DevOps for Data Science:

The machine learning model built by a data scientist has to be deployed into production for usage.
When a new version of a model is available, then the old version in production has to be replaced with a new one.
This process of deploying machine learning models is very similar to the deployment of software code.
As a reason, DevOps practices and habits can be followed while deploying machine learning models into production.

Steps in Machine Learning
As you know, Machine Learning provides computers the ability to perform tasks such as classification and prediction.

The major steps involved in Machine Learning are:

Data Collection
Data Munging
Feature Engineering
Model Building
Testing and Validating Model
Running Models

Data Collection:

The first part of any Data Science project is Data Collection. It enables an organization to derive answers to specific questions.

The data can be collected from multiple sources such as Relational Databases, NoSQL Databases, Spreadsheets, Logfiles and so on.

Collecting high-quality data is essential for getting correct insights from the data.

Data Munging:

Raw data is transformed into another format using techniques like Filtering. This process of transformation is known as Data Munging

In reality, most of the raw data is redundant and can be structured, unstructured or semi-structured.

Feature Engineering:

Feature Engineering refers to the process of deriving new data features, from existing data, which are used further by a machine learning algorithm as input.

Feature Engineering is the most crucial step in Machine Learning.

Some of the tasks carried in feature engineering are:

Identification of correlated data columns. Highly correlated data columns do not add much to the predictive power of the generated model.
Creating new data columns from one or more existing columns.

Building Models:

Building a Model is the most significant step in machine learning.

You have to choose a machine learning algorithm and train the algorithm with extracted features dataset.

The training process involves initialization of random values, known as weights, for the chosen algorithm and then perform the predictions.

The predictions are compared with the known output.

The weights are then readjusted and above step is repeated until model gets better prediction ability.

Running ML Models:

Once models are validated, they can be used for prediction in two ways.
In the first case, the predict methods, associated with a model, can be called directly from a script and obtain the result.
Most of the times, the script is run on a terminal.
In the second case, the model is moved into production and users interact with it over the internet. In this case, predict methods of a model are not directly accessible.
Many data scientists are familiar only with the first case.

Environments in DevOps Cycle:

Software developers, in general, use three different environments:

development, staging, and production environments.

Development of code, required for building model, and unit tests happen in development environment.

In staging environment complete testing is performed in an integrated environment. Here model interaction with external systems such as databases, cache, and other application is tested.

Finally, the model is deployed into production environment. This is the environment exposed to users.

Deploying ML models in Production:
****

In general, data scientists use R/Python languages for building models and for consuming these models, software developers deploy them in a different stack environment.

Deploying ML Models in Production:
****

The two majorly followed approaches for deploying models in production are:

Rewrite the whole code of building and accessing model in language, supported by stack environment.
Develop APIs for accessing models.
The first approach is very tedious and time-consuming.

Alternatively, the second option is a reliable one. In this case, Model can be developed in language and stack environment can be supported by another language.

The front end application interacts with the model through an API.

Deploying Models with Automation:
****
Machine Learning models are generally used for either classifications or regression problems based on some input data.

When the input data to a model, deployed in production, changes a new model needs to be trained, built and deployed in production again.

The above process is iterative, and it is similar to software program development.

The process of deploying models frequently into production is known as Continuous Integration.

Deployment Practices:
****
The major practices to be followed for a deployment of machine learning model are:

Using a Version Control System for models
Perform Canary Deployment
Secure Models in Production
Monitor Performance of Models in Production

1.Using Version Control System:
****

To keep track of different versions of a model, you can use version control systems.

One of the popular version control system is Git.

Git is a distributed version control system.

Git is Open source and it is fast.

Canary Deployment
****
In a canary deployment, new changes to a model are exposed to a limited proportion of entire users.
This can be achieved when multiple application servers are used for a service. The new model is deployed to one of the existing application servers.
In this deployment, only a few users use the new model. The new model usage, it's error rate, and other metrics are measured to verify if any problems exist with a new model.
In case if load balancers are used to distribute load across servers, then load balancer can be configured to distribute traffic to each server.

Git allows multiple local branches that are entirely independent of each other.

Continuous Integration process can be automated using a popular tool – Jenkins.

Securing ML models in Production:
****

Models have to be secured in Production.

Models can be secured using access controls like authentication and authorization.

Models in production must be protected against different type of security attacks which compromise model's integrity or availability. This can be achieved by :

Reviewing the training data at regular intervals.
Reviewing test data and model predictions.
Performing random testing and reviewing the predictions.
You should also follow practices such as encryption, operations security and disaster recovery.

Performance Monitoring in Production:
****
The performance of models, deployed into production, must be monitored continuously to ensure that everything is working properly.

Consumption of various resources such as CPU, memory, disk, network I/O must be monitored for checking how efficiently the model is running.

Data Scientists must primarily check for drift i. e change in model input data. A drift indicates, model input data is not similar to one used in training, thus making the model out dated.

Monitoring further required for meeting SLA's agreed for the business.

