PROJECT REPORT

BREAST CANCER RISK PREDICTION USING AWS SAGEMAKER

Name : Prabhanjan Kumar

Project:Brest Cancer Risk Prediction Using Aws Sagemaker

Domain: Machine Learning

CONTENTS:

1.INTRODUCTION

1.1 OVERVIEW

Cancer has been characterized as a heterogeneous disease consisting of many different subtypes. The early diagnosis and prognosis of a cancer type have become a necessity in cancer research, as it can facilitate the subsequent clinical management of patients. The importance of classifying cancer patients into high or low risk groups has led many research teams, from the biomedical and the bioinformatics field, to study the application of machine learning (ML) methods. Therefore, these techniques have been utilized as an aim to model the progression and treatment of cancerous conditions. In addition, the ability of ML tools to detect key features from complex datasets reveals their importance. A variety of these techniques, including Artificial Neural Networks (ANNs), Bayesian Networks (BNs), Support Vector Machines (SVMs) and Decision Trees (DTs) have been widely applied in cancer research for the development of predictive models, resulting in effective and accurate decision making. Even though it is evident that the use of ML methods can improve our understanding of cancer progression, an appropriate level of validation is needed in order for these methods to be considered in the everyday clinical practice. In this work, we present a review of recent ML approaches employed in the modeling of cancer progression. The predictive models discussed here are based on various supervised ML techniques as well as on different input features and data samples.

2.LITERATURE SURVEY:

2.1EXISTING PROBLEM:

Breast cancer is one of the main causes of cancer death worldwide. Early diagnostics significantly increases the chances of correct treatment and survival, but this process is tedious and often leads to a disagreement between pathologists.

Computer-aided diagnosis systems showed the potential for improving diagnostic accuracy. But early detection and prevention can significantly reduce the chances of death. It is important to detect breast cancer as early as possible.We will be building and deploying the model in AWS SageMaker and use SNS service to generate alerts about the risk.

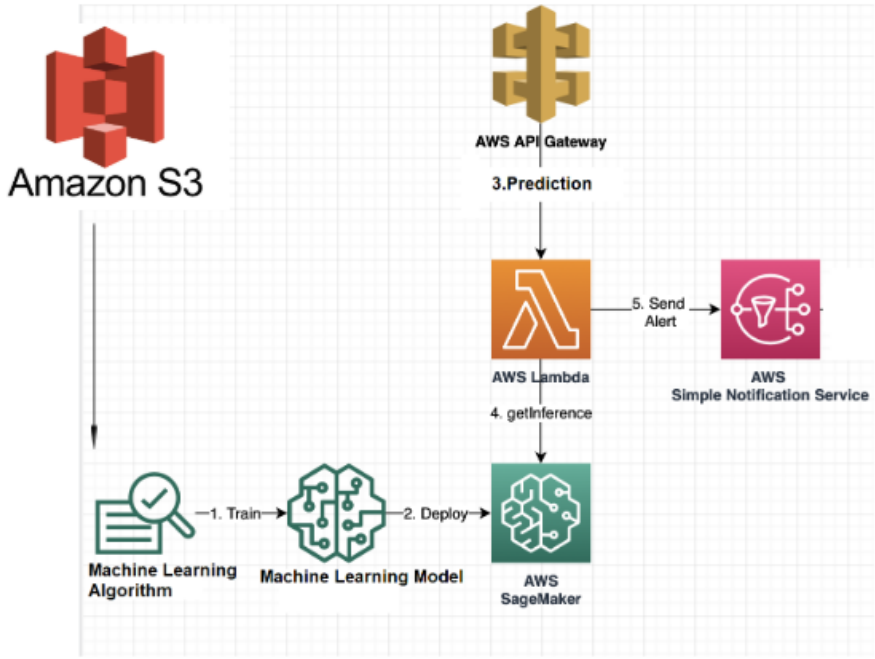
.

2.2PROPOSED SOLUTION:

Develop a model that is capable of detecting the Breast Cancer in early stages. The model must send alerts using the SNS service. The Machine learning model is trained and deployed on Amazon Sage Maker. Create an API Endpoint for the model with the help of API Gateway and AWS Lambda Service

3.THEORITICAL ANALYSIS:

3.1. BLOCK DIAGRAM:



3.2. SOFTWARE DESIGNING:

1. Amazon S3
2. AWS API Gateway
3. AWS Lambda
4. AWS SNS
5. Amazon SageMaker
6. Python 3

4.EXPERIMENTAL INVESTIGATIONS:

Aws Cloud:

Aws Cloud Provides Many Services Such as Sagemaker,lambda and Api Gateway,etc..

Sagemaker:

Amazon SageMaker is a fully managed service that provides every developer and data scientist with the ability to build, train, and deploy machine learning (ML) models quickly. SageMaker removes the heavy lifting from each step of the machine learning process to make it easier to develop high quality models.

Lambda:

With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services

Api Gateway:

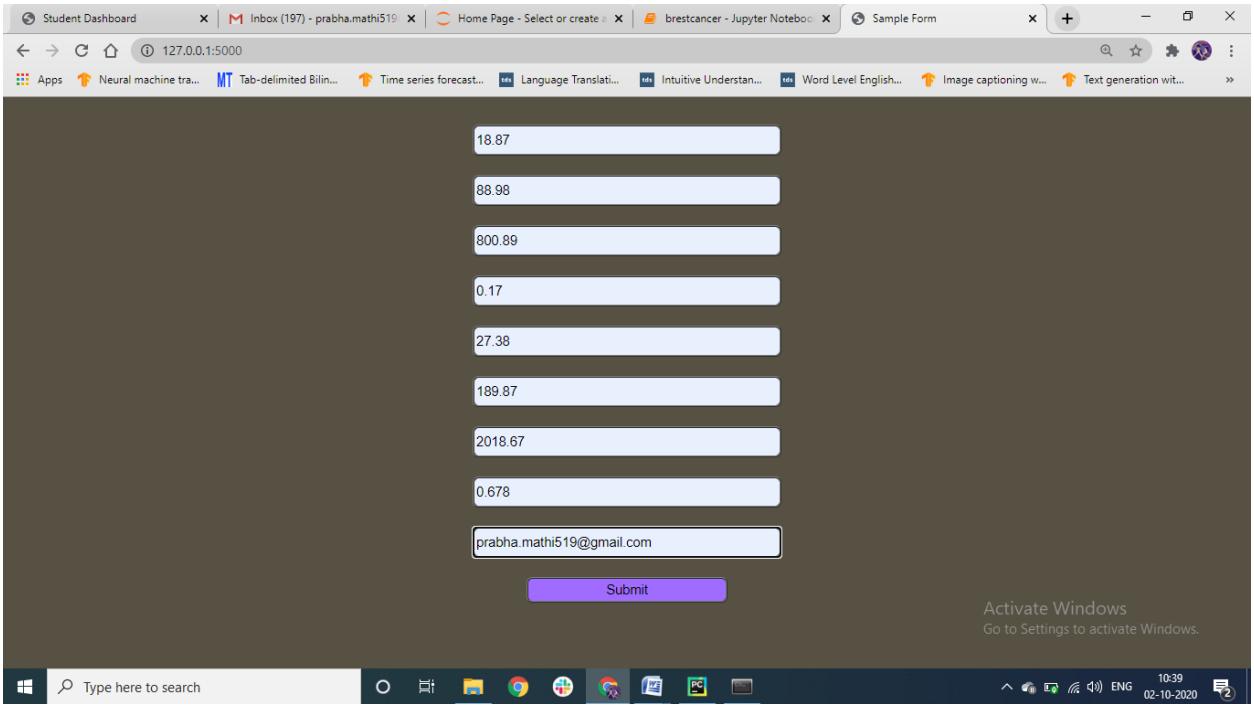
Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud.API Gateway creates RESTful APIs that:Are

