

**Admission Prediction**

Guided by :Pranav Jaipurkar

M.E.

Knowledge Solutions India

Presendted by: **Naaragam Raghavendra Sai Krishna**

**Kunduru Tharun Kumar Reddy**

**Nakkala Sai Teja**

**Vuyyuru Chandra Sekhar**

Table of Contents

[Abstract: 4](#_Toc70354412)

[Introduction: 4](#_Toc70354413)

[Software libraries used: 5](#_Toc70354414)

[NumPy : 5](#_Toc70354415)

[Pandas: 6](#_Toc70354416)

[Matplotlib : 7](#_Toc70354417)

[Seaborn: 8](#_Toc70354418)

[Sklearn: 9](#_Toc70354419)

[Algorithms: 10](#_Toc70354420)

[Exploratory Data Analysis (EDA): 10](#_Toc70354421)

[Data Pre-processing: 11](#_Toc70354422)

[Principal Component Analysis (PCA): 12](#_Toc70354423)

[Multiple Linear Regression (MLR): 15](#_Toc70354424)

[Random Forest: 17](#_Toc70354425)

[Multiple Linear Regression using PCA: 22](#_Toc70354426)

[Random Forest using PCA: 23](#_Toc70354427)

[Conclusion: 25](#_Toc70354428)

[References: 26](#_Toc70354429)

Table of figures:

[Figure 1 8](#_Toc70354367)

[Figure 2 8](#_Toc70354368)

[Figure 3 11](#_Toc70354369)

[Figure 4 12](#_Toc70354370)

[Figure 5 15](#_Toc70354371)

[Figure 6 16](#_Toc70354372)

[Figure 7-a 16](#_Toc70354373)

[Figure 8 17](#_Toc70354374)

[Figure 9 17](#_Toc70354375)

[Figure 10 20](#_Toc70354376)

[Figure 11 20](#_Toc70354377)

[Figure 12 21](#_Toc70354378)

[Figure 13 21](#_Toc70354379)

[Figure 14 21](#_Toc70354380)

[Figure 15 22](#_Toc70354381)

[Figure 16 22](#_Toc70354382)

[Figure 17 22](#_Toc70354383)

[Figure 18 23](#_Toc70354384)

[Figure 19 23](#_Toc70354385)

[Figure 20 23](#_Toc70354386)

[Figure 21 24](#_Toc70354387)

[Figure 22 24](#_Toc70354388)

[Figure 23 24](#_Toc70354389)

[Figure 24 25](#_Toc70354390)

[Figure 25 25](#_Toc70354391)

# Abstract:

# Introduction:

How can we predict whether a student will get an admit or not?

What are the parameters for selection?

Can it be mathematically expressed?

Given the problem statement, this is the pipeline we've followed so that your approach is structured.

1. Define the problem -

Our objective is to predict the chance of a student getting an admission. So it means that this is a Regression problem.

Generate your own hypothesis -

2. Next, list down all the things that you think can affect our objective viz. List down all the possible features with respect to our target feature.

3. In our case, we got ask this question — what are the factors that can affect a student’s admission.

4. We ended up with the following features

5. - GRE Score

6. - TOEFL Score

7. - Statement of Purpose (SOP)

8. - Letter of Recommendation (LOR)

9. - Academic Performance (GPA)

10. - Extra Curricular Activities (Sports, Math Olympiad etc..)

11. - Outstanding Achievements

12. - Projects and Research

13. Get the Dataset -

14. Next step is to get the data. We’re using UCLA hypothetical data for graduate admissions. You can download the dataset here.

# Software libraries used:

NumPy : NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

There are several important differences between NumPy arrays and the standard Python sequences:

* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
* NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
* A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today’s scientific/mathematical Python-based software, just knowing how to use Python’s built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

## Pandas:

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState’s ActivePython.

Pandas makes it simple to do many of the time consuming, repetitive tasks associated with working with data, including:

* Data cleansing
* Data fill
* Data normalization
* Merges and joins
* Data visualization
* Statistical analysis
* Data inspection
* Loading and saving data e.t.c

Key Features of Pandas

* Fast and efficient Data Frame object with default and customized indexing.
* Tools for loading data into in-memory data objects from different file formats.
* Data alignment and integrated handling of missing data.
* Reshaping and pivoting of date sets.
* Label-based slicing, indexing and sub setting of large data sets.
* Columns from a data structure can be deleted or inserted.
* Group by data for aggregation and transformations.
* High performance merging and joining of data.
* Time Series functionality.

## Matplotlib :

A picture is worth a thousand words, and with Python’s matplotlib library, it fortunately takes far less than a thousand words of code to create a production-quality graphic.

A plot is really a hierarchy of nested Python objects. A “hierarchy” here means that there is a tree-like structure of matplotlib objects underlying each plot.

A Figure object is the outermost container for a matplotlib graphic, which can contain multiple Axes objects. One source of confusion is the name: an Axes actually translates into what we think of as an individual plot or graph (rather than the plural of “axis,” as we might expect).

You can think of the Figure object as a box-like container holding one or more Axes (actual plots). Below the Axes in the hierarchy are smaller objects such as tick marks, individual lines, legends, and text boxes. Almost every “element” of a chart is its own manipulable Python object, all the way down to the ticks and labels:

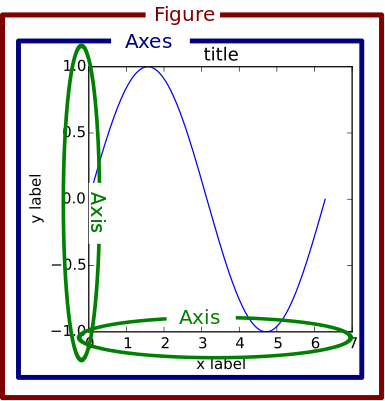


Figure 1

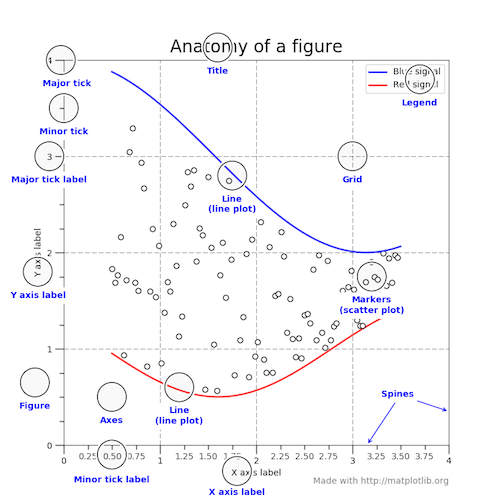


Figure 2

## Seaborn:

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

Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

some of the functionality that seaborn offers:

* A dataset-oriented API for examining relationships between multiple variables
* Specialized support for using categorical variables to show observations or aggregate statistics
* Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
* Automatic estimation and plotting of linear regression models for different kinds dependent variables
* Convenient views onto the overall structure of complex datasets
* High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
* Concise control over matplotlib figure styling with several built-in themes
* Tools for choosing color palettes that faithfully reveal patterns in your data

## Sklearn:

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

* **NumPy**: Base n-dimensional array package
* **SciPy**: Fundamental library for scientific computing
* **Matplotlib**: Comprehensive 2D/3D plotting
* **IPython**: Enhanced interactive console
* **Sympy**: Symbolic mathematics
* **Pandas**: Data structures and analysis

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.

Some popular groups of models provided by scikit-learn include:

* **Clustering**: for grouping unlabeled data such as KMeans.
* **Cross Validation**: for estimating the performance of supervised models on unseen data.
* **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
* **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
* **Ensemble methods**: for combining the predictions of multiple supervised models.
* **Feature extraction**: for defining attributes in image and text data.
* **Feature selection**: for identifying meaningful attributes from which to create supervised models.
* **Parameter Tuning**: for getting the most out of supervised models.
* **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
* **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.
* **Preprocessing**, including Min-Max Normalization

# Algorithms:

Steps we had followed before building out target models (After importing the modules):

1. Exploratory Data Analysis (EDA)

2. Data Pre-processing

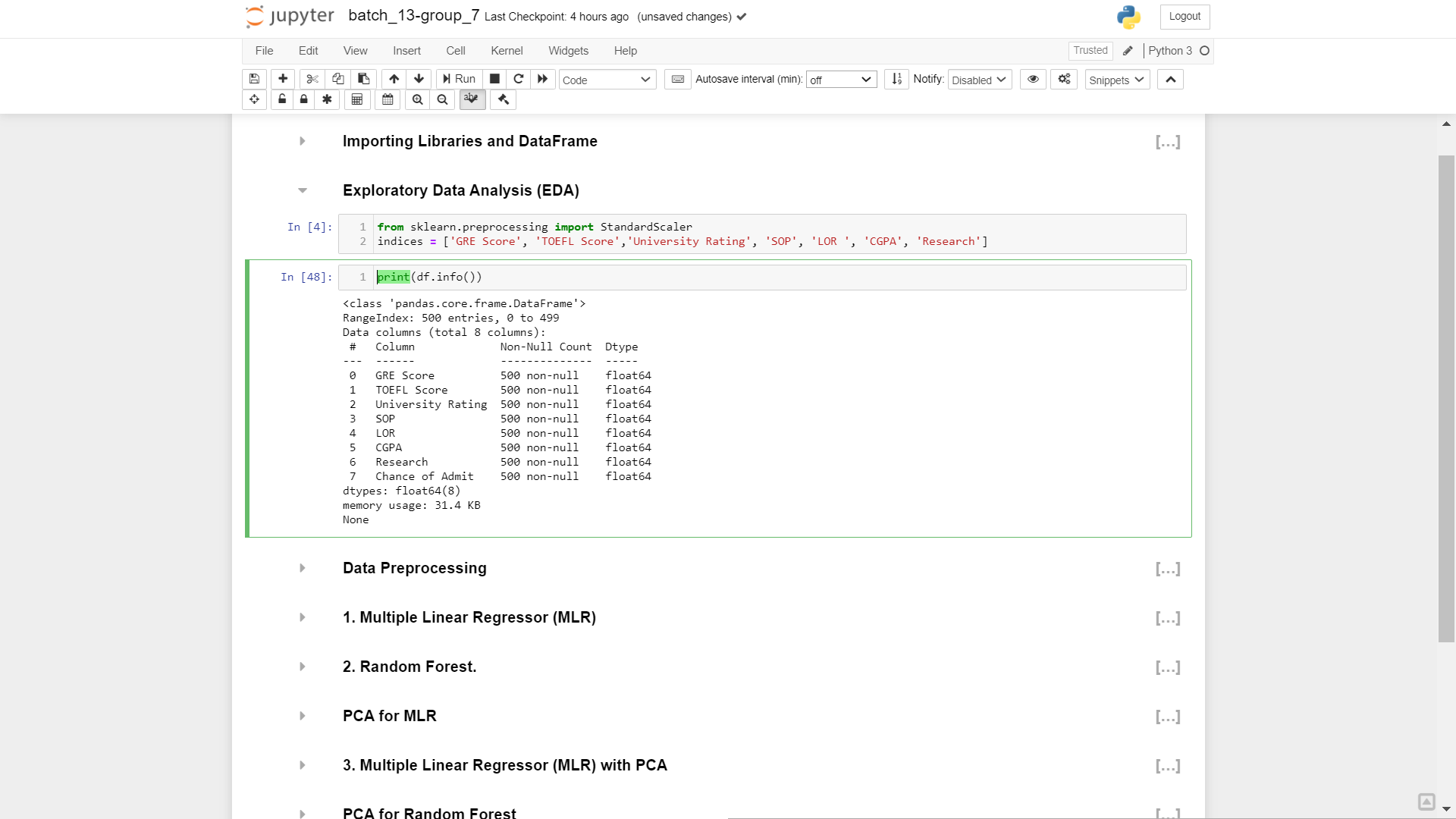
3. Principal Component Analysis (PCA) {if necessary/mentioned}

## Exploratory Data Analysis (EDA):

EDA is the necessary step to most of the machine learning models. From this we can see the analysis of whole data set like type of object, no of null values in the dataset and also we get to a conclusion that whether we go for standard scaling or encoding of categorical data.

Few statements the helps us in analysing the data that had imported using the pandas data frame (data):

* 1. data.info()
  2. data.describe()
  3. data.head()
  4. data.tail()
  5. data.dtypes
  6. data.shape
  7. data.size



Figure

## Data Pre-processing:

This is the important step if we ignore this step we can’t train many of our models and also the steps that should be done can be known with the previous step (EDA).

