**5121-RANIPPETTAI ENGINEERING COLLEGE**

Name -N. DHINESH

register No - 512121104302

**PERSONAL BLONG ON IBM CLOUD STATIC WEB APPS**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# Introduction to loading data

This document provides an overview of loading data into BigQuery.

## Overview

There are several ways to ingest data into BigQuery:

* Batch load a set of data records.
* Stream individual records or batches of records.
* Use queries to generate new data and append or overwrite the results to a table.
* Use a third-party application or service.

### Batch loading

With batch loading, you load the source data into a BigQuery table in a single batch operation. For example, the data source could be a CSV file, an external database, or a set of log files. Traditional extract, transform, and load (ETL) jobs fall into this category.

Options for batch loading in BigQuery include the following:

* **Load jobs.** Load data from Cloud Storage or from a local file by creating a [load job](https://cloud.google.com/bigquery/docs/batch-loading-data). The records can be in Avro, CSV, JSON, ORC, or Parquet format.
* **SQL**. The [LOAD DATA](https://cloud.google.com/bigquery/docs/reference/standard-sql/other-statements" \l "load_data_statement) SQL statement loads data from one or more files into a new or existing table. You can use the LOAD DATA statement to load Avro, CSV, JSON, ORC, or Parquet files.
* **BigQuery Data Transfer Service.** Use [BigQuery Data Transfer Service](https://cloud.google.com/bigquery/docs/dts-introduction) to automate loading data from Google Software as a Service (SaaS) apps or from third-party applications and services.
* **BigQuery Storage Write API.** The [Storage Write API](https://cloud.google.com/bigquery/docs/write-api) lets you batch-process an arbitrarily large number of records and commit them in a single atomic operation. If the commit operation fails, you can safely retry the operation. Unlike BigQuery load jobs, the Storage Write API does not require staging the data to intermediate storage such as Cloud Storage.
* **Other managed services.** Use other managed services to export data from an external data store and import it into BigQuery. For example, you can load data from [Firestore exports](https://cloud.google.com/bigquery/docs/loading-data-cloud-firestore).

When choosing a batch load method, most file-based patterns should use either [load job](https://cloud.google.com/bigquery/docs/batch-loading-data) or [LOAD DATA SQL statement](https://cloud.google.com/bigquery/docs/reference/standard-sql/other-statements" \l "load_data_statement) to batch load data. Other services should generally use BigQuery Data Transfer Service or [export data from Google services](https://cloud.google.com/bigquery/docs/load-data-google-services).

Batch loading can be done as a one-time operation or on a recurring schedule. For example, you can do the following:

* You can run BigQuery Data Transfer Service transfers on a schedule.
* You can use an orchestration service such as Cloud Composer to schedule load jobs.
* You can use a cron job to load data on a schedule.

### Streaming

With streaming, you continually send smaller batches of data in real time, so the data is available for querying as it arrives. Options for streaming in BigQuery include the following:

* **Storage Write API.** The [Storage Write API](https://cloud.google.com/bigquery/docs/write-api) supports high-throughput streaming ingestion with exactly-once delivery semantics.
* **Dataflow.** Use [Dataflow](https://cloud.google.com/dataflow/docs) with the Apache Beam SDK to set up a streaming pipeline that writes to BigQuery. For more information, see [BigQuery I/O connector](https://beam.apache.org/documentation/io/built-in/google-bigquery) in the Apache Beam documentation and the [Stream from Pub/Sub to BigQuery tutorial](https://cloud.google.com/dataflow/docs/tutorials/dataflow-stream-to-bigquery).
* **Datastream.** [Datastream](https://cloud.google.com/datastream-for-bigquery) uses BigQuery change data capture functionality and the [Storage Write API](https://cloud.google.com/bigquery/docs/write-api) to replicate data and schema updates from operational databases directly into BigQuery. Follow this [quickstart](https://cloud.google.com/datastream/docs/quickstart-replication-to-bigquery) for an example of replicating from a Cloud SQL for PostgreSQL database into BigQuery.
* **BigQuery Connector for SAP.** The BigQuery Connector for SAP enables near real time replication of SAP data directly into BigQuery. For more information, see the [BigQuery Connector for SAP planning guide](https://cloud.google.com/solutions/sap/docs/bq-connector-for-sap-planning).
* **Pub/Sub.** [Pub/Sub](https://cloud.google.com/pubsub/docs/overview) is a messaging service you can use to coordinate streaming analytics and data integration pipelines. You can use [BigQuery subscriptions](https://cloud.google.com/pubsub/docs/bigquery) to write messages directly to an existing BigQuery table.

