Interview Questions for Data Analyst:

1. Introduce Yourself.

2. What is Python? How many Data Types are there in Python?

   Ans- Python is a High Level Object Oriented Interpreted Programming Language. It has 7 datatypes Integer, String, List, Tuple, Dictionary, Set, Boolean.

3. What the difference between list and a tuple.

   Ans-Tuple is a sequence type in Python which is immutable and fast while List is a sequence type (dynamic array) in Python which is mutable and slow. Tuple is operated from the cache memory of the computer while a list is executed from the primary memory of the computer. Hence tuple is faster than a list.

4. What are the top 5 types of Clustering Algorithms in Machine Learning?

   Ans- 1. K Means Clustering-
   2. Mean Shift Clustering-
   3. Density Based Spatial Clustering of Applications with Noise
   4. Expectation Maximization Clustering using Gaussian Mixture Models
   5. Agglomerative Hierarchical Clustering.

5. What is MANOVA Analysis in Data science. Kindly Explain.

   Ans-Multivariate Analysis of Variance is an ANOVA with several dependent variables.

   MANOVA is useful in experimental situations where at least some of the independent variables are manipulated. It has several advantages over ANOVA. First, by measuring several dependent variables in a single experiment, there is a better chance of discovering which factor is truly important. Second, it can protect against Type I errors that might occur if multiple ANOVA's were conducted independently. Additionally, it can reveal differences not discovered by ANOVA tests. However, there are several cautions as well. It is a substantially more complicated design than ANOVA, and therefore there can be some ambiguity about which independent variable affects each dependent variable. Thus, the observer must make many potentially subjective assumptions. Moreover, one degree of freedom is lost for each dependent variable that is added. The gain of power obtained from decreased SS error may be offset by the loss in these degrees of freedom. Finally, the dependent variables should be largely uncorrelated. If the dependent variables are highly correlated, there is little advantage in including more than one in the test given the resultant loss in degrees of freedom. Under these circumstances, use of a single ANOVA test would be preferable.

Assumptions Normal Distribution: - The dependent variable should be normally distributed within groups. Overall, the F test is robust to non-normality, if the non-normality is caused by skewness rather than by outliers. Tests for outliers should be run before performing a MANOVA, and outliers should be transformed or removed. Linearity - MANOVA assumes that there are linear relationships among all pairs of dependent variables, all pairs of covariates, and all dependent variable-covariate pairs in each cell. Therefore, when the relationship deviates from linearity, the power of the analysis will be compromised. Homogeneity of Variances: - Homogeneity of variances assumes that the dependent variables exhibit equal levels of variance across the range of predictor variables. Remember that the error variance is computed (SS error) by adding up the sums of squares within each group. If the variances in the two groups are different from each other, then adding the two together is not appropriate, and will not yield an estimate of the common within-group variance. Homoscedasticity can be examined graphically or by means of a number of statistical tests. Homogeneity of Variances and Covariances: - In multivariate designs, with multiple dependent measures, the homogeneity of variances assumption described earlier also applies. However, since there are multiple dependent variables, it is also required that their intercorrelations (covariances) are homogeneous across the cells of the design. There are various specific tests of this assumption.

6. What is Time Series Analysis in Python?

It's a sequence of observations recorded at regular time intervals. me series analysis involves understanding various aspects about the inherent nature of the series so that you are better informed to create meaningful and accurate forecasts.

7. What are the goals of Time Series Analysis?

There are two main goals of time series analysis: (a) identifying the nature of the phenomenon represented by the sequence of observations, and (b) forecasting (predicting future values of the time series variable). Both of these goals require that the pattern of observed time series data is identified and more or less formally described. Once the pattern is established, we can interpret and integrate it with other data (i.e., use it in our theory of the investigated phenomenon, e.g., seasonal commodity prices). Regardless of the depth of our understanding and the validity of our interpretation (theory) of the phenomenon, we can extrapolate the identified pattern to predict future events.

8. Try solving the Business Problem:

Suppose, you are the owner of an online grocery store. You sell 250 products online. A conventional methodology has been applied to determine the price of each product. And, the methodology is very simple – price the product at par with the market price of the product.
You plan to leverage analytics to determine pricing to maximize the revenue earned.
Out of 100000 people who visit your website, only 5000 end up purchasing your products.
Now, all those who made a purchase, you have obtained their buying patterns, including their average unit purchased etc.

To understand the impact of price variations, you tried testing different price points for each product. You got astonished by the results. The impact can be broken down into two aspects:

A lower price point increases the volume of the purchased product.

