

**ITCS 6100**  
**Big Data Analytics for Competitive Advantages**  
**Group 15**  
**canvas-sample-datasets**

**Team Members**

1. Aakshi Soni (801275487)
2. Akshay Narkhede (801275760)
3. Saloni Avhad (801322583)
4. Santosh Reddy (801306775)
5. Soumik Paul (801308500)

**Dataset**

For understanding and exploration of the loan repayment status we have chosen this dataset. The dataset consists of 21 column and around 40000 rows.

S3 PATH `s3://projectbucket15/canvas-loan-dataset.csv`

**AWS Services Used**

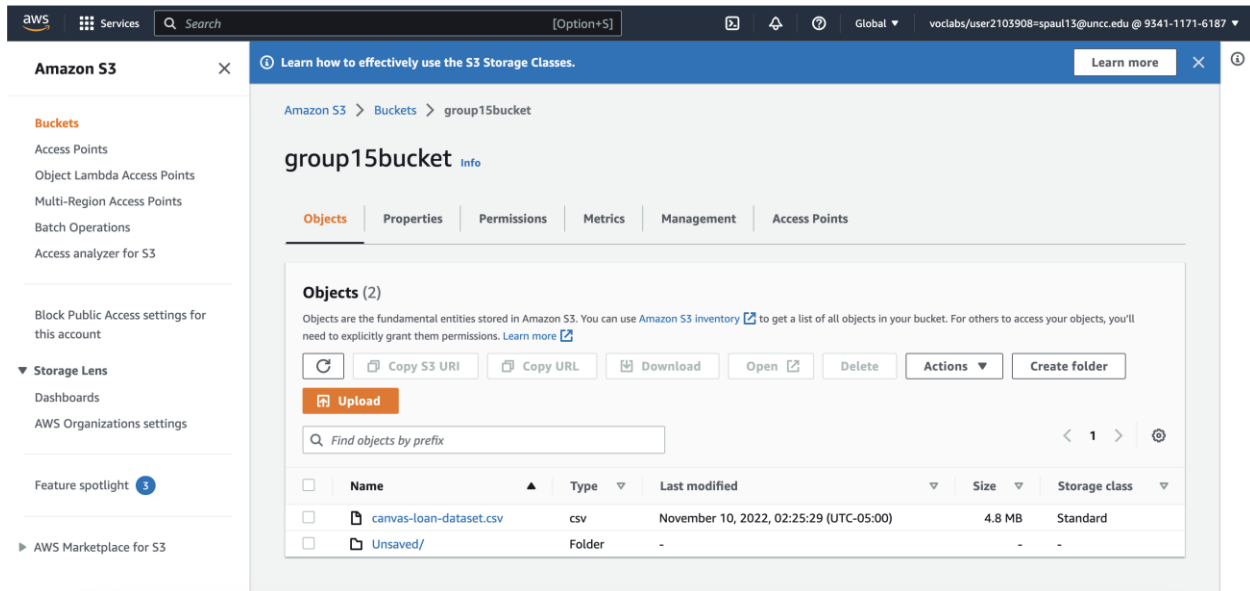
1. AWS S3
2. AWS Glue
3. AWS Athena
4. AWS QuickSight

**About Data**

Research objective is to predict, whether a customer will repay a loan or not. The dataset contains complete loan data for all loans issued from 2007-20011 including the current loan status and latest payment information. Target column for this dataset is `loan_status`. The dataset consists of ~40000 rows, 21 features columns.

## Creation of Bucket and storing the dataset into the bucket

Bucket “group15bucket” has been created and the dataset in csv format was added.



The screenshot shows the AWS S3 console interface. The left sidebar contains navigation options like Buckets, Access Points, and Storage Lens. The main content area displays the 'group15bucket' bucket details. Under the 'Objects' tab, there are two objects listed:

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	canvas-loan-dataset.csv	csv	November 10, 2022, 02:25:29 (UTC-05:00)	4.8 MB	Standard
<input type="checkbox"/>	Unsaved/	Folder	-	-	-

The top of the console shows a notification banner about S3 Storage Classes. The bottom of the image shows the AWS Glue console interface, which is partially visible and shows the 'group15bucket' bucket details in the 'Table details' tab.

**Table details**

Name	Description	Database	Classification
group15bucket	-	project15db	csv

Location	Connection	Deprecated	Last updated
s3://group15bucket/	-	-	November 10, 2022 at 07:27:33

Input format	Output format	Serde serialization lib
org.apache.hadoop.mapred.TextInputFormat	org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

# Data Preparation using AWS Glue

**Step 1:** AWS Glue is used to fetch the data from S3 to AWS Glue.

The screenshot shows the AWS Glue console interface. On the left is a navigation menu with categories like Data Catalog, Data Integration and ETL, and Triggers. The main panel is titled 'Crawlers' and includes a description of what a crawler does. Below this, there's a section for 'Crawlers (1)' with a refresh button, an 'Action' dropdown, a 'Run' button, and a 'Create crawler' button. A search bar labeled 'Filter crawlers' is present. A table lists the crawler 'group15crawler' with a 'Ready' status, a 'Succeeded' last run, and a link to 'View log'. The table also shows '1 created' table changes.