### Generated data

You can use SQL to generate data and store the results in BigQuery. Options for generating data include:

* Use [data manipulation language](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-manipulation-language) (DML) statements to perform bulk inserts into an existing table or store query results in a new table.
* Use a [CREATE TABLE ... AS](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language" \l "create_table_statement) statement to create a new table from a query result.
* Run a query and save the results to a table. You can append the results to an existing table or write to a new table. For more information, see [Writing query results](https://cloud.google.com/bigquery/docs/writing-results).

### Third-party applications

Some third-party applications and services provide connectors that can ingest data into BigQuery. The details of how to configure and manage the ingestion pipeline depend on the application. For example, to load data from external sources to BigQuery's storage, you can use Informatica Data Loader or Fivetran Data Pipelines. For more information, see [Load data using a third-party application](https://cloud.google.com/bigquery/docs/load-data-third-party).

## Choosing a data ingestion method

Here are some considerations to think about when you choose a data ingestion method.

**Data source.** The source of the data or the data format can determine whether batch loading or streaming is simpler to implement and maintain. Consider the following points:

* If the BigQuery Data Transfer Service supports the data source, transferring the data directly into BigQuery might be the simplest solution to implement.
* For data from third-party sources that aren't supported by the BigQuery Data Transfer Service, transform the data into a format supported by [batch loading](https://cloud.google.com/bigquery/docs/batch-loading-data) and use that method instead.
* If your data comes from Spark or Hadoop, consider using [BigQuery connectors](https://cloud.google.com/dataproc/docs/concepts/connectors/bigquery) to simplify data ingestion.
* For local files, consider batch load jobs, especially if BigQuery supports the file format without requiring a transformation or data cleansing step.
* For application data such as application events or a log stream, it might be easier to stream the data in real time, rather than implement batch loading.

**Slow-changing versus fast-changing data.** If you need to ingest and analyze data in near real time, consider streaming the data. With streaming, the data is available for querying as soon as each record arrives. Avoid using DML statements to submit large numbers of individual row updates or insertions. For frequently updated data, it's often better to stream a change log and use a view to obtain the latest results. Another option is to use Cloud SQL as your online transaction processing (OLTP) database and use federated queries to join the data in BigQuery.

If your source data changes slowly or you don't need continuously updated results, consider using a load job. For example, if you use the data to run a daily or hourly report, load jobs can be less expensive and can use fewer system resources.

Another scenario is data that arrives infrequently or in response to an event. In that case, consider using Dataflow to stream the data or use Cloud Functions to call the streaming API in response to a trigger.

**Reliability of the solution**. BigQuery has a [Service Level Agreement](https://cloud.google.com/bigquery/sla) (SLA). However, you also need to consider the reliability of the particular solution that you implement. Consider the following points:

* With loosely typed formats such as JSON or CSV, bad data can make an entire load job fail. Consider whether you need a data cleansing step before loading, and consider how to respond to errors. Also consider using a strongly typed format such as Avro, ORC, or Parquet.
* Periodic load jobs require scheduling, using Cloud Composer, cron, or another tool. The scheduling component could be a failure point in the solution.
* With streaming, you can check the success of each record and quickly report an error. Consider writing failed messages to an unprocessed messages queue for later analysis and processing. For more information about BigQuery streaming errors, see [Troubleshooting streaming inserts](https://cloud.google.com/bigquery/docs/error-messages" \l "streaming).
* Streaming and load jobs are subject to [quotas](https://cloud.google.com/bigquery/quotas). For information about how to handle quota errors, see [Troubleshooting BigQuery quota errors](https://cloud.google.com/bigquery/docs/troubleshoot-quotas).
* Third-party solutions might differ in configurability, reliability, ordering guarantees, and other factors, so consider these before adopting a solution.