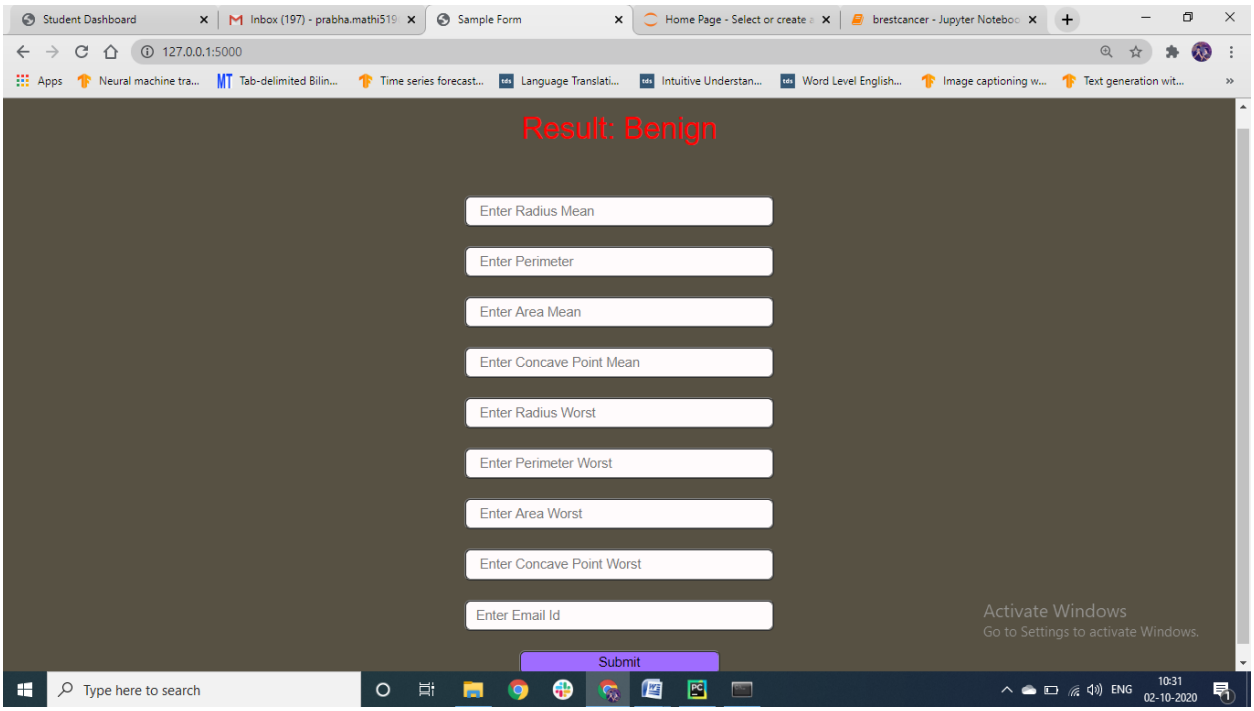
HTTP-based.

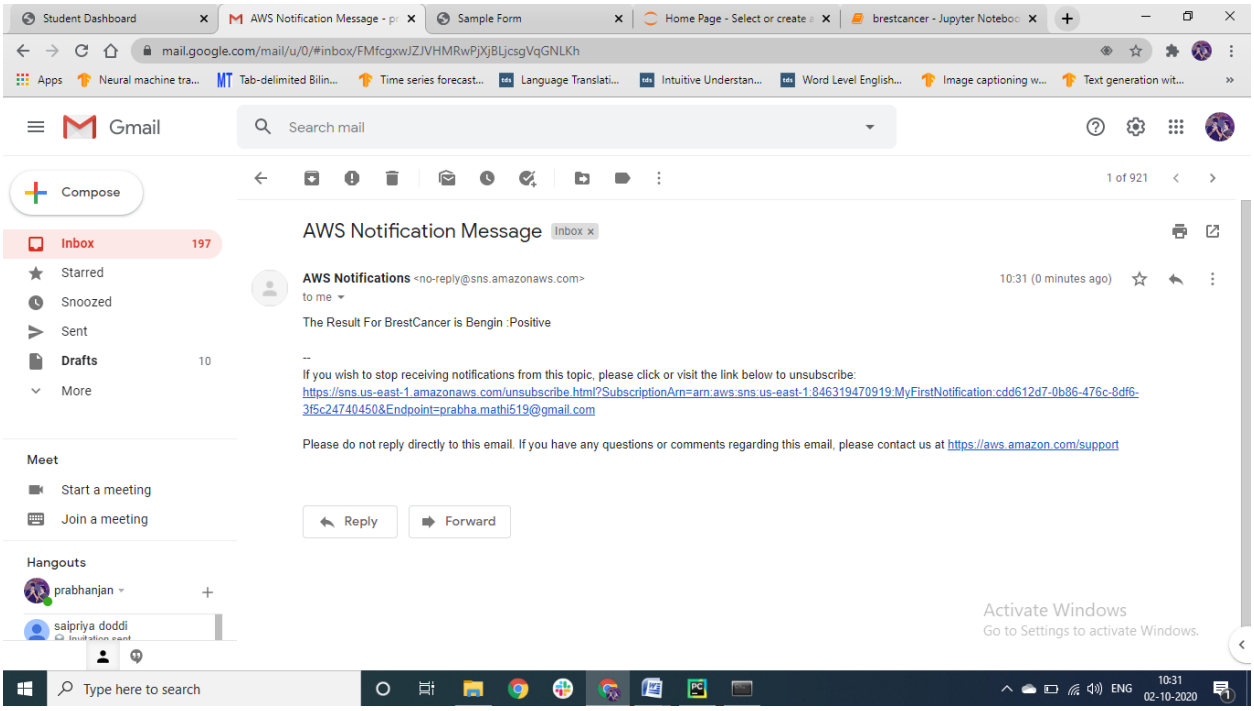
Sns:

Amazon Simple Notification Service (SNS) is a fully managed messaging service for both system-to-system and app-to-person (A2P) communication. It enables you to communicate between systems through publish/subscribe (pub/sub) patterns that enable messaging between decoupled microservice applications or to communicate directly to users via SMS, mobile push and email.

