**Assignment 1 – Team 9**

**Case 1: Creating Data science pipelines**

**Khusboo Parekh**

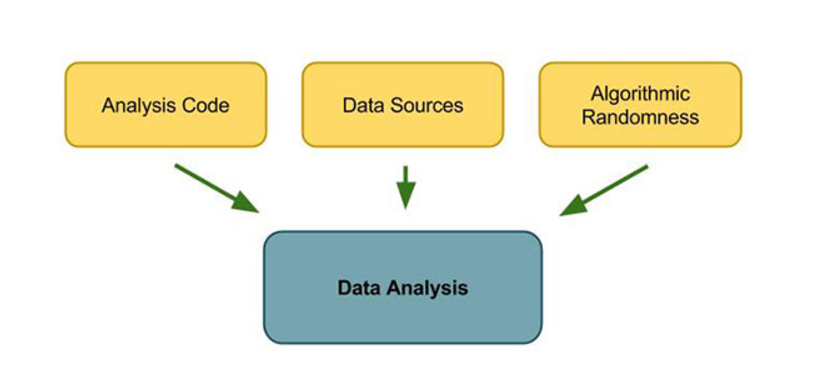
**Tapadyuti Maiti**

**What is Data Pipelining?**

A pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements.

Data Pipeline speeds up your development by providing an easy to use framework for working with batch and streaming data inside your app.

To ensure the reproducibility of your data analysis, there are three dependencies that need to be locked down: analysis code, data sources, and algorithmic randomness.



**Developing a common ETL**

Of course data science that isn’t deployed is useless, and the ability to productionize results is always a central concern of good data scientists. From a data pipelining perspective, one of the key concerns is the development of a common ETL process shared by production and research.

As a simple example, we may want to join-in user data and purchase data to feed into a recommender model for a website. This join would need to happen in two environments: in research, where we need the data for training, and in production, where we need the data for predictions and recommendations. Ideally the ETL code that joins these two chunks of data (and the myriad of data normalization decisions embedded in this code) would be shared between the two environments. In practice, this faces two major challenges:

**Common data format** — In practice, there are a number of constraints to establishing a common data format. You’ll have to pick a data format that plays well with production and [Hadoop](https://hadoop.apache.org/) (or whatever back end datastore you use). Being able to use the same data formats reduces the number of unnecessary data transformations, and helps prevent introducing bugs that contaminate data purity. Using the same data formats also reduces the number of data schemas and formats that your data team has to learn and maintain.

**Isolating library dependencies** — You will want to isolate library dependencies used by your ETL in production. On most research environments, library dependencies are either packaged with the ETL code (e.g. Hadoop) or provisioned on each cluster node (e.g. [mrjob](https://github.com/Yelp/mrjob)). Reducing these dependencies reduces the overhead of running an ETL pipeline. However, production code typically supports a much larger set of of functionality (HTML templating, Web frameworks, task queues) that are not used in research, and vice versa. Data engineers and scientists need to be careful about isolating the ETL production code from the rest of the codebase to keep the dependencies isolated so that the code can be run efficiently on both the front and back ends.

While sharing ETL code between production and research introduces some complexities, it also greatly reduces the potential for errors, by helping guarantee that the transformed data used for model training is the same as the transformed data the models will use in production. Even after locking down your data sources and ensuring data consistency, having separate ETL code can lead to differences in modelling input data that renders the outputs of models completely useless. Common ETL code can go a long way to ensuring reproducible results in your analysis.

**Goal of the Projec**t

Data Service Consultants is putting together a Data-as-a-service application to support their Data science project needs. They need weather data from NOAA(https://www.ncdc.noaa.gov) and are interested in putting together a service that will get them the data they need. So to achieve that following steps need to be performed on the data

1. Data Ingestion
2. Exploratory Data Analysis on raw data
3. Wrangling
4. Exploratory Data Analysis on Clean data

**About the NOAA Data**

The Local Climatological Data (LCD) summaries provide a synopsis of climatic values for a single weather station over a specific month. The summaries are a product of surface observations from both manual and automated (AWOS, ASOS) stations with source data taken from the National Centers for Environmental Information’s Integrated Surface Data (ISD) dataset. Geographic availability includes thousands of locations worldwide. Climatic values given include hourly, daily, and monthly measurements of temperature, dew point, humidity, winds, sky condition, weather type, atmospheric pressure and more.

Few of important parameters form the dataset are as follows:

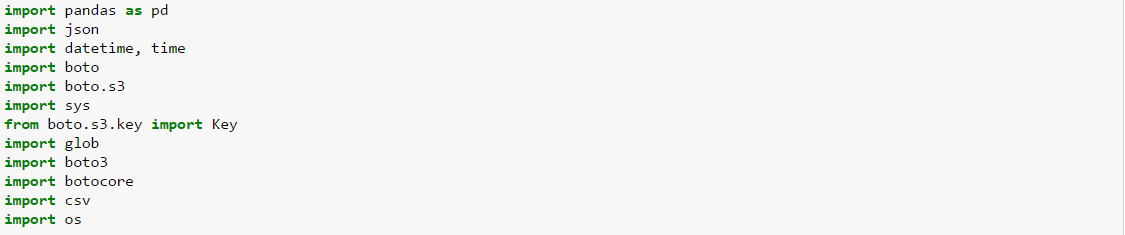
1. Hourly sky condition
2. Precipitation
3. Hourly present weather type
4. Hourly dry bulb temperature
5. Hourly wind speed
6. Relative humidity
7. Sea level pressure
8. Station pressure
9. Altimeter setting
10. Date

**Data Ingestion**

Data ingestion is the process of obtaining and importing data for immediate use or storage in a database. To ingest something is to "take something in or absorb something." Data can be streamed in real time or ingested in batches.

**Steps to perform data ingestion:**

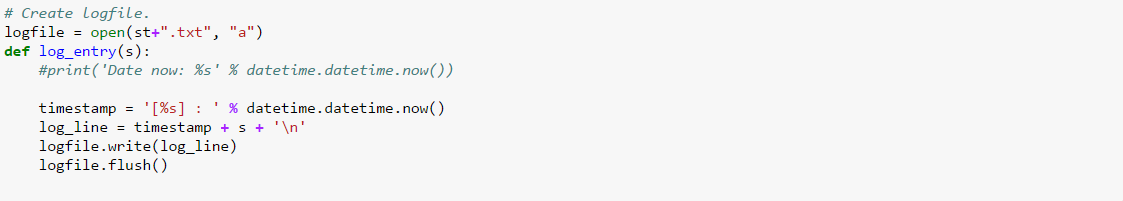
* Imports for the data ingestion code:



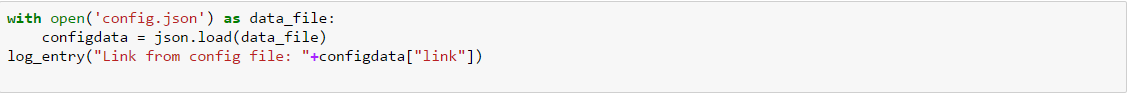
* Code to fetch the timestamp



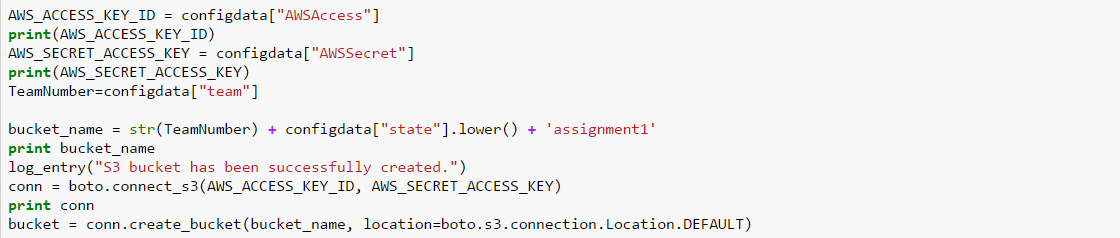
* Code to create log files



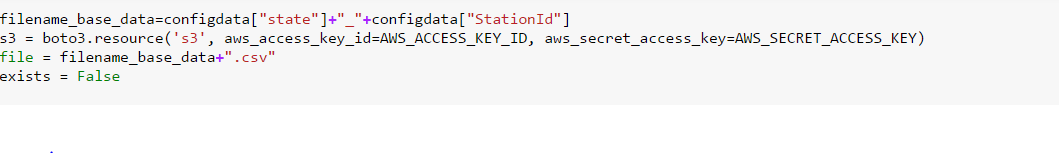
* Code to open and load the config file provided

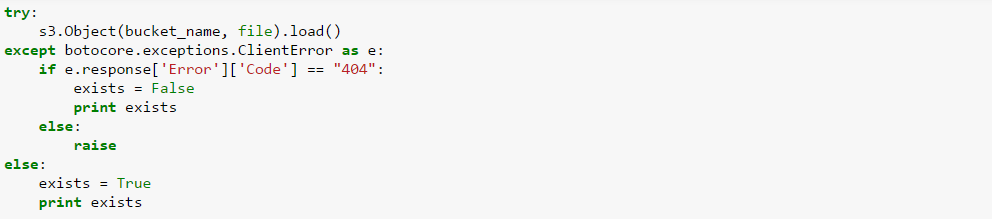


* Code to create bucket on S3



* Code to check for the base file that contains data for 10 years is available on S3 or Local. If the data exists on the local it will upload or else it will download the data from the links provided in the config, merge them and the upload it to s3