Customers compare price points of a few products more than others, to make a decision whether to buy products from your portal or not.

For instance, Product 1 might be a frequently used product. If you decrease the price point of product 1, then the customer response rate which was initially 5% goes up to 5.2% over and above the fact that the customer will purchase more of product 1.

On the other hand, decrease in product price obviously decreases the margin of the product.

Now, you want to find the optimum price points for each of the product to maximize the total profit earned

Following are the variables available in the data set:

Average Price/Unit : Market price of the product

Cost/Unit : Current cost of the product

Average Profit/Unit : Profit for each unit

Average units sold : Average number of units of product sold to a customer who makes a purchase

Incremental acquisition : For every 10% decline in unit price, this is the increase in total customer response rate. Note that overall response rate initially is 5% (5000 out of 100000 make a purchase). You are allowed to decrease the price of a product maximum by 10% by market laws.

Increase in sale volume : For every 10% decline in unit price of product, this is the increase in volume. Again, you are allowed to decrease the price of a product maximum by 10% by market laws.

Note: The maximum price hike permitted is 20%. So, basically the price of a product can be varied between -10% to +20% around the average price/unit.

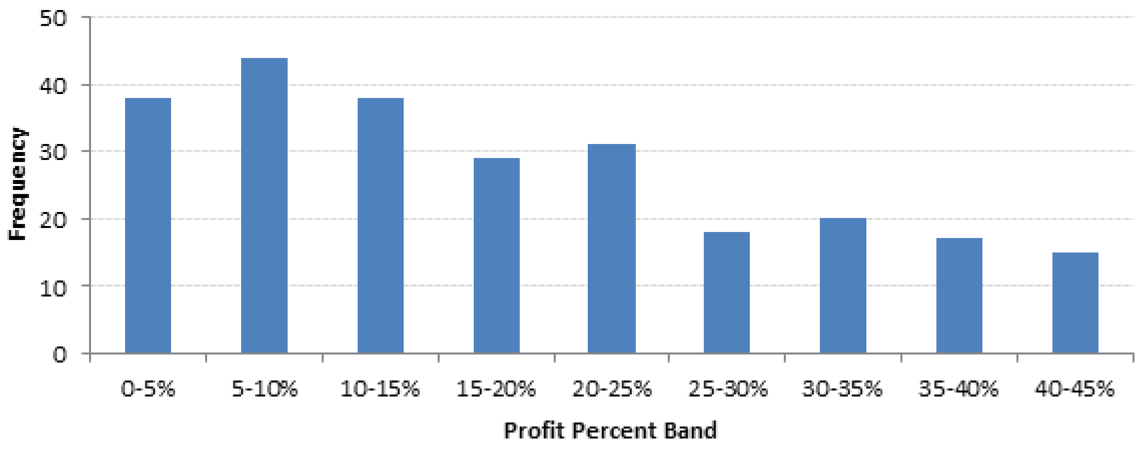
If you make the calculation of the profit earned per customer who comes to your portal :

Total Profit Earned : $165

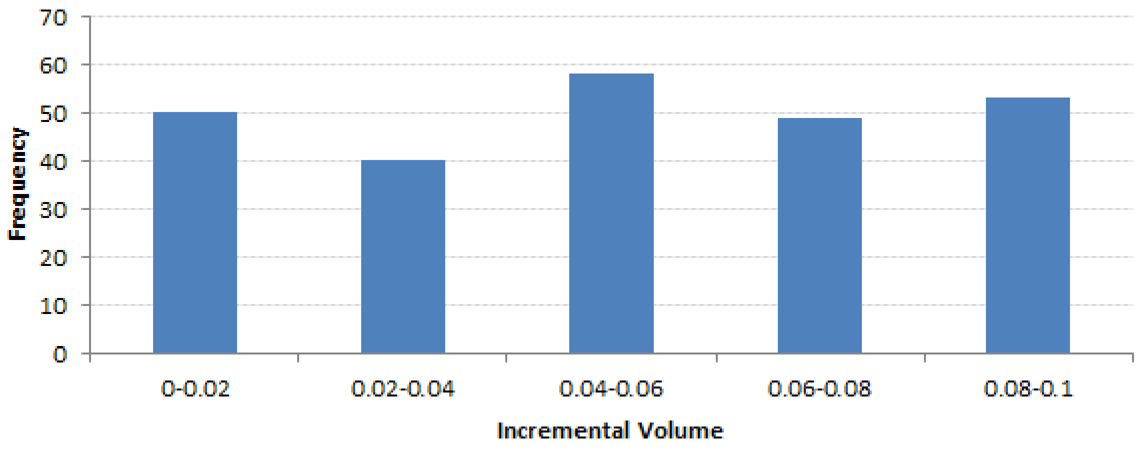
We will try to solve this case study both by a **business approach** and an **analytical approach**.