5.RESULT:







6.ADVANTAGES:

1. It helps to detect the brest cancer in earlier stage
2. It is easy to diagnosis the brest cancer

7.APPLICATIONS:

Health Care Industry

8.FUTURE SCOPE:

The analysis of the results signifies that the integration of multidimensional data along with different classification, feature selection and dimensionality reduction techniques can provide auspicious tools for inference in this domain. Further research in this field should be carried out for the better performance of the classification techniques so that it can predict on more variables. We should intend how to parametrize our classification techniques hence to achieve high accuracy. We should look into many datasets and how further Machine Learning algorithms can be used to characterize Breast Cancer. We want to reduce the error rates with maximum accuracy.

9.CONCLUSION:

There was a striking improvement in the accuracy of classification of women with and without breast cancer achieved with ML algorithms compared to the state-of-the-art model-based approaches. High-accuracy prediction techniques are important in personalized medicine because they facilitate stratification of prevention strategies and individualized clinical management. Predictive models are essential in personalized medicine because they contribute to early identification of high-risk individuals based on known epidemiological and clinical risk factors. Accurate breast cancer risk estimates can inform clinical care and risk management across the breast cancer continuum, e.g., behavioral changes, chemoprevention, personalized screening, and risk-stratified follow-up care. Available risk prediction models have an overall accuracy less than 0.65. ML approaches offer the exciting prospect of achieving improved and more precise risk estimates. This is the first step in developing new risk prediction approaches and further explores diverse risk factors.

10.BIBILOGRAPHY:

1. Evans DG, Howell A. Breast cancer risk-assessment models. Breast Cancer Res. 2007 Sep 12;9(5):213. pmid:17888188

Code:

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
```

```
1 dataset=pd.read_csv('brestcancer.csv')
2 dataset.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	te
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 33 columns

```
1 dataset.shape
```

```
(569, 33)
```

```
1 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
1 dataset=dataset.drop(['Unnamed: 32','id'],axis=1)
2 dataset['diagnosis'].unique()
```

```
array(['M', 'B'], dtype=object)
```

```
1 dataset.isnull().sum()
```

```
diagnosis                 0
radius_mean               0
texture_mean              0
perimeter_mean            0
area_mean                 0
smoothness_mean           0
compactness_mean          0
concavity_mean            0
concave points_mean       0
symmetry_mean             0
fractal_dimension_mean    0
radius_se                 0
texture_se                0
perimeter_se              0
area_se                   0
smoothness_se             0
compactness_se            0
concavity_se              0
concave points_se         0
symmetry_se               0
fractal_dimension_se      0
radius_worst              0
texture_worst             0
perimeter_worst           0
area_worst                0
smoothness_worst          0
compactness_worst         0
concavity_worst           0
concave points_worst      0
symmetry_worst            0
fractal_dimension_worst   0
dtype: int64
```

```
1 from sklearn.preprocessing import LabelEncoder
2 le=LabelEncoder()
3 dataset.iloc[:,0]=le.fit_transform(dataset.iloc[:,0
  ])
4 dataset.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

```
1 from scipy import stats
```

```
corel_points=[]
for i in range(31):
    if i==0:
        continue
    else:

person_cof,p_value=stats.pearsonr(dataset.iloc[:,0],dataset.iloc[:,i])
        print('{0} person_cof= {1} and p value = {2} '.format(keys[i],person_cof,p_value))
        if person_cof>0.7:
            corel_points.append(keys[i])
```

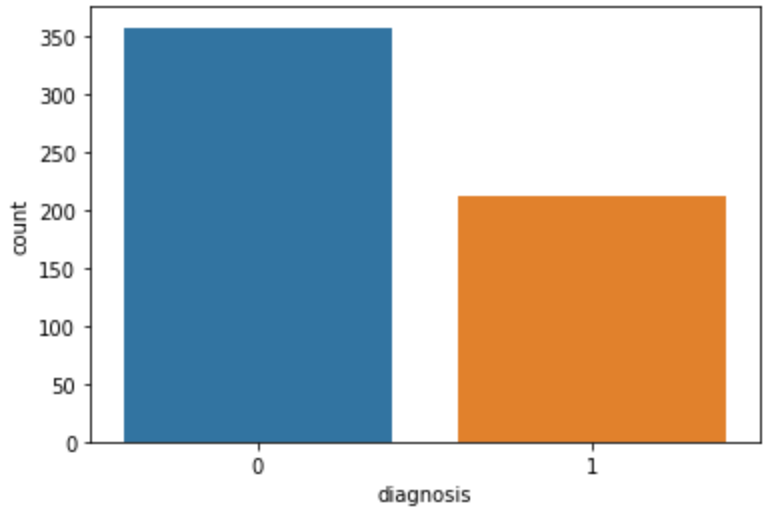
```
radius_mean person_cof= 0.7300285113754564 and p value = 8.465940572258751e-96
texture_mean person_cof= 0.4151852998452044 and p value = 4.058636047896155e-25
perimeter_mean person_cof= 0.742635529725833 and p value = 8.436251036168899e-101
area_mean person_cof= 0.70898383658539 and p value = 4.734564310304486e-88
smoothness_mean person_cof= 0.3585599650859321 and p value = 1.051850359202694e-18
compactness_mean person_cof= 0.5965336775082533 and p value = 3.9382631058850314e-56
concavity_mean person_cof= 0.6963597071719059 and p value = 9.966555755066088e-84
concave points_mean person_cof= 0.7766138400204354 and p value = 7.101150161054297e-116
symmetry_mean person_cof= 0.33049855426254715 and p value = 5.733384028463847e-16
fractal_dimension_mean person_cof= -0.012837602698432378 and p value = 0.7599368037251972
radius_se person_cof= 0.5671338208247177 and p value = 9.738948656455998e-50
texture_se person_cof= -0.008303332973877427 and p value = 0.8433320287665917
perimeter_se person_cof= 0.5561407034314831 and p value = 1.6519051758489454e-47
area_se person_cof= 0.5482359402780244 and p value = 5.895521392602765e-46
smoothness_se person_cof= -0.06701601057948733 and p value = 0.11029660865782694
compactness_se person_cof= 0.29299924424885837 and p value = 9.975994654069982e-13
concavity_se person_cof= 0.25372976598083036 and p value = 8.260176167965638e-10
concave points_se person_cof= 0.4080423327165046 and p value = 3.0723087688164388e-24
symmetry_se person_cof= -0.006521755870647961 and p value = 0.8766418183854183
fractal_dimension_se person_cof= 0.07797241739025615 and p value = 0.06307355082235527
radius_worst person_cof= 0.7764537785950396 and p value = 8.482291921679676e-116
texture_worst person_cof= 0.4569028213967983 and p value = 1.0780574879487261e-30
perimeter_worst person_cof= 0.7829141371737594 and p value = 5.771397139665565e-119
area_worst person_cof= 0.7338250349210511 and p value = 2.828847704284965e-97
smoothness_worst person_cof= 0.42146486106640263 and p value = 6.575143633980474e-26
compactness_worst person_cof= 0.590998237841792 and p value = 7.06981635253457e-55
concavity_worst person_cof= 0.659610210369233 and p value = 2.4646639567816386e-72
concave points_worst person_cof= 0.79356601714127 and p value = 1.9690997072156805e-124
symmetry_worst person_cof= 0.416294311048619 and p value = 2.9511205771522968e-25
fractal_dimension_worst person_cof= 0.3238721887208239 and p value = 2.3164324499817004e-15
```

