Project Introduction

Home Credit Default Rate

- Group 13
- Kalyani Malokar
- Krisha Mehta
- Kunal Mehra
- William Cutchin

INFO-I-526: Applications of Machine Learning

Indiana University Bloomington, Luddy School of Informatics, Computing, and Engineering

Professor Dr. James Shanahan

Date: April 25, 2023

Tierra Mallorca (https://unsplash.com/@tierramallorca?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on Unsplash (https://unsplash.com/photos/rgJ1J8SDEAY?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

Phase Leadership Plan

Phase Number	Team Member	Phase Objective Delegation
Phase 1: Project Proposal	William Cutchin (Phase Leader)	• Schedule Meetings, Organize Tasks, Lead Group Meetings • Format Project Proposal
Phase 1: Project Proposal	Krisha Mehta	• Data Description

Phase 1: Project Proposal	Group	• Machine Algorithms and Metrics
Phase 1: Project Proposal	Kalyani Malokar	• Machine Learning Pipeline (Diagram) • Gantt Chart of Tasks
Phase 1: Project Proposal	Kunal Mehra	• Machine Learning Pipeline Steps & Descriptions • Additional Algorithms (Loss Functions)
Phase 2: EDA & Basic Pipelines	Krisha Mehta (Phase Leader)	• Schedule Meetings, Organize Tasks, Lead Group Meetings • Data Retrieval
Phase 2: EDA & Basic Pipelines	Kalyani Malokar	• Feature Engineering (Round 1)
Phase 2: EDA & Basic Pipelines	Kunal Mehra	• Hyper Parameter Tuning (Round 1)
Phase 2: EDA & Basic Pipelines	William Cutchin	• Exploratory Data Analysis • Video Presentation
Phase 3: Feature Engineering & Hyperparameter Tuning	Kalyani Malokar (Phase Leader)	• Schedule Meetings, Organize Tasks, Lead Group Meetings • Feature Selection
Phase 3: Feature Engineering & Hyperparameter Tuning	Kunal Mehra	• Hyper Parameter Tuning (Round 2)
Phase 3: Feature Engineering & Hyperparameter Tuning	William Cutchin	• Feature Engineering (Round 2)
Phase 3: Feature Engineering & Hyperparameter Tuning	Krisha Mehta	• Video Presentation • Ensemble Methods
Phase 4: Final Submission	Kunal Mehra (Phase Leader)	• Neural Network Implementation • Schedule Meetings, Organize Tasks, Lead Group Meetings
Phase 4: Final Submission	William Cutchin	• Final Report • Video Presentation
Phase 4: Final Submission	Krisha Mehta	• Advanced Model Architectures
Phase 4:		

Credit Assignment

Phase 1

Task	Task Description	Assigned Member	Estimated Hours	Actual Hours	Start Date	Completion Date

Format Project Proposal	Communicate to find group members' desired tasks, write the abstract, and collect and display Team Photos.	William Cutchin	5	5.5	28/3/2023	4/4/2023
Data Description	Create table figures of data sources with descriptions	Krisha Mehta	1	1	29/3/2023	4/4/2023
Machine Algorithms and Metrics	Research and select appropriate metrics and algorithms for the datasets.	Group	1.5	2	04/03/2023	04/04/2023
Machine Learning Pipeline (Diagram)	Construct a block diagram which visualizes the suggested pipeline steps.	Kalyani Malokar	1.5	2	03/31/2023	04/03/2023
Gantt Chart of Tasks	Construct a Gantt chart which displays the waterfall of tasks and their dependencies.	Kalyani Malokar	1	1	04/04/2023	04/04/2023
Machine Learning Pipeline Steps & Descriptions	Describe and reason through the steps the pipeline will take.	Kunal Mehra	2	2	03/28/2023	03/30/2023
Additional	Select reasonable loss functions.					

Phase 2

Task	Task Description	Assigned Member	Estimated Hours	Actual Hours	Start Date	Completion Date
Data Retrieval & Preprocessing	Retrieve data from the Kaggle API and begin loading the data and pre-processing.	Krisha Mehta	3	4	04/04/2023	04/05/2023
Feature Engineering (Round 1)	Develop and deploy initial feature engineering, applying statistical techniques and log experiments.	Kalyani Malokar	6	5.5	04/05/2023	04/06/2023
Machine Pipelines & Baseline Experimentation	Test ranges of parameters for the given features, record experiment results and optimize.	Kunal Mehra	6	5.5	04/05/2023	04/07/2023
Exploratory Data Analysis & Visual Analysis	Handle missing values, perform descriptive analysis, and identify correlations.	William Cutchin	4.5	6	04/07/2023	04/11/2023
Video Presentation	Summarize the project, describe work completed, layout plans for the future, and discuss blockers.	William Cutchin	2	4	04/10/2023	04/11/2023

Phase 3

Task	Task Description	Assigned Member	Estimated Hours	Actual Hours	Start Date	Completion Date
Feature Selection	Observe and compare results from Feature engineering and decide which features are worth exploring further.	Kalyani Malokar	7	8	04/11/2023	04/12/2023
Hyper Parameter Tuning (Round 2)	Test ranges of parameters for the given features that have been selected by the feature selection step.	Kunal Mehra	6	5.5	04/13/2023	04/14/2023

Feature Engineering (Round 2)	Develop and deploy feature engineering on new selected features, log experiments and explore adding or removing features. Log these experiments.	William Cutchin	5	9	04/14/2023	04/17/2023
Ensemble Methods	Combine the multiple models or pipelines used into a single process. Log the results and compare.	Krisha Mehta	3.5	4	04/14/2023	04/18/2023
Video Presentation	Summarize the project, describe work completed, layout plans for the future, and discuss blockers	Krisha Mehta	2	3	04/14/2023	04/18/2023

Phase 4

Task	Task Description	Assigned Member	Estimated Hours	Actual Hours	Start Date	Completion Date
Neural Network Implementation	Develop and deploy an effective neural network, given the reasonings of previous ML algorithms. Test and log all experiments.	Kunal Mehra	8	12	04/18/2023	04/20/2023
Advanced Model Architectures	Combine and understand previous models to construct an effective and advanced model.	Krisha Mehta	4.5	6	04/21/2023	04/25/2023
Advanced Loss & Additional Functions	Continue to iterate and experiment with loss functions, optimizing further the model's performance.	Kalyani Malokar	4	6	04/21/2023	04/25/2023
Final Report Formatting	Accumulate all information and insight to be formatted into an attractive and logical report.	William Cutchin	8	12	04/18/2023	04/25/2023
Video Presentation	Summarize the project, describe work completed, report our successes and process, and describe how we could build on our submission.	William Cutchin	3	5	04/21/2023	04/25/2023
Final Report	Compile and discuss all progress, development, visualizations, and findings to present to peers with great efficiency.	All Members	25	30	04/24/2023	04/25/2023

