FOR DEVELOPERS

How to Build a Machine Learning Pipeline With Scikit-Learn in Python

Share

Machine learning (ML) pipelines comprise a set of steps to follow when working on a project. They help streamline the machine learning workflow, allowing for neat solutions and faster processes. This article will explore how to build a machine learning pipeline in Python using scikit-learn, a popular library used in data science and machine learning tasks. We will begin with an example without a pipeline and then demonstrate how we can use the scikit-learn library to create an ML pipeline. Basic knowledge of Python and machine learning is required to follow this tutorial comfortably.

The article will focus on supervised learning to complete a regression task, which predicts continuous target variables, such as prices or the number of years an NBA player is likely to continue playing.

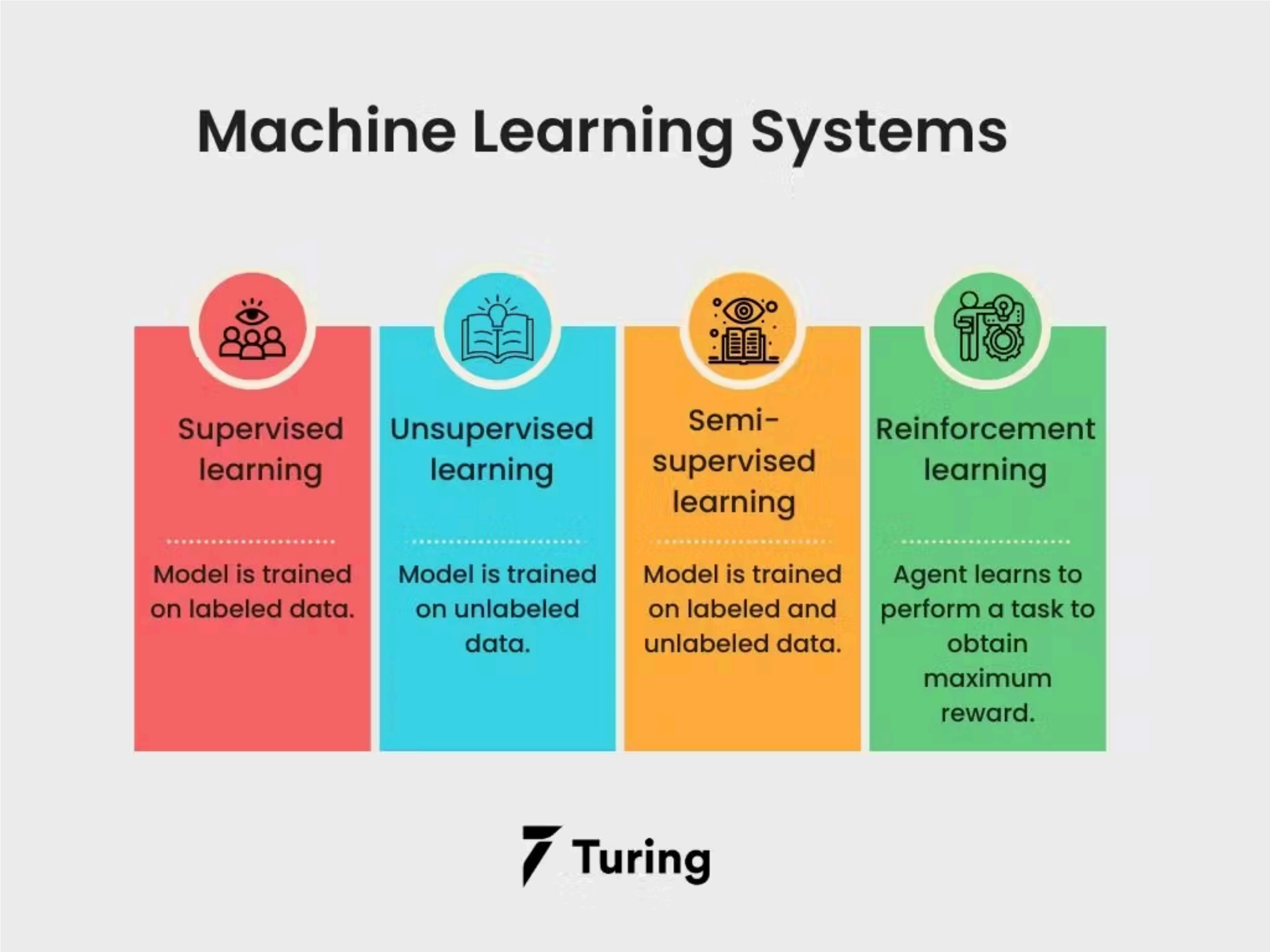
Table of Contents

What is machine learning?

Machine learning is one of the areas of artificial intelligence (AI). It is defined as the study of programs that are not explicitly programmed, but instead, the algorithms learn patterns from data. It is where a computer program is said to learn from experience E with respect to some task T and some performance P if its performance on T, as measured by P, improves with experience E.