```
corel_points.append('diagnosis')
data=dataset[corel_points]
data.head()
```

	radius_mean	perimeter_mean	area_mean	concave points_mean	radius_worst	perimeter_worst	area_worst	concave points_worst	diagnosis
0	17.99	122.80	1001.0	0.14710	25.38	184.60	2019.0	0.2654	1
1	20.57	132.90	1326.0	0.07017	24.99	158.80	1956.0	0.1860	1
2	19.69	130.00	1203.0	0.12790	23.57	152.50	1709.0	0.2430	1
3	11.42	77.58	386.1	0.10520	14.91	98.87	567.7	0.2575	1
4	20.29	135.10	1297.0	0.10430	22.54	152.20	1575.0	0.1625	1

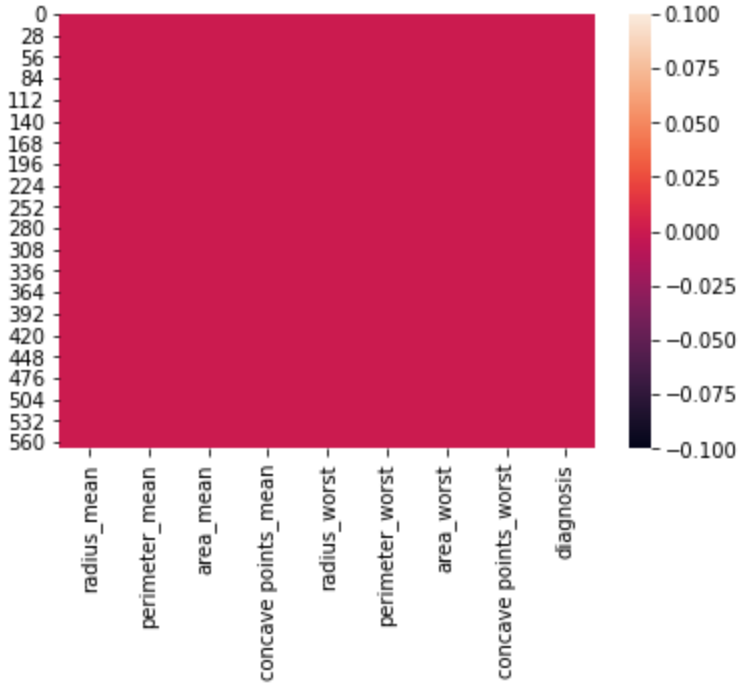
```
sns.countplot(x='diagnosis',data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef6414a320>
```



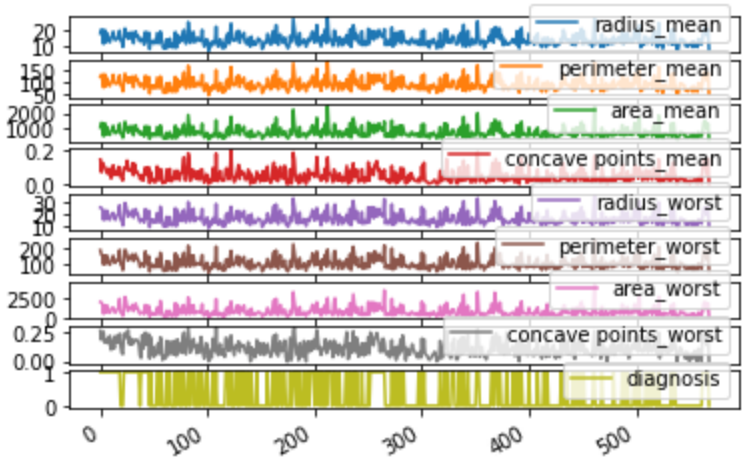

```
sns.heatmap(data.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef5f8f8630>
```



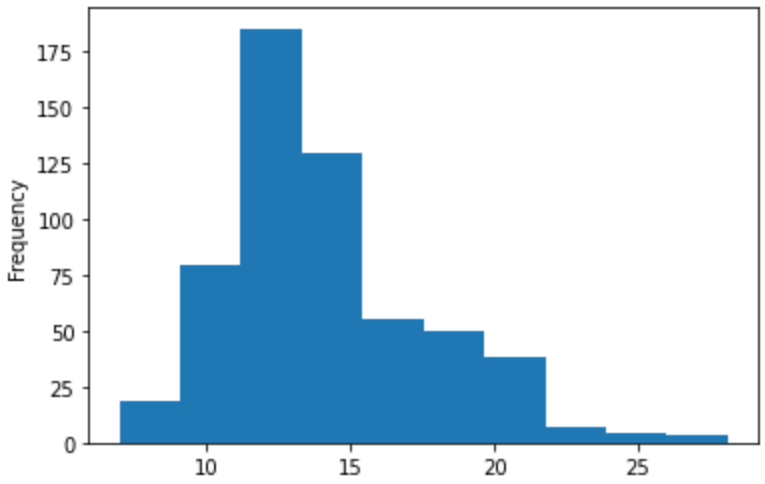