Abstract

The problem that has been provided is the Home Credit Default Risk. In this problem, the machine learning team is looking to create algorithms and pipelines that will predict which individual will have successful repayment on their loan without a traditional credit score. In this phase of the project, we have previously visualized, understood, and explored these data as well as running baseline algorithms. Here we have employed feature engineering, hyperparameter tuning, and a pipleline process to try to improve our score. After our experiments, we have found that the best preforming model is the Random Forest Pipeline with our tuned hyperparameters of max_depth = 10, max_features = "sqrt", n_estimators = 100. We saw that our scores were nearly the same as our Baseline model scores albeit lesser than them. This is because we were only running our pipelines for a subset of data which consists of the top 44 highly correlated features. We have a score of 0.71322 (private) and 0.71845 (public) for our Kaggle submission this time using

Project Description

Project Description: Data - Import & Organize Data

In [ ]:

```
!pip install -q kaggle
```

In [ ]:

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

In [ ]:

```
!mkdir original_data
!mkdir original_zip
```

In [ ]:

```
import os
os.environ["KAGGLE_CONFIG_DIR"] = '/content'
```

In [ ]:

```

!kaggle competitions download -c home-credit-default-risk -p /content/original_zip
```

```
Warning: Your Kaggle API key is readable by other users on this system! To
fix this, you can run 'chmod 600 /content/kaggle.json'
Downloading home-credit-default-risk.zip to /content/original_zip
 99% 681M/688M [00:05<00:00, 85.7MB/s]
100% 688M/688M [00:05<00:00, 130MB/s]
```

In [ ]:

```
! chmod 600 /content/kaggle.json
```

In [ ]:

```
!unzip original_zip/home-credit-default-risk.zip
```

```
Archive:  original_zip/home-credit-default-risk.zip
  inflating: HomeCredit_columns_description.csv
  inflating: POS_CASH_balance.csv
  inflating: application_test.csv
  inflating: application_train.csv
  inflating: bureau.csv
  inflating: bureau_balance.csv
  inflating: credit_card_balance.csv
  inflating: installments_payments.csv
  inflating: previous_application.csv
  inflating: sample_submission.csv
```

In [ ]:

```
# Move all of the original data files from the content directory to the original dat
# This will help us seperate and organize concerns
!mv HomeCredit_columns_description.csv original_data/
!mv POS_CASH_balance.csv original_data/
!mv application_test.csv original_data/
!mv application_train.csv original_data/
!mv bureau.csv original_data/
!mv bureau_balance.csv original_data/
!mv credit_card_balance.csv original_data/
!mv installments_payments.csv original_data/
!mv previous_application.csv original_data/
!mv sample_submission.csv original_data/
```

In [ ]:

```
# Import numpy
import numpy as np
import pandas as pd

# Read each of the CSV files and sensibly name them in a pandas dataframe

df_app_train = pd.read_csv('original_data/application_train.csv')
df_app_test = pd.read_csv('original_data/application_test.csv')
df_bureau = pd.read_csv('original_data/bureau.csv')
df_bureau_bal = pd.read_csv('original_data/bureau_balance.csv')
df_pos_cash_bal = pd.read_csv('original_data/POS_CASH_balance.csv')
df_credit_card_bal = pd.read_csv('original_data/credit_card_balance.csv')
df_pre_app = pd.read_csv('original_data/previous_application.csv')
df_installments_payments = pd.read_csv('original_data/installments_payments.csv')

### Misc Data Frames
# df_sample_sub = pd.read_csv('original_data/sample_submission.csv') ## Need more ra
# df_home_credit_descr = pd.read_csv('original_data/HomeCredit_columns_description.c
```

Project Description: Data Description

Data Description: application_train.csv

DATA DESCRIPTION: application_train.csv
This data table is the primary training data for the HCDR problem. Each of the columns holds some data about the loan applicatant. For each row there is one loan application and the unique applicant is identified by the SK_ID_CURR. This table also holds the target values 0 and 1, where 1 represents that the loan was not repaid and 0 means that the loan was sucessfully repaid.

Type *Markdown* and LaTeX: α^2

In [ ]:

```
# Summary - application_train.csv
print("Number of Rows: " + str(df_app_train.shape[0]) + "\n" + "Number of Columns: "
print("Number of Missing Values: " + str(df_app_train.isna().sum().sum()))

df_app_train.head(10)
```

```
Number of Rows: 307511
Number of Columns: 122
Number of Missing Values: 9152465
```

Out[11]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLA(
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
5	100008	0	Cash loans	M	N	
6	100009	0	Cash loans	F	Y	
7	100010	0	Cash loans	M	Y	
8	100011	0	Cash loans	F	N	
9	100012	0	Revolving loans	M	N	

10 rows × 122 columns

Data Description: application_test.csv

DATA DESCRIPTION: application_test.csv
This table is the test file for our algorithms to be run on and have a predicted target score. This table does not contain all of the same data as the train set, but they have the same features and do not include the target value. This will be used later to predict over for the submission scores of the problem.

In [ ]:

```
# Summary - application_test.csv
print("Number of Rows: " + str(df_app_test.shape[0]) + "\n" + "Number of Columns: "
print("Number of Missing Values: " + str(df_app_test.isna().sum().sum()))

df_app_test.head(10)
```

```
Number of Rows: 48744
Number of Columns: 121
Number of Missing Values: 1404419
```

Out[12]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_RI
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	
5	100042	Cash loans	F	Y	
6	100057	Cash loans	M	Y	
7	100065	Cash loans	M	N	
8	100066	Cash loans	F	N	
9	100067	Cash loans	F	Y	

10 rows × 121 columns

Data Description: bureau.csv

DATA DESCRIPTION: bureau.csv

This dataset contains all of the data of the loan applicatant that has been provided from previous financial instituitons. These credits have thier own row and have the same unique identifier SK_ID_CURR and another identifier SK_ID_BUREAU. This will show all active credit, thier balances, and if they are overdue, among other information.

