

# Project Report

## IDS 594: Machine Learning Deployment

### Estimating Prices of Used Cars

Project by:

Aditya Gajula (660252623)

Niharika Yogesh Apte (676583822)

Krishangi Deka (664357494)

Dhruv Dhami (665626746)

Mehul Sharma (668967914)

Professor Theja Tulabandhula

Date: October 14, 2020

#### Important Links for submission:

GitHub Repository: <https://github.com/krishangi-deka/carprice>

YouTube Video Presentation: <https://www.youtube.com/watch?v=8anY4NGTcBA&feature=youtu.be>

Webpage Link: <http://ec2-54-163-16-187.compute-1.amazonaws.com:8501/>

## **ABSTRACT**

The purpose of this project is to show how to instantiate and execute a project using the AWS EC2 structure and Streamlit templates and to deploy a machine learning model which is used to estimate the prices of used cars.

## **1. INTRODUCTION**

This system provides users with a website that easily predicts the used car prices. Different python packages have been used to develop the ML model. Most importantly, we have used Amazon Elastic Compute Cloud (Amazon EC2) as the web service through which we have deployed the ML model. We have used Streamlit as our app framework. We will be explaining our choice further in this report. This report aims to provide a detailed look at our project progress and the resulting project including the packages, code and the analysis we applied on it.

### **1.1. Background**

We started this project trying to apply the knowledge acquired in the course- Machine Learning Deployment (Fall 2020). Our project covers most of the concepts that we learnt in this course such as the deployment of a Machine Learning model along with exploring and choosing the best service to deploy the model. We previously planned to work with Lambda function but after thorough research found out that the runtime for lambda function is just 900 seconds after which you will have to re-run the app which is not the case with EC2. Also, Lambda functions do not save the state which EC2 provides. Hence, we chose to proceed with EC2.

The disadvantage of using EC2 is that they require security updates and patches which are not required in lambda functions.

### **1.2. Project Brief**

The main goal of this project is to deploy a Web app that estimates used car prices based on the inputs provided. The project provides an easy user interface for the users to select their inputs by specifying the numeric values and the estimated price of the used car is displayed for the specifications given. The continuous variables that we have considered to determine the car prices are max mileage covered by the car, city mileage, highways MPG and car type and model year are our discrete variables.

### **1.3. Objectives of the project**

- To estimate the prices of used cars so that users can determine how much their used car would be sold for.
- Deploy the machine learning model through a cost-free web service.
- Making the system responsive and user-friendly.

## 1.4. Workflow

- Build model
- Host within a source code repository system(GitHub)
- Build/create an EC2 instance
- Add custom TCP ports
- Generate private key-pair (.pem file)
- Connect to instance using Putty (.ppk file)
- Connect to Linux instance: Configure the host settings and authorize instance
- Run Linux instance on cloud (Should run automatically)
- Install python and all libraries and dependencies
- Transfer python script and all data files to EC2 Linux instance.
- Run the app hosted on Streamlit.

## 1.5. Scope of the project

The project is helpful to model the price of used cars with the available independent variables. It will be used by the individuals to understand how exactly the prices vary with the independent variables.

### *Model Specifications:*

Linear regression is an algorithm that uses variables (or features for machine learning) to predict a quantitative variable (the target variable). More specifically it tries to fit a straight line through the data that best represents the relationship between the predictor variables and the target variable. Linear regression model was used to calculate the coefficients and the value of Target variable ('your car is worth approximately') is obtained from the predictor variables.

The model involved choosing the right set of predictor variables which are significant and can be used to estimate the approximate price of a used car. How well these variables describe the price of a used car, based on various market surveys, Max mileage (Miles travelled), City MPG, Highway MPG, Model year (year car was launched), were used as the output determining features.

As developers we must make sure that:

- We have all the necessary information related to the car prices(data)
- Make use of a web framework which could help us deploy the app in a more systematic way.

