**Chapter 3**

**Problem 3: Stars and Galaxies**

As human open his eyes to the world, sky is beyond of his dream. Objects in the sky were mysterious, and myths belonged to stars. After telescope and other vision assistance devices invitation human can explore space by pictures. Nowadays aerospace shuttles, space explores and astronauts takes millions of images that need to be examined to realize how space changes.

It’s time for AI to show out, instead of examining by human, AI can classify and detect objects in the space. This can lead systems to perform faster and dive into details. Although human can detect more precise, sending data like images through space even to the nearest space station, expenses and takes too much time, so processing and predicting these images and sending results, is much more efficient.

**About Dataset - Dataset**

This dataset contains 10 thousand images included 2 classes, Stars and Galaxies. Each image in dataset contains 512 pixels as height, 512 pixels as width and 3 color channels and pixel values are between 0 and 255. There are 8000 images set aside as training data and 2000 as validation and half of each dataset belonged to each class respectively.

**Introduction**

Unlike typical computer programs, Machine Learning techniques will literally learn from data. Machine Learning algorithms can actually find insights and data even if they are specifically instructed on what to look for in that data, and that's what separates a Machine Learning algorithm from a typical computer program. You're just giving the Machine Learning algorithm a set of rules to follow. Instead of actually telling it what to look for, it will find the insights on its own.

**Why do we use Machine Learning to solve physics problems?**

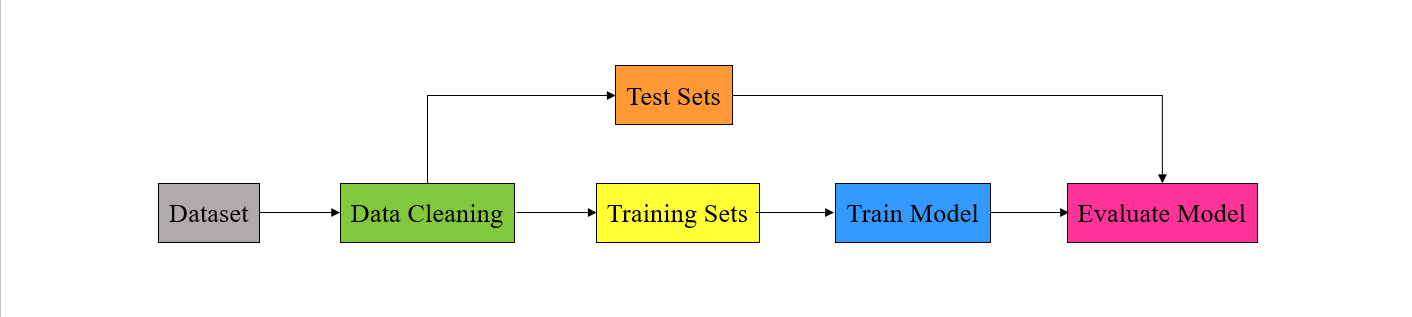
Machine learning is a method for predicting behavior or classifying data sets that, unlike common methods in physics, instead of being based on an intuitive model, uses a mathematical model and arbitrary functions to describe and predict the behavior of systems. In other words, machine learning is a search in the space of algorithms and parameters in such a way that it infers a model from the data (data-driven model) and based on that, predicts or categorizes the studied system. For example, using neural networks as one of the methods of traditional machines, I can perform a set of inputs based on an arbitrary number of intermediate hidden layers to the output image results. In the input and output data that are entered quantities, the middle layers do not necessarily have meanings and other adverbial expressions on them. For this reason, I can choose the number of intermediate layers and the number of nodes in each layer at will, and this approach is completely acceptable in the input to the output image. In particular, the relationship between the data is so complex that the models created with a limited number of adjustable settings express this relationship with sufficient accuracy, the efficiency of the methods using machines can be very important.

**Classification Problems**

High-level machine learning consists of supervised and unsupervised learning. Supervised learning means that we label historical data and use it to inform our model. We call that label or something that we want to predict as a target. So, in supervised learning, we have a specific goal (target) for that past information, and in unsupervised learning, we don't have a specific goal. In supervised learning, we have classification and regression. Classification problems are problems where our goal is a category (that is, we want to see what category it belongs to. It's usually True or False, but it can be multiple categories). Regression problems are those where our target is a numerical value.

**What are we going to do in this Chapter?**

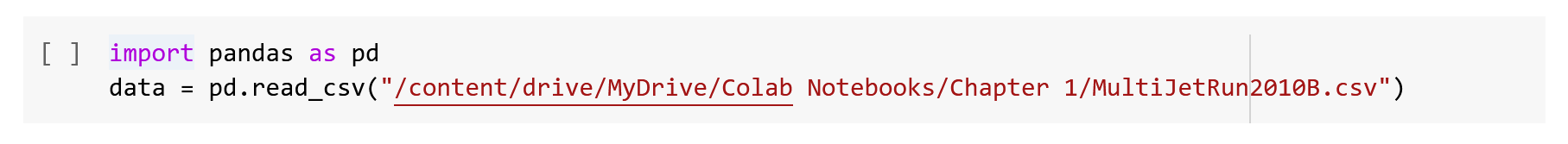
We have a dataset from the Kaggle website and then clean that data. After that, we split our data into two groups (train and test). Then we train our model on the training set and after that, we evaluate our model with the test set we have.





**pandas**

Now, we'll show you how to use pandas to read a CSV file.

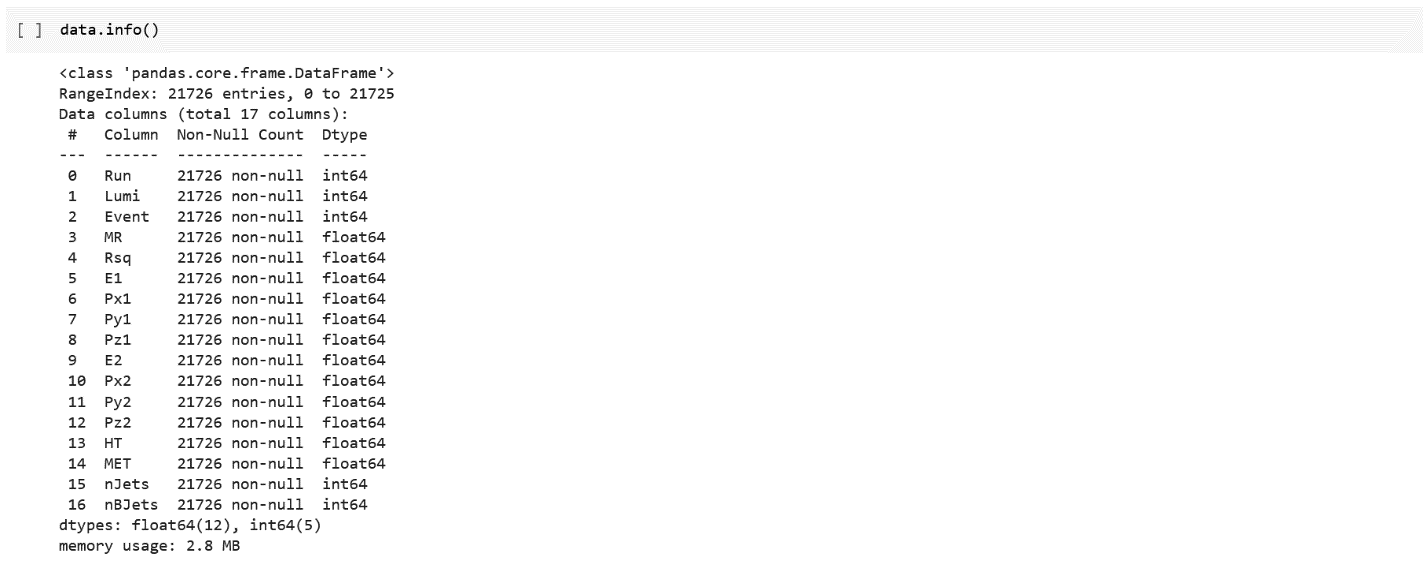
The **.read\_csv()** function is used to load data from a CSV file. Python has become one of the most popular programming languages today due to its power and speed of learning. The reason for this popularity is probably the extensive free libraries and game sources (Open Source) that Python has provided to programmers. One of these functional libraries is pandas, which is a must-know for any Python who intends to analyze data.

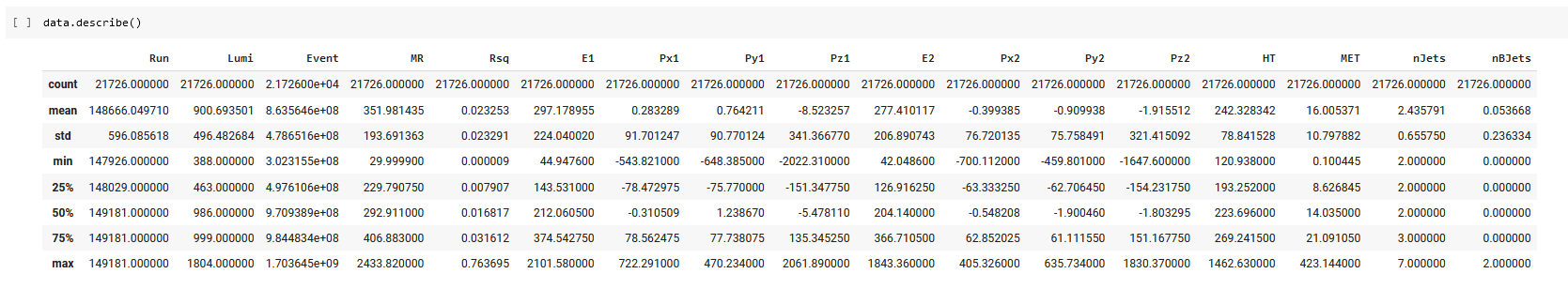
**NumPy**

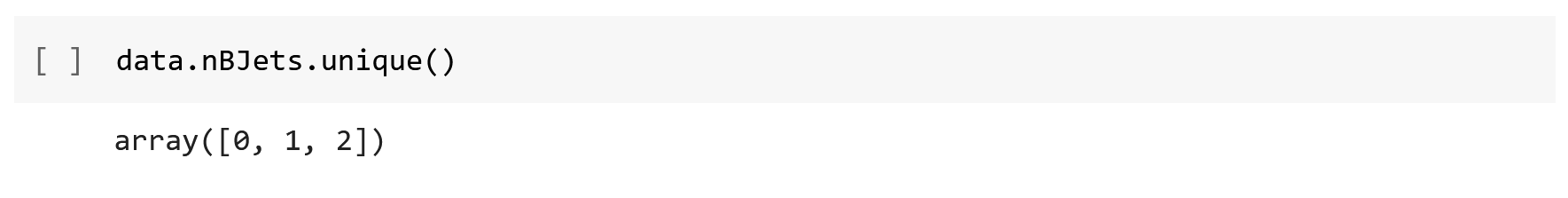
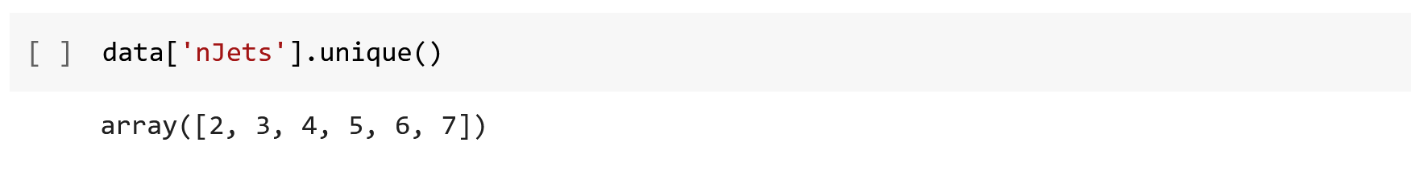
In general, the goal of learning any topic is to simply do things. Therefore, the goal of learning NumPy is to acquire the necessary information to speed up calculations related to arrays and matrices. On the other hand, this library helps you to perform the calculations related to machine learning more accurately and to be more sure of the intended output.

**Step 1. Data Cleaning**

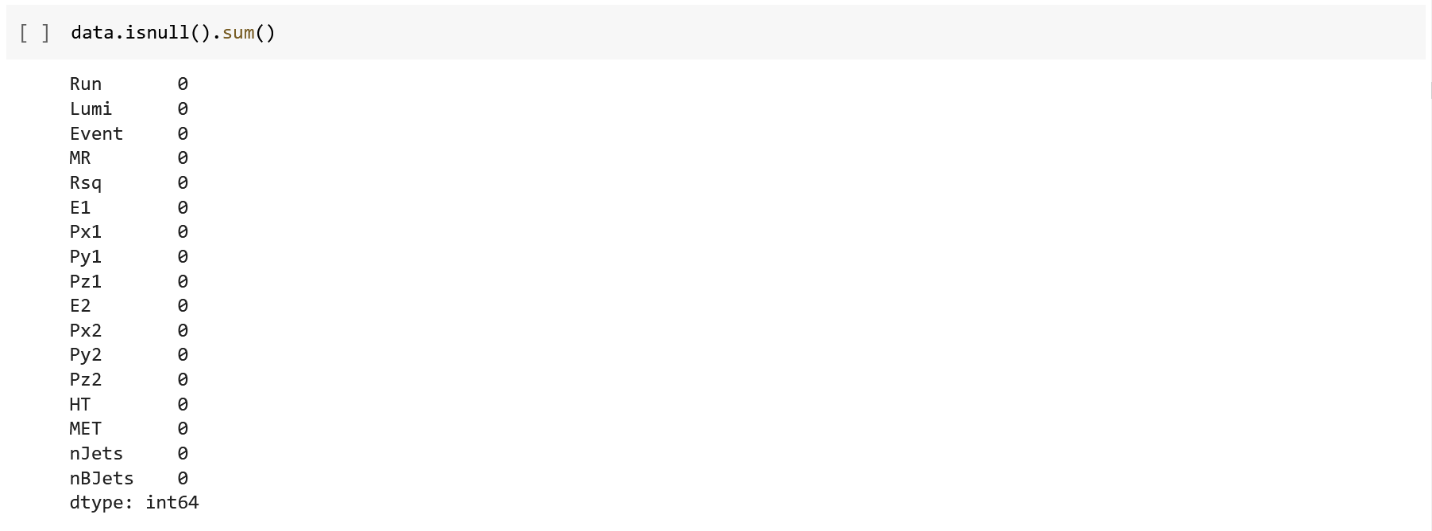
We want to solve a classification problem and the first thing we want to do is clean the data because a lot of times we need to normalize our data. The first thing to do is **.columns** and with that, we get a list of returned columns that give us the column labels of the given Dataframe. Dirty data simply means data that is wrong. Duplicate records, incomplete or outdated data, and incorrect parsing can make data dirty. This data must be deleted. Data cleaning (or data cleansing) refers to the process of "cleaning up" this dirty data by identifying errors in the data and then correcting them [3]. Data loss is consistently a problem in real scenarios. Fields such as machine learning and data mining face serious problems in the accuracy of their model predictions due to poor data quality caused by missing values. In these fields, missing value treatment is the main point to making their models more accurate and valid [4].

**.info()** method prints information about the DataFrame. In the next steps we want to make **nJets** our labels, so make sure it's an **integer** in the data.info() report.