In [ ]:

```
# Summary - bureau.csv
print("Number of Rows: " + str(df_bureau.shape[0]) + "\n" + "Number of Columns: " +
print("Number of Missing Values: " + str(df_bureau.isna().sum().sum()))

df_bureau.head(10)
```

```
Number of Rows: 1716428
Number of Columns: 17
Number of Missing Values: 3939947
```

Out[13]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRE
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	
5	215354	5714467	Active	currency 1	-273	
6	215354	5714468	Active	currency 1	-43	
7	162297	5714469	Closed	currency 1	-1896	
8	162297	5714470	Closed	currency 1	-1146	
9	162297	5714471	Active	currency 1	-1146	

Data Description: bureau_balance.csv

DATA DESCRIPTION: bureau_balance.csv

This data is similar to the bureau table, but it gives monthly previous credits of the bureau. Each of the monthly credits is a new row in the table and shares the same credit identifier SK_ID_BUREAU. This table only shows a breif summary of the credit showing closed, open, and the monthly balance.

In [ ]:

```
# Summary - bureau_balance.csv
print("Number of Rows: " + str(df_bureau_bal.shape[0]) + "\n" + "Number of Columns:
print("Number of Missing Values: " + str(df_bureau_bal.isna().sum().sum()))

df_bureau_bal.head(10)
```

```
Number of Rows: 27299925
Number of Columns: 3
Number of Missing Values: 0
```

Out[14]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C
5	5715448	-5	C
6	5715448	-6	C
7	5715448	-7	C
8	5715448	-8	C
9	5715448	-9	0

Data Description: credit_card_balance.csv

DATA DESCRIPTION: credit_card_balance.csv

This table shows the monthly data about previously held credit cards. This data is linked to the other tables with the SK_ID_PREV and SK_ID_CURR identifiers. Amongst this data is the current balance, their credit imits, and withdraws from the account.

In [ ]:

```
# Summary - credit_card_balance.csv
print("Number of Rows: " + str(df_credit_card_bal.shape[0]) + "\n" + "Number of Colu
print("Number of Missing Values: " + str(df_credit_card_bal.isna().sum().sum()))

df_credit_card_bal.head(10)
```

```
Number of Rows: 3840312
Number of Columns: 23
Number of Missing Values: 5877356
```

Out[15]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACT
0	2562384	378907	-6	56.970	13
1	2582071	363914	-1	63975.555	4
2	1740877	371185	-7	31815.225	45
3	1389973	337855	-4	236572.110	22
4	1891521	126868	-1	453919.455	45
5	2646502	380010	-7	82903.815	27
6	1079071	171320	-6	353451.645	58
7	2095912	118650	-7	47962.125	4
8	2181852	367360	-4	291543.075	29
9	1235299	203885	-5	201261.195	22

10 rows × 23 columns

Data Description: installments_payments.csv

DATA DESCRIPTION: installment_payments.csv
This data set shows previous installment payments on loans at the Home Credit Company, which is the company being aided by this data exploration and machine learning. These data are identified by the SK_ID_PREV and SK_ID_CURR identifiers. This data shows specifically the installment payment amount, the amount paid, the version of the installment payment, and more.

In [ ]:

```
# Summary - installments_payments.csv
print("Number of Rows: " + str(df_installments_payments.shape[0]) + "\n" + "Number c
print("Number of Missing Values: " + str(df_installments_payments.isna().sum().sum()
df_installments_payments.head(10)
```

```
Number of Rows: 13605401
Number of Columns: 8
Number of Missing Values: 5810
```

Out[16]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DA
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	
5	1137312	164489	1.0	12	
6	2234264	184693	4.0	11	
7	1818599	111420	2.0	4	
8	2723183	112102	0.0	14	
9	1413990	109741	1.0	4	

Data Description: previous_application.csv

DATA DESCRIPTION: previous_application.csv

The previous_application data set shows applications that have been provided to Home Credit, the company requesting service. These data display the type of loan, the amount loaned, the payments on that loan, and more data around this topic. They are linked to the currently open loans through their SK_ID_PREV and SK_ID_CURR identifiers.

In [ ]:

```
# Summary - previous_application.csv
print("Number of Rows: " + str(df_pre_app.shape[0]) + "\n" + "Number of Columns: " +
print("Number of Missing Values: " + str(df_pre_app.isna().sum().sum()))

df_pre_app.head(10)
```

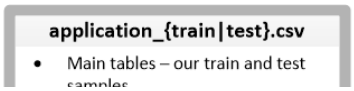
```
Number of Rows: 1670214
Number of Columns: 37
Number of Missing Values: 11109336
```

Out[17]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
0	2030495	271877	Consumer loans	1730.430	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0
5	1383531	199383	Cash loans	23703.930	315000.0
6	2315218	175704	Cash loans	NaN	0.0
7	1656711	296299	Cash loans	NaN	0.0
8	2367563	342292	Cash loans	NaN	0.0
9	2579447	334349	Cash loans	NaN	0.0

10 rows × 37 columns

Project Description: Visualization



DESCRIPTION

In the above figure we can see each of the previously described data sets and their relationship through identifier key to the application_test|train.csv data sets. This is useful in understanding how to handle these data and how they should be cleaned and preprocessed in latter experiments.

Project Description: Tasks

Here is a breif Description of the tasks at hand for this current phase

---

Project Description - Task Table: Phase 4

Task	Task Description	Assigned Member	Estimated Hours	Actual Hours	Start Date	Completion Date
Neural Network Implementation	Develop and deploy an effective neural network, given the reasonings of previous ML algorithms. Test and log all experiments.	Kunal Mehra	8	12	04/18/2023	04/20/2023
Advanced Model Architectures	Combine and understand previous models to construct an effective and advanced model.	Krisha Mehta	4.5	6	04/21/2023	04/25/2023
Advanced Loss & Additional Functions	Continue to iterate and experiment with loss functions, optimizing further the model's performance.	Kalyani Malokar	4	6	04/21/2023	04/25/2023
Final Report Formatting	Accumulate all information and insight to be formatted into an attractive and logical report.	William Cutchin	8	12	04/18/2023	04/25/2023
Video Presentation	Summarize the project, describe work completed, report our successes and process, and describe how we could build on our submission.	William Cutchin	3	5	04/21/2023	04/25/2023
Final Report	Compile and discuss all progress, development, visualizations, and findings to present to peers with great efficiency.	All Members	25	30	04/24/2023	04/25/2023

