EARTHQUAKE PREDICTION MODEL USING PYTHON

TEAM MEMBER

961321104016 – M.NANTHINI

Phase-3: PREPROCESSING THE DATASET

Project Title: Earthquake prediction

OBJECTIVE:

The Objective of the phase 3 is loading and preprocessing the dataset.

Model Development Part 1

1.Data collection

Gather historical earthquake data, including location, magnitude, depth, and time of occurrence. Additionally, collect various geological and seismological data, such as fault line information, tectonic plate movements, and soil properties.

https://www.kaggle.com/datasets/alessandrolobello/theultimate-earthquake-dataset-from-1990-2023

This is the dataset link for this project from kaggle website

2.Data pre-processing

Clean and pre-process the data to handle missing values, outliers, and inconsistencies. You may need to aggregate data at different spatial and temporal scales.

1. Acquire the dataset

Acquiring the dataset is the first step in data pre-processing in machine learning. To build and develop Machine Learning models, you must first acquire the relevant dataset. This dataset will be comprised of data gathered from multiple and disparate sources which are then combined in a proper format to form a dataset. Dataset formats differ according touse cases. For instance, a business dataset will be entirely different from a medical dataset. While a business dataset will contain relevant industry and business data, a medicaldataset will include healthcare-related data.

2. Import all the crucial libraries

Since Python is the most extensively used and also the most preferred library by Data Scientists around the world, we'll show you how to import Python libraries for data preprocessing in Machine Learning. Read more about Python libraries for Data Science here. The predefined Python libraries can perform specific data pre-processing jobs. Importing all the crucial libraries is the second step in data pre-processing in machine learning. The three core Python libraries used for this data pre-processing in Machine Learning are:

Numpy – Numpy is the fundamental package for scientific calculation in Python. Hence, it is used for inserting any type of mathematical operation in the code. Using Numpy, you can also add large multidimensional arrays and matrices in your code.

Pandas – Pandas is an excellent open-source Python library for data manipulation and analysis. It is extensively used for importing and managing the datasets. It packs in highperformance, easy-to-use data structures and data analysis tools for Python.

Matplotlib – Matplotlib is a Python 2D plotting library that is used to plot any type of charts in Python. It can deliver publication-quality figures in numerous hard copy formats and interactive environments across platforms (IPython shells, Jupyter notebook, web application servers, etc.).

3. Import the dataset

In this step, you need to import the dataset/s that you have gathered for the ML project at hand. Importing the dataset is one of the important steps in data pre-processing in machine learning. However, before you can import the dataset/s, you must set the current directory as the working directory. You can set the working directory in Spyder IDE in three simple steps:

Save your Python file in the directory containing the dataset.

Go to File Explorer option in Spyder IDE and choose the required directory.

