Cloud-Based Vehicle Loan Default Prediction

Malavika Sreedhar Tankala
Student, Dept. of Computer Science and Engineering
Santa Clara University
Santa Clara, USA
mtankala@scu.edu
W1628801

Sreevikram Chandrasekhar
Student, Dept. of Computer Science and Engineering
Santa Clara University
Santa Clara, USA
schandrasekhar@scu.edu
W1629488

Jil Dipakkumar Patel
Student, Dept. of Computer Science and Engineering
Santa Clara University
Santa Clara, USA
jdpatel@scu.edu
W1607219

Shubham Singla
Student, Dept. of Computer Science and Engineering
Santa Clara University
Santa Clara, USA
ssingla@scu.edu
W1628734

Project Link: https://github.com/shubhamsingla27/CloudComputing-LoanPredictor-Project

***Abstract*— Loans are one of the most important aspects of the finance world. One of the major sources of profit for financial institutions comes from the same. When a customer applies for a loan, the concerned organization has to review multiple parameters. This process is very time-consuming which increases the workload within the organization. The objective is to predict whether customers are capable of repaying the loan. This paper proposes the review process to be automated using a Machine Learning Model hosted on the cloud. This project has achieved the same using the various services provided by the Amazon Web Services.**

***Key words*: Machine Learning, Amazon Web Services.**

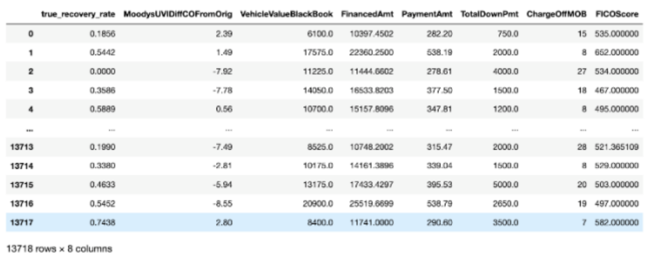
I. INTRODUCTION

Loan lending plays a major role in the growth of spending and the economy in our daily lives. It has always been essential for people to take out loans because people all over the world depend on them to surmount financial obstacles and accomplish their personal goals, as well as because businesses depend on them to increase their output. Most of the time, giving loans is advantageous to both lenders and debtors. Loan failure, which carries a lot of risk and might even result in a financial catastrophe, is still a chance. Determining a candidate's credit eligibility is therefore extremely important. In the past, the majority of the assessment was performed manually, which required a lot of time and effort. Recently, banks have chosen machine learning techniques to forecast loan failures automatically due to their ability to significantly increase projection efficiency and accuracy. Financial organizations now have access to a sizable quantity of transaction data thanks to the popularity of smartphone purchases and online purchasing.

II. BACKGROUND AND RELATED WORK

1. Dataset

In order to get the accurate prediction value, we need to train the model on a huge volume of true dataset. Once the model is trained, we need to verify whether the model is able to predict the correct value on given test data. If we get higher accuracy on test data then we say that model is performing well and we are ready to deploy it. We used the Crescent Bank test data set obtained from hackathon.



	true_recovery_rate	MoodysUVIDiffCOFromOrig	VehicleValueBlackBook	FinancedAmt	PaymentAmt	TotalDownPmt	ChargeOffMOB	FICOScore
0	0.1856	2.39	6100.0	10397.4502	282.20	750.0	15	535.000000
1	0.5442	1.49	17575.0	22360.2500	538.19	2000.0	8	652.000000
2	0.0000	-7.92	11225.0	11444.6602	278.61	4000.0	27	534.000000
3	0.3586	-7.78	14050.0	16533.8203	377.50	1500.0	18	467.000000
4	0.5889	0.56	10700.0	15157.8096	347.81	1200.0	8	495.000000
...	...	...	...	...	...	...	...	...
13713	0.1990	-7.49	8525.0	10748.2002	315.47	2000.0	28	521.365109
13714	0.3380	-2.81	10175.0	14161.3896	339.04	1500.0	8	529.000000
13715	0.4633	-5.94	13175.0	17433.4297	395.53	5000.0	20	503.000000
13716	0.5452	-8.55	20900.0	25519.6699	538.79	2650.0	19	497.000000
13717	0.7438	2.80	8400.0	11741.0000	290.60	3500.0	7	582.000000

13718 rows × 8 columns

In the above data we have 7 Parameters:

MoodysUVIDiffCOFromOrig: Difference between resale Moddy index and original Moody index.

VehicleValueBlackBook : The collateral value.

FinancedAmt: The amount a person is requesting for a loan.

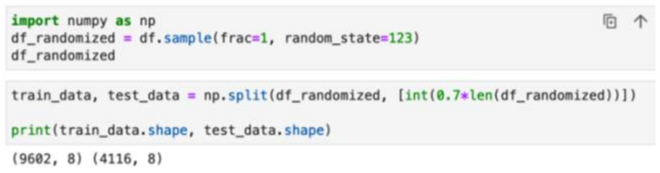
PaymentAmt: Monthly payment

TotalDownPmt: Initial Down payment

ChargeOffMOB: Grace period

FICOScore: Credit score

Below is the screenshot of how we have split our dataset. As explained above, we used 70% of the dataset to train our model which is 9602 records. Also, we used 30% of the dataset to test the behavior of the model.



```
import numpy as np
df_randomized = df.sample(frac=1, random_state=123)
df_randomized
```

```
train_data, test_data = np.split(df_randomized, [int(0.7*len(df_randomized))])

print(train_data.shape, test_data.shape)
(9602, 8) (4116, 8)
```

2. AWS S3

   Customers can store and access any amount of data using the highly scalable, dependable, and secure object storage service known as AWS S3. In S3, an object is a bucket-stored collection of data, a key, and metadata. Durability, affordability, scalability, security, availability, and connection with other Amazon services are just a few advantages that S3 offers. The dataset is kept in an S3 bucket in this instance.

