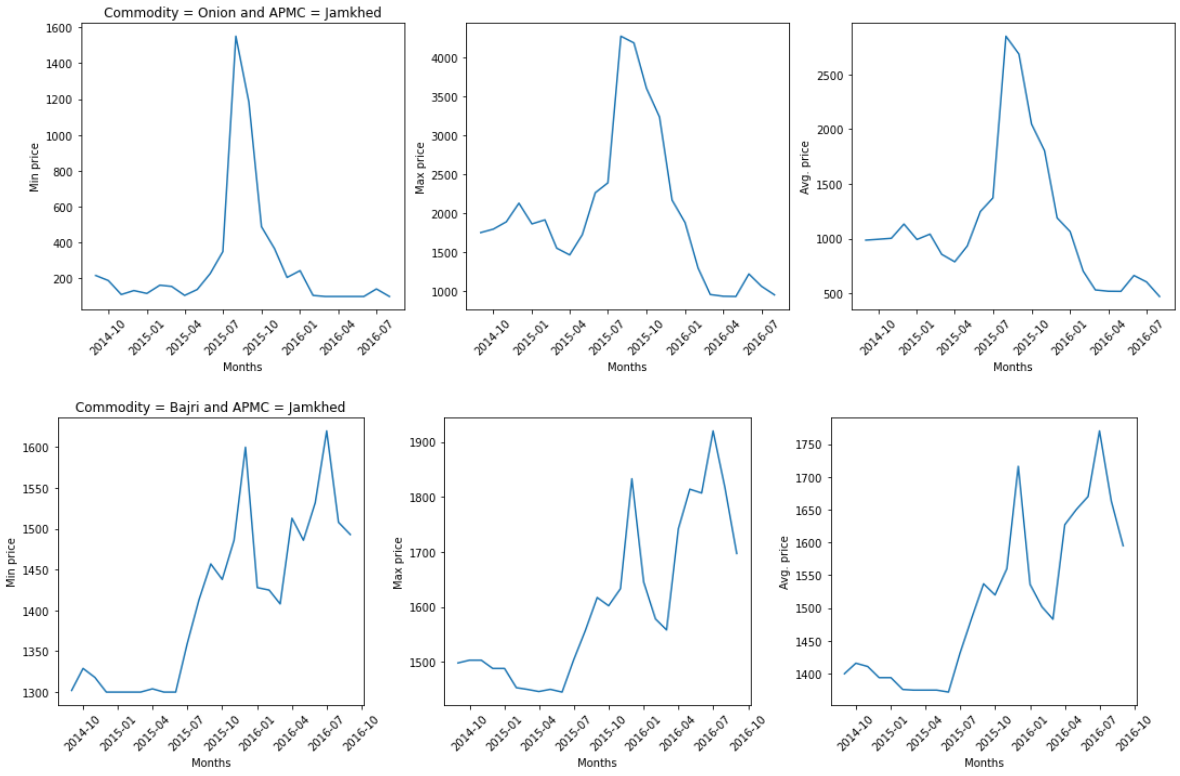
The above line plot is plotted to compare MSP prices for the commodity with its average prices over months.

Analysis: The blue line above for MSPs is below than other lines demonstrating max., average and min. prices of the commodity respectively from top to bottom in 2016. It shows that for the selected commodity, the minimum prices set by the Government are quite low and the prices at which the commodities are being sold at APMCs are soaring high in the sky.

The main reason for such a large price fluctuation between min. price set by the Government and actual price of the commodity in 2016 can be: **Very high transportation cost due to increase in the petrol/diesel prices**. It also depends on the geographical location of APMC. If it is near to the farming grounds, the cost is generally lower and very high in case the transportation cost is large for larger distances.

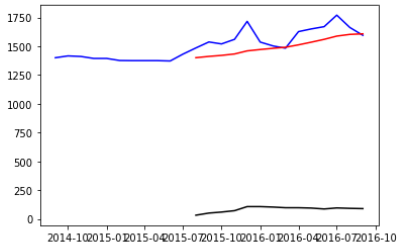
APMC: Jamkhed and Commodities: Bajri and Onion

```
In [124…
plot_prices('Onion', 'Jamkhed')
plot_prices('Bajri', 'Jamkhed')
#plot_prices('Capsicum', 'Jamkhed')
```



In [125...
```
# Testing stationarity of the timeseries using 'Dickey-Fuller' test.
test_stationarity(df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Jamkhed')]['modal_price'])
```

```
Results of Dickey-Fuller Test:
-3.7377092158564813
-2.9922162731481485
-2.635746736111111
```



Analysis: For the commodity 'Bajri' and APMC 'Jamkhed', we applied 'Dickey Fuller Test' to test stationarity of the time series data for 'Modal (Average) price', 'Max price' and 'Min price' of the commodity.

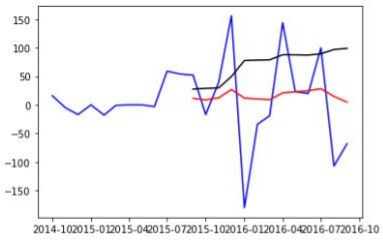
As **Test Stats** > **Critical Values**, we accept the Null Hypothesis to say that the time series is **not stationary** for all 'Avg', 'Max' and 'Min' prices. Here, the seasonality type is **Additive**.

Eliminating Seasonality using Differencing method

In [126...
```
# Making time series for 'Max' prices stationary.
# Eliminating seasonality

ts_log = df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Jamkhed')]['modal_price']
ts_log_diff = eliminate_seasonality(ts_log)

print(ts_log.head())
ts_log_diff.head()
```

```
Results of Dickey-Fuller Test:
-3.859073285322359
-3.0420456927297668
-2.6609064197530863
date
2014-09-01    1400
2014-10-01    1416
2014-11-01    1411
2014-12-01    1394
2015-01-01    1394
Name: modal_price, dtype: int64
```

Out[126...
```
date
2014-10-01    16.0
2014-11-01    -5.0
2014-12-01   -17.0
2015-01-01     0.0
2015-02-01   -18.0
Name: modal_price, dtype: float64
```

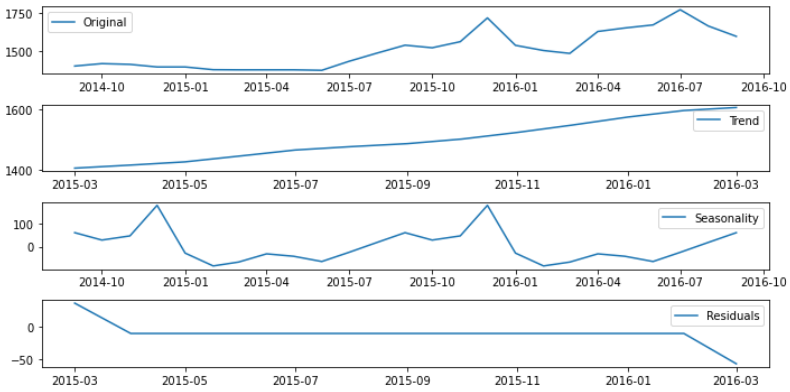


Analysis: Now **'Test Stats' value is less than the 5% Critical value**, we can surely say that the time series for: max, min and average prices for 'Bajri' commodity is **stationary** with **95% confidence** using 'Differencing' method. It means we have deseasonalized the prices for the commodity.

Applying STR Decomposition to separate out trends, seasonality and residuals.

In [127…

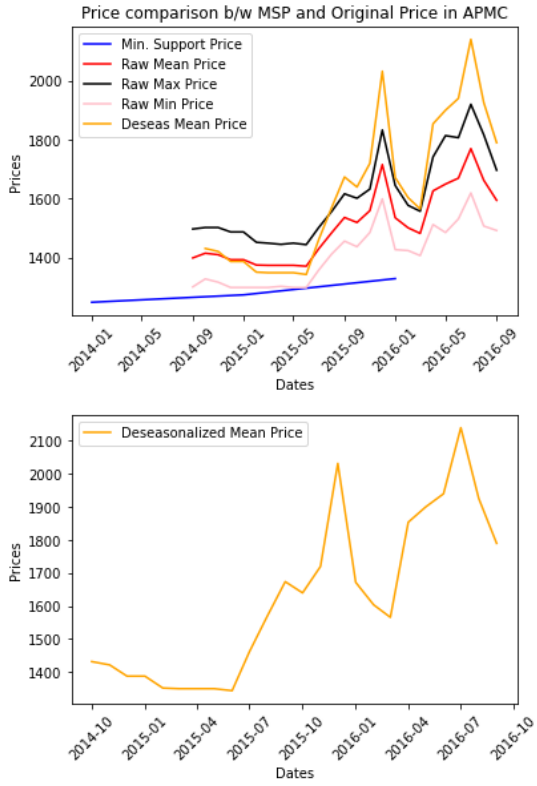
```
# Applying STR Decomposition for Commodity: Bajri and APMC: Jamkhed
decompose(ts_log)
```



Comparison of APMC prices with Min. Support Prices of the commodities

In [129…

```
# Compare prices in APMC: Jamkhed with Minimum Support Price for the Commodity: Bajri

ts = df_monthly.loc[(df_monthly['Commodity']=='Bajri') & (df_monthly['APMC']=='Jamkhed')]
ts_mandi = df_mandi.loc[(df_mandi['commodity'] == 'BAJRI') & (df_mandi.index > datetime.datetime(2013, 12, 1))]

# Code to bring back the TS into its original domain.

ts_log_diff_cumsum = ts_log_diff.cumsum()

ts_pred_log = pd.Series(ts_log, index=ts_log_diff.index)
#print(ts_pred_log.head())
ts_pred_log = ts_pred_log.add(ts_log_diff_cumsum, fill_value=0)

msp = plt.plot(ts_mandi['msprice'], color='blue', label='Min. Support Price')
orig_mean_price = plt.plot(ts['modal_price'], color='red', label='Raw Mean Price')
orig_max_price = plt.plot(ts['max_price'], color='black', label = 'Raw Max Price')
orig_min_price = plt.plot(ts['min_price'], color='pink', label = 'Raw Min Price')
deseas_mean_price = plt.plot(ts_pred_log, color='orange', label = 'Deseas Mean Price')

plt.legend(loc='best')
plt.title('Price comparison b/w MSP and Original Price in APMC')
plt.xticks(rotation=45)
plt.ylabel('Prices')
plt.xlabel('Dates')
plt.show(block=False)


deseasonalized_ts_mean = (ts_pred_log)
plt.figure()
plt.plot(deseasonalized_ts_mean, color='orange', label = 'Deseasonalized Mean Price')
plt.legend(loc='best')
plt.xticks(rotation=45)
plt.ylabel('Prices')
plt.xlabel('Dates')
plt.show()
```



The above line plot is plotted to compare MSP prices for the commodity with its average prices over months.

Analysis: The blue line above for MSPs is below than other lines demonstrating max., average and min. prices of the commodity respectively from top to bottom. It shows that for the selected commodity, the minimum prices set by the Government are quite low and the prices at which the commodities are being sold at APMCs are soaring high in the sky.

The main reason for such a large price fluctuation between min. price set by the Government and actual price of the commodity can be: **Very high transportation cost**. It also depends on the geographical location of APMC. If it is near to the farming grounds, the cost is generally lower and very high in case the transportation cost is large for larger distances.

Section 4: Analyse price fluctuations across different commodities in each season and year

Methodology

Since there are total **352 commodities** and **349 APMCs** in the dataset, we have taken a sample of 4 APMCs: Ahmednagar, Rahata, Jamkhed and Sangamner and again, sample of 4 commodities: Bajri, Onion, Capsicum and Soyabean. But all the methods are written in a generic way so that the analysis can be easily extended to more number of APMCs and commodities just by adding more commodities and APMCs in the list.

Firstly, we separated the whole dataset year-wise (i.e. for 2014, 2015 and 2016) to do yearly analysis.

- No. of entries for year 2016: 28971

- No. of entries for year 2015: 25557
- No. of entries for year 2014: 7901

For each year in [2014, 2015, 2016], we do the following:

- Plot line plot for maximum, average and minimum prices of the commodity for that year.
- Take difference of maximum price and minimum price of the commodity to get the **Maximum Price Fluctuation** in that year.
- Plot 2 bar plots (APMCs v/s Price Fluctuations and Commodities v/s Price Fluctuations) to visually analyse the max. Price Fluctuations corresponding to an APMC and Commodity for each year.

In [130…
```
# Different months in the dataset
set(df_monthly['Month'])
```

Out[130…
```
{'April',
 'August',
 'December',
 'February',
 'January',
 'July',
 'June',
 'March',
 'May',
 'November',
 'October',
 'September'}
```

In [131…
```
# Number of records for each year: 2014, 2015, 2016.
df_2016 = df_monthly.loc[df_monthly['Year'] == 2016]
print('No. of records for year 2016:', len(df_2016))

df_2015 = df_monthly.loc[df_monthly['Year'] == 2015]
print('No. of records for year 2015:', len(df_2015))

df_2014 = df_monthly.loc[df_monthly['Year'] == 2014]
print('No. of records for year 2014:', len(df_2014))
```

```
No. of records for year 2016: 28971
No. of records for year 2015: 25557
No. of records for year 2014: 7901
```

In [132…
```
# List of APMCs and Commodities whose price fluctuations (for each pair) we need to analyze.

apmcs = ['Ahmednagar', 'Rahata', 'Jamkhed', 'Sangamner']
commodities = ['Bajri', 'Onion', 'Capsicum', 'Soybean']
```

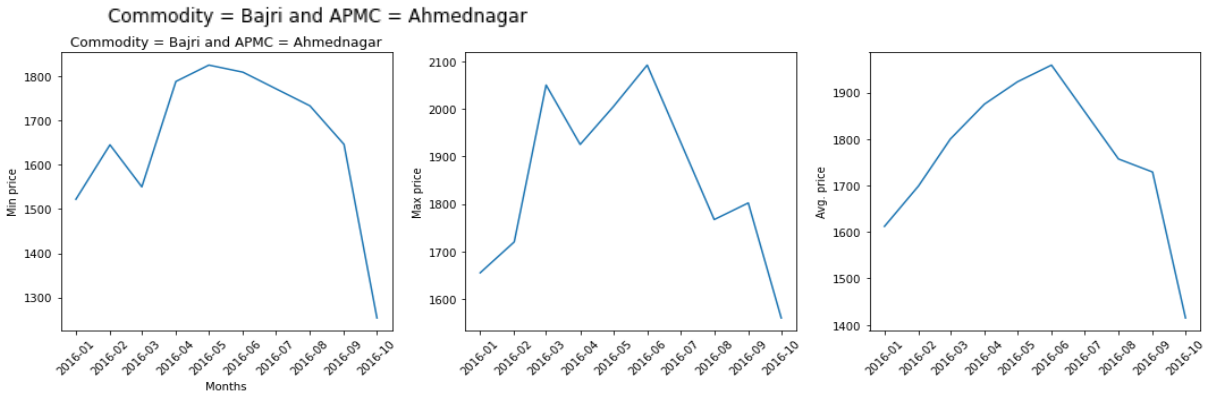
In [133…
```
# Method to plot prices (max, min and average prices) corresponding to the given commodity and APMC.

def plot_prices_of_commodity(df, commodity, apmc):
    plt.figure(figsize=(15,5))

    plt.subplot(1, 3, 1)
    plt.plot(df['min_price'])
    plt.ylabel('Min price')
    plt.xlabel('Months')
    plt.xticks(rotation=45)
    plt.title('Commodity = %s and APMC = %s' % (commodity, apmc))

    plt.subplot(1, 3, 2)
    plt.plot(df['max_price'])
    plt.ylabel('Max price')
    plt.xticks(rotation=45)

    plt.subplot(1, 3, 3)
    plt.plot(df['modal_price'])
    plt.ylabel('Avg. price')
    plt.xticks(rotation=45)

    plt.tight_layout()
```

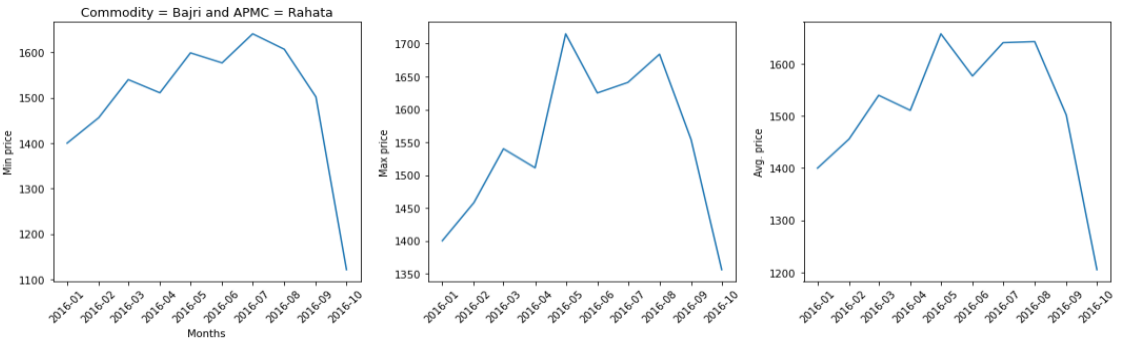
Analyzing price fluctuations for 2016 year

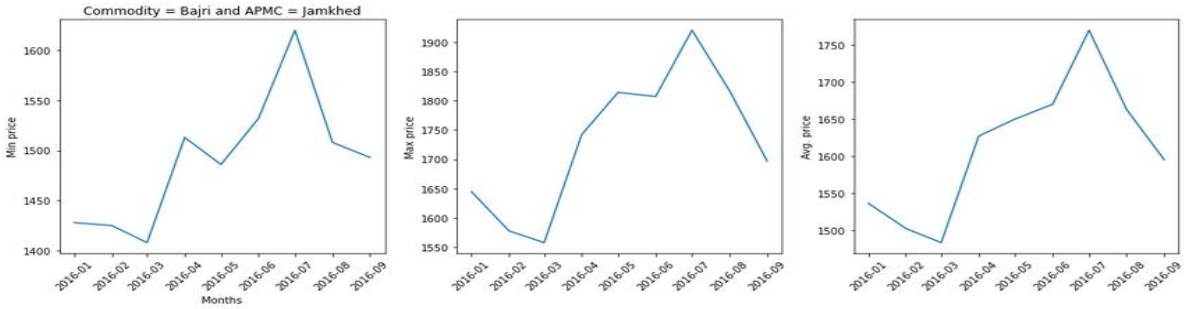
In [134…

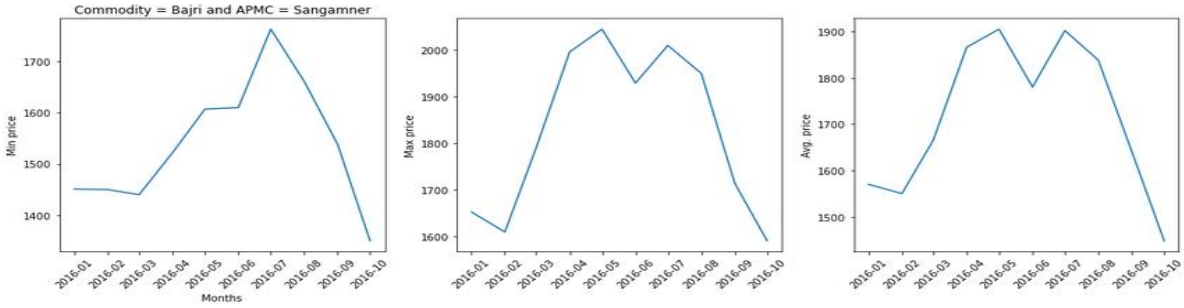
```
comm_fluctuations = {}
apmc_fluctuations = {}

# For each pair of commodity and APMC, this method computes the Highest Price Fluctuations for the year 2016 by
# differencing the maximum and minimum prices across different months of the year to get the max. price fluctuation.

for c in commodities:
    for apmc in apmcs:
        df_2016_filt = df_2016.query("Commodity=='%s' and APMC=='%s'" % (c, apmc))
        if len(df_2016_filt) == 0:
            continue
        plot_prices_of_commodity(df_2016_filt, c, apmc)
        avg_prices = df_2016_filt['modal_price']
        print('Commodity: %s and APMC: %s' % (c, apmc))
        print('Highest Price Fluctuation (per quintal): %d' % (max(avg_prices) - min(avg_prices)))

        if c not in comm_fluctuations:
            comm_fluctuations.update({c: max(avg_prices) - min(avg_prices)})
        else:
            comm_fluctuations[c] = max(comm_fluctuations[c], max(avg_prices) - min(avg_prices))

        if apmc not in apmc_fluctuations:
            apmc_fluctuations.update({apmc: max(avg_prices) - min(avg_prices)})
        else:
            apmc_fluctuations[apmc] = max(apmc_fluctuations[apmc], max(avg_prices) - min(avg_prices))
```