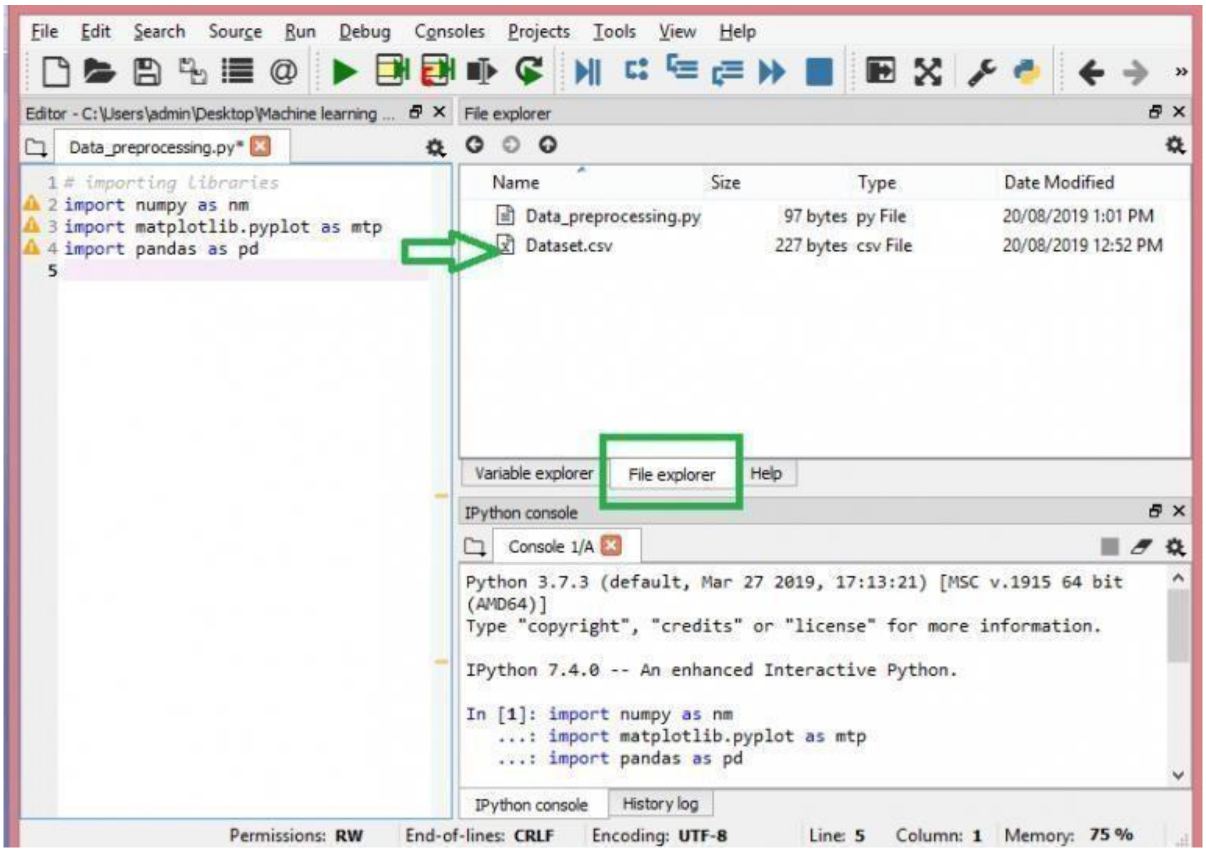
Now, click on the F5 button or Run option to execute the file.

Once you've set the working directory containing the relevant dataset, you can import the dataset using the "read_csv()" function of the Pandas library. This function can read a CSV file (either locally or through a URL) and also perform various operations on it. The read_csv() is written as:

dataset= pd.read_csv('Dataset.csv')

In this line of code, "data_set" denotes the name of the variable wherein you stored the dataset. The function contains the name of the dataset as well. Once you execute this code, the dataset will be successfully imported.

During the dataset importing process, there's another essential thing you must do – extracting dependent and independent variables. For every Machine Learning model, it is necessary to separate the independent variables (matrix of features) and dependent variables in a dataset.



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Name	Size	Type	Date Modified
Data_preprocessing.py	97 bytes	py File	20/08/2019 1:01 PM
Dataset.csv	227 bytes	csv File	20/08/2019 12:52 PM