Project Description - Task breakdown: Phase 4

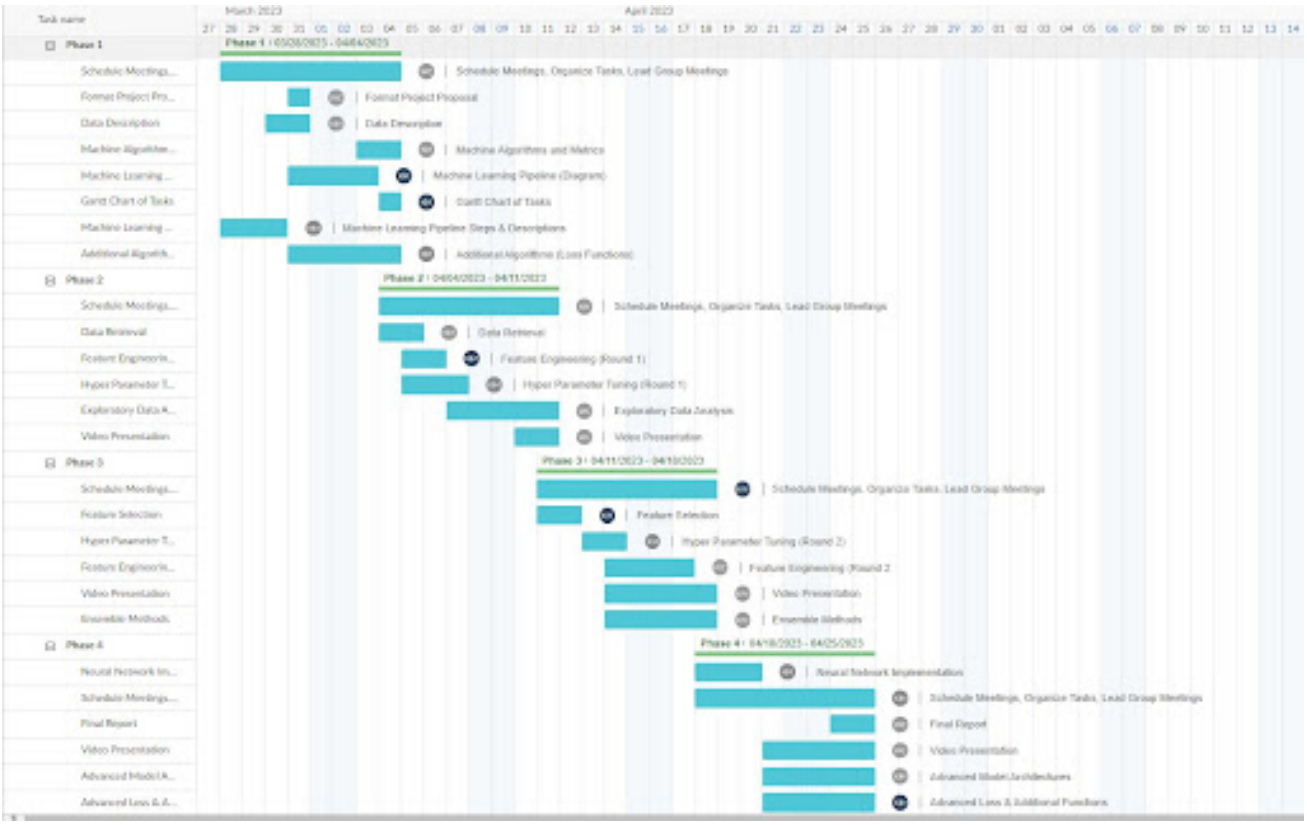
1. General Tasks
    A. Reformatting the current notebook
        a. Updating Introduction Section
        b. Renaming sections and reduce previous sections into summaries

    c. Update Credit Assignment Plan
    d. Update Visualizations
    e. Create full Breakdown
  B. Format Previous sections and reduce unused code
  C. Set the Phase 02 Baseline Experiments as labeled to those experiments
    f. Set up new experiment logs for
      i. Phase 02
      ii. Phase 03
      iii. Phase 04
  D. Redo and reformat Feature Engineering section
    g. Add more features and clean the previous data tables into more digestible datasets
  E. Record Hyperparameters
2. Neural Network/PyTorch (HCDR)
  F. Implement Neural Network (NN) model
  G. Experiment with at least 2 different Network architectures.
  H. Report neural network architecture in string form (e.g., 100 - 200 - Relu - 300 - Relu - 2 Softmax)
3. Leakage (HCDR)
  I. Go through your Pipeline and check if there is any leakage.
  J. Are you violating any cardinal sins of ML?
  K. Describe how your pipeline does not suffer from any leakage problem and does not violate any cardinal sins of ML
4. Modeling Pipelines (HCDR)
  L. -- A visualization of the modeling pipeline (s) and sub pipelines if necessary
  M. -- Families of input features and count per family
  N. -- Number of input features
  O. -- Hyperparameters and settings considered
  P. -- Loss function used (data loss and regularization parts) in latex
  Q. -- Number of experiments conducted
  R. -- Experiment table with the following details per experiment:
    h. ----- Baseline experiment
    i. ----- Any additional experiments
    j. ----- Final model tuned
    k. ----- best results (1 to three) for all experiments you conducted with the following details
    l. ---- The families of input features used
    m. ----- For train/valid/test record the following in a Pandas DataFrame:
5. Project Abstract
  S. Abstract (150 words approximately describing this phase and the previous phases of the project): that details the problem you are tackling, the main goal of the project, feature engineering, what you did (main experiments), what were your results/findings (best pipeline and the corresponding public, private scores)"
6. Results and Discussion
  T. Discussion’s aim is result interpretation, which means explain, analyze, and compare them (results from all the phases). Often, this part is the most important, simply because it lets the researcher take a step back and give a broader look at all experiments conducted. Do not discuss any outcomes not presented in the results part.
7. Conclusion
  U. Expectations here are to address the following following in your conclusion (in about 150 words) in a main section by itself:
    n. Restate your project focus and explain why it’s important. Make sure that this part of the conclusion is concise and clear.
    o. Restate your hypothesis (e.g., ML pipelines with custom features can accurately predict HCDR or Cats/Dogs)

p. Summarize main points of your project: Remind your readers your key points. (e.g, best features, best model, hyper-parameters and so on)
q. Discuss the significance of your results
r. Discuss the future of your project

Project Description: Tasks Visualization

Gantt Chart Visualization



Baseline Inital: Exploritory Data Analysis (EDA)

