Matthew Rivera
CIS 4130 CMWA[28992]
Dr. Richard Holowczak

American Express - Default Prediction

Matthew Rivera (matthew.rivera3@baruchmail.cuny.edu)
GitHub URL (https://github.com/riveramatthew/Proiect-Portfolio)

Table of Contents

Default Predictions

How do banks know we'll pay back what we charge? Can we reliably predict who is likely to default? Credit default prediction is central to managing risk in a consumer lending business. Credit default prediction allows lenders to optimize lending decisions, which leads to a better customer experience and sound business economics. Current models exist to help manage risk. Would it be possible to create better models that can outperform those currently in use? So banks could address credit defaults sooner rather than later.

I like to predict the probability that a customer does not pay back their credit card balance amount in the future based on their monthly customer profile. I intend for each customer_ID, the probability of a future payment default. The dataset is 50GB from **Kaggle** and it contains aggregated profile features for each customer at each statement date. Features are anonymized and normalized, and fall into the following general categories:

- D_* = Delinquency variables
- S_* = Date variables
- P_* = Payment variables
- B_* = Balance variables
- R_* = Risk variables

I'll choose pyspark because it runs operations using multiple machine while pandas only use single machine. Pyspark can perform lazy operation so that we have no to wait every operations to be finished. I look forward to using machine learning algorithms like the pipeline.

```
pipeline = Pipeline(stages=[indexer, encoder, assembler, lr])
```

To help combine multiple algorithms into a single pipeline, in order to help run crossvalidator in retrieving the best logistical model for predictions.

```
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3 )
```

Dataset:
American Express - Default Prediction
https://www.kaggle.com/code/cdeotte/xgboost-starter-0-793/data?select=test.parquet

https://www.kaggle.com/competitions/amex-default-prediction/data

Data Acquisition

Step 1: Install the Kaggle Command Line Interface (CLI) for Python 3.

Step 2: Visit www.kaggle.com. Go to your profile and click on account.

Step 3: On the following page you will see an API section, where you will find a **"Create New API Token"** click on it, and it will download a **kaggle.json** file in which you will get your **username** and **key.**

Step 4: In your EC2 instance[i]

Step 5: Edit the kaggle_api_extended.py and make two changes (shown below) to accommodate writing the file to standard output.[ii]

Step 6: Pick a Kaggle data set you will be working with. You will need the author and data set name. Use the command: kaggle datasets list to search for a data set.

Step 7: Use the kaggle datasets download command to fetch the data set and send the file to standard output. Pipe this output to the aws s3 cp command and direct it to your S3 bucket.

```
kaggle datasets download --quiet -d totoro29/air-pollution-level -p -
$ aws s3 cp - s3://myawsbucketfolder/polution.zip
```

Step 8: Check your S3 bucket to see if the file was downloaded

```
$ aws s3 ls s3://myawsbucketfolder/
```

Step 9: Test the Boto3 module in Python[iii]

```
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
aws-emr-resources-063722171739-us-east-2
aws-logs-063722171739-us-east-2
my-data-bucket-mr
myawsbucketfolder
>>>
```

Step 10: Load up a data file from S3 into a Pandas DataFrame[iv]

```
>>> import boto3
>>> import pandas as pd
>>> df = pd.read_parquet('s3://myawsbucketfolder/train.parquet')
>>> df.head()
                                         customer_ID         S_2       P_2  D_39       B_1  ...  D_141  D_142  D_143     D_144  D_145
0  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-03-09  0.938469     0  0.008724  ...    0.0    NaN      0  0.000610      0
1  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-04-07  0.936665     0  0.004923  ...    0.0    NaN      0  0.005492      0
2  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-05-28  0.954180     3  0.021655  ...    0.0    NaN      0  0.006986      0
3  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-06-13  0.960384     0  0.013683  ...    0.0    NaN      0  0.006527      0
4  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-07-16  0.947248     0  0.015193  ...    0.0    NaN      0  0.008126      0

[5 rows x 190 columns]
>>>
```