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: import numpy as nm
   ...: import matplotlib.pyplot as mtp
   ...: import pandas as pd
```

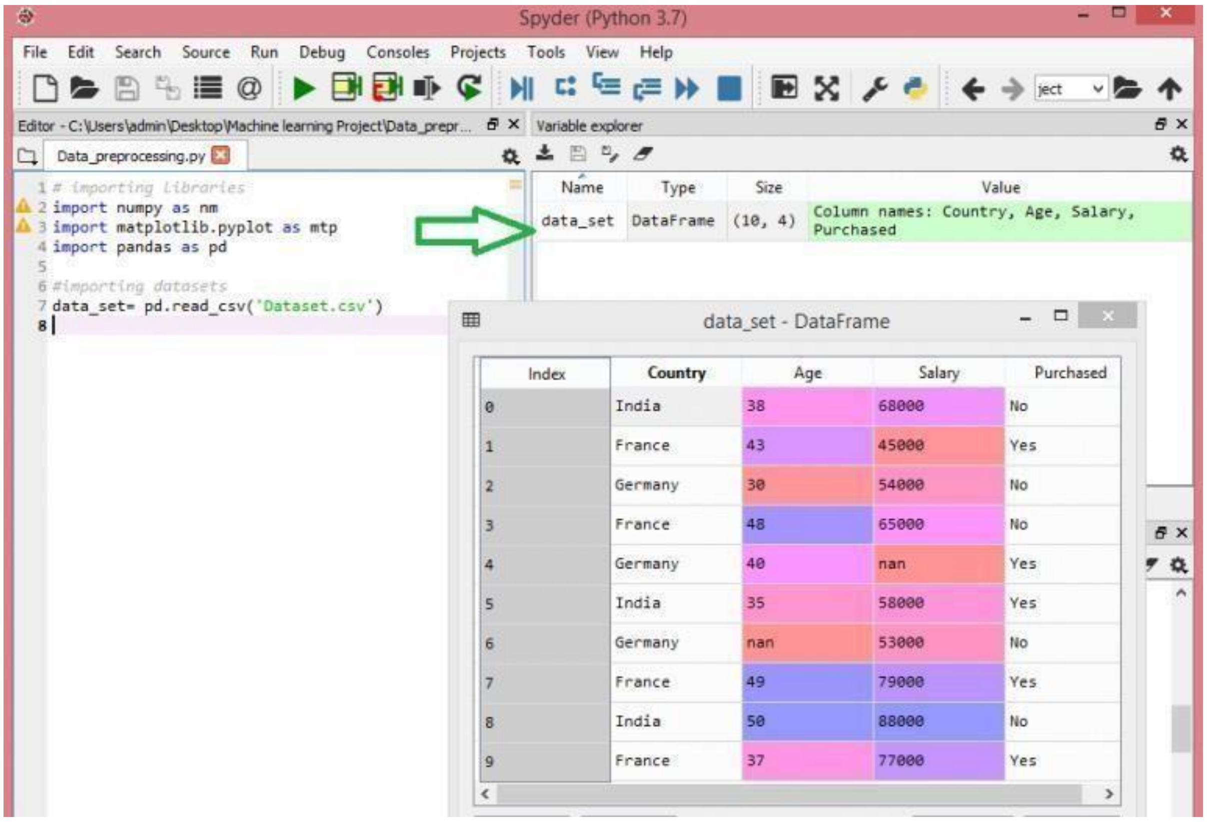
Permissions: RW   End-of-lines: CRLF   Encoding: UTF-8   Line: 5   Column: 1   Memory: 75 %

To extract the independent variables, you can use “iloc[ ]” function of the Pandas library. This function can extract selected rows and columns from the dataset.

x= data_set.iloc[:,:-1].values

In the line of code above, the first colon(:) considers all the rows and the second colon(:) considers all the columns. The code contains “:-1” since you have to leave out the last column containing the dependent variable. By executing this code, you will obtain the matrix of features, like this –

```
[['India' 38.0 68000.0]

 ['France' 43.0 45000.0]

 ['Germany' 30.0 54000.0]

 ['France' 48.0 65000.0]

 ['Germany' 40.0 nan]

 ['India' 35.0 58000.0]

 ['Germany' nan 53000.0]

 ['France' 49.0 79000.0]

 ['India' 50.0 88000.0]

 ['France' 37.0 77000.0]]
```


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\admin\Desktop\Machine learning Project\Data_prepr...
Data_preprocessing.py
Variable explorer
data_set - DataFrame

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Dataset.csv')
```

Name	Type	Size	Value
data_set	DataFrame	(10, 4)	Column names: Country, Age, Salary, Purchased

Index	Country	Age	Salary	Purchased
0	India	38	68000	No
1	France	43	45000	Yes
2	Germany	30	54000	No
3	France	48	65000	No
4	Germany	40	nan	Yes
5	India	35	58000	Yes
6	Germany	nan	53000	No
7	France	49	79000	Yes
8	India	50	88000	No
9	France	37	77000	Yes

You can use the “iloc[ ]” function to extract the dependent variable as well. Here’s how you write it:

Y= data_set.iloc[:,3].values

This line of code considers all the rows with the last column only. By executing the above code, you will get the array of dependent variables, like so –

Array([‘No’, ‘Yes’, ‘No’, ‘No’, ‘Yes’, ‘Yes’, ‘No’, ‘Yes’, ‘No’, ‘Yes’], Dtype=object)

3.Feature Engineering

Feature Engineering helps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

```
Splitted = df[‘Origin Time’].str.split(‘ ‘, n=1,
Expand=True)
```

```
Df[‘Date’] = splitted[0]
```

```
Df[‘Time’] = splitted[1].str[:-4]
```

```
Df.drop(‘Origin Time’,Axis=1 Inplace=True)
```

Df.head() Output:

Now, let’s divide the date column into the day, month, and year columns respectively.

```
Df[‘day’] = splitted[2].astype(‘int’)
```

```
Df[‘month’] = splitted[1].astype(‘int’)
```

```
Df[‘year’] = splitted[0].astype(‘int’)
```

```
Df.drop(‘Date’, axis=1, Inplace=True)
```

```
Df.head()
```