- Maintain the color scheme and style for the web app to make it look appealing, make it as user-friendly as possible.

### *Implementation and maintenance*

This is extremely important, since our web app has chances to break down. Making sure that our web app has less probability of being unavailable due to technical reasons.

### *Scope Planning*

Our system included planning, design, development and testing. We made sure that we will meet or exceed the software standards and also any additional requirements established by clients.

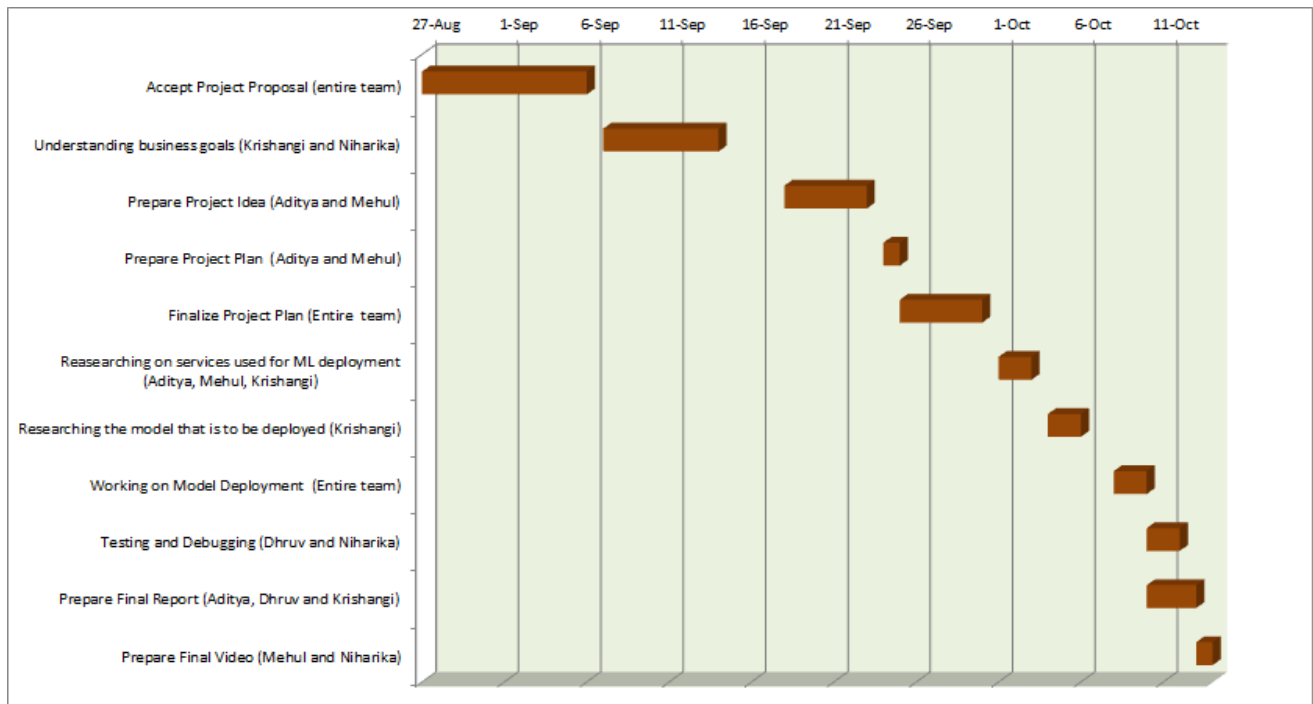
### *Creating the System*

We first planned and finalized the service which we would be using to deploy the model. We then decided the topic and the business problem that we would be solving for this project. We researched about various topics and decided to go with estimating car prices for used cars. The web app framework to be used to deploy this model was then selected. We were initially confused between Flask and Streamlit, but decided to go ahead with Streamlit as we wanted to explore this framework and also wanted a simpler and less complicated front end. The main task was learning how to deploy this framework and the model on EC2. After completing the deployment, we tested the system and improved on the front end layout to make it more aesthetic.