To solve the problem without applying any technique, let's analyze the frequency distributions.

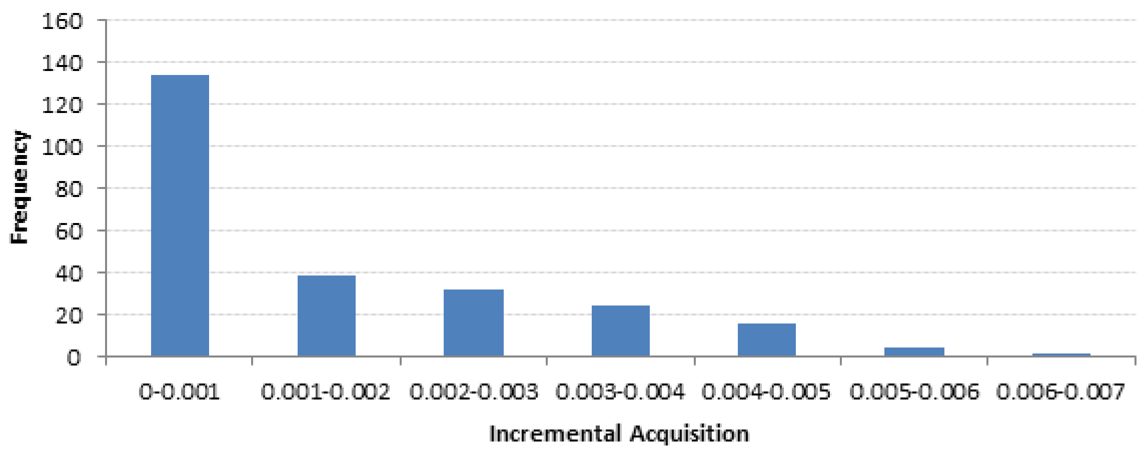
Profit Margin : Here is a distribution of profit margin for each of 250 products.



Let’s divide the products based on profit margin bands.
Low Profit Margin: *Less than 10% Profit*
Medium Profit Margin: *10% – 25% Profit*
High Profit Margin: *25% + Profit*
Incremental volume : Here is a distribution of incremental volume for each of 250 products.



Let’s categorize incremental volume bands:
Low Incremental Volume: *Less than 2%*
Medium Incremental Volume: *2% – 6%*
High Incremental Volume: *6% + Profit*
Incremental Acquisition : Here is a distribution of incremental acquisition for each of 250 products.



Finally, we should split incremental acquisition bands also:
Low Incremental Acquisition*: Less than 0.1% Profit*
Medium Incremental Acquisition*: 0.1% – 0.4% Profit*
High Incremental Acquisition*: 0.4% + Profit*

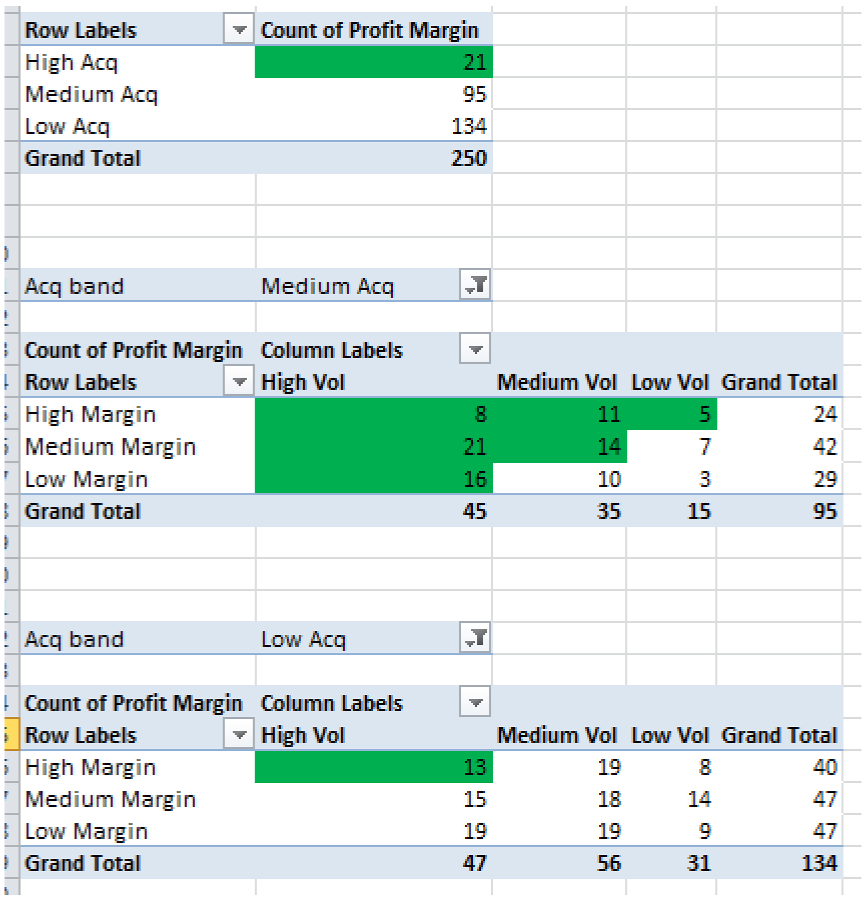
Let's discuss the **pricing strategy** now:

Until here, we have got three decision attributes for each product. The extreme of the pricing is -10% and 20%. We need a -10% for products which can give a big acquisition increment or high volume increment or both. For rest we need to increase the profit margin by increasing pricing by +20%.

Obviously, this is a super simplistic way to solve this problem, but we are on a track of getting low hanging fruits benefit.

Because the incremental acquisition has a mean at 0.4%, we know that if we decrease the cost of products with high acquisition incremental rate, we will have significant incremental overall sales, but quite less impact due to less profit margins.

For medium incremental acquisitions, we need to take decision on profit margins and incremental volumes. All the cells shaded in green are the ones that will be tagged for -10% and rest at 20% price increase. The decision here is purely intuitive taking into account the basic understanding of revenue drivers for which we have seen the distributions above.



Row Labels	Count of Profit Margin
High Acq	21
Medium Acq	95
Low Acq	134
Grand Total	250

Acq band: Medium Acq

Count of Profit Margin	Column Labels			
Row Labels	High Vol	Medium Vol	Low Vol	Grand Total
High Margin	8	11	5	24
Medium Margin	21	14	7	42
Low Margin	16	10	3	29
Grand Total	45	35	15	95

Acq band: Low Acq

Count of Profit Margin	Column Labels			
Row Labels	High Vol	Medium Vol	Low Vol	Grand Total
High Margin	13	19	8	40
Medium Margin	15	18	14	47
Low Margin	19	19	9	47
Grand Total	47	56	31	134

Now if we calculate the total profit earned, we see significant gains over and above the initial profit.

Total Profit earned : $267 (Increase of 62%)

9. Describe Your capstone project or In-house projects. Walkthrough the steps taken to build the model and how did you improvise it.

10. Practice case studies which tend to improve the sales of a product in the market/improve the profits earned.

11. Brush up your Python, SQL and Tableau Basics.

12. Which classification algorithm are you comfortable with. Please explain the flow of the algorithm with the help of a scenario or example.

13. What are the difficulties in working with regression models? What are their corrections?
Ans-
Multicollinearity occurs when independent variables in a regression model are correlated. This correlation is a problem because independent variables should be *independent*. If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results.
There are two basic kinds of multicollinearity:

1. Structural multicollinearity: This type occurs when we create a model term using other terms. In other words, it's a byproduct of the model that we specify rather than being present in the data itself. For example, if you square term X to model curvature, clearly there is a correlation between X and X2.
2. Data multicollinearity: This type of multicollinearity is present in the data itself rather than being an artifact of our model. Observational experiments are more likely to exhibit this kind of multicollinearity.