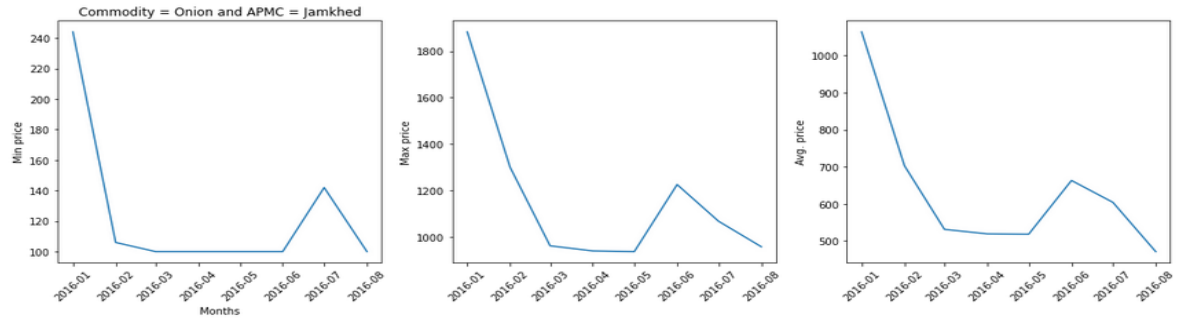
```
Commodity: Bajri and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 544
Commodity: Bajri and APMC: Rahata
Highest Price Fluctuation (per quintal): 453
Commodity: Bajri and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 287
Commodity: Bajri and APMC: Sangamner
Highest Price Fluctuation (per quintal): 457
Commodity: Onion and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 775
Commodity: Onion and APMC: Rahata
Highest Price Fluctuation (per quintal): 706
Commodity: Onion and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 593
Commodity: Onion and APMC: Sangamner
Highest Price Fluctuation (per quintal): 659
Commodity: Capsicum and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 3206
Commodity: Capsicum and APMC: Rahata
Highest Price Fluctuation (per quintal): 3008
Commodity: Soybean and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 1190
Commodity: Soybean and APMC: Rahata
Highest Price Fluctuation (per quintal): 1208
Commodity: Soybean and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 420
Commodity: Soybean and APMC: Sangamner
Highest Price Fluctuation (per quintal): 869
```

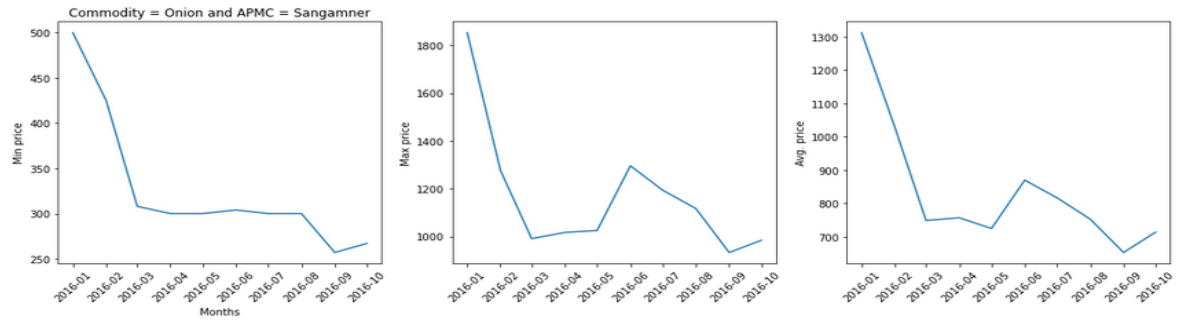


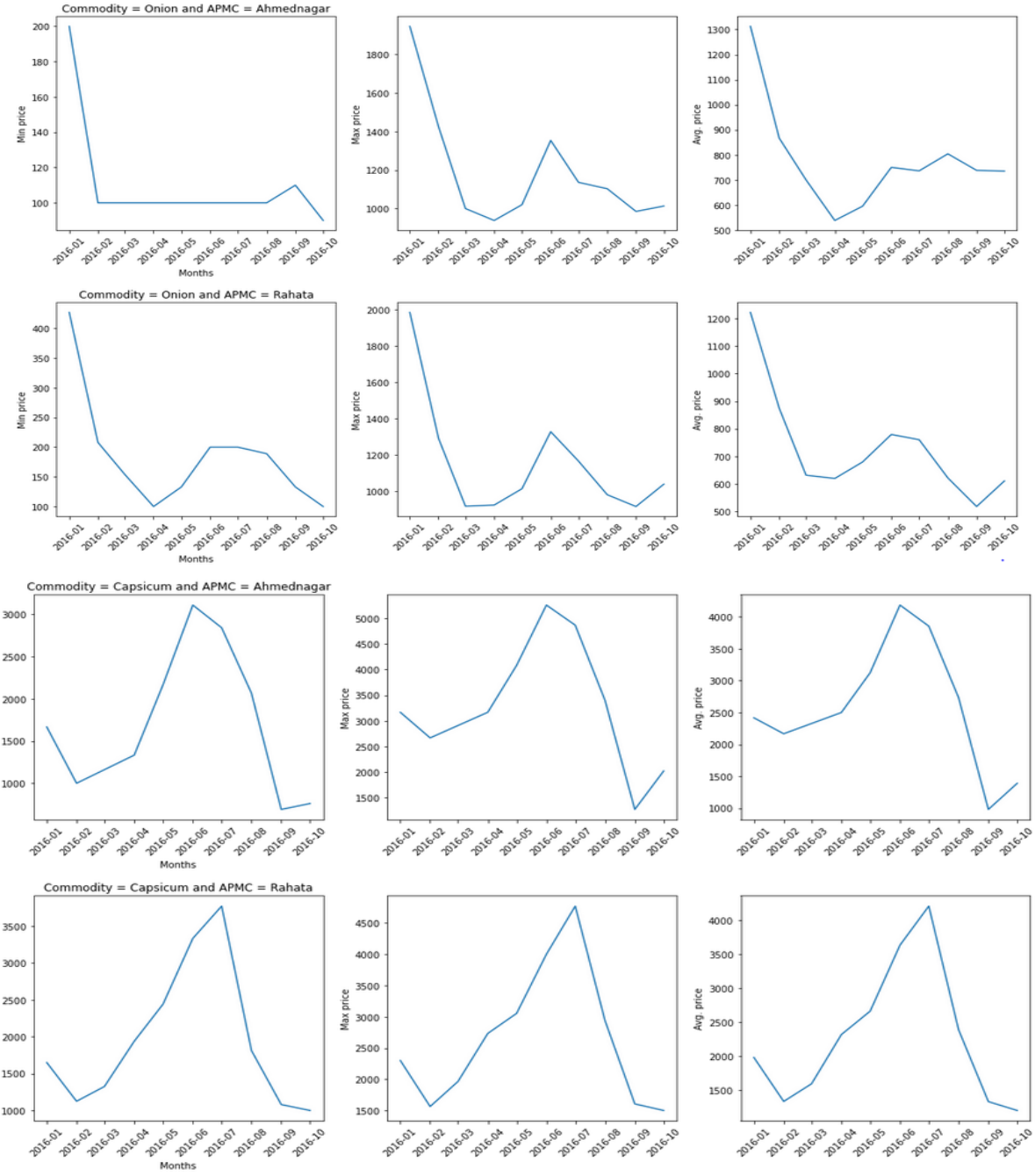

Commodity = Bajri and APMC = Rahata
Min price
Max price
Avg_price
Months

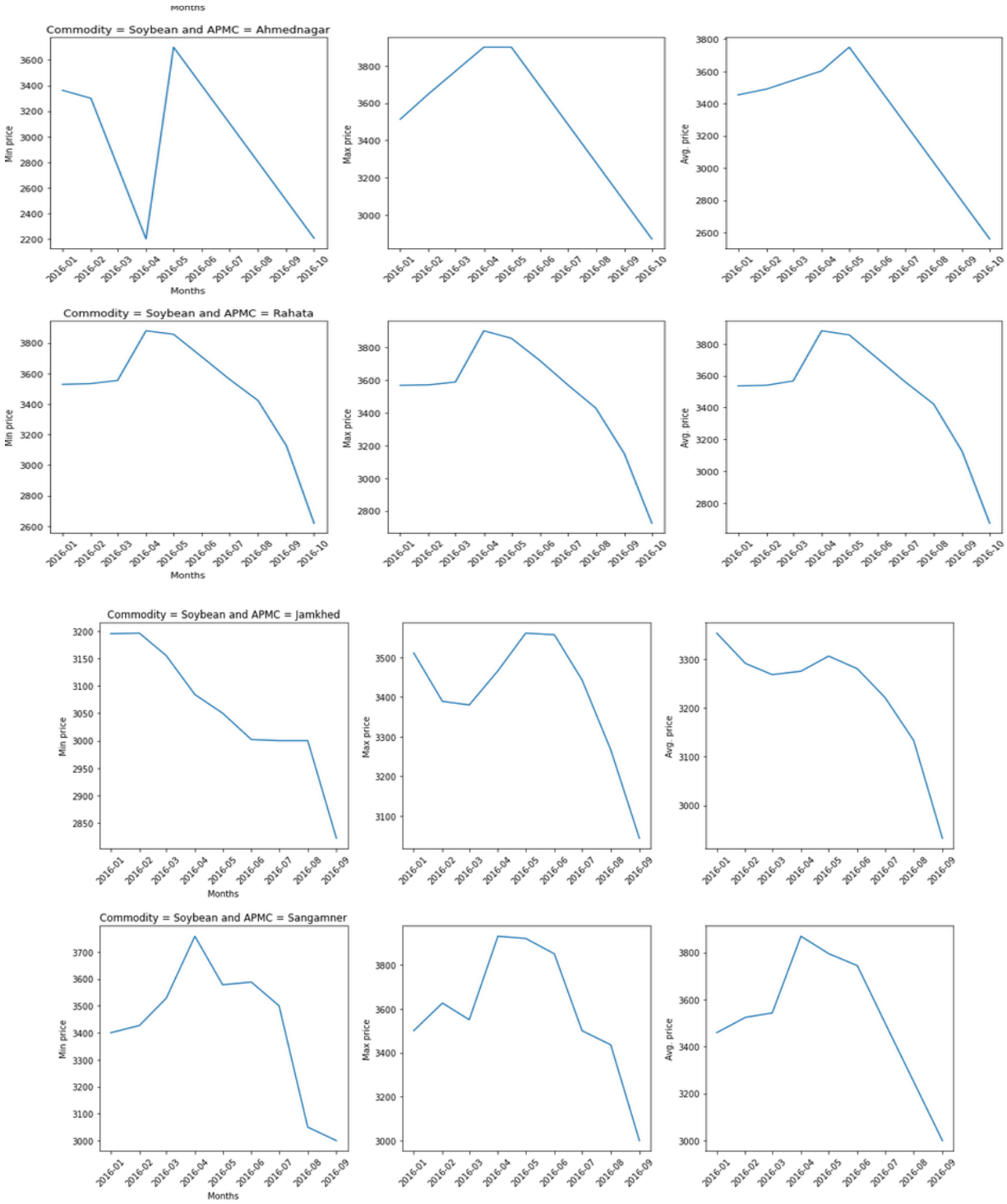

Commodity = Bajri and APMC = Jamkhed
Min price
Max price
Avg_price
Months


Commodity = Bajri and APMC = Sangamner
Min price
Max price
Avg_price
Months


Commodity = Onion and APMC = Jamkhed
Min price
Max price
Avg_price
Months


Commodity = Onion and APMC = Sangamner
Min price
Max price
Avg_price
Months


Commodity = Onion and APMC = Ahmednagar
Min price
Max price
Avg. price
Months
Commodity = Onion and APMC = Rahata
Min price
Max price
Avg. price
Months
Commodity = Capsicum and APMC = Ahmednagar
Min price
Max price
Avg. price
Months
Commodity = Capsicum and APMC = Rahata
Min price
Max price
Avg. price
Months


Months
Commodity = Soybean and APMC = Ahmednagar
Min price
Max price
Avg price
Months
Commodity = Soybean and APMC = Rahata
Min price
Max price
Avg price
Months
Commodity = Soybean and APMC = Jamkhed
Min price
Max price
Avg price
Months
Commodity = Soybean and APMC = Sangamner
Min price
Max price
Avg price
Months

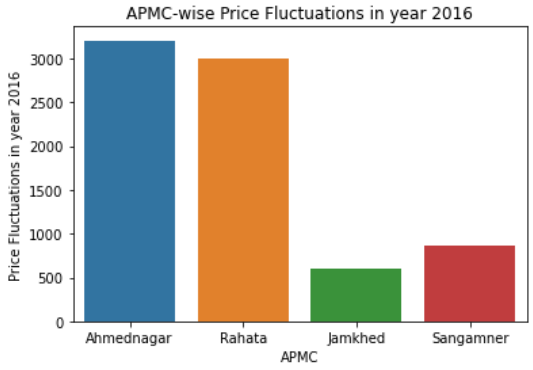
In [135…

```
print(apmc_fluctuations)
print(comm_fluctuations)
```

```
{'Ahmednagar': 3206, 'Rahata': 3008, 'Jamkhed': 593, 'Sangamner': 869}
{'Bajri': 544, 'Onion': 775, 'Capsicum': 3206, 'Soybean': 1208}
```

In [136…

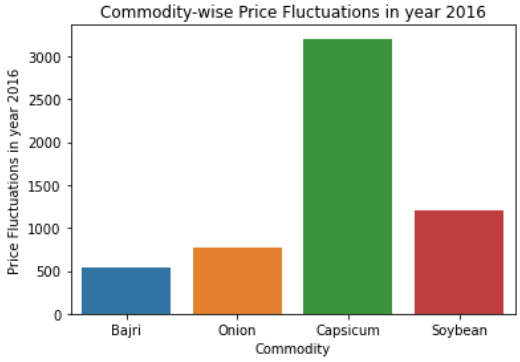
```
# Visualize Price Fluctuations based on APMCs.

ax = sns.barplot(x = list(apmc_fluctuations.keys()), y = list(apmc_fluctuations.values()))
plt.title('APMC-wise Price Fluctuations in year 2016')
plt.xlabel('APMC')
plt.ylabel('Price Fluctuations in year 2016')
plt.show()
```



Analysis: The above bar plot demonstrates that Ahmednagar APMC suffers from maximum Price Fluctuations in the year 2016.

In [137…

```
# Visualize Price Fluctuations based on Commodities.

ax = sns.barplot(x = list(comm_fluctuations.keys()), y = list(comm_fluctuations.values()))
plt.title('Commodity-wise Price Fluctuations in year 2016')
plt.xlabel('Commodity')
plt.ylabel('Price Fluctuations in year 2016')
plt.show()
```



Analysis: The above bar plot demonstrates that Capsicum commodity suffers from maximum Price Fluctuations in the year 2016.

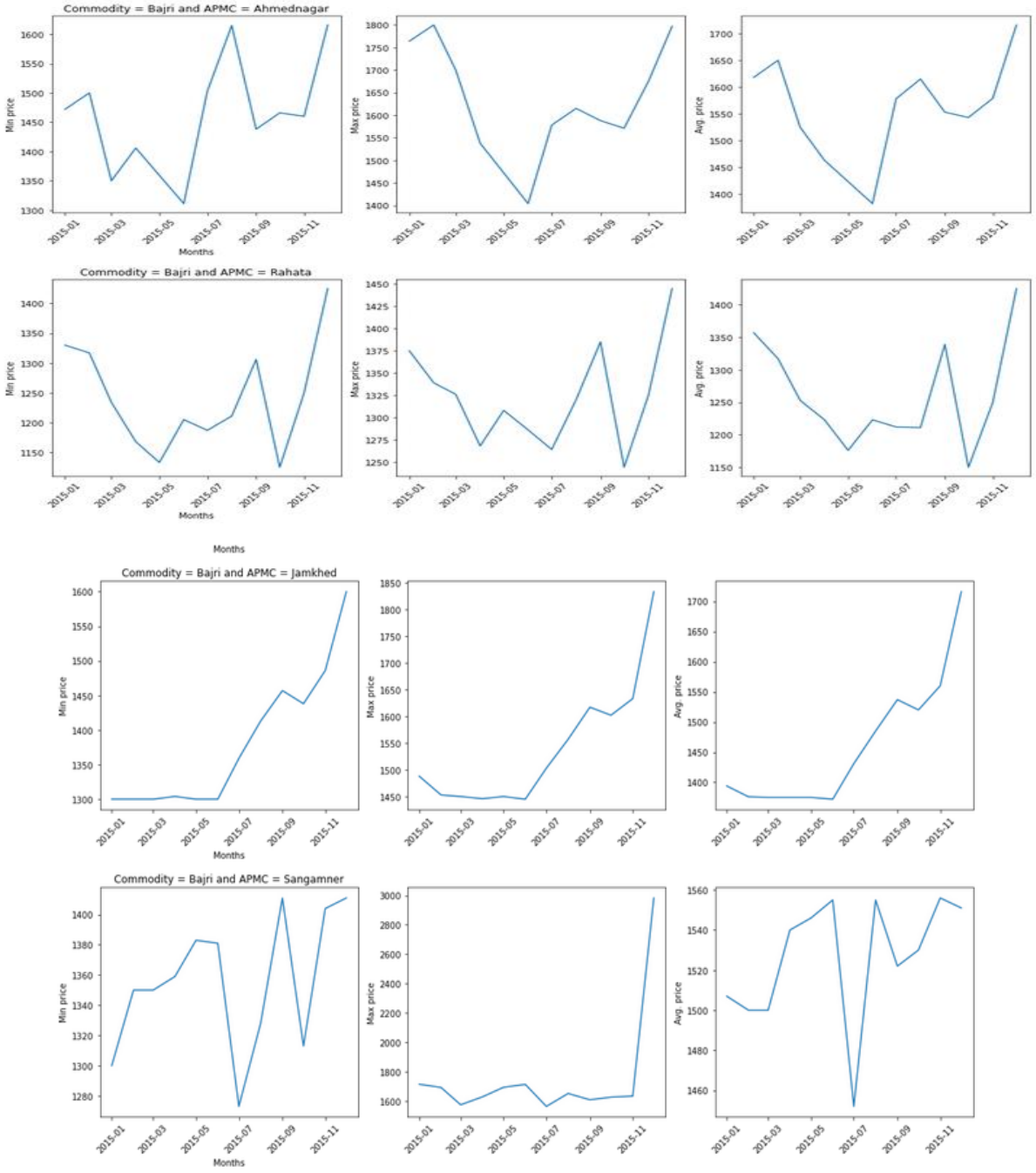
So, we can conclude that **APMC: Ahmednagar and Commodity: Capsicum suffer from the highest Price Fluctuations**.

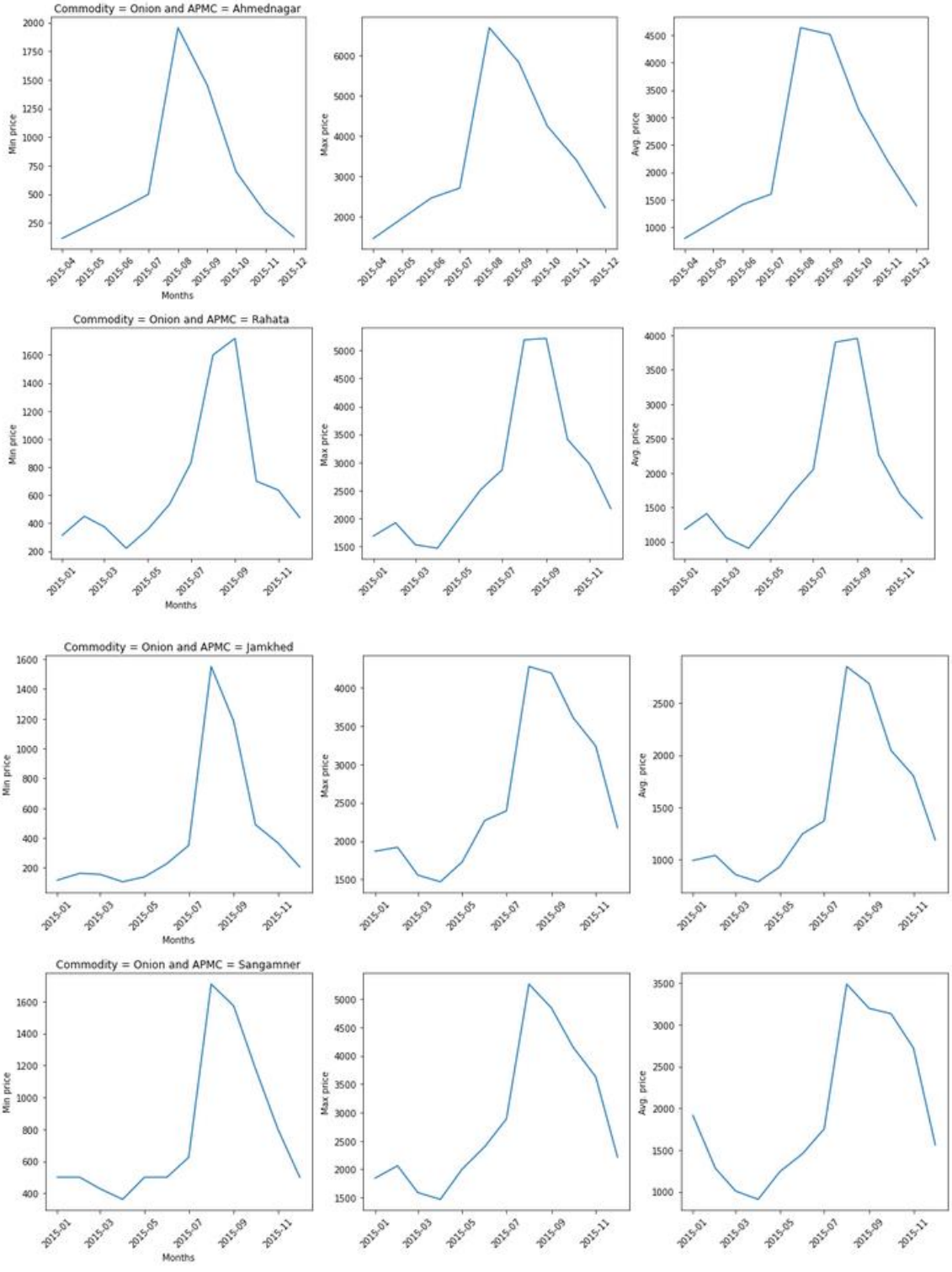
Analyzing price fluctuations for 2015 year

