

PHASE 5

ABSTRACT

Phase 5: Deployment and Productionization

Soumen Chatterjee

Phase 5 – 06-Mar-2022

- At this stage we will work on **Deployment and Productionization of the best model**

There are different approaches of deploying a model – these are: Locally using scikit learn/ AWS/Heroku/GCP/Azure etc.

For this particular project we tried

1. **Deploying locally:** This is the easiest way of testing like how an end user will see the model at his/her end.
This way is good to test the model from end user prospective, but in live scenario we will need to send a link to platform which is always available and user can check the query at any time. In this case there is no such option available
2. **Deploying at AWS:** This approach is something actually being used in live environment. This is one of the ways used in live environment other ways are Heroku, GCP, Azure.
As we have used the free tier resources available, so the resources available is very limited. And for a model which is heavy in nature might get timeout

Deploying model in AWS is a bit easy compared to others (though this statement varies based on users)

To deploy the model we have used –

Flask: To create the user interface & run the model

Joblib: Used this library to zip the trained model so that we can use the same in app

Saving the Logistic Regression model as Pickle file for Production purpose

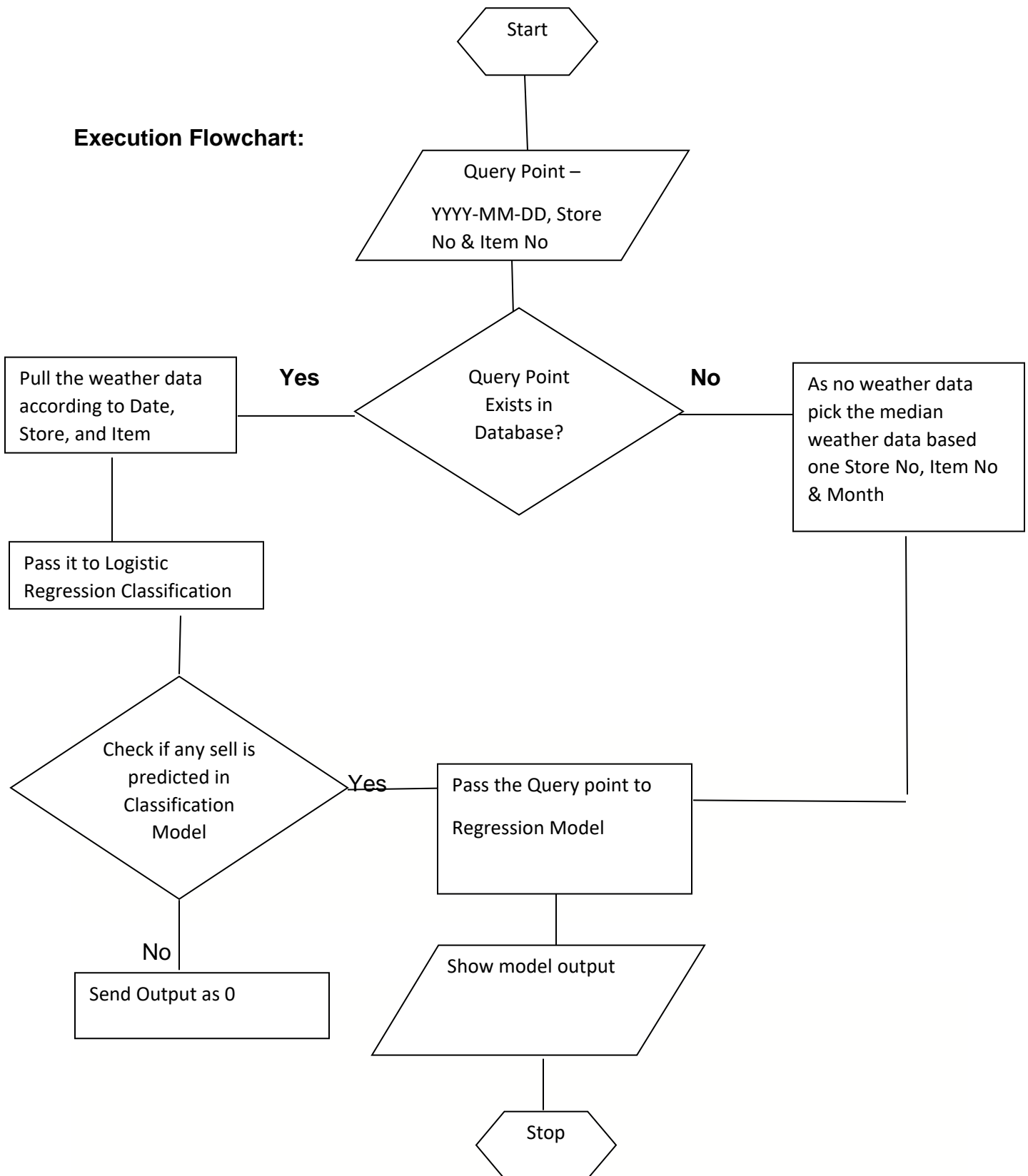
```
[ ] from joblib import dump, load
    dump(model_cls, '/content/drive/MyDrive/Project_TS_Walmart/Phase 2/Data/cls_LogisticRegrsn.pkl')

['/content/drive/MyDrive/Project_TS_Walmart/Phase 2/Data/cls_LogisticRegrsn.pkl']
```