**.describe()** method returns a description of the data in the DataFrame.

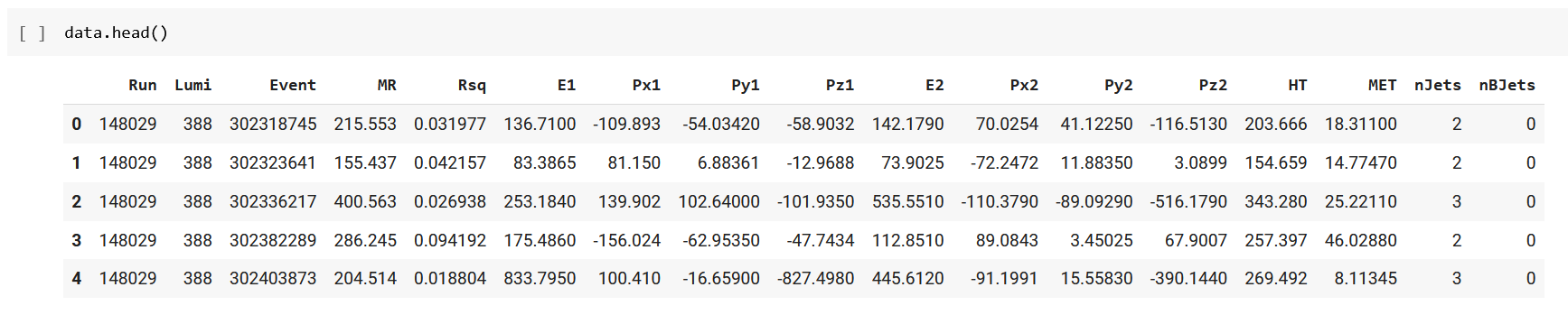
**.unique()** is used to find unique elements of an array.

**.isnull().sum()** returns the number of missing values in the data set.

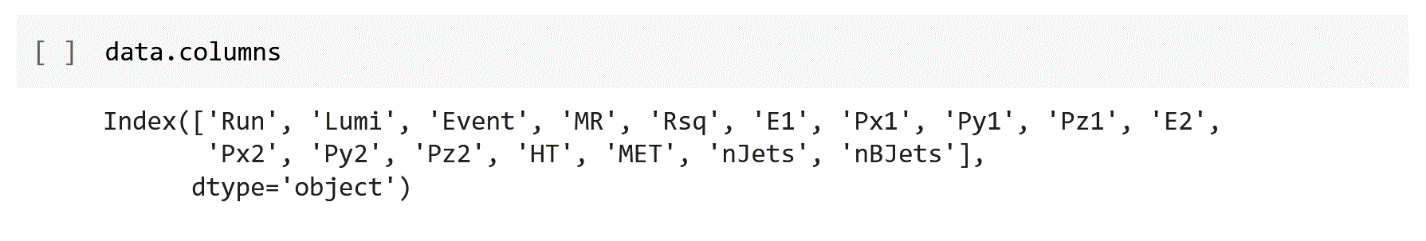


**.shape** returns the shape of our dataset.

**.head()** function is used to get the first n rows.



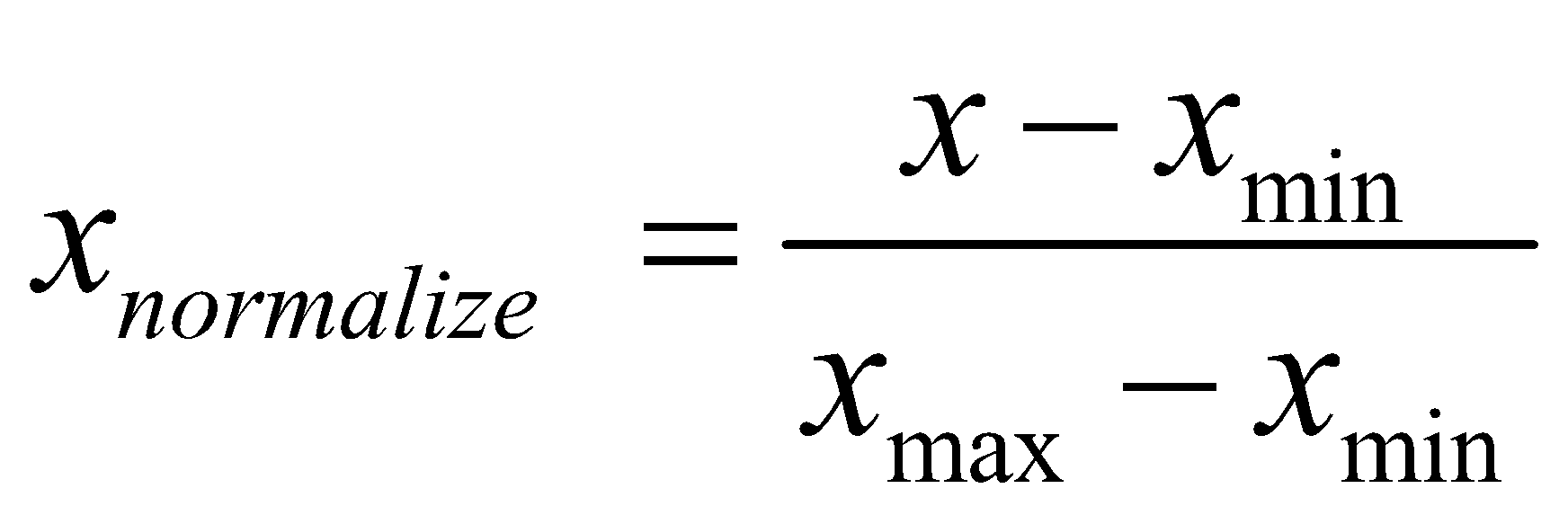
**.columns** return the column labels of our dataset.



**Normalization of Data**

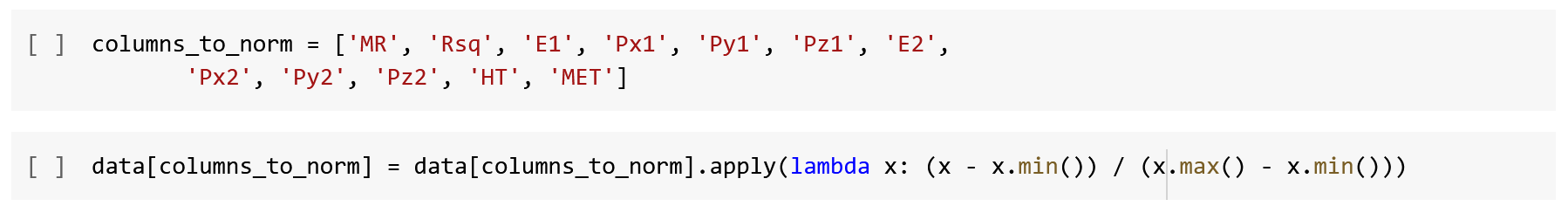
So we pass in the columns to normalize into data and then I'm going to apply a custom function (lambda function is just like any normal python function, except that it has no name when defining it, and it is contained in one line of code).

After that, to normalize the columns, I do the following:

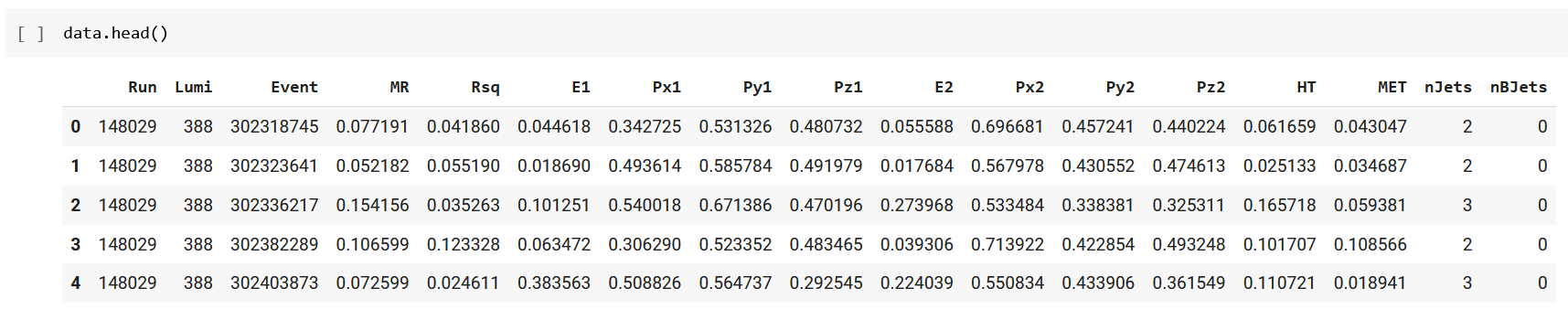


**Why Do We Need To Normalize Data in Python?**

One of the most important topics in the field of Machine Learning and Data Mining, especially in the Data Preparation section, is the topic of Re-scaling of data, which is usually done by There are two methods of Standardization and Normalization. The meaning of normalization is to transform the data into the domain [0 and 1]. Each of the data recorded in the dataset will change to a range between zero and one. This makes the data fall under a shorter domain and the model is trained better.

Now we define columns\_to\_norm and we want to create a list of columns to normalize.

And that should normalize the columns, so let's look at the head again.

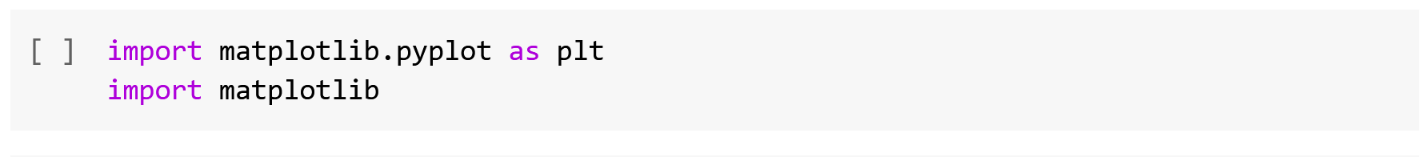


**Step 2. Data Visualization**

Data visualization tools provide a way for us to easily identify outliers, see trends, and recognize patterns in data using charts and graphs. Data visualization helps us understand what the data is trying to tell us, and present it in a way that is accessible to a wide range of audiences (not just data professionals). When the volume and complexity of the data are high, it will be very useful to display them in the form of an image. By visualizing data, patterns, trends, and correlations between data can be better understood and communicated effectively. We will use several data visualization libraries in Python. For example, we use Matplotlib and Seaborn in this chapter.

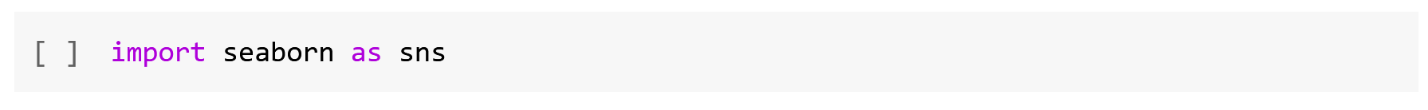
**Matplotlib**

Matplotlib is one of the main libraries of the Python programming language and is the most extensively used among Python visualizations. Matplotlib helps us work with data frames more efficiently. We can create 2D plots of arrays, charts, and graphs in Python and it helps us understand our data set better. The introductory text was written in 2008 by John D. Hunter (1968-2012), the original author of Matplotlib.

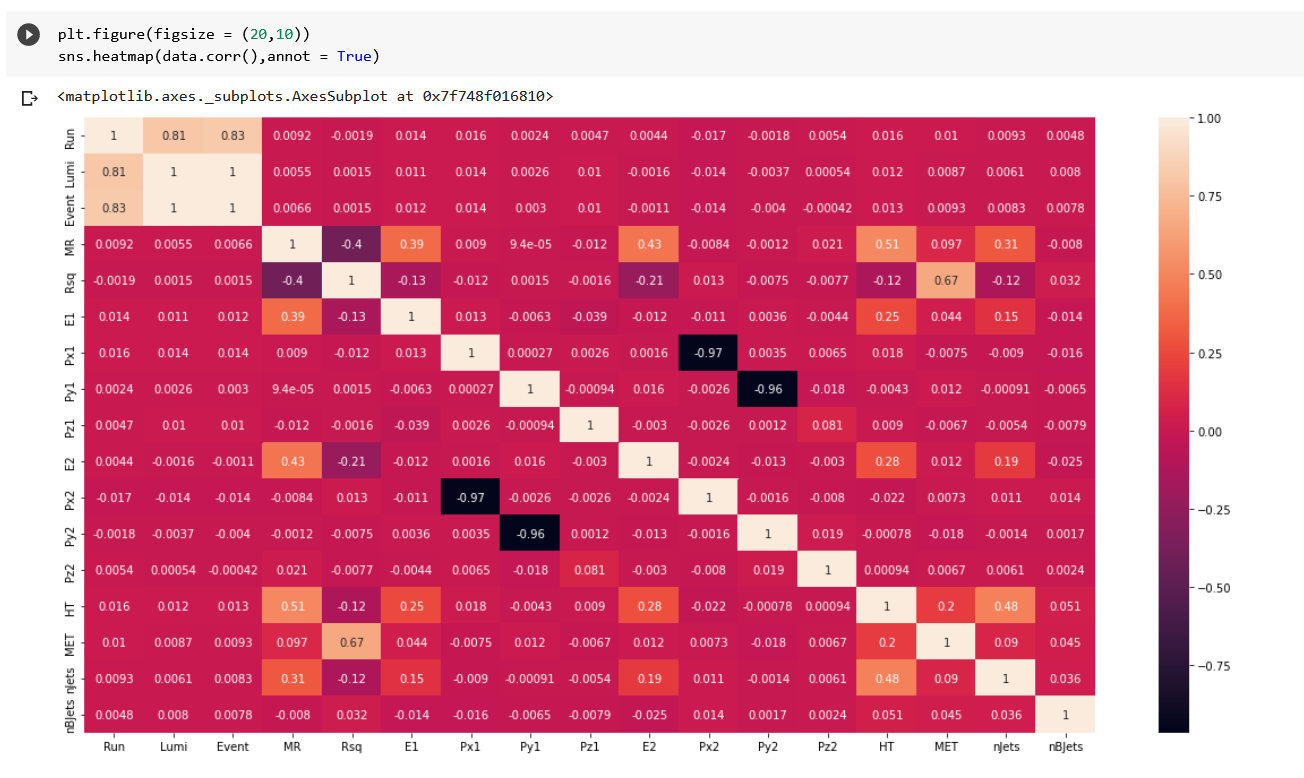


**Seaborn**

The most important feature of Seaborn is to provide professional and high-quality charts. Also, Seaborn's facilities are so many that it answers almost any need and even the smallest details can be changed. This high-level library facilitates the process of drawing statistical charts.