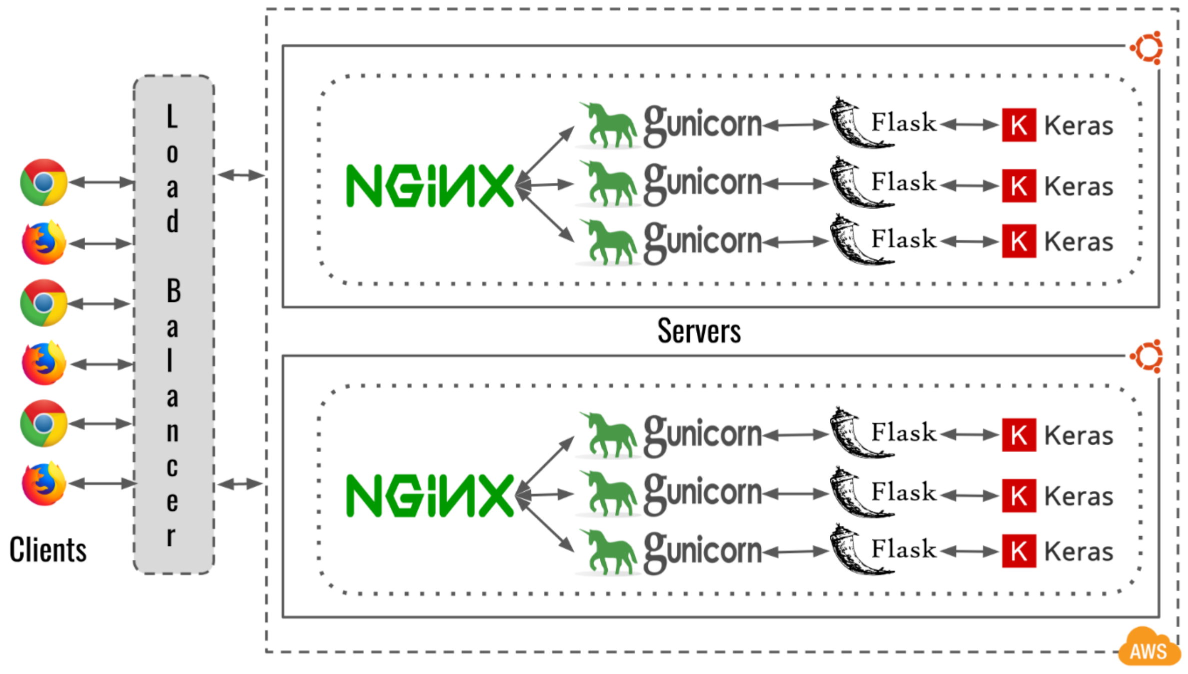
Multicollinearity causes the following two basic types of problems:
The coefficient estimates can swing wildly based on which other independent variables are in the model. The coefficients become very sensitive to small changes in the model.
Multicollinearity reduces the precision of the estimate coefficients, which weakens the statistical power of your regression model. You might not be able to trust the p-values to identify independent variables that are statistically significant

The need to reduce multicollinearity depends on its severity and your primary goal for your regression model. Keep the following three points in mind:
The severity of the problems increases with the degree of the multicollinearity. Therefore, if you have only moderate multicollinearity, you may not need to resolve it.

Multicollinearity affects only the specific independent variables that are correlated. Therefore, if multicollinearity is not present for the independent variables that you are particularly interested in, you may not need to resolve it. Suppose your model contains the experimental variables of interest and some control variables. If high multicollinearity exists for the control variables but not the experimental variables, then you can interpret the experimental variables without problems.

14. What is exploding gradients ?
Gradient is the direction and magnitude calculated during training of a neural network that is used to update the network weights in the right direction and by the right amount. "Exploding gradients are a problem where **large error gradients** accumulate and result in very large updates to neural network model weights during training." At an extreme, the values of weights can become so large as to overflow and result in NaN values.This has the effect of your model being unstable and unable to learn from your training data.

15. Can you give an example on how ML models are deployed?



Components

Let’s break down the above image that depicts the entire API workflow and understand every component.

1.Client: The client in the architecture can be any device or a third party application that tries to request the server hosting the architecture for model predictions. Example: Facebook trying to tag your face on a newly uploaded image.

2. Load Balancer: A load balancer tries to distribute the workload (requests) across multiple servers or instances in a cluster. The aim of the load balancer is to minimize the response time and maximize the throughput by avoiding the overload on any single resource. In the above image, the load balancer is public facing entity and distributes all the requests from the clients to multiple Ubuntu servers in the cluster.

3. Nginx: Nginx is an open-source web server but can also be used as a load balancer. Nginx has a reputation for its high performance and small memory footprint. It can thrive under heavy load by spawning worker processes, each of which can handle thousands of connections. In the image, nginx is local to one server or instance to handle all the requests from the public facing load balancer. An alternative to nginx is Apache HTTP Server.

3. Gunicorn: It is a Python Web Server Gateway Interface (WSGI) HTTP server. It is ported from Ruby’s Unicorn project. It is a pre-fork worker model, which means a master creates multiple forks which are called workers to handle the requests. Since Python is not multithreaded, we try to create multiple gunicorn workers which are individual processes that have their own memory allocations to compensate the parallelism for handling requests. Gunicorn works for various Python web frameworks and a well-known alternative is uWSGI.
4. Flask: It is a micro web framework written in Python. It helps us to develop application programming interface (API) or a web application that responds to the request. Other alternatives to Flask are Django, Pyramid, and web2py. An extension of Flask to add support for quickly building REST APIs is provided by Flask-RESTful.