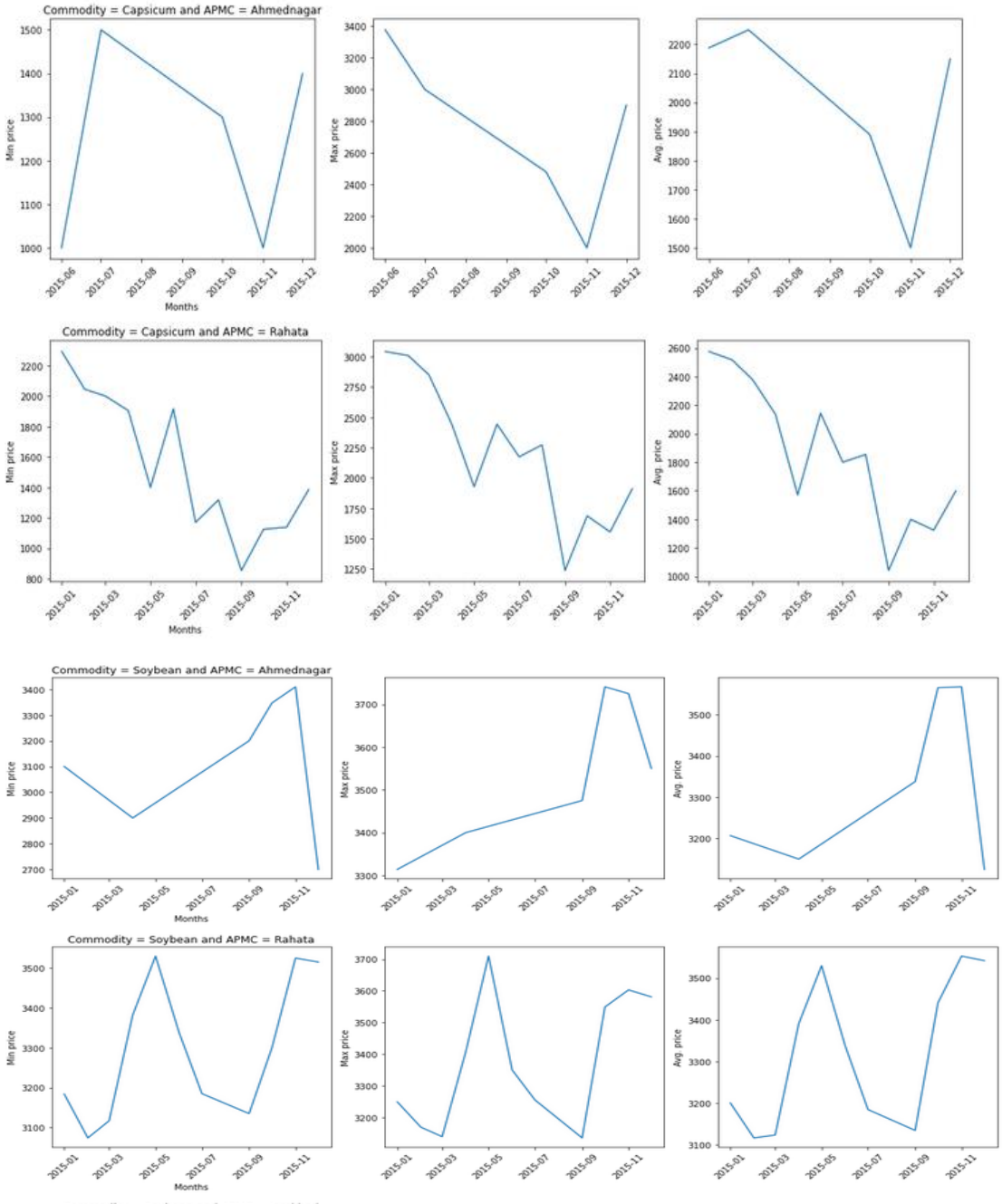
In [138…

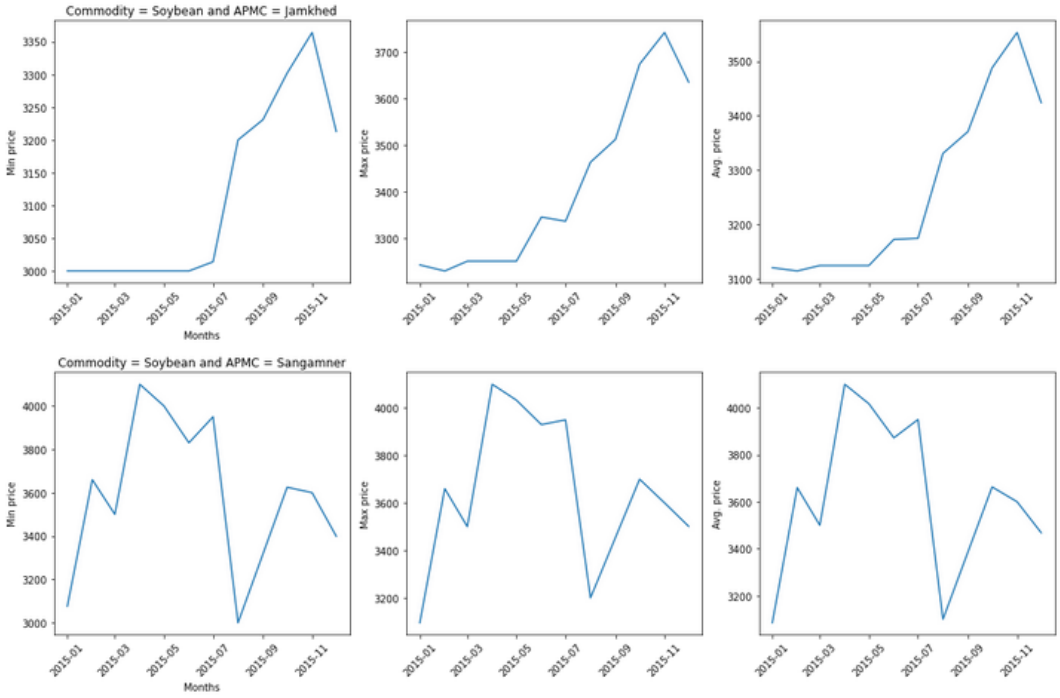
```
comm_fluctuations = {}
apmc_fluctuations = {}

# For each pair of commodity and APMC, this method computes the Highest Price Fluctuations for the year 2015 by
# differencing the maximum and minimum prices across different months of the year to get the max. price fluctuation.

for c in commodities:
    for apmc in apmcs:
        df_2015_filt = df_2015.query("Commodity=='%s' and APMC=='%s'" % (c, apmc))
        if len(df_2015_filt) == 0:
            continue
        plot_prices_of_commodity(df_2015_filt, c, apmc)
        avg_prices = df_2015_filt['modal_price']
        print('Commodity: %s and APMC: %s' % (c, apmc))
        print('Highest Price Fluctuation (per quintal): %d' % (max(avg_prices) - min(avg_prices)))

        if c not in comm_fluctuations:
            comm_fluctuations.update({c: max(avg_prices) - min(avg_prices)})
        else:
            comm_fluctuations[c] = max(comm_fluctuations[c], max(avg_prices) - min(avg_prices))

        if apmc not in apmc_fluctuations:
            apmc_fluctuations.update({apmc: max(avg_prices) - min(avg_prices)})
        else:
            apmc_fluctuations[apmc] = max(apmc_fluctuations[apmc], max(avg_prices) - min(avg_prices))
```

```
Commodity: Bajri and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 334
Commodity: Bajri and APMC: Rahata
Highest Price Fluctuation (per quintal): 275
Commodity: Bajri and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 344
Commodity: Bajri and APMC: Sangamner
Highest Price Fluctuation (per quintal): 104
Commodity: Onion and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 3838
Commodity: Onion and APMC: Rahata
Highest Price Fluctuation (per quintal): 3049
Commodity: Onion and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 2062
Commodity: Onion and APMC: Sangamner
Highest Price Fluctuation (per quintal): 2579
Commodity: Capsicum and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 750
Commodity: Capsicum and APMC: Rahata
Highest Price Fluctuation (per quintal): 1533
Commodity: Soybean and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 443
Commodity: Soybean and APMC: Rahata
Highest Price Fluctuation (per quintal): 436
Commodity: Soybean and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 438
Commodity: Soybean and APMC: Sangamner
Highest Price Fluctuation (per quintal): 1015
```


Commodity = Bajri and APMC = Ahmednagar
Min price
Max price
Avg price
Months
Commodity = Bajri and APMC = Rahata
Min price
Max price
Avg price
Months
Commodity = Bajri and APMC = Jamkhed
Min price
Max price
Avg price
Months
Commodity = Bajri and APMC = Sangamner
Min price
Max price
Avg price
Months


Commodity = Onion and APMC = Ahmednagar
Min price
Max price
Avg price
Months
Commodity = Onion and APMC = Rahata
Min price
Max price
Avg price
Months
Commodity = Onion and APMC = Jamkhed
Min price
Max price
Avg price
Months
Commodity = Onion and APMC = Sangamner
Min price
Max price
Avg price
Months


Commodity = Capsicum and APMC = Ahmednagar
Min price
Max price
Avg price
Months
Commodity = Capsicum and APMC = Rahata
Min price
Max price
Avg price
Months
Commodity = Soybean and APMC = Ahmednagar
Min price
Max price
Avg price
Months
Commodity = Soybean and APMC = Rahata
Min price
Max price
Avg price
Months



In [139…

```
print(apmc_fluctuations)
```

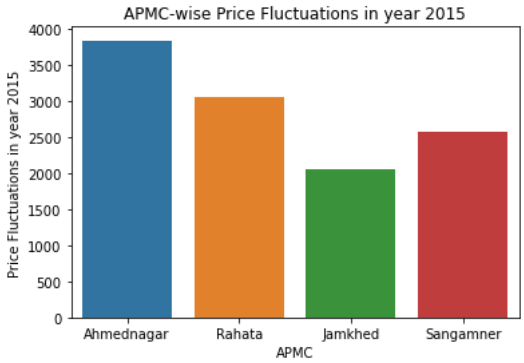
In [139…

```
print(apmc_fluctuations)
print(comm_fluctuations)
```

```
{'Ahmednagar': 3838, 'Rahata': 3049, 'Jamkhed': 2062, 'Sangamner': 2579}
{'Bajri': 344, 'Onion': 3838, 'Capsicum': 1533, 'Soybean': 1015}
```

In [140…

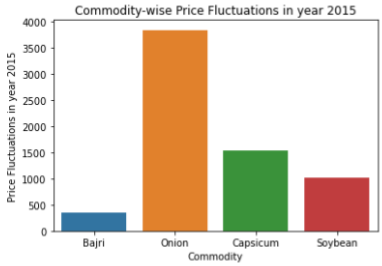
```
# Visualize Price Fluctuations based on APMCs.

ax = sns.barplot(x = list(apmc_fluctuations.keys()), y = list(apmc_fluctuations.values()))
plt.title('APMC-wise Price Fluctuations in year 2015')
plt.xlabel('APMC')
plt.ylabel('Price Fluctuations in year 2015')
plt.show()
```



Analysis: The above bar plot demonstrates that Ahmednagar APMC suffers from maximum Price Fluctuations in the year 2015.

In [141…

```
# Visualize Price Fluctuations based on Commodities.

ax = sns.barplot(x = list(comm_fluctuations.keys()), y = list(comm_fluctuations.values()))
plt.title('Commodity-wise Price Fluctuations in year 2015')
plt.xlabel('Commodity')
plt.ylabel('Price Fluctuations in year 2015')
plt.show()
```



Analysis: The above bar plot demonstrates that Onion commodity suffers from maximum Price Fluctuations in the year 2015. It is quite an expected result because we have already seen that due to **very less supply of Onions** in the market, the **retail prices soared high** in the sky across different APMCs in the country.

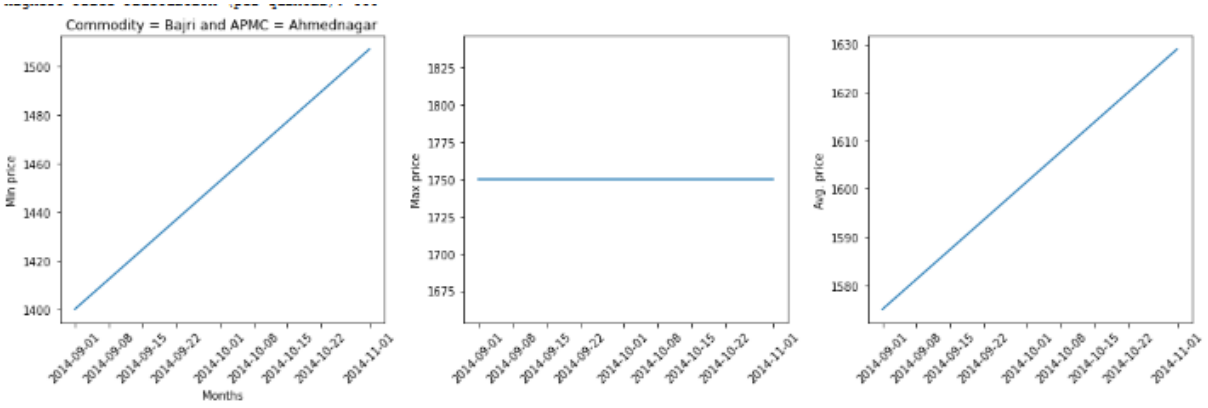
So, we can conclude that **APMC: Ahmednagar and Commodity: Onion suffer from the highest Price Fluctuations**.

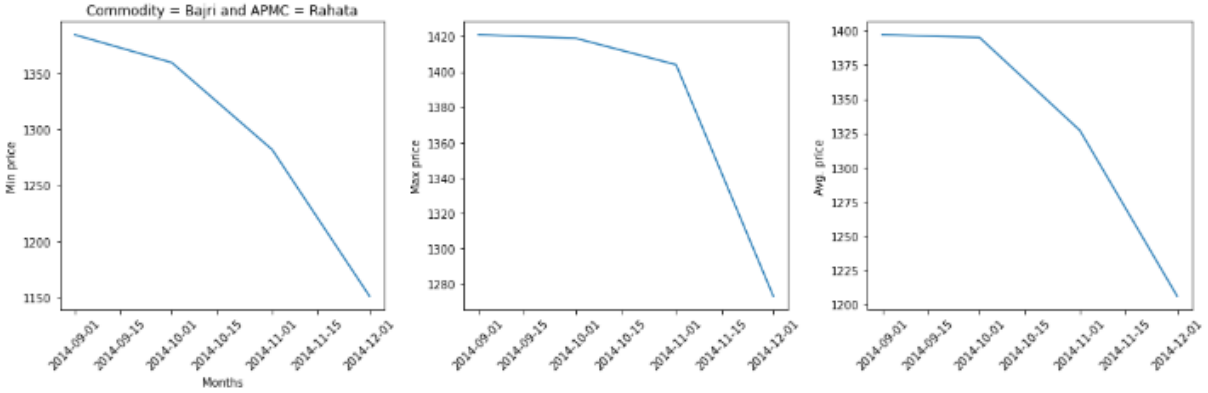
Analyzing price fluctuations for 2014 year

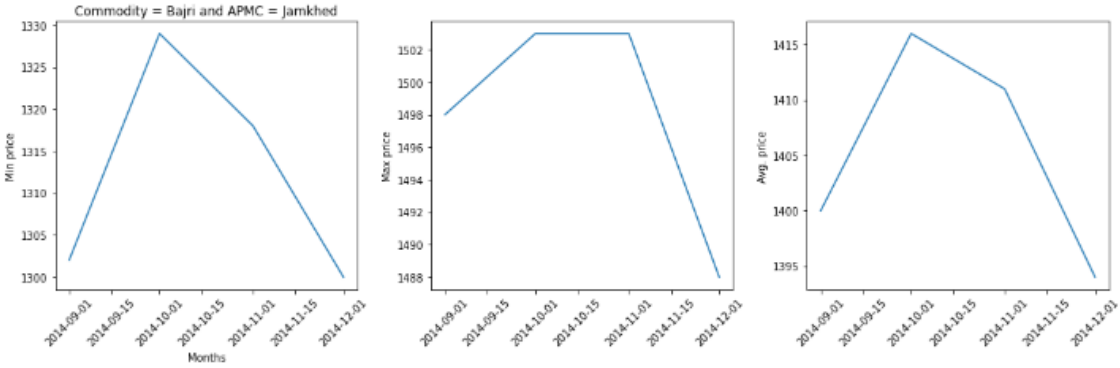
In [142…

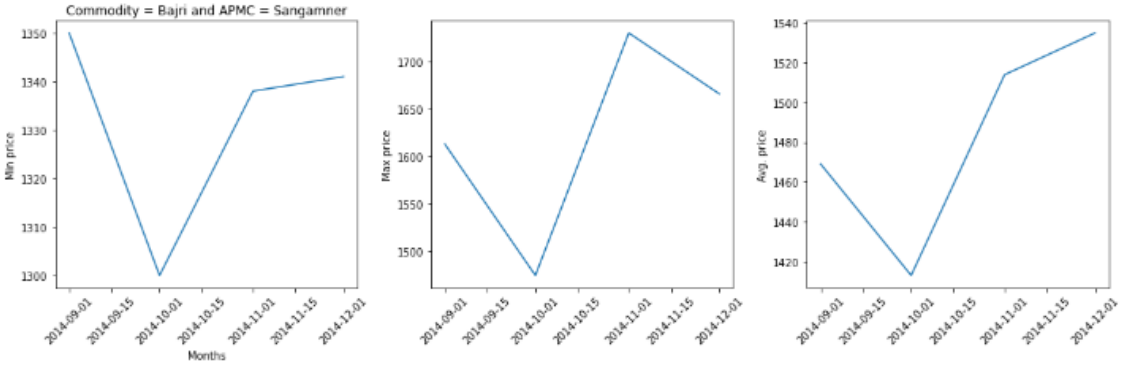
```
comm_fluctuations = {}
apmc_fluctuations = {}

# For each pair of commodity and APMC, this method computes the Highest Price Fluctuations for the year 2014 by
# differencing the maximum and minimum prices across different months of the year to get the max. price fluctuation.

for c in commodities:
    for apmc in apmcs:
        df_2014_filt = df_2014.query("Commodity=='%s' and APMC=='%s'" % (c, apmc))
        if len(df_2014_filt) == 0:
            continue
        plot_prices_of_commodity(df_2014_filt, c, apmc)
        avg_prices = df_2014_filt['modal_price']
        print('Commodity: %s and APMC: %s' % (c, apmc))
        print('Highest Price Fluctuation (per quintal): %d' % (max(avg_prices) - min(avg_prices)))

        if c not in comm_fluctuations:
            comm_fluctuations.update({c: max(avg_prices) - min(avg_prices)})
        else:
            comm_fluctuations[c] = max(comm_fluctuations[c], max(avg_prices) - min(avg_prices))

        if apmc not in apmc_fluctuations:
            apmc_fluctuations.update({apmc: max(avg_prices) - min(avg_prices)})
        else:
            apmc_fluctuations[apmc] = max(apmc_fluctuations[apmc], max(avg_prices) - min(avg_prices))
```

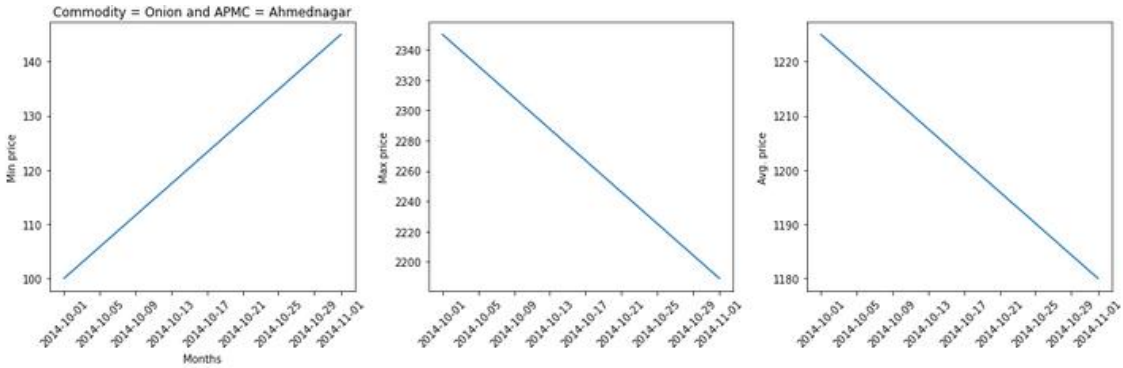
```
Commodity: Bajri and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 54
Commodity: Bajri and APMC: Rahata
Highest Price Fluctuation (per quintal): 191
Commodity: Bajri and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 22
Commodity: Bajri and APMC: Sangamner
Highest Price Fluctuation (per quintal): 122
Commodity: Onion and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 45
Commodity: Onion and APMC: Rahata
Highest Price Fluctuation (per quintal): 119
Commodity: Onion and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 147
Commodity: Onion and APMC: Sangamner
Highest Price Fluctuation (per quintal): 58
Commodity: Capsicum and APMC: Rahata
Highest Price Fluctuation (per quintal): 1419
Commodity: Soybean and APMC: Ahmednagar
Highest Price Fluctuation (per quintal): 0
Commodity: Soybean and APMC: Rahata
Highest Price Fluctuation (per quintal): 350
Commodity: Soybean and APMC: Jamkhed
Highest Price Fluctuation (per quintal): 194
Commodity: Soybean and APMC: Sangamner
Highest Price Fluctuation (per quintal): 335
```