	Name	State	Schedule	Last run	Log	Table changes fr...
<input type="checkbox"/>	group15crawler	Ready		Succeeded	<a href="#">View log</a>	1 created

The screenshot shows the AWS Glue console interface for the 'Tables' section. It includes a description of tables and their use in job definitions. The 'Tables (1)' section features a refresh button, a 'Delete' button, and buttons for 'Add tables using crawler' and 'Add table'. A search bar labeled 'Filter tables' is available. A table lists the table 'group15bucket' with details: Database 'project15db', Location 's3://group15bucket/', Classification 'csv', and a 'View data' link.

	Name	Database	Location	Classification	Deprecated	View data
<input type="checkbox"/>	group15bucket	project15db	s3://group15bucket/	csv	-	<a href="#">Table data</a>

## Properties of the table

The screenshot shows the AWS Glue console interface. On the left is a navigation pane with categories like Data Catalog, Data Integration and ETL, and Legacy pages. The main area displays the 'Table properties (14)' for a selected table. At the top, a small table shows 'field.delim' with a value of '.'. Below this, a larger table lists 14 properties with their keys and values.

Key	Value
skip.header.line.count	1
sizeKey	4999412
objectCount	1
UPDATED_BY_CRAWLER	group15crawler
CrawlerSchemaSerializerVersion	1.0
recordCount	28245
averageRecordSize	177
CrawlerSchemaDeserializerVersion	1.0
compressionType	none
classification	csv
columnsOrdered	true
areColumnsQuoted	false
delimiter	,
typeOfData	file

## Table Schema

The screenshot shows the AWS Glue console interface for viewing and managing a table schema. The main area displays 'Schema (21)' with a search bar and a table listing 12 columns. The table has columns for #, Column name, Data type, Partition key, and Comment.

#	Column name	Data type	Partition key	Comment
1	id	bigint	-	-
2	loan_status	string	-	-
3	loan_amount	bigint	-	-
4	funded_amount_by_inv...	double	-	-
5	loan_term	bigint	-	-
6	interest_rate	double	-	-
7	installment	double	-	-
8	grade	string	-	-
9	sub_grade	string	-	-
10	verification_status	string	-	-
11	issued_on	string	-	-
12	purpose	string	-	-

## Step 2: Previewed the data on AWS Athena

The screenshot shows the AWS Athena console interface. On the left, the 'Data' sidebar is visible with 'Data source' set to 'AwsDataCatalog' and 'Database' set to 'project15db'. Below this, 'Tables and views' are listed, including 'group15bucket'. The main area displays the SQL query: `SELECT * FROM "project15db"."group15bucket" limit 10;`. The query status is 'Completed', with a 'Time in queue' of 185 ms, 'Run time' of 502 ms, and 'Data scanned' of 607.40 KB. The 'Results (10)' section shows a table with 7 columns: #, id, loan\_status, loan\_amount, funded\_amount\_by\_investors, loan\_term, and interest\_rate. The first two rows of data are visible.

#	id	loan_status	loan_amount	funded_amount_by_investors	loan_term	interest_rate
1	1077501	fully paid	5000	4975.0	36	10.65
2	1077430	charged off	2500	2500.0	60	15.27

This screenshot shows the same AWS Athena console interface, but with the full 'Results (10)' table displayed. The table contains 10 rows of data, showing loan details such as ID, status, amount, and interest rate.

#	id	loan_status	loan_amount	funded_amount_by_investors	loan_term	interest_rate
1	1077501	fully paid	5000	4975.0	36	10.65
2	1077430	charged off	2500	2500.0	60	15.27
3	1077175	fully paid	2400	2400.0	36	15.96
4	1076863	fully paid	10000	10000.0	36	13.49
5	1075358	current	3000	3000.0	60	12.69
6	1075269	fully paid	5000	5000.0	36	7.9
7	1069639	fully paid	7000	7000.0	60	15.96
8	1072053	fully paid	3000	3000.0	36	18.64
9	1071795	charged off	5600	5600.0	60	21.28
10	1071570	charged off	5375	5350.0	60	12.69

### Step 3: Run few SQL queries on AWS Athena for understanding the datasets.

#### 1. Avg of loan amount

The screenshot shows the Amazon Athena Query Editor interface. The 'Data' section on the left indicates the data source is 'AwsDataCatalog' and the database is 'project15db'. The 'Tables and views' section shows a table named 'group15bucket'. The SQL editor contains the following query:

```
1 SELECT AVG(Loan_amount) FROM "project15db"."group15bucket";
```

The query is labeled 'Query 11' and is in the 'Run' state. The 'Query results' section shows the query is 'Completed' with the following statistics: Time in queue: 192 ms, Run time: 836 ms, Data scanned: 4.77 MB. The 'Results (1)' section shows a single row with the value 11219.443814991062.