3. Jupyter Instance to Develop ML Algorithm

   The link between a dependent variable, also known as the response variable, and one or more independent variables, also known as explanatory variables or predictors, is modelled using the statistical method of linear regression. The main aim of linear regression is to find a straight line that minimizes the sum of squared discrepancies between the observed and projected values. Y = mx + b is the equation for the line, where m is its slope, y is its dependent variable, x is its independent variable, and y is its y-intercept. The slope of the line represents the change in y for each unit increment in x. Several disciplines, including engineering, finance, biology, and economics, commonly use linear regression. Predictive models are created using the well-liked machine learning technique XGBoost (Extreme Gradient Boosting). It is a member of the family of gradient boosting algorithms, which repeatedly train and merge a number of weak predictive models to produce a powerful model. A machine learning approach called XGBoost is employed for regression problems in which the objective is to forecast a continuous numerical value. Implementing XGBoost for regression requires the following fundamental steps: loading and preprocessing the data, defining the model, developing the model, testing the model, fine-tuning the model, and making predictions. A strong and effective method that can handle huge datasets with high-dimensionality characteristics is XGBoost.

4. AWS Sagemaker

   Fully managed machine learning software is available through Amazon SageMaker. Machine learning models may be rapidly and simply built, trained, and deployed into a hosted environment that is prepared for production by data scientists and developers. Data sources may be quickly accessed for exploration and analysis thanks to the integrated Jupyter writing notebook instance. It also offers popular machine learning techniques that have been enhanced to function well in a distributed setting against very big data sets. As it is done in this application, SageMaker integrates with AWS S3 to store the dataset and utilizes an Amazon EC2 instance to train the models.

5. AWS API Gateway

   The managed AWS API Gateway service makes it simple to create, publish, maintain, monitor, and secure APIs on a large scale. Supporting the creation of WebSocket APIs and RESTful APIs is one of AWS API Gateway's key capabilities. Web applications and containerized and serverless workloads are also supported by AWS API Gateway. The API Gateway service was used to connect the application's front end to the back end in this paper.

6. AWS Lambda

   AWS Lambda, a serverless processing tool, is provided by Amazon Web Services (AWS). The computing duties that the Lambda functions can perform include serving web sites, handling data streams, contacting APIs, and integrating with other AWS services.

   The phrase "serverless" computing alludes to the fact that none of these activities require managing one's own servers. It means that the infrastructure, including the servers, operating systems, network layer, and other components, has already been taken care of, freeing you to focus on writing application code.

7. AWS Amplify

   AWS Amplify is a development tool that creates, distributes, and hosts full-stack apps using the cloud architecture of Amazon Web Services (AWS).
   The adaptable setting provided by AWS Amplify facilitates application creation, testing, and implementation. By utilizing the broad range of AWS services, it enables developers to build apps that are scalable and reliable.

   With AWS Amplify, developers can build both the front end and the back end of their apps. The infrastructure is compatible with React, Angular, Vue, and other well-known front-end frameworks. Additionally, it supports a variety of back-end technologies, including Python, Java, Go, and Node.js.The front-end website for this study, which is linked to the AWS API Gateway, was hosted by AWS Amplify.

8. AWS Cognito

For both online and mobile applications, the Amazon Cognito service provides APIs and infrastructure that make it easier to do crucial user management tasks including user repository maintenance, authentication, and authorisation. User Pool and Identity Pool make up its two primary parts; the former offers sign-up and sign-in choices and serves as a user directory, while the latter grants access to other Amazon services.

With capabilities like username and password storage, session management, and forgotten password recovery, Amazon Cognito enables developers to quickly add user sign-up, sign-in, and access control to their apps. In general, Amazon Cognito provides a complete user authentication solution for online and mobile apps.

9. AWS Simple Notification Service

You can deliver messages or alerts to endpoints or clients that have subscribed using Amazon Web Services' AWS SNS (Simple Notification Service). Many protocols, including HTTP, HTTPS, Email, SMS, Lambda function, and others are supported. You may build topics using Amazon SNS, which are channels for sending messages to endpoints or clients that have subscribed. Endpoints include email addresses, HTTP/S webhooks, Lambda functions, SMS or push notifications sent to mobile devices, and Amazon SQS queues.
When a message is sent to a topic, SNS sends it to any clients or endpoints who have subscribed to that subject and shown an interest in receiving messages from it. SNS provides message filtering based on qualities, enabling you to deliver targeted messages to particular people.

10. AWS Simple Email Service

You may send and receive email using your own email addresses and domains with the help of AWS SES, a service offered by Amazon Web Services.

Using the SMTP (Simple Mail Transfer Protocol) interface or the API, you may use Amazon SES to deliver transactional or marketing email messages to your clients or subscribers. Also, you may receive emails from your clients or subscribers that are sent to an Amazon S3 bucket or an Amazon SNS subject.

11. Related Work

For this project, related work was found in [1],[2],[3],[4] where they implemented various Machine Learning models and libraries including but not limited to Logistic Regression, Decision Trees, Random Forest and Support Vector Machine.

In comparison, although there might be variances in the performance of the models within these papers, our implementation uses the Linear Regression model and has a significant advantage as the application is hosted on the cloud.
This allows the reviewing financial institutions to leverage cloud specific features like scalability, high-availability, security, reliability and cost-effectiveness.