Exploratory Data Analysis

Step 1: Get the number of records in the dataframe

```
>>> df.shape[0]
5531451
>>> df.shape[1]
190
>>>
```

Step 2: Get some statistics on each of the columns

```
>>> df.describe()
                P_2          D_39          B_1          B_2  ...         D_142         D_143         D_144         D_145
count  5.485466e+06  5.531451e+06  5.531451e+06  5.529435e+06  ...  944408.000000  5.531451e+06  5.490724e+06  5.531451e+06
mean   6.563343e-01  5.035986e+00  1.240101e-01  6.214890e-01  ...       0.390799  1.522530e-01  5.238953e-02  6.007445e-01
std    2.446494e-01  9.181833e+00  2.119869e-01  4.014876e-01  ...       0.236182  4.071715e-01  1.825135e-01  2.119894e+00
min   -4.589548e-01  0.000000e+00 -7.588799e+00  9.192280e-09  ...      -0.014539 -1.000000e+00  2.500991e-09 -1.000000e+00
25%    4.803307e-01  0.000000e+00  8.863645e-03  1.053313e-01  ...       0.199399  0.000000e+00  2.752895e-03  0.000000e+00
50%    6.942950e-01  0.000000e+00  3.132968e-02  8.143328e-01  ...       0.382136  0.000000e+00  5.508129e-03  0.000000e+00
75%    8.648159e-01  8.000000e+00  1.259019e-01  1.002403e+00  ...       0.559307  0.000000e+00  8.260448e-03  0.000000e+00
max    1.010000e+00  1.830000e+02  1.324060e+00  1.010000e+00  ...       2.229368  1.000000e+00  1.343331e+00  5.300000e+01

[8 rows x 188 columns]
>>>
```

Step 3: Check to see if some of the columns have null values

```
>>> df.isnull()
        customer_ID    S_2    P_2   D_39    B_1    B_2    R_1    S_3   D_41  ...  D_137  D_138  D_139  D_140  D_141  D_142  D_143  D_144  D_145
0             False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
1             False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
2             False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
3             False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
4             False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
...             ...    ...    ...    ...    ...    ...    ...    ...    ...  ...    ...    ...    ...    ...    ...    ...    ...    ...
5531446       False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
5531447       False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
5531448       False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
5531449       False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
5531450       False  False  False  False  False  False  False  False  False  ...  False  False  False  False  False   True  False  False  False
```

Step 4: Date variables include min and max dates

```
>>> import numpy as np
>>> df2 = df.agg(Minimum_Date=('S_2', np.min), Maximum_Date=('S_2', np.max))
>>> print(df2)
                 S_2
Minimum_Date  2017-03-01
Maximum_Date  2018-03-31
>>>
```

Conclusion

With Amazon EC2, I have gathered the summary statistic that describes features from American Express - Default Prediction dataset. Counted the number of rows and columns of the DataFrame.

```
df.shape
```

The **describe()** function returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column: count - The number of not-empty values. mean - The average (mean) value. std - The standard deviation.

```
df.describe()
```

The **isnull()** function is used to return the value given, in case the stated expression is NULL. Moreover, in case the given expression is not NULL then it returns the stated expression.

```
Null = train_df.isna().sum().div(len(train_df)).mul(100).sort_values(ascending=False)
```

The American Express - Default Prediction dataset creates trouble with machine learning algorithms might fail if the dataset contains missing values. creating biased machine learning model which will lead to incorrect results if the missing values are not handled properly. The other challenge will be the cleaning of the data with it's large size. I had help with the descriptive statistics with code borrowed from R. Holowcak found at Appendix B.

Coding and Modeling

Code for Reading of the Source Data

Step 1: Import some functions we will need later on

Step 2: Create a Spark Dataframe from the file on S3 & Check the schema

Cleaning and Normalizing the Data

Step 1: Consist of multiple groups of columns, there are id, date etc. which has to be define.