```
1 data.plot(kind='line',subplots=True)
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e8bfa20>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e872cc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e828cc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e7ddcc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e793cc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e7c7cc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e77ecc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e734c88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fef5e734cf8>],
      dtype=object)
```



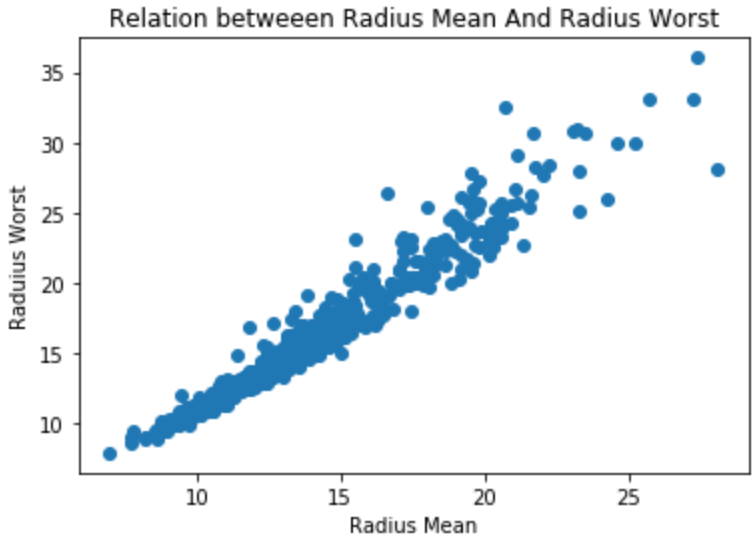

```
1 data['radius_mean'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef5e5be2e8>
```




```
1 plt.scatter(dataset['radius_mean'],dataset['radius_
```

```
  worst'])
2 plt.xlabel('Radius Mean')
3 plt.ylabel('Raduius Worst')
4 plt.title('Relation betweeen Radius Mean And Radius
  Worst')
5 plt.show()
```




```
1 data_in=data.iloc[:,:-1]
2 data_out=data.iloc[:,-1]
```

```
1 from sklearn.preprocessing import MinMaxScaler
2 from sklearn.model_selection import
  train_test_split
```

```
1 sc=MinMaxScaler(feature_range=(0,1))
2 data_preprocess=sc.fit_transform(data_in)
3 final_keys=data_in.keys()
```

```
dicti={}
for i in range(len(final_keys)):

    dicti.update({final_keys[i]:data_preprocess[:,i]})
dataset=pd.DataFrame(dicti)
dataset.head()
```

	radius_mean	perimeter_mean	area_mean	concave points_mean	radius_worst	perimeter_worst	area_worst	concave points_worst
0	0.521037	0.545989	0.363733	0.731113	0.620776	0.668310	0.450698	0.912027
1	0.643144	0.615783	0.501591	0.348757	0.606901	0.539818	0.435214	0.639175
2	0.601496	0.595743	0.449417	0.635686	0.556386	0.508442	0.374508	0.835052
3	0.210090	0.233501	0.102906	0.522863	0.248310	0.241347	0.094008	0.884880
4	0.629893	0.630986	0.489290	0.518390	0.519744	0.506948	0.341575	0.558419

```
final_data=pd.concat([data.iloc[:,-1],dataset],axis=1)
train,test=train_test_split(final_data,test_size=0.2)
```

```
import boto3,re,os,json,sagemaker
from sagemaker import get_execution_role
```

```
role=get_execution_role()
my_region=boto3.session.Session().region_name
```

```
containers = {'us-west-2': '433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest',
              'us-east-1': '811284229777.dkr.ecr.us-east-1.amazonaws.com/xgboost:latest',
```

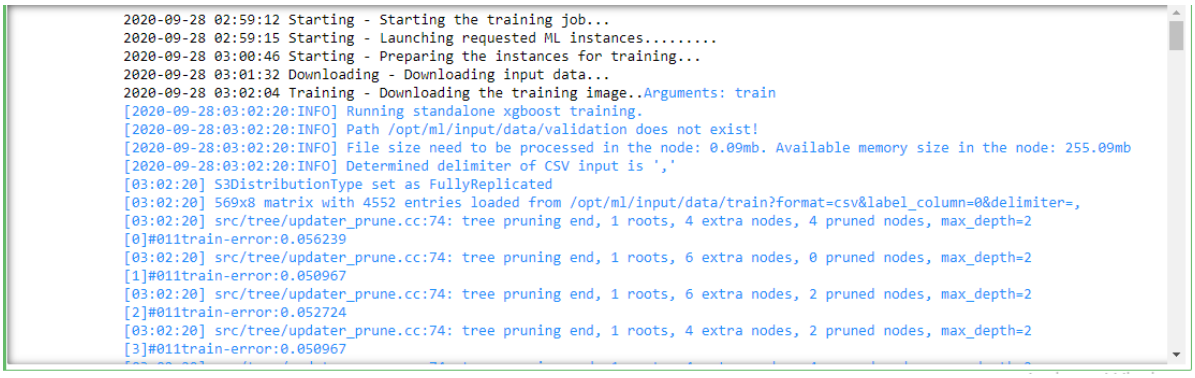
```
3                'us-east-2':
  '825641698319.dkr.ecr.us-east-2.amazonaws.com/xgboo
  st:latest',
4                'eu-west-1':
  '685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboo
  st:latest'}
```

```
1 prefix='sagemaker/Brest-Cancer'
2 bucket_name='buildathonproject1'
```

```
1 final_data.to_csv('train.csv',index=False,header=Fa
  lse)
2 boto3.Session().resource('s3').Bucket(bucket_name).
  Object(os.path.join(prefix,'train/train.csv')).uplo
  ad_file('train.csv')
3 s3_input_train=sagemaker.s3_input(s3_data='s3://{}/
  {}/train'.format(bucket_name,
  prefix),content_type='csv')
```

```
1 sess=sagemaker.Session()
2 brest_cancer_model=sagemaker.estimator.Estimator(co
  ntainers[my_region],role,train_instance_count=1,tra
  in_instance_type='ml.m5.large',output_path='s3://{}
  /{}/output'.format(bucket_name,prefix),sagemaker_se
  ssion=sess)
3 brest_cancer_model.set_hyperparameters(max_depth=5,
  eta=0.2,gamma=4,min_child_weight=6,subsample=0.8,si
  lent=0,objective='binary:logistic',num_round=100)
```