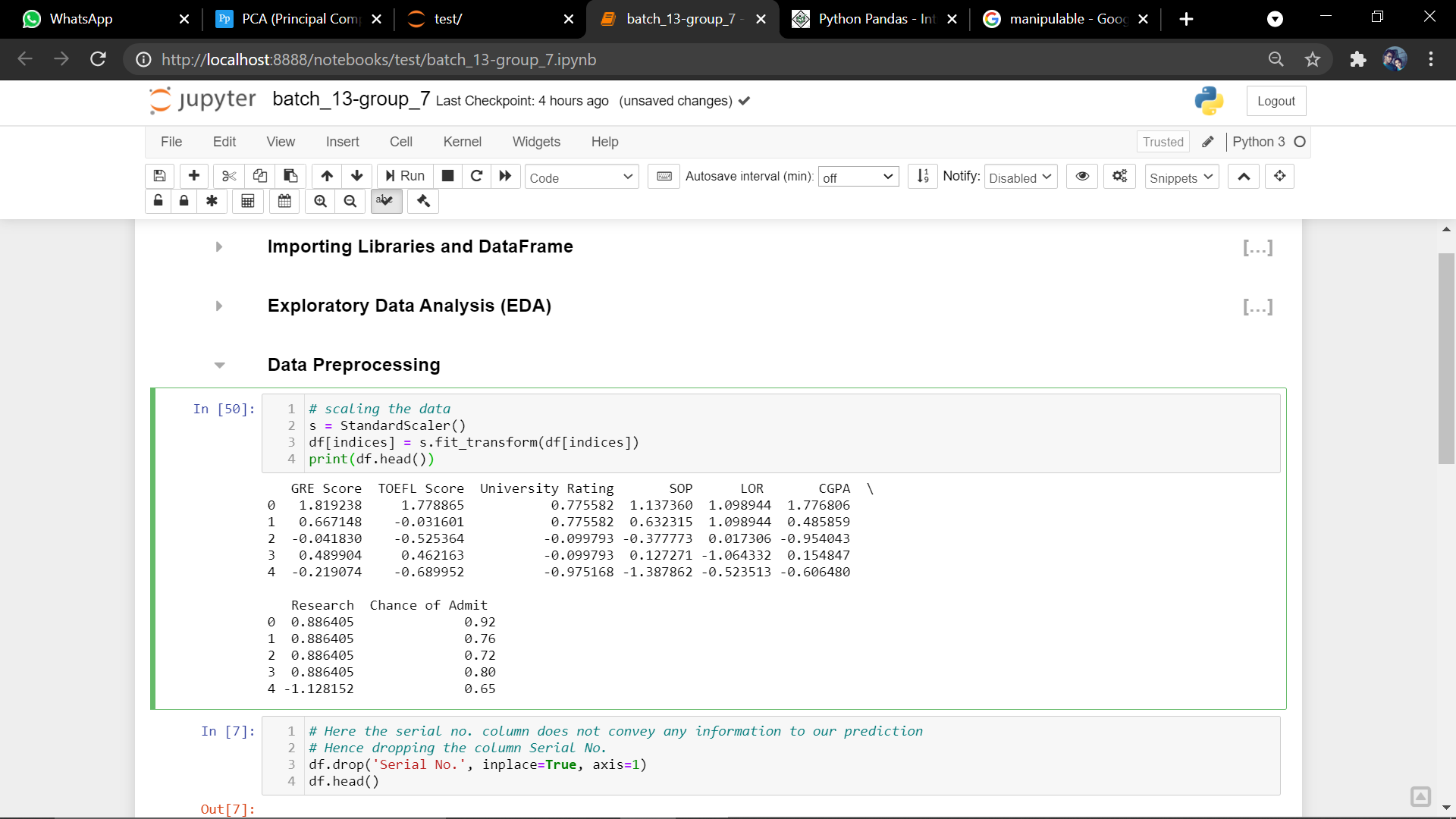
If our data contains any null values we should fill the null value with mean or median or any other value. If columns contain different Dtypes we should convert them to float or same value. If a column is categorical then we should use encoding of data so that a categorical value can be converted to numerical value so that we can train our model using this data.

But these steps are not prescribed to every model. It differs from model to model.

## Principal Component Analysis (PCA):

 Principal component analysis (PCA) is used to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order.

So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.Importantly, the dataset on which PCA technique is to be used must be scaled. The results are also sensitive to the relative scaling.



Figure

Dimensionality: The number of columns present in your dataset.

Correlation: It shows how strongly two variables are related to each other. The value of the same ranges for -1 to +1. Positive indicates that when one variable increases, the other increases as well, while negative indicates the other decreases on increasing the former. And the modulus value of indicates the strength of relation.

Orthogonal: Uncorrelated to each other, i.e., correlation between any pair of variables is 0.

Eigenvectors: Eigenvectors and Eigen values are in itself a big domain, consider a non-zero vector **v**. It is an eigenvector of a square matrix **A**, if **Av** is a scalar multiple of **v**. Or simply:

**Av = ƛv**

Here, **v** is the eigenvector and **ƛ** is the Eigen value associated with it.

Covariance Matrix: This matrix consists of the covariance between the pairs of variables. The (i , j)th element is the covariance between i-th and j-th variable.

**Step 1: Normalize the data:**

First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become 𝔁- and all Y become 𝒚-. This produces a dataset whose mean is zero.

**Step 2: Calculate the covariance matrix:**

Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

https://s3.amazonaws.com/files.dezyre.com/images/Tutorials/Covariance+Matrix.JPG

Please note that Var[X1] = Cov[X1,X1] and Var[X2] = Cov[X2,X2].

**Step 3: Calculate the eigenvalues and eigenvectors:**

Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. ***ƛ*** is an eigenvalue for a matrix ***A*** if it is a solution of the characteristic equation:

***det( ƛI - A ) = 0***

Where, ***I*** is the identity matrix of the same dimension as ***A*** which is a required condition for the matrix subtraction as well in this case and ‘***det’***is the determinant of the matrix. For each eigenvalue ***ƛ***, a corresponding eigen-vector ***v***, can be found by solving:

***( ƛI - A )v = 0***

**Step 4: Choosing components and forming a feature vector:**

We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with *n* variables, then we have the corresponding *n* eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first *p* eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater Eigenvalue or simply take both.

*Feature Vector = (eig1, eig2)*

**Step 5: Forming Principal Components:**

This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

NewData = FeatureVectorT x ScaledData*T*

Here,

NewDatais the Matrix consisting of the principal components,

FeatureVectoris the matrix we formed using the eigenvectors we chose to keep, and

ScaledDatais the scaled version of original dataset

(‘T’ in the superscript denotes transpose of a matrix)

If we go back to the theory of Eigen values and eigenvectors, we see that, essentially, eigenvectors provide us with information about the patterns in the data. In particular, in the running example of 2-D set, if we plot the eigenvectors on the scatter plot of data, we find that the principal eigenvector (corresponding to the largest Eigen value) actually fits well with the data. The other one, being perpendicular to it, does not carry much information and hence, we are at not much loss when deprecating it, hence reducing the dimension.

All the eigenvectors of a matrix are perpendicular to each other. So, in PCA, what we do is represent or transform the original dataset using these orthogonal (perpendicular) eigenvectors instead of representing on normal ***x*** and ***y*** axes. We have now classified our data points as a combination of contributions from both ***x*** and ***y.***The difference lies when we actually disregard one or many eigenvectors, hence, reducing the dimension of the dataset. Otherwise, in case, we take all the eigenvectors in account, we are just transforming the co-ordinates and hence, not serving the purpose.

Models that are used by us:

1. Multiple Linear Regression (MLR)
2. Random Forest
3. Multiple Linear Regression using PCA
4. Random Forest using PCA

## Multiple Linear Regression (MLR):

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable.



Figure

We used this model after EDA and Data Pre-processing steps.

Assumptions made while defining the multiple linear regression:

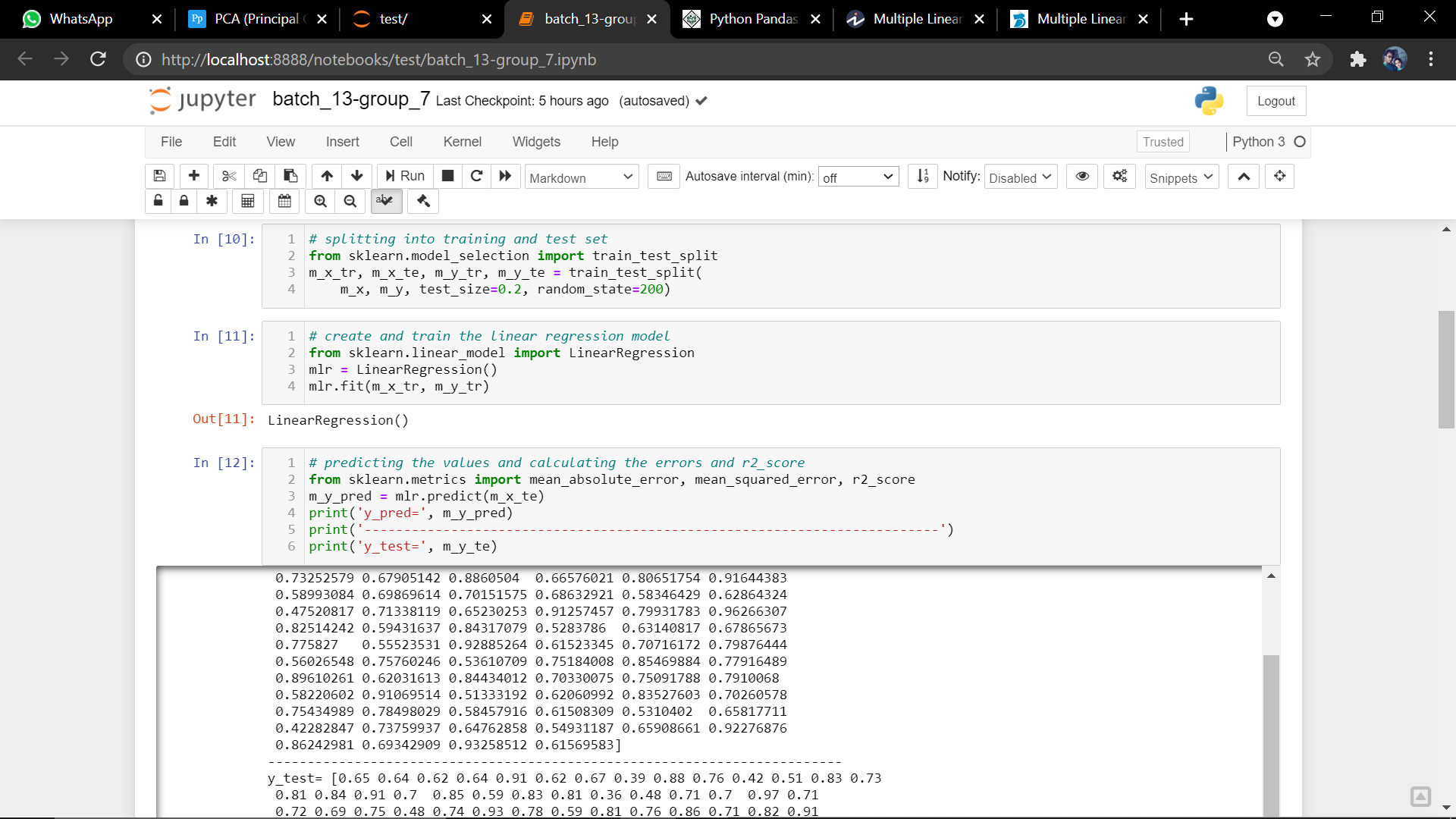
1. Relationship between your independent and dependent variables should always be linear.

2. Mean of residuals should be zero or close to 0 as much as possible. It is done to check whether our line is actually the line of “best fit”.

3. There should be homoscedasticity or equal variance in our regression model. This assumption means that the variance around the regression line is the same for all values of the predictor variable (X).

To be precise our may not be the perfect fit to MLR but we should consider that the real world problems are far different from our created models. So we have to check how much near our problem to the model by checking the assumptions.

We have to check the errors so that we can get to accuracy of our model.



Figure

In we had split our data set to test and train our model. But as we came to real world there is no need of splitting the data. The data will have a flow from the users.