#	_col0
1	11219.443814991062

#### 2. Avg of interest rate

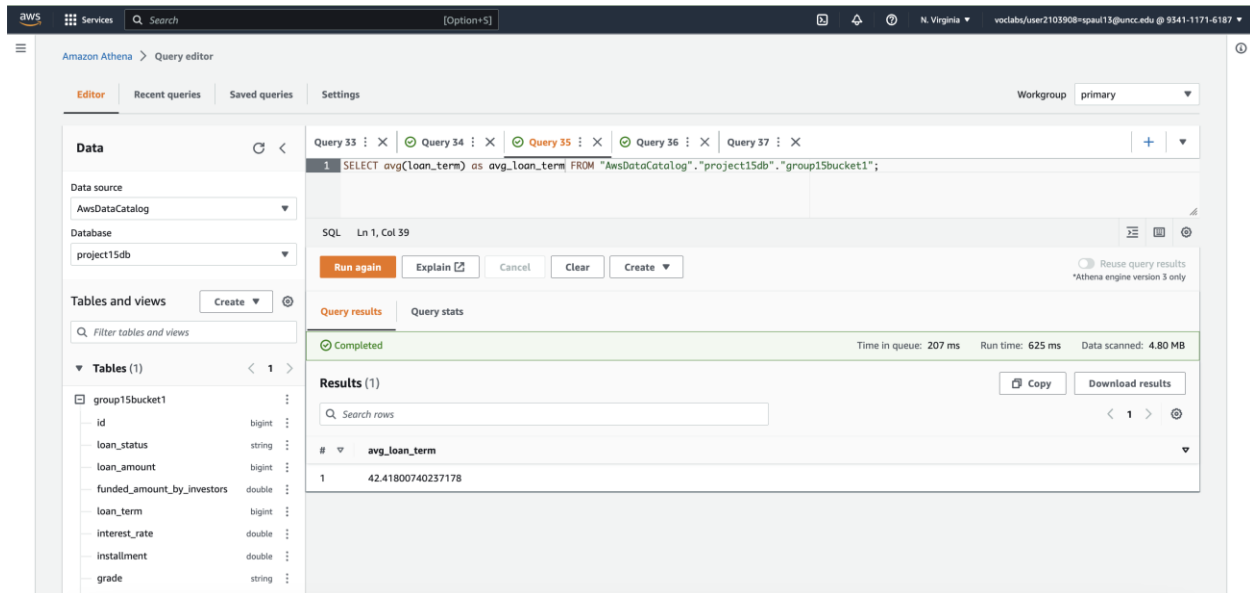
The screenshot shows the Amazon Athena Query Editor interface. The 'Data' section on the left indicates the data source is 'AwsDataCatalog' and the database is 'project15db'. The 'Tables and views' section shows a table named 'group15bucket1'. The SQL editor contains the following query:

```
1 SELECT avg(interest_rate) as average_interest_rate FROM "AwsDataCatalog"."project15db"."group15bucket1";
```

The query is labeled 'Query 36' and is in the 'Run' state. The 'Query results' section shows the query is 'Completed' with the following statistics: Time in queue: 207 ms, Run time: 612 ms, Data scanned: 4.80 MB. The 'Results (1)' section shows a single row with the value 12.02117657426169.