**Latency.** Consider how much data you load and how soon you need the data to be available. Streaming offers the lowest latency of data being available for analysis. Periodic load jobs have a higher latency, because new data is only available after each load job finishes.

Load jobs use a shared pool of [slots](https://cloud.google.com/bigquery/docs/slots) by default. A load job might wait in a pending state until slots are available, especially if you load a very large amount of data. If that creates unacceptable wait times, you can purchase dedicated slots, instead of using the shared slot pool. For more information, see [Introduction to Reservations](https://cloud.google.com/bigquery/docs/reservations-intro).

Query performance for external data sources might not be as high as query performance for data stored in BigQuery. If minimizing query latency is important, then we recommend loading the data into BigQuery.

**Data ingestion format**. Choose a data ingestion format based on the following factors:

* **Schema support.** Avro, ORC, Parquet, and Firestore exports are self-describing formats. BigQuery creates the table schema automatically based on the source data. For JSON and CSV data, you can provide an explicit schema, or you can use [schema auto-detection](https://cloud.google.com/bigquery/docs/schema-detect).
* **Flat data or nested and repeated fields.** Avro, CSV, JSON, ORC, and Parquet all support flat data. Avro, JSON, ORC, Parquet, and Firestore exports also support data with nested and repeated fields. Nested and repeated data is useful for expressing hierarchical data. Nested and repeated fields also reduce data duplication when [loading the data](https://cloud.google.com/bigquery/docs/loading-data" \l "loading_nested_and_repeated_data).
* **Embedded newlines.** When you are loading data from JSON files, the rows must be newline delimited. BigQuery expects newline-delimited JSON files to contain a single record per line.
* **Encoding.** BigQuery supports UTF-8 encoding for both nested or repeated and flat data. BigQuery supports ISO-8859-1 encoding for flat data only for CSV files.

## Load nested and repeated data

You can load data into nested and repeated fields in the following data formats:

* Avro
* JSON (newline delimited)
* ORC
* Parquet
* Datastore exports
* Firestore exports

For information about specifying nested and repeated fields in your schema when you're loading data, see [Specifying nested and repeated fields](https://cloud.google.com/bigquery/docs/nested-repeated).

## Load data from other Google services

Some Google services export data to BigQuery using scheduled queries, exports, or transfers. For more information about services that support exports to BigQuery, see [Load data from Google services](https://cloud.google.com/bigquery/docs/load-data-google-services).

Other Google services support data exports initiated from BigQuery Data Transfer Service. For more information about services that support exports initiated by BigQuery Data Transfer Service, see [BigQuery Data Transfer Service](https://cloud.google.com/bigquery/docs/dts-introduction).

## Quota

For information about quotas, see the following sections:

* [Load jobs quota](https://cloud.google.com/bigquery/quotas" \l "load_jobs).
* [BigQuery Storage Write API quota](https://cloud.google.com/bigquery/quotas" \l "write-api-limits).
* [Streaming inserts quota](https://cloud.google.com/bigquery/quotas" \l "streaming_inserts).

## Alternatives to loading data

You don't need to load data before running queries in the following situations:

**Public datasets**

Public datasets are datasets stored in BigQuery and shared with the public. For more information, see [BigQuery public datasets](https://cloud.google.com/bigquery/public-data).

**Shared datasets**

You can share datasets stored in BigQuery. If someone has shared a dataset with you, you can run queries on that dataset without loading the data.

**External data sources**

BigQuery can run queries on certain forms of external data, without loading the data into BigQuery storage. This approach lets you take advantage of the analytic capabilities of BigQuery without moving data that is stored elsewhere. For information about the benefits and limitations of this approach, see [external data sources](https://cloud.google.com/bigquery/external-data-sources).