Exploring data analysis

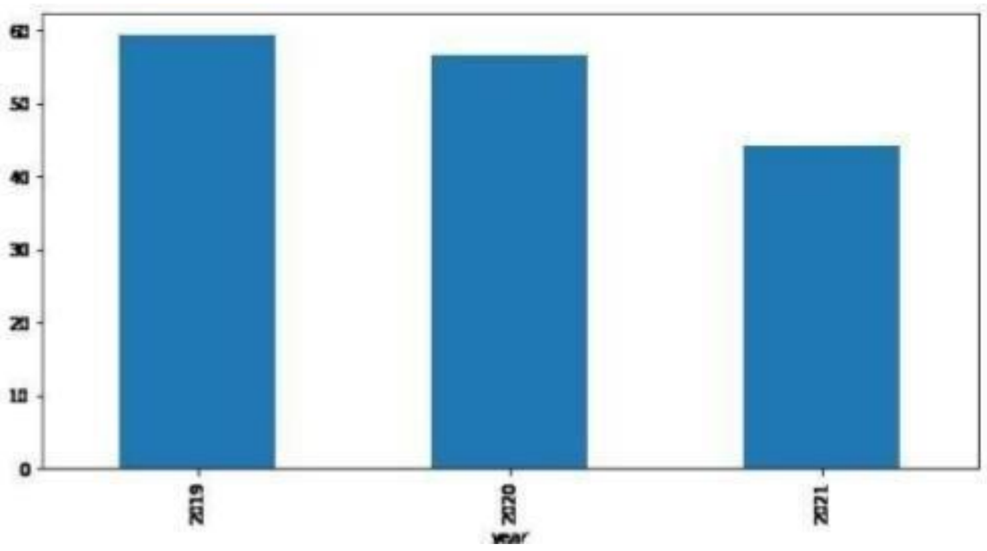
EDA is an approach to analyzing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations

Plt.f Plt.

plt.figure(figsize=(10, 5)) x =

df.groupby('year').mean()['Depth']

x.plot.bar() plt.show()



4. Data splitting

Divide the dataset into training, validation, and test sets. Ensure that the temporal aspect is preserved, as earthquakes can exhibit temporal dependencies.

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are Timestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

X = final_data[['Timestamp', 'Latitude', 'Longitude']]

Y = final_data[['Magnitude', 'Depth']]

From sklearn.cross_validation import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

Print(X_train.shape, X_test.shape, y_train.shape, X_test.shape)

(18727, 3) (4682, 3) (18727, 2) (4682, 3)

/opt/conda/lib/python3.6/site-
packages/sklearn/cross_validation.py:41: DeprecationWarning: This
module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and
functions are moved. Also note that the interface of the new CV
iterators are different from that of this module. This module will be
removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Here, we used the RandomForestRegressor model to predict the outputs, we see the strange prediction from this with score above 80% which can be assumed to be best fit but not due to its predicted values.

```
From sklearn.ensemble import RandomForestRegressor


Reg = RandomForestRegressor(random_state=42)

Reg.fit(X_train, y_train)

Reg.predict(X_test)

/opt/conda/lib/python3.6/sitepackages/sklearn/ensemble/weight_b
oosting.py:29:
DeprecationWarning: numpy.core.umath_tests is an internal NumPy
module and should not be imported. It will be removed in a future
NumPy release.
  From numpy.core.umath_tests import inner1d
```

```
Array([[  5.96,  50.97],
       [ 5.88,  37.8 ],
[  5.97, 37.6 ],
      ...,
      [ 6.42,  19.9 ],
      [  5.73, 591.55],
      [      5.68,      33.61]])
Reg.score(X_test, y_test)
Out:
0.8614799631765803
From sklearn.model_selection import GridSearchCV

Parameters = {'n_estimators':[10, 20, 50, 100, 200, 500]}

Grid_obj = GridSearchCV(reg, parameters)
Grid_fit = grid_obj.fit(X_train, y_train)
Best_fit = grid_fit.best_estimator_
Best_fit.predict(X_test)
Array([[  5.8888 ,  43.532  ],
      [ 5.8232 ,   31.71656],
      [ 6.0034 ,   39.3312 ],
      ...,
      [ 6.3066 ,   23.9292 ],
      [ 5.9138 , 592.151  ],
   [   5.7866  ,    38.9384  ]])
```

Best_fit.score(X_test, y_test)

Out:

0.8749008584467053

5.Model selection

From the kaggle dataset we have to implement neural network modal

In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.