### *Scope Verification*

We distributed responsibilities amongst the members of our team on the basis of researching on EC2 deployment, writing the Streamlit code for deployment, testing and documentation. We distributed tasks in such a way that every member could contribute to the main task, which was the ML deployment on EC2. We made sure that each member stuck to deadlines and we followed up with each other through online zoom meetings. In this way we could make sure that the work is equally distributed and each of us had an opportunity to learn and explore.

*Gantt Chart for task distribution*



## 2. REQUIREMENT SPECIFICATIONS

### *Hardware:*

- Monitor/ Desktop

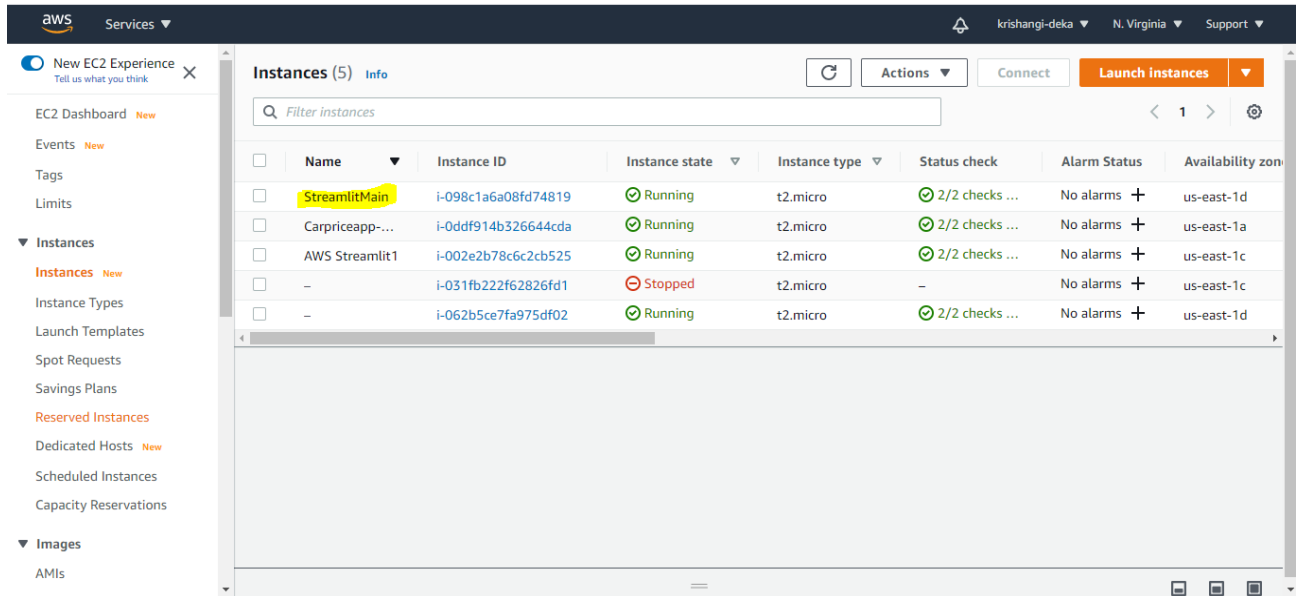
### *Software:*

- Web service : AWS EC2 (Deep Learning AMI (Ubuntu 16.04) Version 35.0)
- Operating System : Windows 10/Linux
- Programming Language : Python
- Web App Framework: Streamlit
- Other soft wares: Putty and Puttygen ( for SSH)