EDA: Data Dictionary

In [ ]:

```
#####################################
# Exploritory Data Analysis: Methods
#####################################

def EDA(eda_list):

  # Pulling information from df list
  df_name = eda_list[0]
  df = eda_list[1]

  # Header Section
  print("************************************************")
  print("                                                ")
  print("           DATAFRAME: " + df_name + "           ")
  print("                                                ")
  print("************************************************")

  print("\n")

  # Data Frame: Size & Shape
  print("================================================")
  print("Data Frame: Size, Shape & Total Missing Values")
  print("------------------------------------------------")

  print("Number of Rows: " + str(df.shape[0]))
  print("Number of Columns: " + str(df.shape[1]))
  print("Number of Total Missing Values: " + str(df.isna().sum().sum()))
  print("Data Frame Shape: " + str(df.shape))

  print("================================================")

  print("\n")

  # Data Frame: Missing Values by Feature
  print("================================================")
  print("Data Frame: Missing Values by Feature")
  print("------------------------------------------------")

  print("Number of Missing Values by Feature: " + str(df.isna().sum()))

  print("================================================")

  print("\n")

  # Data Frame: Data Types
  print("================================================")
  print("Data Frame: Data Types")
  print("------------------------------------------------")

  print(df.dtypes)

  print("================================================")

  print("\n")

  # Data Frame: Data Type Count
  print("================================================")
  print("Data Frame: Data Types")
  print("------------------------------------------------")
```

```
60
61  print(df.dtypes.value_counts())
62
63  print("================================================")
64
65  print("\n")
66
67  # Data Frame: Summary Statistics
68  print("================================================")
69  print("Data Frame: Summary Statistics")
70  print("------------------------------------------------")
71
72  print(df.describe())
73
74  print("================================================")
75
76  print("\n")
77
78  # Data Frame: Correlation Statistics
79  print("================================================")
80  print("Data Frame: Correlation Statistics")
81  print("------------------------------------------------")
82
83  print(df.corr())
84
85  print("================================================")
86
87  print("\n")
88
89  # Data Frame: Additional Text Based Analysis
90  print("================================================")
91  print("Data Frame: Additional Information")
92  print("------------------------------------------------")
93
94  print(df.info())
95
96  print("================================================")
```

EDA Data Dictionary: application_train.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_app_train = ['Application Train', df_app_train]

# Calling EDA Method
EDA(eda_info_app_train)
```

```
**********************************************

           DATAFRAME: Application Train

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 307511
Number of Columns: 122
Number of Total Missing Values: 9152465
Data Frame Shape: (307511, 122)
================================================


================================================
Data Frame: Missing Values by Feature
```

EDA Data Dictionary: application_test.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_app_test = ['Application Test', df_app_test]

# Calling EDA Method
EDA(eda_info_app_test)
```

```
**********************************************

           DATAFRAME: Application Test

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 48744
Number of Columns: 121
Number of Total Missing Values: 1404419
Data Frame Shape: (48744, 121)
================================================


================================================
Data Frame: Missing Values by Feature
```

EDA Data Dictionary: bureau.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_bureau = ['Bureau', df_bureau]

# Calling EDA Method
EDA(eda_info_bureau)
```

```
**********************************************

           DATAFRAME: Bureau

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 1716428
Number of Columns: 17
Number of Total Missing Values: 3939947
Data Frame Shape: (1716428, 17)
================================================


================================================
Data Frame: Missing Values by Feature
```

EDA Data Dictionary: bureau_balance.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_bureau_bal = ['Bureau Balance', df_bureau_bal]

# Calling EDA Method
EDA(eda_info_bureau_bal)
```

```
**********************************************

           DATAFRAME: Bureau Balance

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 27299925
Number of Columns: 3
Number of Total Missing Values: 0
Data Frame Shape: (27299925, 3)
================================================


================================================
Data Frame: Missing Values by Feature
------------------------------------------------
Number of Missing Values by Feature: SK_ID_BUREAU      0
MONTHS_BALANCE    0
STATUS            0
dtype: int64
================================================


================================================
Data Frame: Data Types
------------------------------------------------
SK_ID_BUREAU       int64
MONTHS_BALANCE     int64
STATUS            object
dtype: object
================================================


================================================
Data Frame: Data Types
------------------------------------------------
int64     2
object    1
dtype: int64
================================================


================================================
Data Frame: Summary Statistics
------------------------------------------------
       SK_ID_BUREAU  MONTHS_BALANCE
count  2.729992e+07    2.729992e+07
mean   6.036297e+06   -3.074169e+01
std    4.923489e+05    2.386451e+01
min    5.001709e+06   -9.600000e+01
25%    5.730933e+06   -4.600000e+01
50%    6.070821e+06   -2.500000e+01
75%    6.431951e+06   -1.100000e+01
max    6.842888e+06    0.000000e+00
================================================
```

```
================================================
Data Frame: Correlation Statistics
------------------------------------------------

<ipython-input-18-21e78107dac0>:83: FutureWarning: The default value of nu
meric_only in DataFrame.corr is deprecated. In a future version, it will d
efault to False. Select only valid columns or specify the value of numeric
_only to silence this warning.
  print(df.corr())

                SK_ID_BUREAU  MONTHS_BALANCE
SK_ID_BUREAU        1.000000        0.011873
MONTHS_BALANCE      0.011873        1.000000
================================================


================================================
Data Frame: Additional Information
------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None
================================================
```

EDA Data Dictionary: POS_CASH_balance.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_pos_cash_bal = ['POS_CASH Balance', df_pos_cash_bal]

# Calling EDA Method
EDA(eda_info_pos_cash_bal)
```

```
**********************************************

           DATAFRAME: POS_CASH Balance

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 10001358
Number of Columns: 8
Number of Total Missing Values: 52158
Data Frame Shape: (10001358, 8)
================================================


================================================
Data Frame: Missing Values by Feature
```

EDA Data Dictionary: credit_card_balance.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_credit_card_bal = ['Credit Card Balance', df_credit_card_bal]

# Calling EDA Method
EDA(eda_info_credit_card_bal)
```

```
**********************************************

          DATAFRAME: Credit Card Balance

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 3840312
Number of Columns: 23
Number of Total Missing Values: 5877356
Data Frame Shape: (3840312, 23)
================================================


================================================
Data Frame: Missing Values by Feature
```

Data Dictionary: previous_application.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_pre_app = ['Previous Application', df_pre_app]