In [16]:

```
from keras.models import Sequential from keras.layers
import Dense
 def create_model(neurons, activation, optimizer, loss):     model = Sequential()
    model.add(Dense(neurons, activation=activation, input_shape=(3,)))     model.add(Dense(neurons,
activation=activation))     model.add(Dense(2, activation='softmax'))

    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

    return model
```

```
Using TensorFlow backend.
```

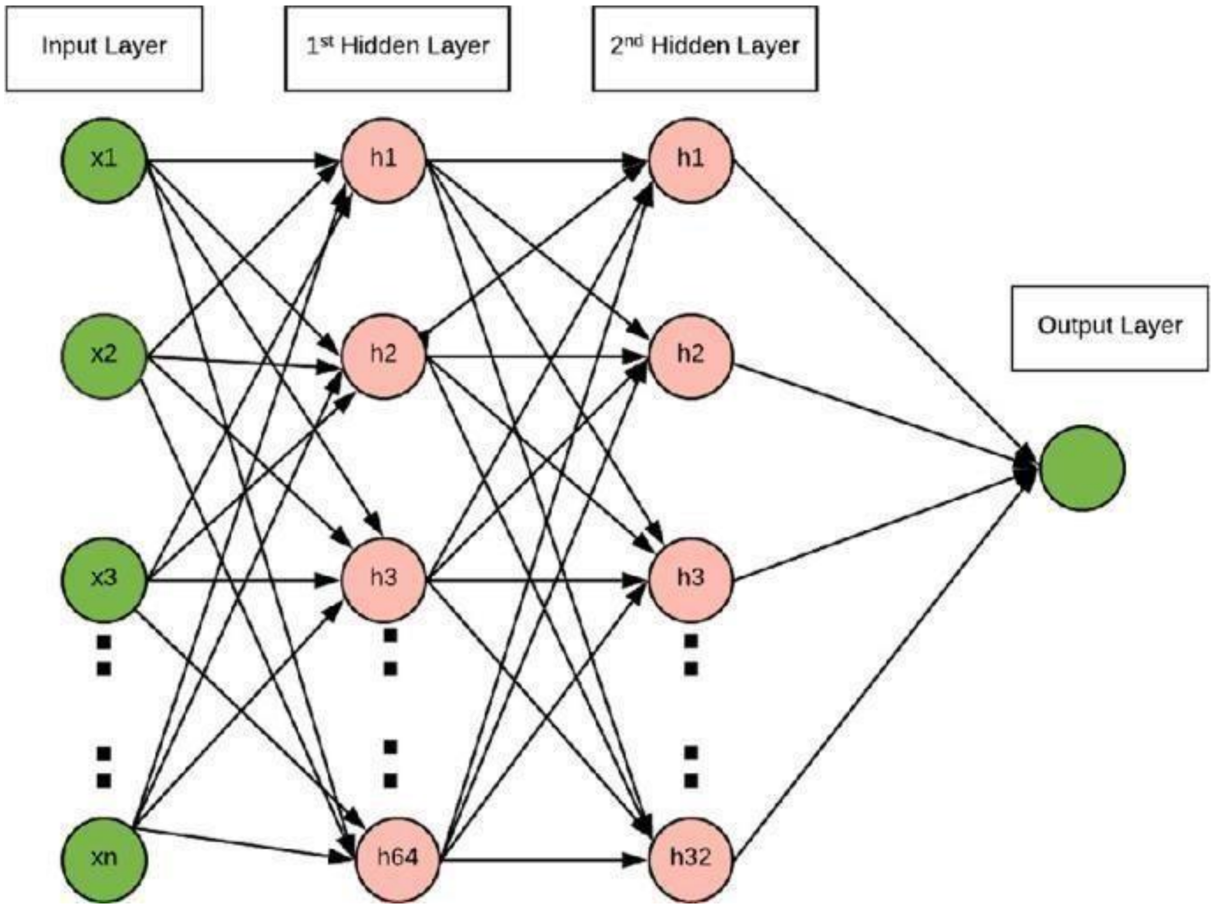
In this, we define the hyperparameters with two or more options to find the best fit.