5. Keras: It is an open-source neural network library written in Python. It has the capability to run on top of TensorFlow, CNTK, Theano or MXNet. There are plenty of alternatives to Keras: TensorFlow, Caffe2 (Caffe), CNTK, PyTorch, MXNet, Chainer, and Theano .
6. Cloud Platform: If there is one platform that intertwines all the above-mentioned components, then it is the cloud. It is one of the primary catalysts for the proliferation in the research of Artificial Intelligence, be it Computer Vision, Natural Language Processing, Machine Learning, Machine Translation, Robotics or Medical Imaging. Cloud has made computational resources accessible to a wider audience at a reasonable cost. Few of the well-known cloud web services are Amazon Web Services (AWS), Google Cloud and Microsoft Azure.

Architecture Setup
By now you should be familiar with the components mentioned in the previous section. In the following section, Let's understand the setup from an API perspective since this forms the base for a web application as well.
Note: This architecture setup will be based on Python.

Development Setup
Train the model: The first step is to train the model based on the use case using Keras or TensorFlow or PyTorch. Make sure you do this in a virtual environment, as it helps in isolating multiple Python environments and also it packs all the necessary dependencies into a separate folder.
Build the API: Once the model is good to go into an API, you can use Flask or Django to build them based on the requirement. Ideally, you have to build Restful APIs, since it helps in separating between the client and the server; improves visibility, reliability, and scalability; it is platform agnostic. Perform a thorough test to ensure the model responds with the correct predictions from the API.
Web Server: Now is the time to test the web server for the API that you have built. Gunicorn is a good choice if you have built the APIs using Flask.

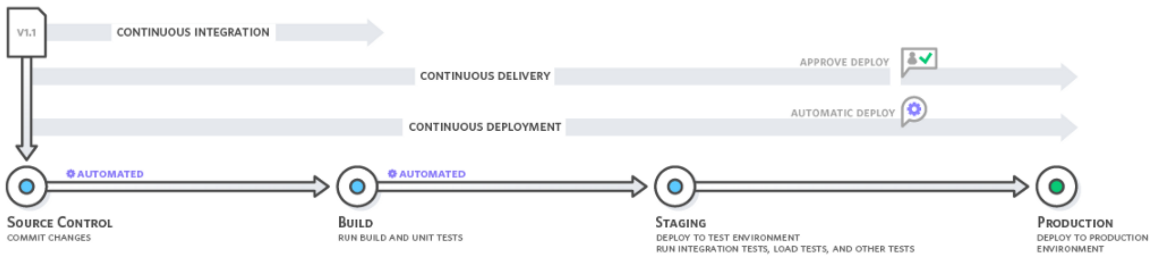
Load Balancer: You can configure nginx to handle all the test requests across all the gunicorn workers, where each worker has its own API with the DL model. Refer this resource to understand the setup between nginx and gunicorn.
Load / Performance Testing: Take a stab at Apache Jmeter, an open-source application designed to load test and measure performance. This will also help in understanding nginx load distribution. Alternative is Locust.

Production Setup
Cloud Platform: After you have chosen the cloud service, set up a machine or instance from a standard Ubuntu image (preferably the latest LTS version). The choice of CPU machine really depends on the DL model and the use case. Once the machine is running, setup nginx, Python virtual environment, install all the dependencies and copy the API. Finally, try running the API with the model (it might take a while to load all the model(s) based on the number of workers that are defined for gunicorn).
Custom API image: Ensure the API is working smoothly and then snapshot the instance to create the custom image that contains the API and model(s). Snapshot will preserve all the settings of the application. Reference: AWS, Google, and Azure.
Load Balancer: Create a load balancer from the cloud service, it can be either public or private based on the requirement. Reference: AWS, Google, and Azure.

A Cluster of instances: Use the previously created custom API image to launch a cluster of instances. Reference: AWS, Google, and Azure.
Load Balancer for the Cluster: Now, link the cluster of instances to a load balancer, this will ensure the load balancer to distribute work equally among all the instances. Reference: AWS, Google, and Azure.
Load/Performance Test: Just like the load/performance testing in the development, a similar procedure can be replicated in production, but now with millions of requests. Try breaking the architecture to check it's stability and reliability (not always advisable).
Wrap-up: Finally, if everything works as expected, you'll have your first production level DL architecture to serve millions of requests.