Let’s check the scores and errors the we have got after fitting the model and their plotting features vs. chance of admit:

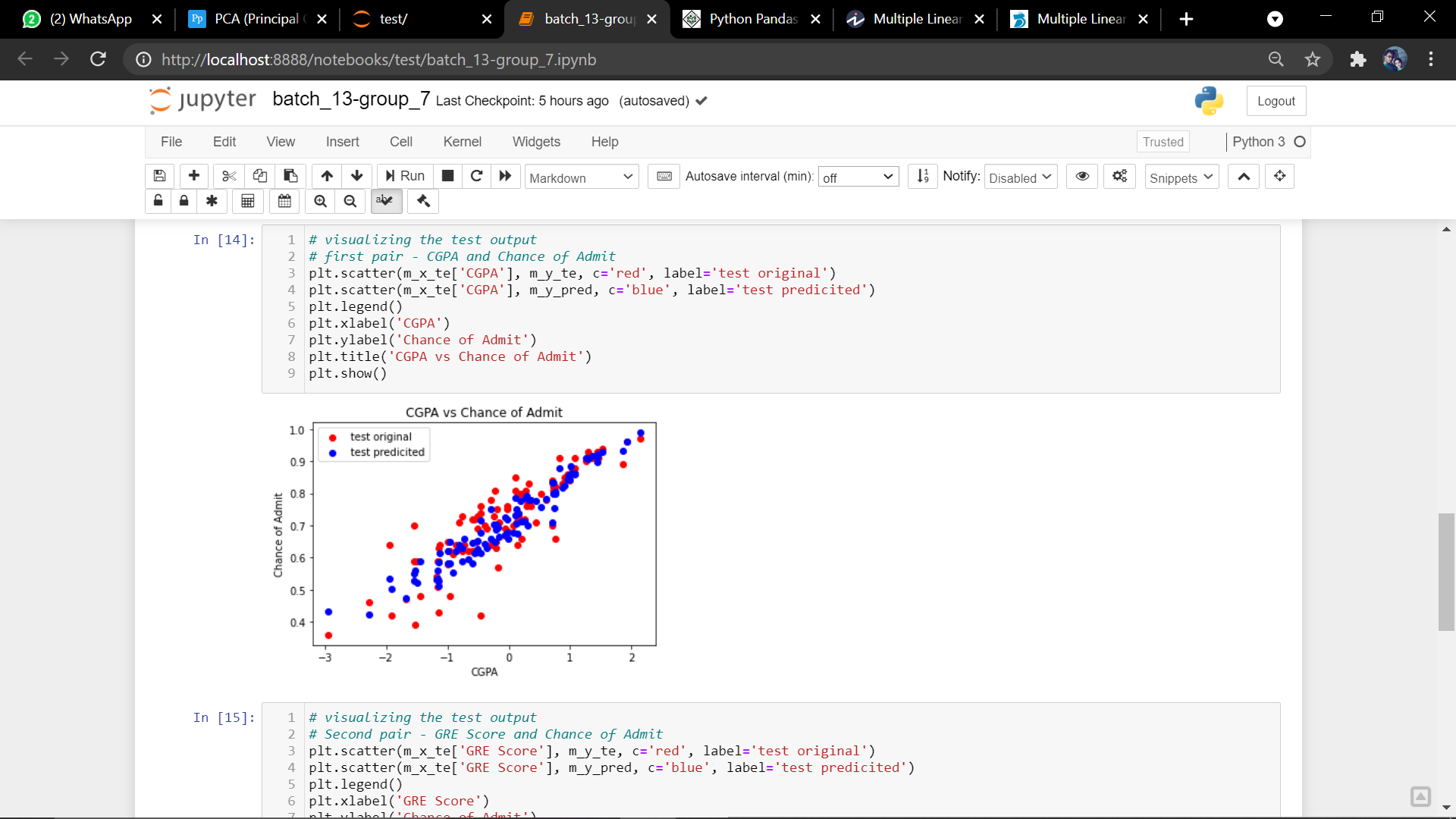
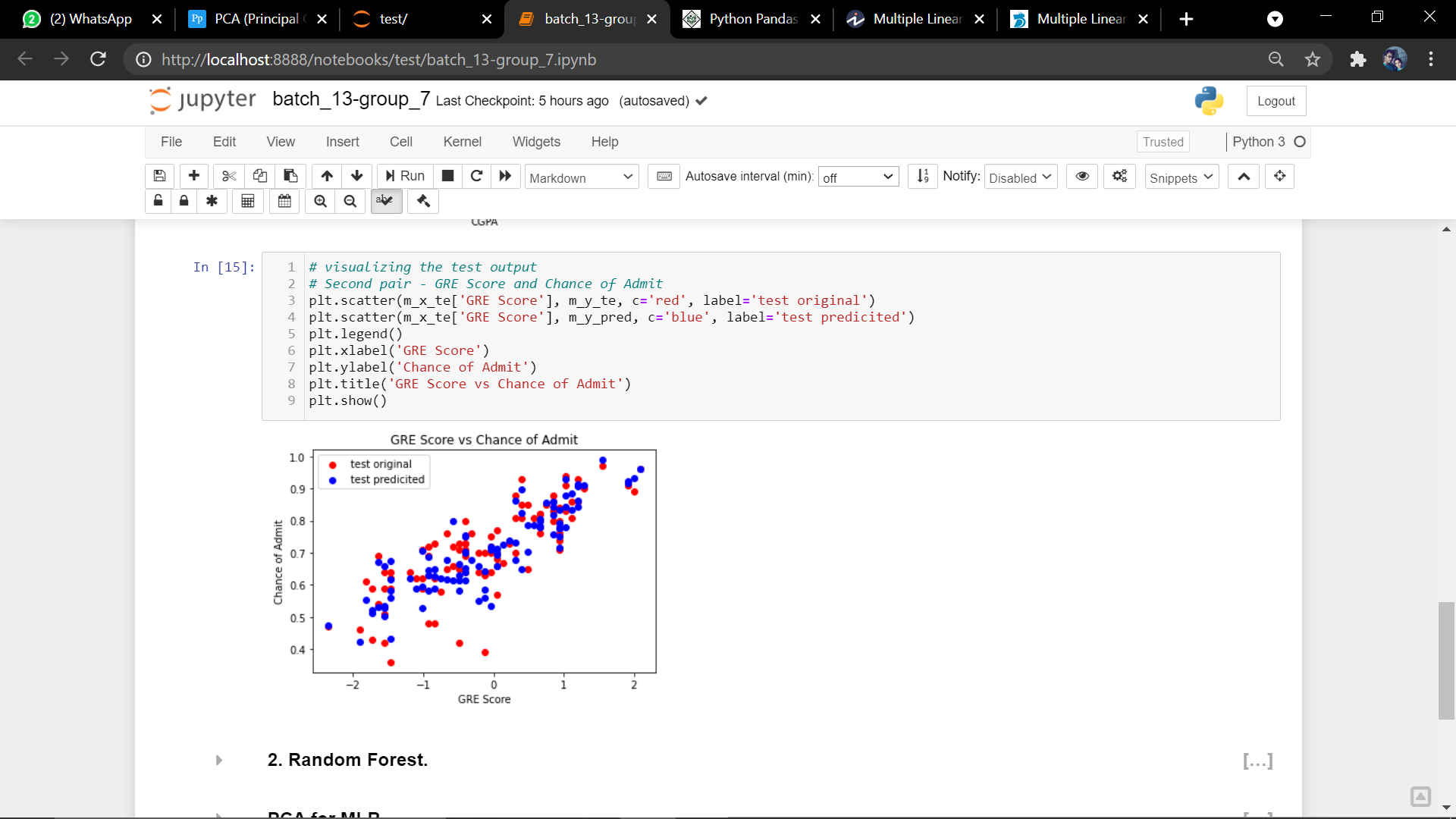
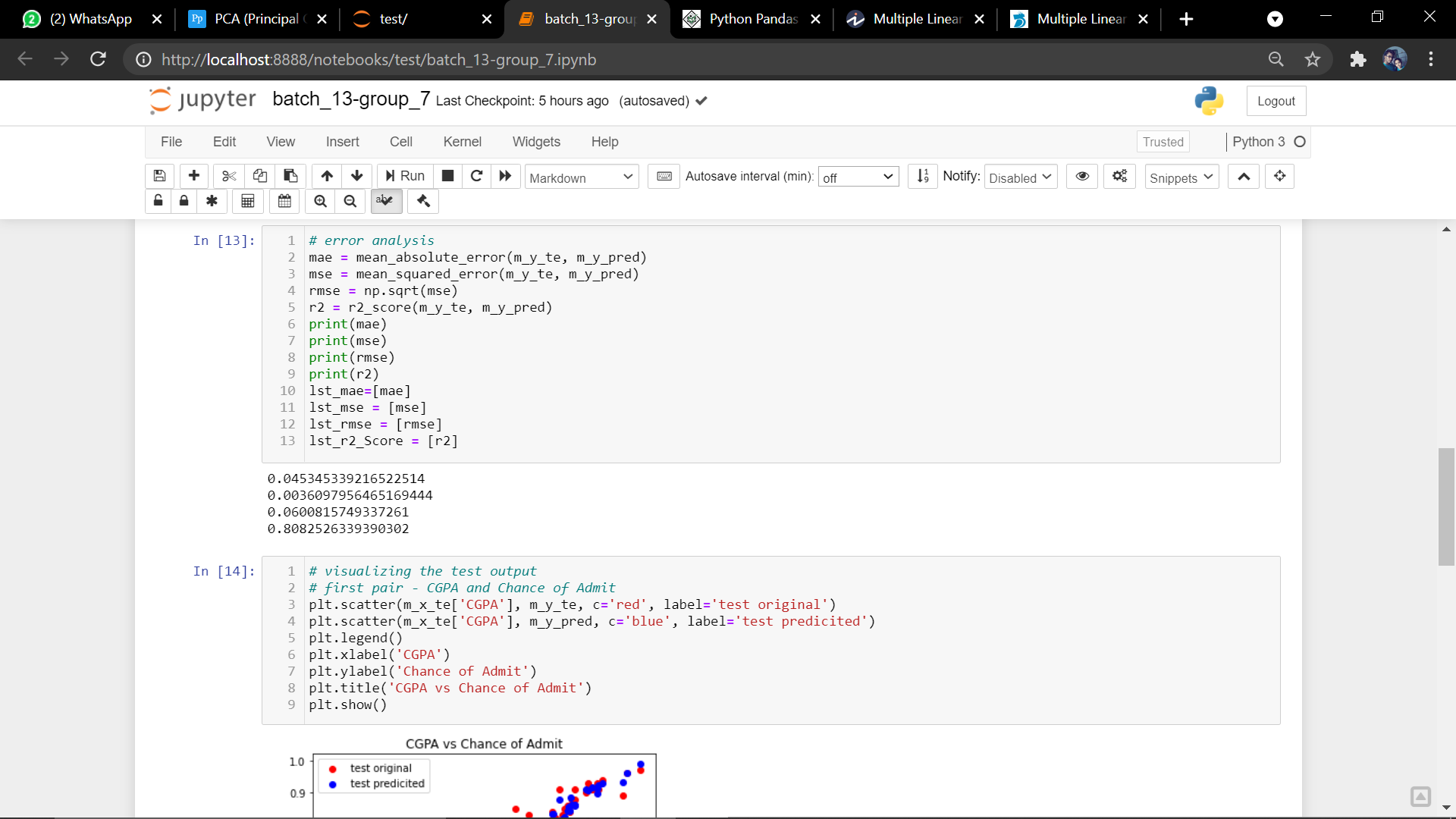


Figure -a



Figure



Figure

## Random Forest:

**Ensemble Learning:**

An Ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model. A model comprised of many models is called an **Ensemble model**.

Advantages due to ensemble learning:

**1.Boosting**

Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners. Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on.

In **boosting** as the name suggests, one is learning from other which in turn **boosts** the learning.

**2. Bootstrap Aggregation (Bagging)**

Bootstrap refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset. Bootstrap involves random sampling of small subset of data from the dataset.