Loading the .pkl file

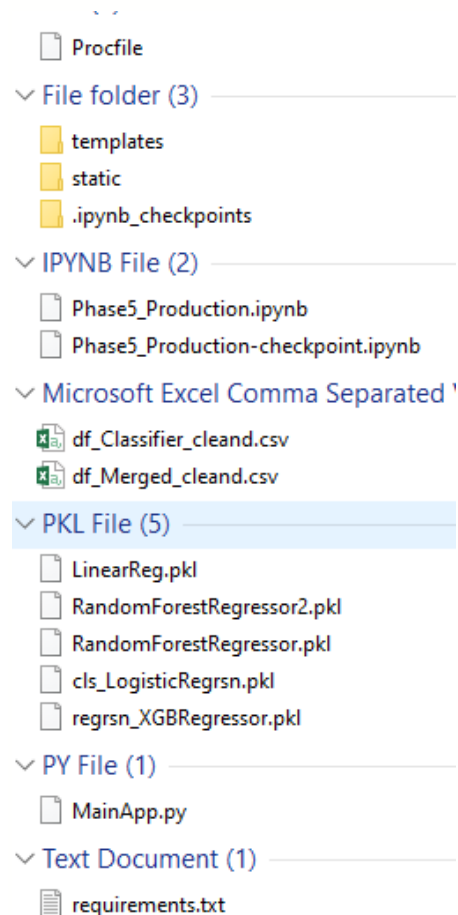
```
# Our best model is XGBRegressor, then Random Forest.
#But for both these Model- while loading pkl file we faced error in terms of Parameter so used Linear Regression
reg_RF = joblib.load('LinearReg.pkl')
predictionReg = reg_RF.predict(QueryPoint)
FinalPrediction = predictionReg
```

Execution Flowchart:



Below is the folder structure, which contains -

1. **"MainApp.py"** : has all the codes to read through the query point from user interface (created by Flask API)
2. **"df_Classifier_cleand.csv"**, **"df_Merged_cleand.csv"**: Cleaned merged data created after EDA
3. **".pkl"** files: For all the models we created at Phase 4 modelling stage. Linear Regression classification model (LinearReg.pkl) will be used for sure, and any one of the regression models will be used.
4. **"static" folder**: Holds the images, static files if any
5. **"templates" folder**: Holds the .html files which are used for user interfaces
 - a) **"querypage.html"**: Is the landing page where user will input the Date, Store No, Item no as Query point
 - b) **"outputpage.html"**: Once the above is submitted **"outputpage.html"** will pop up with the output
 - c) **"close.html"**: And lastly "close.html" will be called at the end