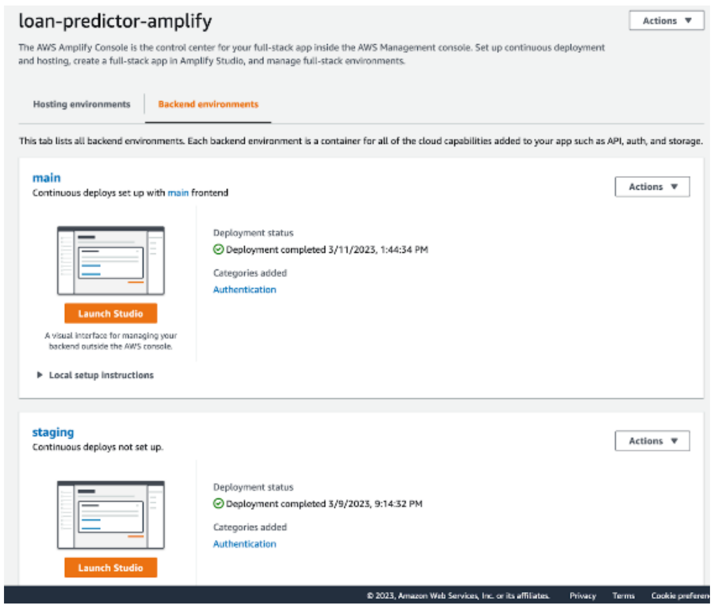
III. APPROACH

For the front-end side of this project, a webpage was created using HTML, CSS, Javascript and React.js that requires personal and financial information of the loan applicant . The personal and financial information are categorized as follows:

1. Personal information: Name, Email.
2. Financial information: Account number, Moody value, Vehicle value, Financed amount, Payment amount, Down payment, Charge value and FICO score.

<u>AWS Amplify</u>

AWS Amplify was used to host the webpage on AWS. Amplify provides us with the options to host frontend using Hosting environment as well as the Backend Environment. In the Backend environment we configured the authentication which behind the scenes uses Amazon Cognito to create user pools and in turn authenticate registered users as well as enable new registrations. Afterwards, using the Amplify CLI we pulled the configuration into the React project and hosted the same on the Amplify Hosting environment using a CI/CD (Continuous Deployment) pipeline through a GitHub repository.



loan-predictor-amplify

The AWS Amplify Console is the control center for your full-stack app inside the AWS Management console. Set up continuous deployment and hosting, create a full-stack app in Amplify Studio, and manage full-stack environments.

This tab lists all backend environments. Each backend environment is a container for all of the cloud capabilities added to your app such as API, auth, and storage.

main
Continuous deploys set up with main frontend

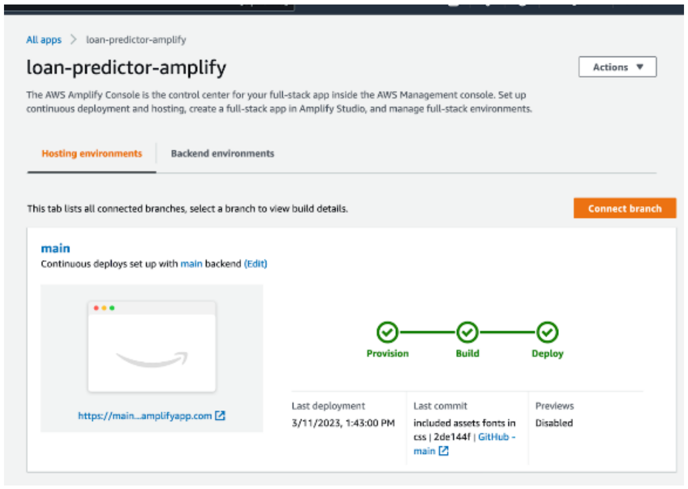
A visual interface for managing your backend outside the AWS console.

Deployment status
Deployment completed 3/11/2023, 1:44:34 PM
Categories added
Authentication

staging
Continuous deploys not set up.

Deployment status
Deployment completed 3/9/2023, 9:14:32 PM
Categories added
Authentication

© 2023, Amazon Web Services, Inc. or its affiliates.



All apps > loan-predictor-amplify

loan-predictor-amplify

The AWS Amplify Console is the control center for your full-stack app inside the AWS Management console. Set up continuous deployment and hosting, create a full-stack app in Amplify Studio, and manage full-stack environments.

This tab lists all connected branches, select a branch to view build details.

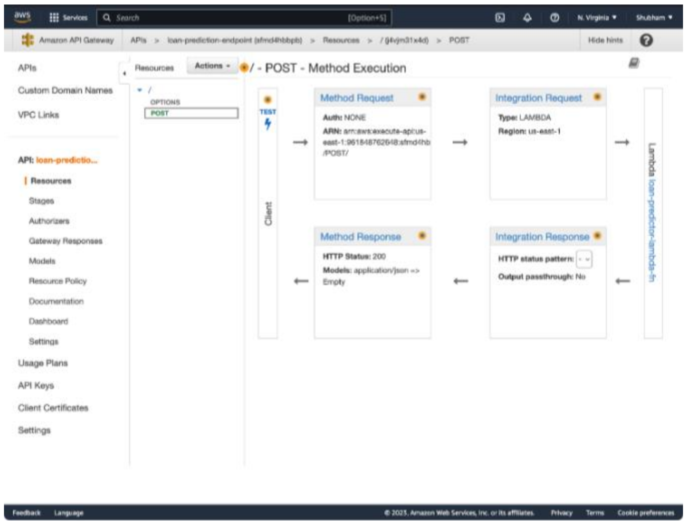
main
Continuous deploys set up with main backend (Edit)

https://main...amplifyapp.com

Last deployment	Last commit	Previews
3/11/2023, 1:43:00 PM	included assets fonts in css | 2de144f | GitHub - main	Disabled

AWS API Gateway

We initialized the API gateway by creating a REST API protocol with a regional endpoint type and created a POST method. We integrated the API Gateway with Amplify, and the AWS API Gateway was used to trigger the Lambda function. The reason we need to use lambda function and not directly invoke the Sagemaker endpoint through API gateway is, Sagemaker endpoint expects specific format of input upon which it can predict the value. So when we connect API Gateway to lambda function, it passes the information in an event. The lambda function will serialize the data from the event and will invoke the Sagemaker endpoint. By using the POST method, the customer specific input information from Amplify was passed onto Lambda function via API Gateway as a JSON object.