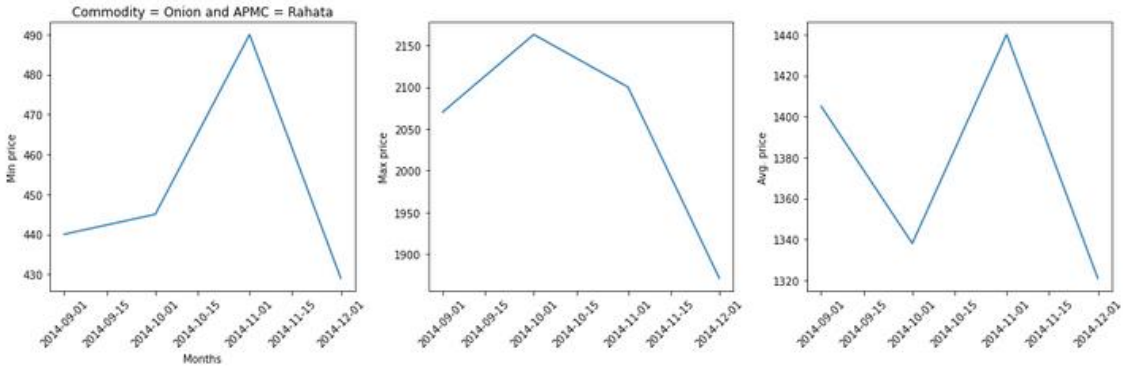

Commodity = Bajri and APMC = Ahmednagar
Min price
Max price
Avg price
Months

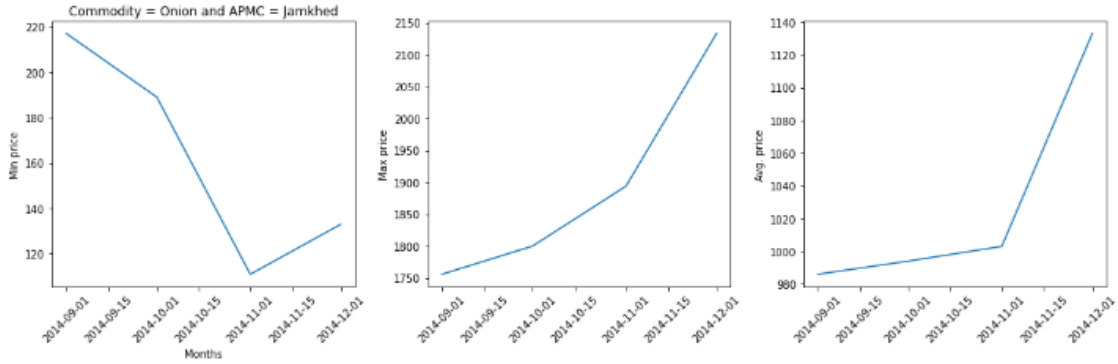

Commodity = Bajri and APMC = Rahata
Min price
Max price
Avg price
Months

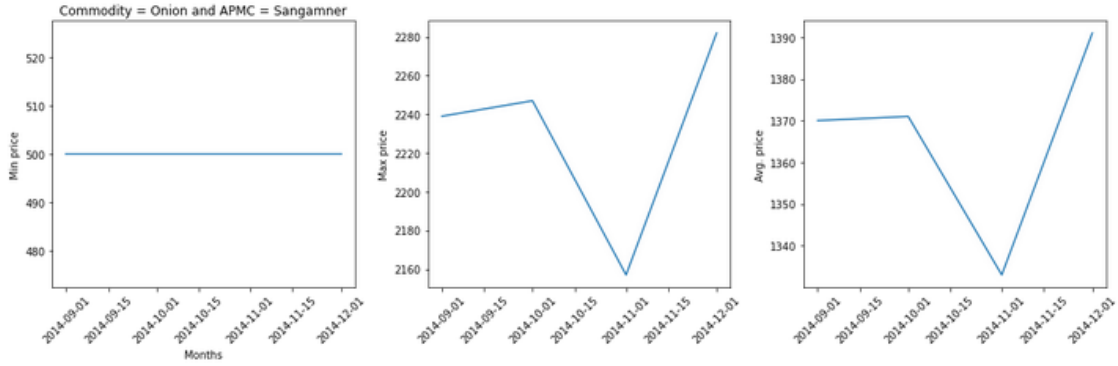

Commodity = Bajri and APMC = Jamkhed
Min price
Max price
Avg price
Months

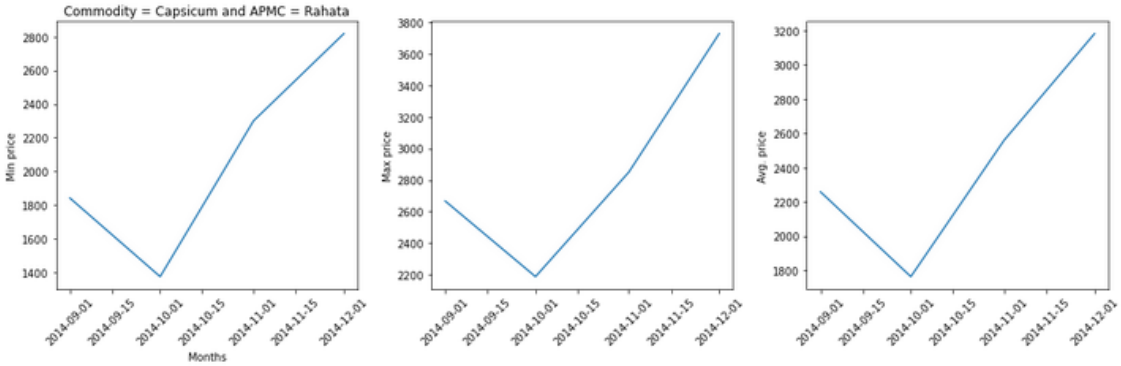

Commodity = Bajri and APMC = Sangamner
Min price
Max price
Avg price
Months

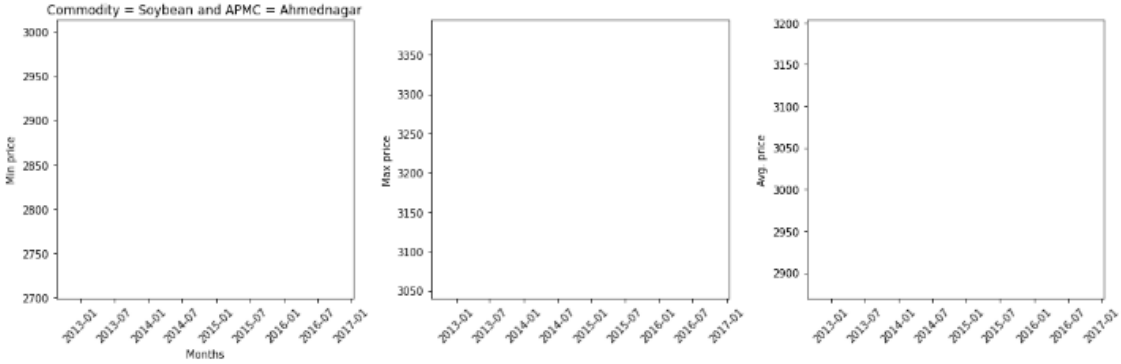

Commodity = Onion and APMC = Ahmednagar
Min price
Max price
Avg price
Months

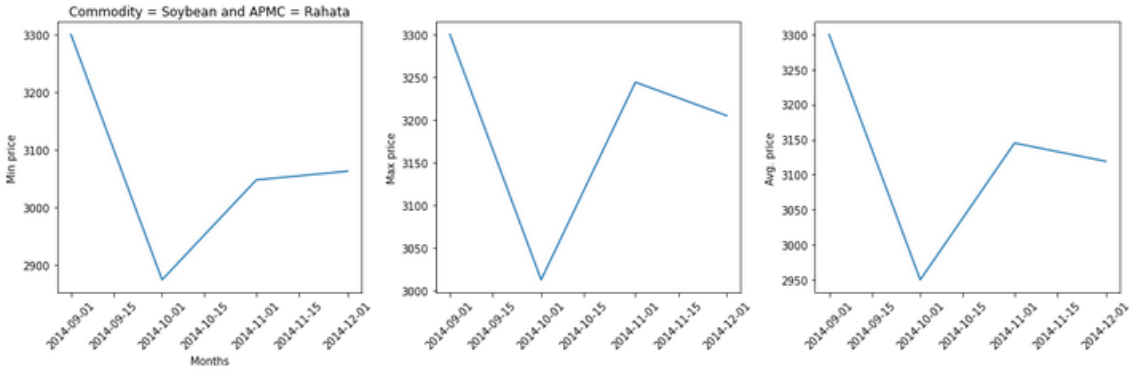

Commodity = Onion and APMC = Rahata
Min price
Max price
Avg price
Months

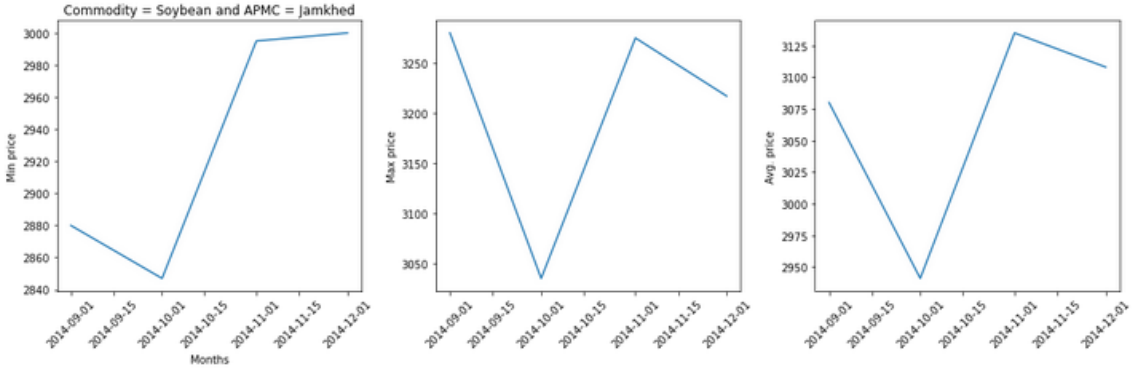

Commodity = Onion and APMC = Jamkhed
Min price
Max price
Avg price
Months

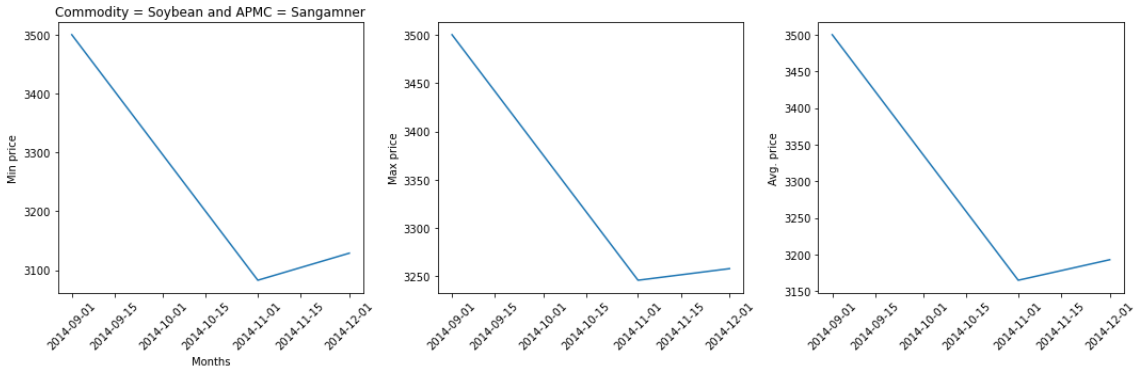

Commodity = Onion and APMC = Sangamner
Min price
Max price
Avg price
Months


Commodity = Capsicum and APMC = Rahata
Min price
Max price
Avg price
Months


Commodity = Soybean and APMC = Ahmednagar
Min price
Max price
Avg price
Months


Commodity = Soybean and APMC = Rahata
Min price
Max price
Avg price
Months


Commodity = Soybean and APMC = Jamkhed
Min price
Max price
Avg price
Months

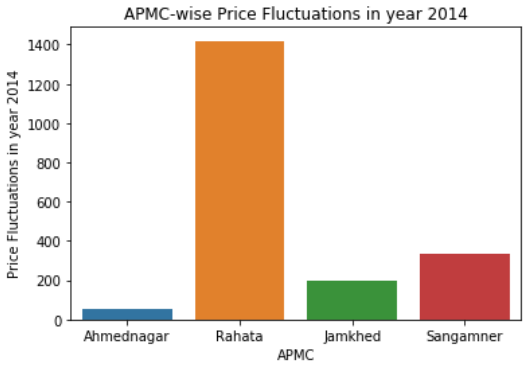


Note: There is no plot for Commodity: Soyabean and APMC: Ahmednagar since there is no prices information of Soyabean in the APMC.

In [143…
```
print(apmc_fluctuations)
print(comm_fluctuations)
```

```
{'Ahmednagar': 54, 'Rahata': 1419, 'Jamkhed': 194, 'Sangamner': 335}
{'Bajri': 191, 'Onion': 147, 'Capsicum': 1419, 'Soybean': 350}
```

In [112…
```
# Visualize Price Fluctuations based on APMCs.

ax = sns.barplot(x = list(apmc_fluctuations.keys()), y = list(apmc_fluctuations.values()))
plt.title('APMC-wise Price Fluctuations in year 2014')
plt.xlabel('APMC')
plt.ylabel('Price Fluctuations in year 2014')
plt.show()
```



Analysis: The above bar plot demonstrates that Rahata APMC suffers from maximum Price Fluctuations in the year 2014.

In [1]:
```
# Visualize Price Fluctuations based on Commodities.

ax = sns.barplot(x = list(comm_fluctuations.keys()), y = list(comm_fluctuations.values()))
plt.title('Commodity-wise Price Fluctuations in year 2014')
plt.xlabel('Commodity')
plt.ylabel('Price Fluctuations in year 2014')
plt.show()
```

Analysis: The above bar plot demonstrates that Capsicum commodity suffers from maximum Price Fluctuations in the year 2014.

So, we can conclude that **APMC: Rahata and Commodity: Capsicum suffer from the highest Price Fluctuations**.

Chapter 5

SOFTWARE DEVELOPMENT

5.1 Installing Anaconda in Windows

This tutorial will demonstrate how you can install Anaconda, a powerful package manager, on Microsoft Windows.

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. This is advantageous as when you are working on a data science project, you will find that you need many different packages (numpy, scikit-learn, scipy, pandas to name a few), which an installation of Anaconda comes preinstalled with. If you need additional packages after installing Anaconda, you can use Anaconda's package manager, conda, or pip to install those packages. This is highly advantageous as you don't have to manage dependencies between multiple packages yourself. Conda even makes it easy to switch between Python 2 and 3 (you can learn more here). In fact, an installation of Anaconda is also the recommended way to install Jupyter Notebooks which you can learn more about here on the DataCamp community.

This tutorial will include:

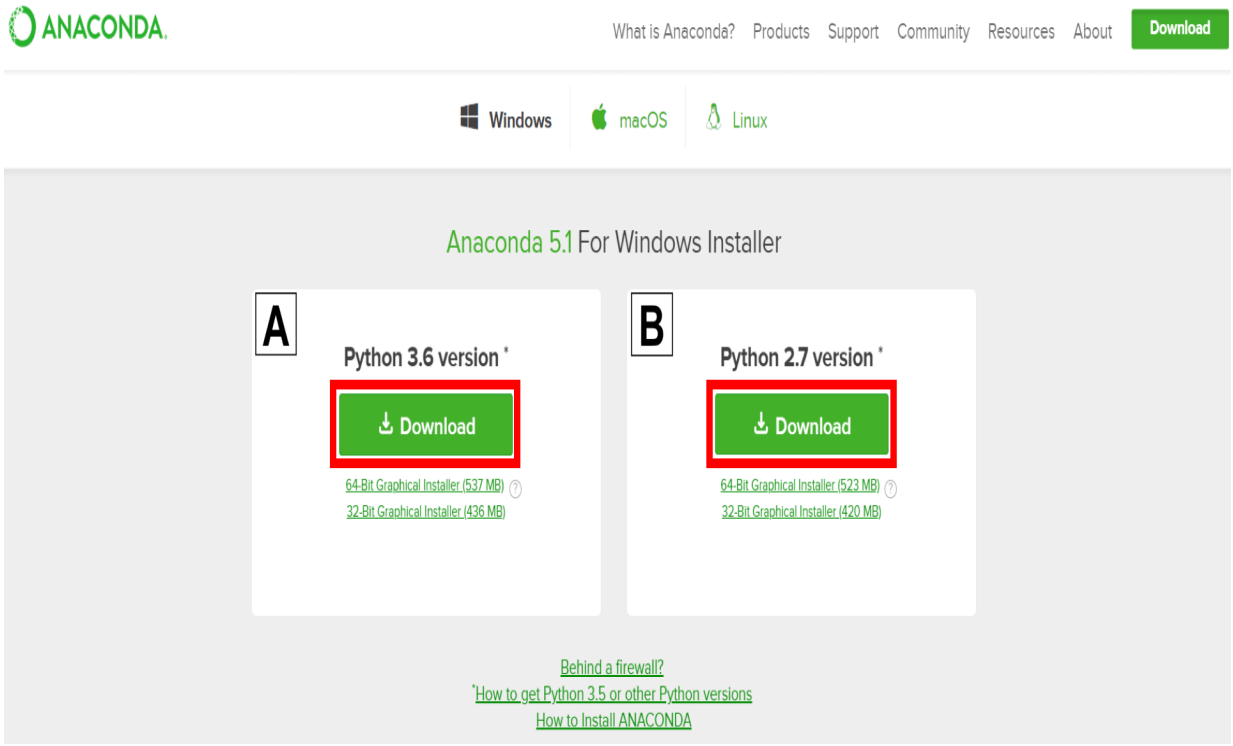
- How to Install Anaconda on Windows
- How to test your installation and fix common installation issues
- What to do after installing Anaconda.

Download and Install Anaconda

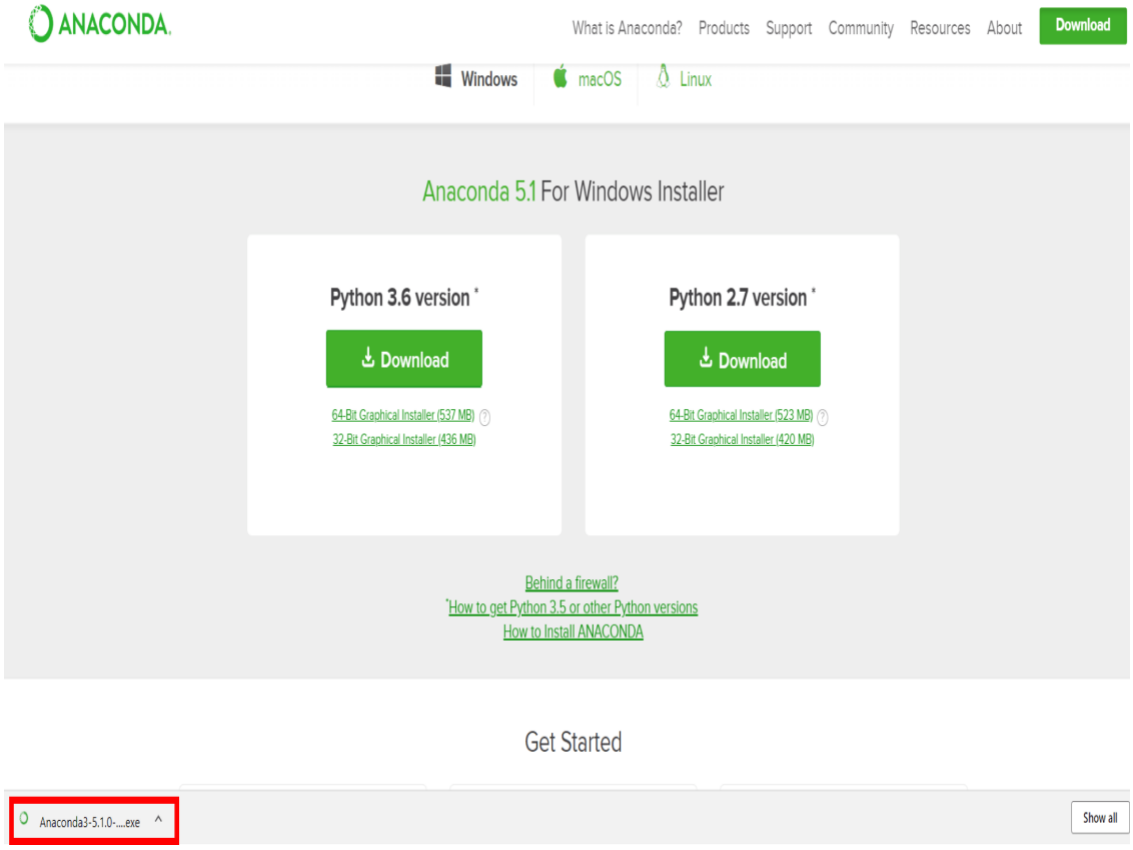
Go to the Anaconda Website and choose a Python 3.x graphical installer (A) or a Python 2.x graphical installer (B).