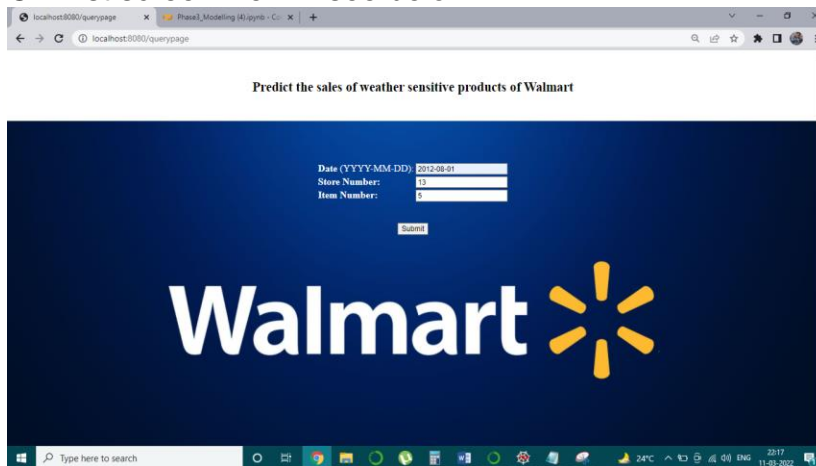
Deploying and checking the Model Locally:

1. First we will need to run the “app.py” (a code is there to load the corresponding Model’s .pkl file)
2. Then our system will act as server, based on the port no given a link will be generated. Something like below –

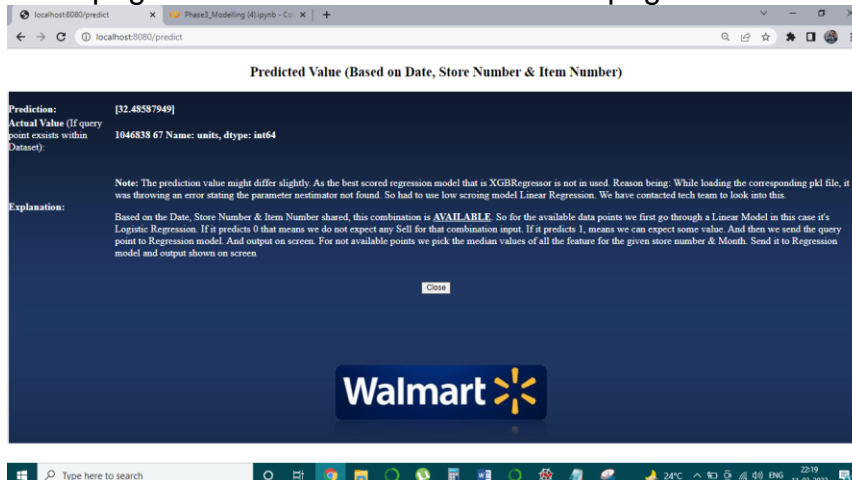
```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=8080)
```

```
* Running on http://0.0.0.0:8080/ (Press CTRL+C  
to quit)
```

3. Use the link: <http://localhost:8080/querypage> (change the yellow part as per the port no given)
4. On first screen we will see below –



3. Need to provide the query point details date, store no, item no
4. Once page is submitted we will see below page –



Deploying the Model at AWS:

Steps:

1. Build a model on our local system and store the model and other key model related variables in .pkl files
2. Launch a micro instance on AWS.
3. Connect to the AWS box [ssh]
4. Move the files to an AWS EC2 instance/box [scp]
5. Install all packages needed on the AWS box.
6. Run app.py on the AWS box.
7. Check the output in the browser.