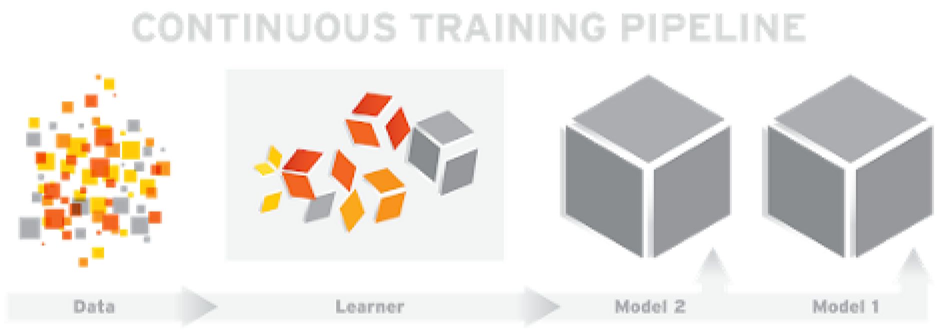
Additional Setup (Add-ons)

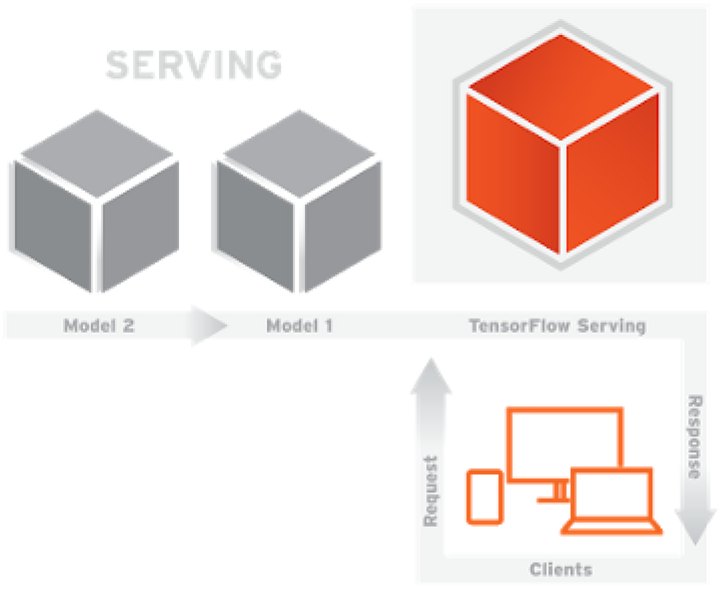
Apart from the usual setup, there are few other things to take care of to make the setup self-sustaining for the long run.
Auto-scaling: It is a feature in cloud service that helps in scaling up the instances in for the application based on the number of requests received. We can scale-out when there is a spike in the requests and scale-in when the requests have reduced. Reference: AWS, Google, and Azure.
Application updates: There comes a time when you have to update the application with a latest DL model or update the features of the application, but how to update all the instances without affecting the behavior of the application in production. Cloud services provide a way to perform this task in various ways and they can be very specific to a particular cloud service provider. Reference: AWS, Google, and Azure.
Continuous Integration: It refers to the build and unit testing stages of the software release process. Every revision that is committed triggers an automated build and test. This can be used to deploy latest versions of the models into production.