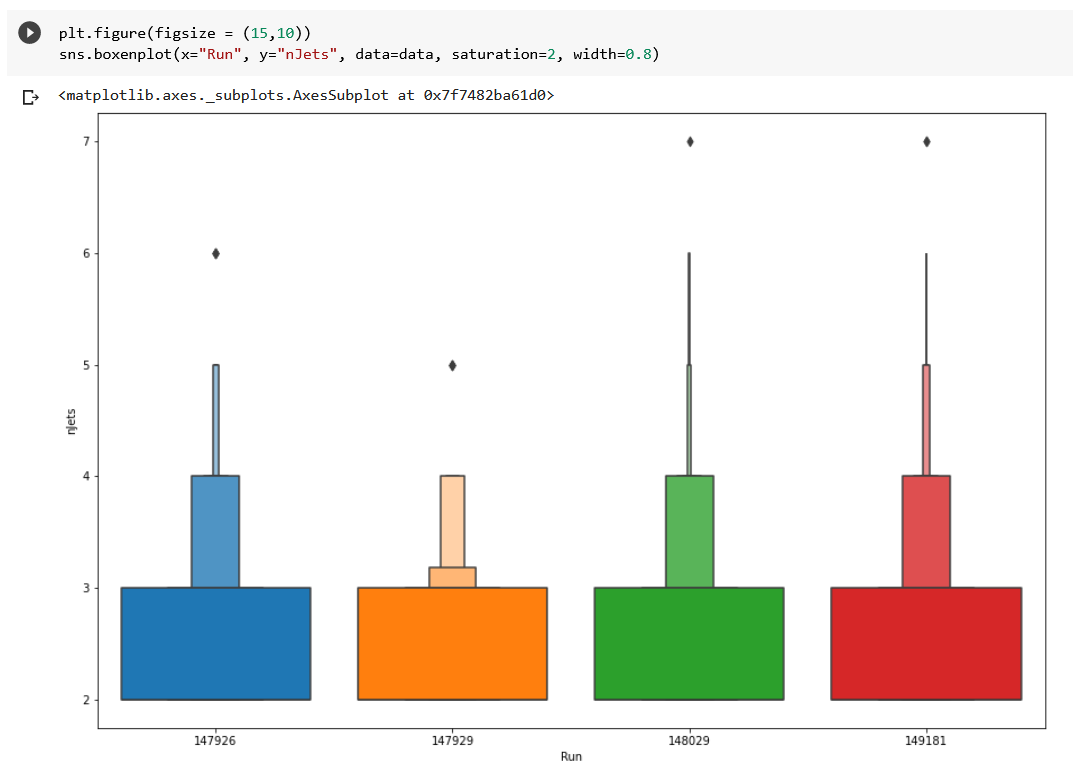
**heatmap**

Heatmap is a color matrix. This chart contains values that represent different shades of the same color for each value, and commonly darker shades represent more values than lighter shades.

Line 1: first of all, we want to create a new figure, so we use **.figure** for that. Then we set the desired size of my figure with **figsize**.

Line 2: when we say **data.corr()** it finds the correlation of each column in a dataset and the **annot** is for an array of the same shape as data which is used to annotate the heatmap.

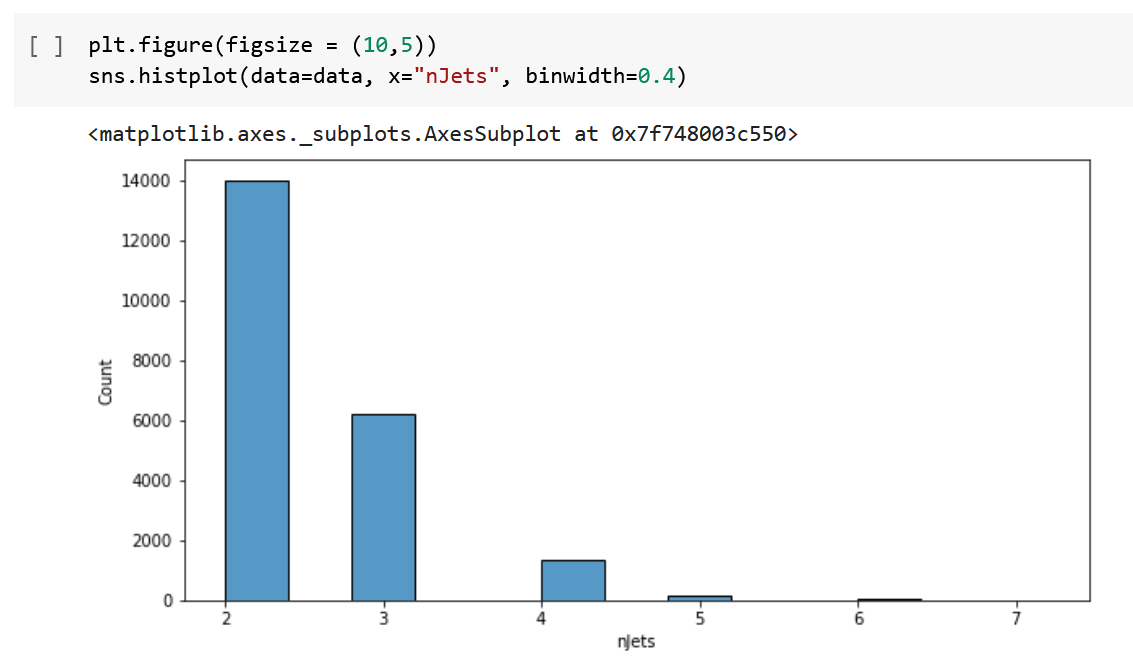
**boxenplot**

Boxenplots (or letter-value plots) are an evolution of boxplots designed to visualize distributions more accurately. The Boxenplot is very analogous to a boxplot, except for the fact that it plots different quartile values. By plotting the different quartile values, we can find out the shape of the distribution, especially at the head and tail ends [5].

Line 1: first of all, we want to create a new figure, so we use **.figure** for that. Then we set the desired size of my figure with **figsize**.

Line 2: The **saturation** is the proportion of the original saturation to draw colors at and **width** is the width of a full element. With the x and y elements set as you can see in the plot, on the x-axis, we have four main numbers for the Run column in the dataset, and for the y-axis, we have the same conditions with its numbers.

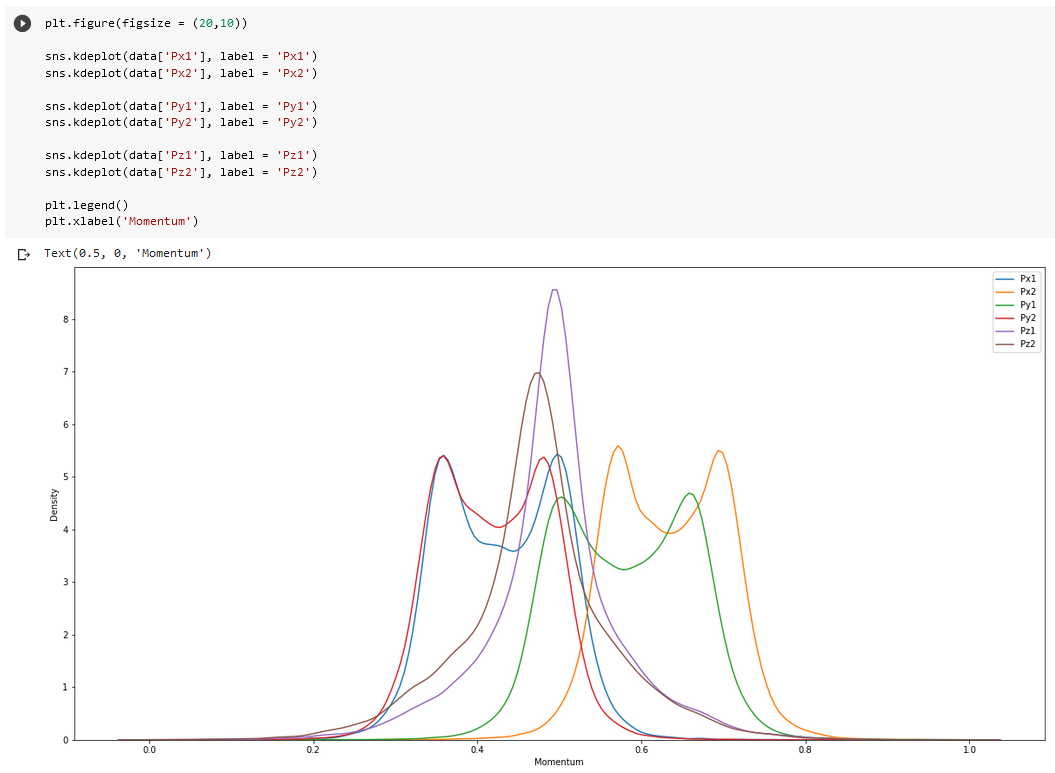
**histplot**

A histogram is a set of columns next to each other, the height of each column is different from the others, and this height in each column indicates the frequency of that column's category. Column charts can be considered the most widely used type of charts in mathematics and statistics. In different industries, these charts are used to define and categorize data. The histogram is actually a column chart; In this diagram, each axis represents information from the data. For example, in a category of statistical data, each column can be considered as a category of this data. The height of each column also indicates the frequency of each category in the available data.

Line 1: first of all, we want to create a new figure, so we use **.figure** for that. Then we set the desired size of my figure with **figsize**.

Line 2:We saw the y-axis in the previous diagram. Let's take a look at the values of each of its numbers. with **binwidth** is the Width of each bin.

**kdeplot**

KDE stands for Kernel Density Estimation. A two-dimensional kernel density plot can be plotted using the kdeplot function, and as you can guess, this plot is an estimate of the kernel distribution that represents the probability density function of continuous or nonparametric data variables.

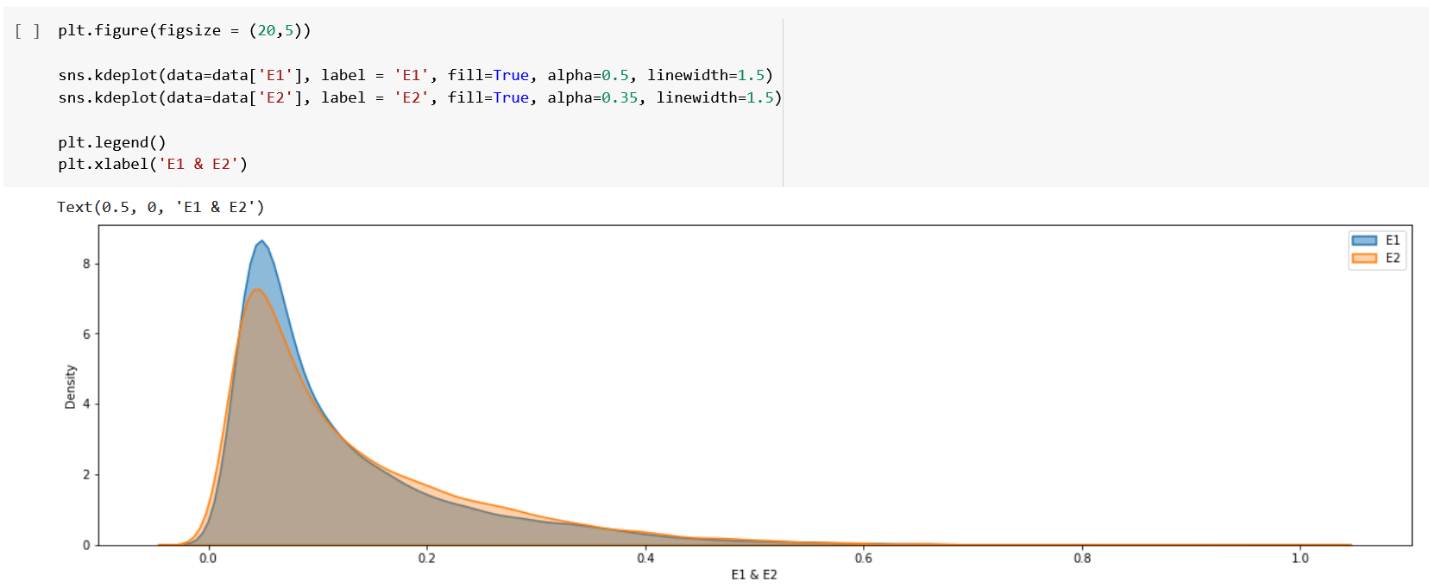
Line 1: first of all, we want to create a new figure, so we use **.figure** for that. Then we set the desired size of my figure with **figsize**.

Line 2-7: First of all, we're going to set one of our columns to plot, then with the **label**, as you can see we're going to create a list of labels on the right side of our plot.

Line 8: with **.legend()** we automatically detect the elements that are shown in the plot.

Line 9: with **.xlabel** we set the label for the x-axis.

In this beautiful plot, we see the difference between six columns in the dataset.



Line 1: first of all, we want to create a new figure, so we use **.figure** for that. Then we set the desired size of my figure with **figsize**.

Line 2-3: Just like before, first of all, we place one of our columns to draw. Some elements are the same as before, let's look at the new ones. for **fill** If True, fill in the area under univariate density curves or between bivariate contours. If None, the default depends on multiple. With **alpha**, we can change the intensity of the color used. And with **linewidth**, I change the width of the line on the plot.

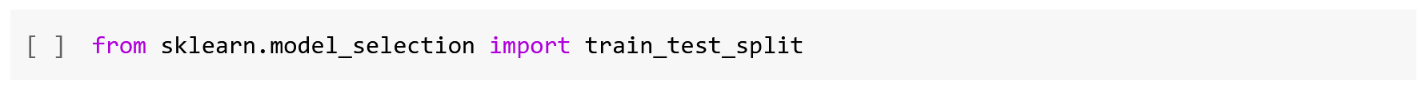
Line 4: with **.legend()** we automatically detect the elements that are shown in the plot.

Line 5: with **.xlabel** we set the label for the x-axis.

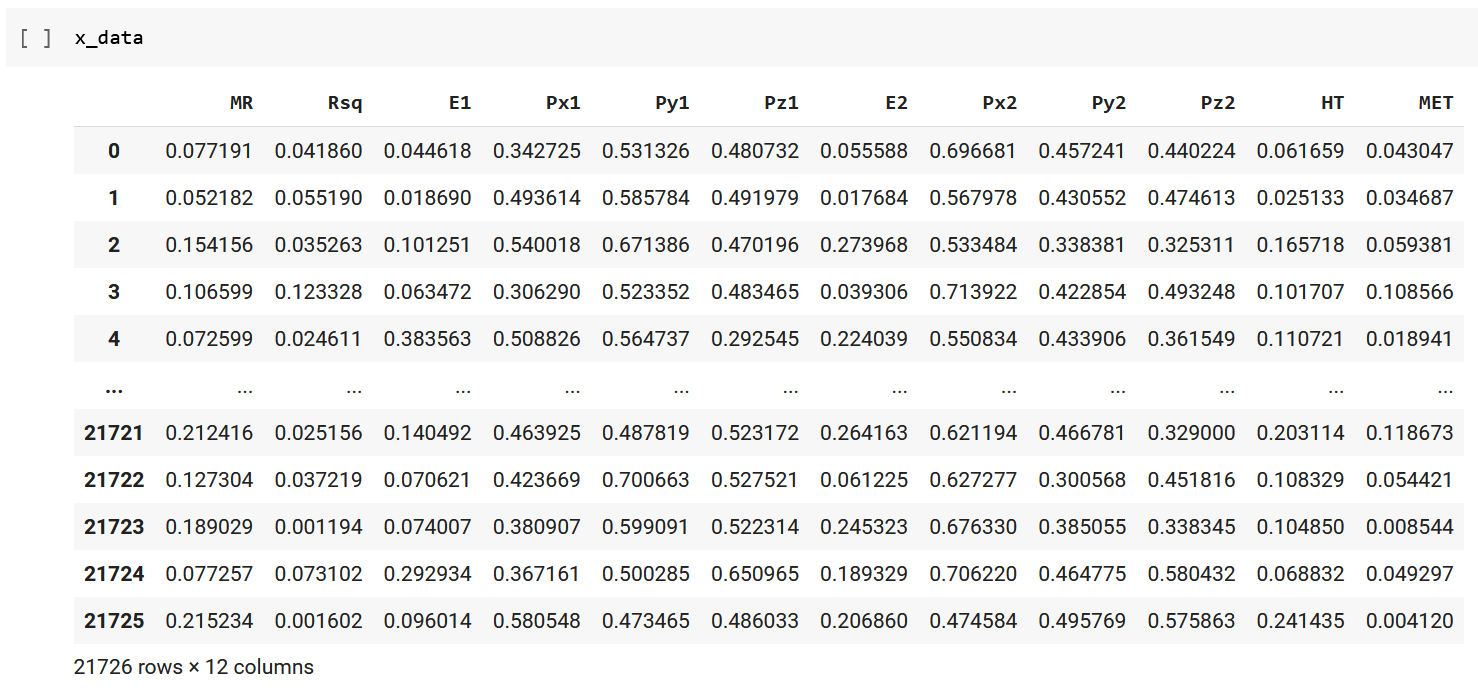
In this plot, we see the difference between E1 & E2 columns in the dataset.