### 3. SCREENSHOTS

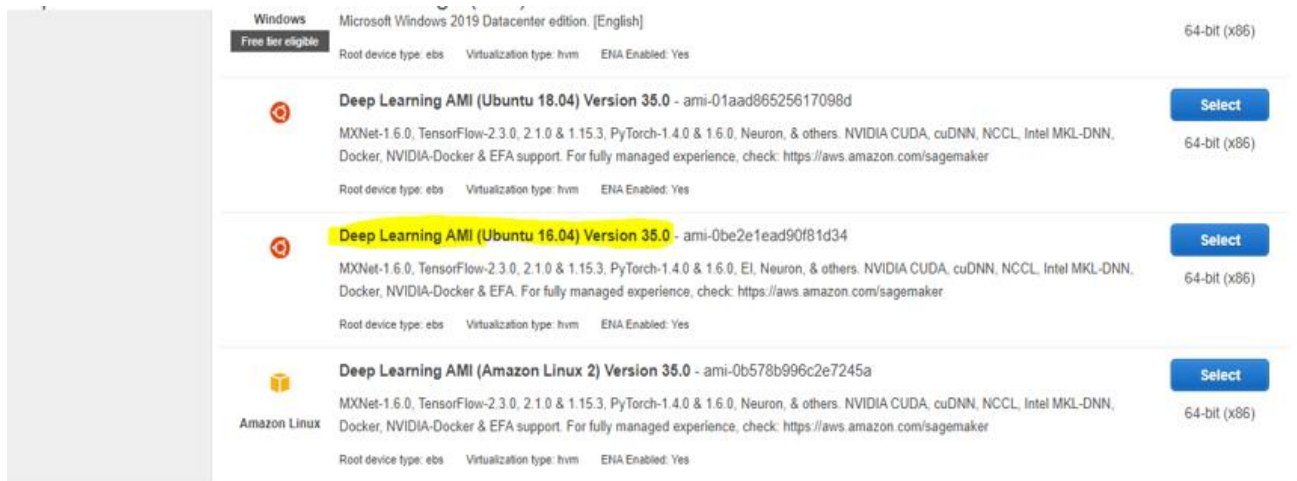
#### AWS EC2 setup:

##### Step 1. Create Instance



##### Step 2. Choose an Amazon Machine Image

Here we have chosen the Deep learning AMI (free tier version)



### Step 3. Configure Security Group

#### Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

**Assign a security group:** ☒ Create a **new** security group  
☐ Select an **existing** security group

**Security group name:**

**Description:**

Type <sup>i</sup>	Protocol <sup>i</sup>	Port Range <sup>i</sup>	Source <sup>i</sup>	Description <sup>i</sup>	
SSH <sup>v</sup>	TCP	22	Custom <sup>v</sup> 0.0.0.0/0	e.g. SSH for Admin Desktop	<sup>x</sup>
Custom TCP Rule <sup>v</sup>	TCP	8501	Anywhere <sup>v</sup> 0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop	<sup>x</sup>

### Step 4. Download pem key pair file

aws Services <sup>v</sup>

krishangi-deka <sup>v</sup> N. Virginia <sup>v</sup> Support <sup>v</sup>

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

#### Step 7: Review Instance Launch

t2.micro Variable 1

**Security Groups**

Security group name	Description
launch-wizard-	launch-wizard-

Type <sup>i</sup>	Protocol
SSH	TCP
Custom TCP Rule	TCP
Custom TCP Rule	TCP

**Instance Details**

**Storage**

**Tags**

**Select an existing key pair or create a new key pair** <sup>x</sup>

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

**Select a key pair**

**awsstreamit12** <sup>v</sup>

☐ I acknowledge that I have access to the selected private key file (awsstreamit12.pem), and that without this file, I won't be able to log into my instance.

Step 5. Note the DNS address and make sure the instance is running.