If you aren't sure which Python version you want to install, choose Python

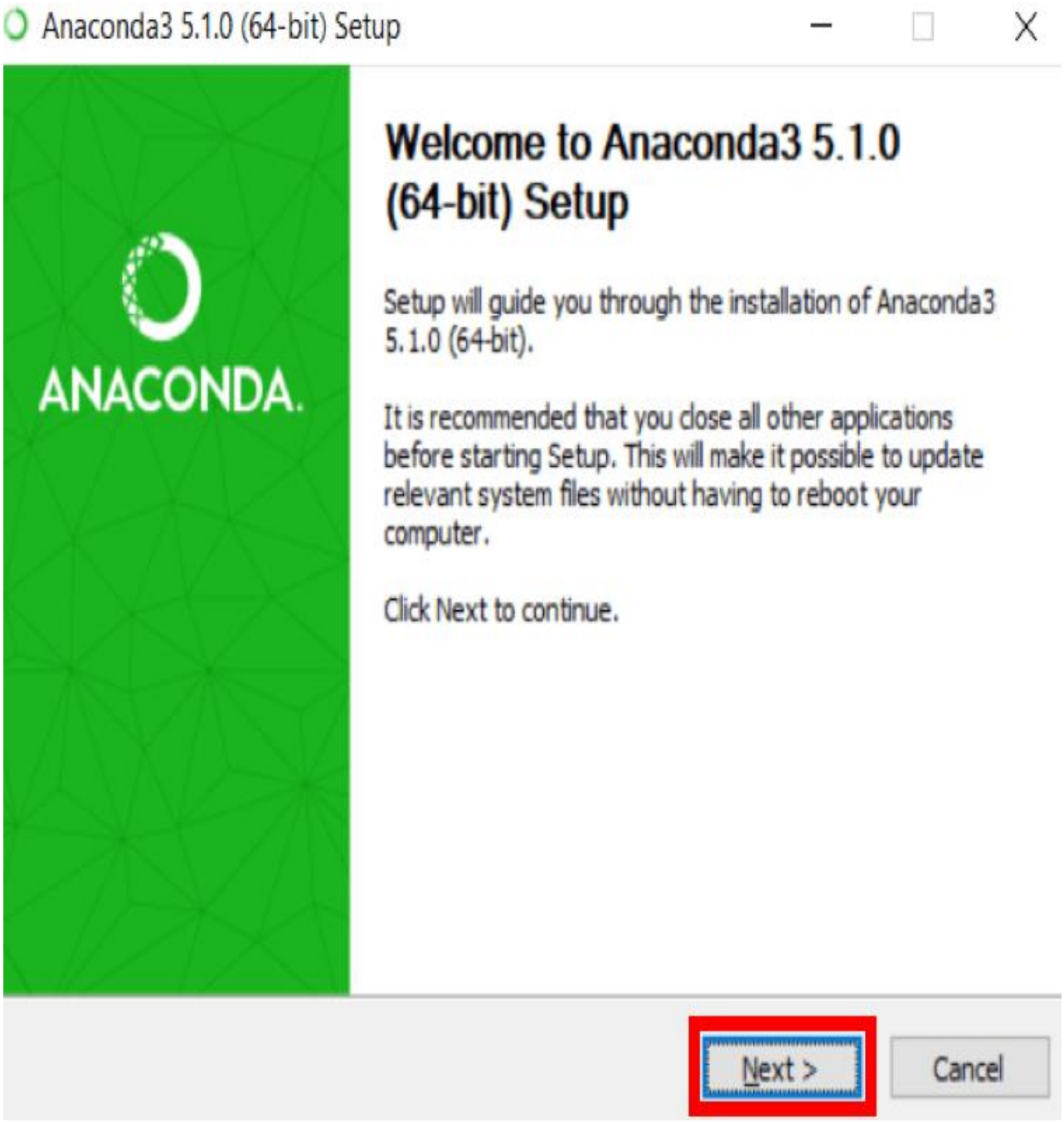
Do not choose both.



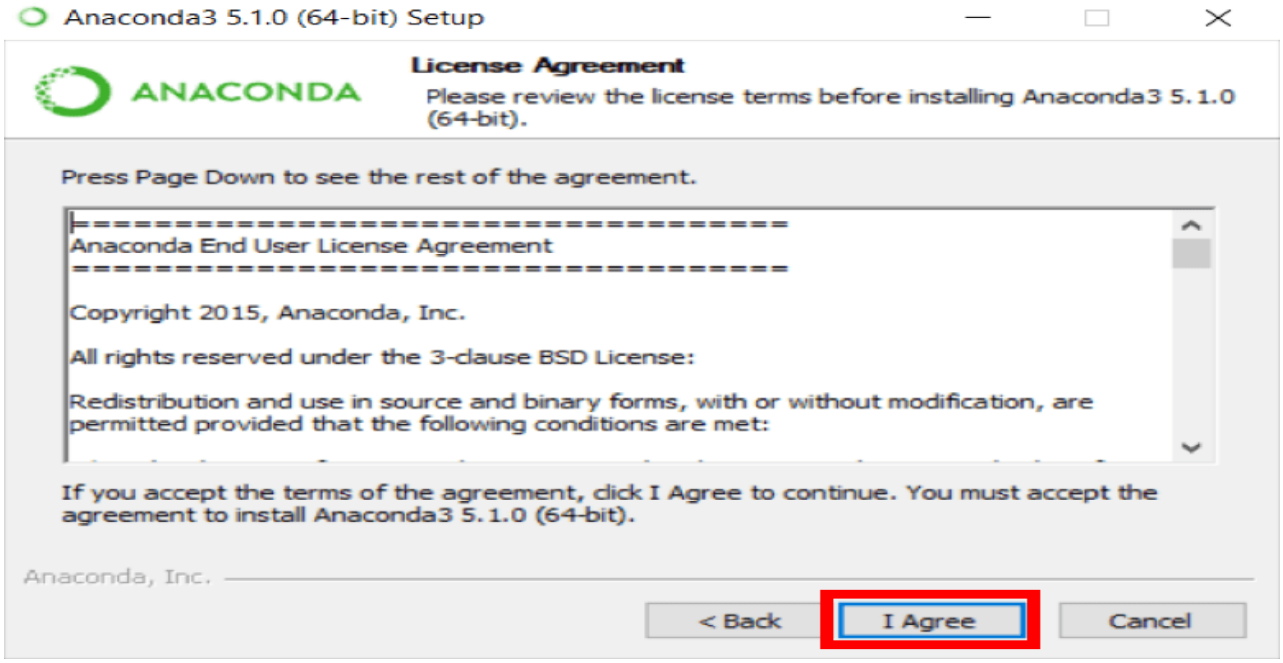
2. Locate your download and double click it.



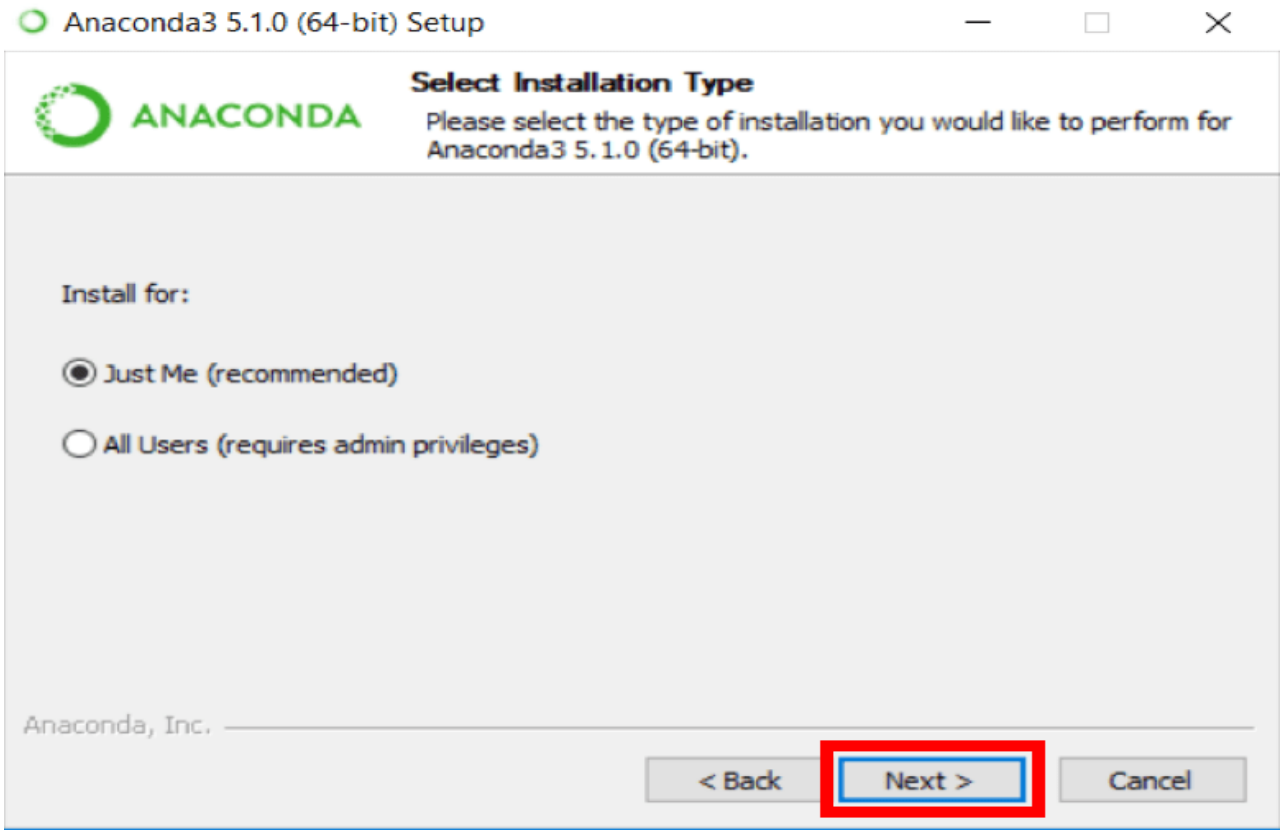
When the screen below appears, click on Next.



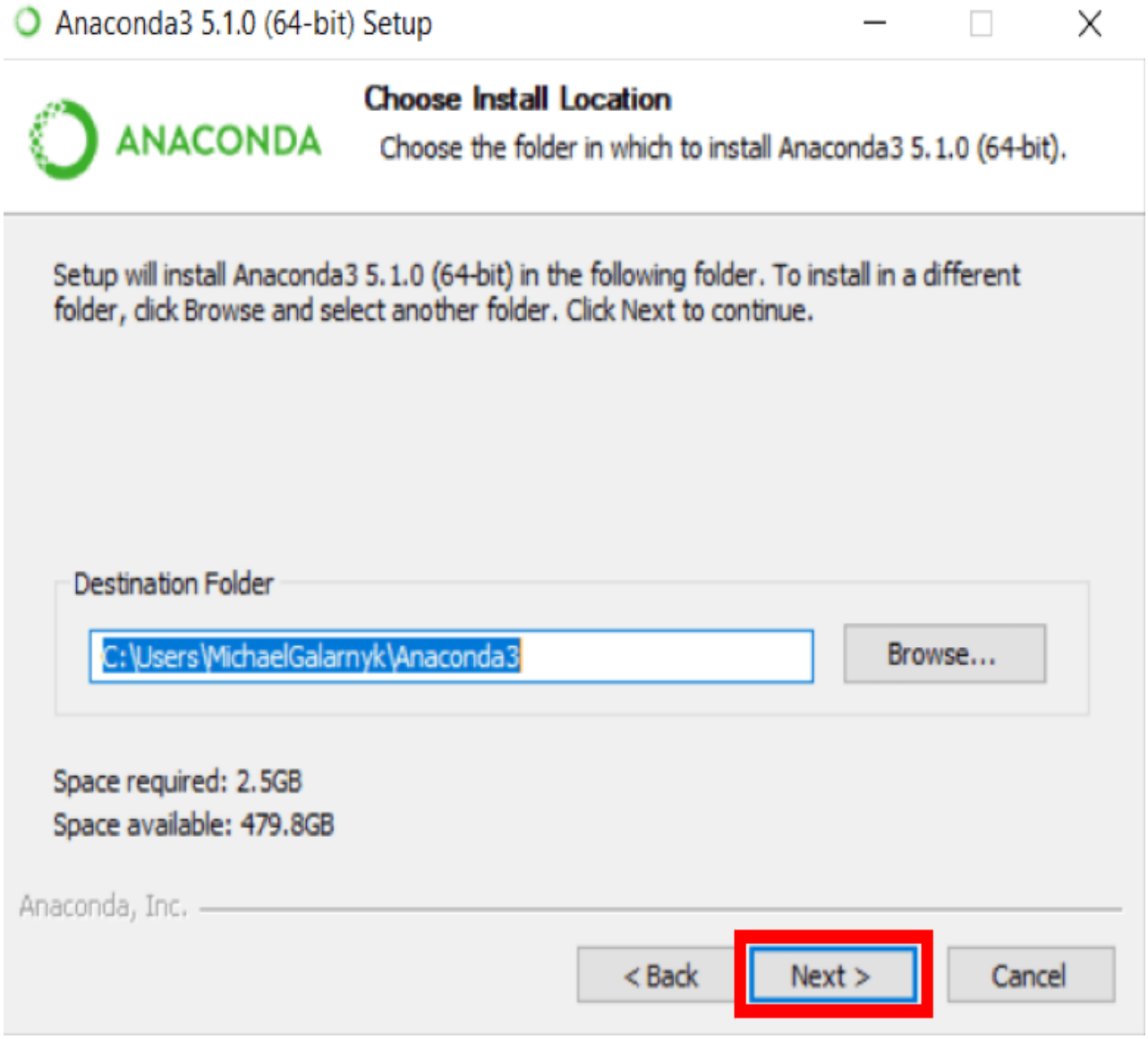
3. Read the license agreement and click on I Agree.



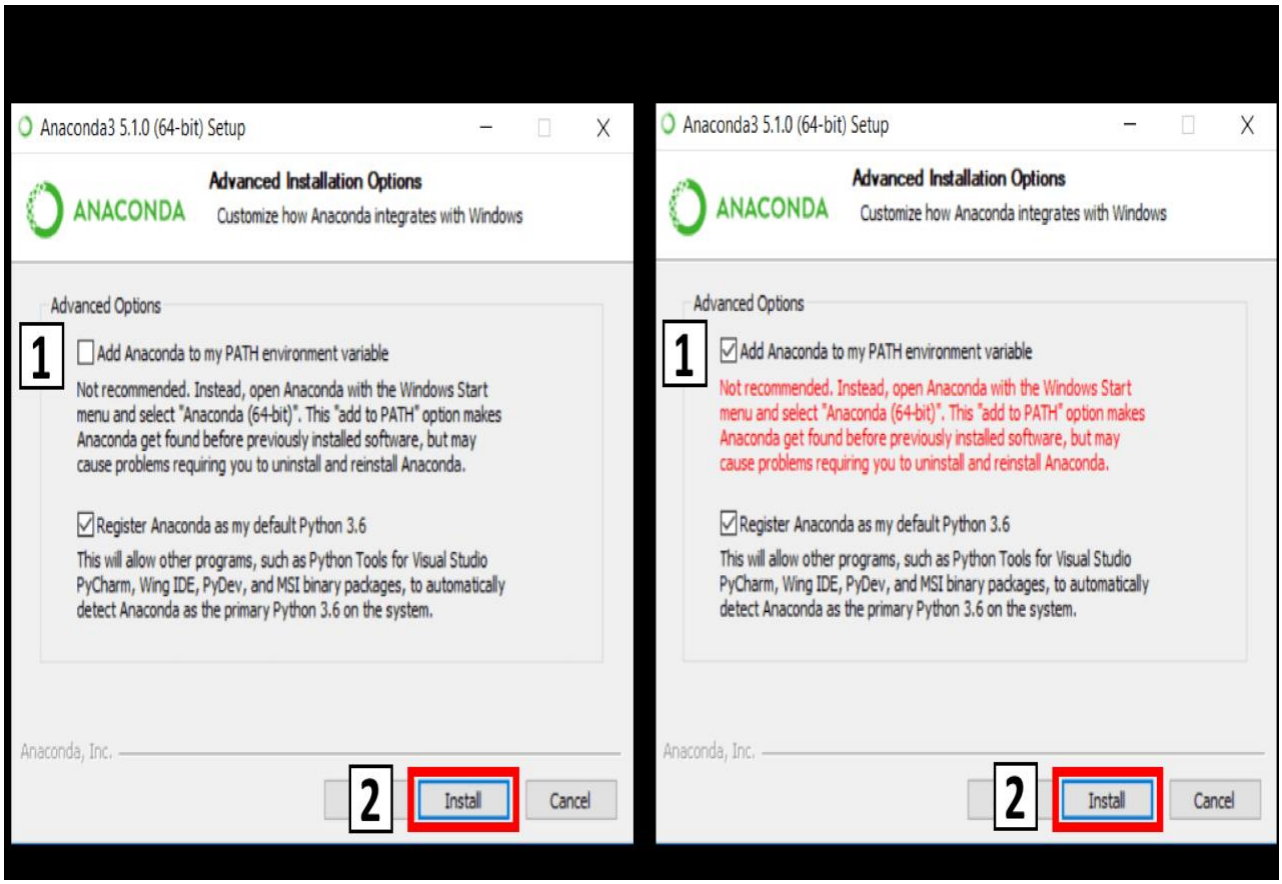
4. Click on Next.



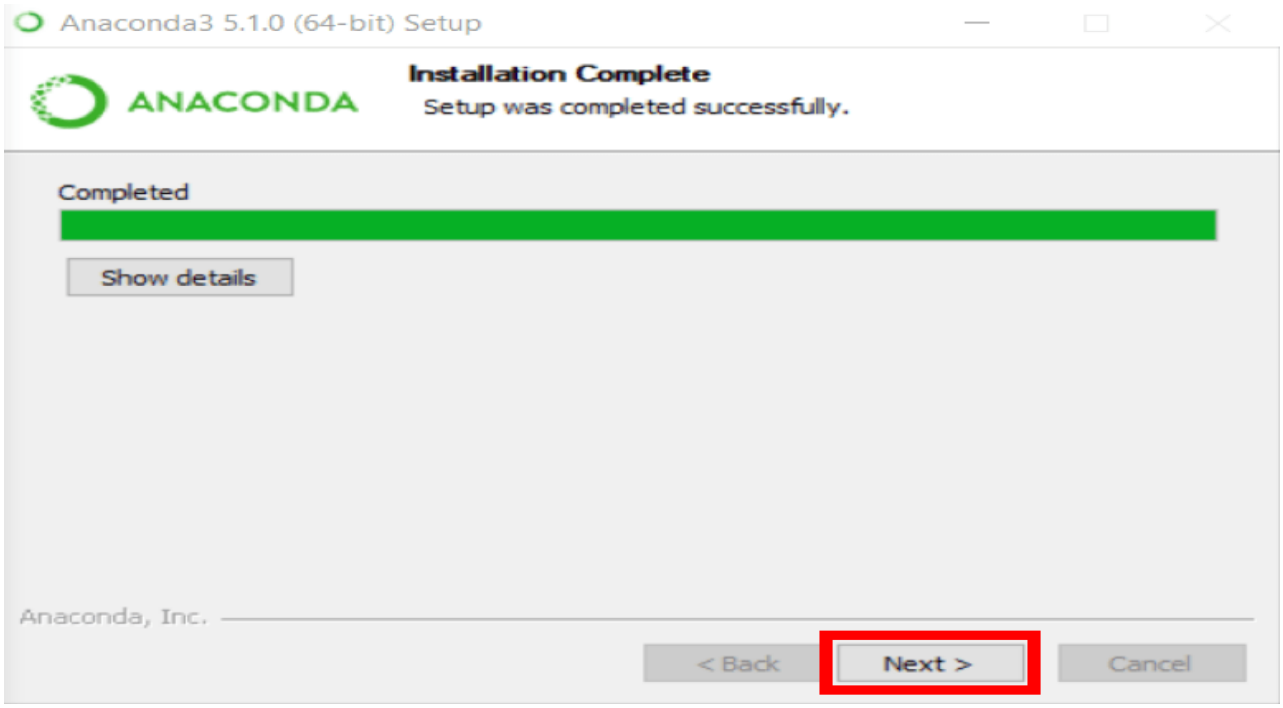
5. Note your installation location and then click Next.