Amazon API Gateway > APIs > loan-prediction-endpoint (sfmd4hbbpb) > Resources > / (j4vjm31x4d) > POST

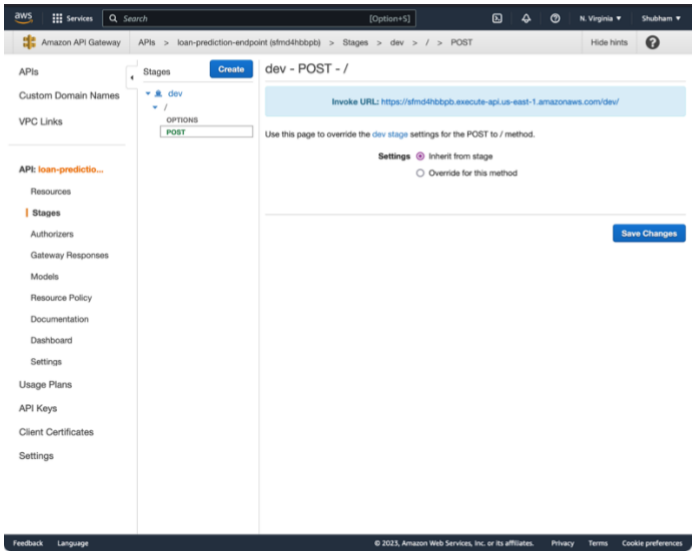
/ - POST - Method Execution

Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-east-1:961848762648:sfmd4hb /POST/

Integration Request
Type: LAMBDA
Region: us-east-1

Method Response
HTTP Status: 200
Models: application/json => Empty

Integration Response
HTTP status pattern: -
Output passthrough: No



Amazon API Gateway > APIs > loan-prediction-endpoint (sfmd4hbbpb) > Stages > dev > / > POST

dev - POST - /

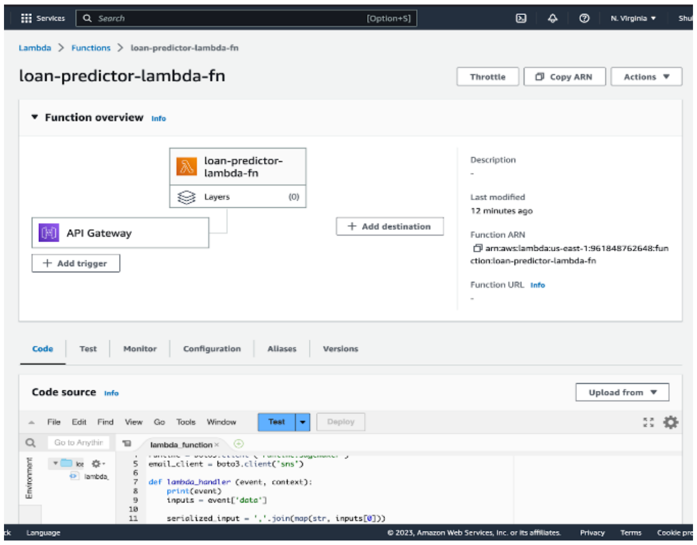
Invoke URL: https://sfmd4hbbpb.execute-api.us-east-1.amazonaws.com/dev/

Use this page to override the dev stage settings for the POST to / method.

Settings [✓ Inherit from stage] Override for this method

AWS Lambda

Using the Amazon Lambda service, we developed a lambda function that contains details about the SageMaker endpoint. The procedure in our function code that handles the events is called a lambda function handler. The handler method is executed by Lambda when the function is called. Event object and context object are the two arguments that Lambda runtime delivers to the function handler. An event is a document with data in JSON format that the lambda function can process. The event is transformed into an object by the Lambda runtime, which then provides the data to our function code. Our Lambda function receives the context object at runtime. The object has properties and methods that reveal details about the run-time environment and invocation function. Our Lambda function will serialize the input and contact the response-returning Sagemaker endpoint. Also, each time the lambda function is called, it looks for a space on a distant server where it may perform the function on a container before stopping it when the job is finished. In this way, we just have to pay for the actual running of it.



Lambda > Functions > loan-predictor-lambda-fn

loan-predictor-lambda-fn

Function overview

Description
-

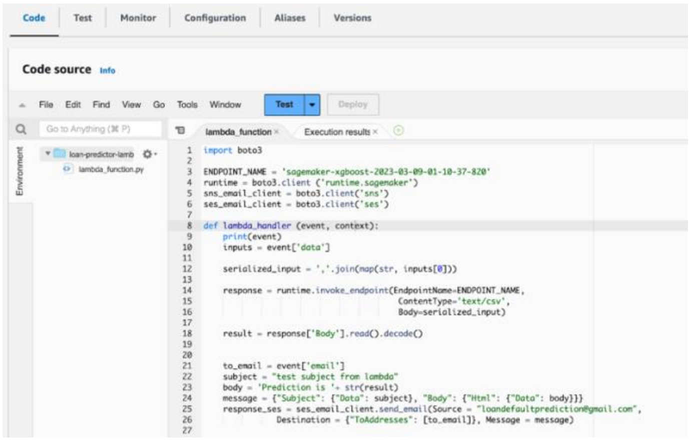
Last modified
12 minutes ago

Function ARN
arn:aws:lambda:us-east-1:961848762648:function:loan-predictor-lambda-fn