Step 2: There are some columns in this dataframe that have two or more null that has to be fix

Feature Engineering the Data

Step 1: Before using OneHotEncoding, we need to index our categoric into integer

Step 2: After Step 6, we can apply OneHotEncoder on the indexed categoric columns

Step 3: The features is fit for machine learning model few adjustment about the input

Evaluation of the models

Step 1: Creating the logistic regression model for predictions

Step 2: Running the Cross-Validator on Training and Test Data

Step 3: Testing the Best Model on the Test Data set then saving the Best Model

```
>>> test_results.show()
+--------------------+--------------------+------+-----+--------------------+--------------------+----------+
|         customer_ID|            features|target|label|       rawPrediction|         probability|prediction|
+--------------------+--------------------+------+-----+--------------------+--------------------+----------+
|00dbda372d97f2357...|(66,[0,1,4,5,7,13...|     0|    0|[0.587359770026 08...|[0.64275912530235...|       0.0|
|017c62c2d536df824...|(66,[0,1,4,5,9,13...|     0|    0|[-0.663071261290 8...|[0.34005003313781...|       1.0|
|01bc5e75de384a9bc...|(66,[0,1,4,5,9,13...|     0|    0|[0.445077594004 36...|[0.60946825245418...|       0.0|
|02d2e4eec5ba89aff...|(66,[0,1,4,5,8,13...|     0|    0|[3.045818610428 29...|[0.95460166032121...|       0.0|
|0317a4d09d430eba4...|(66,[0,1,4,5,8,13...|     0|    0|[2.578554034404 08...|[0.92946853510856...|       0.0|
|03386adba54a8f9de...|(66,[0,1,4,5,12,1...|     0|    0|[-0.053276268514 5...|[0.48668408233726...|       1.0|
|03fcc93ecd0223ce0...|(66,[0,1,4,5,11,1...|     1|    1|[0.220435335393 84...|[0.55488676005075...|       0.0|
|0407c5c405c45f8d5...|(66,[0,2,4,5,10,1...|     0|    0|[-2.409366838463 5...|[0.08246121132772...|       1.0|
|04540448ff70732fb...|(66,[0,1,4,5,8,13...|     0|    0|[3.061769937426 08...|[0.95528795701785...|       0.0|
|049e88a375fc3d30e...|(66,[0,1,4,5,6,13...|     0|    0|[3.774537112382 63...|[0.97756707389667...|       0.0|
|04b57853b44030e7d...|(66,[0,1,4,5,7,13...|     0|    0|[1.699454278600 91...|[0.84546344712483...|       0.0|
|04c67aa295d53e1b2...|(66,[0,1,4,5,8,13...|     0|    0|[3.398350370659 67...|[0.96765294051824...|       0.0|
|04e7a8463a8ba68ab...|(66,[0,1,4,5,6,13...|     0|    0|[4.081627885689 93...|[0.98340023876078...|       0.0|
|057b584e6d2cdbcc8...|(66,[0,1,4,5,6,13...|     0|    0|[2.839770477293 44...|[0.94478749059657...|       0.0|
|05aa14a4d6145b178...|(66,[0,1,4,5,6,13...|     0|    0|[2.664576250876 71...|[0.93490372638267...|       0.0|
|06319d8daa0194482...|(66,[0,1,4,5,6,13...|     0|    0|[3.037752159733 45...|[0.95425079716162...|       0.0|
|069777b9c75da0265...|(66,[0,1,4,5,6,13...|     0|    0|[1.294273974007 41...|[0.78486972504555...|       0.0|
|06cf699a0dba04828...|(66,[0,1,4,5,6,13...|     0|    0|[4.101596243749 99...|[0.98372307940043...|       0.0|
|078eaa3cde74245ec...|(66,[0,1,4,5,6,13...|     0|    0|[3.408379538390 49...|[0.96796539268897...|       0.0|
|0809f2a6bd6106996...|(66,[0,2,4,5,9,13...|     1|    1|[-0.829304720929 6...|[0.30379210334674...|       1.0|
+--------------------+--------------------+------+-----+--------------------+--------------------+----------+
only showing top 20 rows

>>>
```