```
1 brest_cancer_model.fit({'train':s3_input_train})
```



```
2020-09-28 02:59:12 Starting - Starting the training job...
2020-09-28 02:59:15 Starting - Launching requested ML instances.....
2020-09-28 03:00:46 Starting - Preparing the instances for training...
2020-09-28 03:01:32 Downloading - Downloading input data...
2020-09-28 03:02:04 Training - Downloading the training image..Arguments: train
[2020-09-28:03:02:20:INFO] Running standalone xgboost training.
[2020-09-28:03:02:20:INFO] Path /opt/ml/input/data/validation does not exist!
[2020-09-28:03:02:20:INFO] File size need to be processed in the node: 0.09mb. Available memory size in the node: 255.09mb
[2020-09-28:03:02:20:INFO] Determined delimiter of CSV input is ','
[03:02:20] S3DistributionType set as FullyReplicated
[03:02:20] 569x8 matrix with 4552 entries loaded from /opt/ml/input/data/train?format=csv&label_column=0&delimiter=,
[03:02:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 4 pruned nodes, max_depth=2
[0]#011train-error:0.056239
[03:02:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 6 extra nodes, 0 pruned nodes, max_depth=2
[1]#011train-error:0.050967
[03:02:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 6 extra nodes, 2 pruned nodes, max_depth=2
[2]#011train-error:0.052724
[03:02:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 2 pruned nodes, max_depth=2
[3]#011train-error:0.050967
```

```
1 detector=brest_cancer_model.deploy(initial_instance
  _count=1,instance_type='ml.m5.large')
```

```
Parameter image will be renamed to image_uri in SageMaker Python SDK v2.
-------------!
```

```
1 detector.endpoint
```

```
'xgboost-2020-09-28-02-59-12-505'
```

```
1 from sagemaker.predictor import csv_serializer
2 test_data_array=test.drop('diagnosis',axis=1).value
  s #load the data into an array
3 detector.content_type = 'text/csv' # set the data
  type for an inference
4 detector.serializer = csv_serializer # set the
  serializer type
5 predictions=detector.predict(test_data_array).decod
  e('utf-8') # predict!
6 predictions_array = np.fromstring(predictions[1:],
  sep=',')
7 print(predictions)
```

0.984164893627,0.984164893627,0.984164893627,0.00840793922544,0.00840793922544,0.984164893627,0.164673551917,0.00840793922544,
0.698992073536,0.984164893627,0.00840793922544,0.00840793922544,0.00840793922544,0.00840793922544,0.00840793922544,0.9841648936
27,0.00840793922544,0.984164893627,0.0126698166132,0.81916064024,0.984164893627,0.00840793922544,0.00840793922544,0.00840793922
544,0.984164893627,0.00840793922544,0.880655944347,0.00840793922544,0.980865299702,0.00840793922544,0.984164893627,0.0269436519
593,0.119824945927,0.00840793922544,0.733735799789,0.00840793922544,0.984164893627,0.0277226809412,0.984164893627,0.03022056818
01,0.00840793922544,0.00840793922544,0.0277226809412,0.0715783983469,0.975009799004,0.0317783281207,0.477484881878,0.0084079392
2544,0.984164893627,0.00840793922544,0.950887620449,0.00840793922544,0.00840793922544,0.984164893627,0.0269436519593,0.98416489
3627,0.0317783281207,0.00840793922544,0.00840793922544,0.0180823151022,0.00840793922544,0.950887620449,0.340621441603,0.9750097
99004,0.980555832386,0.00840793922544,0.980555832386,0.984164893627,0.00840793922544,0.929939746857,0.00840793922544,0.00840793
922544,0.984164893627,0.950887620449,0.582483589649,0.0180823151022,0.00840793922544,0.00840793922544,0.00840793922544,0.211229
816079,0.0201756022871,0.984164893627,0.00840793922544,0.137797668576,0.0126698166132,0.00840793922544,0.0492947809398,0.008407
93922544,0.966929972172,0.00840793922544,0.00840793922544,0.984164893627,0.0634867548943,0.175529614091,0.984164893627,0.051321
6592371,0.00840793922544,0.942702412605,0.00840793922544,0.00840793922544,0.00840793922544,0.00840793922544,0.00840793922544,0.303190141916,0.01
26698166132,0.984164893627,0.00840793922544,0.154683455825,0.00840793922544,0.984164893627,0.0277226809412,0.00840793922544,0.0
0840793922544,0.0277226809412,0.0180823151022

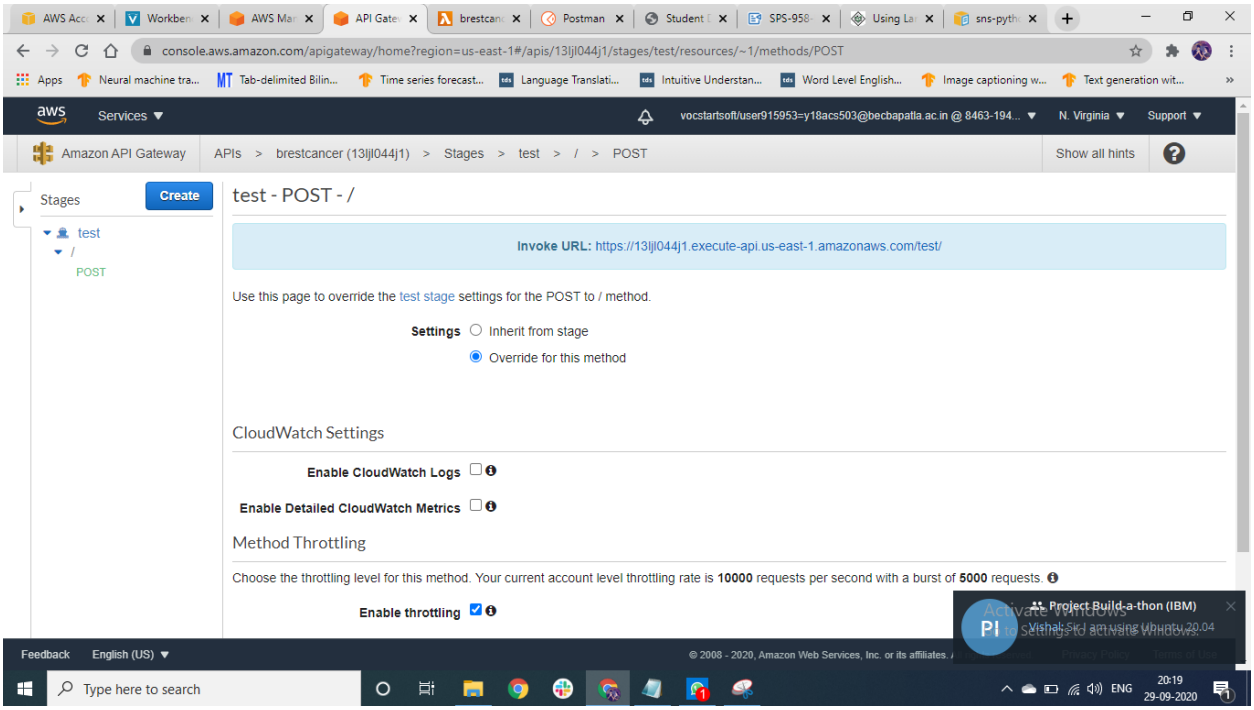
LambdaCode:

```
1 import os
2 import io
3 import boto3
4 import json
5 import csv
6 def lambda_handler(event, context):
7     ENDPOINT_NAME =
  os.environ['environment_variable']
8     runtime= boto3.client('runtime.sagemaker')
9     print(ENDPOINT_NAME)
10    print("Received event: " , json.dumps(event,
  indent=2))
11    data = json.loads(json.dumps(event))
12    print("Data:",data)
13    payload = data['data']
14    print("Payload:",payload)
15    response =
  runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
16
  ContentType='text/csv',
17
```

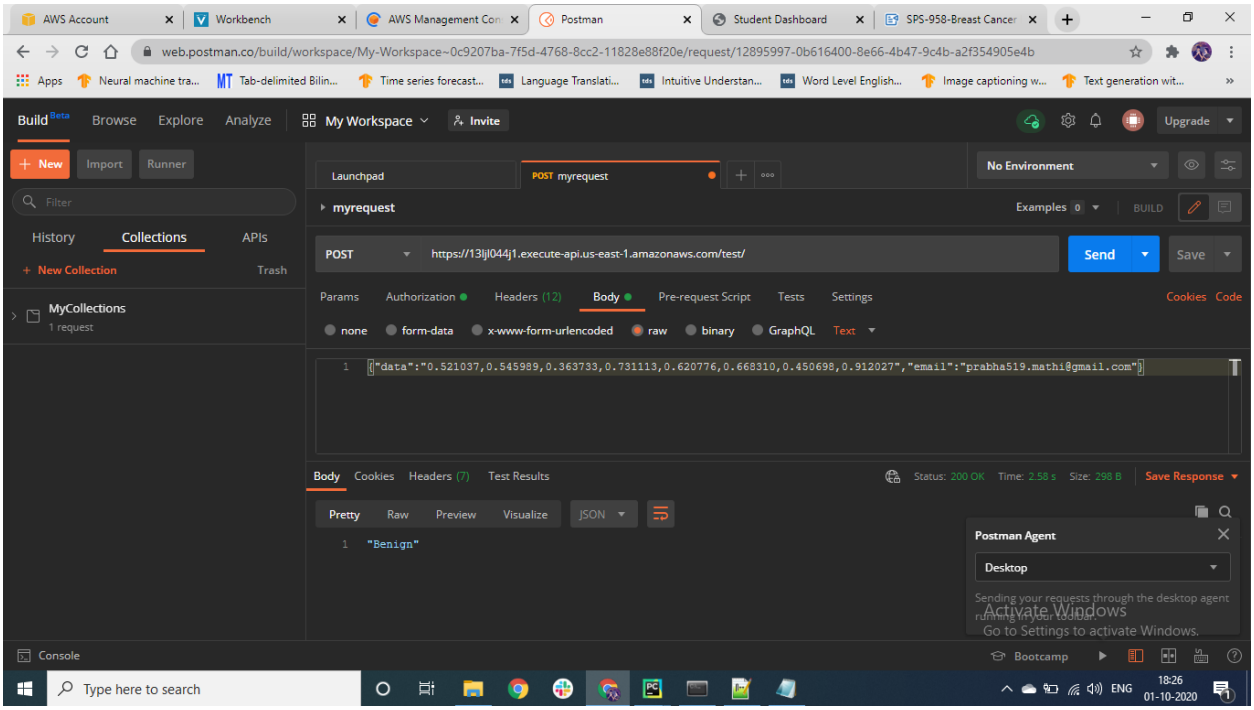
```
  Body=payload)
18    result =
  json.loads(response['Body'].read().decode())
19    sns = boto3.client('sns')
20
  sub=sns.subscribe(TopicArn='arn:aws:sns:us-east-1:8
  46319470919:MyFirstNotification',Protocol='email',E
  ndpoint=data['email'])
21
22    print(result)
23
24    if result>0.5:
25        response =
  sns.publish(TopicArn='arn:aws:sns:us-east-1:8463194
  70919:MyFirstNotification',Message='The Result For
  BrestCancer is Bengin :Positive',)
26        return "Benign"
27    else:
28        response =
  sns.publish(TopicArn='arn:aws:sns:us-east-1:8463194
  70919:MyFirstNotification',Message='The Result For
  BrestCancer is Malignant :Negitive',)
29        return "Malignant"
```

```
Response:
"Benign"
```

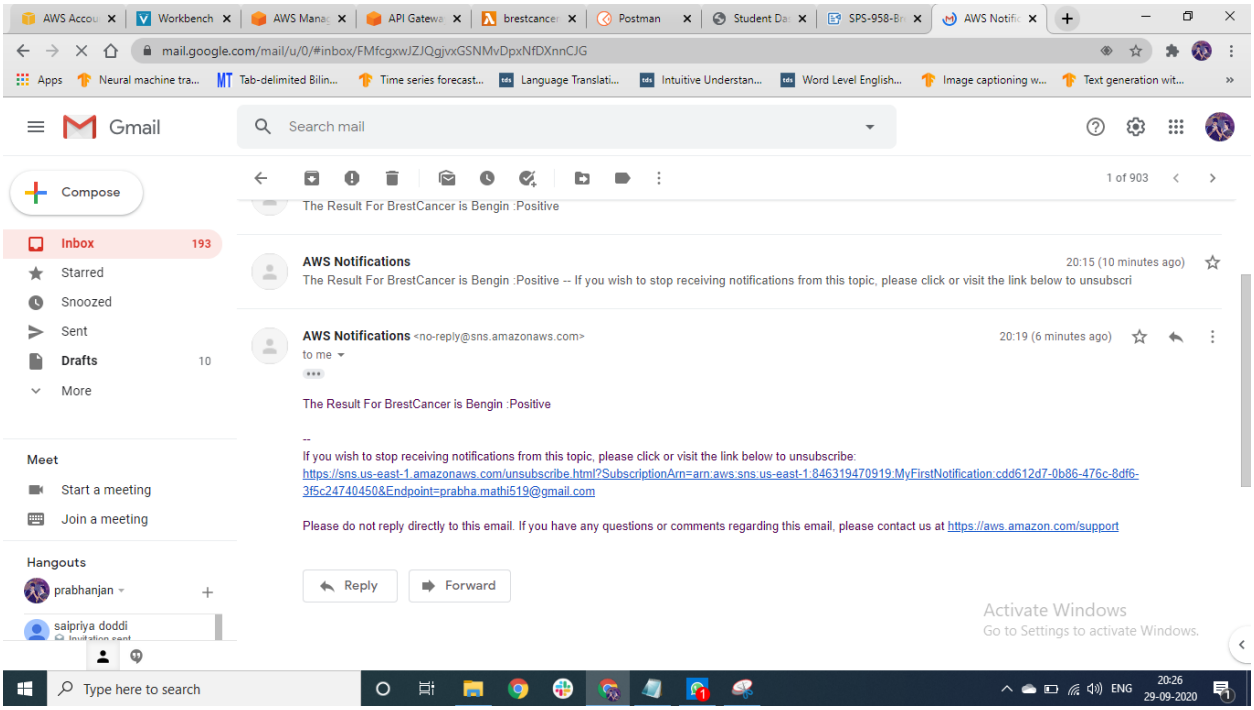
Creation Of Api:


console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/13ljl044j1/stages/test/resources/~1/methods/POST
Services
Amazon API Gateway
APIs > brestcancer (13ljl044j1) > Stages > test > / > POST
Show all hints
Stages
Create
test
/
POST
test - POST - /
Invoke URL: https://13ljl044j1.execute-api.us-east-1.amazonaws.com/test/
Use this page to override the test stage settings for the POST to / method.
Settings
Inherit from stage
Override for this method
CloudWatch Settings
Enable CloudWatch Logs
Enable Detailed CloudWatch Metrics
Method Throttling
Choose the throttling level for this method. Your current account level throttling rate is 10000 requests per second with a burst of 5000 requests.
Enable throttling
Project Build-a-thon (IBM)
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates.
Type here to search
ENG
20:19
29-09-2020

Testing Api:


web.postman.co/build/workspace/My-Workspace~0c9207ba-7f5d-4768-8cc2-11828e88f20e/request/12895997-0b616400-8e66-4b47-9c4b-a2f354905e4b
Build
Browse
Explore
Analyze
My Workspace
Invite
Upgrade
New
Import
Runner
History
Collections
APIs
New Collection
Trash
MyCollections
1 request
Launchpad
POST myrequest
No Environment
myrequest
Examples 0
BUILD
POST
https://13ljl044j1.execute-api.us-east-1.amazonaws.com/test/
Send
Save
Params
Authorization
Headers (12)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Text
{"data":"0.521037,0.545989,0.363733,0.731113,0.620776,0.668310,0.450698,0.912027","email":"prabha519.mathi@gmail.com"}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 2.58 s
Size: 298 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
"Benign"
Postman Agent
Desktop
Sending your requests through the desktop agent
Activate Windows
Go to Settings to activate Windows.
Console
Bootcamp
Type here to search
ENG
18:26
01-10-2020

EmailNotification:



UI:

hello.py