**Step 3. Split Training and Testing Data Sets**

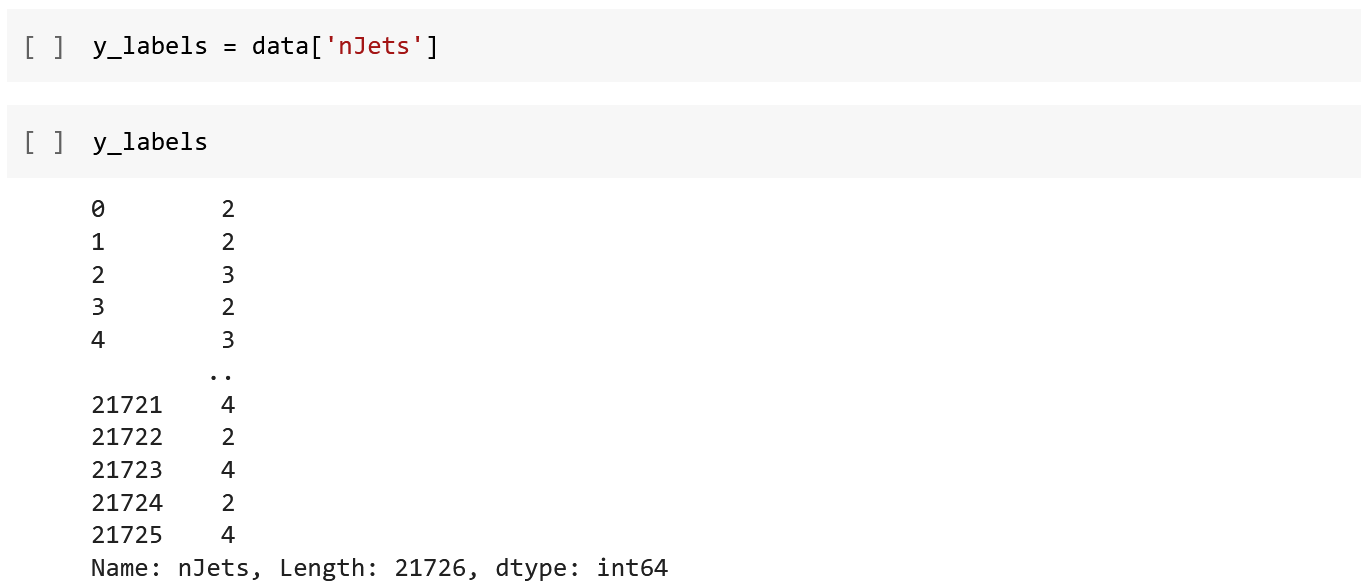
As we saw at the beginning first thing you have to do is actually acquire the data and this really depends on what task you're trying to solve. Then the next step as we saw in the previous section is to clean and organized the data. So once you do that, you need to do what's called a Train-Test Split, and as the name of this step indicates you're going to split your data into a training set and a testing set. There are lots of split ratios you can use. A really common split ratio is to have 30 percent of your data be tested and then 70 percent of your data be training but it really depends on the situation, how clean your data is, how much data you have, etc. Preprocessing is one of the important issues in machine learning algorithms. At this stage, we need to pre-process the data to provide a suitable prediction. For this, we use the preprocessing module of the scikit-learn library. There are a number of methods whose job is to predict the outcome of some events. In fact, to find out whether the model we have reached is good enough or not, we can use a method called Train-Test Split.



with **.drop** we can drop specified labels from rows or columns.

and now if we take a look at **x\_data**, it no longer has the dropped ones.

so now we can set our **y\_labels**.

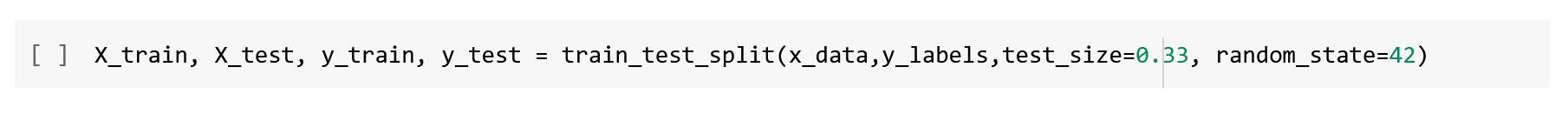


Now we use the following line, one of the great examples on the **scikit-learn** website. For more information please check the references [6].

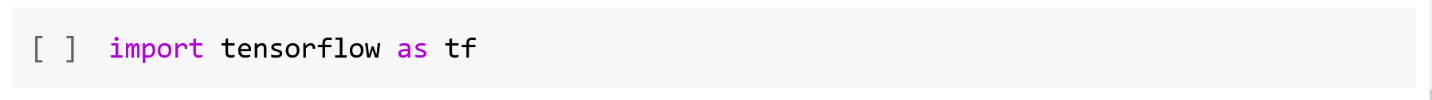


**test\_size:** This can be float or int but will be set to 0.25 if default = None.

**random\_state:** Controls the combination applied to the data before the split is applied.



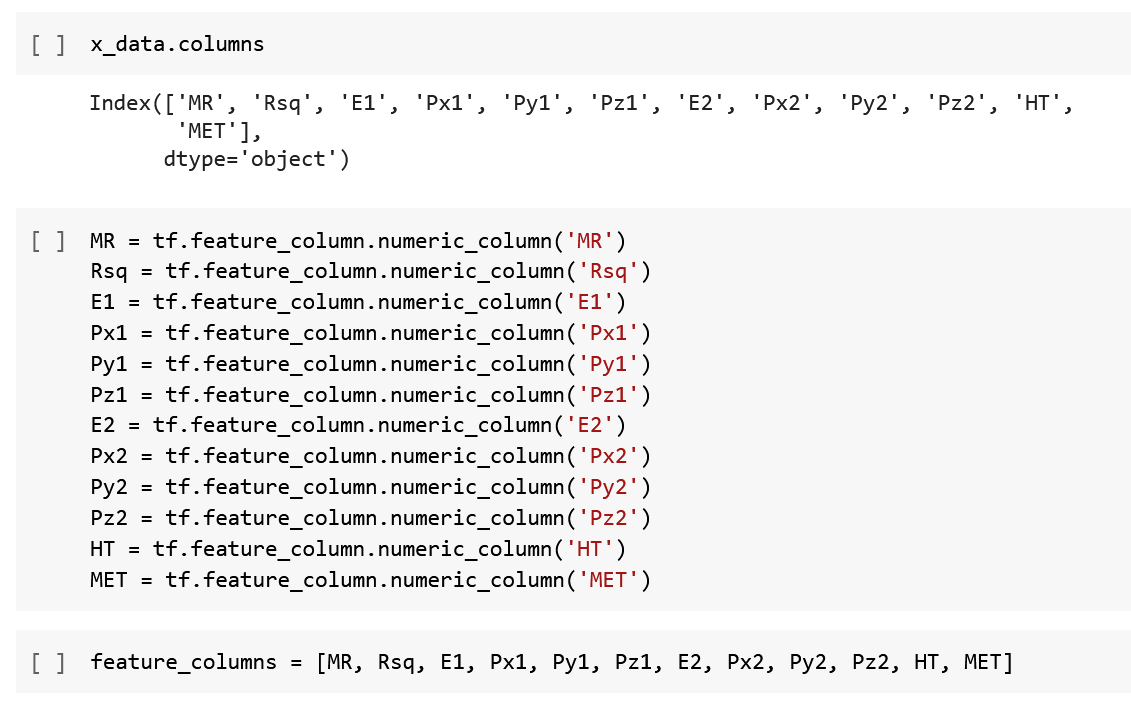
**Step 4. Training Models**

It's time to train our model on the training data. TensorFlow is the most widely used library in developing deep learning models. Many developers use TensorFlow in their sensitive and advanced projects. One of the main reasons for TensorFlow's popularity is its open source. Tensorflow is superior to other libraries in numerical calculations, which is one of the most vital needs of deep learning. TensorFlow combines machine learning and deep learning (aka neural networks) models and algorithms and transforms them into a useful and usable form.

**Define the feature columns**

Each tf.feature\_column specifies a feature name, its type, and any input preprocessing. Consider feature columns as intermediaries between raw data and estimators. Feature columns are very feature-rich and enable you to transform a wide variety of raw data into formats that estimators can use, allowing for easy testing [7]. The feature column describes a set of transformations to the inputs. we apply this for each of these continuous columns. we create a new variable for each column. we use tf. feature\_column. numeric\_column command in this way:

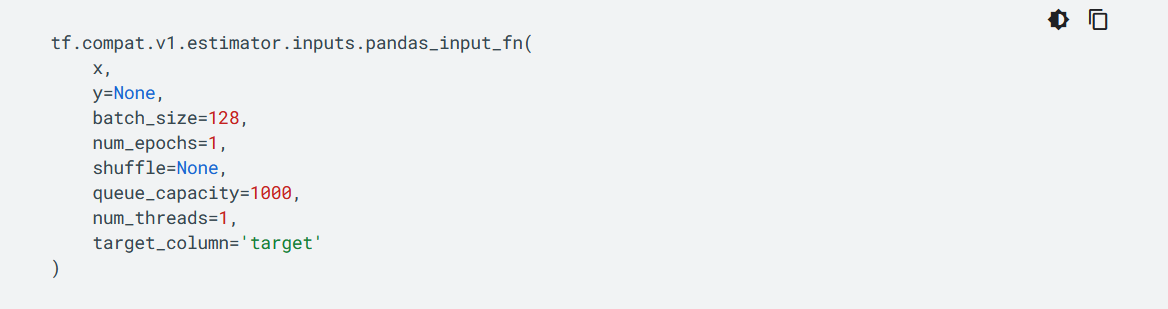
**Z = tf. feature\_column. numeric\_column(‘Z’)**

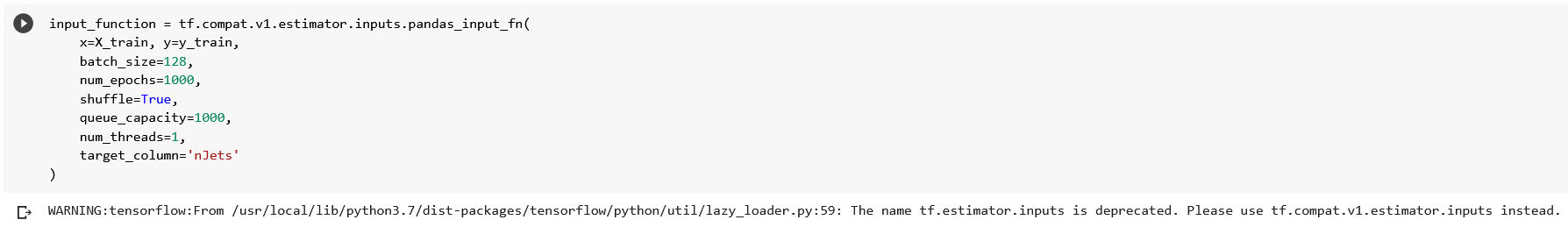


Before we begin, there is one more thing to know about Estimators in TensorFlow (tf.estimator). Estimators offer advantages, for instance, you can run estimator-based models on a local host or in a distributed multi-server environment without changing your model. Additionally, you can run estimator-based models on a CPU, GPU, or TPU without recoding your model and it provide a secure distributed learning loop that controls how and when, for example, data is loaded, exceptions are handled, checkpoint files are created and crashes are recovered, and summaries are saved to TensorBoard. Prebuilt estimators encode best practices and benefits such as determining where to execute different parts of the computational graph and implementing strategies on a machine or a cluster. They are also terrific for writing events and universally useful summaries.

There's a great explanation for it in the guide on the TensorFlow website. For more information please check the references [8].

So now we are ready to start our work. First of all, we want to define the input function which is the returns input function that would feed Pandas DataFrame into the model.

We use the description of the tf.compat.v1.estimator.inputs.pandas\_input\_fn on the TensorFlow website. For more information please check the references [9].



Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: As we are talking about estimators in previous slides, we will now define one of the pre-built estimators here.

Line 2: We define our x and y for training.

Line 3: **batch\_size** It's the size of batches to return.

Line 4: **num\_epochs** It's the number of epochs to iterate over data.

Line 5: **shuffle** It's for whether to read the records in random order.

Line 6: **queue\_capacity** It's the size of the read queue.