In [17]:

```
from keras.wrappers.scikit_learn import KerasClassifier

model = KerasClassifier(build_fn=create_model, verbose=0)
# neurons = [16, 64, 128, 256] neurons = [16]
# batch_size = [10, 20, 50, 100] batch_size =
[10] epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'expon ential']
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', '
Nadam']
optimizer = ['SGD', 'Adadelta'] loss =
['squared_hinge']

param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs,
activation=activation, optimizer=optimizer, loss=loss)
```

Here, we find the best fit of the above model and get the mean test score and standard deviation of the best fit model.

Neural Network:



6. Model Training

Train the selected model on the training dataset. Experiment with hyperparameter tuning to optimize model performance.

Step 1: Begin with existing data

Machine learning requires us to have existing data—not the data our application will use when we run it, but data to learn from. You need a lot of real data, in fact, the more the better. The more examples you provide, the better the computer should be able to learn. So just collect every scrap of data you have and dump it and voila! Right?

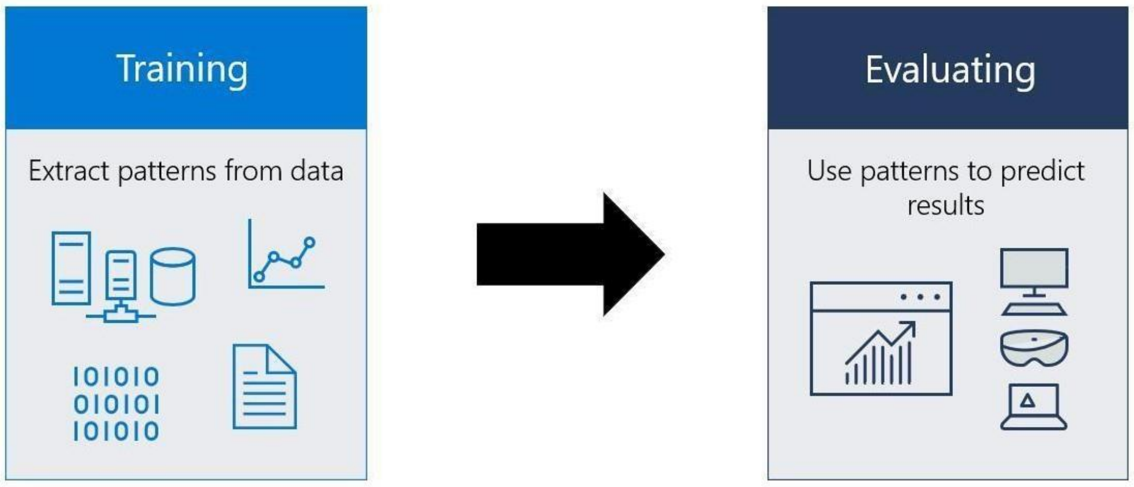
Wrong. In order to train the computer to understand what we want and what we don't want, you need to prepare, clean and label your data. Get rid of garbage entries, missing pieces of information, anything that's ambiguous or confusing. Filter your dataset down to only the information you're interested in right now. Without high quality data, machine learning does not work. So take your time and pay attention to detail.

Step 2: Analyze data to identify patterns

Unlike conventional software development where humans are responsible for interpreting large data sets, with machine learning, you apply a machine learning algorithm to the data. But don't think you're off the hook. Choosing the right algorithm, applying it, configuring it and testing it is where the human element comes back in.

There are several platforms to choose from both commercial and open source. Explore solutions from Microsoft, Google, Amazon, IBM or open source frameworks like TensorFlow, Torch and Caffe. They each have their own strengths and downsides, and each will interpret the same dataset a different way. Some are faster to train. Some are more configurable. Some allow for more visibility into the decision process. In order to make the right choice, you need to experiment with a few algorithms and test until you find the one that gives you the results most aligned to what you're trying to achieve with your data.

When it's all said and done, and you've successfully applied a machine learning algorithm to analyze your data and learn from it, you have a trained model.



Step 3: Make predictions

There is so much you can do with your newly trained model. You could import it into a software application you're building, deploy it into a web back end or upload and host it into a cloud service. Your trained model is now ready to take in new data and feed you predictions, aka results.