Packages needed:

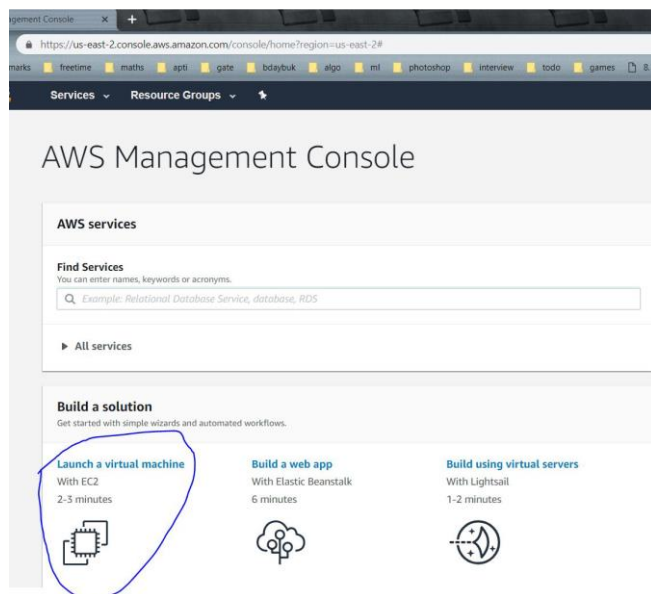
1. pip3
2. pandas
3. numpy
4. sklearn
5. beautifulsoup4
6. lxml
7. flask
8. re

Launch a micro instance on AWS.

Creating an instance:

1. Create an AWS account <https://aws.amazon.com> ,
<https://portal.aws.amazon.com/billing/signup#/start>
2. Login: <https://console.aws.amazon.com>

After login:




Launch the EC2 instance


3. Choose the ubuntu free tire

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


Step 1: Choose an Amazon Machine Image (AMI) Cancel and E

**Red Hat**
Free tier eligible

Red Hat Enterprise Linux version 7.6 (HVM), EBS General Purpose (SSD) Volume Type
Root device type: ebs Virtualization type: hvm

**SUSE Linux**
Free tier eligible

SUSE Linux Enterprise Server 15 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.
Root device type: ebs Virtualization type: hvm

**Ubuntu Server 18.04 LTS (HVM), SSD Volume Type**
Free tier eligible

ami-0c55b159cbf1f0 (64-bit x86) / ami-0f2057f28f0a44d06 (64-bit Arm)
Ubuntu Server 18.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
Root device type: ebs Virtualization type: hvm

64-bit (x86)
64-bit (Arm)

Select

64-bit (x86)
64-bit (Arm)

Select

4. Choose t2.micro free tier eligible & then Click on review and launch

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more about instance types and how they can meet your computing needs.](#)

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)


	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
<input type="checkbox"/>	General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

AMI Details Edit AMI

**Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0c55b159cbf1f0**
Free tier eligible
Ubuntu Server 18.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
Root Device Type: ebs Virtualization type: hvm

Instance Type Edit instance type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

Security Groups Edit security groups

Security group name: launch-wizard-2
Description: launch-wizard-2 created 2019-03-23T17:21:15.794+05:30

Type	Protocol	Port Range	Source	Description
This security group has no rules				

Cancel Previous Launch

5. Click on “Download Key Pair” and save the .pem file then click on “Launch Instance”

Select an existing key pair or create a new key pair ✕

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.](#)

Create a new key pair ▼

Key pair name
my_key_pair

Download Key Pair

💬 You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel Launch Instances

6. You will see this screen, you have successfully launched the an EC2 instance, now we need to Launch an flask api in it

aws Services Resource Groups

EC2 Dashboard

Launch Instance Connect Actions

Filter by tags and attributes or search by keyword

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 Public IP
	i-06fe95c371927683f	t2.micro	us-east-2b	running	2/2 checks passed	None	i-06fe95c371927683f.us-east-2.elb.amazonaws.com	13.59.191.237	