Function URL
-

Code source

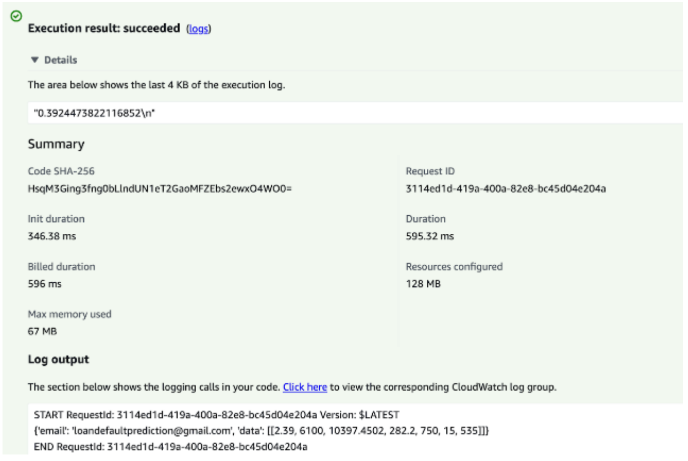
```
email_client = boto3.client('sns')

def lambda_handler (event, context):
    print(event)
    inputs = event['data']

    serialized_input = ','.join(map(str, inputs[0]))
```



```
import boto3

ENDPOINT_NAME = 'sagemaker-xgboost-2023-03-09-01-10-37-820'
runtime = boto3.client ('runtime.sagemaker')
sns_email_client = boto3.client('sns')
ses_email_client = boto3.client('ses')

def lambda_handler (event, context):
    print(event)
    inputs = event['data']

    serialized_input = ','.join(map(str, inputs[0]))

    response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
                                       ContentType='text/csv',
                                       Body=serialized_input)

    result = response['Body'].read().decode()


    to_email = event['email']
    subject = "test subject from lambda"
    body = 'Prediction is '+ str(result)
    message = {"Subject": {"Data": subject}, "Body": {"Html": {"Data": body}}}
    response_ses = ses_email_client.send_email(Source = "loandefaultprediction@gmail.com",
                   Destination = {"ToAddresses": [to_email]}, Message = message)

```

Once we obtain the prediction value, we use SNS service to send email to all the subscribers. The intended subscribers of this system will be the agent who wants to predict the loan default value and the higher level authority such as Manager who also has records on which person has requested for loan and their loan default value.

Below is the screenshot of how we tested the lambda function by invoking it through AWS UI and it returned results.



The area below shows the last 4 KB of the execution log.

```
"0.3924473822116852\n"
```

Summary

Code SHA-256
HsqM3Ging3fng0bLlndUN1eT2GaoMFZEbs2ewxO4WO0=

Request ID
3114ed1d-419a-400a-82e8-bc45d04e204a

Init duration
346.38 ms

Duration
595.32 ms

Billed duration
596 ms

Resources configured
128 MB

Max memory used
67 MB

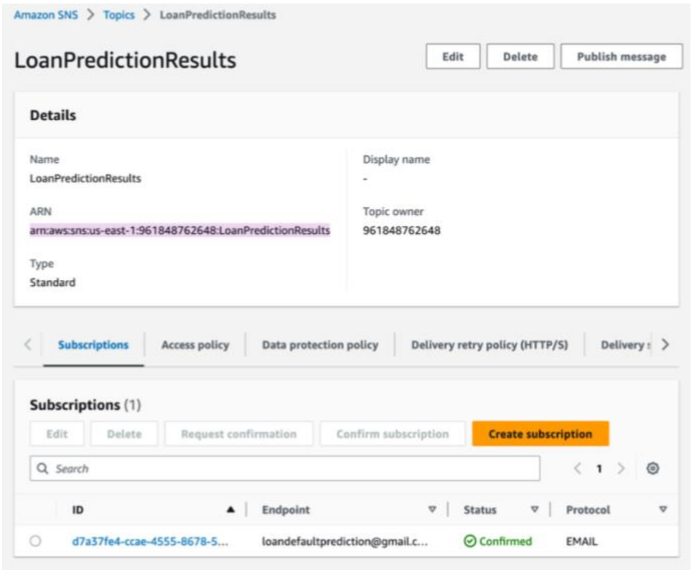
Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 3114ed1d-419a-400a-82e8-bc45d04e204a Version: $LATEST
{'email': 'loandefaultprediction@gmail.com', 'data': [[2.39, 6100, 10397.4502, 282.2, 750, 15, 535]]}
END RequestId: 3114ed1d-419a-400a-82e8-bc45d04e204a
```

AWS SNS

A completely managed messaging solution for application to application and application to person communication, Amazon Simple Notification Service. AWS Lambda function, Amazon SQS queues, and HTTPS endpoints are just a few examples of publish-subscribe systems that can fan out messages to many subscribers using SNS. We can communicate with people on a broad scale through SMS, push alerts for mobile devices, and emails thanks to the A2P capability.
To resend the email notification to the affected bank workers, we are utilizing A2P. In order for the involved bank workers to get email notifications when an application is completed, we achieved this by creating a subject in Amazon SNS and adding them as subscribers.



LoanPredictionResults

Details

Name
LoanPredictionResults

Display name
-

ARN
arn:aws:sns:us-east-1:961848762648:LoanPredictionResults

Topic owner
961848762648

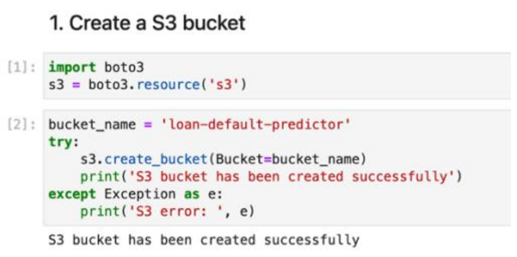
Type
Standard

Subscriptions (1)

ID	Endpoint	Status	Protocol
d7a37fe4-ccae-4555-8678-5...	loandefaultprediction@gmail.c...	Confirmed	EMAIL

Loan Default Prediction Algorithm

The CLI command below is using boto3 from AWS Software Development Kit. It allows us to create, update and delete AWS resources directly from our python notebook. Boto3 makes it easy to integrate our python application, library or script with AWS services including Amazon S3, EC2, DynamoDB and so on. The following creates a bucket named loan-default-predictor to store the train and test datasets.



1. Create a S3 bucket

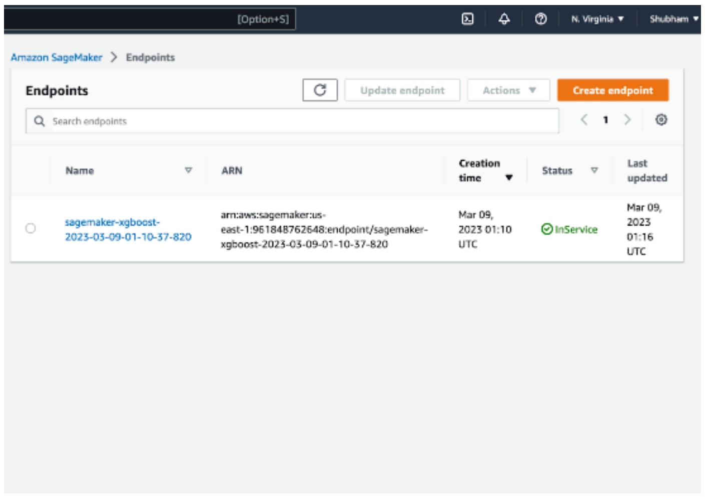
```
[1]: import boto3
     s3 = boto3.resource('s3')