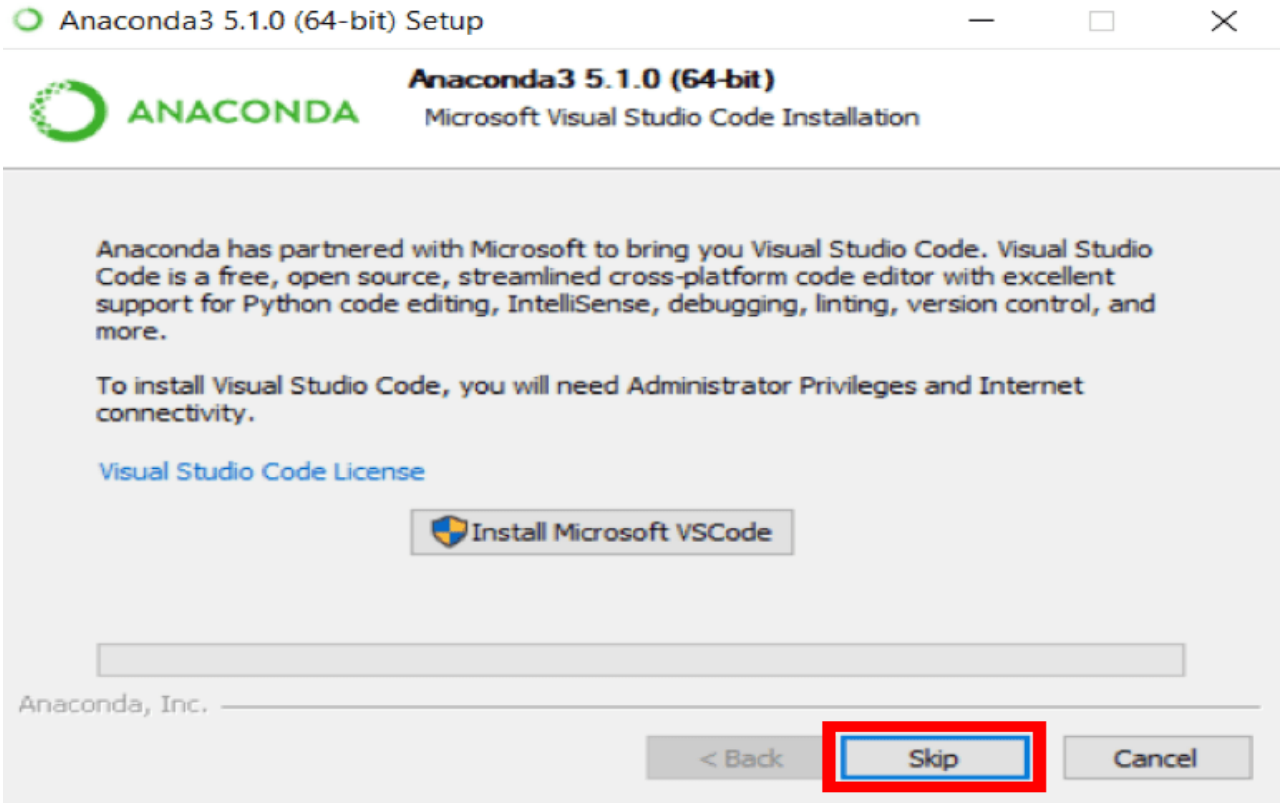
6. This is an important part of the installation process. The recommended approach is to not check the box to add Anaconda to your path. This means you will have to use Anaconda Navigator or the Anaconda Command Prompt (located in the Start Menu under "Anaconda") when you wish to use Anaconda (you can always add Anaconda to your PATH later if you don't check the box). If you want to be able to use Anaconda in your command prompt (or git bash, cmder, powershell etc), please use the alternative approach and check the box.



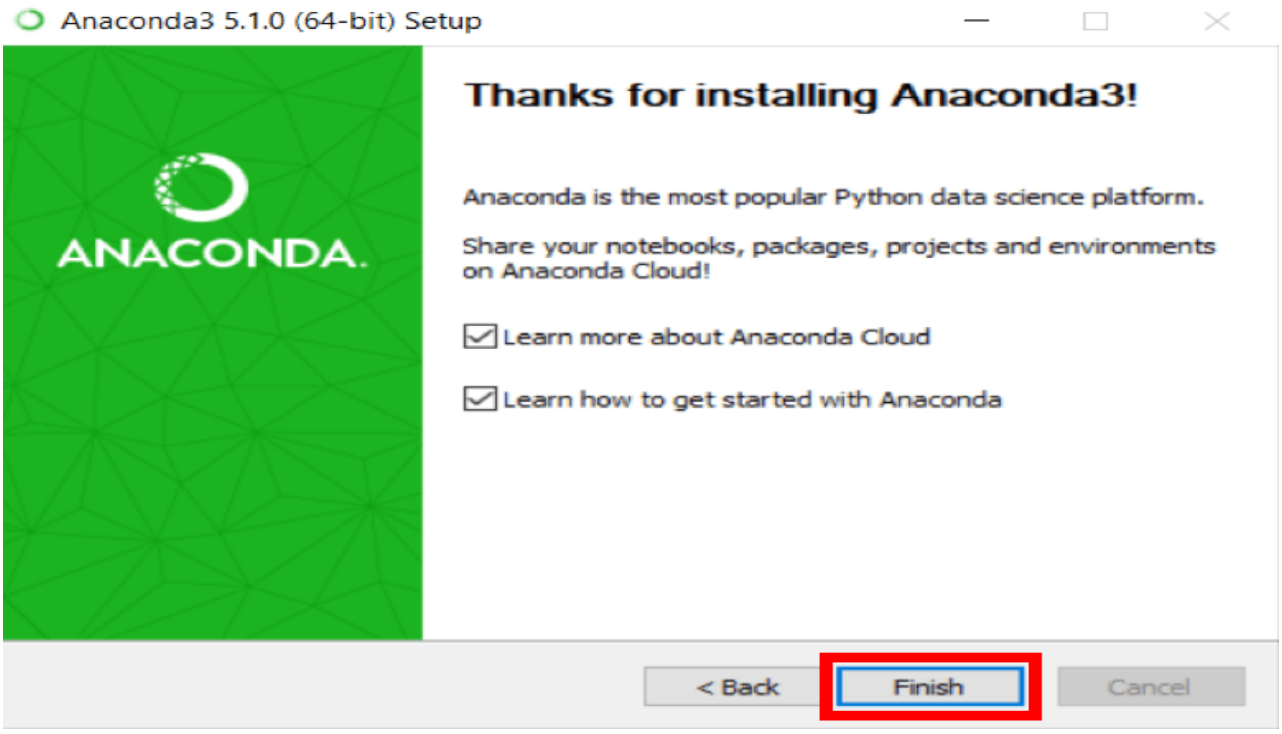
7. Click on Next.



8. You can install Microsoft VSCode if you wish, but it is optional.



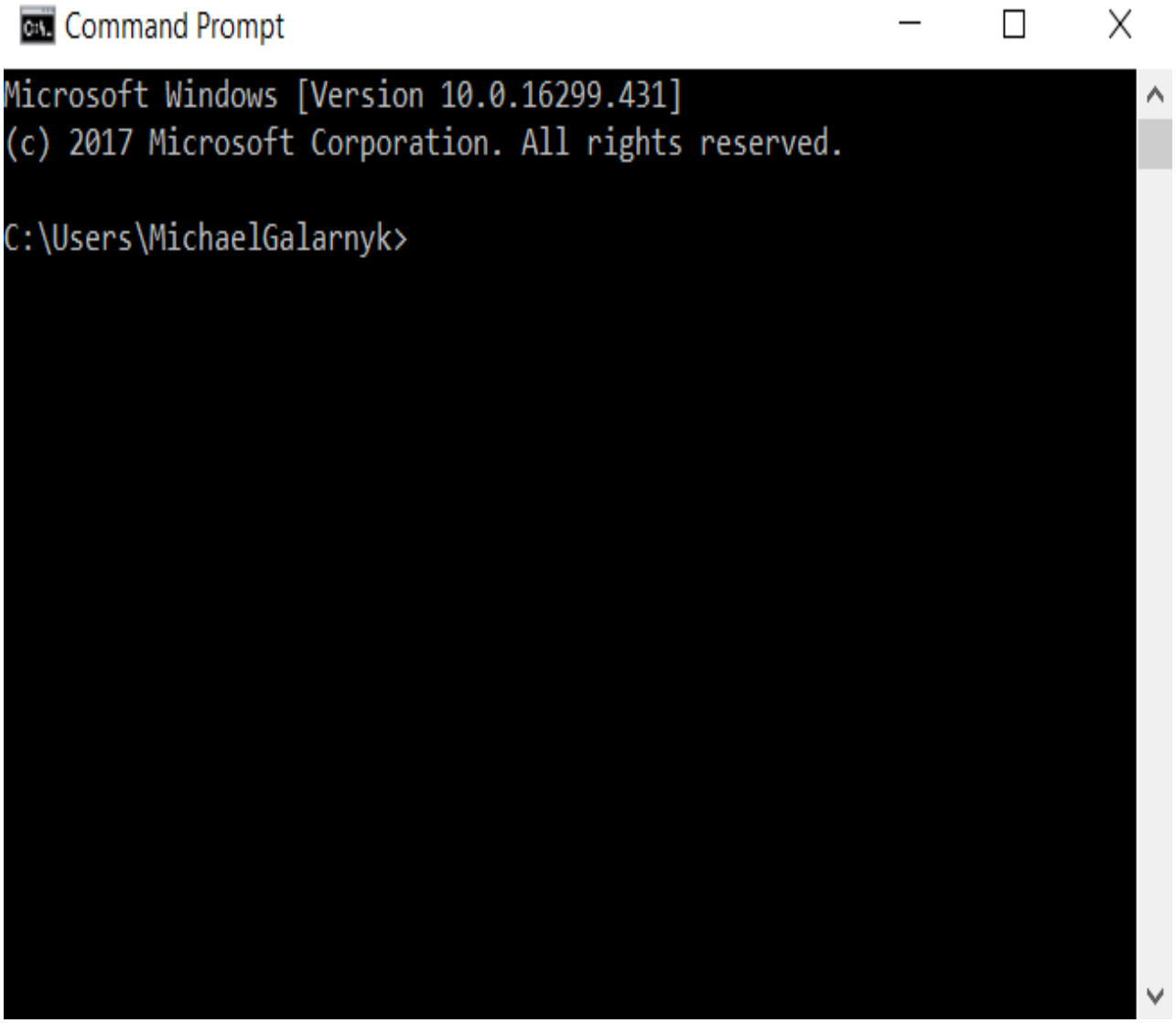
9. Click on Finish.



5.2 Add Anaconda to Path

This is an optional step. This is for the case where you didn't check the box in step 6 and now want to add Anaconda to your Path. The advantage of this is that you will be able to use Anaconda in your Command Prompt, Git Bash, cmder etc.

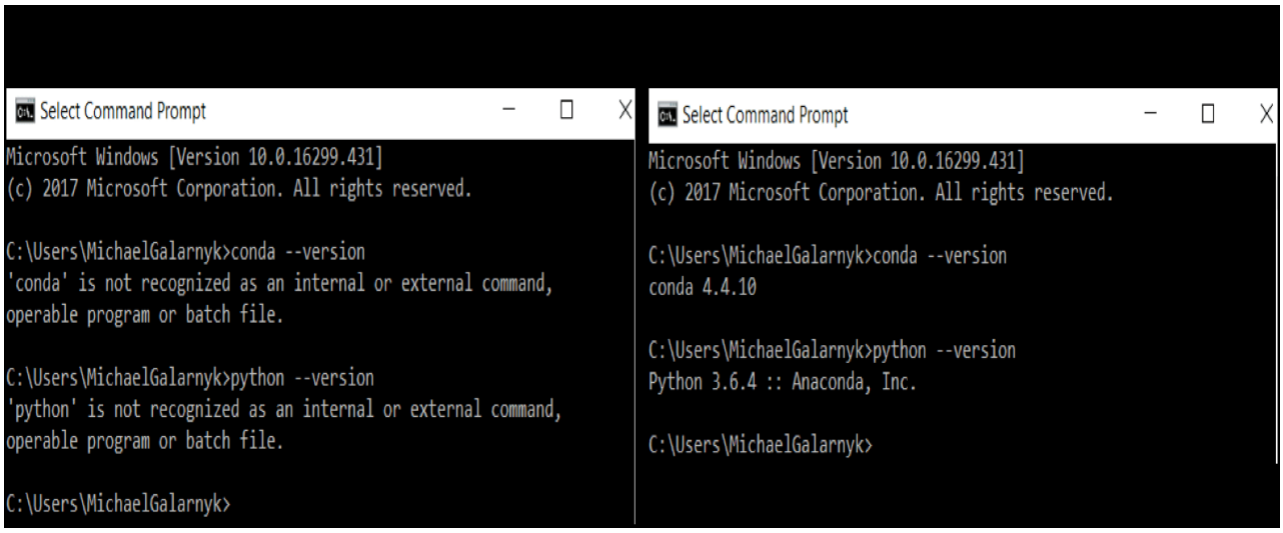
1. Open a Command Prompt.



2. Check if you already have Anaconda added to your path. Enter the commands below into your Command Prompt. This is checking if you already have Anaconda added to your path. If you get a command not recognized error like in the left side of the image below, proceed to step 3. If you get an output similar to the right side of the image below, you have already added Anaconda to your path.

conda --version

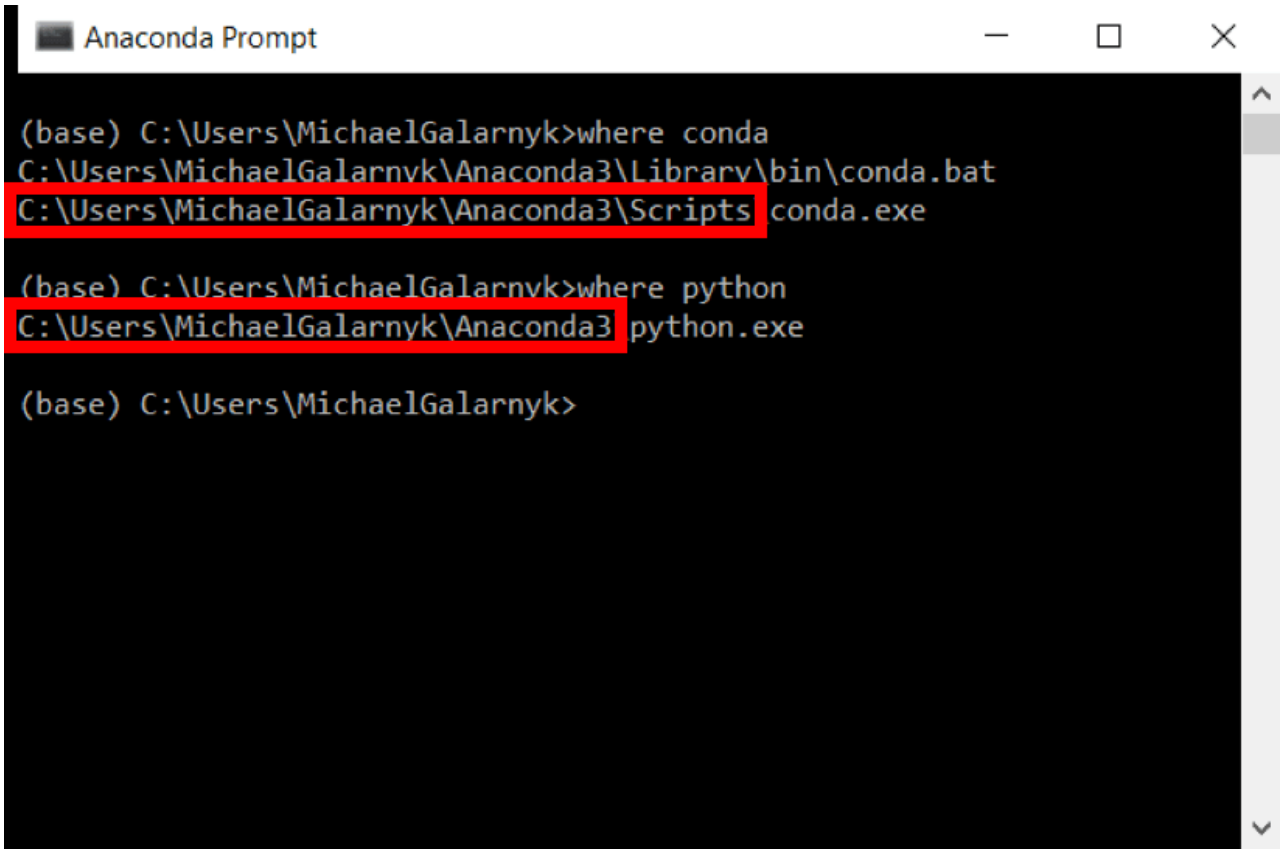
python --version



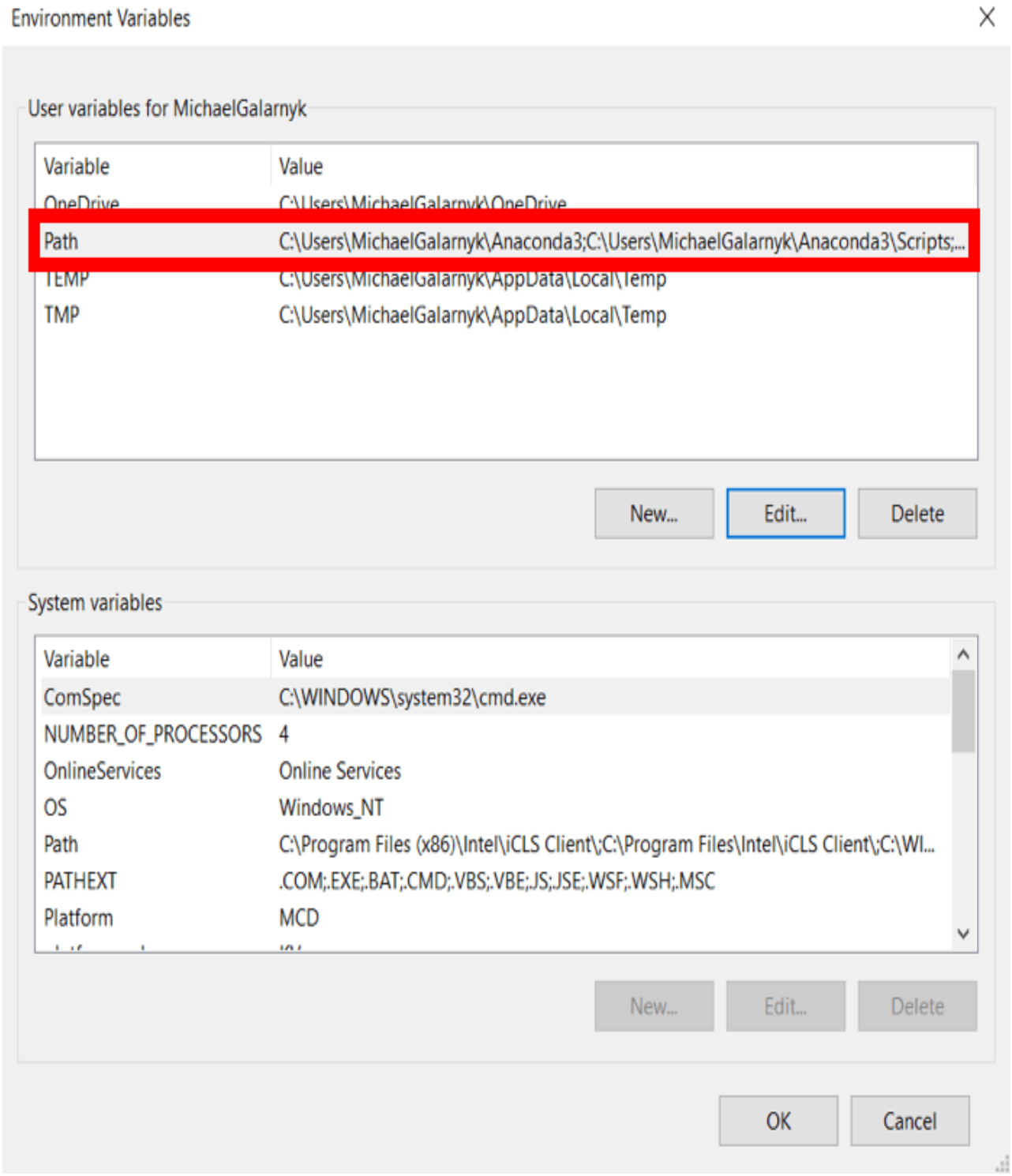
3. If you don't know where your conda and/or python is, open an Anaconda Prompt and type in the following commands. This is telling you where conda and python are located on your computer.

where conda

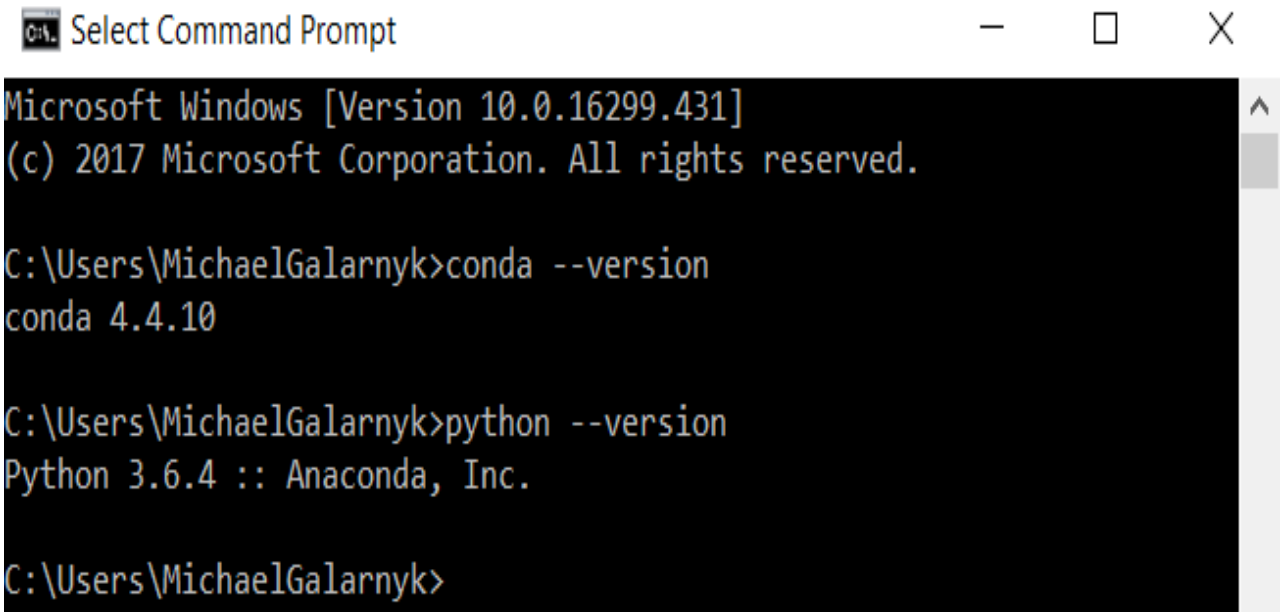
where python



4. Add conda and python to your PATH. You can do this by going to your Environment Variables and adding the output of step 3 (enclosed in the red rectangle) to your path. If you are having issues, here is a short video on adding conda and python to your PATH.