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

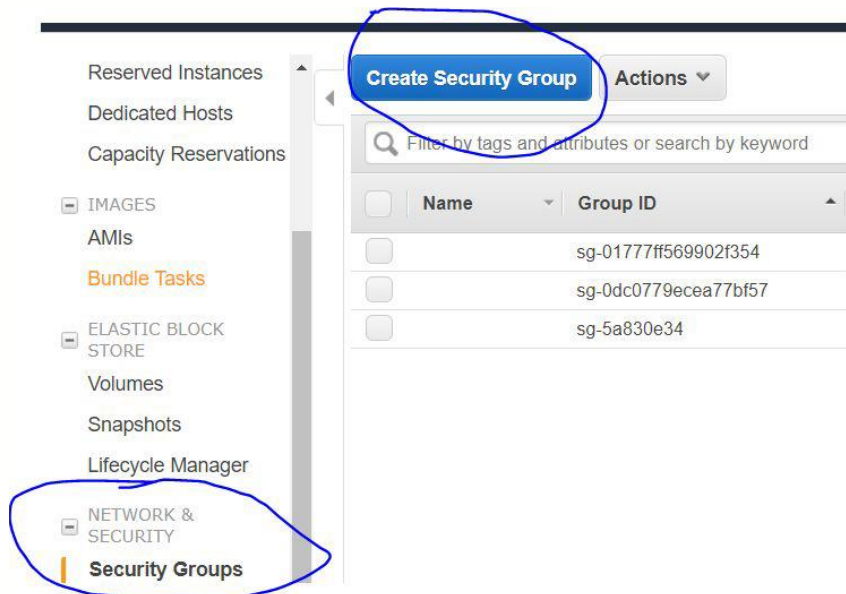
Capacity Reservations

IMAGES

AMIs

Run the Task

7. Select the “Network & security” -> Security groups and then click “Create Security Group”



The 'Create Security Group' dialog box is shown. It contains the following fields and options:



- Security group name:** anywhere
- Description:** anywhere
- VPC:** vpc-26713942 (default)
- Security group rules:**
 - Inbound/Outbound:** Inbound (selected)
 - Type:** All traffic
 - Protocol:** All
 - Port Range:** 0 - 65535
 - Source:** Anywhere (0.0.0.0/0, ::/0)
- Buttons:** Add Rule, Cancel, Create

Then add the specific security group to **network interface**

Connect to the AWS box

Connect To Your Instance

I would like to connect with

☒ A standalone SSH client 
☐ A Java SSH Client directly from my browser (Java required) 

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))

2. Locate your private key file (for_live.pem). The wizard automatically detects the key you used to launch the instance.

3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 for_live.pem
```

4. Connect to your instance using its Public DNS:

```
ec2-13-59-191-237.us-east-2.compute.amazonaws.com
```

Example:

```
ssh -i "for_live.pem" ubuntu@ec2-13-59-191-237.us-east-2.compute.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

If you need any assistance connecting to your instance, please see our [connection documentation](#).

Close

Move the files to an AWS EC2 instance/box

Command line to copy files

```
C:\Users\Asus\OneDrive\Desktop> scp -r -i "for_live.pem" ./AFR  
ubuntu@ec2-13-59-191-237.us-east-2.compute.amazonaws.com :~/
```

Page 9 of 11

Install all packages needed on the AWS box

```
sudo apt-get install python3-pip
pip3 install <each of the following packages>
Packages needed:
pip3
pandas
numpy
sklearn
beautifulsoup4
lxml
flask
re
```

Run app.py on the AWS box.

```
ubuntu@ip-172-31-27-97: ~/AFR
ubuntu@ip-172-31-27-97:~/AFR$ python3 app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
183.83.170.52 - - [24/Mar/2019 04:00:38] "GET /index HTTP/1.1" 200 -
```

And then check in browser

For our case test link will be like –

ec2-18-224-96-3.us-east-2.compute.amazonaws.com:8080/querypage

➤ Scalability, limitations, latency of our system

In terms of scalability the model could have been designed better, at this stage predicting sales of future dates does not provide much accurate results, but in case of regression it is always an issue.

Whereas we have used free tier to deploy the project, so the latency is not good. But deploying at proper infrastructure it would have been results better.

Though we have not measure the latency.

➤ Given more time

LSTM model could have been reengineered and used in production. Also standard Time series forecasting models like AR, MA, ARMA, ARIMA, SARIMA, SARIMAX, VAR, VARMA, VARMAX etc.

References

1. <https://www.appliedroots.com/>