[2]: bucket_name = 'loan-default-predictor'
     try:
         s3.create_bucket(Bucket=bucket_name)
         print('S3 bucket has been created successfully')
     except Exception as e:
         print('S3 error: ', e)

     S3 bucket has been created successfully
```

The next step is to preprocess and cleanse the data by inserting mean values into the blank field. We compare the columns' correlation to the desired values and eliminate any columns with poor correlation scores. We randomly divided the data into test-train datasets of 30 to 70. The S3 bucket is then filled with these datasets.
Our machine learning model's use case was the built-in XGBoost technique of Sagemaker. XGBoost implements the well-liked and successful open-source gradient boosted trees technique. Gradient boosting is a supervised learning method that combines predictions from a number of weaker, simpler models in an effort to predict a target variable with some degree of accuracy. Due to its robust handling of a variety of data types, relationships, distributions, and hyperparameters that can be fine-tuned, the XGBoost algorithm performs well in machine learning competitions.
We can automatically spot the XGBoost built-in algorithm image URI using the Sagemaker image _uris.retrieve API. After specifying the image URI, we use the XGBoost container to construct an estimator using the Sagemaker

estimator API and initiate a training job. We are setting the hyperparameters for the XGBoost algorithm by calling the set_hyperparameters method of the estimator.

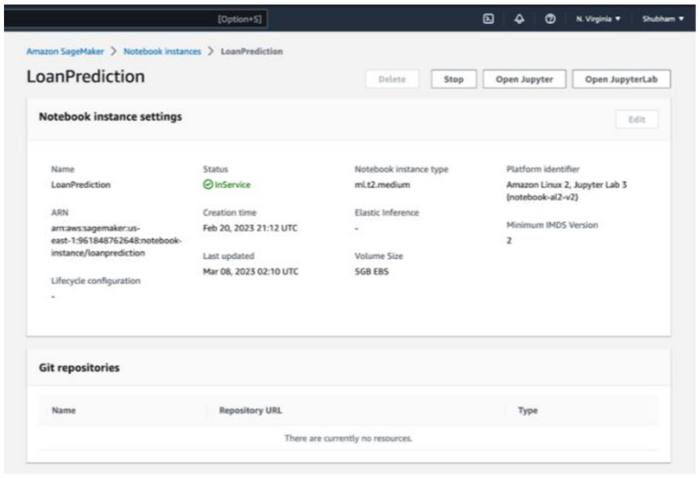
- **max_depth** is the tree's maximum depth. The model becomes more complicated and more likely to be overfitted as this value rises. Zero denotes no limit.
- **eta** is a step size shrinkage technique utilized in updates to avoid overfitting. You can get the weights of new features directly after each boosting step. The boosting procedure is made more conservative by the fact that the eta parameter reduces the feature weights.
- **gamma** is the minimum loss reduction that must be achieved in order to partition an additional leaf node in the tree. The algorithm is more cautious the larger it is.
- **min_child_weight** is the minimum amount of weight a child must have. The building process stops partitioning further if the tree partition step results in a leaf node whose instance weight sum is less than min_child_weight. This simply refers to the minimum number of instances required for each node in linear regression models. The algorithm's level of conservatism increases with its size.

To start the model training, we call the estimator's fit method with the training and test datasets that we previously uploaded to the S3 bucket. After the training job has completed, we are storing the trained XGBoost model in the S3 bucket. Next we hosted the trained model through Amazon EC2 using Amazon Sagemaker, by calling the deploy method of the xgb_model estimator. This endpoint can now be accessed using the Lambda function.



Endpoints

Name	ARN	Creation time	Status	Last updated
sagemaker-xgboost-2023-03-09-01-10-37-820	arn:aws:sagemaker:us-east-1:961848762648:endpoint/sagemaker-xgboost-2023-03-09-01-10-37-820	Mar 09, 2023 01:10 UTC	InService	Mar 09, 2023 01:16 UTC



LoanPrediction

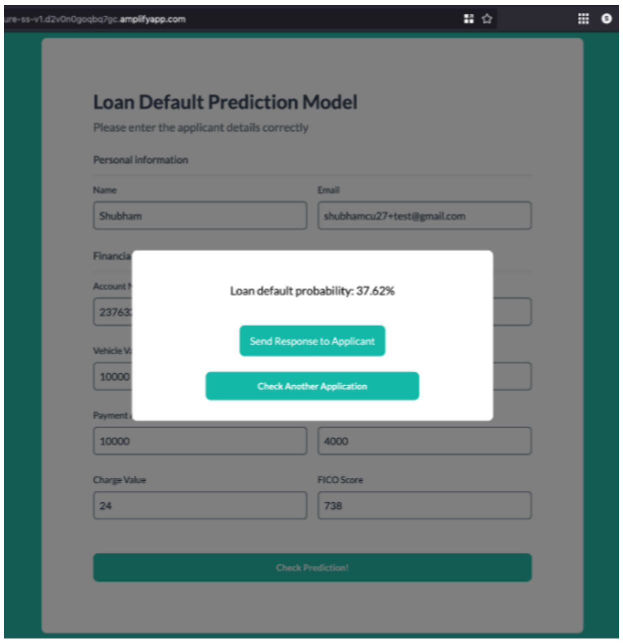
Notebook instance settings

Name: LoanPrediction
Status: InService
Notebook instance type: ml.t2.medium
Platform identifier: Amazon Linux 2, Jupyter Lab 3 (notebook-al2-v2)
ARN: arn:aws:sagemaker:us-east-1:961848762648:notebook-instance/loanprediction
Creation time: Feb 20, 2023 21:12 UTC
Elastic Inference: -
Minimum IMDS Version: 2
Last updated: Mar 08, 2023 02:10 UTC
Volume Size: 5GB EBS
Lifecycle configuration: -

Git repositories

Name	Repository URL	Type
There are currently no resources.		