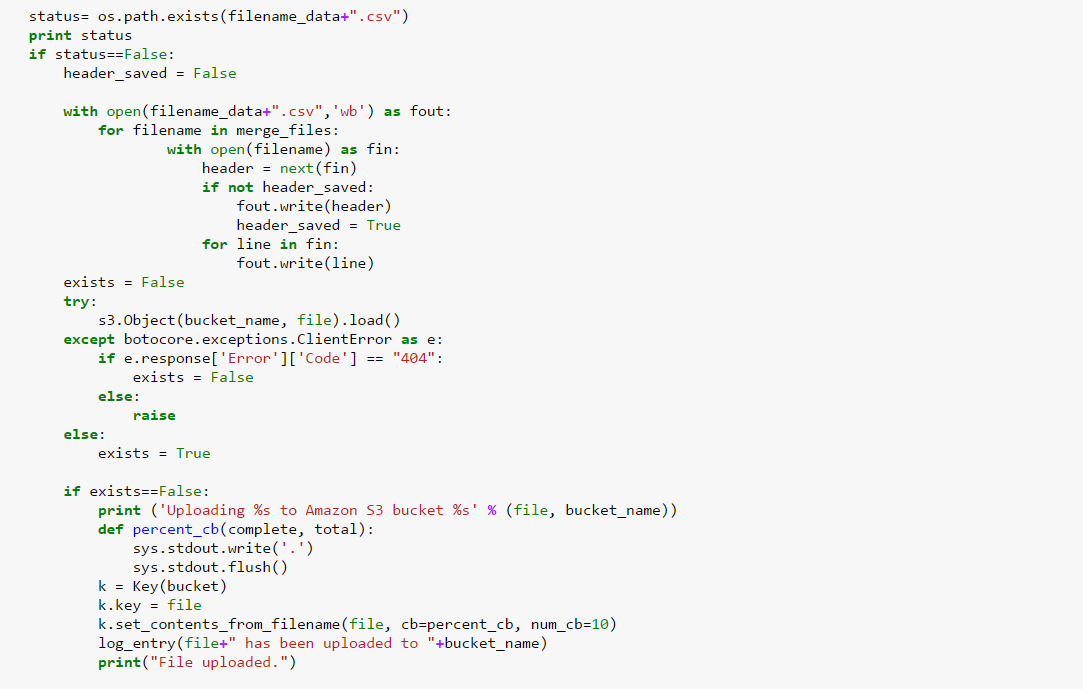




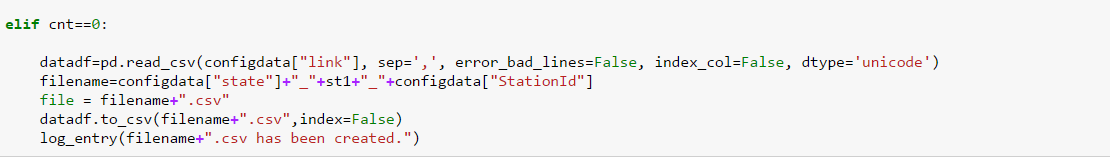


* Code to upload data which is new the new link and adds only the remaining data which has not been added to the base file, creates a new merged file and is uploaded to the s3 bucket. If the file is on local but not on s3 bucket it will upload it / or if the data is not on the local it will download and create new file with new data and upload it again with the new time stamp









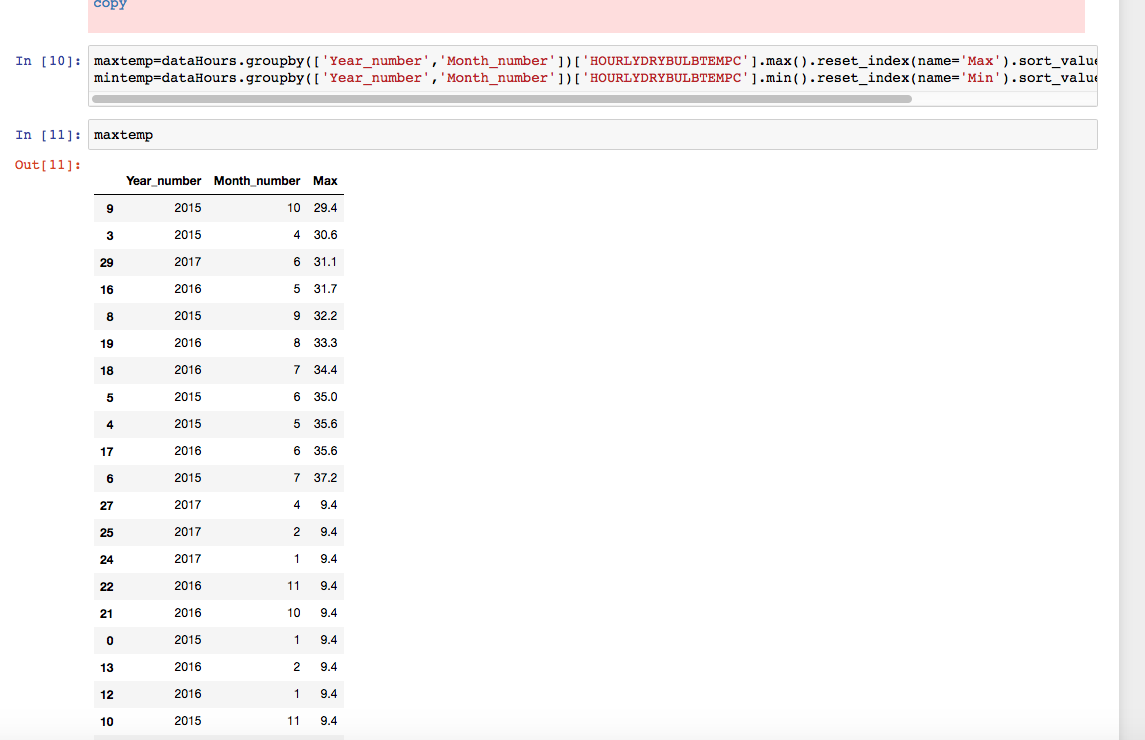
**Raw Data EDA**

Exploratory data analysis (EDA) is an approach to [analyzing](https://en.wikipedia.org/wiki/Data_analysis) [data sets](https://en.wikipedia.org/wiki/Data_set) to summarize their main characteristics, often with visual methods. A [statistical model](https://en.wikipedia.org/wiki/Statistical_model) can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. Exploratory data analysis was promoted by [John Tukey](https://en.wikipedia.org/wiki/John_Tukey) to encourage statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments.

**Analysis Performed on the data:**

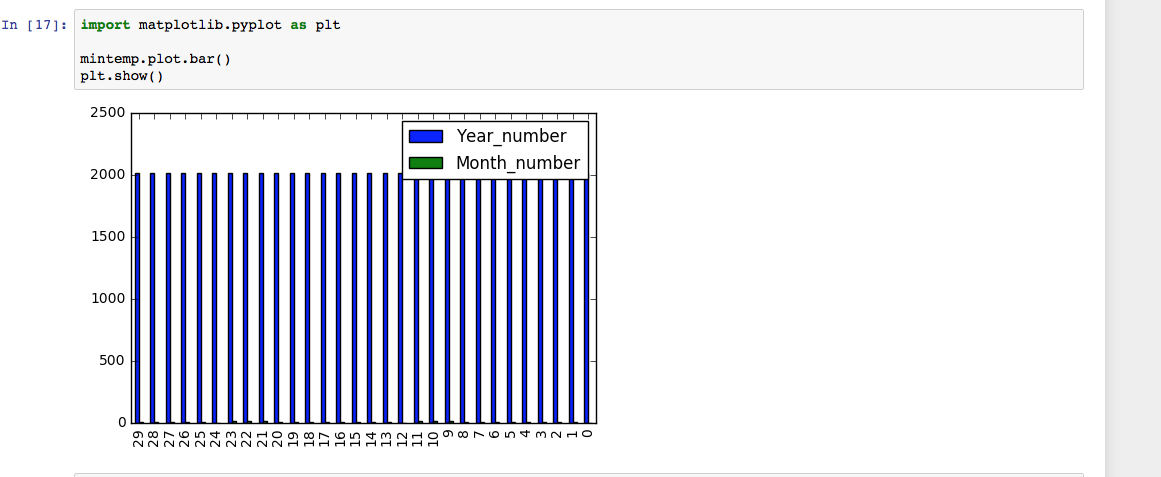
As in Raw EDA , we are importing the data form the Config Link as mentioned in the COnfig.json.

We have found out the maximum temperature in the data sorted according to the Year and Month Number.   
We can clearly allocate and say about the highest peak of the temperature.



Similar analysis is done to find the minimum temperature from the data.

AS we can find from the plot and table that minimum temperature is occurring in the Year End Time.



* Basic Analysis on the REPORT TYPE of weather Data :   
  This would explain us clearly how we are getting data.The code that denotes the type of geophysical surface observation.

Few details of codes that we encountered:

FM-12 = SYNOP Report of surface observation form a fixed land station

FM-13 = SHIP Report of surface observation from a sea station

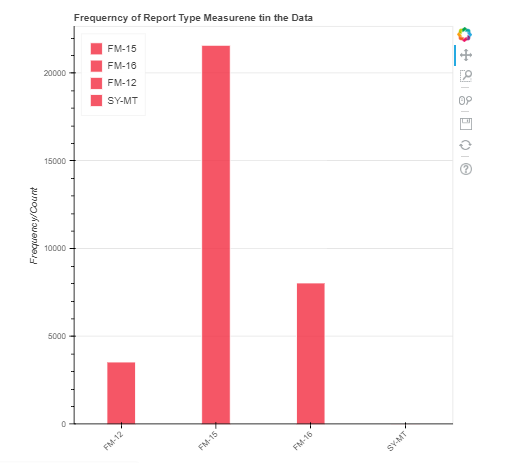
FM-14 = SYNOP MOBIL Report of surface observation from a mobile land station

FM-15 = METAR Aviation routine weather report

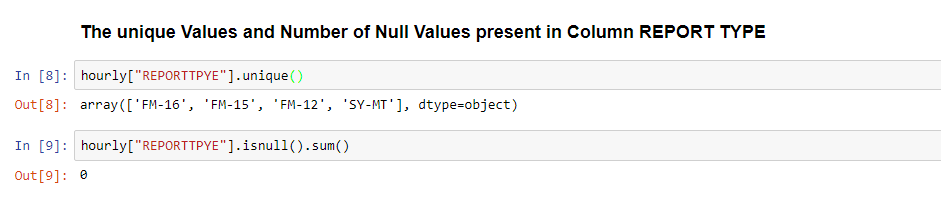
FM-16 = SPECI Aviation selected special weather report