#	average_interest_rate
1	12.02117657426169

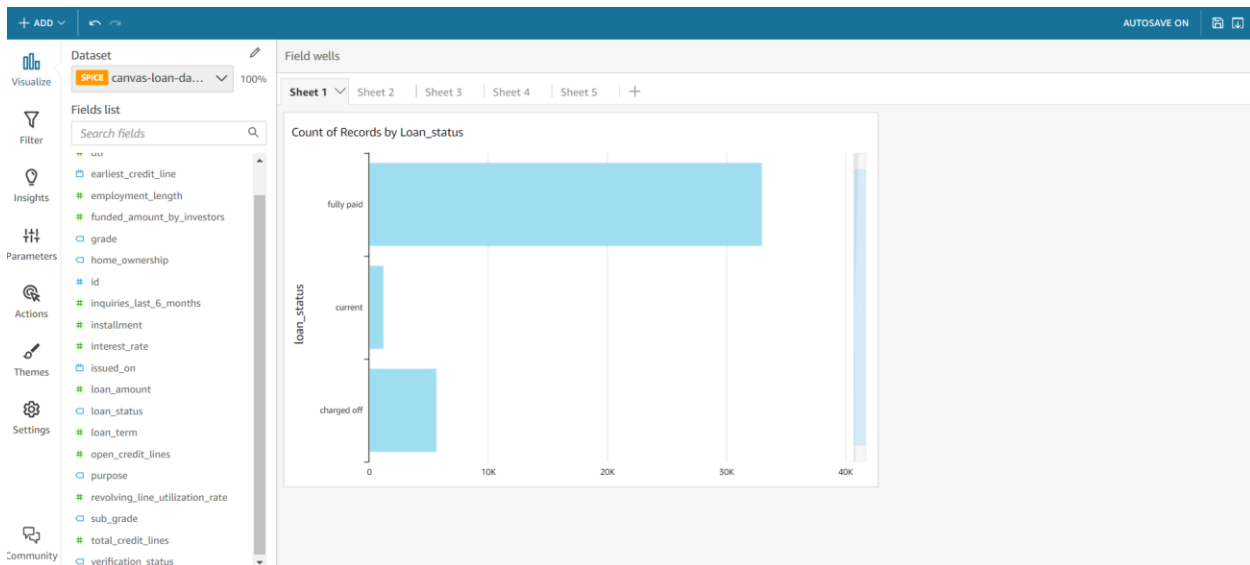
### 3. Avg of Loan Term



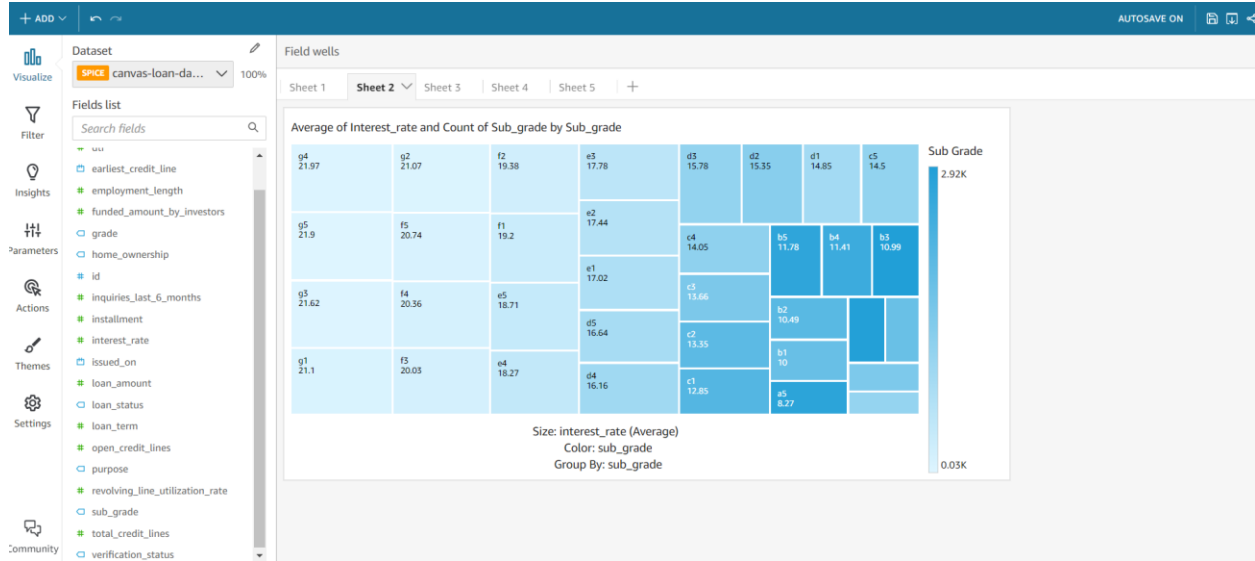
## Data Visualization on Quick Sight

The dataset has been visualize on AWS Quick Sight to get a better understanding of the data.

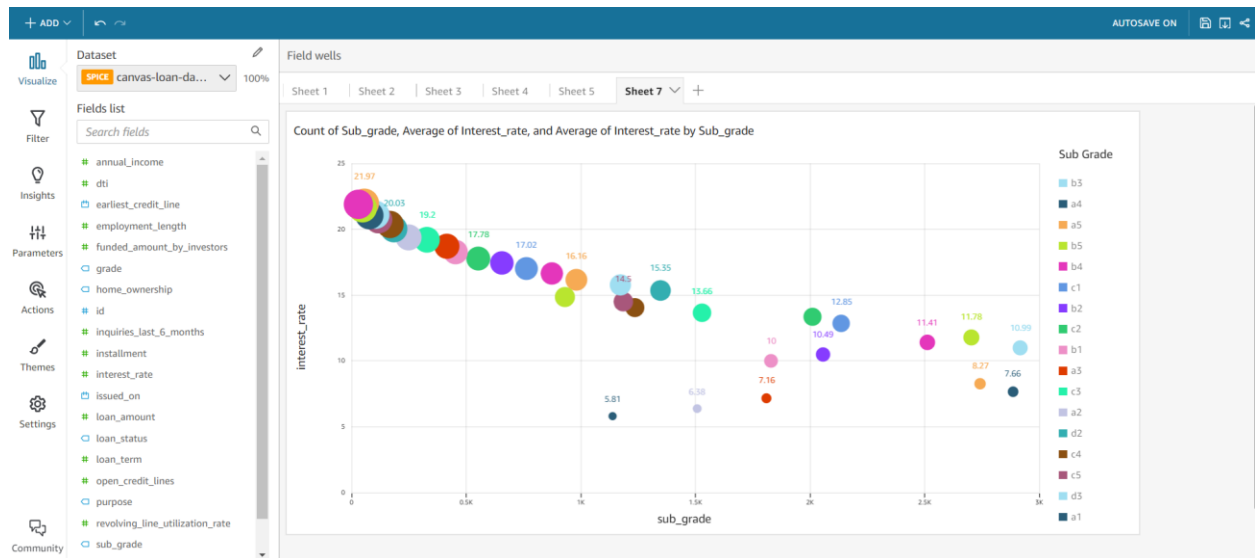
Count of output column (loan\_status)