Initialization of Git Repository in an Existing Directory
Create a folder SampleProject using mkdir command.

mkdir SampleProject

Move into SampleProject folder using cd command.

cd SampleProject

Create helloworld.py by clicking on below expression.

echo ″print('Hello World')″>helloworld.py
View the contents of the folder using ls command.
ls'

Now initialize a git repository with git command.
git init

Now run the command ls –la and observe if a new directory .git is created.

As of now no files in SampleProject folder are tracked yet.

Adding 'helloworld.py' file to Git for tracking:

In order to track helloworld.py, add the file to git using add command. It adds the file to staging area.

```
git add helloworld.py
```

Even now git does not keep tracking contents of helloworld.py file.
You can view the status using status command.

```
git status
```

The next step is to Commit the changes available in staging area. This can be achieved using commit method.

git commit –m "Initial Version"

View the Status again.

```
git status
```

The initial commit creates a master branch. You can view the list of all branches using 'branch' command.

```
git branch.
```

Creating a new branch
Let's create a new branch, new_branch using

Now view the list of all available branches using branch command. The active branch is prefixed with an asterisk symbol.

```
git branch
```

Now move to new_branch using checkout command

```
git checkout new_branch
```

View all branches and observe that now active branch is new_branch git branch

CONTINUE

Making Changes to 'helloworld.py' file

Append the below line to helloworld.py file.

echo ″print('Newly added line in new_branch')″>>helloworld.py

Now add the changes done to helloworld.py to git.

git add helloworld.py

Commit the changes using commit command

git commit –m ″Added a feature in new_branch″

Merge 'new_branch' with 'master'

Let's view the contents of helloworld.py file in new_branch. It contains two lines. cat helloworld.py

Let's move to master branch and view
git checkout master

Now view the contents of helloworld.py file in master. It contains just one line, indicating that master branch and new_branch are different. cat helloworld.py

Let's merge new_branch with master branch using merge command. git merge new_branch

View the contents of helloworld.py file again to confirm the merge. cat helloworld.py

After a successful merge, if no longer the new_branch is required, it can be deleted. git
branch –d new_branch

Run branch command to view the list of branches.

What is a Web Service?

A Web Service is a software program available to users over the internet.

A user invokes a web service by sending a web request and waits for its response.

The logic over here is that you make use of the widely used interface,

i.e. internet, to call or invoke the software program and also receive the response from the service, over the interface.

In this topic, you will hear about RESTful Web Services / API's which are invoked using HTTP protocols

Running a Model as a Service

To run a machine learning model as a service, the model must be wrapped in a program/API, which must be capable of invoking model functions like predict.

When a model is embedded in a program/API, it acts as a service. This service can be invoked through HTTP protocols.

When a model is moved into production, it's function can be invoked as a service.

Also, a service also acts as an abstraction to a machine learning model from users.

RESTful APIs for ML Model Services

A RESTful API invokes a machine learning model functionality, executes it and captures the response.

Machine Learning APIs are invoked using HTTP Requests, which send the inputs required for executing a model functionality.

A HTTP Request includes an URL and a HTTP method.

HTTP Methods:

The four HTTP methods allowed in a RESTful API are:

GET: GET method sends the required input data for the model, as a query string of the requested URL. It then processes the input data and returns a response.
POST: POST method sends the required input data for the model, through the HTTP message body. It further processes the data and returns a response.
PUT: PUT method is used generally to edit some content, which is already present at the server side.
DELETE: DELETE method is generally used to delete some data, available at the server side.

While dealing with machine learning models, you mostly deal with GET and POST methods.

The allowed HTTP methods are : GET, POST, PUT, and DELETE.

Using Plumber

Plumber is used to quickly deploy models, built using R language, as services.

A REST API can be easily created by decorating a R function with special comments.

API Definition of three examples, written in serviceAPI file, are shown below.

```
# serviceAPI.R

#* Echo back the input
#* @param msg The message to echo
#* @get /echo
function(msg=""){
  list(msg = paste0("The message is: '", msg, "'"))
}

#* Plot a histogram
#* @png
#* @get /plot
function(){
  rand <- rnorm(100)
  hist(rand)
}

#* Return the sum of two numbers
#* @param a The first number to add
#* @param b The second number to add
#* @post /sum
function(a, b){
  as.numeric(a) + as.numeric(b)
}
```

Using Flask

Flask is one of the Python frameworks used to quickly deploy models, built using python language, as services.

Best Practices for API Design

Some of the best practices to be followed for API design are listed below:

Use a RESTful interface because it is easy for developers to work with HTTP and JSON.

Use GET or POST methods to invoke machine learning models over HTTP.

Document the API using tools like Swagger.

Include the term api in the path of URLs. This distinguishes a standard HTTP request from an API call.

Also use version number in an API call.

Use API keys for controlling access to services

Installation
Let's get the installations done, prior to creating tasks.