**Logging files**

Cloud Logging provides an option to export log files into BigQuery. See [Configure and manage sinks](https://cloud.google.com/logging/docs/export/configure_export) for more information.

**Note:** Loading data into BigQuery from Google Drive is not currently supported, but you can query data in Drive by using an [external table](https://cloud.google.com/bigquery/external-data-drive).

## Monitor usage of load jobs

You can monitor usage of load jobs using the following two ways:

* **Use Cloud Monitoring.** For more information, see [BigQuery metrics](https://cloud.google.com/monitoring/api/metrics_gcp" \l "gcp-bigquery). Specifically, you can monitor the amount of data and number of rows uploaded to a specific table. If your load jobs upload data to a specific table, this can be a proxy for monitoring load job upload data usage.
* **Use INFORMATION\_SCHEMA.JOBS\_BY\_PROJECT.** You can use the INFORMATION\_SCHEMA.JOBS\_BY\_PROJECT view to [get the number of load jobs per table per day](https://cloud.google.com/bigquery/docs/information-schema-jobs" \l "load-job-quota).

## Example use case

The following examples explain the methods to use based on your use case and how to use them with other data analytics solutions.

### Stream data using Storage Write API

Suppose that there is a pipeline processing event data from endpoint logs. Events are generated continuously and need to be available for querying in BigQuery as soon as possible. As data freshness is paramount for this use case, the [Storage Write API](https://cloud.google.com/bigquery/docs/write-api) is the best choice to ingest data into BigQuery. A [recommended architecture](https://cloud.google.com/dataflow/docs/guides/templates/provided-streaming) to keep these endpoints lean is sending events to Pub/Sub, from where they are consumed by a streaming Dataflow pipeline which directly streams to BigQuery.

A primary reliability concern for this architecture is how to deal with failing to insert a record into BigQuery. If each record is important and cannot be lost, data needs to be buffered before attempting to insert. In the recommended architecture above, Pub/Sub can play the role of a buffer with its message retention capabilities. The Dataflow pipeline should be configured to retry BigQuery streaming inserts with [truncated exponential backoff](https://en.wikipedia.org/wiki/Exponential_backoff" \t "external). After the capacity of Pub/Sub as a buffer is exhausted, for example in the case of prolonged unavailability of BigQuery or a network failure, data needs to be persisted on the client and the client needs a mechanism to resume inserting persisted records once availability is restored. For more information about how to handle this situation, see the [Google Pub/Sub Reliability Guide](https://medium.com/google-cloud/google-cloud-pub-sub-reliability-user-guide-part-1-publishing-12577b9069fd) blog post.

Another failure case to handle is that of a poison record. A poison record is either a record rejected by BigQuery because the record fails to insert with a non-retryable error or a record that has not been successfully inserted after the maximum number of retries. Both types of records should be stored in a "[dead letter queue](https://cloud.google.com/pubsub/docs/handling-failures" \l "dead_letter_topic)" by the Dataflow pipeline for further investigation.

If exactly-once semantics are required, create a write stream in [committed type](https://cloud.google.com/bigquery/docs/write-api-streaming" \l "exactly-once), with record offsets provided by the client. This avoids duplicates, as the write operation is only performed if the offset value matches the next append offset. Not providing an offset means records are appended to the current end of the stream and retrying a failed append could result in the record appearing more than once in the stream.

If exactly-once guarantees are not required, [writing to the default stream](https://cloud.google.com/bigquery/docs/write-api" \l "default_stream) allows for a higher throughput and also does not count against the [quota limit](https://cloud.google.com/bigquery/docs/write-api" \l "quotas) on creating write streams.

[Estimate the throughput of your network](https://cloud.google.com/bigquery/docs/write-api" \l "calculate-throughput) and ensure in advance that you have an adequate quota to serve the throughput.