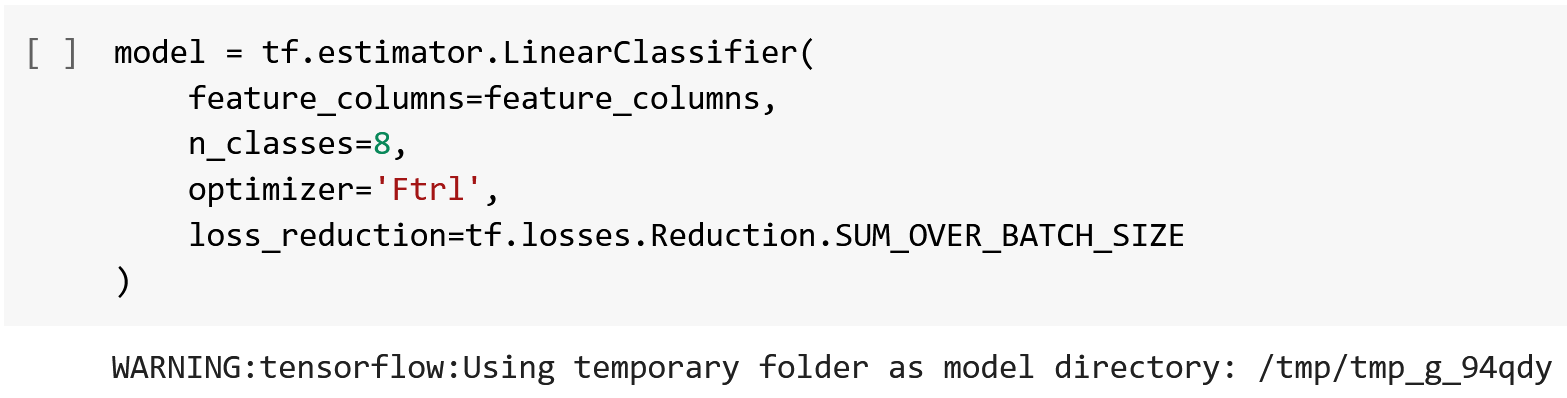
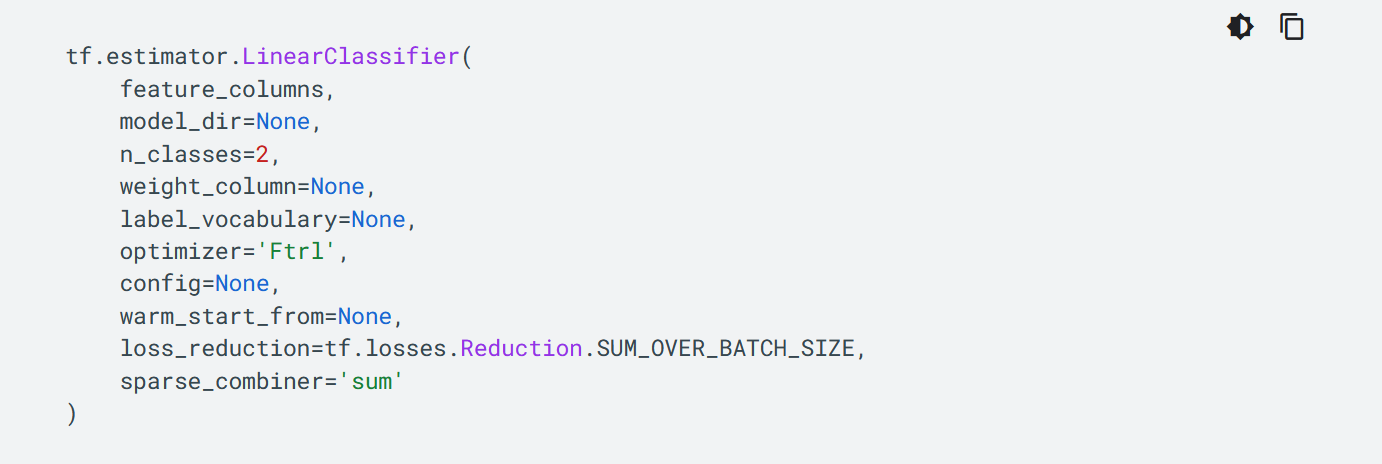
Line 7: **num\_threads** It's the number of threads used for reading and enqueueing.

Line 8: **target\_column** It's the name to give the target column y. This parameter is not used when y is a DataFrame.

If you change the **shuffle** you will get this error:

**ValueError: shuffle must be provided and explicitly set as boolean (it is recommended to set it as True for training); got None**

**4.1 LinearClassifier**

Now we use the description of the LinearClassifier model on the TensorFlow website. For more information please check the references [10].

Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: As we are talking about estimators in previous slides, we will now define one of the pre-built estimators here.

Line 2: As we saw before we're going to say feature columns are equal to the feature columns that we made earlier in the previous Lines and once you run that, you should see an output basically telling you the configuration of the model.

Line 3: **n\_classes** It's the number of label classes.

Line 4: **optimizer** It's used to train the model. Can also be a string (one of 'Adagrad', 'Adam', 'Ftrl', 'RMSProp', 'SGD'), or callable. Defaults to FTRL optimizer.

Line 5: **loss\_reduction** It describes how to reduce training loss over batch.

As you noticed we have changed the n\_classes because otherwise, we get this error:

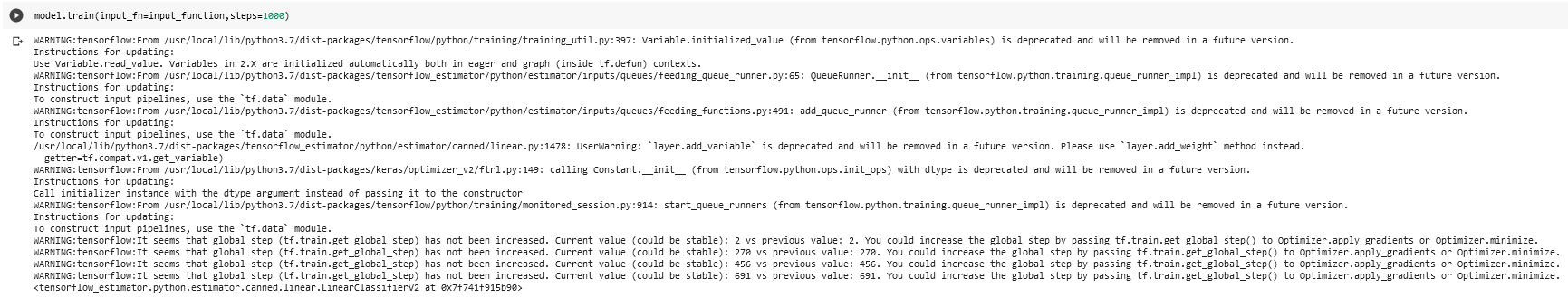
**InvalidArgumentError: Graph execution error:**

assertion failed: [Labels must be <= n\_classes - 1] [Condition x <= y did not hold element-wise:] [x (head/losses/Cast:0) = ] [[2][2][2]...] [y (head/losses/check\_label\_range/Const:0) = ] [1] [[{{node Assert}}]]

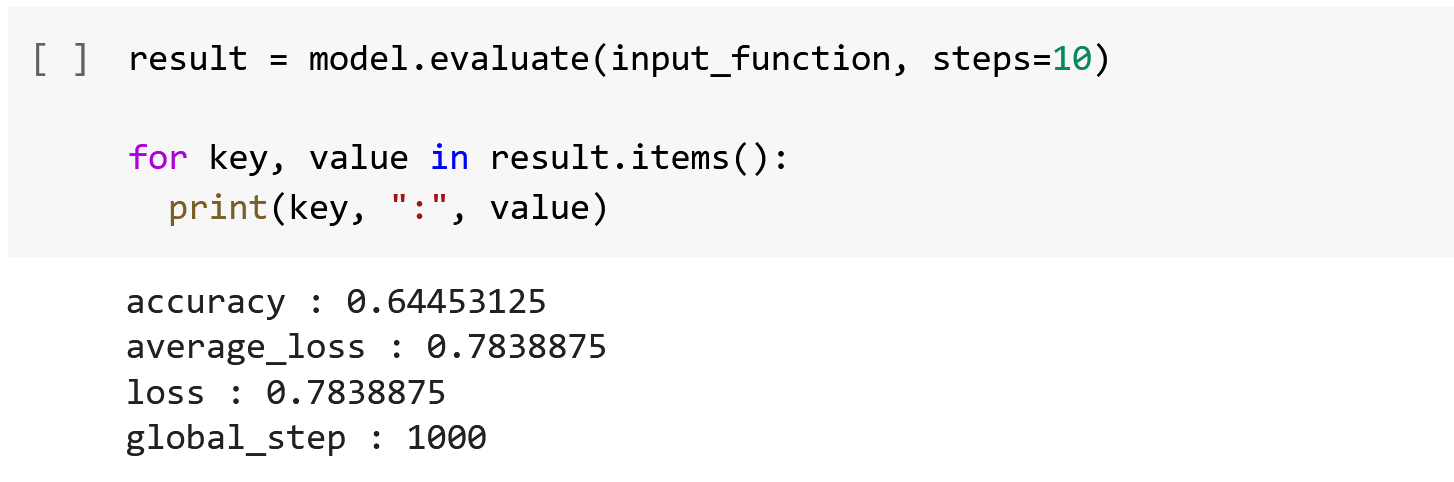
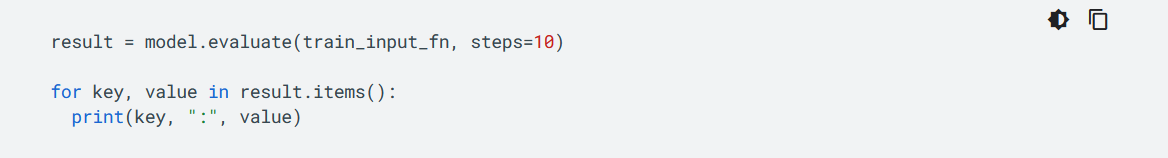
You can read more information about this error in this link:

<https://stackoverflow.com/questions/65545303/tensorflow-invalidargumenterror-assertion-failed-labels-must-be-n-classes>

**Call a training, evaluation, or inference method**

Then once you've trained that model, it's time to evaluate that model, and that's where the test set comes in, which is we want to check if the model is generalizable. The reason we use separate tests is that the model has already been trained on the training set, we want to evaluate it against data it hasn't seen before, just like in the real world when it comes time to deploy that model.

**4.1.1. Evaluation of the LinearClassifier model**

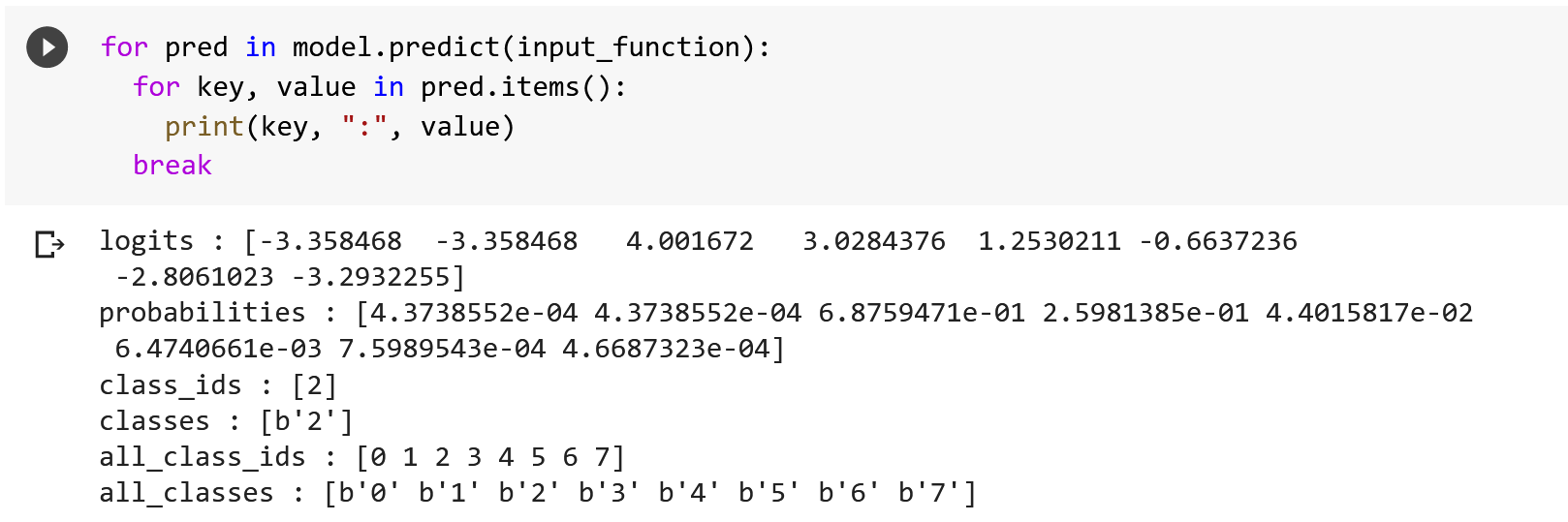
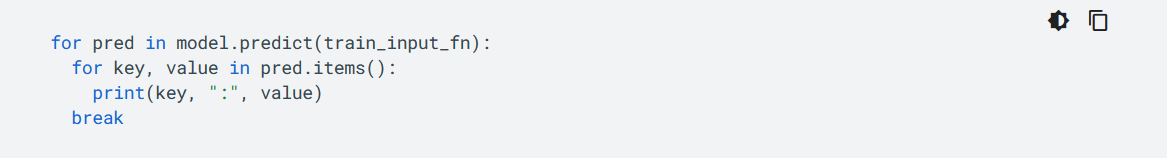
Now we use the TensorFlow website description. For more information please check the references [8].

Let's see what happened in this line of code.

Line 1: **.evaluate()** is for evaluating the already trained model using the validation data and the corresponding labels.

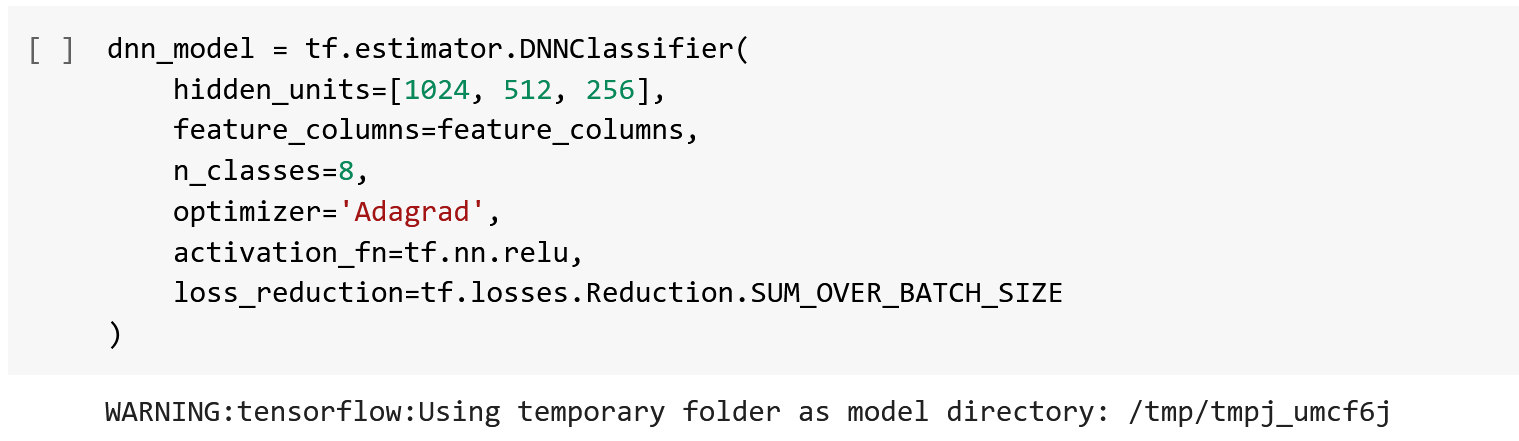
Line 2: As you see to iterate the dictionary with a for loop, we use the **key**, **value**, and **.items()**.

Line 3: This line will print the key and its value of the result of our training.

Now we use the TensorFlow website description. For more information please check the references [8].

And finally, by executing the following cell, we can see our different predictions with their values.

**4.2 DNNClassifier**

Let's see another Classifier model and evaluate this model as well. Just like we had before, we're going to use the description of DNNClassifier model on the TensorFlow website:

Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: As we are talking about estimators in previous slides, we will now define one of the pre-built estimators here.