Over the last decade, ML has been adopted in different industries for various tasks, including breast cancer detection, building recommendation systems, and chatbots. The availability of large datasets and cheaper computing power has helped progress the growth of ML.

Types of machine learning systems


Machine Learning Systems
Supervised learning
Model is trained on labeled data.
Unsupervised learning
Model is trained on unlabeled data.
Semi-supervised learning
Model is trained on labeled and unlabeled data.
Reinforcement learning
Agent learns to perform a task to obtain maximum reward.
Turing

There are different types of machine learning systems, including:

Supervised learning

In supervised learning, the model is trained on labeled data. The most common supervised learning tasks are:

- Regression (predicting values)

Unsupervised learning

In unsupervised learning, the model is trained on unlabeled data and the algorithm learns to identify patterns.

Semi-supervised learning

In semi-supervised learning, the model is trained on labeled and unlabeled data, with most of the samples being unlabeled.

Reinforcement learning

In reinforcement learning, the model learns a policy by observing its environment where its actions are either rewarded or penalized.

Building a machine learning model step-by-step

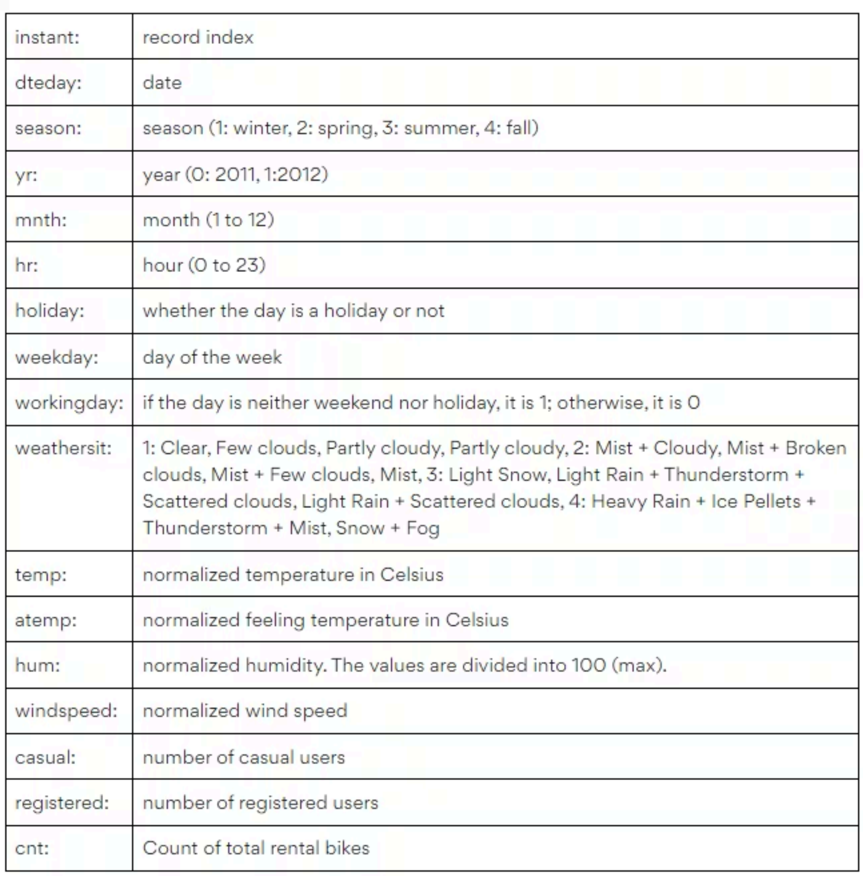
In this section, we will understand how to build a machine learning model by following the workflow step-by-step.