5. Open a new Command Prompt. Try typing conda --version and python --version into the Command Prompt to check to see if everything went well.



5.3 Anaconda Conclusion

This tutorial provided a quick guide on how to install Anaconda on Windows as well as how to deal with a common installation issue. If you would like to learn more about Anaconda, you can learn more about it here. If you aren't sure what to do to start coding on your computer, I recommend you check out the the Jupyter Notebook Definitive Guide to learn how to code using Jupyter Notebooks. If you want to learn about Python for Data Science, I suggest you check out the Data Camp course Intro to Python for Data Science.

5.4 What is Jupiter Notebook?

If you are a student, you must be using a class notebook to take various class notes, or if you are a business professional, you might be using a writing pad to pen down important notes, either for your purpose or to present to someone else. The typical content that goes in a student's notebook can be text in various formats such as **bold** and *italic*, or a table, a mathematical equation, or creative stuff like hand-drawn images and so on. Not to forget that if it's a programmer's notebook, it will also contain a lot of programming code.

Now, you want to continue with this practice of writing all such stuff in a single place online. Jupyter notebook comes to the rescue here. It is a web-based platform that allows you to write narrative text in various formats, include a table or an image, write equations and code in various programming

languages, all in one place. Apart from this, Jupyter notebook also allows you to write LaTeX code, include HTML code and embed a YouTube video.

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

You can refer to the below-given list to get an intuitive idea about how a Jupyter notebook looks like:

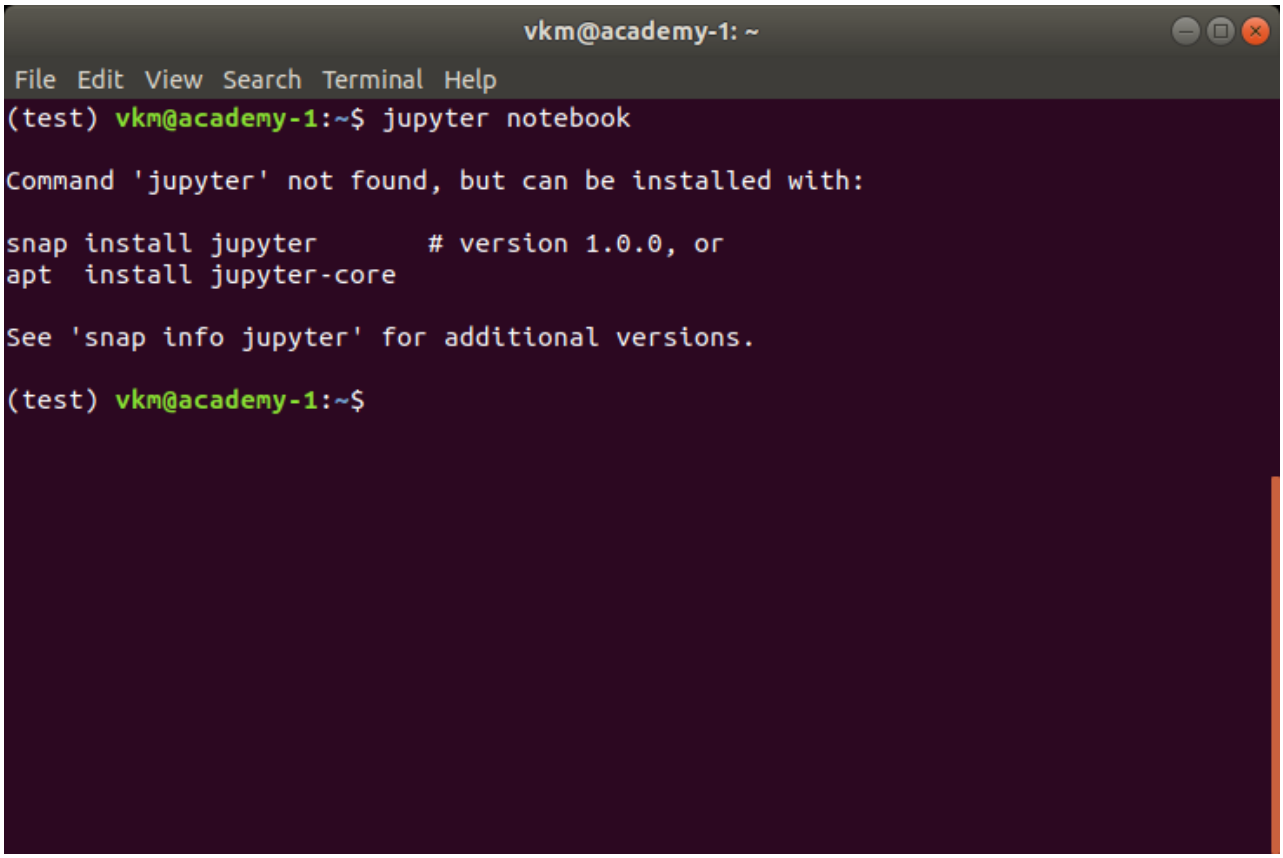
- Support Vector Classifier
- Trade Analysis
- Sun Pharma Vs HDFC Bank
-

A backdrop: The Jupyter Notebook is one of two facets of the Jupyter project which started to develop open-source, open-standards for interactive computing across dozens of programming languages. Another being JupyterLab which is the advanced version of Jupyter Notebook interface. However, both operate in a similar fashion.
Now that you have an idea about what Jupyter notebooks are, let's see its installation process.

5.5 How to install Jupyter Environment?

jupyter notebook

If the above command fails, you can continue reading this section. Otherwise, you can safely skip this and proceed to the next section. In case the command fails and you get the error similar (not exact) to the one shown below, continue with this section to understand the installation process.



Two popular methods using which Python can be installed on your workstation. They are

1. Installing Python using Anaconda Distribution
2. Installing Raw Python

While Jupyter runs code in many programming languages, Python is a requirement for installing the Jupyter Notebook.

5.6 Installing Jupiter Notebook using Anaconda

If you have installed Python using Anaconda Distribution, you are good to go. Anaconda Distribution includes Python, the Jupyter Notebook, and other commonly used packages for the scientific community.

If you don't have any version of Python installed, the recommended way to install Python is using Anaconda Distribution. It should be pretty simple to get Python installed.

- First, download the latest version of Anaconda Distribution.
- Second, install the downloaded version of Anaconda.

Bam! You have installed Jupyter Notebook. To check whether the installation is successful or not, and run the Jupyter Notebook, run the following command in the Anaconda prompt or command prompt (Windows) or terminal (Mac/Linux).

jupyter notebook

If you get an error upon running the above command (which should not happen), try the following method.

5.7 I Installing Jupiter Notebook using PIP

When you install Python directly from its official website, it does not include Jupyter Notebook in its standard library. In this case, you need to install Jupyter Notebook using the pip. The process is as follows:

1. Open a new command prompt (Windows) or terminal (Mac/Linux)
2. Execute the following command to install Jupyter Notebook

python -m pip install jupyter

or if you are using Python3

python3 -m pip install jupyter

or simply

pip install jupyter

Congratulations, you have installed Jupyter Notebook! Onward to running it.

5.8 How to Run or Open Jupiter Notebook?

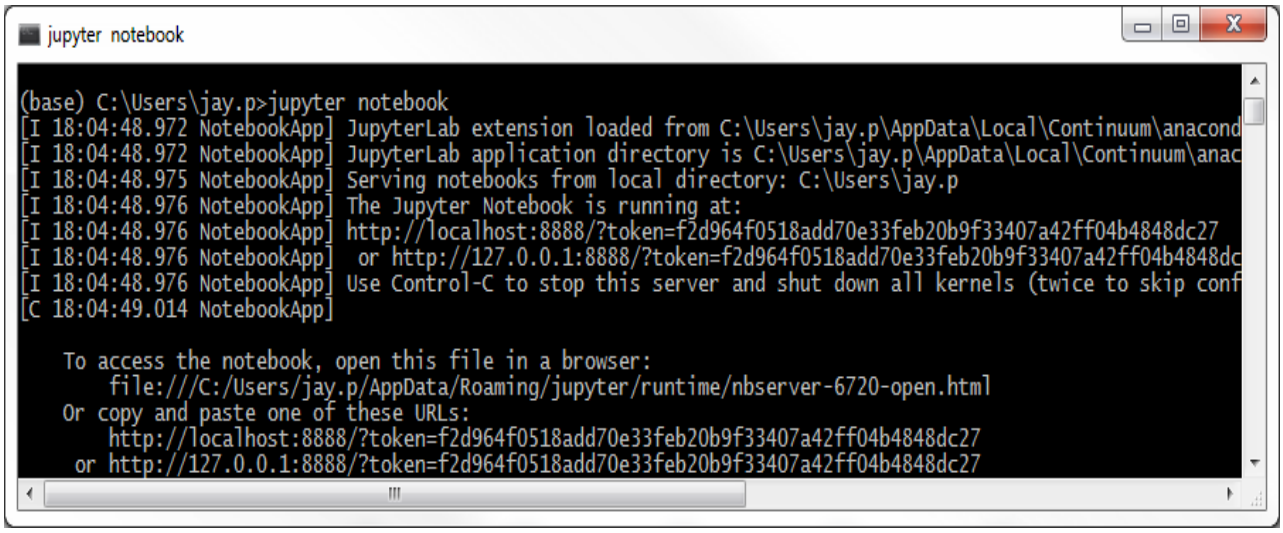
After you have installed the Jupyter Notebook on your computer, you are ready to run the notebook server.

If you are reading this article from the beginning, then you might already know how to run Jupyter Notebook. However, if you have skipped the previous sections and directly jumped here, the below mentioned steps shows how to run the Jupyter Notebook.

First, open a new command prompt (Windows) or terminal (Mac/Linux) on your workstation, and second, execute the following command:

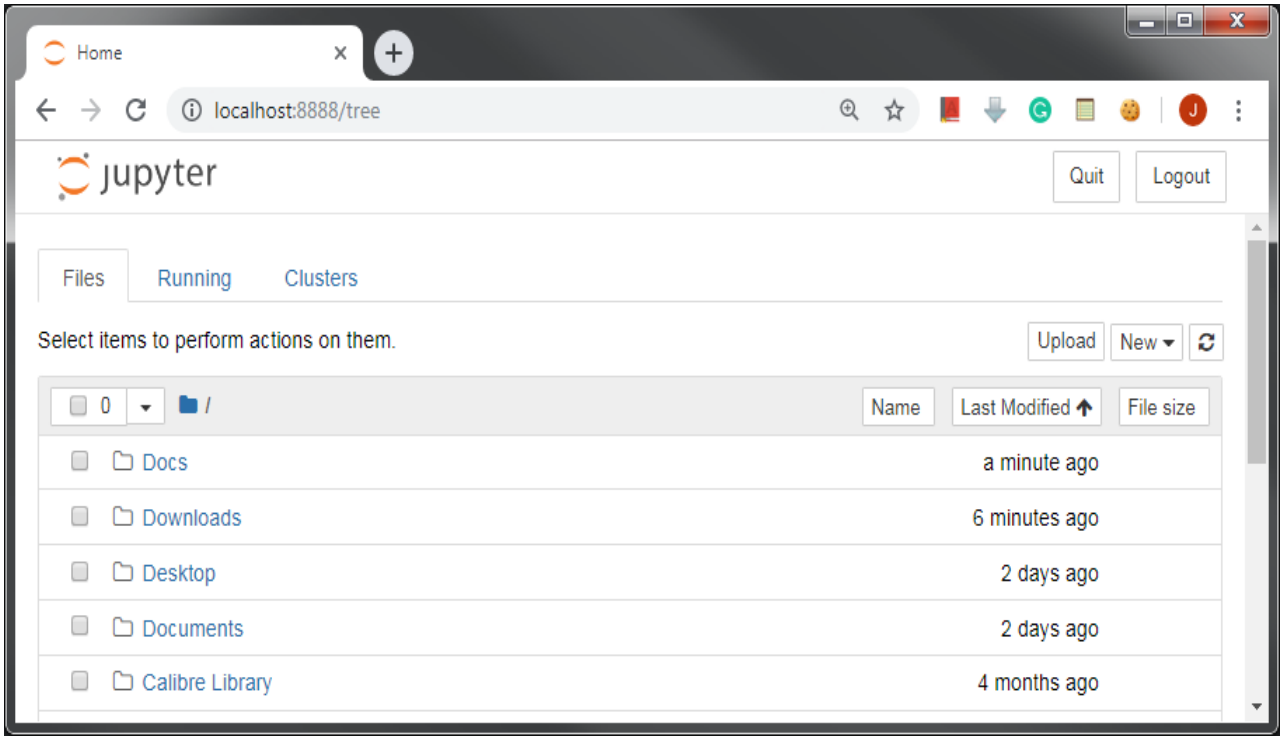
jupyter notebook

Upon executing the above command, the terminal or command prompt will print some information about the Jupyter Notebooks being loaded. It might look something like as shown in the below snapshot. Be mindful that the information printed would be different for each workstation.

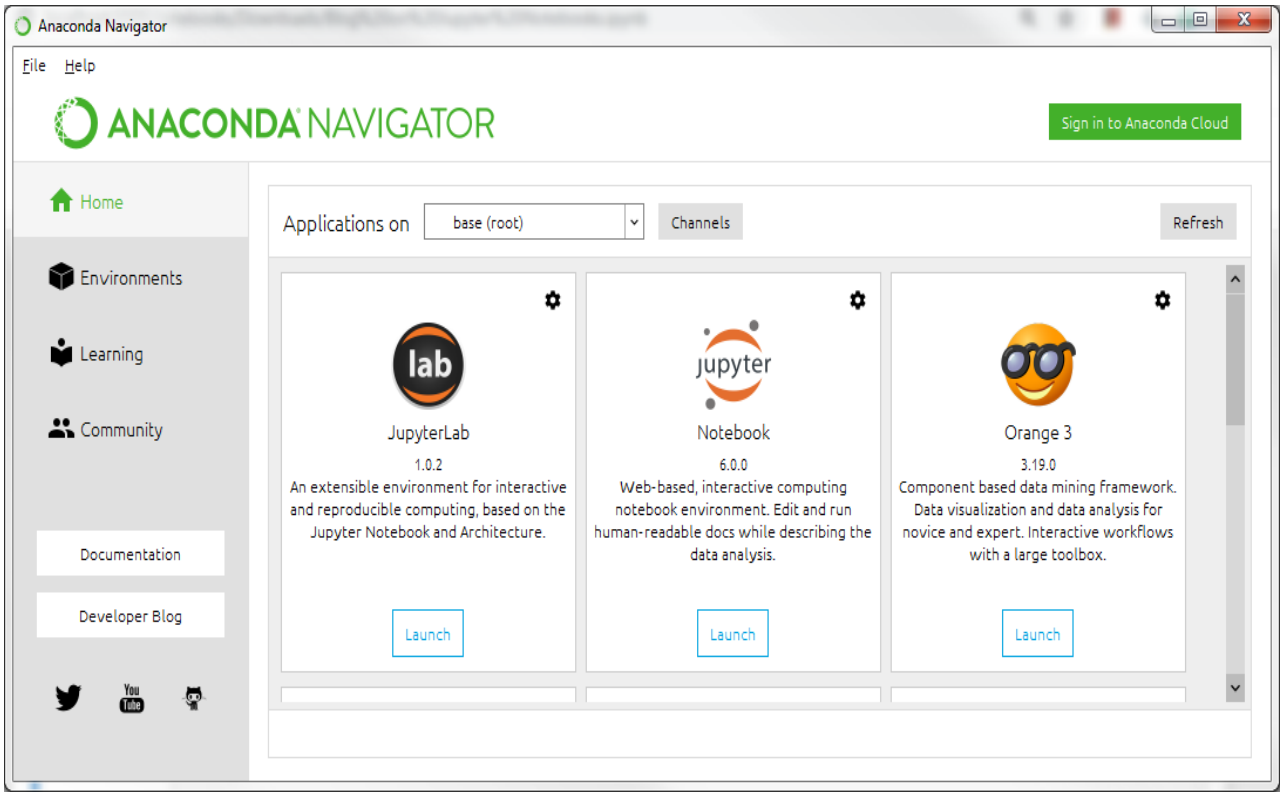


Keep the terminal open as it is. It will then open the default web browser with the URL mentioned in the command prompt or terminal.

When the notebook opens in your browser, you will see the *Notebook Homepage* as shown in the below snapshot. This will list the notebook files and subdirectories in the directory where the notebook server was started.



In case you are using Anaconda distribution for Python, you can open Anaconda Navigator (using the Start Menu(Windows), Applications folder(Mac), or Softwares folder(Linux)) shown below which allows you to open Jupyter Notebook using point and click.



Once the Navigator application is open, you can click on the Launch button within the Notebook dialogue to launch the Jupyter Notebook application. Upon clicking the Launch button, you will be presented by the homepage that we'd seen earlier.

Now, let's understand how Jupyter environment works, I won't be going technical, though. As the Jupyter Notebook is a web application, it works on a server-client architecture. When you execute the command jupyter notebook, the Jupyter software starts the server locally in the console where the command is executed, and the Jupyter Notebook homepage that opens in the web browser works as the client. Whatever you perform, that is, create or upload a new notebook, or save the existing one, the client notebook on which you are working, will keep communicating with the server running in the console/command line.

To keep notebooks running smoothly, we need to keep the command prompt or terminal open, even after it has opened homepage. If you close it, notebooks that you are working with, won't

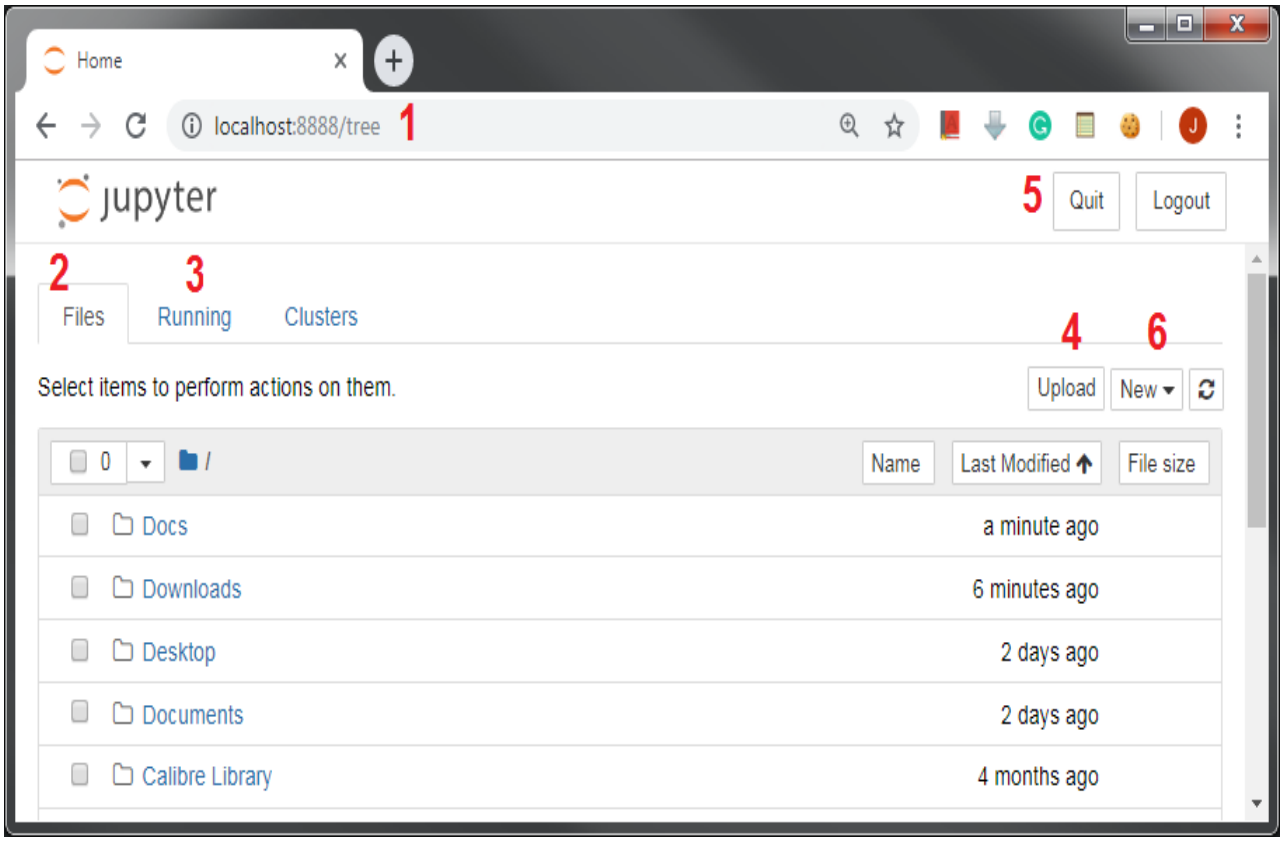
be able to communicate with the local server, and hence, any work you do, will not be saved on persistent memory.