Finally, the prediction and the emails to the client and higher authorities of the financial institution can be seen as below.


ure-ss-v1.d2v0n0goqbq7gc.amplifyapp.com
Send Response to Applicant
Check Another Application
Check Prediction!

Loan Default Prediction Model

Please enter the applicant details correctly

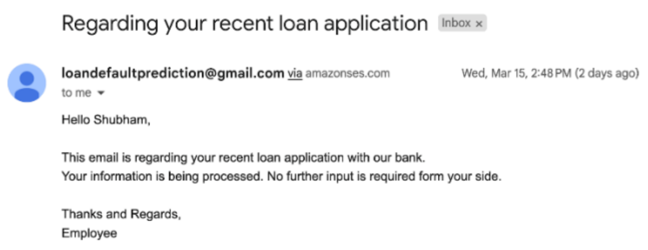
Personal information

Name: Shubham
Email: shubhamcu27+test@gmail.com

Financia

Account N: 23763
Vehicle V: 10000
Payment: 10000 | 4000
Charge Value: 24
FICO Score: 738

Loan default probability: 37.62%



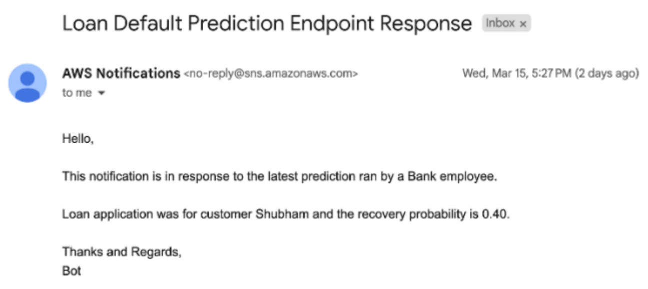
Regarding your recent loan application

loandefaultprediction@gmail.com via amazonses.com Wed, Mar 15, 2:48 PM (2 days ago)

Hello Shubham,

This email is regarding your recent loan application with our bank.
Your information is being processed. No further input is required form your side.

Thanks and Regards,
Employee



Loan Default Prediction Endpoint Response

AWS Notifications <no-reply@sns.amazonaws.com> Wed, Mar 15, 5:27 PM (2 days ago)

Hello,

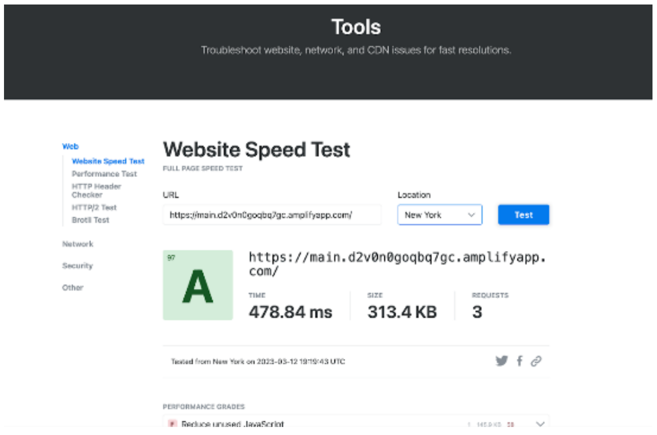
This notification is in response to the latest prediction ran by a Bank employee.

Loan application was for customer Shubham and the recovery probability is 0.40.

Thanks and Regards,
Bot

IV. OUTCOME

We tried testing the performance of our web application using an online Speed test tool, KeyCDN, which provides us with the exact time taken to load our website. As the AWS servers are physically located in N. Virginia, the load time is much better when tested from New York as compared to Frankfurt, Germany.



Website Speed Test

FULL PAGE SPEED TEST

URL: https://main.d2v0n0goqbq7gc.amplifyapp.com/

Location: New York

97 A

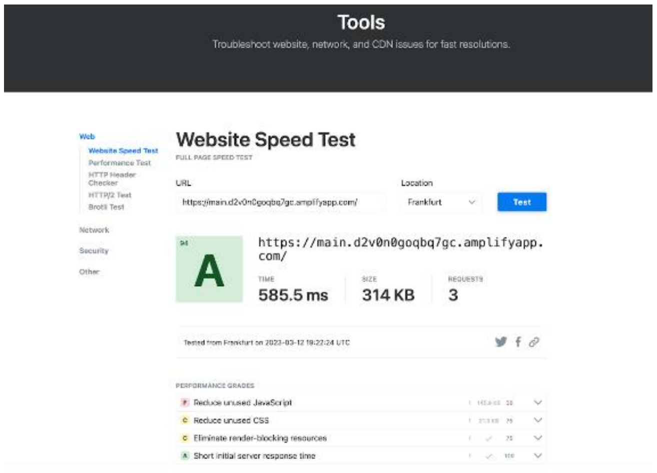
https://main.d2v0n0goqbq7gc.amplifyapp.com/

TIME	SIZE	REQUESTS
478.84 ms	313.4 KB	3

Tested from New York on 2023-03-12 19:19:43 UTC

PERFORMANCE GRADES

F Reduce unused JavaScript



Website Speed Test

FULL PAGE SPEED TEST

URL: https://main.d2v0n0goqbq7gc.amplifyapp.com/

Location: Frankfurt

94 A

https://main.d2v0n0goqbq7gc.amplifyapp.com/

TIME	SIZE	REQUESTS
585.5 ms	314 KB	3

Tested from Frankfurt on 2023-03-12 18:22:24 UTC

PERFORMANCE GRADES

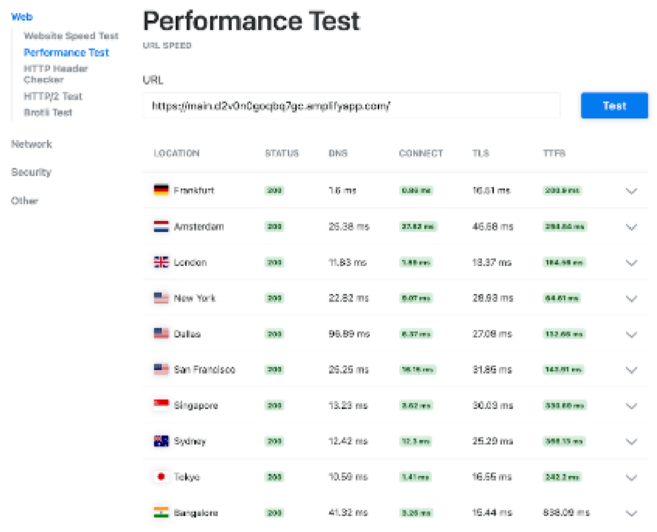
F Reduce unused JavaScript

C Reduce unused CSS

C Eliminate render-blocking resources

A Short initial server response time

We also tested the time to load the first byte (TTFB) from multiple locations across the world and got similar results. So the locations that are physically located near our AWS server, got response time of 100 ms on average. For Europe, the load times increased to 200 ms and it kept on increasing as we moved further apart. These results can also be seen in the figure below:



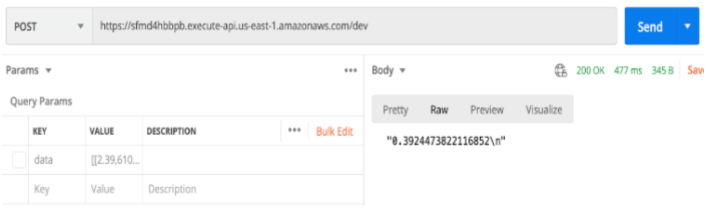
Performance Test

URL SPEED

URL: https://main.d2v0n0goqbq7gc.amplifyapp.com/

LOCATION	STATUS	DNS	CONNECT	TLS	TTFB
Frankfurt	200	1.6 ms	0.99 ms	16.61 ms	200.9 ms
Amsterdam	200	26.38 ms	27.82 ms	46.68 ms	284.84 ms
London	200	11.83 ms	1.88 ms	13.37 ms	184.98 ms
New York	200	22.82 ms	9.07 ms	28.93 ms	64.61 ms
Dallas	200	96.89 ms	6.37 ms	27.08 ms	132.68 ms
San Francisco	200	25.25 ms	16.16 ms	31.86 ms	143.91 ms
Singapore	200	13.23 ms	3.62 ms	30.03 ms	393.89 ms
Sydney	200	12.42 ms	12.3 ms	25.29 ms	398.13 ms
Tokyo	200	10.59 ms	1.41 ms	16.55 ms	242.2 ms
Bangalore	200	41.32 ms	3.26 ms	15.44 ms	838.09 ms

API Response Time Testing:
As it can be seen, initially the response time is 477 ms which is quite high because of cold start and on the subsequent requests, the response time improved to a much acceptable value of 130 ms.



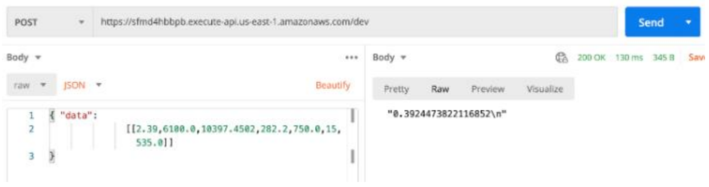
POST https://sfmd4hbbpb.execute-api.us-east-1.amazonaws.com/dev

Query Params

KEY	VALUE	DESCRIPTION
data	[[2.39,610...	
Key	Value	Description

Body 200 OK 477 ms 345 B