Conclusion

With pyspark, I took the American Express - Default Prediction dataset to predict the probability that a customer does not pay back their credit card balance amount in the future. I form a spark Dataframe from the file on s3.

```
sdf = spark.read.parquet('s3://myawsbucketfolder/train.parquet')
```

Machine learning models can only take numeric input therefore I decided to take the categoric variables for training. I had to convert the categoric into numeric type and for this specific task, I had to use the OneHotEncoder. Before using OneHotEncoding, I had to index the categoric into integer and then use the StringIndexer for this task.

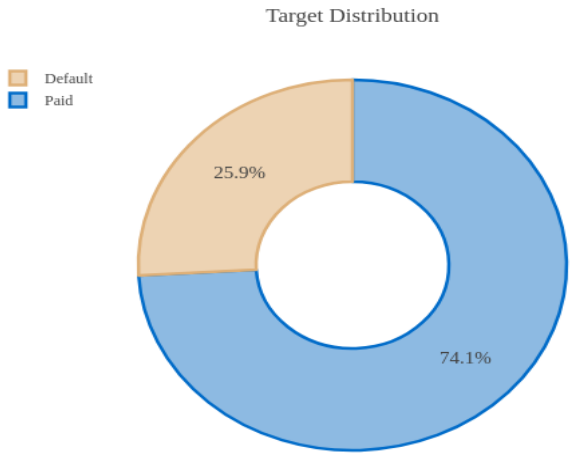
```
indexers = StringIndexer(inputCols=cat_cols, outputCols=cat_index_cols)
```

Logistic regression analysis is valuable for predicting the likelihood of an event I have chosen it to determine the probabilities whether or not the customer paid their credit card balances. The biggest challenge was with the requirements for my machine learning algorithm. The other trouble I had to face was the categoric data and trying to convert it to numeric values. I had help with these challenges using Alhazami, Rakka[1] code located at Appendix C of this document.

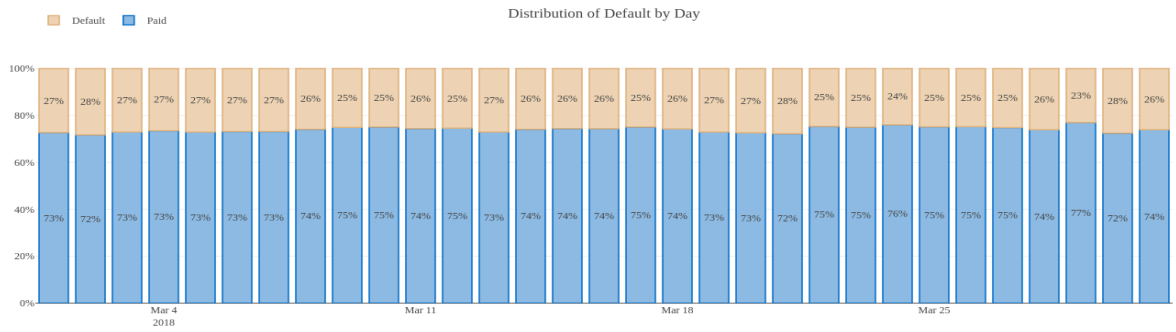
[1] https://www.kaggle.com/code/rakkaalhazimi/ml-with-pyspark/notebook