Machine learning workflow

Problem statement and data collection

We will use the Bike sharing dataset which you can download here.
The data contains records of the number of bikes rented in a given hour and other features. We will use these features to build a model that can predict the number of bikes that will be rented.

The table below describes the features of the dataset.



instant:	record index
dteday:	date
season:	season (1: winter, 2: spring, 3: summer, 4: fall)
yr:	year (0: 2011, 1:2012)
mnth:	month (1 to 12)
hr:	hour (0 to 23)
holiday:	whether the day is a holiday or not
weekday:	day of the week
workingday:	if the day is neither weekend nor holiday, it is 1; otherwise, it is 0
weathersit:	1: Clear, Few clouds, Partly cloudy, Partly cloudy, 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist, 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds, 4: Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog
temp:	normalized temperature in Celsius
atemp:	normalized feeling temperature in Celsius
hum:	normalized humidity. The values are divided into 100 (max).
windspeed:	normalized wind speed
casual:	number of casual users
registered:	number of registered users
cnt:	Count of total rental bikes

Bike rental is affected by weather conditions and factors, such as whether it is a holiday.

incomplete or in a form that cannot be used directly.

Let us begin by loading the data into our notebook after importing some libraries that we will use.

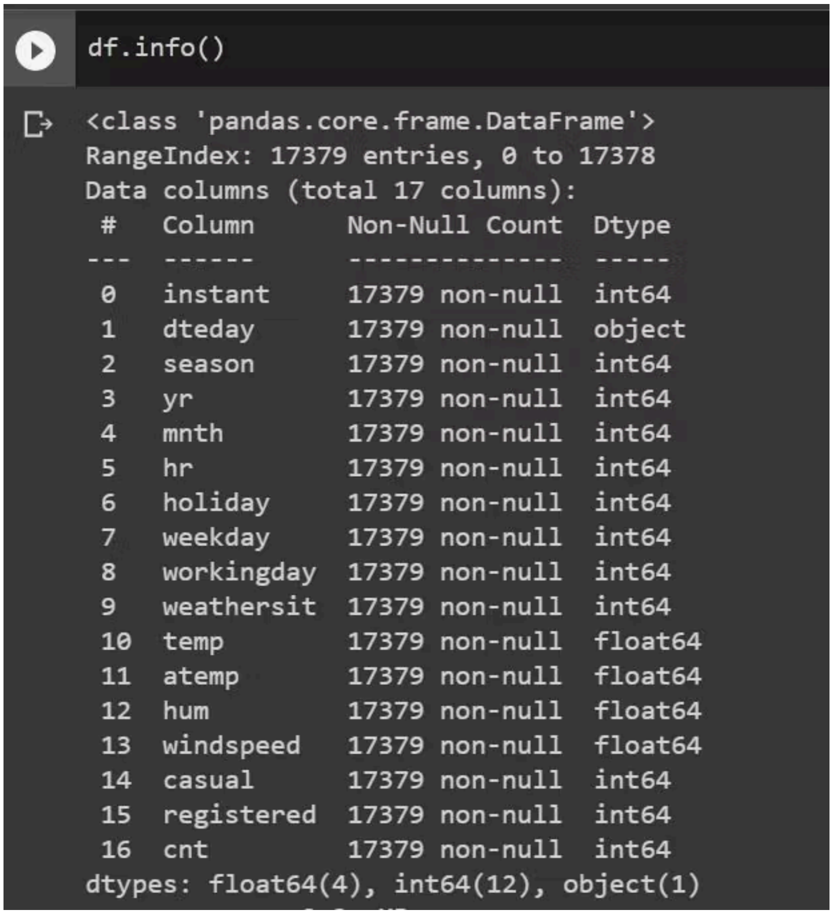
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the data:

```
df = pd.read_csv("hour.csv")
df.head()
```