## Average of interest rate vs count of sub grade

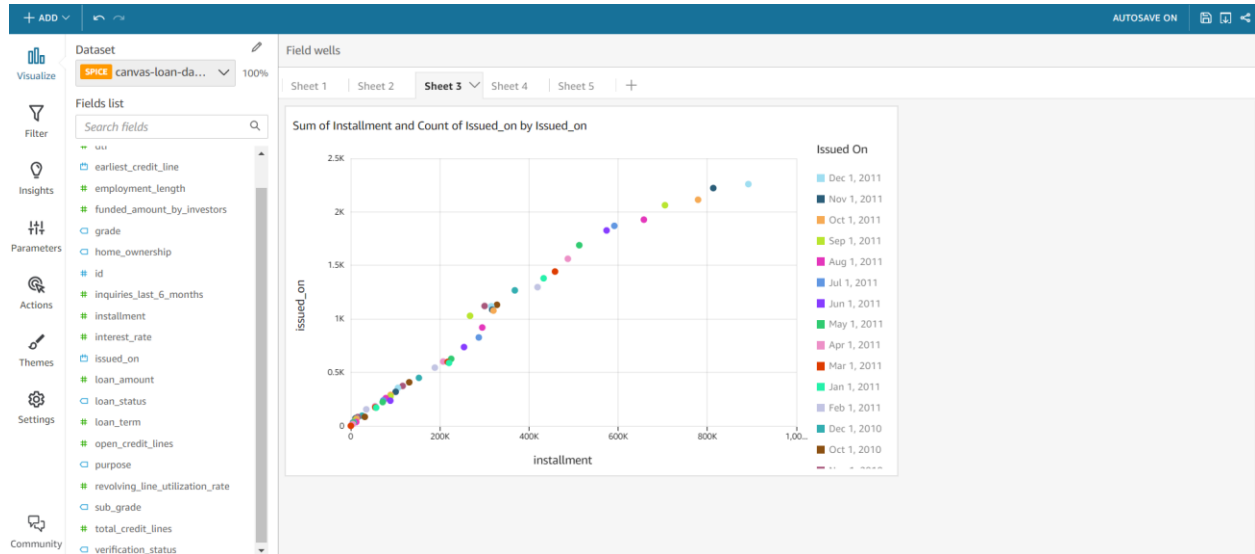


## Count of sub grade, average of interest rate and average of interest rate by sub grade.

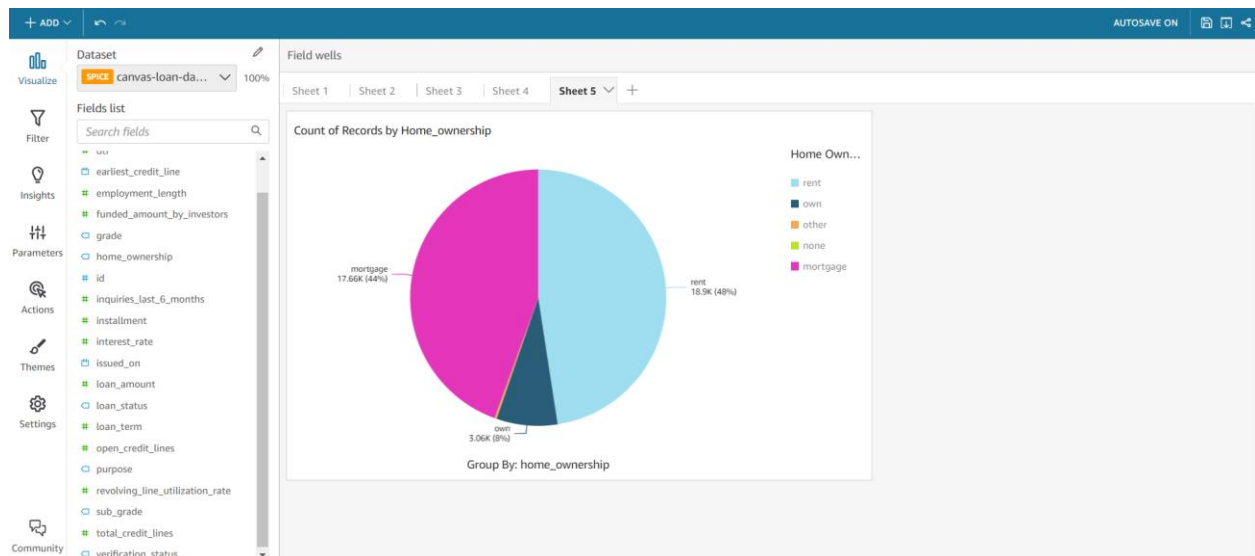




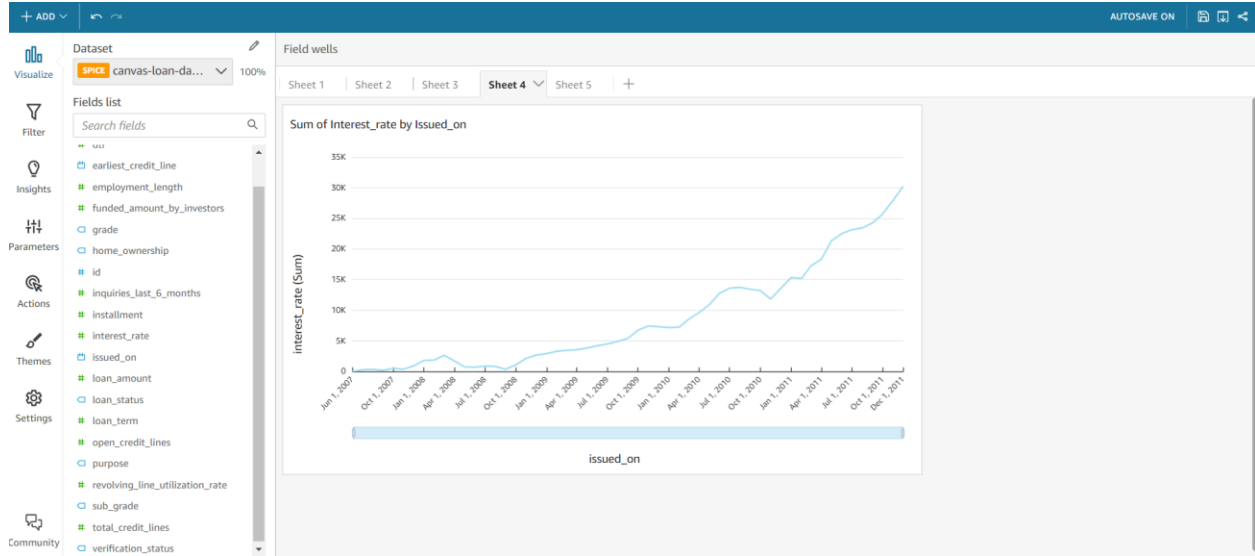
## Installment vs count of issued on by date column