Data Visualization

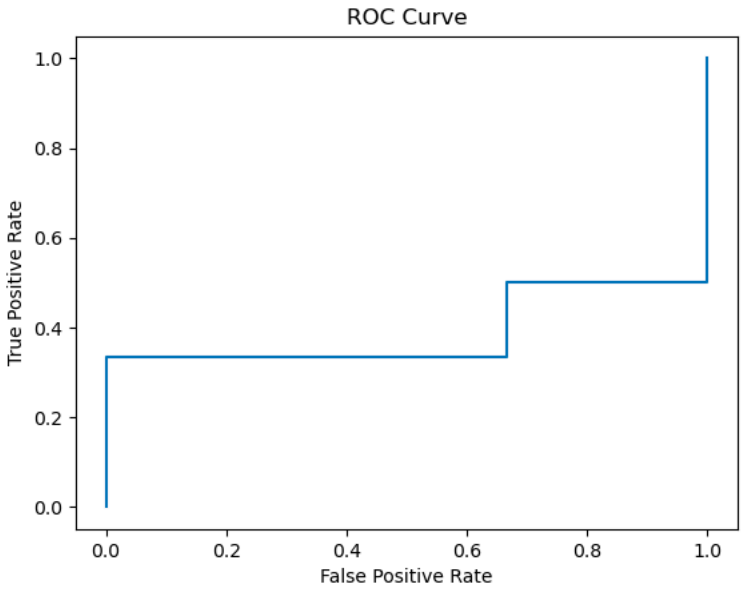
About 25% of customers in the data have defaulted shown with a pie chart.



This proportion is consistent across each day in the data set, with a weekly seasonal trend in the day of the month when customers receive their statements shown with both bar and line chart.



The **roc curve** (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds**.**



Confusion matrix a specific table layout that allows visualization of the performance.

```
+-----+-----+-----+
|label|  0.0|  1.0|
+-----+-----+-----+
|    1|15406|20209|
|    0|92520| 9606|
+-----+-----+-----+
```

Summary

With EMR I have created the means of automating the machine learning workflow by enabling data to be transformed and correlated into a model that can then be analyzed to achieve outputs. Typically when running a machine learning algorithm, it involves a sequence of tasks including pre-processing, feature extraction, model fitting, and validation stages. First I imported data and then created a label through:

```
train_ready_df= train_ready_df.withColumn("label", when(train_ready_df.target == 1, 1).otherwise(0))
```

There are two basic types of pipeline stages: Transformer and Estimator. Transformer takes a dataset as input and produces an augmented dataset as output. E.g., a tokenizer is a Transformer that transforms a dataset. An Estimator must be first fit on the input dataset to produce a model. E.g., logistic regression is an Estimator that trains on a dataset with labels and features and produces. Second through the Pipeline function I have automated the transformer process. Through ML Pipeline I alter the American Express - Default Prediction dataset to fit data into a logistic model. I have split the data into training and test sets so not to overfit the data to the model. I fitted the model to the training data and had been successful showing the test results. I then created the cross-validator to test the logistic model through:

```
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3 )
```

CrossValidator requires an Estimator, a set of Estimator ParamMaps, and an Evaluator. I have used the ParamGridBuilder to construct a grid of parameters to search over. Throught the ParamGridBuilder code for hyperparameter tuning, I help the CrossValidator select the best logistic model. I found the Area Under the Curve (AUC) of 0.87 % helping me visualize how well the machine learning model does. I finally created the visualization to have the best logistic model be seen. To show the results I had to run:

```
test_results.select('customer_ID','features', 'rawPrediction','probability','prediction','label').show()
```

Conclusion

The AUC tells that the logistic model is capable of distinguishing whether a customer defaults 87% of the time. In this project I have used the categoric data for training and the predictions, because the number of features get bloated when adding numeric features with mean, std, min and max. I had help calculating accuracy, precision, recall, and the f1_score with code borrowed from R. Holowcak found at Appendix C. I look to using machine learning algorithms in creating a model for predictions and Logistic regression analysis is valuable for prediction and I have chosen it to determine the probabilities whether or not the customer paid their credit card balances. I found that the accuracy was 0.18, precision was 0.32, recall 0.094, and the f1_score 0.15. With the AUC, I found that the logistic model had been successful for predicting whether a customer would default on a payment. The biggest challenge with the project had been the cleaning of the data and the size of the dataset. If I had to go and fix aspects of the project it'll be the exploratory data analysis (descriptive statistics) to better code the ML Pipeline.