We can get a summary of the dataset using Panda's dataframe.info () method which shows the column names, the number of non-null rows, and their data types.

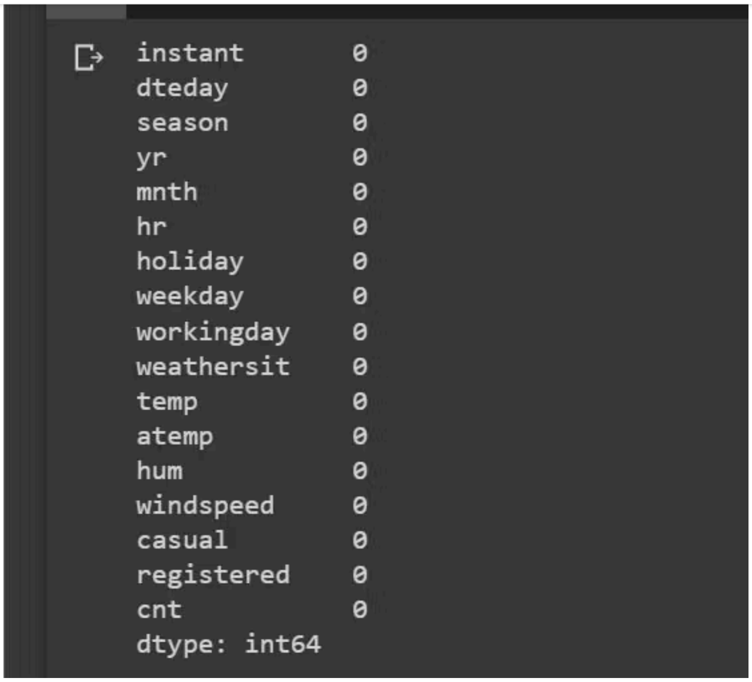
```
df.info()
```



```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instant     17379 non-null  int64
 1   dteday      17379 non-null  object
 2   season      17379 non-null  int64
 3   yr          17379 non-null  int64
 4   mnth        17379 non-null  int64
 5   hr          17379 non-null  int64
 6   holiday     17379 non-null  int64
 7   weekday     17379 non-null  int64
 8   workingday  17379 non-null  int64
 9   weathersit  17379 non-null  int64
 10  temp        17379 non-null  float64
 11  atemp       17379 non-null  float64
 12  hum         17379 non-null  float64
 13  windspeed   17379 non-null  float64
 14  casual      17379 non-null  int64
 15  registered  17379 non-null  int64
 16  cnt         17379 non-null  int64
dtypes: float64(4), int64(12), object(1)
```

Let us check if the dataset has any missing values to ensure they do not affect the model's overall performance.

```
df.isna().sum()
```



```
instant       0
dteday        0
season        0
yr            0
mnth          0
hr            0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

The data does not have missing values. However, the column names could be more informative. Let us rename them to more legible labels.