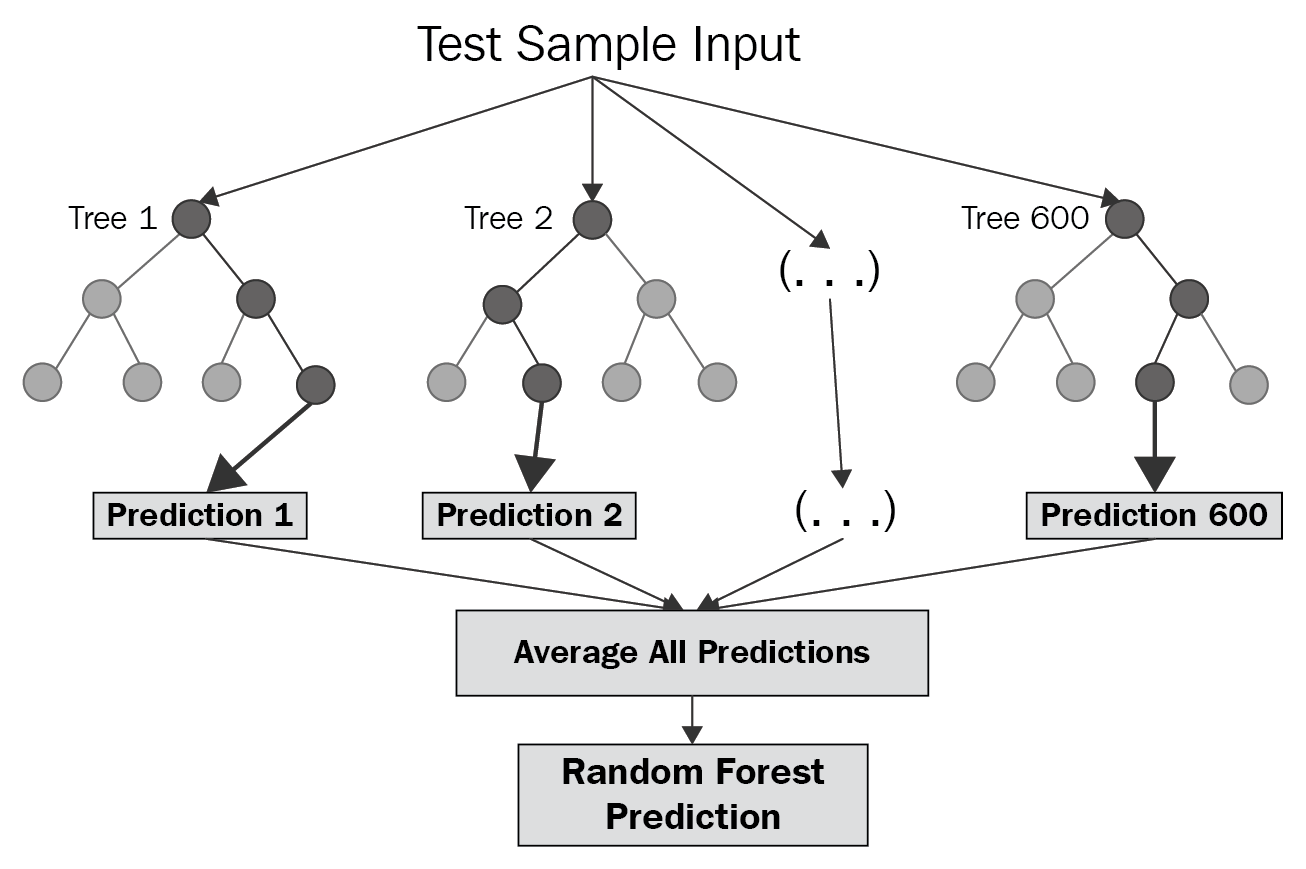
It is a general procedure that can be used to reduce the variance for those algorithm that have high variance, typically decision trees. Bagging makes each model run independently and then aggregates the outputs at the end without preference to any model.

Random Forest:

Random forest is a Supervised Learning algorithm which uses ensemble learning method for classification and regression.

Random forest is a bagging technique and not a boosting technique. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees.

It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.



**Some points on random forest**:

The number of features that can be split on at each node is limited to some percentage of the total (which is known as the hyperparameter). This ensures that the ensemble model does not rely too heavily on any individual feature, and makes fair use of all potentially predictive features.

Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents over fitting.

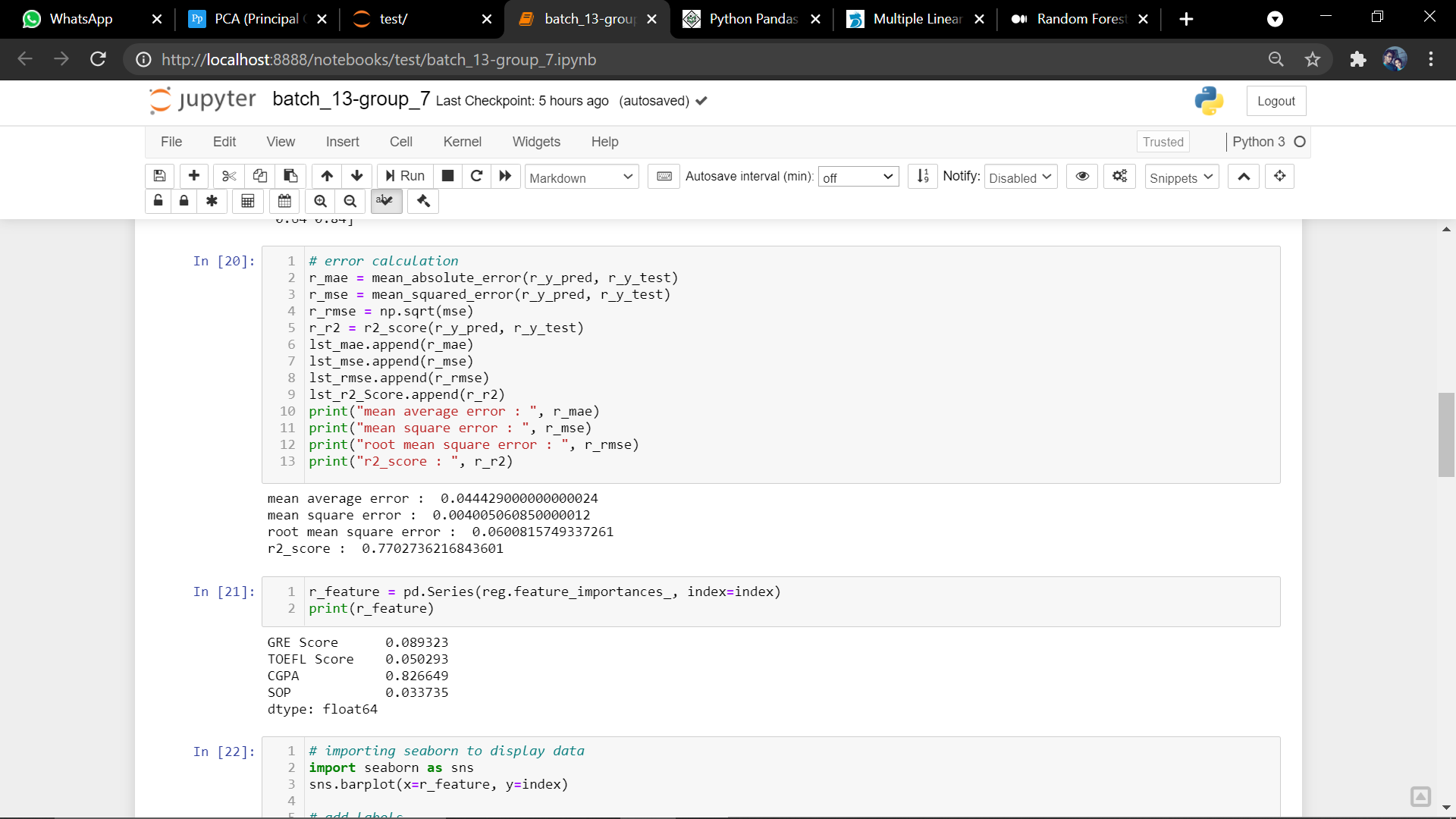
**Feature and Advantages of Random Forest** :

1. It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
2. It runs efficiently on large databases.
3. It can handle thousands of input variables without variable deletion.
4. It gives estimates of what variables that are important in the classification.
5. It generates an internal unbiased estimate of the generalization error as the forest building progresses.
6. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

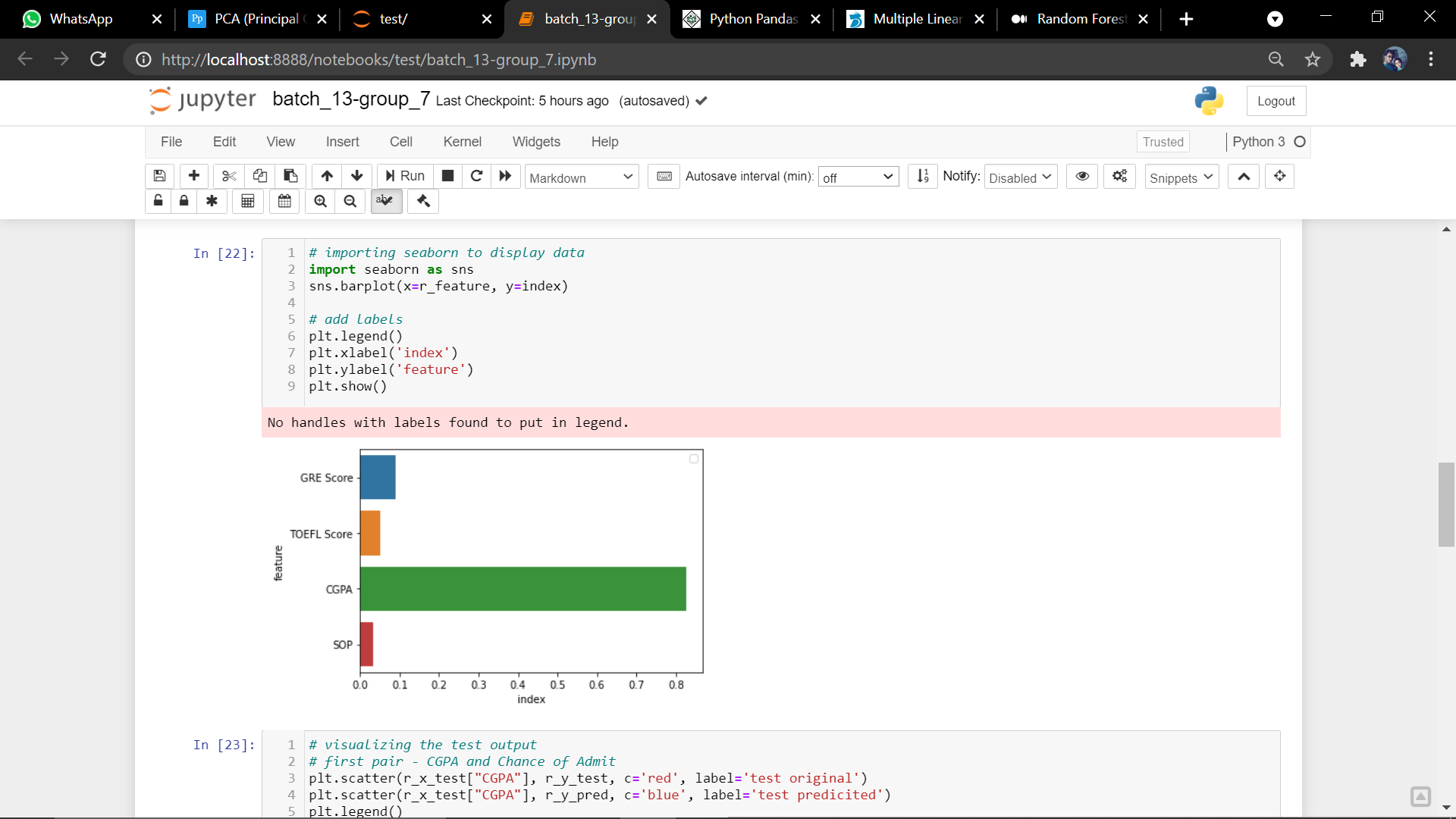
**Disadvantages of Random Forest** :

1. Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
2. For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

The error values in our model:

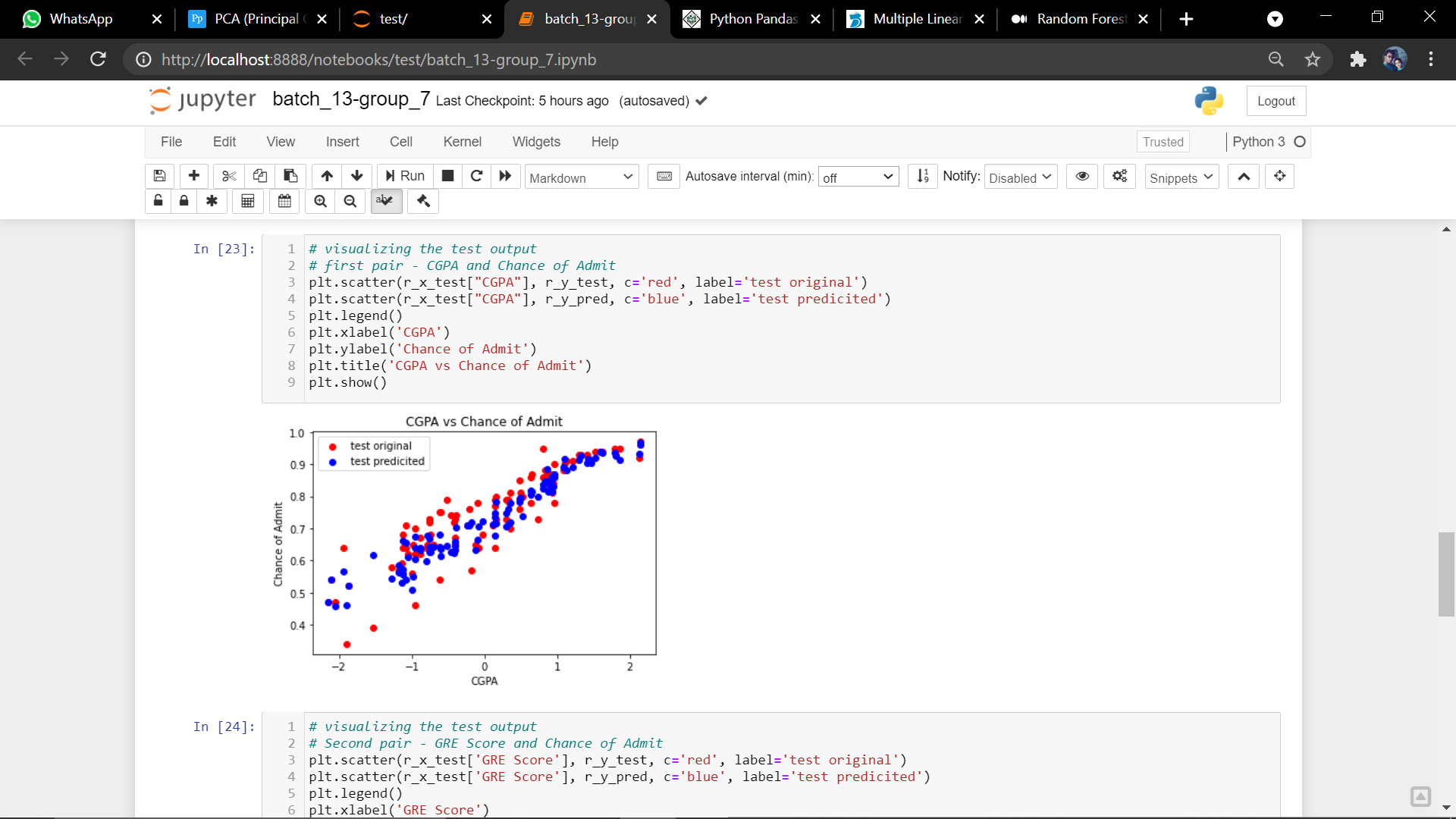


Figure

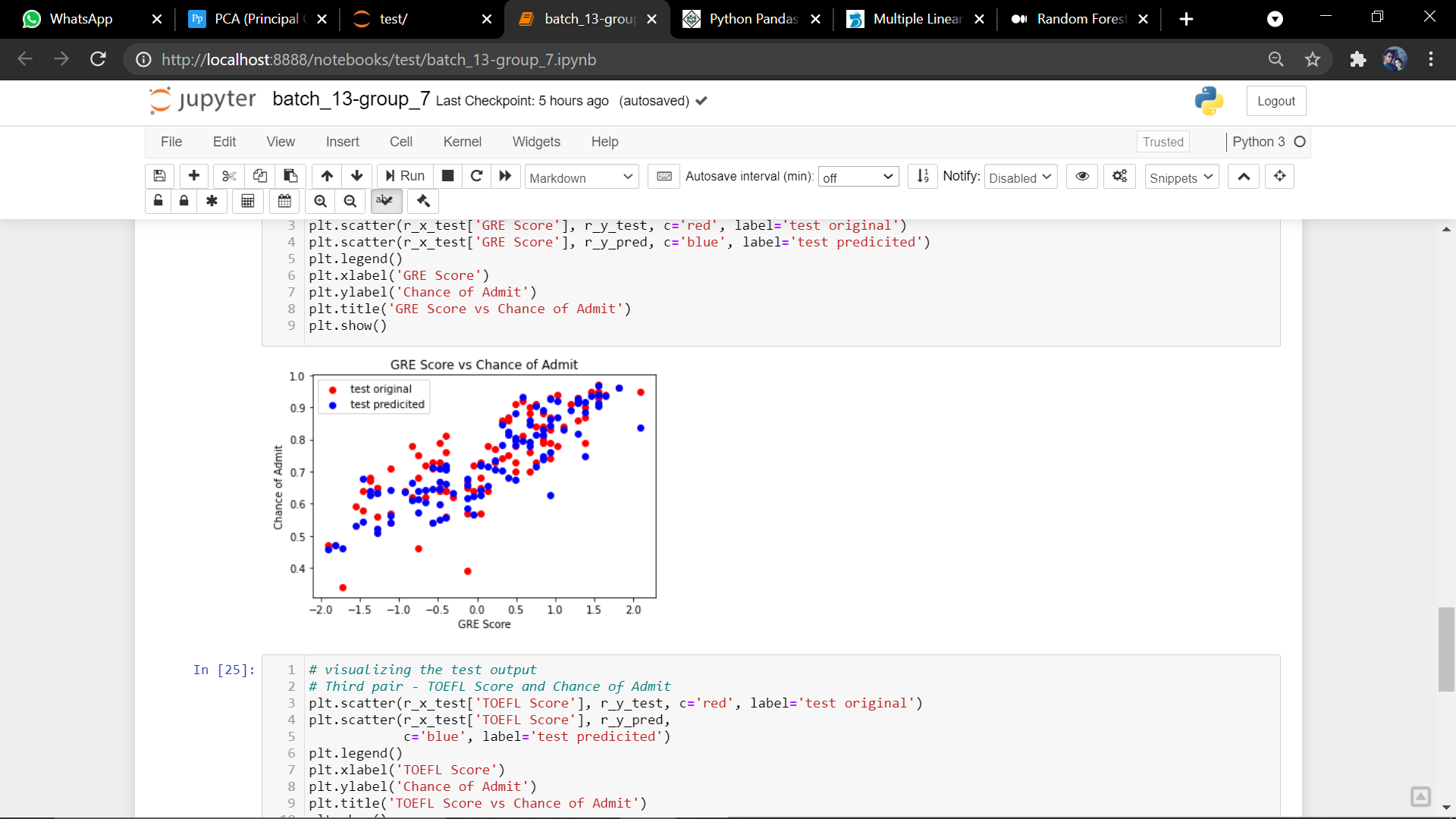


Figure

The above figure shows us the feature importance of our project.



Figure



Figure



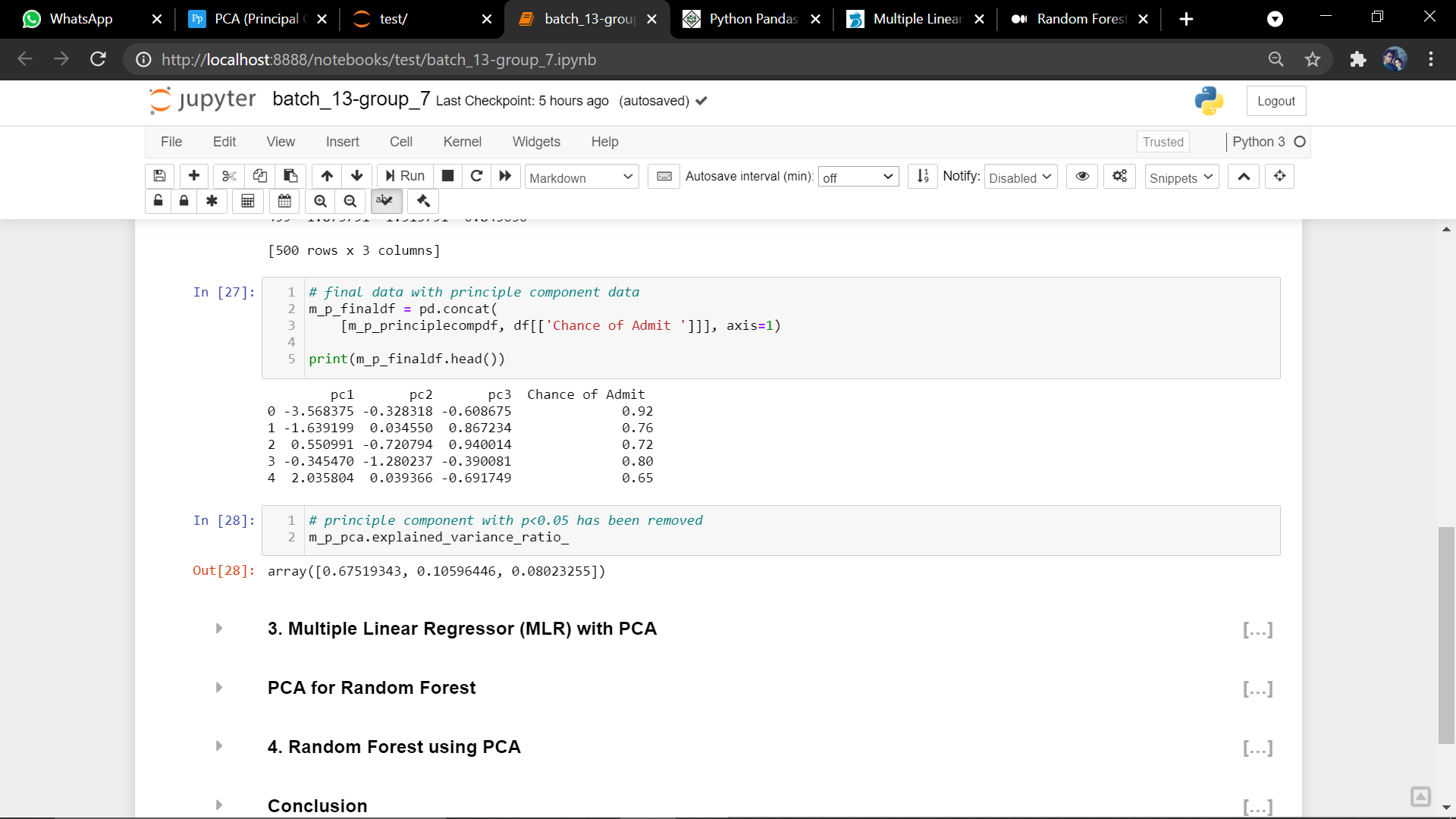
Figure

Figure 12-14 shows us the plot against different features used in random forest.

## Multiple Linear Regression using PCA:

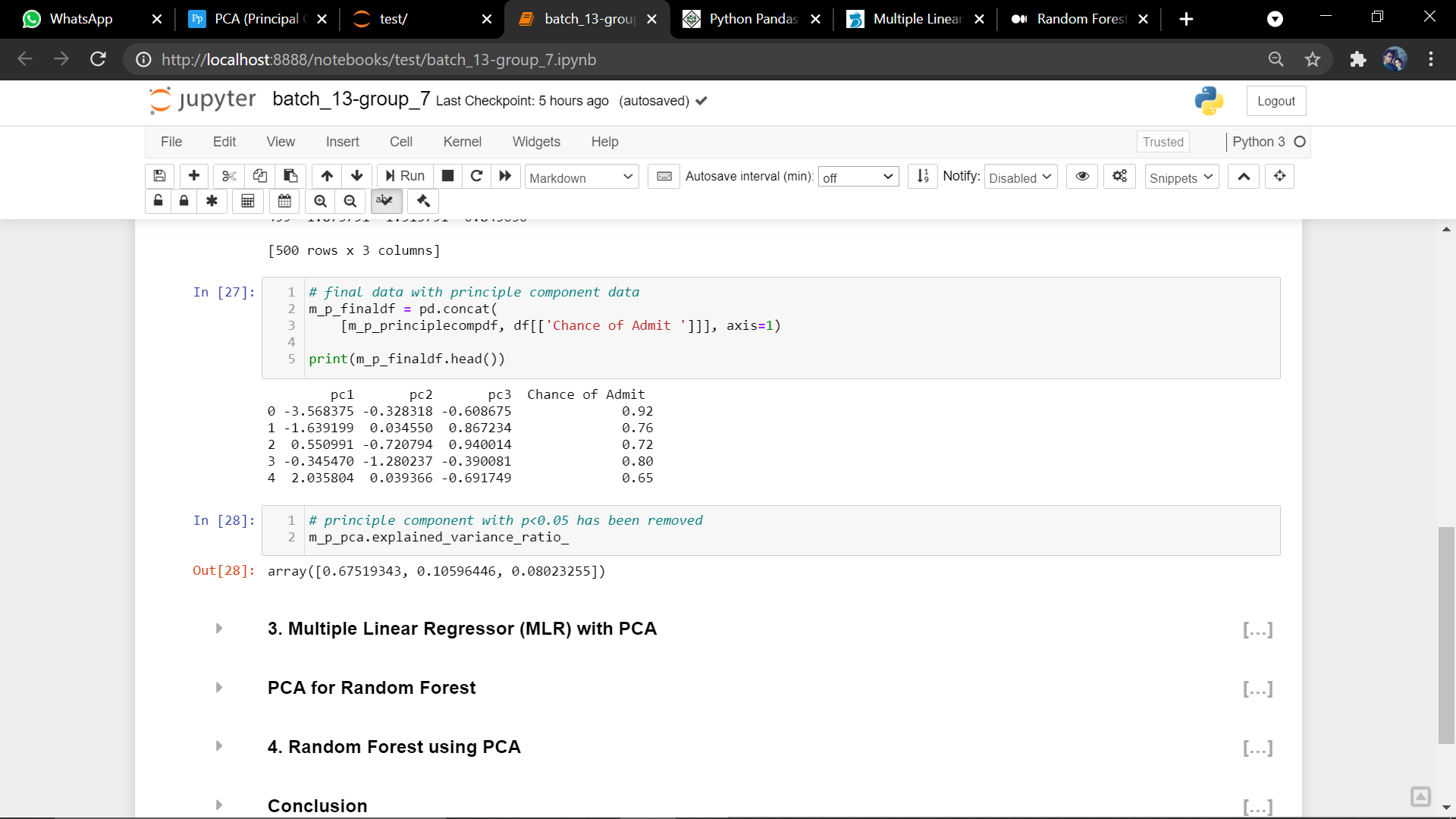
As MLR and PCA is explained above we directly jumping into errors, plotting and some variance details in PCA.