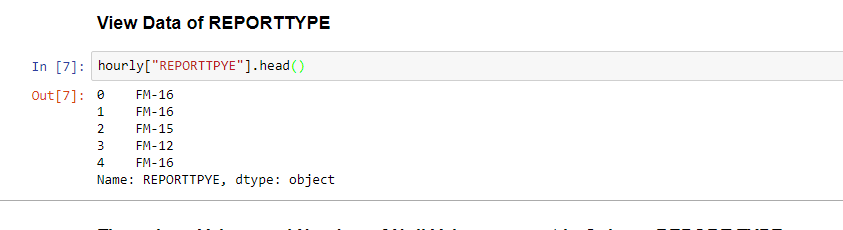
SY-MT = Synoptic and METAR merged report

Form details of other codes : [DetailsofNOAA](https://www1.ncdc.noaa.gov/pub/data/ish/ish-format-document.pdf)



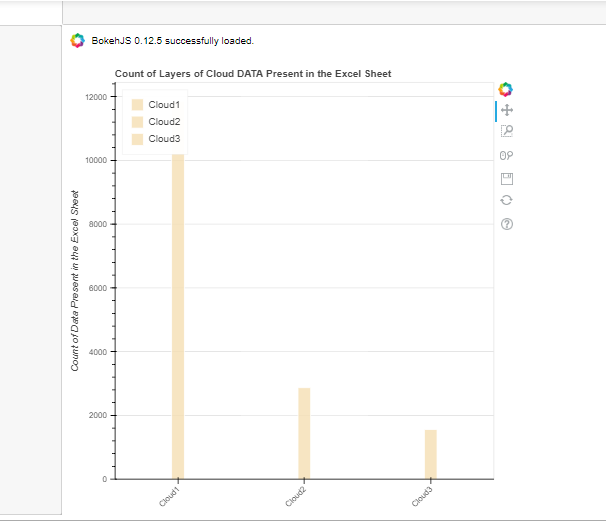
* Finding out how many Null Values present in the Column REPORTTYPE





* **HOURLY SKY CONDITIONS Analysis Information on Layers of Cloud**

This would tell us how the Layers of CLoud would help us to get correlation with other important variables of observation.First we would try to get how much information we got in hand to analyse.Following graph represent how much information we got for the Cloud Layers.

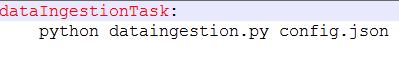


**Data Pipelining Process**

**Make File**

Makefiles are a simple way to organize code compilation. This tutorial does not even scratch the surface of what is possible using *make*, but is intended as a starters guide so that you can quickly and easily create your own makefiles for small to medium-sized projects.

Makefile for data ingestion:



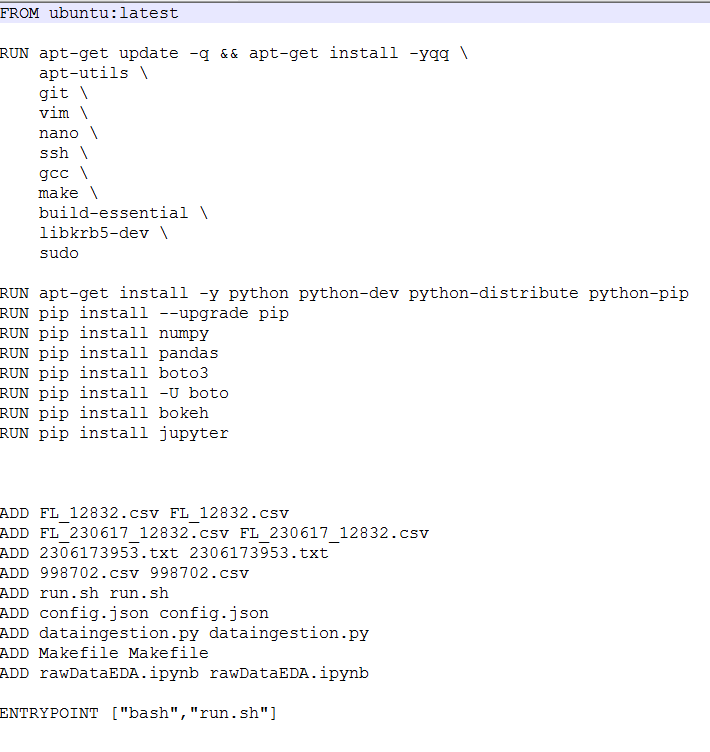
**Run.sh**

It will execute the make command to execute the makefile when invoked by the docker container.



**Docker File**

It invokes the Run.sh when the image is executed.