## Count of records by home ownership

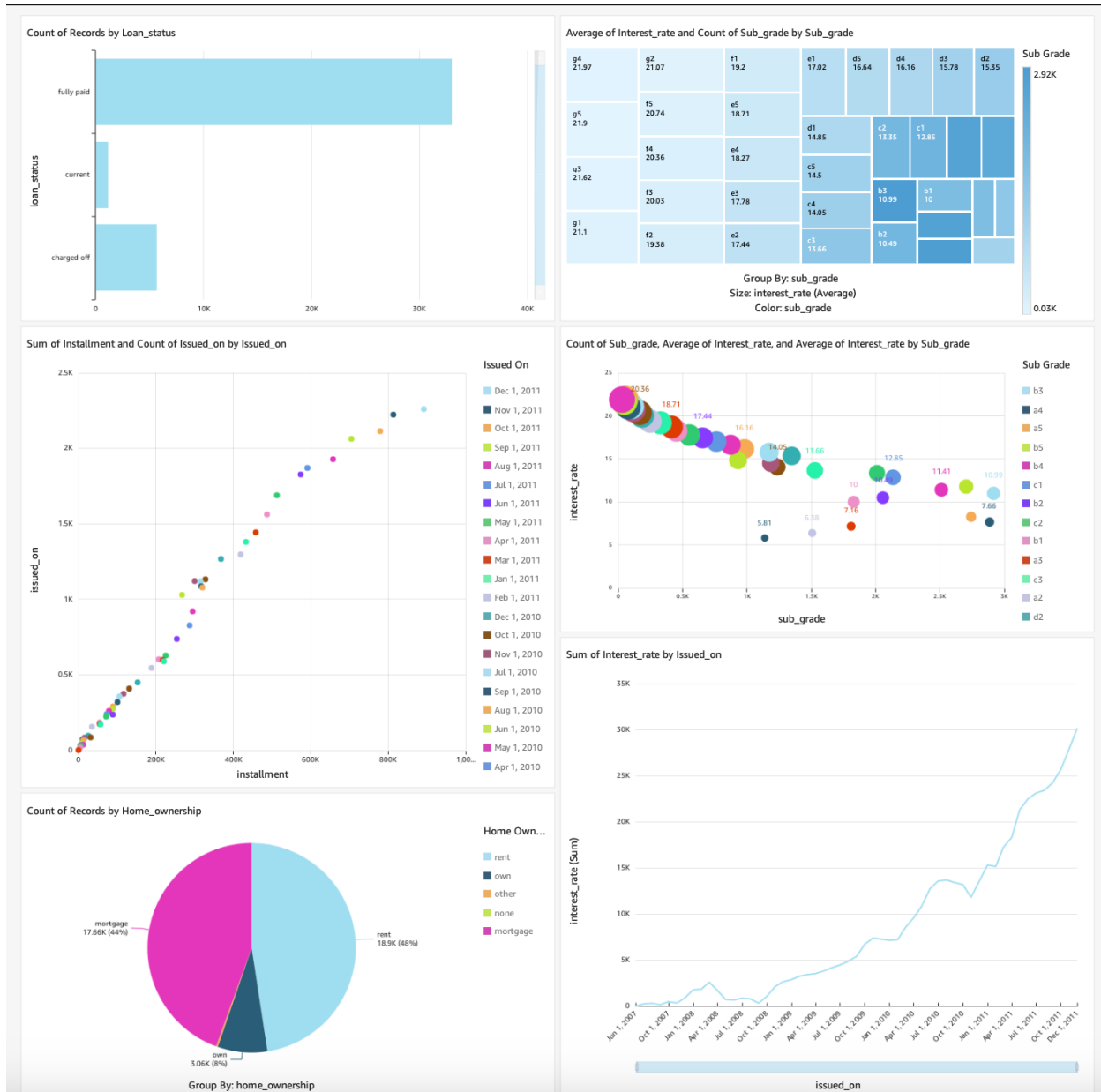


## interest rate by date



## Quick Sight Dashboard

[https://github.com/SoumikPaul108/BigData\\_Final\\_Project\\_Group15/blob/main/Sheet\\_8\\_2022-11-14T21\\_28\\_53%20\(1\).pdf](https://github.com/SoumikPaul108/BigData_Final_Project_Group15/blob/main/Sheet_8_2022-11-14T21_28_53%20(1).pdf)



# Analytics, Machine Learning

## 1. XGboost

```
In [47]: from xgboost import XGBClassifier

In [48]: # fit model to training data
model = XGBClassifier()
model.fit(X_train, y_train)

Out[48]: XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                        colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                        early_stopping_rounds=None, enable_categorical=False,
                        eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
                        grow_policy='depthwise', importance_type=None,
                        interaction_constraints='', learning_rate=0.300000012,
                        max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
                        max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
                        missing=nan, monotone_constraints=('', n_estimators=100,
                        n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0, ...)

In [49]: preds = model.predict(X_test)

In [50]: from sklearn.metrics import accuracy_score, plot_confusion_matrix, classification_report

In [51]: print(classification_report(y_test, preds))
```