If your workload is generating or processing data at a very uneven rate, then try to smooth out any load spikes on the client and stream into BigQuery with a constant throughput. This can simplify your capacity planning. If that is not possible, ensure you are prepared to handle 429 (resource exhausted) errors if and when your throughput goes over quota during short spikes.

### Batch data processing

Suppose there is a nightly batch processing pipeline that needs to be completed by a fixed deadline. Data needs to be available by this deadline for further processing by another batch process to generate reports to be sent to a regulator. This use case is common in regulated industries such as finance.

[Batch loading of data with load jobs](https://cloud.google.com/bigquery/docs/batch-loading-data) is the right approach for this use case because latency is not a concern provided the deadline can be met. Ensure your Cloud Storage buckets [meet the location requirements](https://cloud.google.com/bigquery/docs/batch-loading-data" \l "data-locations) for loading data into the BigQuery dataset.

The result of a BigQuery load job is atomic; either all records get inserted or none do. As a best practice, when inserting all data in a single load job, create a new table by using the WRITE\_TRUNCATE disposition of the [JobConfigurationLoad](https://cloud.google.com/bigquery/docs/reference/rest/v2/Job" \l "jobconfigurationquery) resource. This is important when retrying a failed load job, as the client might not be able to distinguish between jobs that have failed and the failure caused by for example in communicating the success state back to the client.

Assuming data to be ingested has been successfully copied to Cloud Storage already, retrying with exponential backoff is sufficient to address ingestion failures.

It's recommended that a nightly batch job doesn't hit the [default quota](https://cloud.google.com/bigquery/quotas" \l "load_jobs) of 1,500 loads per table per day even with retries. When loading data incrementally, the default quota is sufficient for running a load job every 5 minutes and have unconsumed quota for at least 1 retry per job on average.

## What's next

* To learn how to load data from Cloud Storage into BigQuery, see the documentation for your data format:
  + [Avro](https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-avro)
  + [CSV](https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-csv)
  + [JSON](https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-json)
  + [ORC](https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-orc)
  + [Parquet](https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-parquet)
  + [Datastore exports](https://cloud.google.com/bigquery/docs/loading-data-cloud-datastore)
  + [Firestore exports](https://cloud.google.com/bigquery/docs/loading-data-cloud-firestore)
* To learn how to load data from a local file, see [Loading data from local files](https://cloud.google.com/bigquery/docs/batch-loading-data" \l "loading_data_from_local_files).
* For information about streaming data, see [Streaming data into BigQuery](https://cloud.google.com/bigquery/docs/streaming-data-into-bigquery).

## Data Preprocessing In Machine Learning: What Is It?

Data preprocessing steps are a part of the data analysis and mining process responsible for converting raw data into a format understandable by the ML algorithms.

Text, photos, video, and other types of unprocessed, real-world data are disorganized. It may not only be inaccurate and inconsistent, but it is frequently lacking and doesn’t have a regular, consistent design. Machines prefer to process neat and orderly information; they read data as binary – 1s and 0s.

So, it is simple to calculate structured data like whole numbers and percentages. But before analysis, unstructured data, such as text and photos, must be prepped and formatted with the help of data preprocessing in Machine Learning.

Now that you know what is data preprocessing in machine learning, explore the major tasks in data preprocessing.

## Data Preprocessing Steps In Machine Learning: Major Tasks Involved

Data cleaning, Data transformation, Data reduction, and Data integration are the major steps in data preprocessing.

### Data Cleaning

Data cleaning, one of the major preprocessing steps in machine learning, locates and fixes errors or discrepancies in the data. From duplicates and outliers to missing numbers, it fixes them all. Methods like transformation, removal, and imputation help ML professionals perform data cleaning seamlessly.

### Data Integration

Data integration is among the major responsibilities of data preprocessing in machine learning. This process integrates (merges) information extracted from multiple sources to outline and create a single dataset. The fact that you need to handle data in multiple forms, formats, and semantics makes data integration a challenging task for many ML developers.