**Steps to execute a Docker File:**

1. Command to build the docker image

*docker build -t path1 .*

1. Command to run the docker image

*docker run -ti path1*

1. Command to enter into the bash of a container

*docker exec -it containerid bash*

1. Command to create a directory to include all the files required while executing docker image

*docker mkdir assignment1*

1. Command to exit bash

*exit*

1. Command to commit changes

*docker commit containerid khushbuprkh/path1*

1. Command to login to docker hub

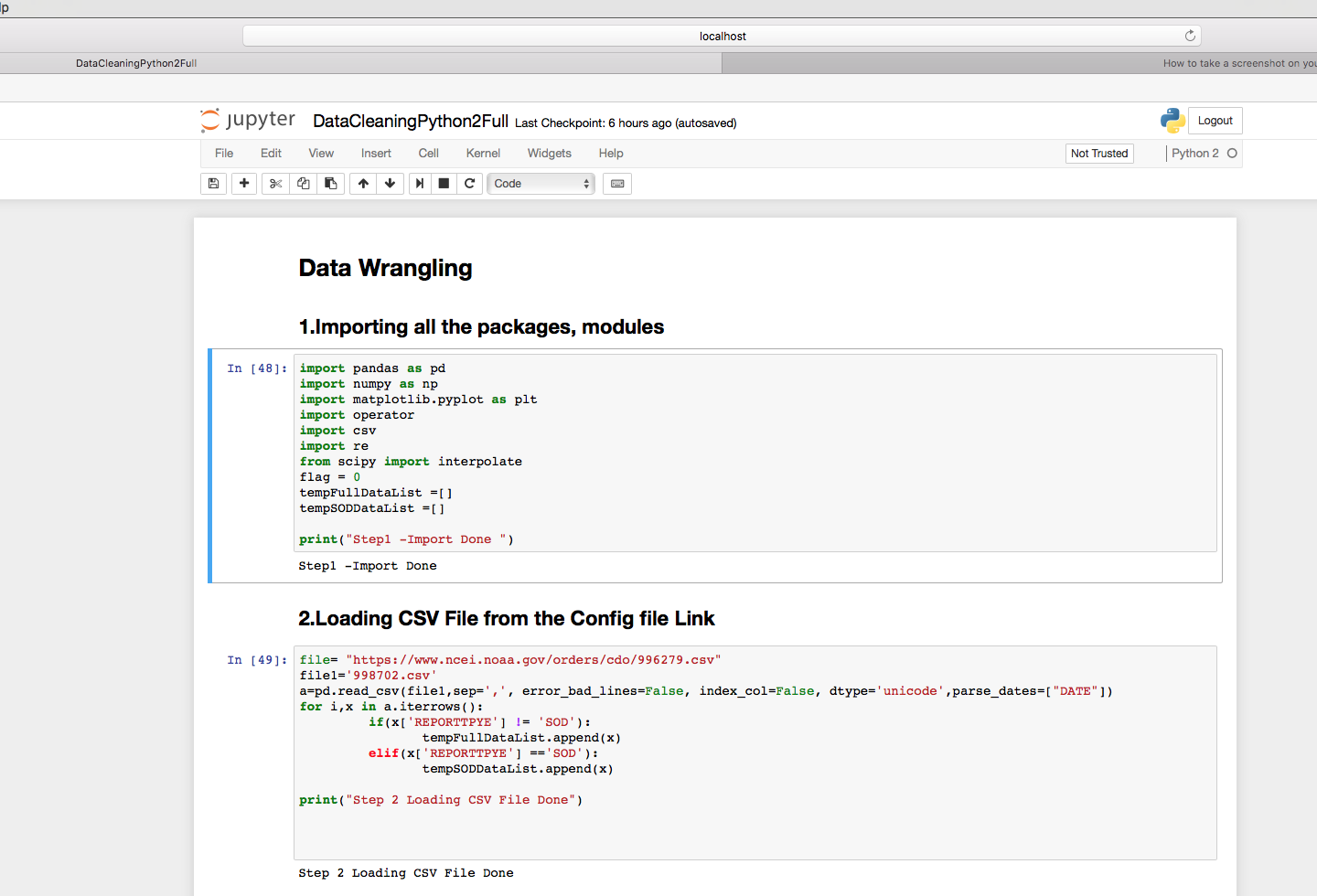
*docker login*

1. Command to push to docker image form local to docker hub

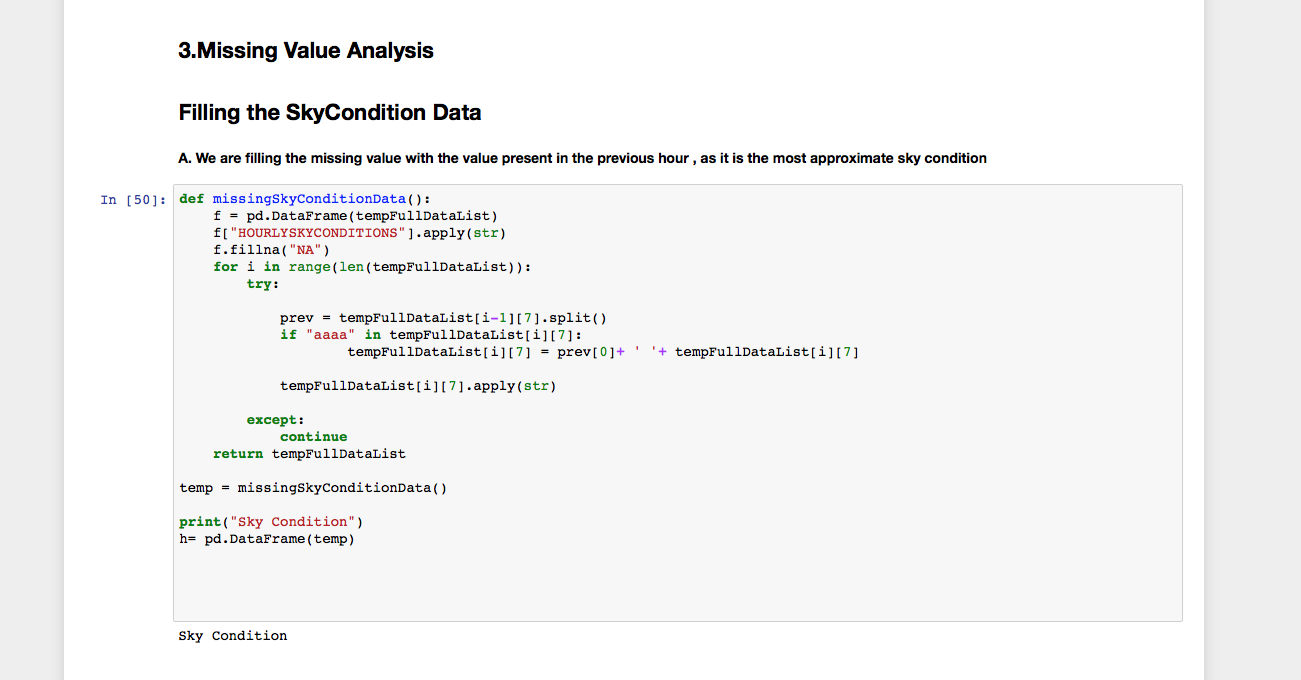
*docker login*

**Clean Data EDA**

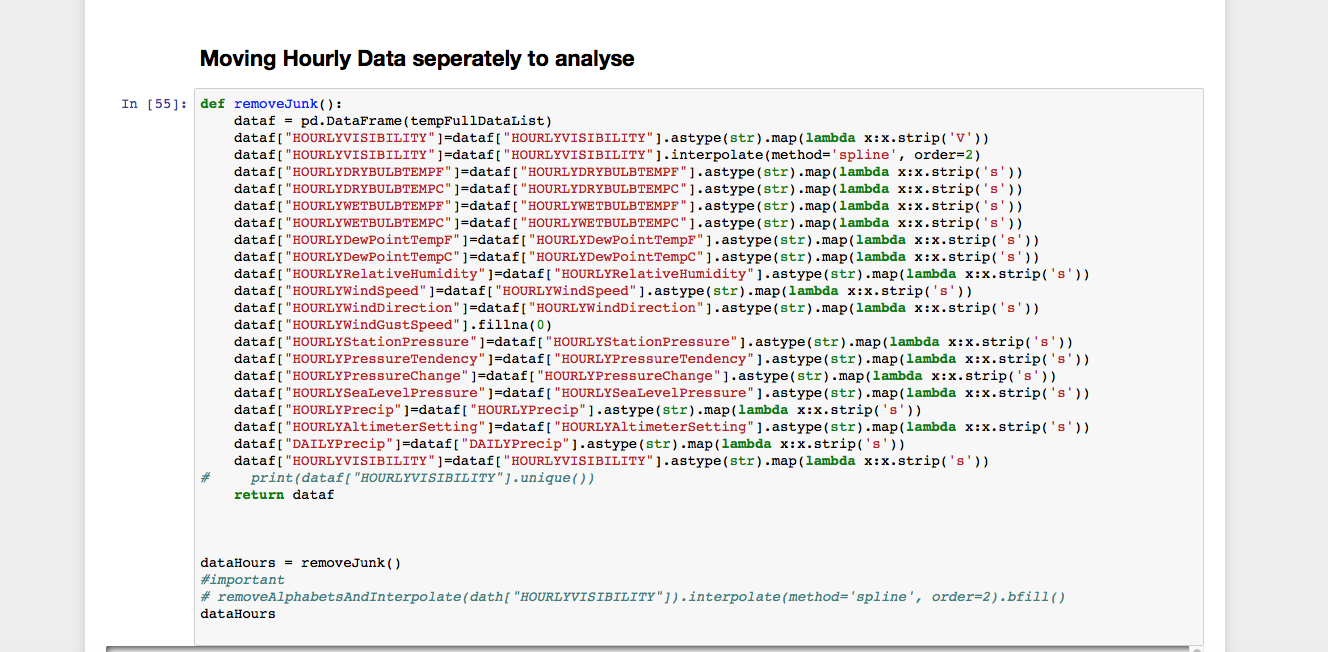
* 1.Before Analysing we are Cleaning the data in the following step:



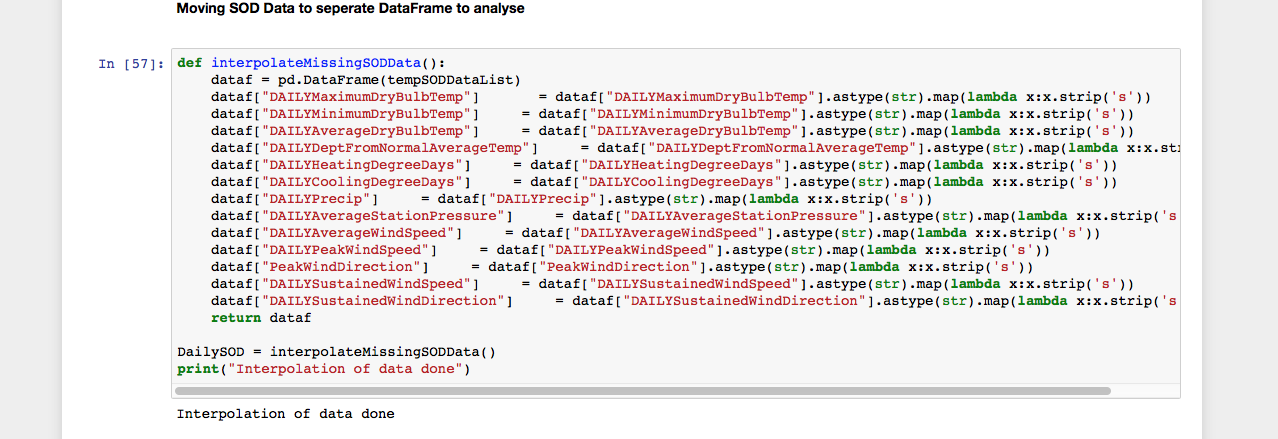
* 2.Populating the missing value in the Hourly Sky Condition Column.   
  It is done on the basis of assumption that whenever the data is missing in the present column , we are taking the value present in the above column.  
  It almost stated that the missing value of Sky condition would be almost similar to the previous hour.



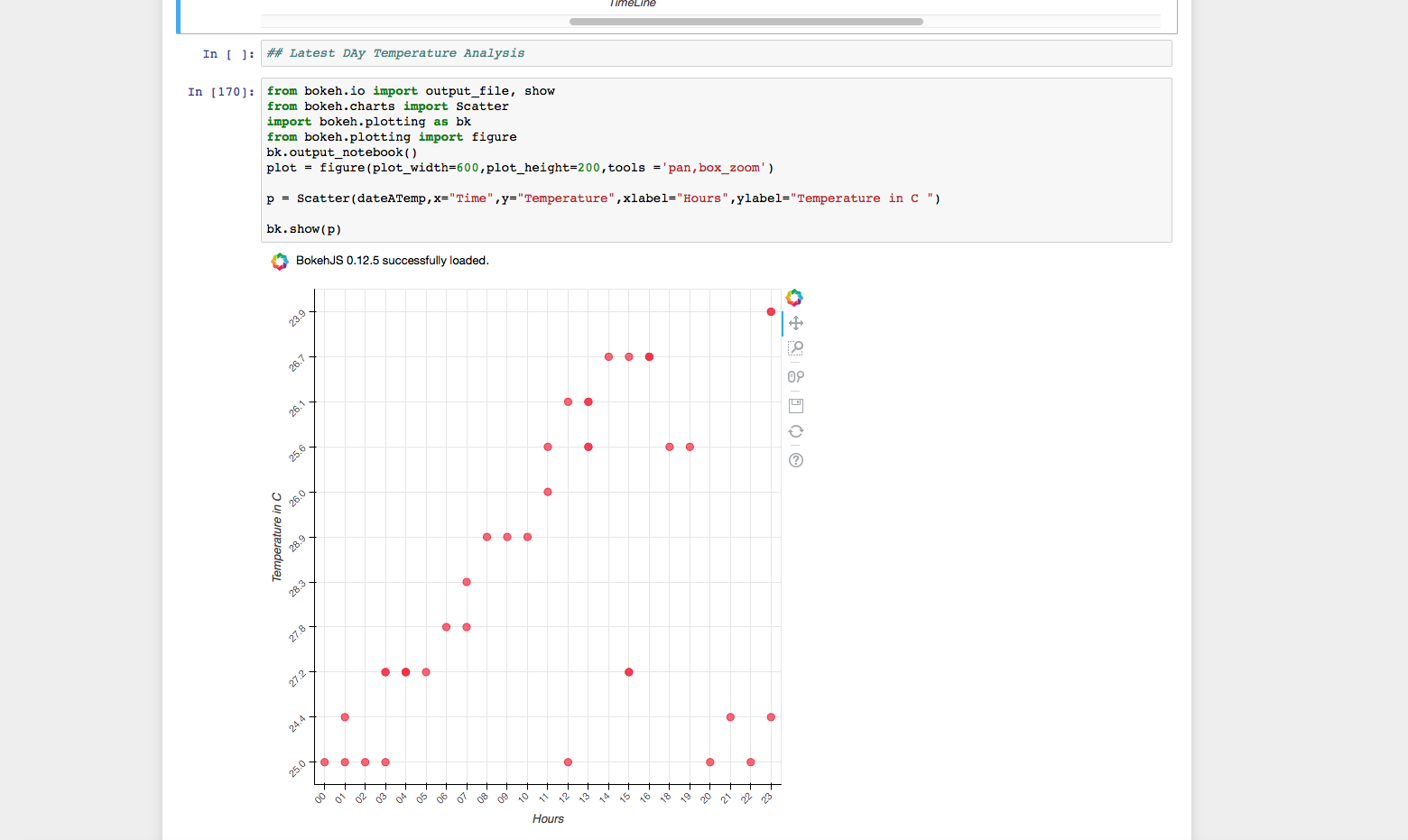
* For Analysing our Data in more explorative way we are using the Bokeh Library to plot our graph.
* Removing the junks from the file for hourly Data:

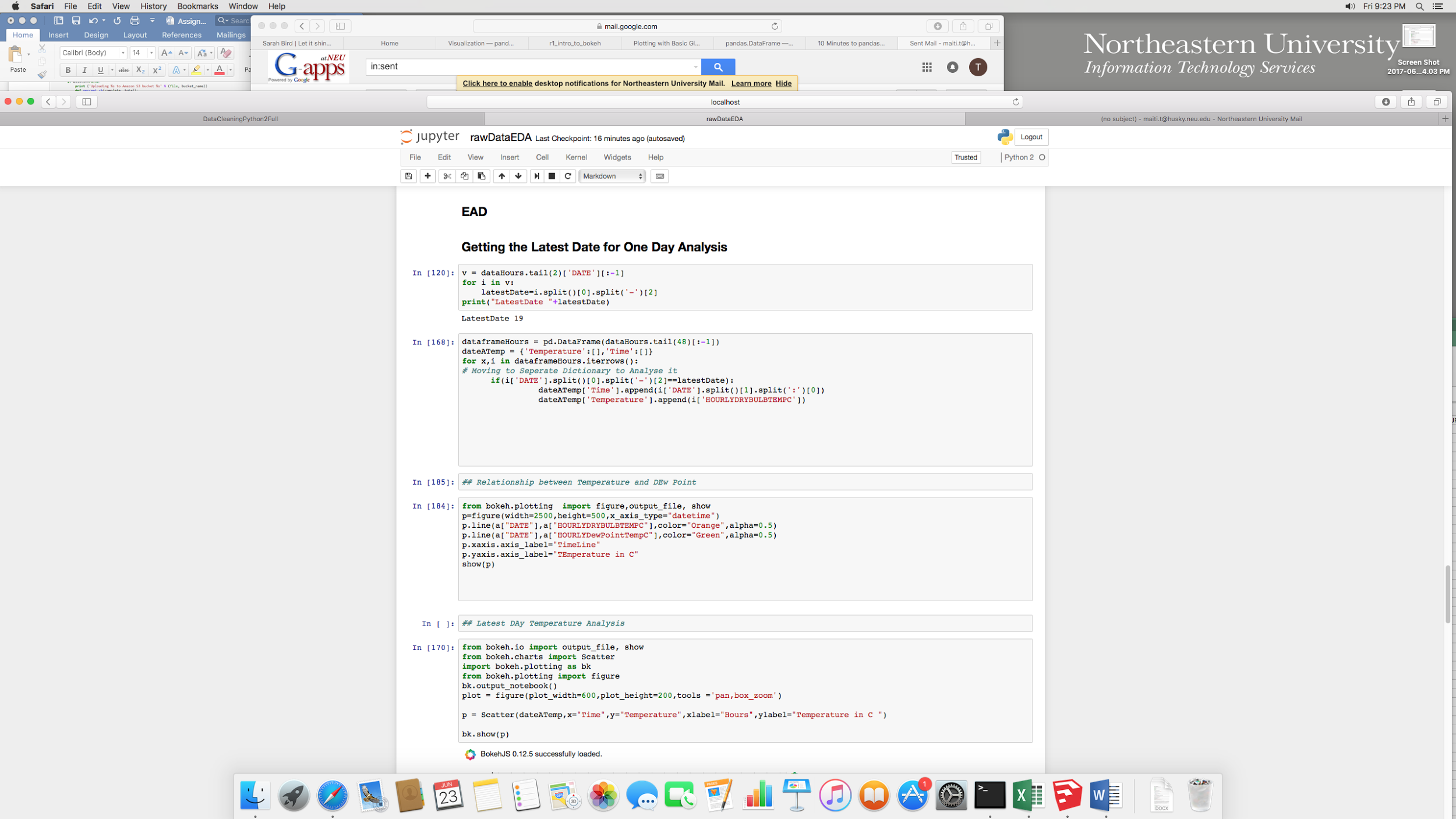


* Removing the junks for Monthly Data

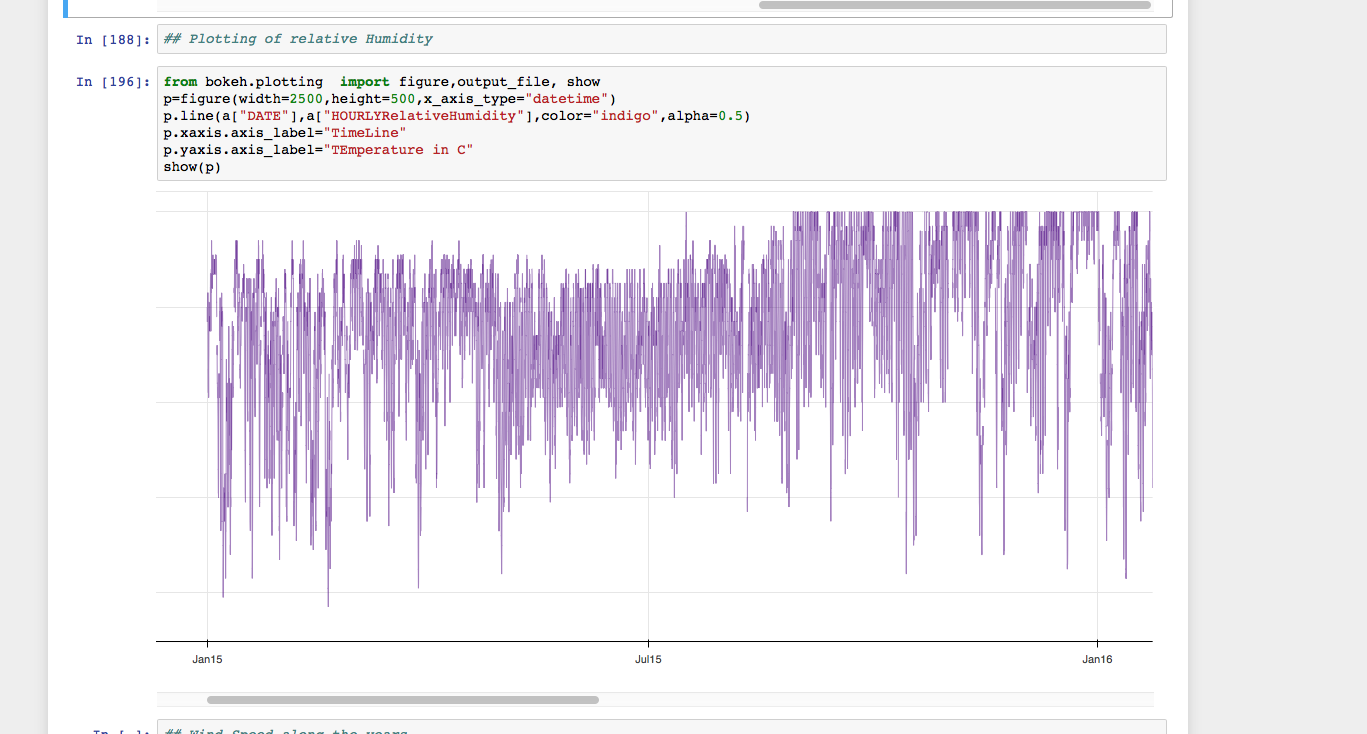


* 3.We are analysing the latest day analysis of last 24 hours on the temperature variance.  
  We have written an function where it will catch all the required data of past 24 hours data.
* Below are few other codes for the Bokeh Plot analysis.





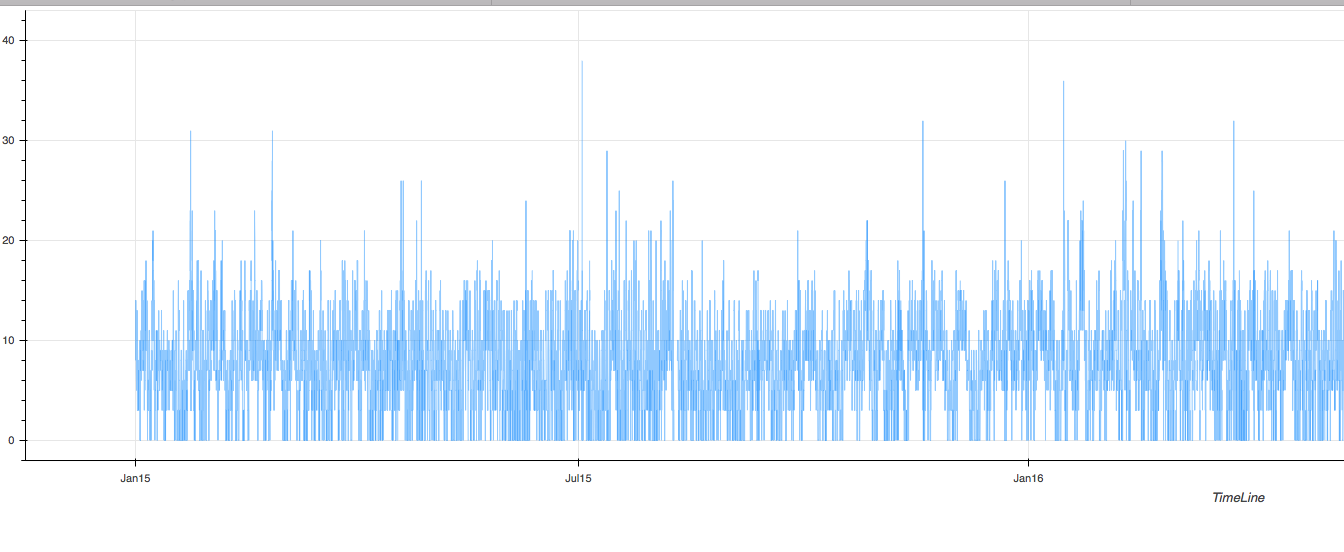
* 4.We are tried to plot Relative Humidity as a Time Series analysis.
* As we can see from the graph Humidity have a lot of spikes , but still we can plot an average tendency to increase along the year from Jan to Jul Time period.