Appendix A

i

A. Make a directory for Kaggle: $ `mkdir .kaggle`
B. Create a new kaggle.json file using nano editor: `$ nano .kaggle/kaggle.json`
C. Paste in the text from your kaggle.json file that you downloaded from Kaggle.com.
D. Save the kaggle.json file and exit nano.
E. Secure the file: `$ chmod 600 .kaggle/kaggle.json`

ii

A. Used the nano text editor to edit the `kaggle_api_extended.py` file.
B. Use the "Go To" key in Nano: `^_` to go to line 1582 in the file.
C. Change line 1582 from:

```
if not os.path.exists(outpath):
```

to

```
if not os.path.exists(outpath) and outpath != "-":
```

D. Change line 1594 From:

```
with open(outfile, 'wb') as out:
```

to

```
with open(outfile, 'wb') if outpath != "-" else
os.fdopen(sys.stdout.fileno(), 'wb', closefd=False) as out:
```

iii

```
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...  print(bucket.name)
```

iv

```
>>> import boto3
>>> import pandas as pd
>>> df = pd.read_parquet('s3://myawsbucketfolder/train.parquet',
 engine='pyarrow')
>>> df.head()
```

Appendix B

```
# Summary statistics
df.describe()
# the dimensions of the data
df.shape
# min and the max dates of the data
import numpy as np
df2 = df.agg[Minimum_Date=('S_2', np.min) Maximum_Date('S_2', np.max))
print(df2)
df['S_2'] = pd.to_datetime(df['S_2'])
print(f'Train dates range is from {df["S_2"].min()} to {df["S_2"].max()}.')
# distribution of the categorical features
categorical_features = ['B_30', 'B_38', 'D_63', 'D_64', 'D_66', 'D_68', 'D_114', 'D_
116', 'D_117', 'D_120', 'D_126']
train_df[categorical_features] = train_df[categorical_features].astype("category")
train_df[categorical_features].dtypes
# detect the missing values of the data set
df.isnull()
Null = train_df.isna().sum().div(len(train_df)).mul(100).sort_values(ascending=False)
Null
plt.style.use('Solarize_Light2')
fig, ax = plt.subplots(2,1, figsize=(25,10))
sns.barplot(x=tmp[:100].index, y=tmp[:100].values, ax=ax[0])
sns.barplot(x=tmp[100:].index, y=tmp[100:].values, ax=ax[1])
ax[0].set_ylabel("Percentage [%]"), ax[1].set_ylabel("Percentage [%]")
ax[0].tick_params(axis='x', rotation=90); ax[1].tick_params(axis='x', rotation=90)
plt.suptitle("Amount of missing data")
plt.tight_layout()
plt.show()
```