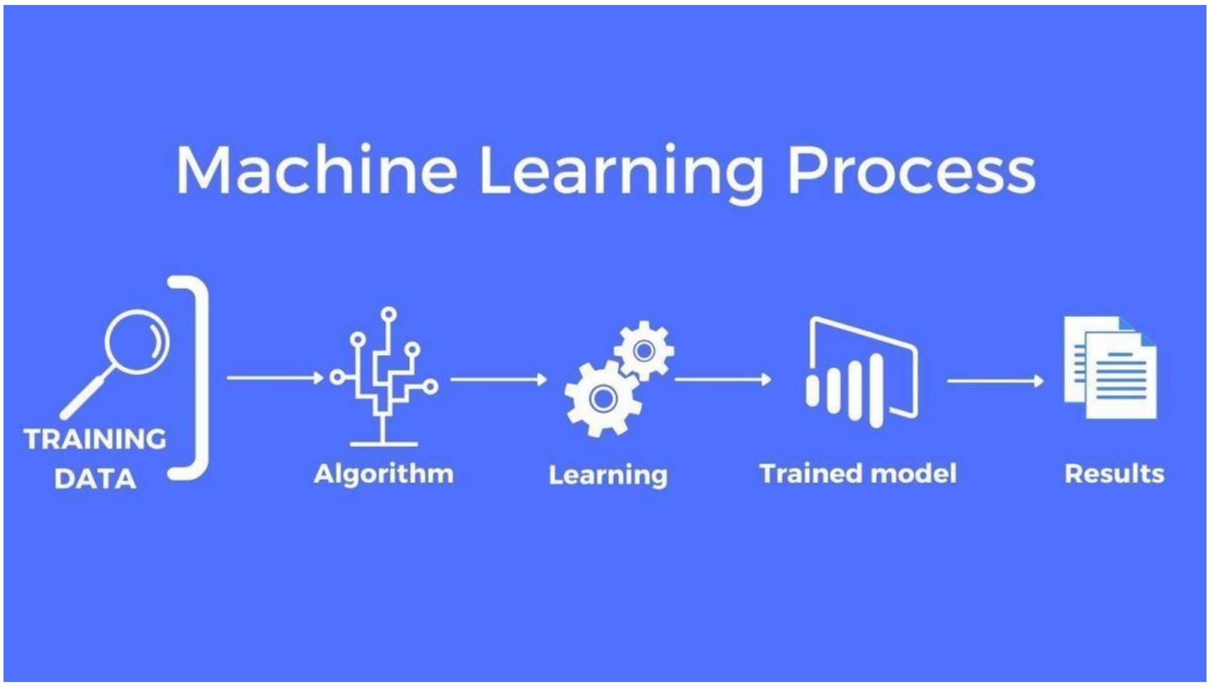
These results can look different depending on what kind of algorithm you go with. If you need to know what something is, go with a classification algorithm, which comes in two types. Binary classification categorizes data between two categories. Multi-class classification sorts data between—you guessed it—multiple categories.

When the result you're looking for is an actual number, you'll want to use a regression algorithm. Regression takes a lot of different data with different weights of importance and analyzes it with historical data to objectively provide an end result.

Both regression and classification are supervised types of algorithms, meaning you need to provide intentional data and direction for the computer to learn. There is also unsupervised algorithms which don't require labeled data or any guidance on the kind of result you're looking for.

One form of unsupervised algorithms is clustering. You use clustering when you want to understand the structure of your data. You provide a set of data and let the algorithm identify the categories within that set. On the other hand, anomaly is an unsupervised algorithm you can use when your data looks normal and uniform, and you want the algorithm to pull anything out of the ordinary that doesn't fit with the rest of the data.

Although supervised algorithms are more common, it's good to play around with each algorithm type and use case to better understand probability and practice splitting and training data in different ways. The more you toy with your data, the better your understanding of what machine learning can accomplish will become.



Data: The dataset you want to use must be well-structured, accurate. The data you use can be labeled or unlabeled. Unlabeled data are sample items — e.g. photos, videos, news articles — that don't need additional explanation, while labeled ones are augmented: unlabeled information is bundled and an explanation, with a tag, is added to them.

Algorithm: there are different types of algorithms that can be used (e.g. linear regression, logistic regression). Choosing the right algorithm is both a combination of business need, specification, experimentation and time available.

Model: A "model" is the output of a machine learning algorithm run on data. It represents the rules, numbers, and any other algorithmspecific data structures required to make predictions