Line 2: **hidden\_units** It's iterable of the number of hidden units per layer. hidden units basically define how many neurons you want and how many layers, so you provide a list of neurons per layer.

Line 3: As we saw before we're going to say feature columns are equal to the feature columns that we made earlier in the previous Lines and once you run that, you should see an output basically telling you the configuration of the model.

Line 4: **n\_classes** It's the number of label classes.

Line 5: **Adagrad** is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

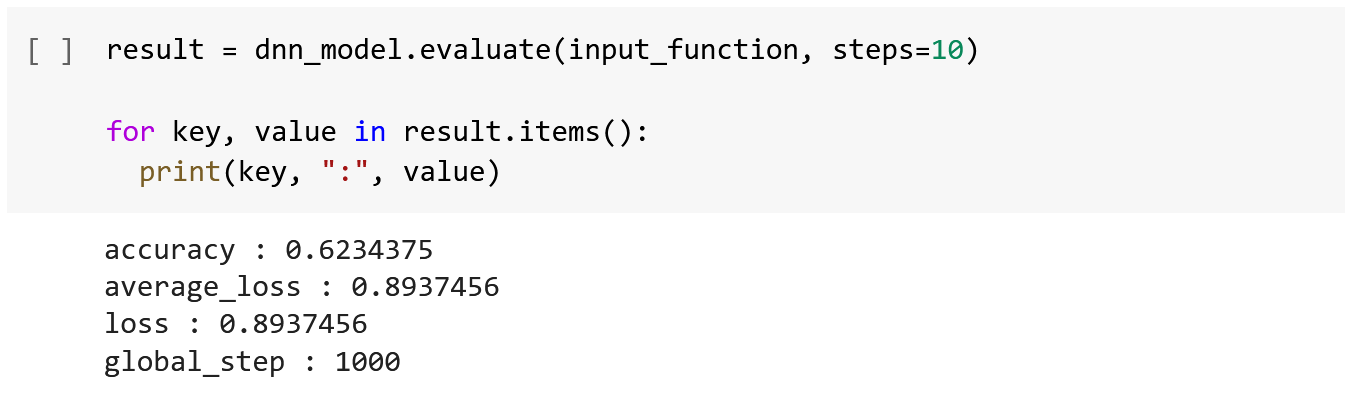
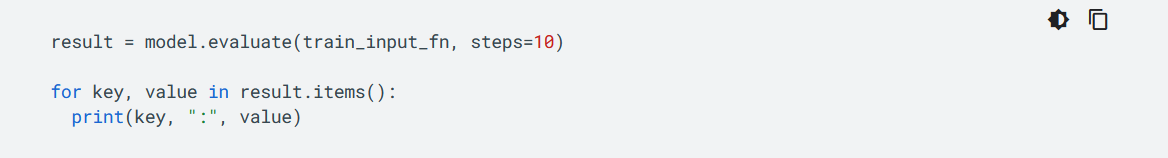
Line 6: **activation\_fn** It's the activation function applied to each layer. If None, will use tf.nn.relu.

Line 7: **loss\_reduction** It describes how to reduce training loss over batch.

For more information please check the references [11]. For a better understanding of hidden units, we recommend checking out this great website at reference [12].



**4.2.1. Evaluation of the DNNClassifier model**

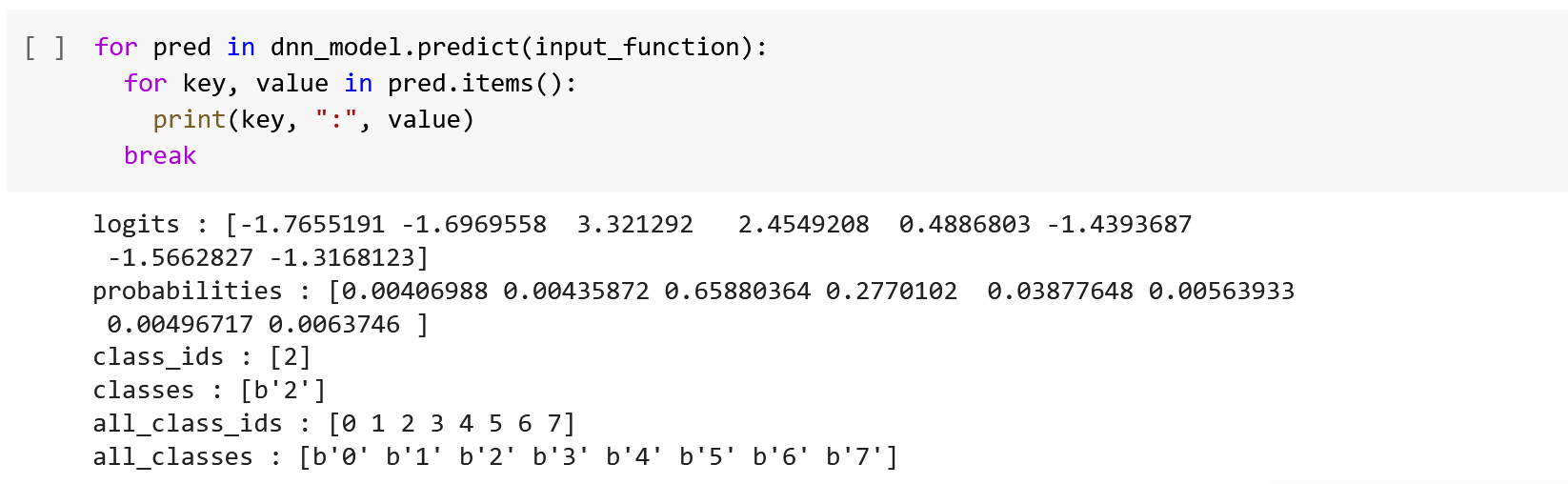
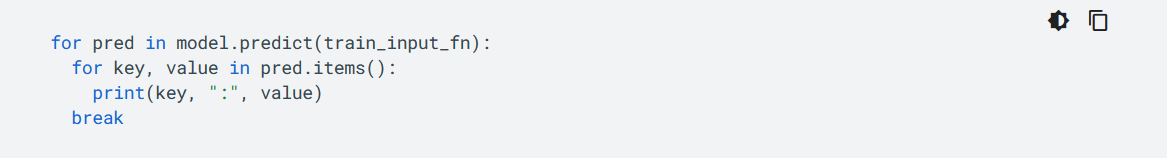
Now we use the TensorFlow website description. For more information please check the references [8].

Let's see what happened in the line code below.

Line 1: **.evaluate()** is for evaluating the already trained model using the validation data and the corresponding labels.

Line 2: As you see to iterate the dictionary with a for loop, we use the **key**, **value**, and **.items()**.

Line 3: This line will print the key and its value of the result of our training.

Now we use the TensorFlow website description. For more information please check the references [8].

And finally, by executing the following cell, we can see our different predictions with their values.

**Step 5. CatBoostClassifier**

CatBoost is a decision tree-based gradient boosting algorithm that is free and open source and is very fast in prediction. The CatBoost algorithm implements symmetric trees that help reduce prediction time and also has a shallower tree depth by default.

To apply Catboost we need to install the package.

to install additional packages in Python, we use this command line:

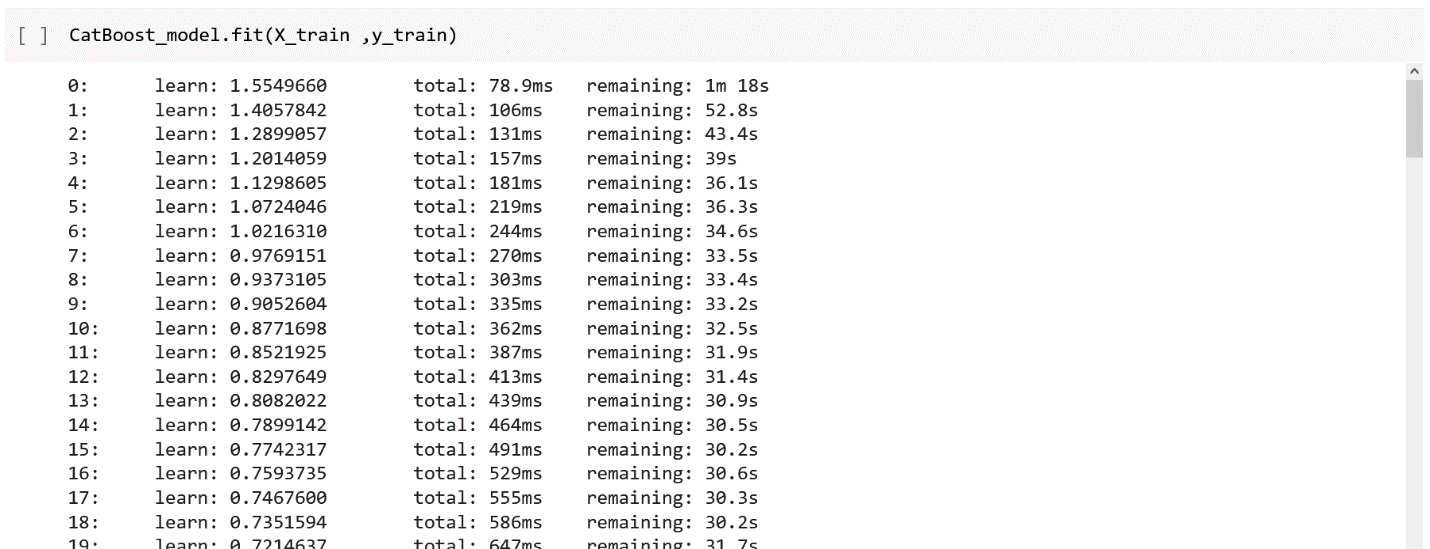
**! pip install (name of the package)**



Let's see what happened in this line of code.

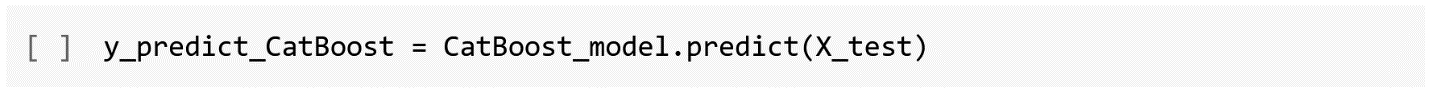
Line 1: **random\_seed** It's the random seed used for training.

Line 2: **learning\_rate** It's the learning rate used for training.

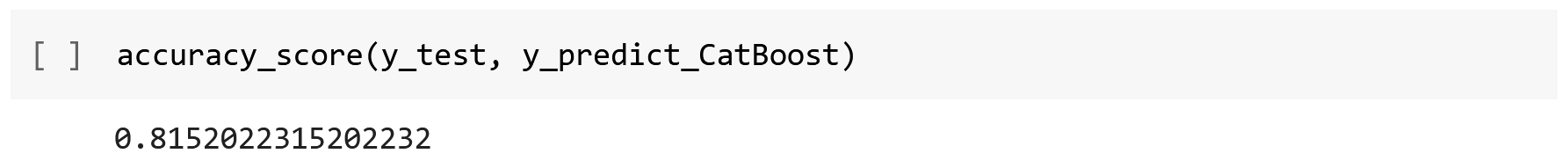
**.fit()** method is for training a model.

**5.1 Accuracy score of the CatBoost model**

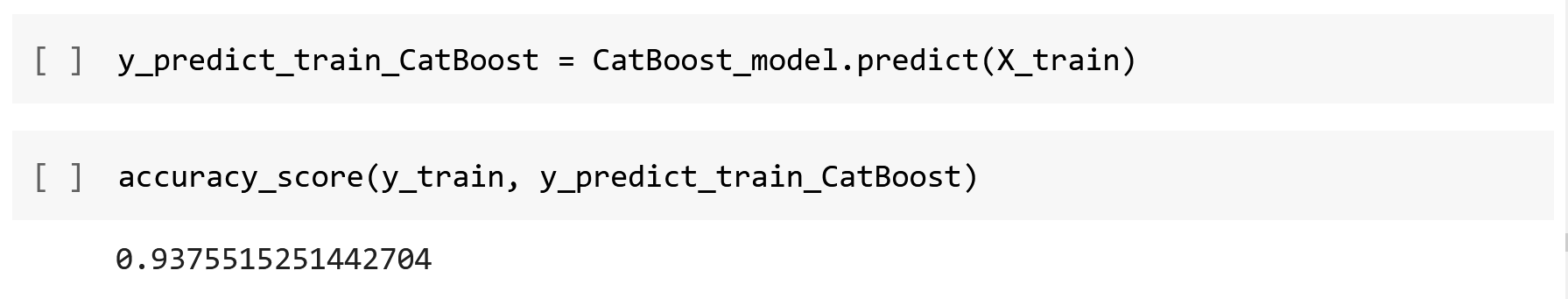
The **accuracy\_score** will give us the accuracy classification score.



**.predict()** method is for applying the model to the given dataset. so with that, we can predict the results



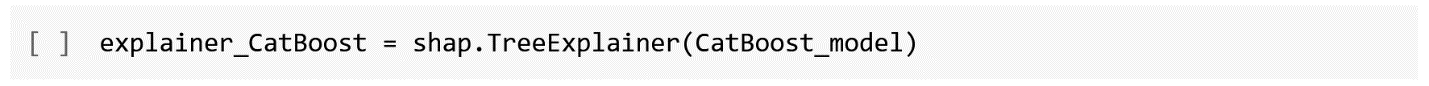
y\_test is the true class labels and y\_predict\_CatBoost are the predicted class labels in the test set.

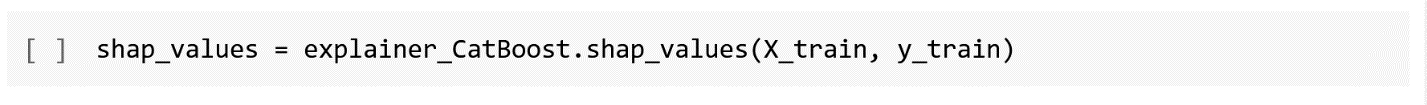
Now, we will compare the train-set and test-set accuracy.

**5.2 SHAP**

In this section, we want to have a better understanding of the Catboost model, so we want to use SHAP. To do this, we will be plotting sample code from the diagrams on the SHAP website.

We install SHAP just like we did for CatBoost in section 5.

The examples on the website are a great resource and very helpful for learning. You can see other examples in [13].