```
df.rename(columns={'instant':'record_id',
                        'dteday':'datetime',
                        'holiday':'is_holiday',
                        'workingday':'is_workingday',
                        'weathersit':'weather_condition',
                        'hum':'humidity',
                        'mnth':'month',
                        'cnt':'total_count',
                        'hr':'hour',
                        'yr':'year'},inplace=True)
```

Drop the ('record_id') because it does not contain any additional information about bike rentals. The 'casual' and 'registered' columns are combined to form the 'total_count' column; therefore, we can also drop these columns. This helps avoid data leakage, which happens when the training data contains information about the target variable that may not be available in the test set. In production, the model's performance will be affected.

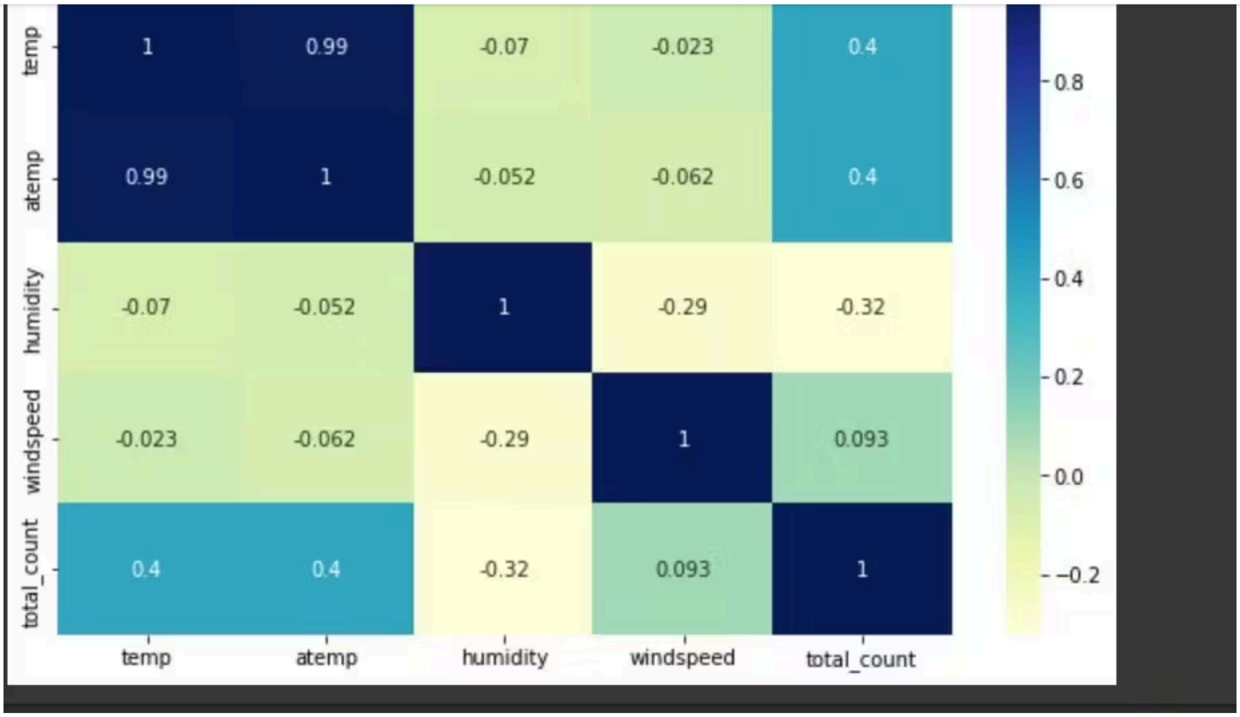
```
 #the registered and casual columns are added to the total count
df. drop(['record_id','casual', 'registered'], axis=1, inplace=True)
```

The columns have different data types which we must convert to the most appropriate data type.