## 2. Random Forest

### Random Forests

```
In [53]: from sklearn.ensemble import RandomForestClassifier

In [54]: rf = RandomForestClassifier(n_estimators=100)

In [55]: rf.fit(X_train, y_train)

Out[55]: RandomForestClassifier()

In [56]: preds = rf.predict(X_test)

In [57]: print(classification_report(y_test, preds))
```

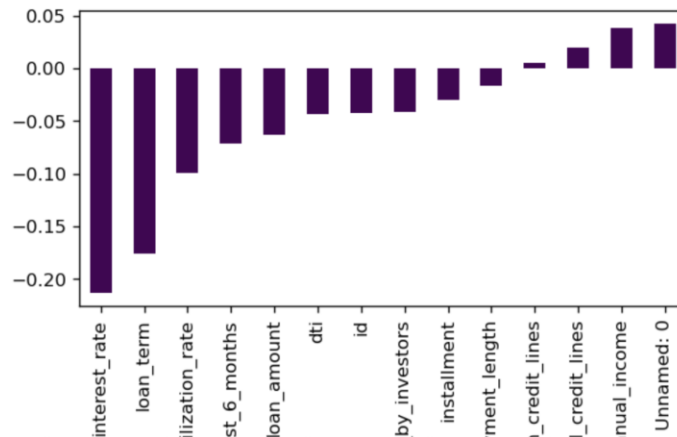
	precision	recall	f1-score	support
0	0.37	0.03	0.05	1049
1	0.86	0.99	0.92	6451
accuracy			0.86	7500
macro avg	0.62	0.51	0.49	7500
weighted avg	0.79	0.86	0.80	7500

# Evaluation and Optimization

```
In [21]: # Create dummies for loan_status so that correlation can be calculated wrt to loan_status
#for other continuous features.
```

```
In [22]: df_temp = df5.copy() # copy so that it does not affect the original data frame
df_temp['loan_status'] = pd.get_dummies(df_temp['loan_status'], drop_first=True)
```

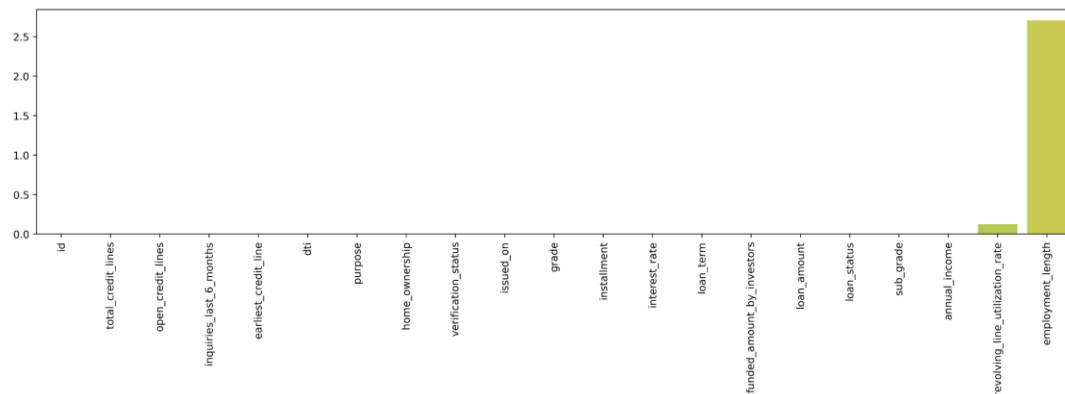
```
In [23]: plt.figure(figsize=(6,3),dpi=120)
df_temp.corr()['loan_status'].sort_values().drop('loan_status').plot(kind='bar', cmap='viridis') # correlation with loan_status
plt.xticks(rotation=90);
```



```
In [24]: # interest_rate, loan_term are highly correlated with loan_status compared to other features.
```

```
In [25]: # calculate features with missing values.
```

```
In [26]: plt.figure(figsize=(18,4),dpi=400)
sns.barplot(y=((df.isnull().sum()/len(df))*100).sort_values(), x=((df.isnull().sum()/len(df))*100).sort_values().index,
plt.xticks(rotation=90);
```



```
In [32]: df5['loan_status'] = df5['loan_status'].map({'fully paid':1,'charged off':0})
df5
```

/tmp/ipykernel\_11002/87717867.py:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
df5['loan\_status'] = df5['loan\_status'].map({'fully paid':1,'charged off':0})

```
Out[32]:
```

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loan_term	interest_rate	installment	grade	sub_grade	...	purpose
0	0	1077501	1	5000	4975.0	36	10.65	162.87	b	b2	...	credit_car
1	1	1077430	0	2500	2500.0	60	15.27	59.83	c	c4	...	ce
2	2	1077175	1	2400	2400.0	36	15.96	84.33	c	c5	...	small_busines
3	3	1076863	1	10000	10000.0	36	13.49	339.31	c	c1	...	othe
5	5	1075269	1	5000	5000.0	36	7.90	156.46	a	a4	...	weddin
...	...	...	...	...	...	...	...	...	...	...	...	...
39712	39712	92187	1	2500	1075.0	36	8.07	78.42	a	a4	...	home_improvement
39713	39713	90665	1	8500	875.0	36	10.28	275.38	c	c1	...	credit_car
39714	39714	90395	1	5000	1325.0	36	8.07	156.84	a	a4	...	debt_consolidation
39715	39715	90376	1	5000	650.0	36	7.43	155.38	a	a2	...	othe
39716	39716	87023	1	7500	800.0	36	13.75	255.43	e	e2	...	debt_consolidation

```
In [33]: df6=df5[["loan_status","loan_amount","funded_amount_by_investors","loan_term","interest_rate","installment","dti","inquiries_last_6_months","open_credit_lines","revolving_line_of_credit"]]
df6
```

```
Out[33]:
```