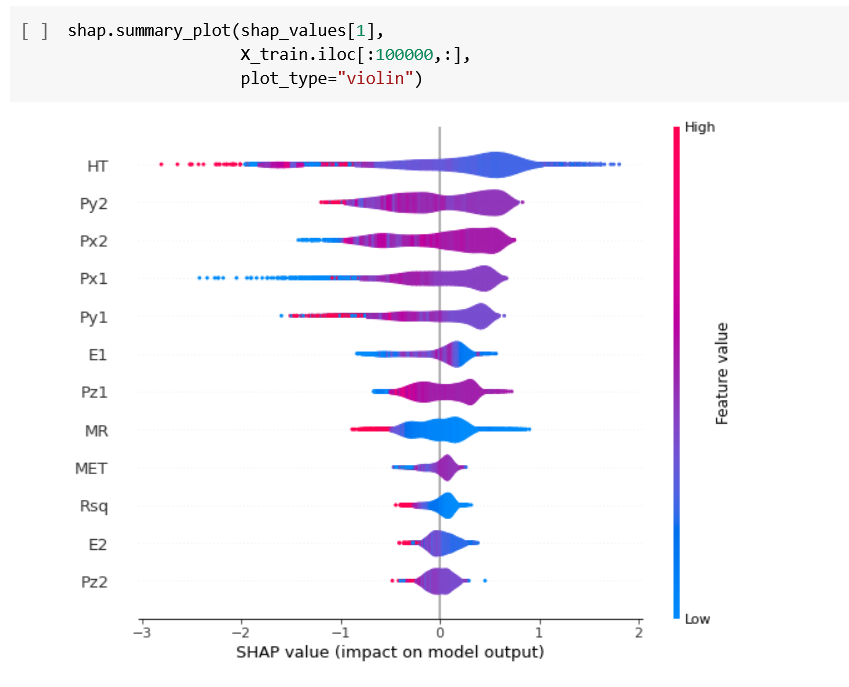
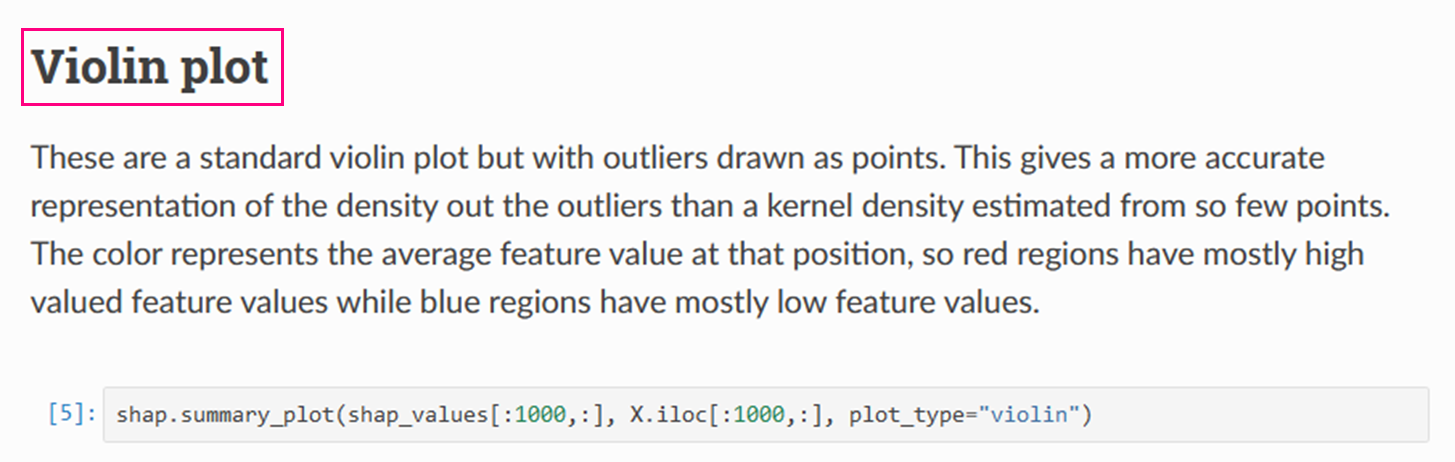
**.TreeExplainer()** method uses Tree SHAP algorithms to explain the output of ensemble tree models.

**.shap\_values()** method estimates SHAP values for a set of samples.

**5.2.1 A better understanding of the CatBoost model using SHAP plots**

In this section, we use two types of **shap.summary\_plot**. for that, we use violin and bar plot.

**5.2.1.1 Violin plot**

Now we use the description of the **Violin plot** on the SHAP website. For more information please check the references [14].

Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: **shap\_values** For single-output descriptions, this is a matrix of SHAP values. For multi-output descriptions, this is a list of such matrices of SHAP values.

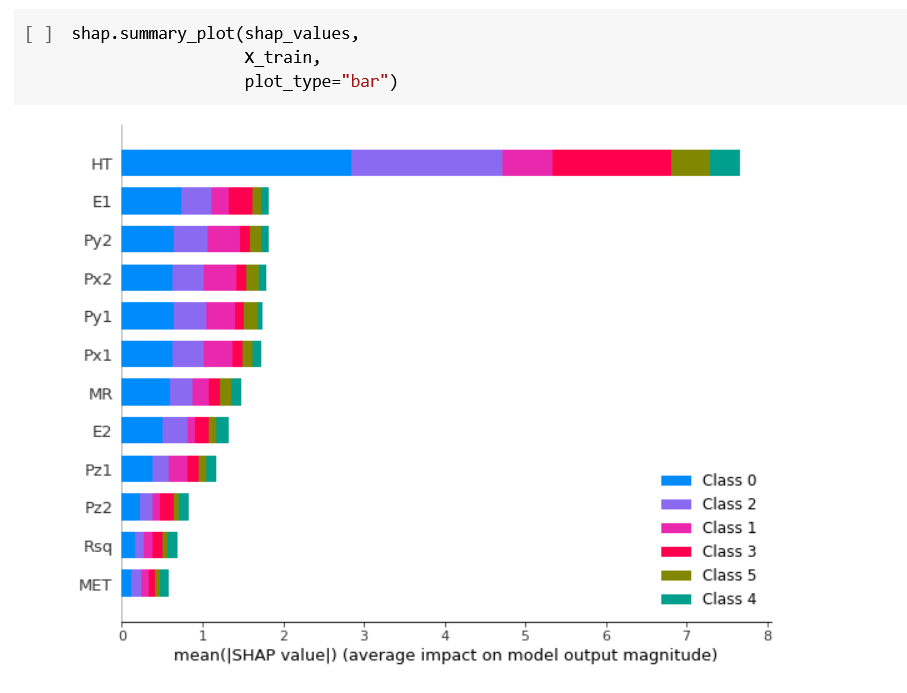
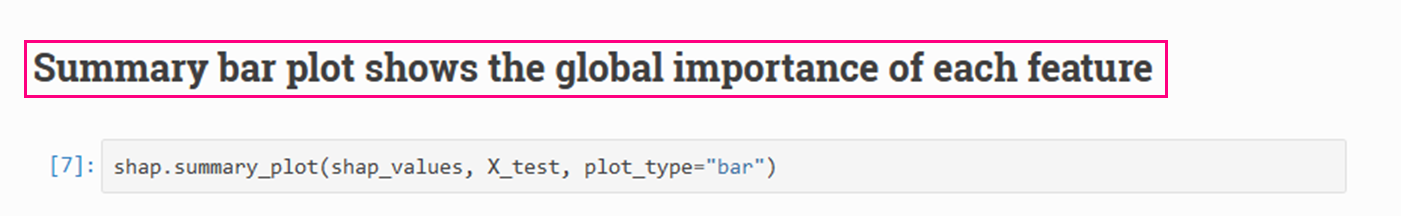
Line 2: **.iloc[]** This means we need to pass an integer index to the method to select a particular row/column.

Line 3: **plot\_type** We name the plot we want to see.

As you noticed we have changed the shap\_values because otherwise, we get this error:

**TypeError: list indices must be integers or slices, not tuple**

**5.2.1.2 SHAP Summary Plot**

Now we use the description of the **bar plot** on the SHAP website. For more information please check the references [15].

This plot shows the global importance of each feature we have. Let's see what happened in the following line of code in our Colab-Notebook:

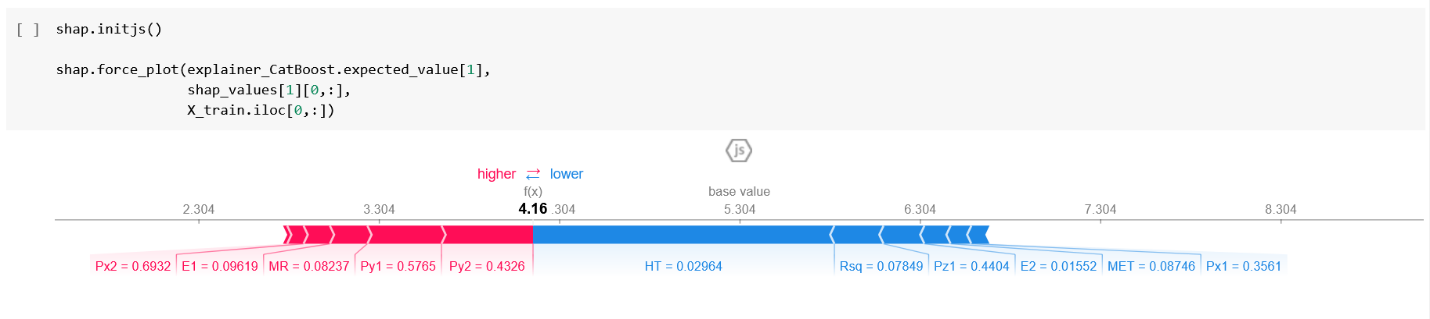
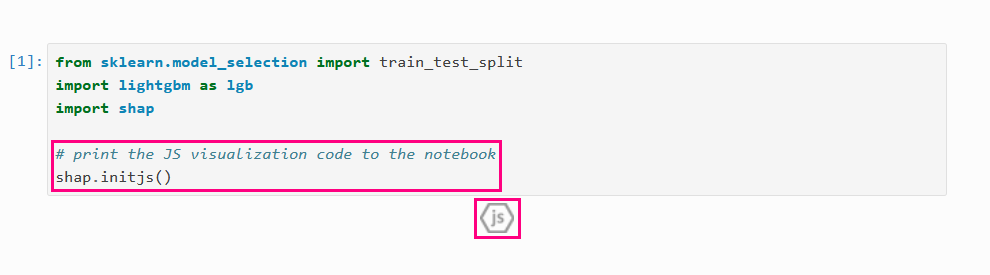
Line 1: **shap\_values** For single-output descriptions, this is a matrix of SHAP values. For multi-output descriptions, this is a list of such matrices of SHAP values.

Line 2: **.iloc[]** This means we need to pass an integer index to the method to select a particular row/column.

Line 3: **plot\_type** We name the plot we want to see.

In this section, we use two types of **shap.force\_plot**. and with these types of plots, we can explain predictions.

**5.2.1.3 Visualize a single prediction**

Now we use the SHAP website description. For more information please check the references [16]. For this type of plot, we must first print the JS visualization code to the notebook. JS (JavaScript data visualization) is one of the most popular data visualization libraries used by developers around the world to manipulate documents based on data [20].

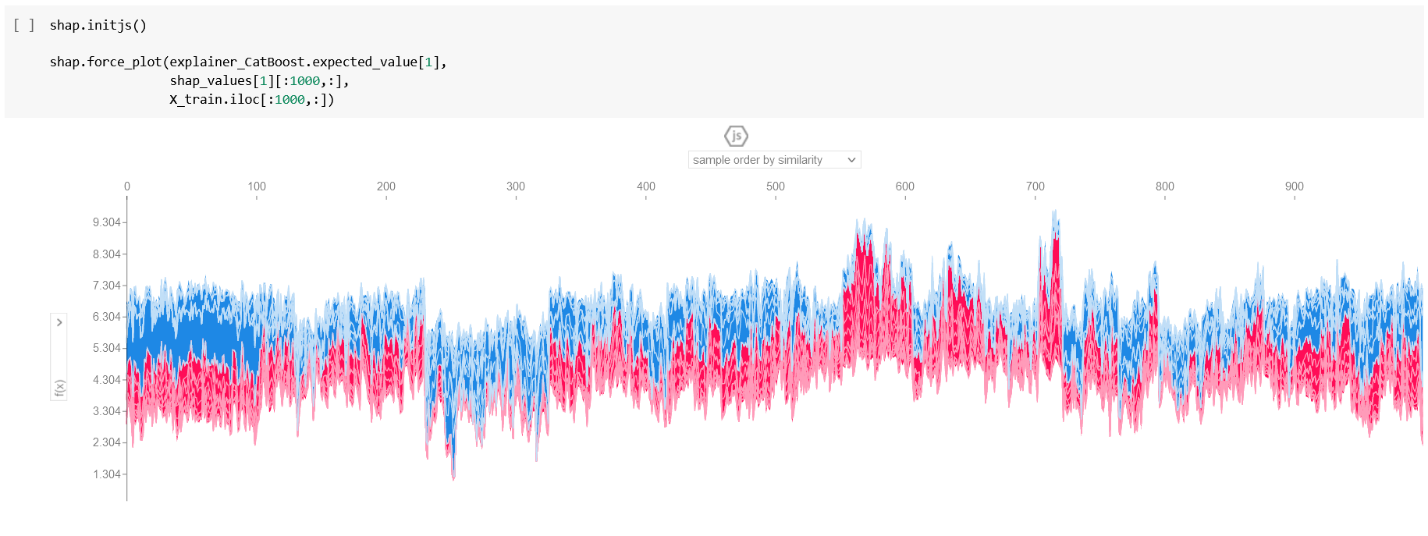
Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: **.expected\_value[]** all SHAP values are relative to the model's expected value.

Line 2: **shap\_values** For single-output descriptions, this is a matrix of SHAP values. For multi-output descriptions, this is a list of such matrices of SHAP values.

Line 3: **.iloc[]** This means we need to pass an integer index to the method to select a particular row/column.

**5.2.1.4 Visualize many predictions**

Now we use the SHAP website description. For more information please check the references [16].

Let's see what happened in the following line of code in our Colab-Notebook:

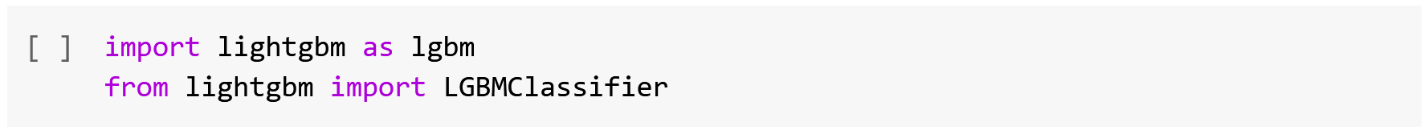
Line 1: **.expected\_value[]** all SHAP values are relative to the model's expected value.

Line 2: **shap\_values** For single-output descriptions, this is a matrix of SHAP values. For multi-output descriptions, this is a list of such matrices of SHAP values.

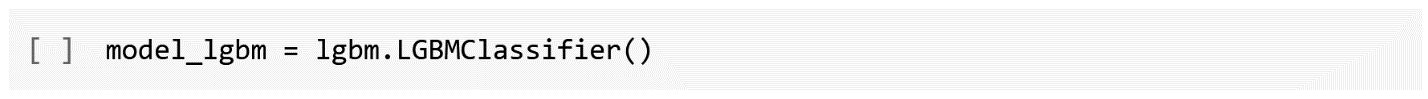
Line 3: **.iloc[]** This means we need to pass an integer index to the method to select a particular row/column.

Remember to also print the **JS** visualization code in the notebook for this.

**Step 6. LightGBM**

LightGBM algorithm uses only tree-based algorithms. Since LightGBM is designed based on decision tree algorithms, it extracts from the leaf of the tree with the best fit, while other algorithms operate through the depth of the tree or its surface instead of the leaf. Therefore, when LightGBM is grown on a leaf, this leaf algorithm can achieve more loss reduction than surface algorithms, and hence its results are much better In terms of accuracy. This performance is rarely achieved by other existing optimization algorithms. In addition, this algorithm is amazingly fast, and the term Light in its title refers to this point.