```
df['datetime'] = pd.to_datetime(df.datetime)
categorical variables


df['season'] = df.season.astype('category')
df['is_holiday'] = df.is_holiday.astype('category')
df['weekday'] = df.weekday.astype('category')
df['weather_condition'] = df.weather_condition.astype('category')
df['is_workingday'] = df.is_workingday.astype('category')
df['month'] = df.month.astype('category')
df['year'] = df.year.astype('category')
df['hour'] = df.hour.astype('category')
```

To understand the correlation between different features, create a correlation heat map. Such a map shows a correlation matrix between two dimensions. The Seaborn library is used to visualize the correlation map.

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, cmap="Blues")
```



There is a high correlation between the temp and atemp features; therefore, we drop the 'atemp' column to avoid the effects of multicollinearity. Multicollinearity refers to where an independent variable can be predicted from another, making it difficult to determine the effects of each variable on the model. A deeper dive into multicollinearity is provided here.

```
df.drop('atemp', axis=1, inplace=True)
```

We will also drop the ‘datetime’ column

We have completed the data cleaning process and are ready to train our model.

First, we specify our features X and target variable Y and split the dataset into training and test sets. We use scikit-learn's train_test_split() method to split the dataset into 70% training and 30% test data.

```
from sklearn.model_selection import train_test_split


X = df.drop(['total_count'],axis=1)
y = df['total_count']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Now, we can train our model.

Modeling: Training and testing

After preprocessing the data, we choose a machine learning algorithm to train and test the data on and make predictions. We use two algorithms employed in regression tasks: linear regression and random forest regression.

Linear regression is a parametric algorithm that finds the linear relationship between the features X and the target Y. It uses this relationship to predict the target variable given the dependent variables (features) during testing and production.

Learn more about the inner workings of linear regression here.

The scikit-learn library has several machine learning algorithms, including linear regression and random forest regression. We do not have to write them from scratch. We can import the algorithm and use the fit() and predict() methods to train the model and test, respectively.

```
import math
import sklearn.metrics
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
predictions = regressor.predict(X_test)


mse = sklearn.metrics.mean_squared_error(y_test, predictions)

rmse = math.sqrt(mse)
```

We test the performance of the model on the test dataset using the root mean square error (RMSE) which determines the accuracy of the model. A lower RMSE value indicates that the model makes fewer mistakes in its predictions.

Let us train the model with the same data using the random forest regression algorithm and compare the RMSE values.

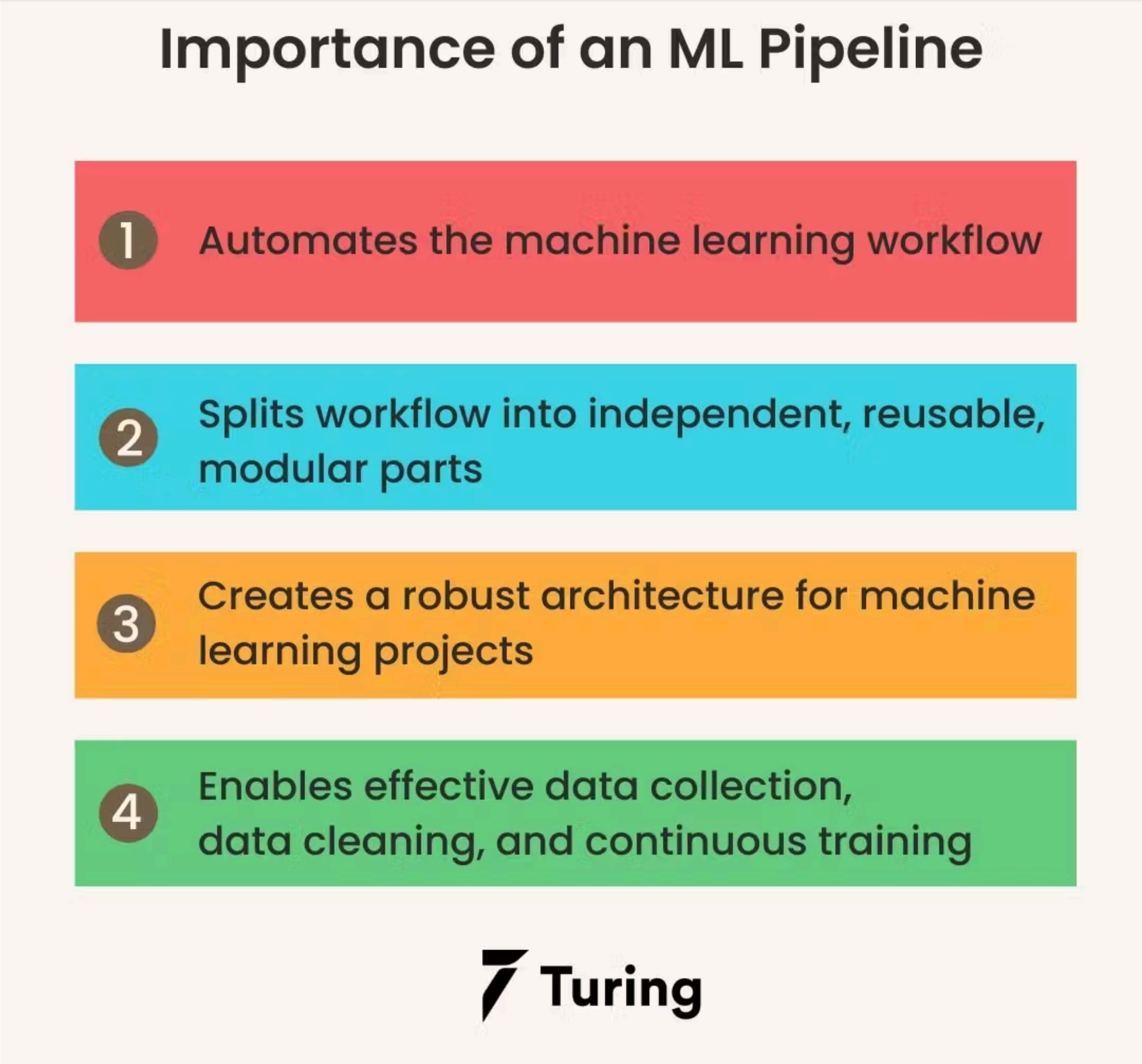
```
from sklearn.ensemble import RandomForestRegressor