The next step is to learn various components of Jupyter software.

What are the different components of Jupyter notebooks?

I assume that you are following the chronological order of this article. If so, then you have learned what Jupyter notebooks are, how to install it, and how to run and open it. If not, then I would recommend you to go through them to get an overall picture. Nevertheless, if you are already familiar with those parts, go on and keep learning.

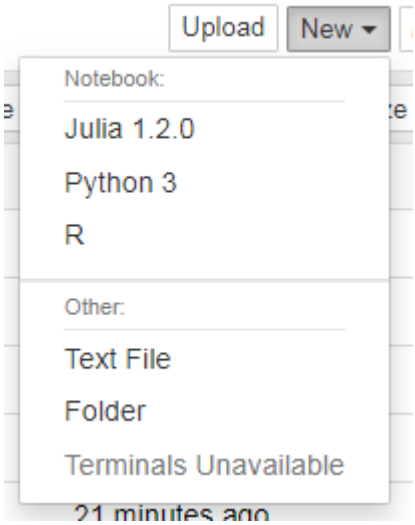
In this section, I will explain the various components of Jupyter software. When you start Jupyter Notebook application, and you'll be presented with the homepage. Let's start exploring it. Below shown is the snapshot of the homepage that you'd seen earlier, but with numbers assigned to each component to make our learning easier.



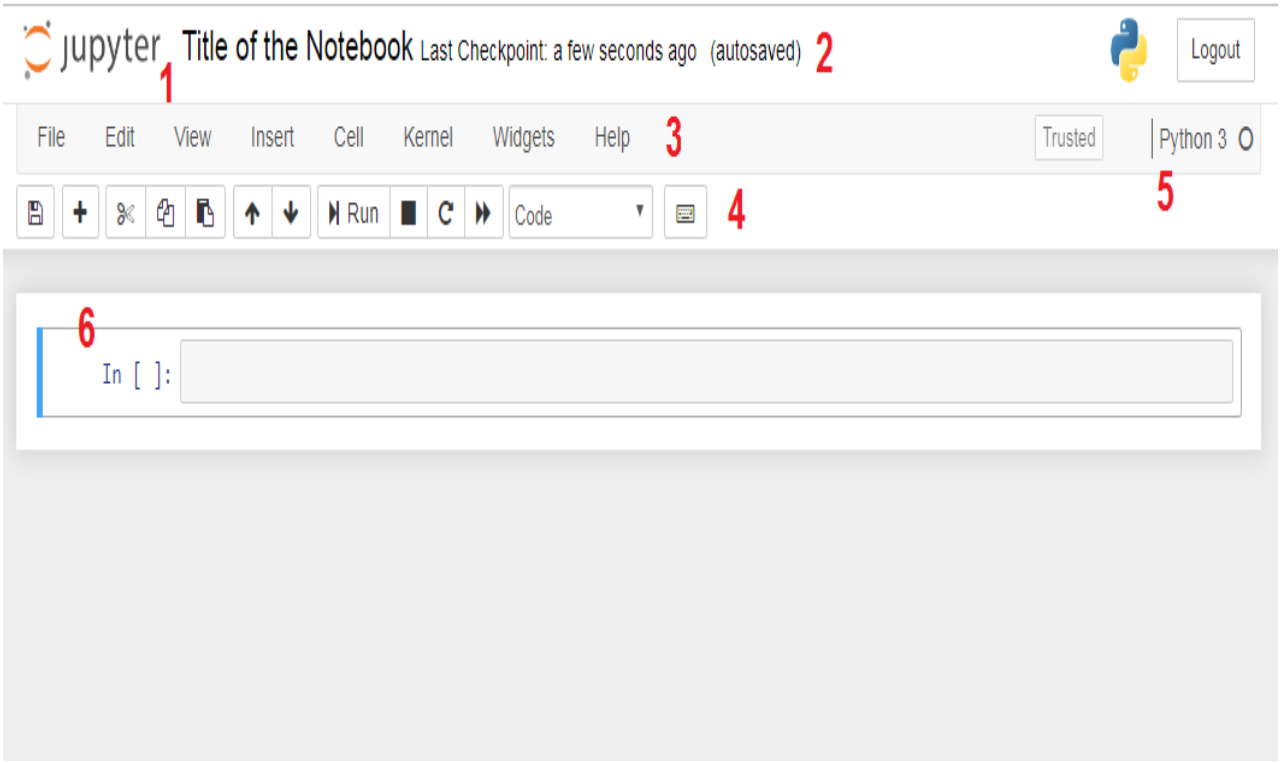
Here's the description of each numbered point shown in the snapshot.

1. It is the *URL* on which Jupyter server is running. If you are running the Jupyter on localhost, this would be the same URL shown in the console when you started Jupyter software.

2. The *Files* tab lists the directories and files in the home folder, which usually is the home directory of the user logged in to the computer.
3. The *Running* tab shows you a list of all open notebooks. When you start a new notebook or open an existing notebook, a kernel will get attached to it. All such running kernels will be listed under this tab.
4. If you want to open an existing Jupyter notebook(that ends with .ipynb extension), it needs to be listed in the Files tab. If it is not listed, you need to upload it using the *Upload* button which will open a file browser for you to load the file.
5. *Quit* and *Logout* buttons allow you to logout and shutdown the server. When you quit, all the opened notebooks will be closed, and the server will get shutdown.
6. The *New* button allows you to create a new notebook, text file, folder, or terminal. The snapshot shown below shows that you can create a notebook in Python, Julia, and R language. The notebook that you create will be associated with a respective kernel. When you install Jupyter environment as shown in the previous section, it is very likely that you will have only one option for the kernel, that is, Python. In order to add new languages, you can refer to this link.



You can create a new notebook by clicking on the respective language name. Regardless of what language you choose, the new notebook that you create will have the same appearance. The difference would be in terms of the kernel attached to it. If I click on *Python 3* on the dropdown opened, a new notebook with Python kernel attached to it will be created. The empty Jupyter notebook is shown in the snapshot below. *(Obviously without numbers :p)*



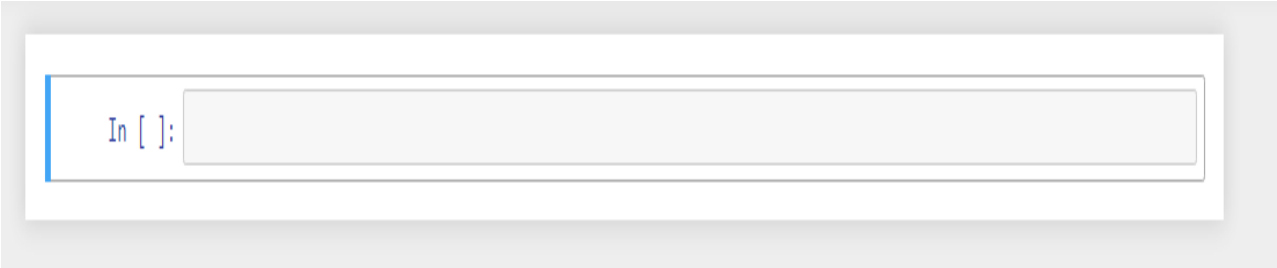
The newly created notebook has various components which are explained below:

1. The *title* is the name of the notebook. The title you set becomes the file name for the notebook, and it will have the extension as *.ipynb* which stands for *IPython NoteBook*.
2. The *checkpoint* shows you the time when your notebook was saved last.
3. The *menu bar* lists various menus that allow you to download the notebook(in multiple formats), open a new notebook, edit the notebook, customise the headers, manipulate cells, nudge the kernel, access help and so on.
4. The *shortcut bar* lists commonly used shortcuts such as *save* to save the notebook, *add a cell*, *cut*, *copy* to manipulate cells, *up* and *down* to navigate between cells, *run* to execute the cell, and so on. Any extension that you add to Jupyter will have its shortcut on this bar. We will learn what extensions are in the latter part of this article.
5. The *Kernel* shows you the current kernel associated with the notebook. In our case, the kernel is *Python 3*. The circle beside the kernel shows the status of the kernel. The hollow circle represents that it is ready to take input and run a cell. When a kernel is executing code or processing anything, it changes to solid.
6. A *cell* is the part of a notebook where all the magic happens. Cells are explained in detail in the following section.

5.9 What are cells in a Jupiter Notebook?

Any text or code that you write goes in the cell. Cells are the building block of any Jupyter notebook. Cells operate in two modes: *command* and *edit* mode, and they are of mainly three types: *code*, *markdown*, and *Raw NBConvert*.

The *command* mode allows you to manipulate cells. That is, the action you perform has to do with the cell as a whole. The command mode is represented by a grey border around the cell with a blue indication, as shown in the below snapshot.

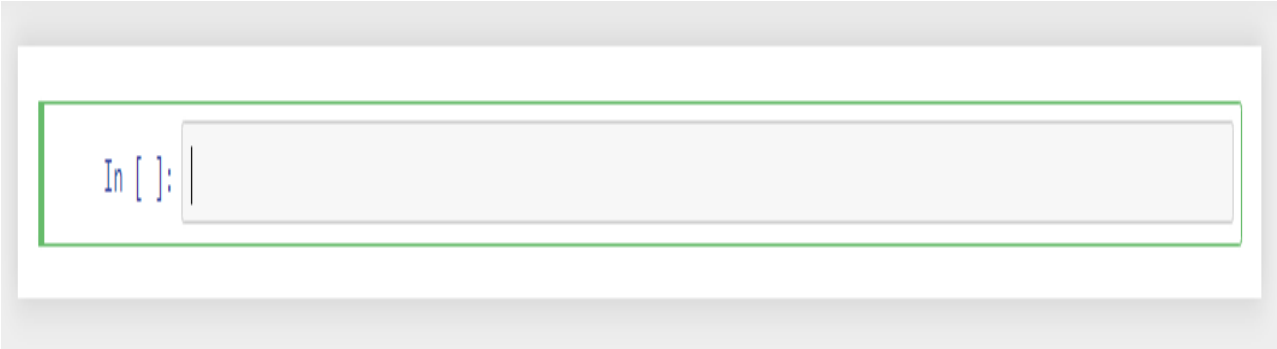


Some of the operations (along with their shortcuts) that you can perform when a cell is in the command mode are as follows:

- *Insert* a new cell -- Key in A to insert a new cell above, and B to insert a new cell below the current cell.
    - When a new cell is inserted, it will be a code type cell, by default. We will look at various types in a while.
- *Merge* existing cells -- Shift-M allows to merge selected cells or to merge the current cell with the cell below the current cell.
- *Copy* cells -- C copies selected cells.
- *Cut* cells -- X cuts selected cells.
- *Paste* cells -- Use Shift-V to paste cells.
- *Delete* existing dells -- Pressing D, D deletes the current cell. Be careful with this shortcut.
- *Change* the type of a cell -- The shortcut Y changes the cell type to *code*, M changes the cell type to *markdown*, and R changes the cell type to *raw*.
- *Convert* the cell to a heading -- There are six heading types in a Jupyter notebook. This works with *markdown* cell type only. Heading 1 is the largest heading and heading 6 is the smallest heading. 1, 2, 3, 4, 5, and 6 are used to change the cell type to the respective heading size.

- *Find and Replace* in existing cells -- Pressing F opens find and replace dialogue box.
- *Save* and mark the *Checkpoint* of the notebook -- Use Shift-S to save the notebook.
- *Toggle* line numbers in a cell -- Pressing L toggles line number in the current cell.
- *Toggle* the output of a cell -- O allows you to toggle the output of the current cell.
- *Interrupt the kernel* -- Keying I, I interrupts the kernel. That is, if any process is being executed by the kernel gets stopped.
- *Scroll* the notebook -- Space scrolls the notebook down, Shift-Space scrolls the notebook up.
- *Enter* the edit mode -- Pressing return key changes the mode of a cell to the edit mode.

The shortcuts mentioned above work only in the *command* mode cells. Another mode that a Jupyter notebook cell supports is the *edit* mode. This mode specifically allows you to edit the content of a cell and work with it. You can enter into the edit mode of a cell by pressing the return key or by a mouse click inside a cell. The border around cell changes to Green when the cell is in the *edit* mode, as shown in the below snapshot.



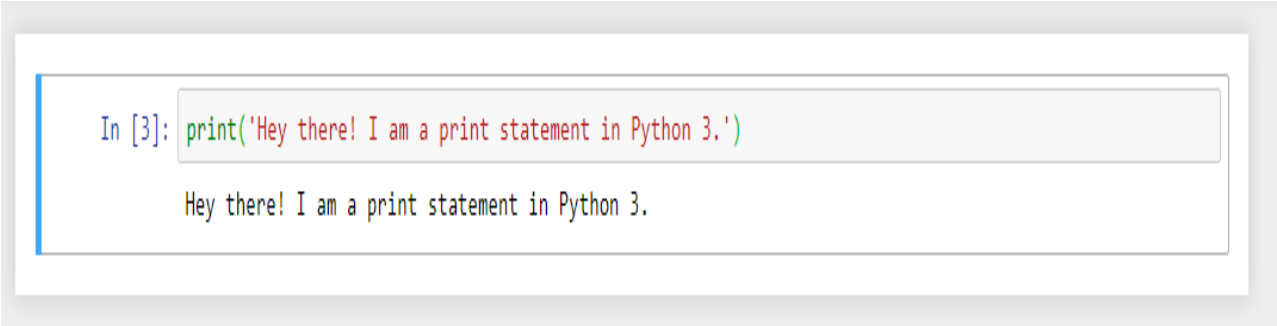
Once the cell is in *edit* mode, you can start writing code or text. The below-mentioned are some of the operations that you can perform while the cell is in the *edit* mode.

- *Auto-Completion* of code -- Use Tab key to use this facility. It works only for code type cells. In markdown cells, it will simply put tab spaces.
- *Indentation* of code -- Jupyter notebooks inherently perform indentation whenever required. However, if you want to change indentation manually, use Ctrl-] to indent the code in code type cells. In markdown cells, it will insert spaces according to the specifications of the tab key.
- *Dedentation* of code -- Anytime you want to dedent the code, use Ctrl-[to do so. In markdown cells, this shortcut works similar to Shift-Tab and dedents the content.
- *Commenting* a code -- Use Ctrl-/ to comment a code. In markdown cells, this shortcut does not have any effect.

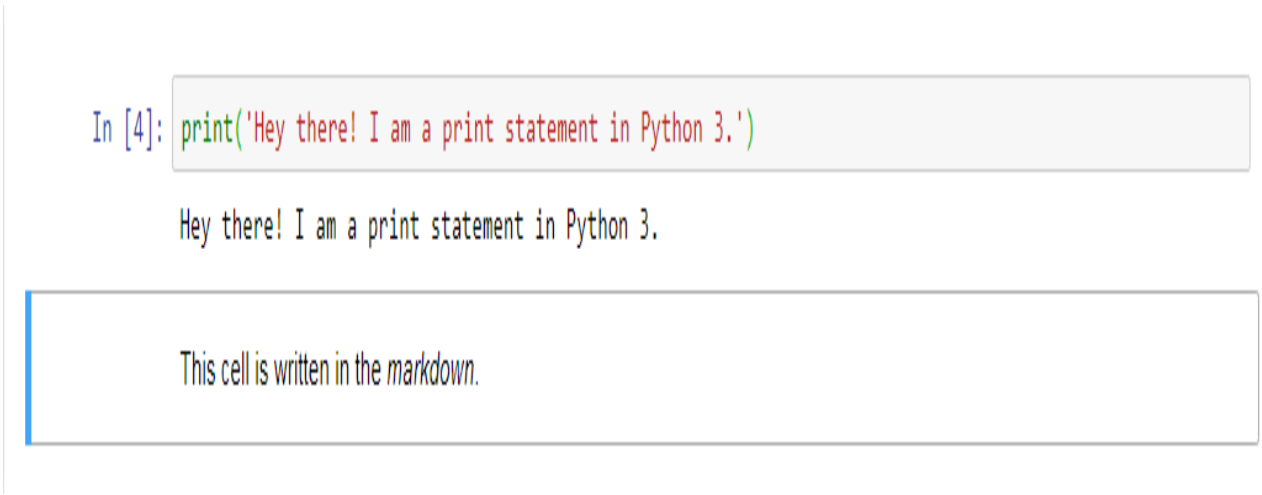
- *Execute of a cell* -- Once you write code or text in the cell, you need to execute it to process the content written by you. There are three primary ways to do so.
    - *Run the current cell and select the next cell* - Use Shift-Enter to perform this action.
    - *Run selected cells* - Use Ctrl-Enter to run selected cells or the current cell.
    - *Run the current cell and insert a new cell* - Press Alt-Enter to execute the current cell and insert the new cell below the current cell.
- *Split a cell* -- Use the shortcut Ctrl-Shift-Minus to split the current cell into two separate cells at the cursor.
- *Entering a Command Mode* -- Use Ctrl-M or press Esc key to exit from the edit mode and enter into the command mode.

By now, you have already encountered *code* and *markdown* cell types quite a few times. In case you are not aware of the two, now is the time where I will explain them in detail. I will restrict the discussion for these two types only; I won't be covering the *Raw NBConvert* type in this article.

Jupyter notebook cells can be multiple types. Often used types are *code* and *markdown*. The code type cells allow you to write live programming code. That is, you can perform any sort of programming in them. Once you run or execute a code cell, Jupyter notebook will present the output just below the cell. This is shown in the below snapshot.



In contrast, whatever written in the markdown cell, will get printed in the cell itself, as shown below:



There are two cells in the above snapshot. The first cell numbered four, is the code cell, which allows tying in Python code as we are working with Python kernel notebook. The next cell is the markdown cell where the normal text is written.

Chapter-6

CONCLUSION

6.1 Conclusion

The immense development of technology it is significant to store the maximum size and the complex datasets. The real-time time series of the datasets may involve the size up to the trillion observations and more. The goal is to extract new knowledge which is hidden in these large datasets. This survey provides the broad and deep knowledge regarding the challenges and techniques involved in time series data mining research field.

In this paper, the study is done on the various methodologies in data mining used for disease prediction. Classification Accuracy has been increased by these methodologies. For clinical dataset, classification techniques are used for disease prediction with high accuracy. The time consumption of prediction of diseases was reduced by using different classification techniques.

6.2 Reference

[1] Z. Yin and J. Zhang, "Identification of temporal variations in mental workload using locally-linear-embedding-based EEG feature reduction and supportvector-machine-based clustering and classification techniques," Comput. Methods Programs Biomed., vol. 115, no. 3, pp. 119–134, 2014.

[2] R. Sitaram, H. Zhang, C. Guan, M. Thulasidas, Y. Hoshi, A. Ishikawa, K. Shimizu, and N. Birbaumer, "Temporal classification of multichannel nearinfrared spectroscopy signals of motor imagery for developing abrain– computer interface," NeuroImage, vol. 34, no. 4, pp. 1416–1427, 2007.

[3] M. F. Ghalwash and Z. Obradovic, "Early classification of multivariate temporal observations by extraction of interpretable shapelets," BMC Bioinformat., vol. 13, art. no. 195, 2012.

[4] I. Batal, H. Valizadegan, G. F. Cooper, and M. Hauskrecht, "A pattern mining approach for classifying multivariate temporal data," in Proc. IEEE Int. Conf. Bioinformat. Biomed., vol. 2011, Nov. 12, 2011, pp. 358–365.

[5] R. Schmidt and L. Gierl, “A prognostic model for temporal courses that combines temporal abstraction and case-based reasoning,” Int. J. Med. Informat., vol. 74, nos. 2–4, pp. 307–315, 2005.

[6] R. Bellazzi, C. Larizza, P. Magni, and R. Bellazzi, “Temporal data mining for the quality assessment of hemodialysis services,” Artif. Intell. Med., vol. 34, no. 1, pp. 25–39, 2005.

[7] C. H. Lee, J. C. Chen, and V. S. Tseng, “A novel data mining mechanism considering bio-signal and environmental data with applications on asthma monitoring,” Comput.Methods Programs Biomed., vol. 101, no. 1, pp. 44–61, Jan. 2011.

[8] V. S. Tseng and C.-H. Lee, “Effective temporal data classification by integrating sequential pattern mining and probabilistic induction,” Expert Syst. Appl., vol. 36, no. 5, pp. 9524–9532, 2009.

[9] T. Exarchos,M. Tsipouras, C. Papaloukas, and D. Fotiadis, “An optimized sequential pattern matching methodology for sequence classification,” Knowl. Inf. Syst., vol. 19, no. 2, pp. 249–264, 2009.

[10] D. Bargiel and S. Herrmann, “Multi-temporal land-cover classification of agricultural areas in two European regions with high resolution spotlight terraSAR-X data,” Remote Sens., vol. 3, no. 5, pp. 859–877, 2011.

[11] J. O. Sexton, D. L. Urban, M. J. Donohue, and C. Song, “Long-term land cover dynamics by multi-temporal classification across the Landsat-5 record,” Remote Sens. Environ., vol. 128, pp. 246–258, 2013.