EC2 > Instances > i-098c1a6a08fd74819

**Instance summary for i-098c1a6a08fd74819 (StreamlitMain)** Info

Updated less than a minute ago

Refresh Connect Actions

<b>Instance ID</b> i-098c1a6a08fd74819 (StreamlitMain)	<b>Public IPv4 address</b> 54.163.16.187   <a href="#">open address</a>	<b>Private IPv4 addresses</b> 172.31.29.216
<b>Instance state</b> Running	<b>Public IPv4 DNS</b> ec2-54-163-16-187.compute-1.amazonaws.com   <a href="#">open address</a>	<b>Private IPv4 DNS</b> ip-172-31-29-216.ec2.internal
<b>Instance type</b> t2.micro	<b>Elastic IP addresses</b> -	<b>VPC ID</b> vpc-5db77720
<b>IAM Role</b> -	<b>Subnet ID</b> subnet-c8c88c85	

## PUTTYGEN and PUTTY setup (for connecting AWS instance):

Step 1. Save the .pem file as .ppk file through Puttygen.

**Putty Key Generator**

File Key Conversions Help

**Key**

Public key for pasting into OpenSSH authorized\_keys file:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCA  
+AXuOL27zFRVDbAHFPdugAFEDZbYgGdOcH/nVQJ5fp4582QHo1TOytlz  
+3249o0velLcpFaUWQfn7N03BgTQqHdGtWBKrc8SgwyRLdCauzpwwW8LbnbA  
YyukR/By3MmeJ185/QK5eJHxpVF7LyoFazJbGvmlRp7vKfHsCezPLswy9eb9TGSnY  
Ypg5yPoGByeQdAlgg
```