* 5. Analytics of DewPoint and Temperature :  
  As we can infer from the Graph that Dew Point and the Temperature both mapped almost same.



* 6.Wind Speed Analytics L  
  AS we can infer form the graph that wind Speed follows similar pattern of up and Down in a single day.



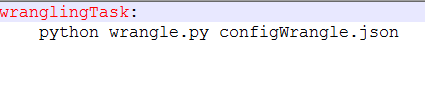
* 7.Analytics of Hourly SeaLevel Pressure , Altimeter Setting, Station Pressure
* All the variables plot almost similar as we can infer all these follow similar fashion in Time series plot.



**Make File**

Makefiles are a simple way to organize code compilation. This tutorial does not even scratch the surface of what is possible using *make*, but is intended as a starters guide so that you can quickly and easily create your own makefiles for small to medium-sized projects.

Makefile for data wrangling:



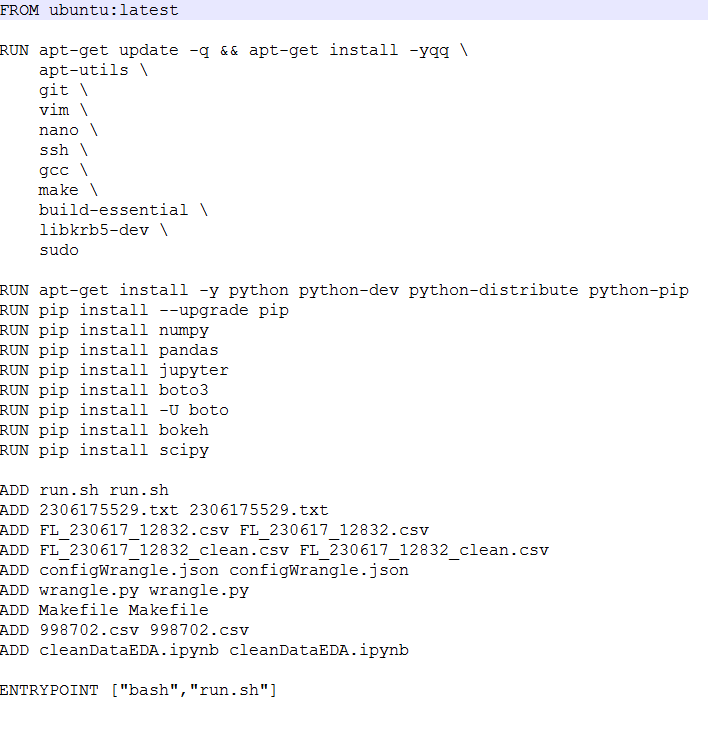
**Run.sh**

It will execute the make command to execute the makefile when invoked by the docker container.



**Docker File**

It invokes the Run.sh when the image is executed.



**Steps to execute a Docker File:**

1. Command to build the docker image

*docker build -t path1 .*

1. Command to run the docker image

*docker run -ti path1*

1. Command to enter into the bash of a container

*docker exec -it containerid bash*

1. Command to create a directory to include all the files required while executing docker image

*docker mkdir assignment1*

1. Command to exit bash

*exit*

1. Command to commit changes

*docker commit containerid khushbuprkh/path1*

1. Command to login to docker hub

*docker login*

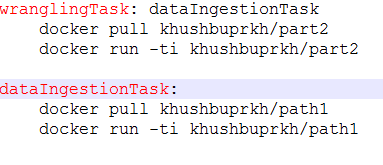
1. Command to push to docker image form local to docker hub

*docker login*

**Creating Pipeline Using Makefile**

Makefiles are a simple way to organize code compilation. This tutorial does not even scratch the surface of what is possible using make, but is intended as a starters guide so that you can quickly and easily create your own makefiles for small to medium-sized projects.

* Initially the dataIngestionTask function is executed first and then wranglingTask gets executed once the DataIngestion task passes.



**AWS Batch – Run Batch Computing Jobs on AWS**

**Job** – A unit of work (a shell script, a Linux executable, or a container image) that you submit to AWS Batch. It has a name, and runs as a containerized app on EC2 using parameters that you specify in a Job Definition. Jobs can reference other jobs by name or by ID, and can be dependent on the successful completion of other jobs.

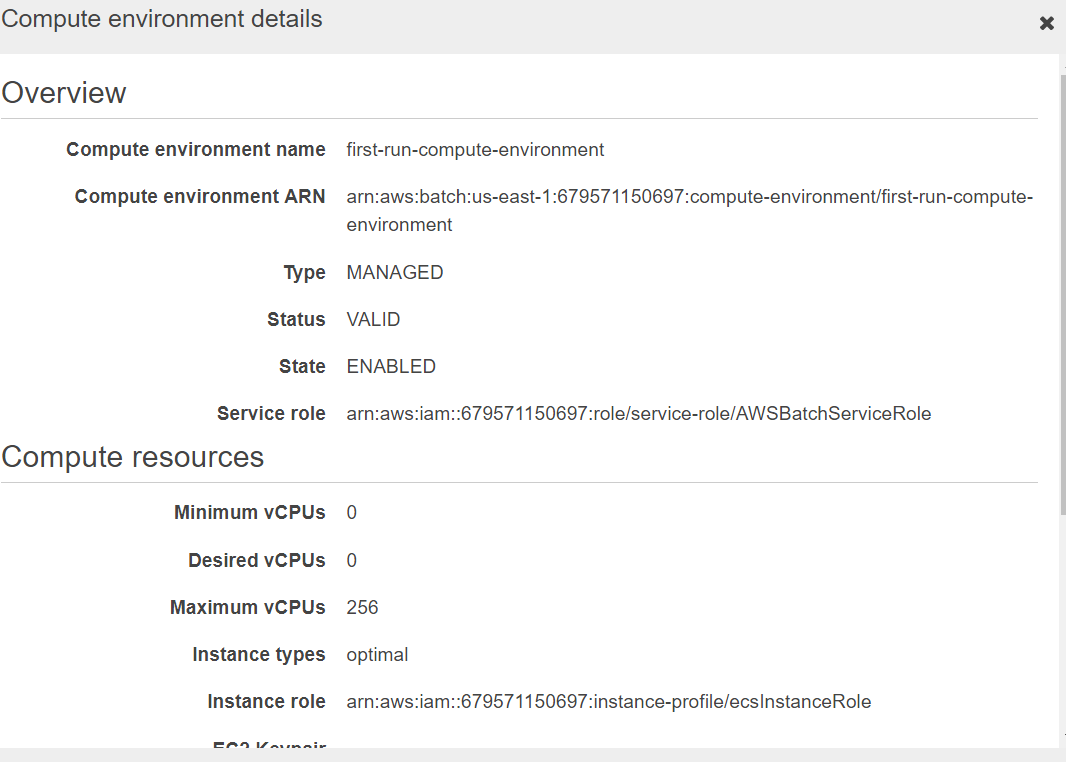
**Job Definition** – Specifies how Jobs are to be run. Includes an [AWS Identity and Access Management (IAM)](https://aws.amazon.com/iam/) role to provide access to AWS resources, and also specifies both memory and CPU requirements. The definition can also control container properties, environment variables, and mount points. Many of the specifications in a Job Definition can be overridden by specifying new values when submitting individual Jobs.

**Job Queue** – Where Jobs reside until scheduled onto a Compute Environment. A priority value is associated with each queue.

**Compute Environment** – A set of managed or unmanaged compute resources that are used to run jobs. Managed environments allow you to specify desired instance types at several levels of detail. You can set up Compute Environments that use a particular type of instance, a particular model such as **c4.2xlarge** or **m4.10xlarge**, or simply specify that you want to use the newest instance types. You can also specify the minimum, desired, and maximum number of vCPUs for the environment, along with a percentage value for bids on the [Spot Market](https://aws.amazon.com/ec2/spot/) and a target set of VPC subnets. Given these parameters and constraints, AWS Batch will efficiently launch, manage, and terminate EC2 instances as needed. You can also launch your own Compute Environments. In this case you are responsible for setting up and scaling the instances in an [Amazon ECS](https://aws.amazon.com/ecs/) cluster that AWS Batch will create for you.