```
"0.3924473822116852\n"
```



POST https://sfmd4hbbpb.execute-api.us-east-1.amazonaws.com/dev

```
{ "data":
        [[2.39,6100.0,10397.4502,282.2,750.0,15,
          535.0]]
}
```

Body 200 OK 130 ms 345 B

```
"0.3924473822116852\n"
```

V. ANALYSIS AND FUTURE WORK

The program is now functional for the business case of Vehicle Loans, but it may be scaled for more use cases.
Today, we just provide the percentage of loan default value, but in the future, we will be able to inform clients of the reasons that caused a rejection of loan application.
Another possible future work could be building a dashboard that records the history of loan applications received.

VI. CONCLUSION

We used a Regression XGBoost model using AWS to determine the likelihood of a customer defaulting on a loan. The model's accuracy was around 79%, and we identified significant features that impacted the prediction, such as FICO score, Moody value, Down payment, Vehicle value, Charge off value and Financed amount. We gained proficiency in multiple AWS services, including Amplify, Cognito, Lambda, Simple Notification Service, S3, API

Gateway and Sagemaker. Our team successfully deployed instances to train our models and generate predictions.

VII. REFERENCES

1. https://www.sciencedirect.com/science/article/pii/S1877050919320277
2. https://iopscience.iop.org/article/10.1088/1757-899X/1022/1/012042/pdf
3. https://www.itm-conferences.org/articles/itmconf/pdf/2022/04/itmconf_icacc2022_03019.pdf
4. https://ijcrt.org/papers/IJCRT2106313.pdf
5. https://docs.amazonaws.cn/en_us/cognito/latest/developerguide/what-is-amazon-cognito.html
6. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
7. https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html
8. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
9. https://docs.aws.amazon.com/amplify/latest/userguide/welcome.html
10. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html