Click on below given commands, one by one, to get required packages installed automatically:

pip3 install --user --upgrade setuptools

pip3 install --user numpy

pip3 install --user scikit-learn

pip3 install --user flask

pip3 install --user flask-restful

Generating a Model
Create a folder SampleProject using mkdir command.
mkdir SampleProject

Change the working directory to SampleProject using cd command. cd SampleProject

Create a empty folder models, for storing created models mkdir models

Create a file model_generator.py with below content.

```
echo "from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import pickle

iris = datasets.load_iris()
validation_size = 0.20
seed = 100
X_train, X_test, Y_train, Y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    test_size=validation_size,
                                                    random_state=seed)
```

```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)

with open('models/iris_classifier_model.pk', 'wb') as model_file:
    pickle.dump(knn, model_file)"">model_generator.py
```
Run the model_generator.py script. It creates the required model and stores in models folder. python3 model_generator.py

Developing a RESTful API using Flask
Add below api to file iris_classifier.py

```
echo "from flask import Flask, request
from flask_restful import Resource, Api
import pickle

app = Flask(__name__)

api = Api(app)


def classify(petal_len, petal_wd, sepal_len, sepal_wd):
    species = ['Iris-Setosa', 'Iris-Versicolour', 'Iris-Virginica']
    with open('models/iris_classifier_model.pk', 'rb') as model_file:
        model = pickle.load(model_file)

    species_class = int(model.predict([[petal_len, petal_wd, sepal_len, sepal_wd]])[0])
    return species[species_class]


class IrisPredict(Resource):

    def get(self):
        sl = float(request.args.get('sl'))
        sw = float(request.args.get('sw'))
        pl = float(request.args.get('pl'))
        pw = float(request.args.get('pw'))


        result = classify(sl, sw, pl, pw)

        return {'sepal_length':sl,
                'sepal_width':sw,
                'petal_length':pl,
                'petal_width':pw,
                'species':result}
```

api.add_resource(IrisPredict, '/classify/')″>iris_classifier.py

Running Model as a service
Set FLASK_APP environment variable export FLASK_APP=iris_classifier.py

Set other required environment variables export LC_ALL=C.UTF-8

export LANG=C.UTF-8

Start the Server , which exposes the api as a service. python3 -m flask run --host=0.0.0.0 --port=8000

Open a new terminal and try access the api with classify end point. curl ″http://0.0.0.0:8000/classify/?sl=5.1&sw=3.5&pl=1.4&pw=0.3″

What are Containers?

A container is a standard unit of software that packages up the code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. – www.docker.com

A container consists of an application, all of its dependencies, libraries and other configuration files required for running it.

Applications running in containers access shared operating system kernel.

Containers are reliable when moving software from one environment to another such as movement from a staging environment to production.

Why Containers?

Containers are different from virtual machines.

Application Scaling
Application Scaling refers to the alteration of computing resources required by the application, as load changes.

Applications can be scaled vertically or horizontally.

Vertical Scaling: It increases computing resources by adding more resources, directly, to an existing server.

Horizontal Scaling: It increases computing resources by adding more physical machine or servers. The application runs on multiple servers, present in the cluster.

ntroduction to Docker
Docker is the widely used containerization tool.

The three main components of Docker are:

Docker Engine: An application which runs Docker images and also contains a command line utility.
Docker Client: It contains tools for building and running Docker images. It helps in interacting with other Docker components.
Docker Registry: It stores Docker images. Docker Hub is a public registry, which can be used by anyone to share and access public Docker images.

Docker Images
A Docker Image is a binary file, containing details of resources required to execute an application.

A Docker Image can be obtained either by building it using a Dockerfile or from a Docker Registry.

A Dockerfile is a configuration file that specifies the components to be included in an image.

A Dockerfile also contains commands, which are invoked when Docker Image gets executed.

Docker Registry
Docker Registry: Docker Registry is a storage and delivery system for named Docker Images.

It can be taught as a collection of repositories identified by a name.

A Repository is just a collection of different versions of a single Docker Image.

Docker Images can be stored either in a public registry like Docker Hub or a private registry.

pull and push commands are used to retrieve and store an image from a Docker Registry.

Running a ML Service in a Docker container:

Need for Running Multiple Instances

ntroduction to Kubernetes
Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. – https://kubernetes.io

A Kubernetes cluster has two types of nodes, masters and minions.

Master node is a single node that manages the entire cluster. It coordinates activities like

scheduling applications, maintaining applications, scaling applications and so on.

Minions are the virtual machines that run containers having machine learning service. They serve as worker machines.

Each minion node contains a Kubelet, the agent which manages the node and communicates with master.

Creating a Kubernetes Cluster:

A Kubernetes cluster can be deployed either on a physical or a virtual machine.

In this course, you will see the creation of Kubernetes cluster using Minikube.

Installation of Minikube is different for different operating systems.

You can interact with the cluster using Minikube command line interface, kubectl.

After successfully installing Minikube you can start a kubernetes cluster using below shown command.

```
$ minikube start
```