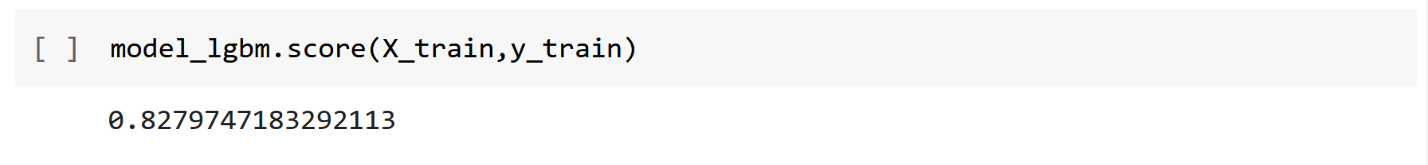
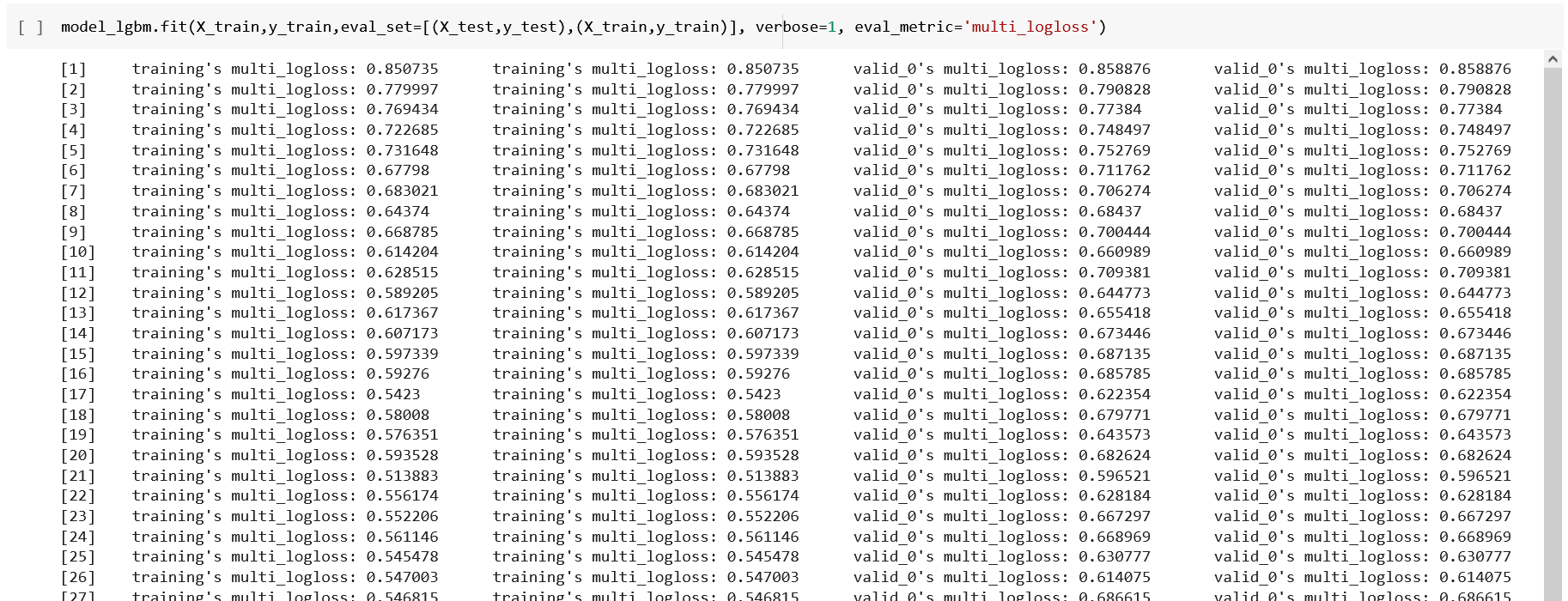
**6.1 LGBMClassifier**

**.LGBMClassifier()** is the structure of a gradient amplification model.

Once you perform the **LGBMClassifier** modelit’s time to actually train (or fit) your model on the training data. There is a very good guide on the parameters of the .fit command on the **LightGBM** website. In the code, as you can see I use these two parameters as you can see below. For more information please check the references [17].

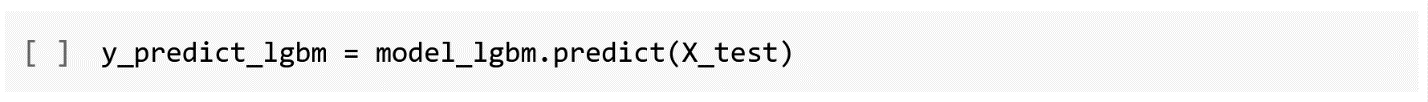






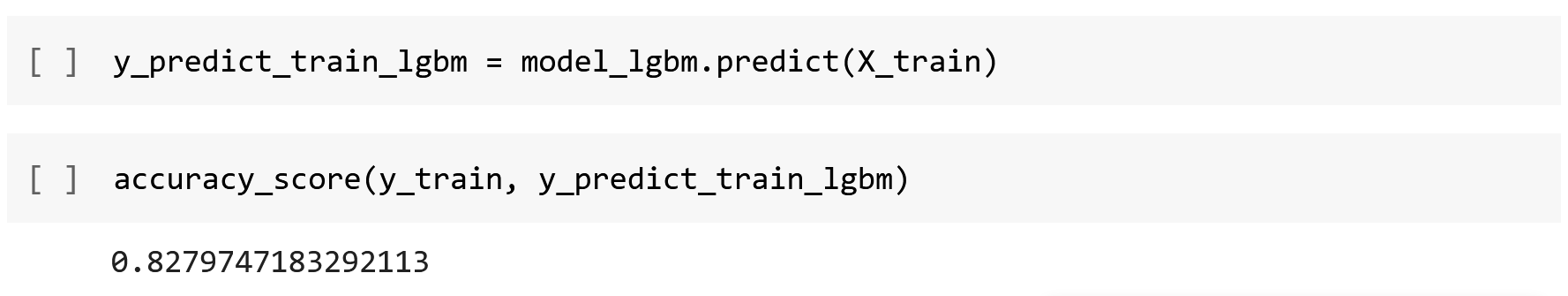
**6.2 Accuracy score of LightGBM model**

The codes in section 6.2 are similar to the previous codes in section 5.1. The **accuracy\_score** will give us the accuracy classification score.

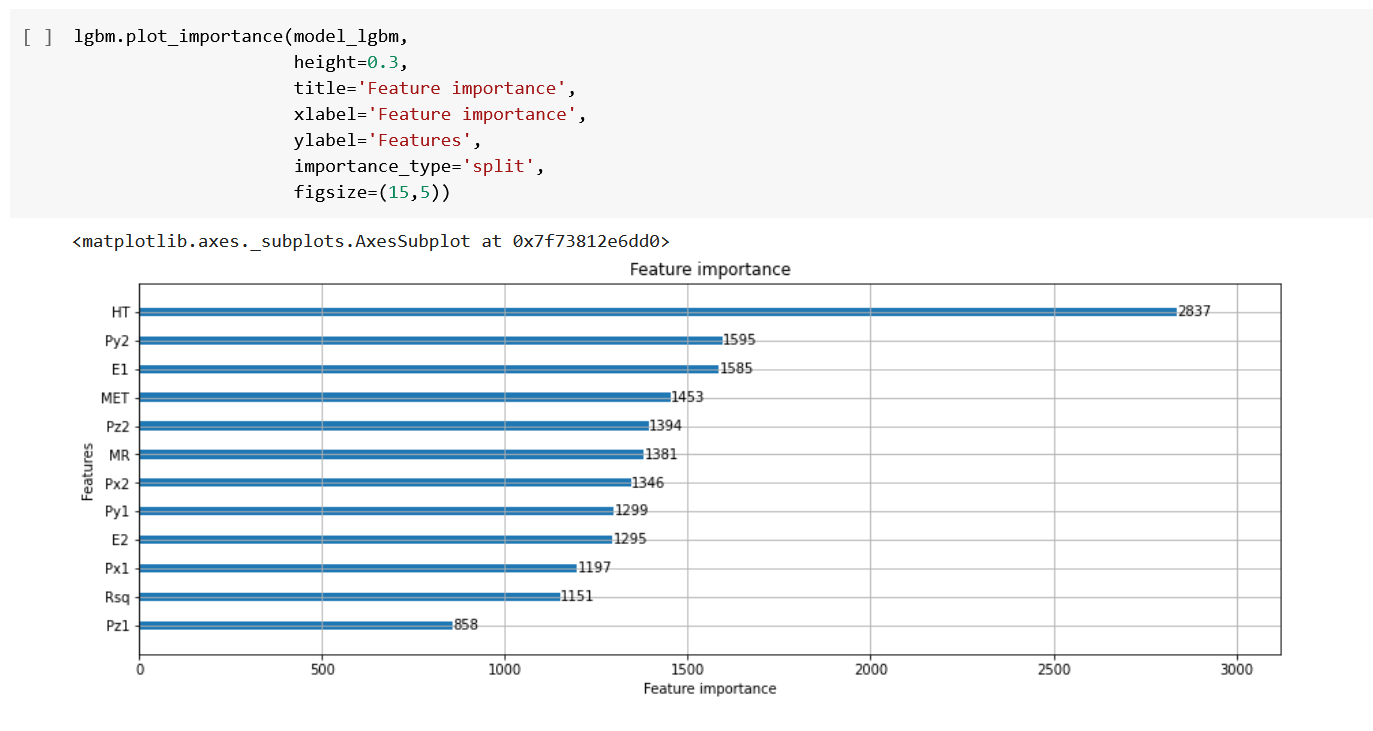


**.predict()** method is for applying the model to the given dataset. so with that, we can predict the results.

y\_test is the true class labels and y\_predict\_lgbm are the predicted class labels in the test set.

Now, we will compare the train-set and test-set accuracy.

**6.3 A better understanding of the LightGBM model using plots**

**6.3.1 plot\_importance**

Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: we add our model

Line 2: **height** we can adjust the height of the bar we want

Line 3: **title** with this, we create a title for our plot

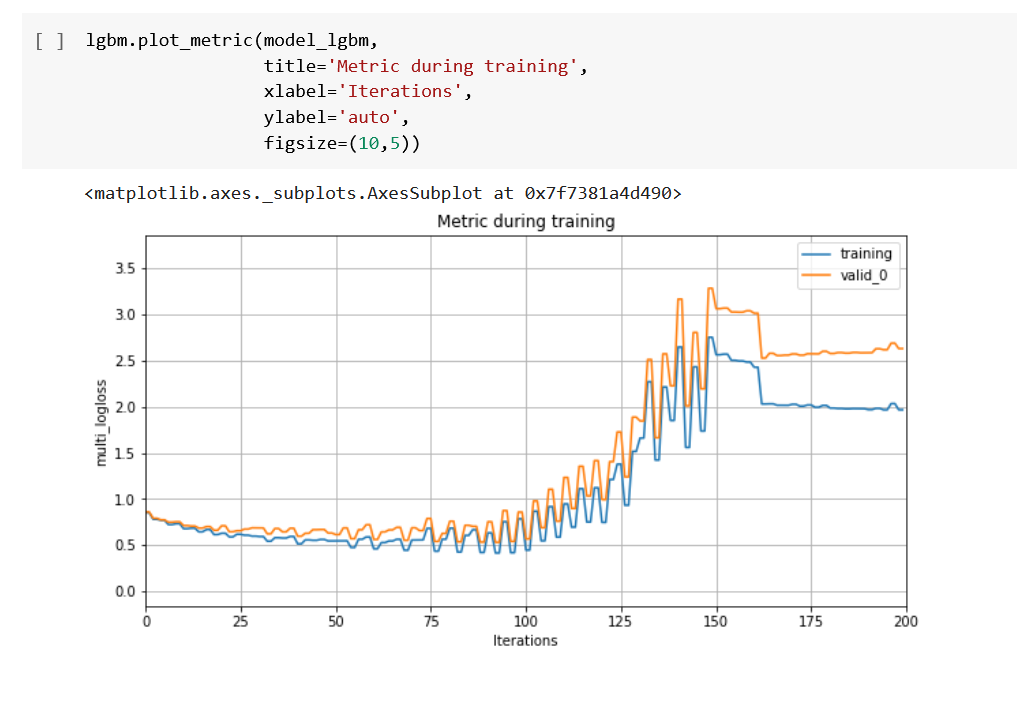
Line 4: with **xlabel** we set the label for the x-axis.

Line 5: with **ylabel** we set the label for the x-axis.

Line 6: **importance\_type** this is about how significance is calculated

Line 7: we set the desired size of my figure with **figsize**.

For more information please check the references [18].

**6.3.2 plot\_metric**

This one plots a metric during the training. Let's see what happened in the following line of code in our Colab-Notebook:

Line 1: we add our model

Line 2: **title** with this, we create a title for our plot

Line 3: with **xlabel** we set the label for the x-axis.

Line 4: with **ylabel** we set the label for the x-axis.

Line 5: we set the desired size of my figure with **figsize**.

For more information please check the references [19].

**References**

[1]<https://stanford.edu/group/stanford_atlas/4Particle%20Collision%20and%20Detection#:~:text=In%20most%20proton%20collisions%20the,particles%20for%20us%20to%20find>.

[2] <https://www.kaggle.com/datasets/fedesoriano/multijet-primary-dataset>

[3] <https://www.analyticsvidhya.com/blog/2021/06/how-to-clean-data-in-python-for-machine-learning/>

[4] <https://www.tutorialspoint.com/python_pandas/python_pandas_missing_data.htm>

[5] <https://towardsdatascience.com/letter-value-plot-the-easy-to-understand-boxplot-for-large-datasets-12d6c1279c97>

[6] <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>

[7] <https://medium.com/ml-book/demonstration-of-tensorflow-feature-columns-tf-feature-column-3bfcca4ca5c4>

[8] <https://www.tensorflow.org/guide/estimator>

[9] <https://www.tensorflow.org/api_docs/python/tf/compat/v1/estimator/inputs/pandas_input_fn>

[10] <https://www.tensorflow.org/api_docs/python/tf/estimator/LinearClassifier>

[11] <https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier>

[12][https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.25713&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false](https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.25713&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&co)

[13] <https://shap-lrjball.readthedocs.io/en/latest/examples.html>

[14] <https://shap-lrjball.readthedocs.io/en/latest/example_notebooks/tree_explainer/Scatter%20Density%20vs.%20Violin%20Plot%20Comparison.html>

[15] <https://shap-lrjball.readthedocs.io/en/latest/example_notebooks/tree_explainer/Census%20income%20classification%20with%20XGBoost.html>

[16] <https://shap-lrjball.readthedocs.io/en/latest/example_notebooks/tree_explainer/Census%20income%20classification%20with%20LightGBM.html>

[17] <https://lightgbm.readthedocs.io/en/latest/Parameters.html>

[18] <https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.plot_importance.html>

[19] <https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.plot_metric.html>

[20] <https://www.softwaretestinghelp.com/best-javascript-visualization-libraries/>