Key fingerprint: ssh-rsa 2048 12:9f:af:b5:21:dc:4b:91:a4:3a:9a:b8:12:46:5c:d4

Key comment: imported-openssh-key

Key passphrase:

Confirm passphrase:

**Actions**

Generate a public/private key pair **Generate**

Load an existing private key file **Load**

Save the generated key **Save public key** **Save private key**

**Parameters**

Type of key to generate:  
☒ RSA ☐ DSA ☐ ECDSA ☐ Ed25519 ☐ SSH-1 (RSA)

Number of bits in a generated key: 2048

**Save private key as:**

< Desktop > AWS keys

Search AWS keys

Organize New folder

Name	Date modified	Type
awsstream	10/9/2020 8:06 PM	PuTT
awsstream12	10/9/2020 8:15 PM	PuTT
awsstreamlit	10/9/2020 7:56 PM	PuTT

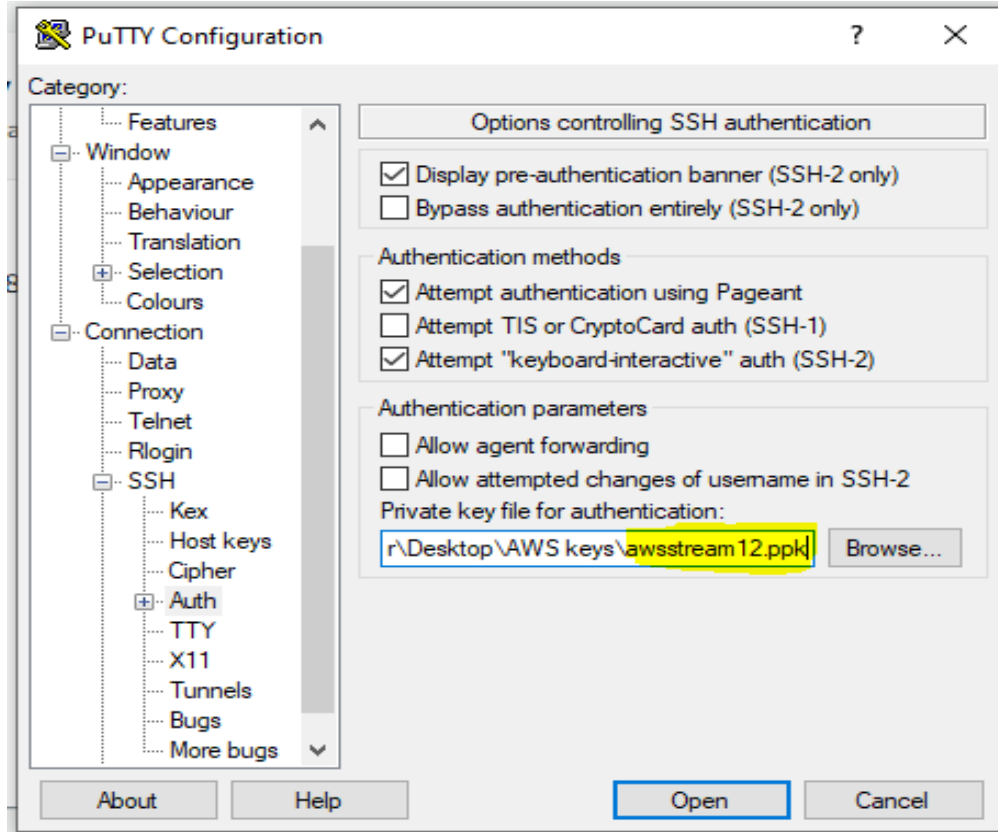
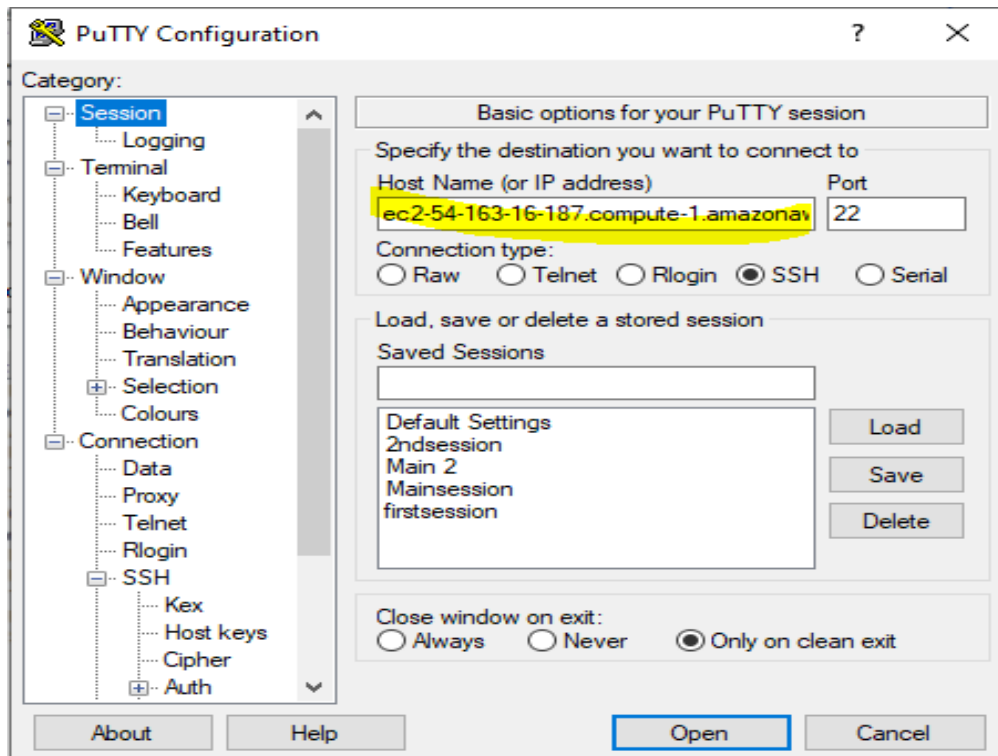
File name:

Save as type: PuTTY Private Key Files (\*.ppk)

**Save** **Cancel**



Step 2. Load the DNS address in the host name and the key in Putty.



### Step 3. Launch the Putty session and enter login details

```

PuTTY (inactive)
login as: ubuntu
Authenticating with public key "imported-openssh-key"
=====
      _ _ _ _ _
     _ _ _ _ _ /  Deep Learning AMI (Ubuntu 16.04) Version 35.0
    _ _ _ _ _
=====

Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-1114-aws x86_64v)