Appendix C

```
import re
import itertools
import multiprocessing

from pyspark import StorageLevel
from pyspark.sql import functions as F
from pyspark.sql import SparkSession, types
from pyspark.sql.functions import col, isnan, when, count, udf
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler

from pyspark.ml import Pipeline
from pyspark.ml.regression import GeneralizedLinearRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.classification import LogisticRegression,
LogisticRegressionModel

train_df = spark.read.parquet('s3://myawsbucketfolder/train.parquet',
header=True, inferSchema=True)
label_df = spark.read.csv('s3://myawsbucketfolder/train_labels.csv',
header=True, inferSchema=True)

def add_suffix(names, suffix):
    return [name + suffix for name in names]

# Known Columns
 info_cols = ['customer_ID', 'S_2']
 target_cols = ['target']
 cat_cols = [
... 'B_30', 'B_38',
... 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68']

# Define Numeric Columns
excluded = info_cols + cat_cols
num_cols = [col for col in train_df.columns if col not in excluded]

# Define Feature Columns
features = cat_df + num_df

train_df = (train_df.fillna(0, subset=num_df)
...                     .fillna("null", subset=cat_df))

 # Create columns aliases
catindex = add_suffix(cat_df, "_index")

 # Fit StringIndexer
indexers = StringIndexer(inputCols=cat_df, outputCols= catindex)
indexers_model = indexers.fit(train_df)

 # Transform to data
train_df_indexed = indexers_model.transform(train_df)

 # Create columns aliases
catVector = add_suffix(cat_df, "_ohe")

 # Fit OneHotEncoder
ohe = OneHotEncoder(inputCols=cat_index_cols, outputCols=cat_ohe_cols)
ohe_model = ohe.fit(train_df_indexed)

 # Transform to data
train_df_ohed = ohe_model.transform(train_df_indexed)
```

```
# Functions for each type
# each tuple consist of: (function, column's suffix)
>>> num_funcs = [
          (F.mean, "_mean"),
          (F.stddev, "_std"),
          (F.min, "_min"),
          (F.max, "_max"),
]

>>> cat_funcs = [
          (F.count, "_count"),
          (F.last, "_last"),
          (F.countDistinct, "_nunique"),
]

# Arguments for .agg method
# each arg consist of: func(colname).alias(colname + suffix)
agg_num_args = [
...     func(col).alias(col + suffix)
...     for col, (func, suffix) in itertools.product(num_cols, num_funcs)]
agg_cols_args = [
...     func(col).alias(col + suffix)
...     for col, (func, suffix) in itertools.product(cat_ohe_cols,
cat_funcs)]

# Combine numeric and categoric agg arguments
agg_args = agg_num_args + agg_cols_args
agg_args[0]

unused_cols = cat_cols + num_cols + cat_index_cols + cat_ohe_cols

# Apply the agg while also dropping unused columns
train_df_grouped = (train_df_ohed.groupBy("customer_ID")
...                                  .agg(*agg_cols_args)
...                                  .drop(*unused_cols))

train_df = train_df.join(F.broadcast(label_df), on="customer_ID")

va = VectorAssembler(
...     inputCols=train_joined_df.drop("customer_ID", "target").columns,
...     outputCol="features",
...     handleInvalid="error",
... )

train_ready_df = (va.transform(train_joined_df)
...                     .select(["customer_ID", "features", "target"])
...                     .persist(StorageLevel.DISK_ONLY))

# Create a label. =1 if default, =0 otherwise
train_ready_df= train_ready_df.withColumn("label", when(train_ready_df.target
== 1, 1).otherwise(0))

# Split the data into training and test sets
trainingData, testData = train_ready_df.randomSplit([0.7, 0.3])

# Create a LogisticRegression Estimator
lr = LogisticRegression()

# Fit the model to the training data
model = lr.fit(trainingData)

# Show model coefficients and intercept
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)
```

```
# Test the model on the testData
 test_results = model.transform(testData)
# Show the test results
 test_results.select('customer_ID','features',
'rawPrediction','probability','prediction','label').show()

# Create the pipeline
  pipeline = Pipeline(stages=[indexer, encoder, assembler, lr])

# Create a grid to hold hyperparameters
  grid = ParamGridBuilder()
  grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
  grid = grid.addGrid(lr.elasticNetParam, [0, 0.5, 1])

# Build the parameter grid
  grid = grid.build()

# How many models to be tested
  print('Number of models to be tested: ', len(grid))
  Number of models to be tested:  18

# Create a BinaryClassificationEvaluator to evaluate how well the model works
  evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

# Create the CrossValidator using the hyperparameter grid
  cv = CrossValidator(estimator=lr, estimatorParamMaps=grid,
  evaluator=evaluator)
# Train the models
  cv = cv.fit(trainingData)

# Test the predictions
  predictions = cv.transform(testData)

# Calculate AUC
  auc = evaluator.evaluate(predictions)
  print('AUC:', auc)
  AUC: 0.8762406874932296

# Look at the parameters for the best model that was evaluated from the grid
  parammap = cv.bestModel.stages[0].extractParamMap()
  for p, v in parammap.items():
  ...     print(p, v)
  cm = predictions.groupby('label').pivot('prediction').count().fillna(0).collect()
  def calculate_recall_precision(cm):
  ...     tn = cm[0][1] # True Negative
  ...     fp = cm[0][2] # False Positive
  ...     fn = cm[1][1] # False Negative
  ...     tp = cm[1][2] # True Positive
  ...     precision = tp / ( tp + fp )
  ...     recall = tp / ( tp + fn )
  ...     accuracy = ( tp + tn ) / ( tp + tn + fp + fn )
  ...     f1_score = 2 * ( ( precision * recall ) / ( precision + recall ) )
  ...     return accuracy, precision, recall, f1_score

  print( calculate_recall_precision(cm) )
```