In this figure the pca components after fitting and transforming has been shown



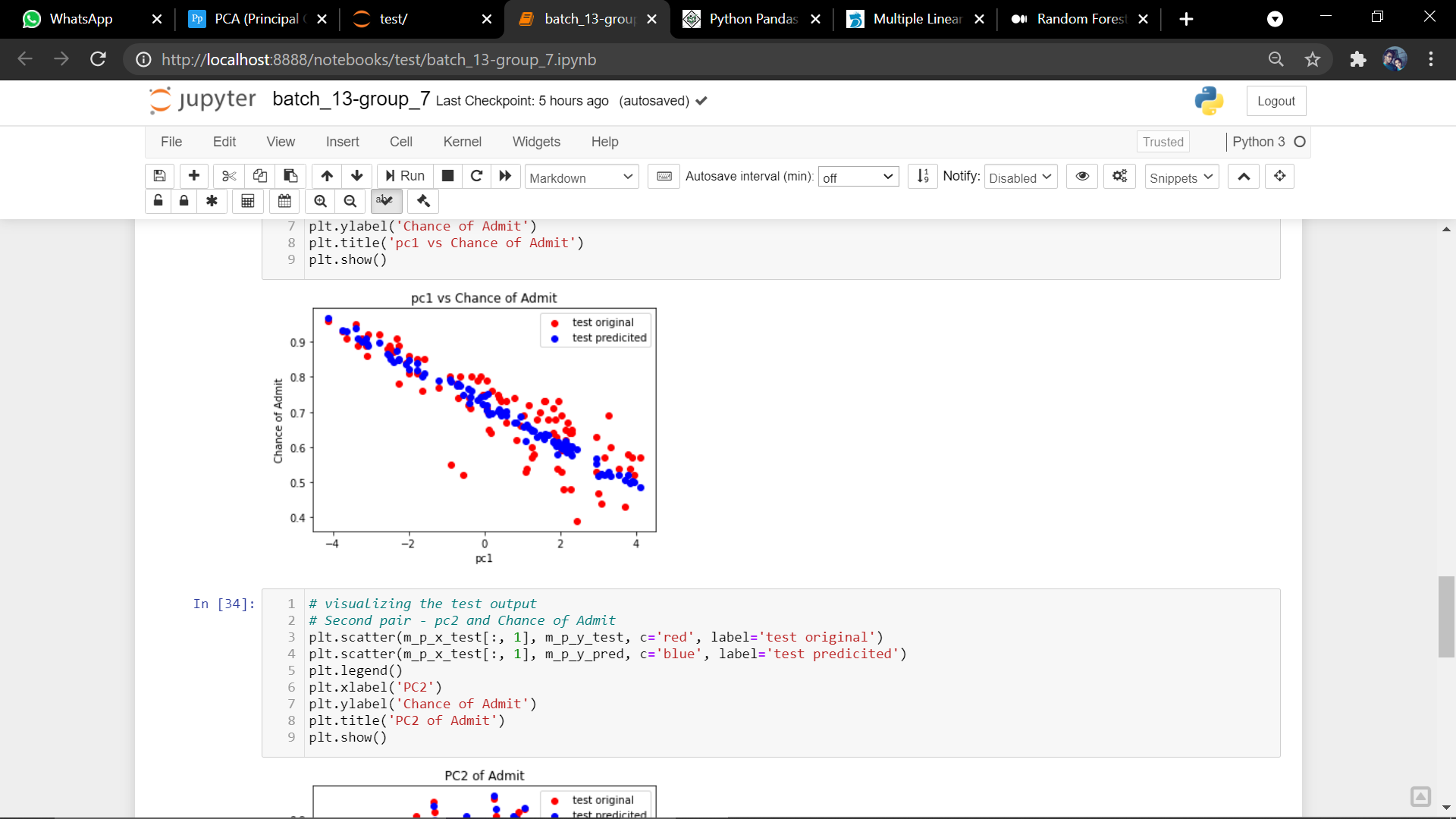
Figure

In the below figure we are showing the variance of each component:

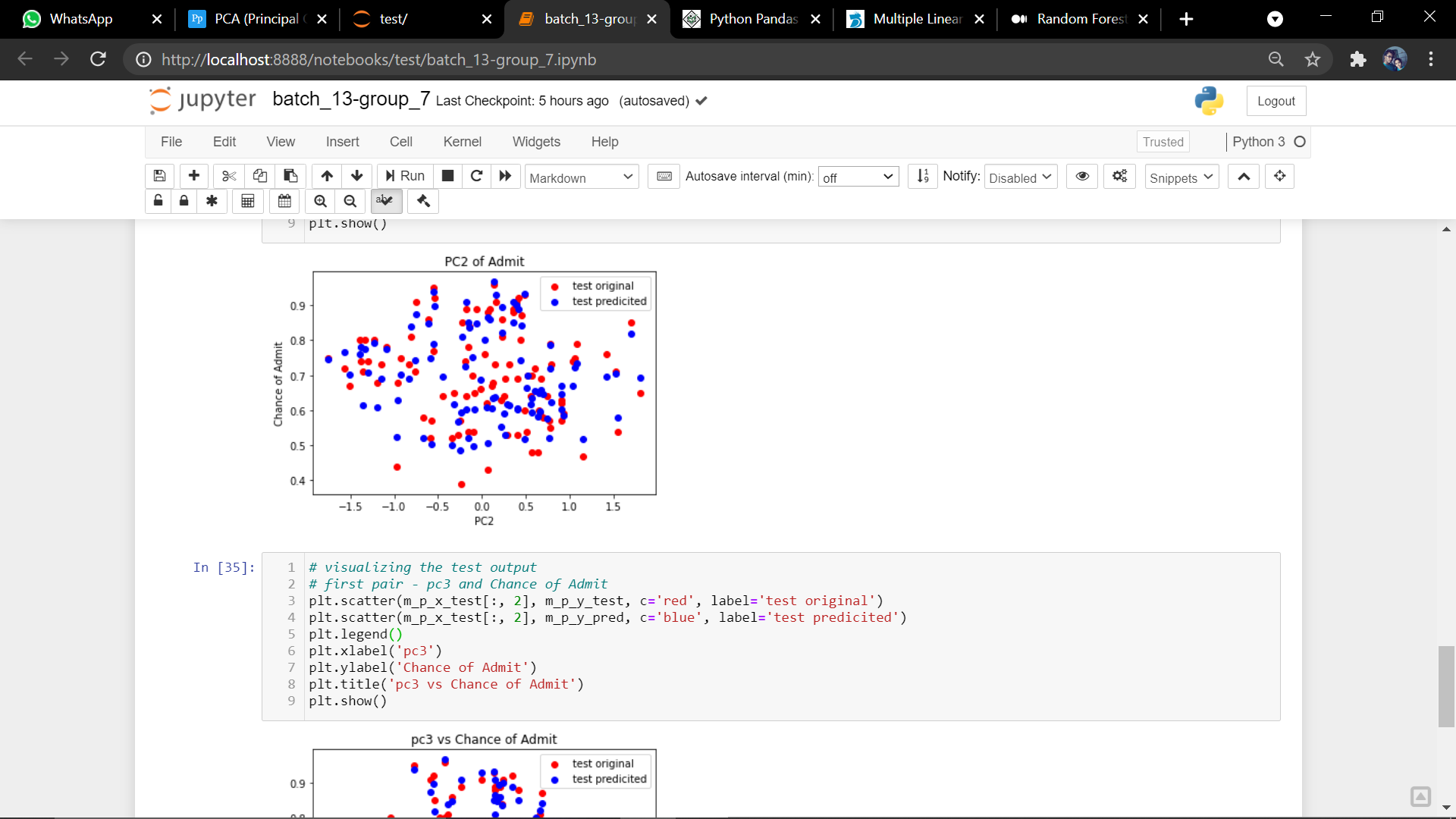


Figure

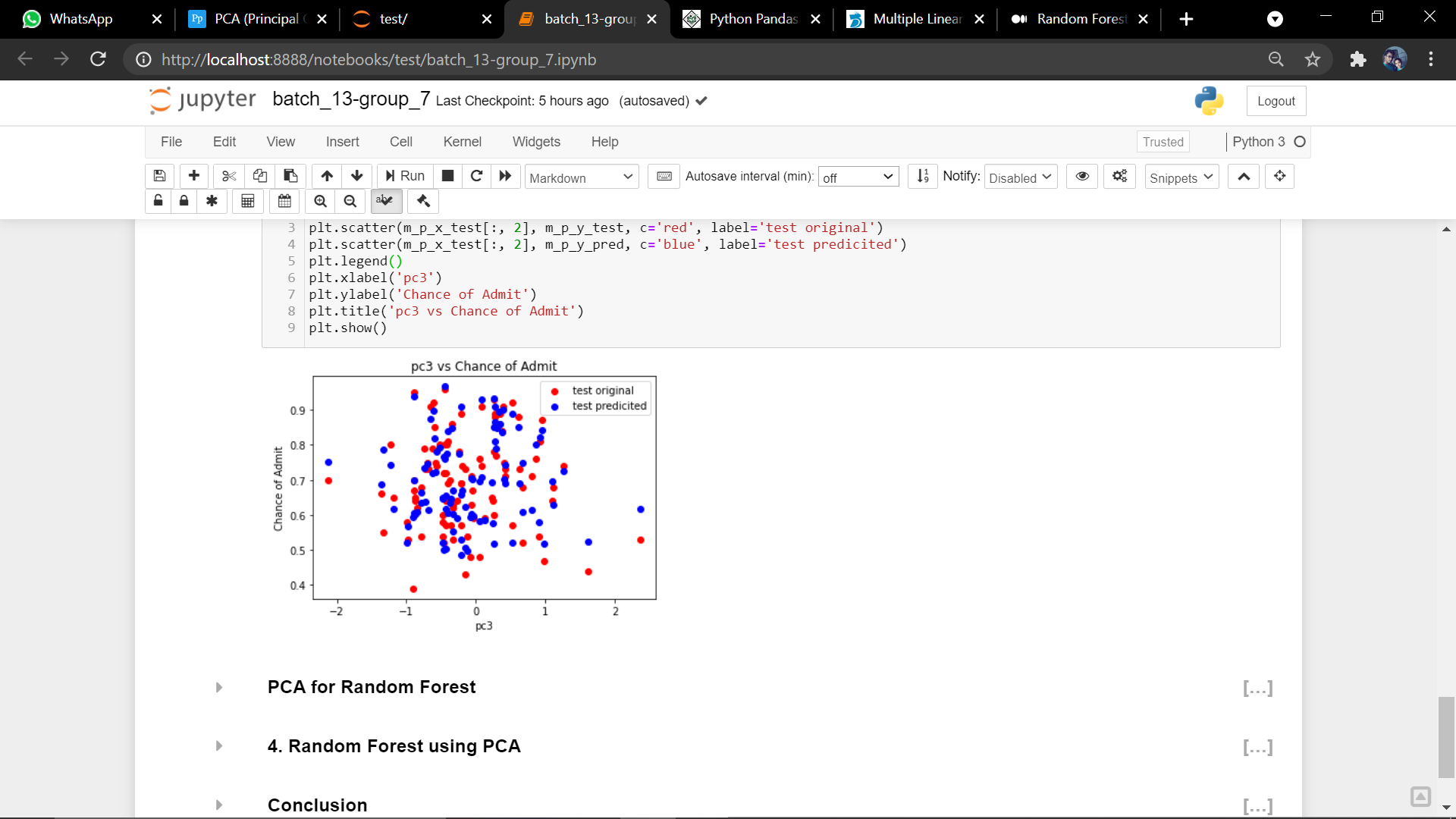
Next figures show us the plot of pc’s plot against chance of admit and errors:



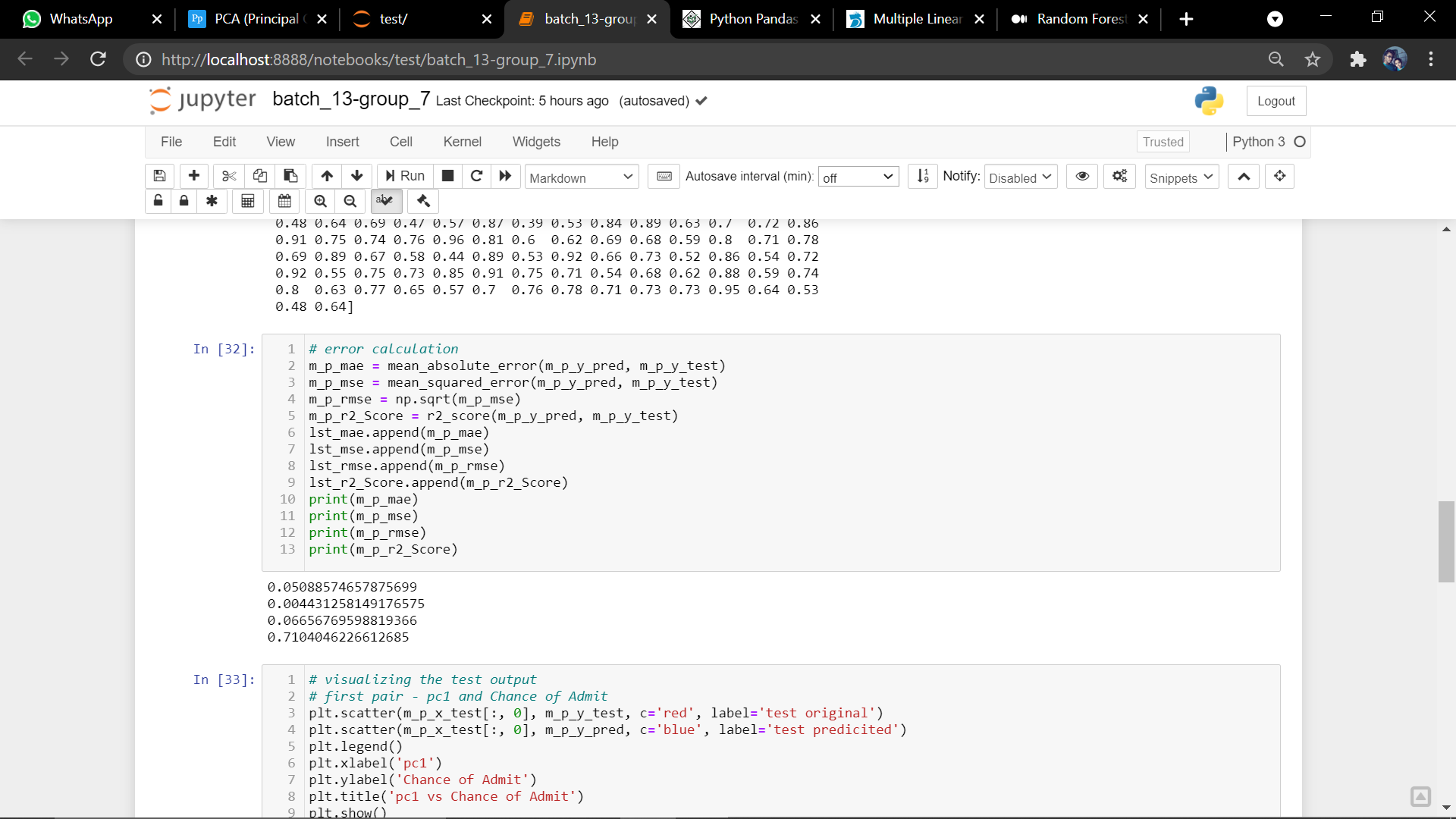
Figure



Figure



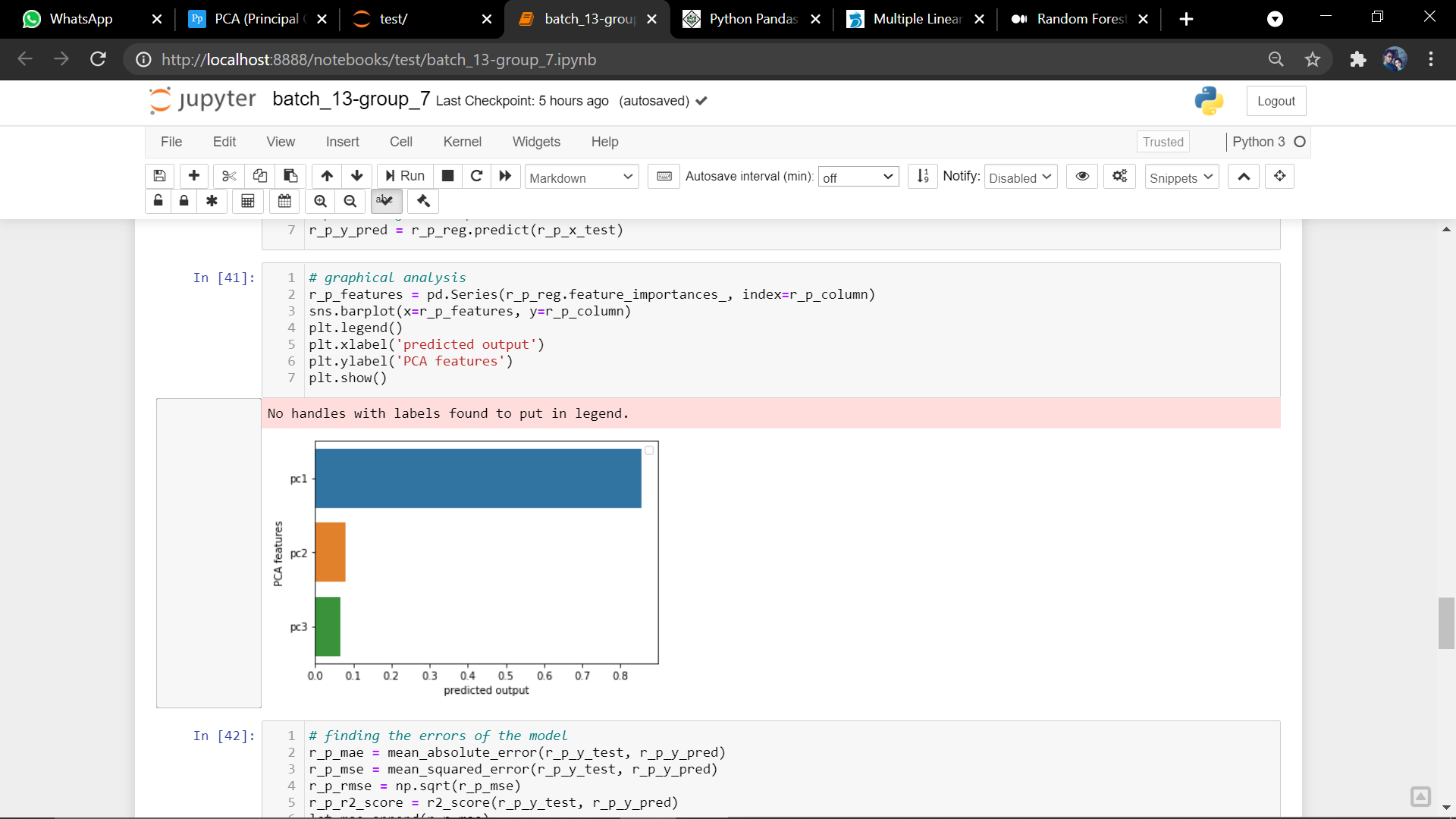
Figure



Figure

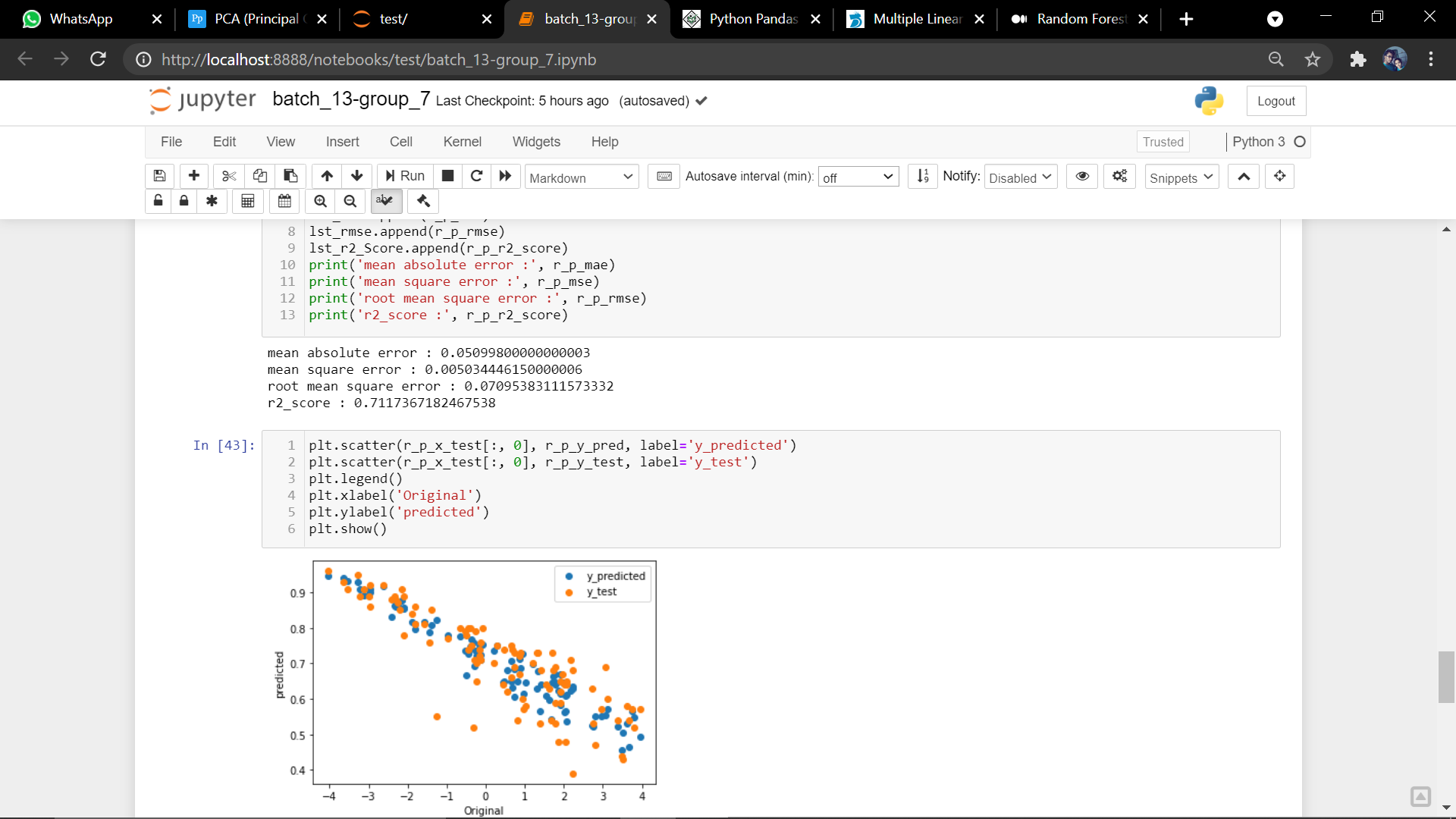
## Random Forest using PCA:

As the both are discussed earlier hear also we are moving directly to analysis part:

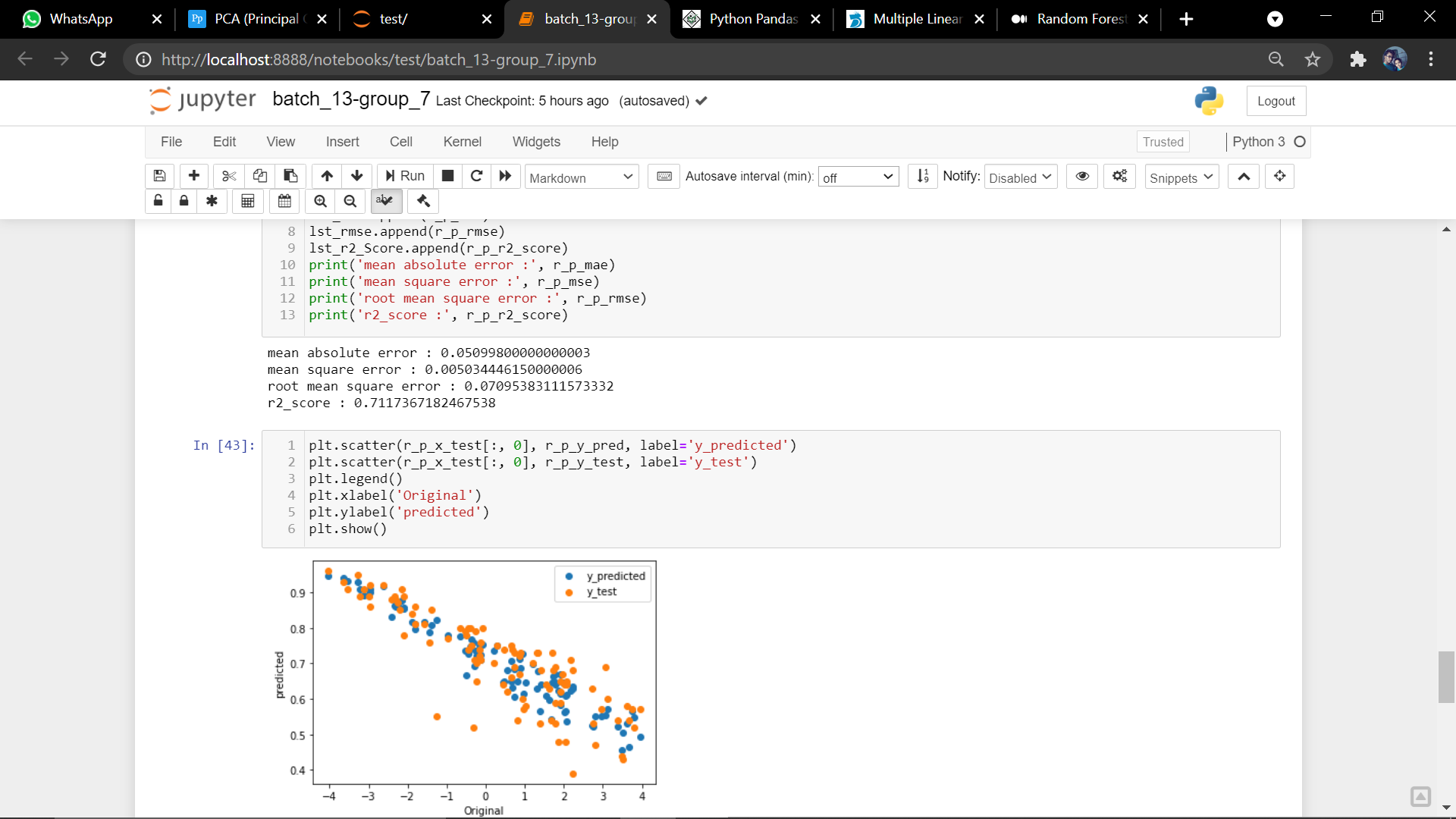


Figure

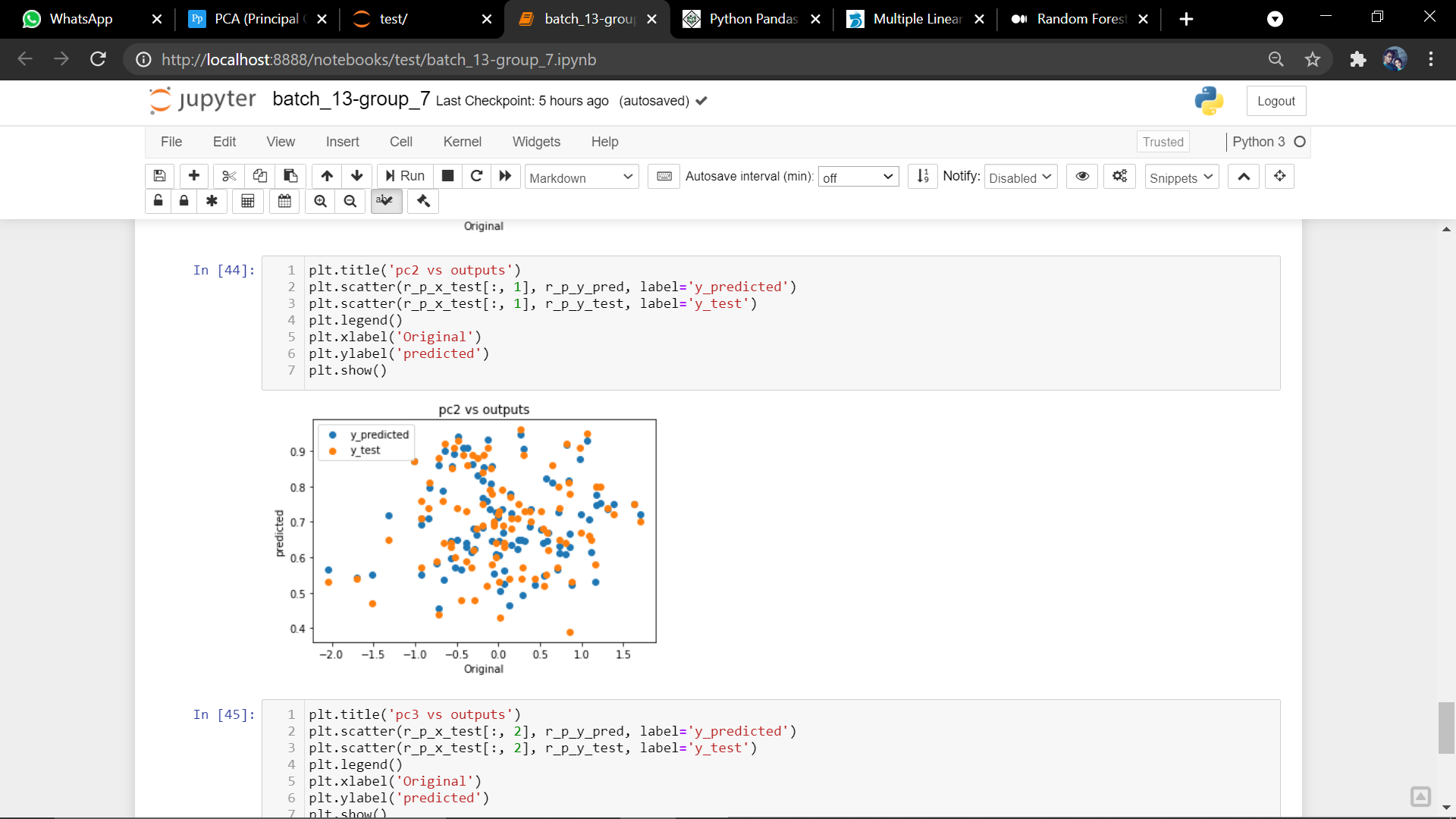
In the above figure we are showing the feature importance of the pca values after the fitting of random forest model.



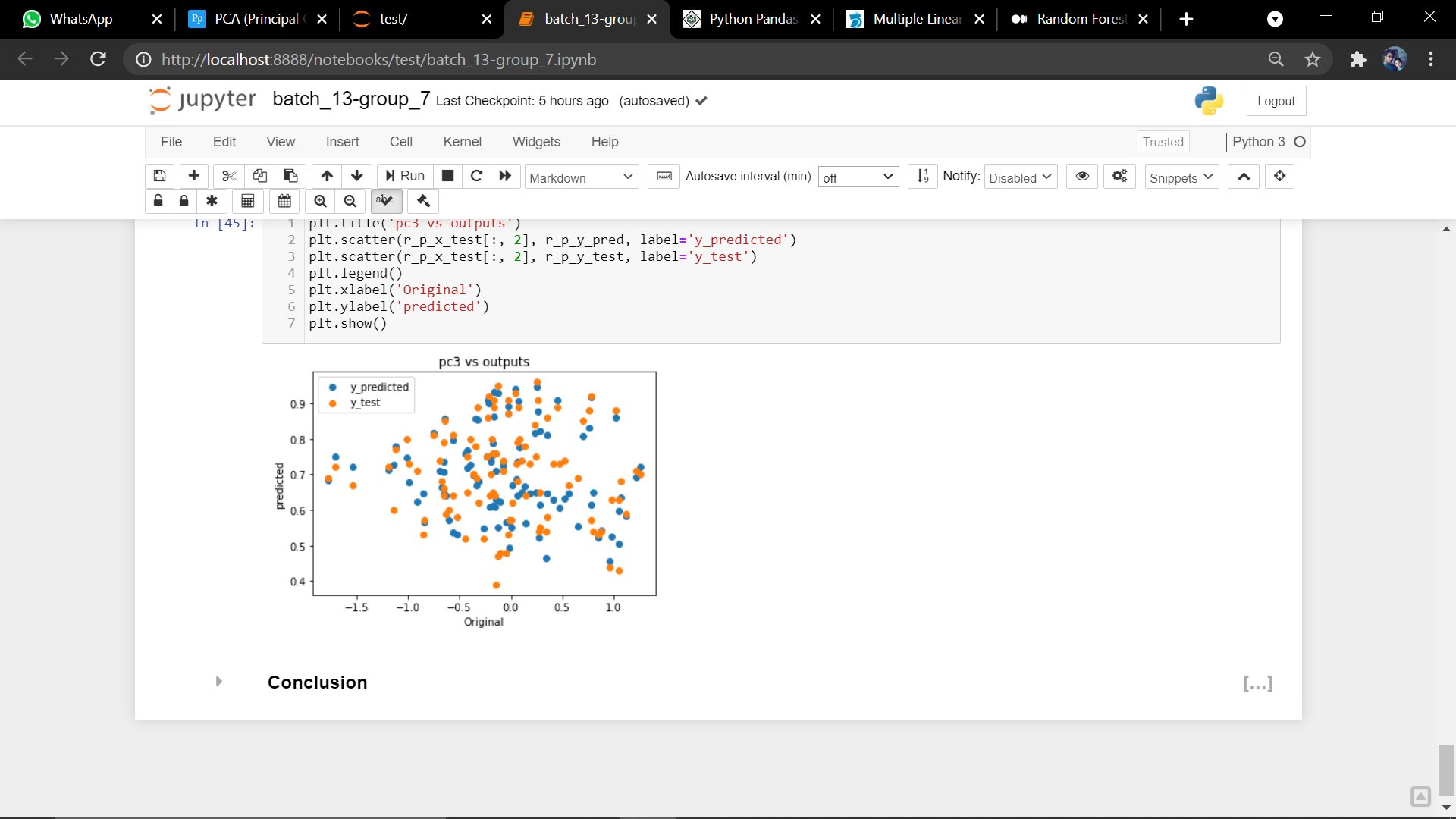
Figure



Figure



Figure



Figure

In the above figures we had show the plot of chance of admit against the pca components.

With pca components we loss our accuracy but we got good timing imporovement and less computation resources are required after the pca.

# Conclusion:

We can observe that our model is well fit with Multiple linear regression according to the assumptions and values as well as the plots too. We can see the cgpa vs chance of admit plot is similar to linear line, and remaining plots also looks linear but not accurate and we get an r2 score of 80% in MLR where as 71% in MLR using PCA so from this we can say that PCA will reduce the accuracy considerable but it will decrease required computation resources and time. We finally conclude that MLR will be best fit for our problem statement.

# References:

[1] <https://realpython.com/python-matplotlib-guide/>

[2] <https://pypi.org/project/seaborn/>

[3] <https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/>

[4] <https://www.investopedia.com/terms/m/mlr.asp>

[5] <https://dimensionless.in/multiple-linear-regression-assumptions-of-linear-regression-a-z/>

[6] <https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f#:~:text=Random%20forest%20is%20a%20Supervised,trees%20while%20building%20the%20trees>.