```
from flask import
Flask,render_template,request,url_for
import requests
app=Flask(__name__)
@app.route('/',methods=['POST','GET'])
def hello():
    if request.method=='POST':
        radme=request.form['a']
        permn=request.form['b']
        armn=request.form['c']
        compm=request.form['d']
        radw=request.form['e']
        perw=request.form['f']
```

```
13          arw=request.form['g']
14          copw=request.form['h']
15          email=request.form['i']
16          try:
17              radme=float(radme)
18              permn=float(permn)
19              armn=float(armn)
20              compm=float(compm)
21              radw=float(radw)
22              perw=float(perw)
23              arw=float(arw)
24              copw=float(copw)
25              email=str(email)
26          except:
27              return
    render_template('data.html',err_msg='Enter Valid
    Data')
28          url =
    "https://13ljl044j1.execute-api.us-east-1.amazonaws
    .com/test/"
29          payload = " {\"data\":\" " + str(radme) +
    ',' + str(permn) + ',' + str(armn) + ',' +
    str(compm) + ',' + str(radw) + ',' + str(perw) +
    ',' + str(arw) + ',' + str(copw) + "\"" + ','+
    "\"email\"" +":\""+email+"\""+"}"
30          print(payload)
31
32          headers = {
33              'X-Amz-Content-Sha256':
    'beaead3198f7da1e70d03ab969765e0821b24fc913697e929e
    726aeaebf0eba3',
```

```
            'X-Amz-Date': '20200930T095337Z',
            'Authorization': 'AWS4-HMAC-SHA256
  Credential=ASIA4KDESJFDUSSQKJSG/20200930/us-east-1/
  execute-api/aws4_request,
  SignedHeaders=host;x-amz-content-sha256;x-amz-date,
  Signature=b81935cc533d5efb8db465da9c12f4a3ed76ca800
  89dfc3edebdb39df4fe5f7c',
            'Content-Type': 'text/plain'
        }

        response = requests.request("POST", url,
  headers=headers, data=payload)
        response=response.text.encode('utf8')
        response=str(response)
        print(response)
        result=response[3:-2]
        print(result)
        return
  render_template('data.html',result=result)
    else:
        return render_template('data.html')

if __name__ == '__main__':
    app.run(debug=True)

```

data.html:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <title>Sample Form</title>
     <style>
         .brest
         {
             width:300px;
             height:25px;
             background-color:#fffbfc;
             border-style: ridge;
             border-color:gray;
             border-radius:6px;
         }
         body
         {
             font-family:sans-serif;
             background-image: url('{{
  url_for('static', filename='back.png') }}');
         }
         #sub
         {
             width:200px;
             height:25px;
             background-color:#9f6cff;
             border-style: ridge;
             border-color:gray;
             border-radius:6px;
         }
    </style>
</head>
<body style="background-color:#575143">
    <center>
```

```
33          {% if result %}
34          <p style="color:red;font-size:30px;">
  Result: {{result}}</p>
35          {% endif %}
36          {% if err_msg %}
37          <p
  style="color:red;font-size:15px;">{{err_msg}}</p>
38          {% endif %}
39          <br/>
40          <form method="post" action="/">
41              <input type="text" name="a"
  class="brest" placeholder="   Enter Radius Mean"
  required><br/><br/>
42              <input type="text" name="b"
  class="brest" placeholder="   Enter Perimeter "
  required><br/><br/>
43              <input type="text" name="c"
  class="brest" placeholder="   Enter Area Mean"
  required><br/><br/>
44              <input type="text" name="d"
  class="brest" placeholder="   Enter Concave Point
  Mean" required><br/><br/>
45              <input type="text" name="e"
  class="brest" placeholder="   Enter Radius Worst"
  required><br/><br/>
46              <input type="text" name="f"
  class="brest" placeholder="   Enter Perimeter
  Worst" required><br/><br/>
47              <input type="text" name="g"
  class="brest" placeholder="   Enter Area Worst"
  required><br/><br/>
```

```
48            <input type="text" name="h"
  class="brest" placeholder="   Enter Concave Point
  Worst" required><br/><br/>
49            <input type="email" name="i"
  class="brest" placeholder="  Enter Email Id"
  required><br/><br/>
50            <input type="submit" value="Submit"
  id="sub">
51        </form>
52    </center>
53</body>
54</html>
```