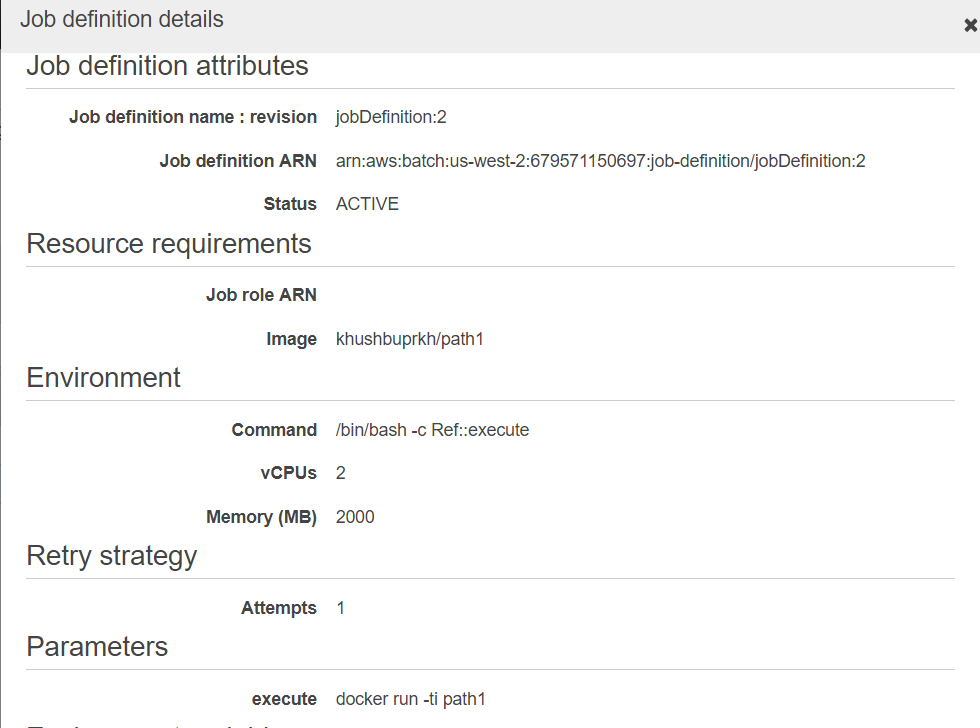
**Creating Batch Job on AWS**

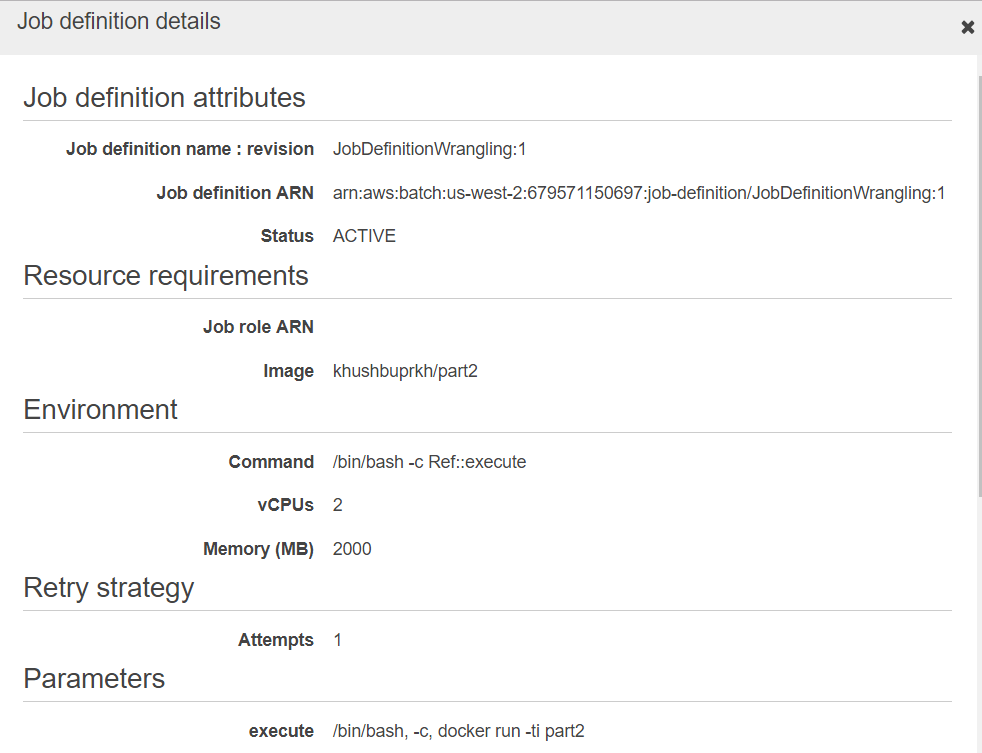
Compute environments contain EC2 instances that are used to run containerized batch jobs. In a managed compute environment AWS handles the scaling and provisioning of your instances for you; in an unmanaged compute environment you control the instances.



**Job Definition**

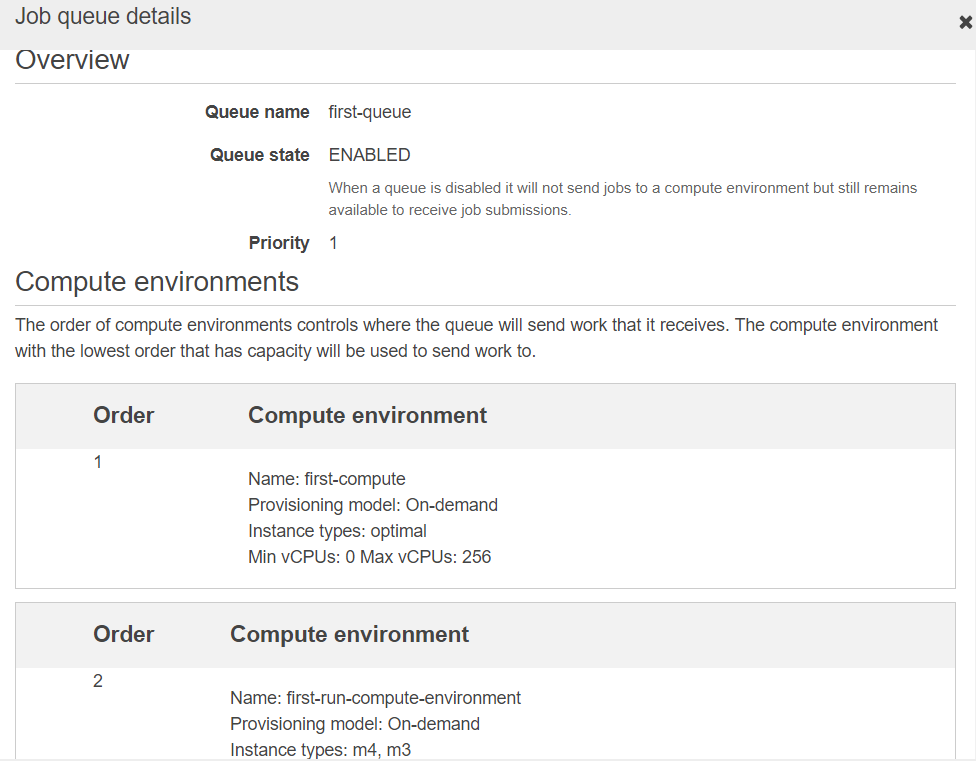
Job definitions specify information about your job, such as the container image, command, CPU and memory resources, and environment variables. You can choose to override several job definition parameters at submission time.