	loan_status	loan_amount	funded_amount_by_investors	loan_term	interest_rate	installment	dti	inquiries_last_6_months	open_credit_lines	revolving_line_of_credit
0	1	5000	4975.0	36	10.65	162.87	27.65	1	3	
1	0	2500	2500.0	60	15.27	59.83	1.00	5	3	
2	1	2400	2400.0	36	15.96	84.33	8.72	2	2	
3	1	10000	10000.0	36	13.49	339.31	20.00	1	10	
5	1	5000	5000.0	36	7.90	156.46	11.20	3	9	
...	...	...	...	...	...	...	...	...	...	
39712	1	2500	1075.0	36	8.07	78.42	11.33	0	13	
39713	1	8500	875.0	36	10.28	275.38	6.40	1	6	
39714	1	5000	1325.0	36	8.07	156.84	2.30	0	11	
39715	1	5000	650.0	36	7.43	155.38	3.72	0	17	
39716	1	7500	800.0	36	13.75	255.43	14.29	0	7	

37497 rows x 13 columns

# Results

## XGBoost

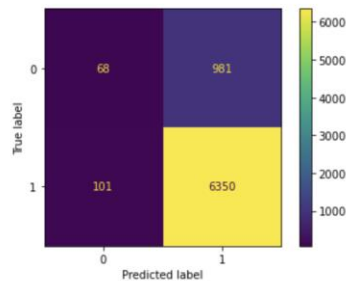
```
In [51]: print(classification_report(y_test,preds))
```

	precision	recall	f1-score	support
0	0.40	0.06	0.11	1049
1	0.87	0.98	0.92	6451
accuracy			0.86	7500
macro avg	0.63	0.52	0.52	7500
weighted avg	0.80	0.86	0.81	7500

```
In [52]: plot_confusion_matrix(model,X_test,y_test)
```

/home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot\_confusion\_matrix is deprecated; Function 'plot\_confusion\_matrix' is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from\_predictions or ConfusionMatrixDisplay.from\_estimator.  
warnings.warn(msg, category=FutureWarning)

```
Out[52]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f4f5b059280>
```



## Random Forest

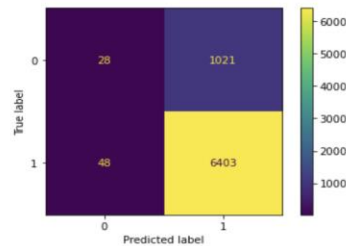
```
In [57]: print(classification_report(y_test,preds))
```

	precision	recall	f1-score	support
0	0.37	0.03	0.05	1049
1	0.86	0.99	0.92	6451
accuracy			0.86	7500
macro avg	0.62	0.51	0.49	7500
weighted avg	0.79	0.86	0.80	7500

```
In [58]: plot_confusion_matrix(rf,X_test,y_test)
```

/home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot\_confusion\_matrix is deprecated; Function 'plot\_confusion\_matrix' is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from\_predictions or ConfusionMatrixDisplay.from\_estimator.  
warnings.warn(msg, category=FutureWarning)

```
Out[58]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f4f4596ae20>
```



## **Future Work**

### **What was unique about the data? Did you have to deal with imbalance?**

The entire dataset has around 40k rows and 21 columns. The dataset was split into two halves. As a result, we had to conduct an inner join operation on the datasets in order to operate on only one.

### **What data cleaning did you do? Outlier treatment? Imputation?**

First, we determined the percentage of null values in each column, ordered in descending order of missing values; "employment length" was the sole column with 2.7% of the null value. Second, we view the null values and drop them entirely.

### **Did you create any new additional features / variables?**

We have not added any additional features. Only the features and functionality described in the instructions and AWS labs have been preserved in the project.

### **What was the process you used for evaluation? What was the best result?**

Our machine learning model was constructed using the XGboost and Random Forest algorithms. In the case of XGboost, we reached 86% accuracy, and in the case of Random Forest, we also achieved 86% accuracy. Using both techniques, we were able to obtain the same accuracy.

### **Is there Bias in your work? What were the problems you faced? How did you solve them?**

The only challenge we had was that all the text columns' data became corrupt when we uploaded our information to AWS Athena via AWS Glue, while all of the integer and double columns' data remained in pristine condition. Initial inquiries were successful; however, the dataset began to display erroneous data after a few AWS Athena queries. The dataset, however, worked flawlessly when visualized using AWS Glue.



**What future work would you like to do?**

To determine how many of the current loans were paid off, defaulted on, or even charged off, we may utilize an updated data frame that includes the numbers for the next three years (2007-2011). These new data points can then be utilized for forecasting or for developing new models.

In order to improve the model's ability to anticipate competent borrowers, we may wish to look into this matter more since the algorithm places about 36% of non-defaulters in the default class.

**Instructions for individuals that may want to use your work**

LendingClub must use caution when finding potential borrowers who meet specific requirements. Borrowers who do not own a home, for example, and apply for a small company or wedding loan may have a bad combination that leads to the borrower defaulting on a future loan.

LendingClub must be aware that low-graded loans are undoubtedly more likely to default. They must be prepared to collaborate with these borrowers to secure appropriate and timely payments. Reduced interest rates or installment payments for these consumers might be beneficial.