Appendix D

```
# plot percentage of distribution of target variable
  target=train.target.value_counts(normalize=True)
  target.rename(index={1:'Default',0:'Paid'},inplace=True)
  pal, color=['#016CC9','#DEB078'], ['#8DBAE2','#EDD3B3']
  fig=go.Figure()
  fig.add_trace(go.Pie(labels=target.index, values=target*100, hole=.45,
                       showlegend=True,sort=False,
                       marker=dict(colors=color,line=dict(color=pal,width=2.5)),
                       hovertemplate = "%{label} Accounts: %{value:.2f}%<extra></extra>"))
  fig.update_layout(template=temp, title='Target Distribution',
                    legend=dict(traceorder='reversed',y=1.05,x=0),
                    uniformtext_minsize=15, uniformtext_mode='hide',width=700)
  fig.show()

# proportion of defaults across each day in the data set
  target=pd.DataFrame(data={'Default':data.groupby('S_2')['target'].mean()*100})
  target['Paid']=np.abs(data.groupby('S_2')['target'].mean()-1)*100
  rgb=['rgba'+str(matplotlib.colors.to_rgba(i,0.7)) for i in pal]
  fig=go.Figure()
  fig.add_trace(go.Bar(x=target.index, y=target.Paid, name='Paid',
                       text=target.Paid, texttemplate='%{text:.0f}%',
                       textposition='inside',insidetextanchor="middle",
                       marker=dict(color=color[0],line=dict(color=pal[0],width=1.5)),
                       hovertemplate = "<b>%{x}</b><br>Paid accounts: %{y:.2f}%"))
  fig.add_trace(go.Bar(x=target.index, y=target.Default, name='Default',
                       text=target.Default, texttemplate='%{text:.0f}%',
                       textposition='inside',insidetextanchor="middle",
                       marker=dict(color=color[1],line=dict(color=pal[1],width=1.5)),
                       hovertemplate = "<b>%{x}</b><br>Default accounts: %{y:.2f}%"))
  fig.update_layout(template=temp,title='Distribution of Default by Day',
                    barmode='relative', yaxis_ticksuffix='%', width=1400,
                    legend=dict(orientation="h", traceorder="reversed",
yanchor="bottom",y=1.1,xanchor="left", x=0))
  fig.show()

# Grab the model from Stage 0 of the pipeline
  mymodel = cv.bestModel.stages[0]
  import matplotlib.pyplot as plt
  plt.figure(figsize=(5,5))
  plt.plot(mymodel.summary.roc.select('FPR').collect(),
  ...      mymodel.summary.roc.select('TPR').collect())
  plt.xlabel('False Positive Rate')
  Text(0.5, 0, 'False Positive Rate')
  plt.ylabel('True Positive Rate')
  Text(0, 0.5, 'True Positive Rate')
  plt.title("ROC Curve")
  Text(0.5, 1.0, 'ROC Curve')
  plt.savefig("roc1.png")

# Create the confusion matrix
  predictions.groupby('label').pivot('prediction').count().fillna(0).show()
```