Please use one of the following commands to start the required environment with
the framework of your choice:
for MXNet 1.6 (+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) _____
    source activate mxnet_p36
for MXNet 1.6 (+Keras2) with Python2 (CUDA 10.1 and Intel MKL-DNN) _____
    source activate mxnet_p27
for MXNet 1.7 (+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) _____
    source activate mxnet_latest_p37
for MXNet(+Amazon Elastic Inference) with Python3 _____
    source activate amazonai_mxnet_p36
for MXNet(+Amazon Elastic Inference) with Python2 _____
    source activate amazonai_mxnet_p27
for MXNet(+AWS Neuron) with Python3 _____
    source activate aws_neuron_mxnet_p36
for TensorFlow(+Keras2) with Python3 (CUDA 10.0 and Intel MKL-DNN) _____
    source activate tensorflow_p36
for TensorFlow(+Keras2) with Python2 (CUDA 10.0 and Intel MKL-DNN) _____
    source activate tensorflow_p27
for TensorFlow 2(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) _____
    source activate tensorflow2_p36
for TensorFlow 2(+Keras2) with Python2 (CUDA 10.1 and Intel MKL-DNN) _____
    source activate tensorflow2_p27
for TensorFlow 2.3 with Python3 (CUDA 10.2 and Intel MKL-DNN) _____
    source activate tensorflow2_latest_p37
for Tensorflow(+Amazon Elastic Inference) with Python2 _____
    source activate amazonai_tensorflow_p27
for Tensorflow(+Amazon Elastic Inference) with Python3 _____
    source activate amazonai_tensorflow_p36
for Tensorflow 2(+Amazon Elastic Inference) with Python2 _____
    source activate amazonai_tensorflow2_p27
for Tensorflow 2(+Amazon Elastic Inference) with Python3 _____
    source activate amazonai_tensorflow2_p36
for Tensorflow(+AWS Neuron) with Python3 _____

```

Enter commands to deploy the model through Streamlit.

```

ubuntu@ip-172-31-29-216:~$ git clone https://github.com/krishangi-deka/carprice
fatal: destination path 'carprice' already exists and is not an empty directory.
ubuntu@ip-172-31-29-216:~$ sudo rm -r carprice
ubuntu@ip-172-31-29-216:~$ ls
carprice
ubuntu@ip-172-31-29-216:~$ cd carprice
ubuntu@ip-172-31-29-216:~/carprice$ sudo docker image build -t streamlit:app .
ubuntu@ip-172-31-29-216:~/carprice$ sudo docker container run -p 8501:8501 -d streamlit:app
82fa8158d76298b5296dd22ece7eec42f84c23edf08d791b7288468e97c5463b
ubuntu@ip-172-31-29-216:~/carprice$

```

#### 4. LESSONS LEARNED

- Streamlit is not always an easy framework to deploy through AWS EC2 as it is a separate module which requires installation and configuration which we couldn't easily install on the virtual server and faced many issues and bugs due to this.
- We chose to systematically run all the commands through a Docker file which could easily install all modules needed for the deployment which solved most of the issues.

#### 5. CONCLUSION

Elastic Cloud Compute is an easy to use service that lets you create and delete virtual machines in the cloud easily. We basically ran our Streamlit web app on an EC2 instance and created a web page which could solve our business solution in an easy and aesthetic way. Hence through this project we learnt that creating web apps is an easy task, but for the world to see your web app you need a web service. Without these web services, creating machine learning models and converting them to solve business solutions would be an impossible task.

#### REFERENCES

1. <https://aws.amazon.com/blogs/compute/continuous-deployment-to-amazon-ecs-using-aws-codepipeline-aws-codebuild-amazon-ecr-and-aws-cloudformation/>
2. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html>
3. <https://github.com/smeetvikani/Used-Car-Price-Predictor-Model>
4. [https://github.com/hamiltonchangcodes/Used\\_Car\\_Linear\\_Regression\\_Prediction](https://github.com/hamiltonchangcodes/Used_Car_Linear_Regression_Prediction)