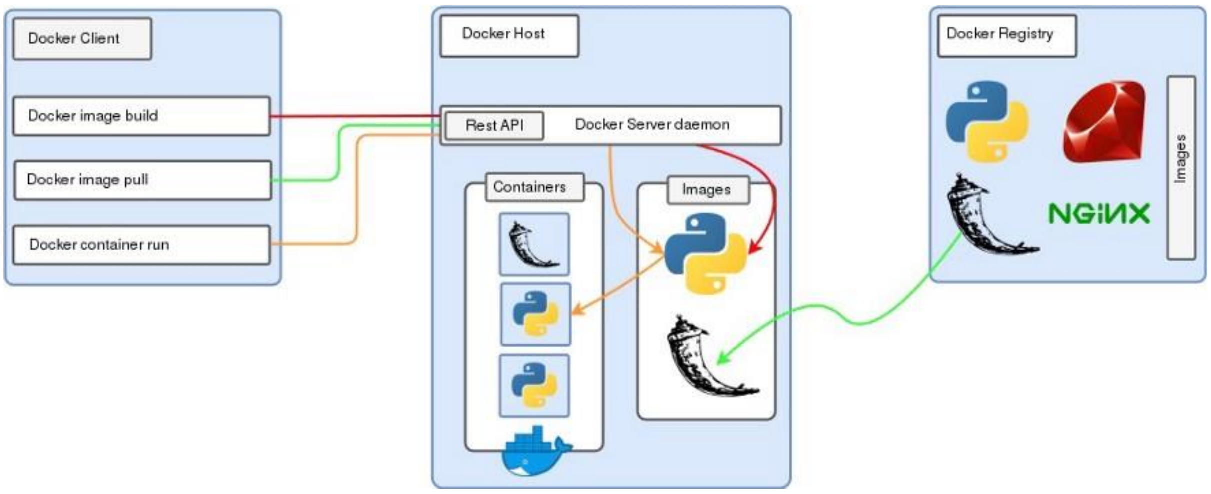
Alternate platforms

There are other systems that provide a structured way to deploy and serve models in the production and few such systems are as follows:
TensorFlow Serving: It is an open-source platform software library for serving machine learning models. It's primary objective based on the inference aspect of machine learning, taking trained models after training and managing their lifetimes. It has out-of-the-box support for TensorFlow models.

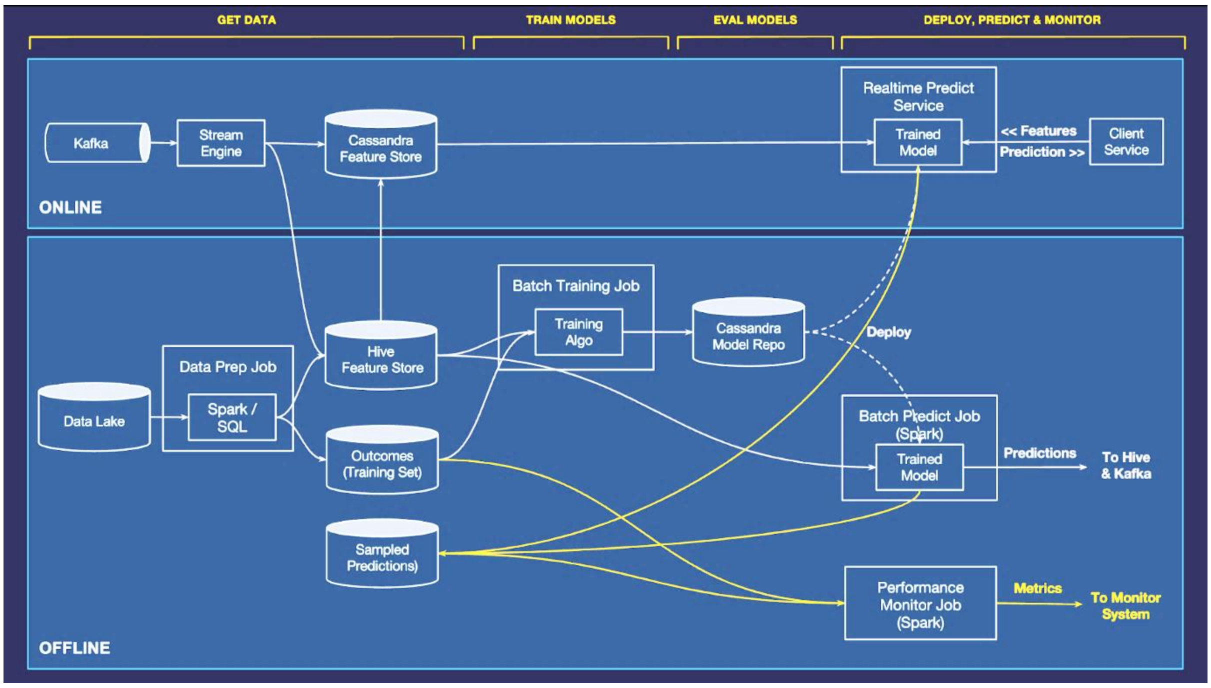




Docker: It is a container virtualization technology which behaves similarly to a light-weighted virtual machine. It provides a neat way to isolate an application with its dependencies for later use in any operating system. We can have multiple docker images with different applications running on the same instance but without sharing the same resources



Michelangelo: It is Uber's Machine Learning platform, which includes building, deploying and operating ML solutions at Uber's scale.


GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
Kafka
Stream Engine
Cassandra Feature Store
Realtime Predict Service
Trained Model
<< Features
Prediction >>
Client Service
ONLINE
Batch Training Job
Training Algo
Cassandra Model Repo
Deploy
Data Prep Job
Spark / SQL
Hive Feature Store
Data Lake
Outcomes (Training Set)
Batch Predict Job (Spark)
Trained Model
Predictions
To Hive & Kafka
Sampled Predictions
Performance Monitor Job (Spark)
Metrics
To Monitor System
OFFLINE