random_regressor = RandomForestRegressor()


random_regressor.fit(X_train, y_train)
preds = random_regressor.predict(X_test)


mean_error = sklearn.metrics.mean_squared_error(y_test, preds)

root_mean = math.sqrt(mean_error)
root_mean
```

```
42.5293221034906
```

The random forest regression model has a lower RMSE value; therefore, it is a better model compared to linear regression.

We will use the random forest regression model to build the ML pipeline. Note that you can experiment with other models to see how they perform and select the best.

The necessity of a machine learning pipeline



Importance of an ML Pipeline

1 Automates the machine learning workflow

2 Splits workflow into independent, reusable, modular parts

3 Creates a robust architecture for machine learning projects

4 Enables effective data collection, data cleaning, and continuous training

As observed from the steps above, the machine learning workflow involves several stages. They can be time-consuming, especially data cleaning and data processing. Every time we need to train the model with additional data, we have to repeat these steps. Hence, it is useful to automate the process.

A machine learning pipeline is a means of automating the end-to-end machine learning workflow. The ML pipeline uses the defined preprocessing steps on the supplied input to produce the expected output in each stage. ML pipelines can be implemented as a sequence of components where the ML workflow is split up into independent, reusable, modular parts that can then be combined to build models. This creates a more robust architecture for machine learning projects similar to the microservices architecture.

An ML pipeline can also be used as a single component for data preprocessing, taking data as input and predictions as output. Using a pipeline allows for effective data collection, data cleaning, and continuous training.

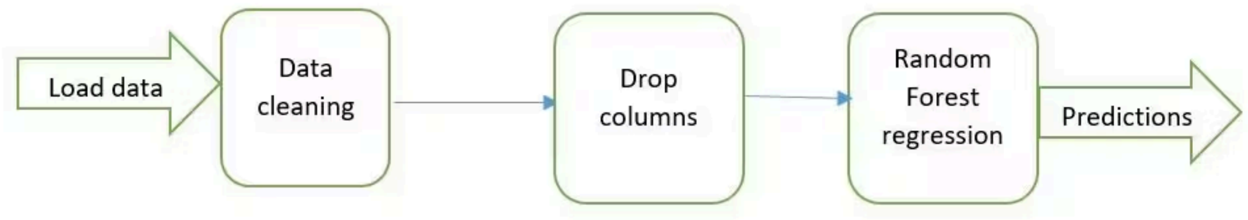
Machine learning pipeline example

Scikit-learn provides a built-in function for creating pipelines. The library offers two functions, sklearn pipeline and sklearn make_pipeline, which simplifies pipeline construction further. You can learn more about make_pipeline here and explore all the parameters of the sklearn pipeline in the documentation.

Below, we build a pipeline based on the data and steps we worked with previously.

3. Split the data
4. Apply transformations to the data using the 'fit()' method
5. Make predictions and evaluate model performance.

The figure below shows the process in our pipeline:



Let us begin by importing the necessary packages and loading the dataset.