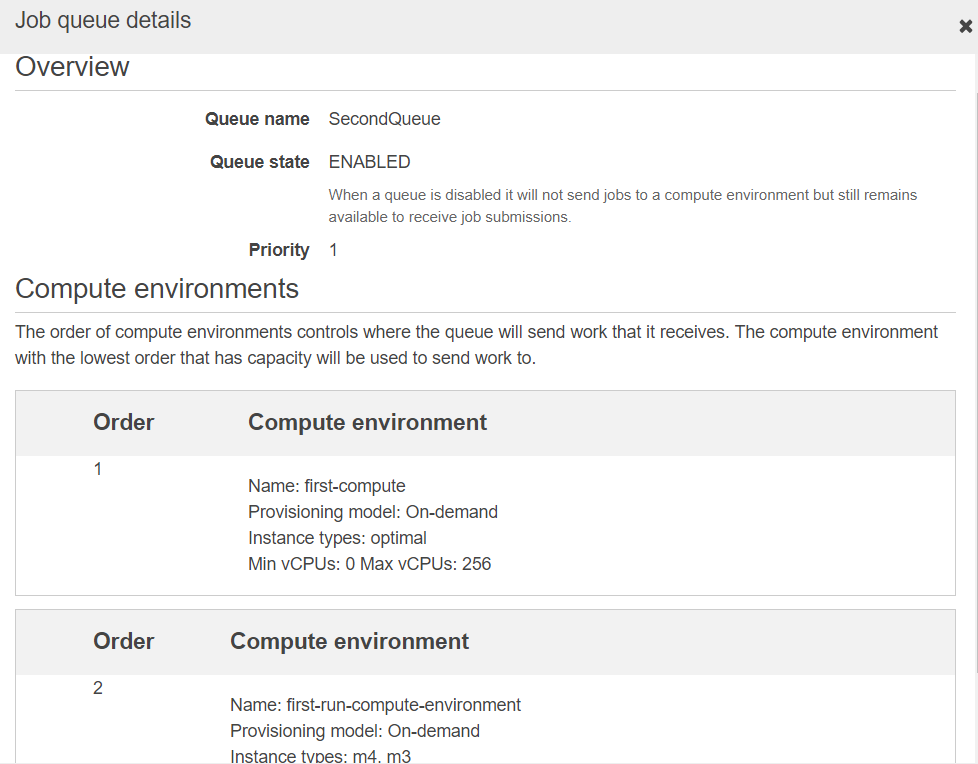




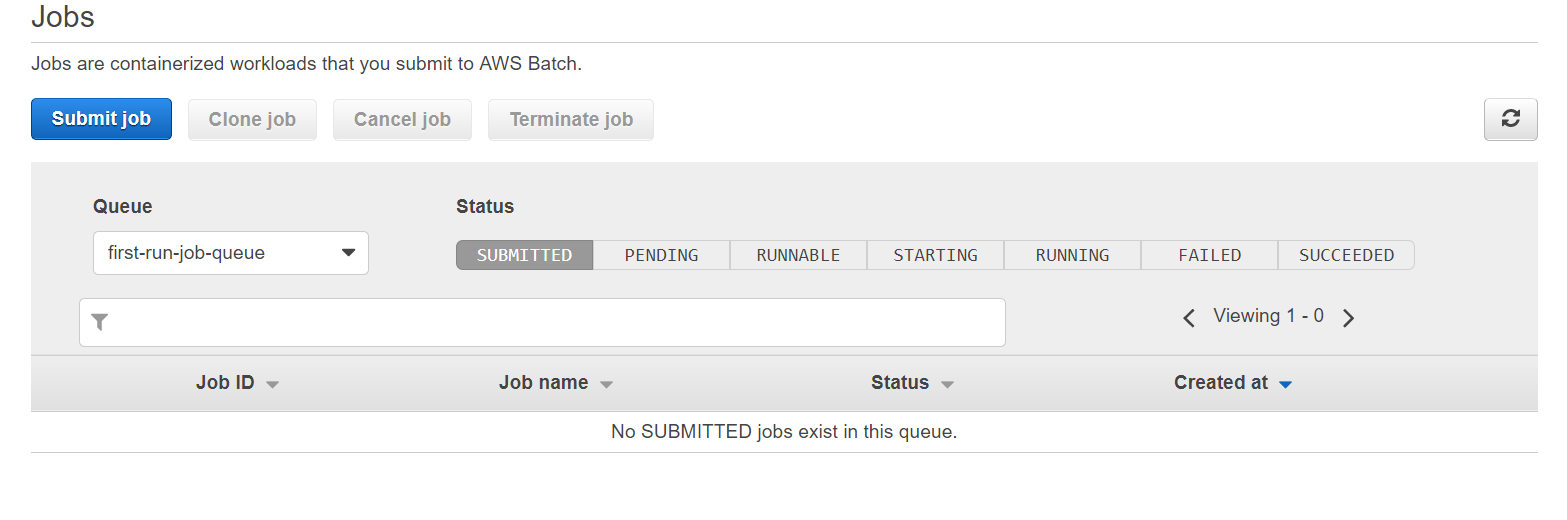
**Job Queue**

You submit AWS Batch jobs to a job queue. Job queues are given a priority value and job queues with a higher integer value for priority are given preference for compute resources.

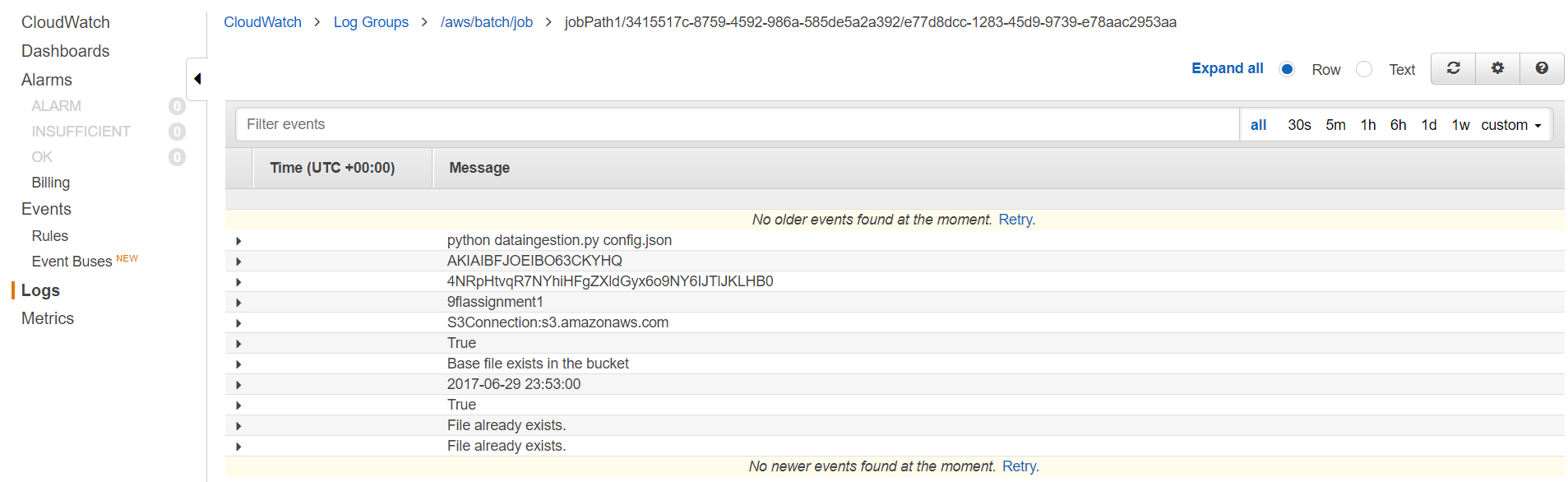




**Submitting Job for Execution**

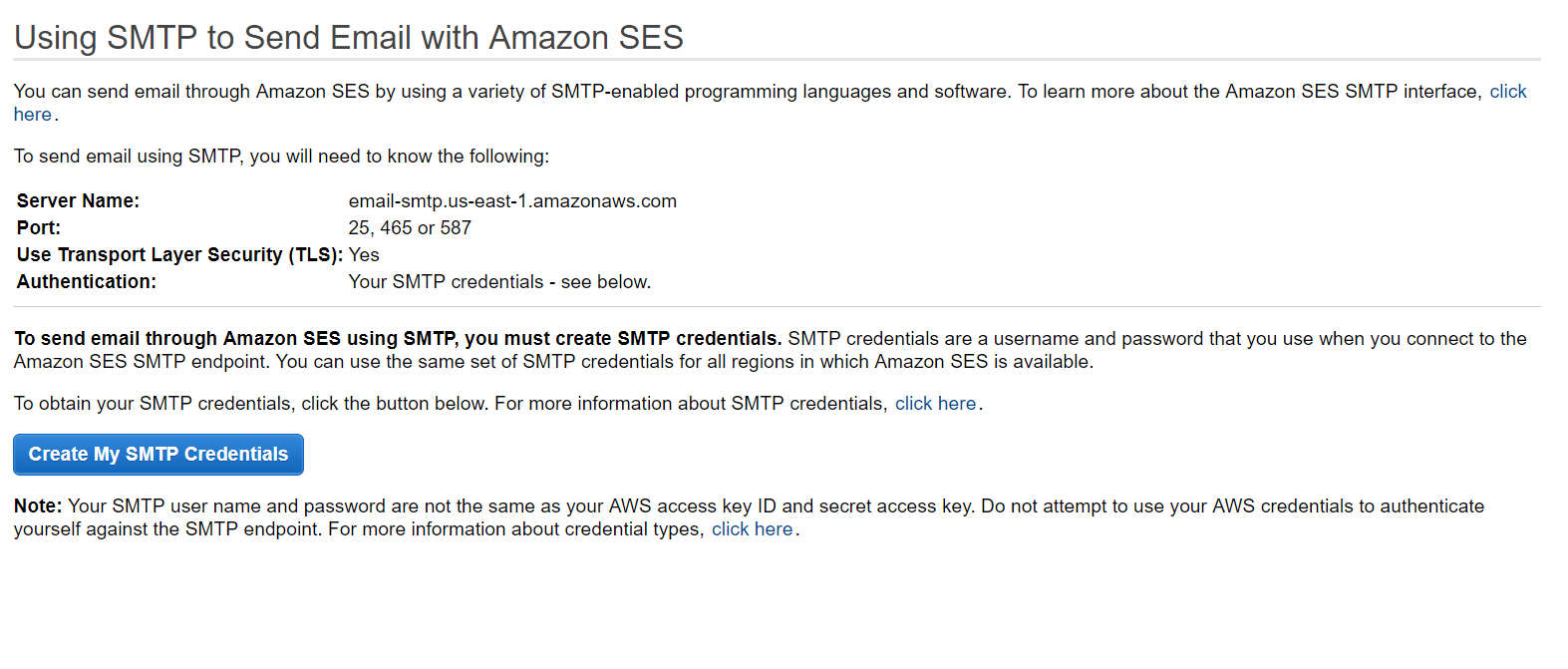


**Output of the Job execution viewed using cloud watch**

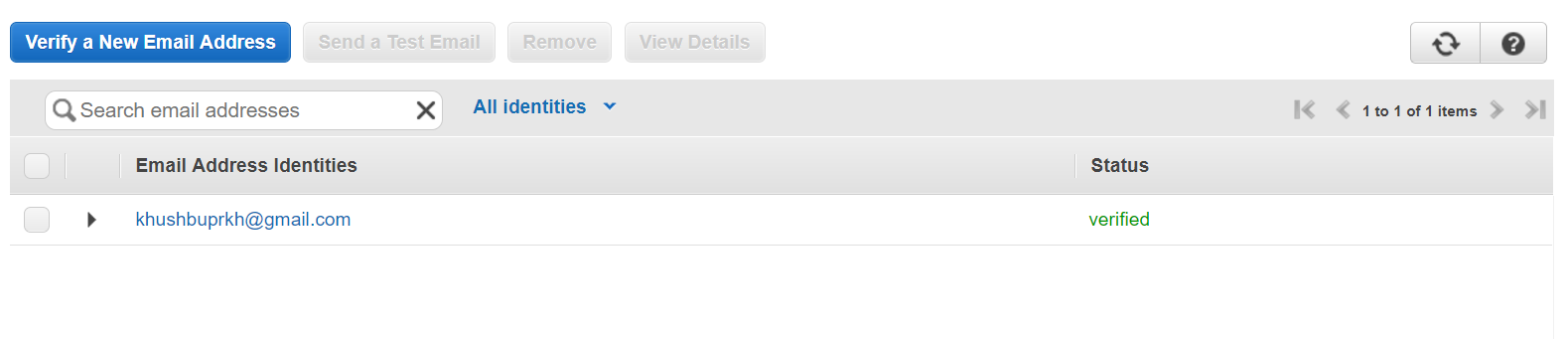


**SES**

**Creating SMTP Credentials**



**Register for email Address**



**Receive Mail when data is cleaned**