### Data Transformation

ML programmers must pay close attention to data transformation when it comes to data preprocessing steps. This process entails putting the data in a format that will allow for analysis. Normalization, standardization, and discretisation are common data transformation procedures. While standardization transforms data to have a zero mean and unit variance, normalization scales data to a common range. Continuous data is discretized into discrete categories using this technique.

### Data Reduction

Data reduction is the process of lowering the dataset’s size while maintaining crucial information. Through the use of feature selection and feature extraction algorithms, data reduction can be accomplished. While feature extraction entails translating the data into a lower-dimensional space while keeping the crucial information, feature selection requires choosing a subset of pertinent characteristics from the dataset.

## Why Data Preprocessing in Machine Learning?

When it comes to creating a Machine Learning model, data preprocessing is the first step marking the initiation of the process. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. This is where data preprocessing enters the scenario – it helps to clean, format, and organize the raw data, thereby making it ready-to-go for Machine Learning models. Let’s explore various steps of data preprocessing in machine learning.

*Join [Artificial Intelligence Course](https://www.upgrad.com/machine-learning-course/) online from the World’s top Universities – Masters, Executive Post Graduate Programs, and Advanced Certificate Program in ML & AI to fast-track your career.*

## Steps in Data Preprocessing in Machine Learning

 There are seven significant steps in data preprocessing in Machine Learning:

### ****1. Acquire the dataset****

Acquiring the dataset is the first step in data preprocessing 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 to use 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 medical dataset will include healthcare-related data.

There are several online sources from where you can download datasets like [https://www.kaggle.com/uciml/datasets](https://www.kaggle.com/uciml/datasets" \t "_blank) and [https://archive.ics.uci.edu/ml/index.php](https://archive.ics.uci.edu/ml/index.php" \t "_blank). You can also create a dataset by collecting data via different Python APIs. Once the dataset is ready, you must put it in CSV, or HTML, or XLSX file formats.

### 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.](https://www.upgrad.com/blog/python-libraries-for-data-science/) The predefined Python libraries can perform specific data preprocessing jobs. Importing all the crucial libraries is the second step in data preprocessing in machine learning. The three core Python libraries used for this data preprocessing 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 high-performance, 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.).

**Read**: [Machine Learning Project Ideas for Beginners](https://www.upgrad.com/blog/machine-learning-project-ideas-for-beginners/)

### ****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 preprocessing 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:

1. Save your Python file in the directory containing the dataset.
2. Go to File Explorer option in Spyder IDE and choose the required directory.
3. Now, click on the F5 button or Run option to execute the file.
4. **This is how the working directory should look.**
5. 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:
6. data\_set= pd.read\_csv(‘Dataset.csv’)
7. 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.
8. 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.
9. [Source](https://static.javatpoint.com/tutorial/machine-learning/images/data-preprocessing-machine-learning-2.png" \t "_blank)
10. This dataset contains three independent variables – country, age, and salary, and one dependent variable – purchased.
11. **How to extract the independent variables?**
12. 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.
13. x= data\_set.iloc[:,:-1].values
14. 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 –
15. [[‘India’ 38.0 68000.0]
16. [‘France’ 43.0 45000.0]
17. [‘Germany’ 30.0 54000.0]
18. [‘France’ 48.0 65000.0]
19. [‘Germany’ 40.0 nan]
20. [‘India’ 35.0 58000.0]
21. [‘Germany’ nan 53000.0]
22. [‘France’ 49.0 79000.0]
23. [‘India’ 50.0 88000.0]
24. [‘France’ 37.0 77000.0]]
25. **Must Read**: [Free deep learning course](https://www.upgrad.com/blog/deep-learning-free-online-course/)!
26. **How to extract the dependent variable?**
27. You can use the “iloc[ ]” function to extract the dependent variable as well. Here’s how you write it:
28. y= data\_set.iloc[:,3].values
29. 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 –
30. array([‘No’, ‘Yes’, ‘No’, ‘No’, ‘Yes’, ‘Yes’, ‘No’, ‘Yes’, ‘No’, ‘Yes’],
31. dtype=object)