```
import pandas as pd
import numpy as np
df = pd.read_csv("hour.csv")
```

From the data exploration and data visualization we did previously, we know that some column names in our dataset need to be more descriptive and we need to rename them.

```
df.rename(columns={'instant':'record_id',
                        'dteday':'datetime',
                        'holiday':'is_holiday',
                        'workingday':'is_workingday',
                        'weathersit':'weather_condition',
                        'hum':'humidity',
                        'mnth':'month',
                        'cnt':'total_count',
                        'hr':'hour',
                        'yr':'year'},inplace=True)
```

Let us drop columns that we will not use in training the model.

```
df. drop(['record_id','casual', 'registered', 'datetime', 'temp'], axis=1, inplace=True)
```

The data is ready for model training. We will be using the random forest regression algorithm.

To create a pipeline, we import Pipeline from the scikit-learn package.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline
```

We need to split the data into a training and test set.

```
X = df.drop(['total_count'],axis=1)
y = df['total_count']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
```

The Pipeline class takes several parameters:

```
class sklearn.pipeline.Pipeline(steps, *, memory=None, verbose=False)
```

Steps is a list of tuples of the transforms to be performed on the data. Each tuple consists of a name and a transform. The last transform must be an estimator - the machine learning algorithm which will be used for training and making predictions. The other parameters are optional depending on your specific implementation of scikit-learn pipelines.

In this example, the Pipeline class only has one tuple because the data preprocessing steps have already been implemented. This tuple will be used as the estimator.

```
pipeline = Pipeline(steps = [
            ('regressor',RandomForestRegressor())
           ])
```

Train the model using the fit() method, which uses the training dataset.

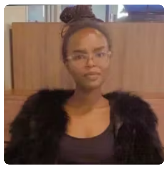
```
predictions = model.predict(X_test)
print(math.sqrt(mean_squared_error(y_test,predictions)))
```

```
42.64736859923981
```

Here is the entire code used in this tutorial.

In this article, we learned to build a simple machine learning pipeline in Python with scikit-learn since our dataset did not need a lot of transformation to be used in training. We can build more complex pipelines depending on the transformations performed on the dataset, such as encoding categorical variables and replacing missing values.

Author



Bernice Waweru

Bernice Waweru is a competent technical writer and software engineer with an interest in machine learning. She has experience working in startups and large corporations and is looking forward to growing her tech skills further.

Related articles

How Does a Recommendation Engine Work With Predicting Likes?

An automated recommendation engine suggests products and services to users based on...

Read more

Applications of Genetic Algorithms in Machine Learning

Genetic algorithm in machine learning is mainly adaptive heuristic or search engine algorithms...

Read more

10 Best Python Libraries for Machine Learning in 2024

Python programming language is widely used for machine learning as it is one of the easiest...

Read more