# Calling EDA Method
EDA(eda_info_pre_app)
```

```
**********************************************

           DATAFRAME: Previous Application

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 1670214
Number of Columns: 37
Number of Total Missing Values: 11109336
Data Frame Shape: (1670214, 37)
================================================


================================================
Data Frame: Missing Values by Feature
```

Data Dictionary: installment_payments.csv

In [ ]:

```
# Entering information to call the EDA Method
eda_info_installments_payments = ['Installment Payments', df_installments_payments]

# Calling EDA Method
EDA(eda_info_installments_payments)
```

```
**********************************************

           DATAFRAME: Installment Payments

**********************************************


================================================
Data Frame: Size, Shape & Total Missing Values
------------------------------------------------
Number of Rows: 13605401
Number of Columns: 8
Number of Total Missing Values: 5810
Data Frame Shape: (13605401, 8)
================================================


================================================
Data Frame: Missing Values by Feature
```

Baseline Inital: Visual Exploritory Data Analysis (VEDA)

VEDA: Input & Target Features

In [ ]:

```
# Import Libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

VEDA: Target Feature Visualiaztion

In [ ]:

```
# First lets see numerically the distribution of targets
df_app_train["TARGET"].value_counts()
```

Out[28]:

```
0    282686
1     24825
Name: TARGET, dtype: int64
```

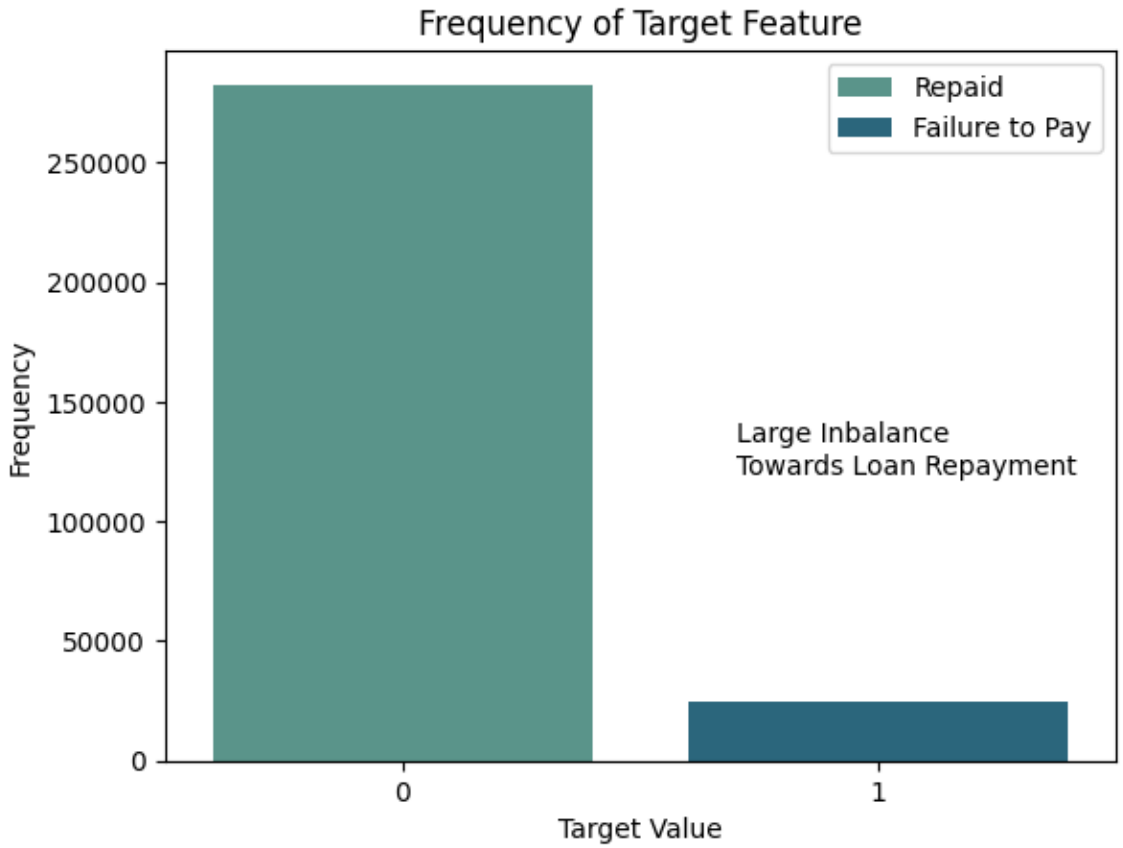
We can see that there is a large imbalance between the targets, with most customers lent to repaying the loan.

In [ ]:

```
# Lets visualize this

# Bar Plot
g = sns.countplot(data = df_app_train, x = "TARGET", palette="crest", hue="TARGET",
g.legend(loc="upper right", labels=["Repaid", "Failure to Pay"])
g.set_title("Frequency of Target Feature")
g.set_ylabel("Frequency")
g.set_xlabel("Target Value")
g.annotate("Large Inbalance \nTowards Loan Repayment", xy = (0.7, 120000))

```

Out[29]:

```
Text(0.7, 120000, 'Large Inbalance \nTowards Loan Repayment')
```



VEDA: Input Feature Visualiaztion (application_train.csv)

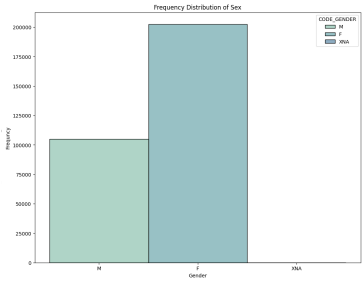
VEDA: Input Feature Visualiaztion: Demographics

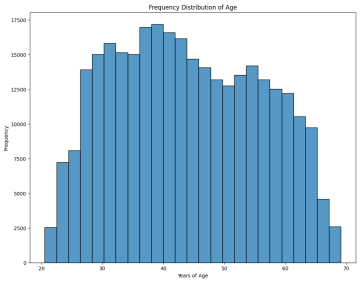
In [ ]:

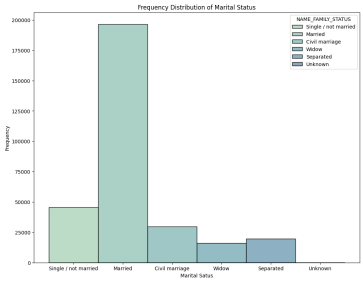
```
#pre-vis processing
df_app_train_age = df_app_train['DAYS_BIRTH'] / 365 * -1

# set up fig
fig, ax = plt.subplots(2,3, sharex=False, figsize=(40,20))

# Set Figure Labels
ax[0,0].set_title('Frequency Distribution of Sex')
ax[0,1].set_title('Frequency Distribution of Age')
ax[0,2].set_title('Frequency Distribution of Marital Status')
ax[1,0].set_title('Frequency Distribution of Child Count')
ax[1,1].set_title('Frequency Distribution of Family Member Count')
ax[1,2].set_title('Frequency Distribution of Client Education')

# Set Lables
ax[0,0].set_ylabel('Frequncy')
ax[0,1].set_ylabel('Frequency')
ax[0,2].set_ylabel('Frequency')
ax[1,0].set_ylabel('Frequency')
ax[1,1].set_ylabel('Frequency')
ax[1,2].set_ylabel('Frequency')

# Set Lables
ax[0,0].set_xlabel('Gender')
ax[0,1].set_xlabel('Years of Age')
ax[0,2].set_xlabel('Marital Satus')
ax[1,0].set_xlabel('Number of Children')
ax[1,1].set_xlabel('Number of Family Members')
ax[1,2].set_xlabel('Level of Education')

# Set histogram
sns.histplot(ax = ax[0,0], data = df_app_train, palette="crest", x = "CODE_GENDER",
sns.histplot(ax = ax[0,1], data = df_app_train_age, bins=25)
sns.histplot(ax = ax[0,2], data = df_app_train, palette="crest", x = "NAME_FAMILY_ST
sns.countplot(ax = ax[1,0], data = df_app_train, palette="crest", x = "CNT_CHILDREN"
sns.countplot(ax = ax[1,1], data = df_app_train, palette="crest", x = "CNT_FAM_MEMBE
sns.histplot(ax = ax[1,2], data = df_app_train, palette="crest", x = "NAME_EDUCATION


```

Out[30]:

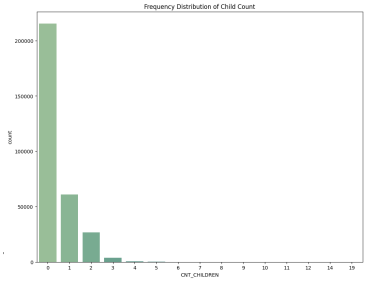
```
<Axes: title={'center': 'Frequency Distribution of Client Education'}, xla
bel='Level of Education', ylabel='Frequency'>
```

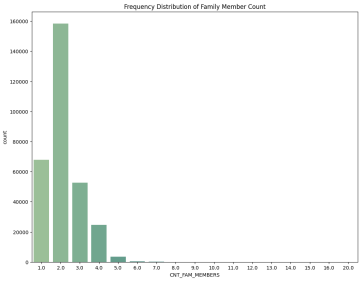


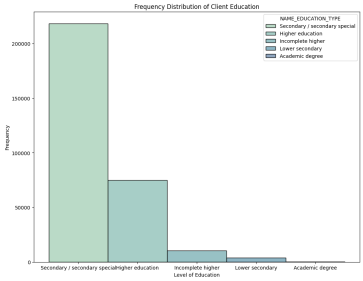




y are most







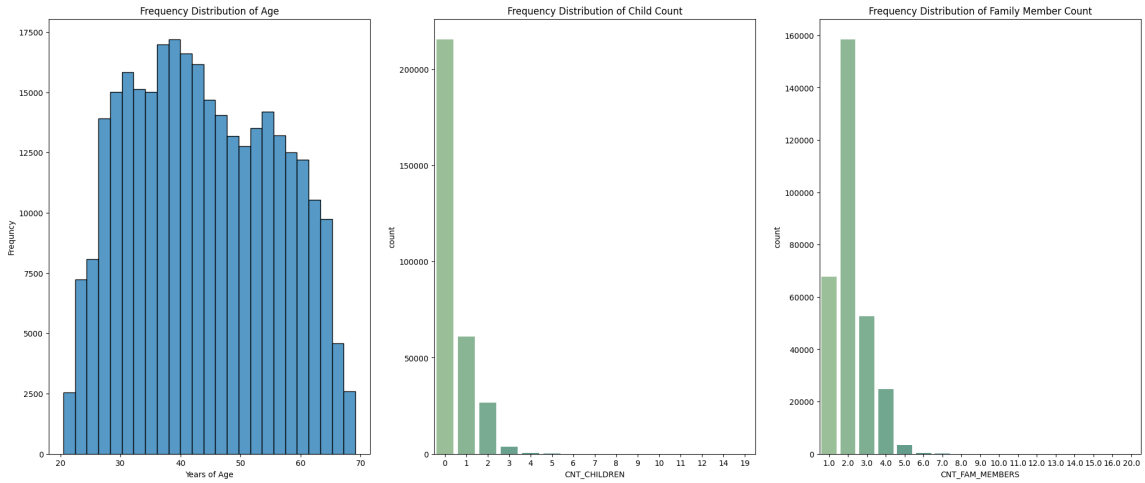
e can look

In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,3, sharex=False, figsize=(25,10))

# Set Figure Labels
ax[0].set_title('Frequency Distribution of Age')
ax[1].set_title('Frequency Distribution of Child Count')
ax[2].set_title('Frequency Distribution of Family Member Count')

# Set Lables
ax[0].set_ylabel('Frequncy')
ax[1].set_ylabel('Frequency')
ax[2].set_ylabel('Frequency')

# Set Lables
ax[0].set_xlabel('Years of Age')
ax[1].set_xlabel('Number of Children')
ax[2].set_xlabel('Number of Family Members')

# Set histogram
sns.histplot(ax = ax[0], data = df_app_train_age, bins=25)
sns.countplot(ax = ax[1], data = df_app_train, palette="crest", x = "CNT_CHILDREN")
sns.countplot(ax = ax[2], data = df_app_train, palette="crest", x = "CNT_FAM_MEMBERS
```

Out[31]:

```
<Axes: title={'center': 'Frequency Distribution of Family Member Count'},
xlabel='CNT_FAM_MEMBERS', ylabel='count'>
```

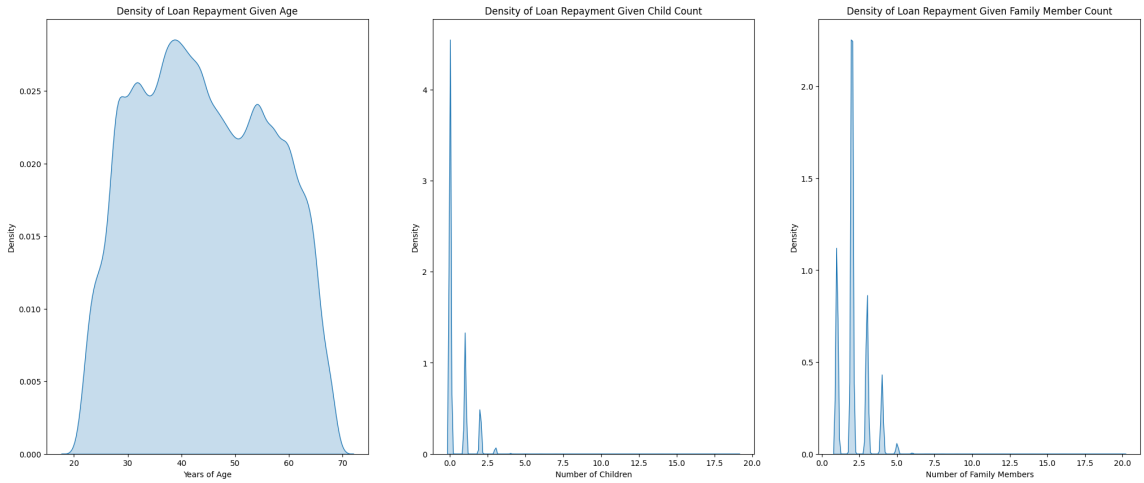


In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,3, sharex=False, figsize=(25,10))

# Set Figure Labels
ax[0].set_title('Density of Loan Repayment Given Age')
ax[1].set_title('Density of Loan Repayment Given Child Count')
ax[2].set_title('Density of Loan Repayment Given Family Member Count')

# Set Lables
ax[0].set_ylabel('Density')
ax[1].set_ylabel('Density')
ax[2].set_ylabel('Density')

# Set Lables
ax[0].set_xlabel('Years of Age')
ax[1].set_xlabel('Number of Children')
ax[2].set_xlabel('Number of Family Members')

# Set KDE
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'DAYS_BIRTH'] / 365 * -1,
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'CNT_CHILDREN'], label = '
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'CNT_FAM_MEMBERS'], label
```

Out[32]:

```
<Axes: title={'center': 'Density of Loan Repayment Given Family Member Cou
nt'}, xlabel='Number of Family Members', ylabel='Density'>
```



DISCUSSION

These visualizations give us some great insight into the general trends of where loan repayment is most common in these data. We can see that repayment is most common in individuals around 40 years old, with no children, and a family size around 2.

IMPORTANT

Since we are using a KDE or Kernal Density Estimate, this just shows where the highest amount of occurences happen, not who is most likely to do so. This instead will give us insight on where we might be able to reduce features to understand where people are not repaying their loans.

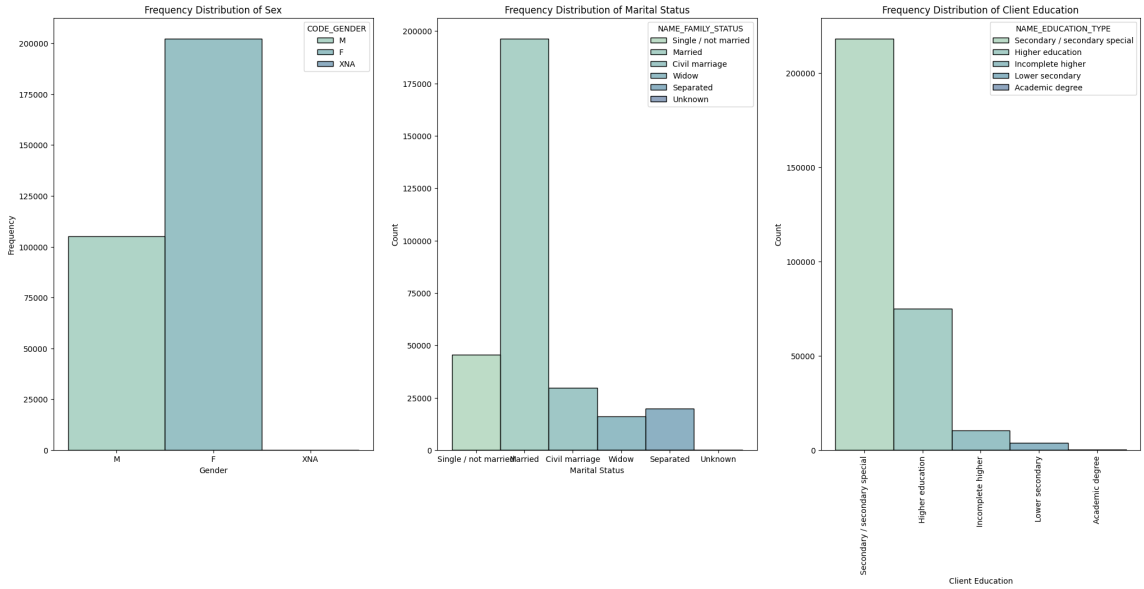
Lets now Take a look at the categorical side

In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,3, sharex=False, figsize=(25,10))

# Set Figure Labels
ax[0].set_title('Frequency Distribution of Sex')
ax[1].set_title('Frequency Distribution of Marital Status')
ax[2].set_title('Frequency Distribution of Client Education')

# Set Lables
ax[0].set_ylabel('Frequncy')
ax[0].set_ylabel('Frequency')
ax[0].set_ylabel('Frequency')

# Set Lables
ax[0].set_xlabel('Gender')
ax[1].set_xlabel('Marital Status')
ax[2].set_xlabel('Client Education')

# Set histogram
sns.histplot(ax = ax[0], data = df_app_train, palette="crest", x = "CODE_GENDER", hu
sns.histplot(ax = ax[1], data = df_app_train, palette="crest", x = "NAME_FAMILY_STAT
sns.histplot(ax = ax[2], data = df_app_train, palette="crest", x = "NAME_EDUCATION_T
plt.xticks(rotation=90)
```

Out[33]:

```
([0, 1, 2, 3, 4],
 [Text(0, 0, 'Secondary / secondary special'),
  Text(1, 0, 'Higher education'),
  Text(2, 0, 'Incomplete higher'),
  Text(3, 0, 'Lower secondary'),
  Text(4, 0, 'Academic degree')])
```

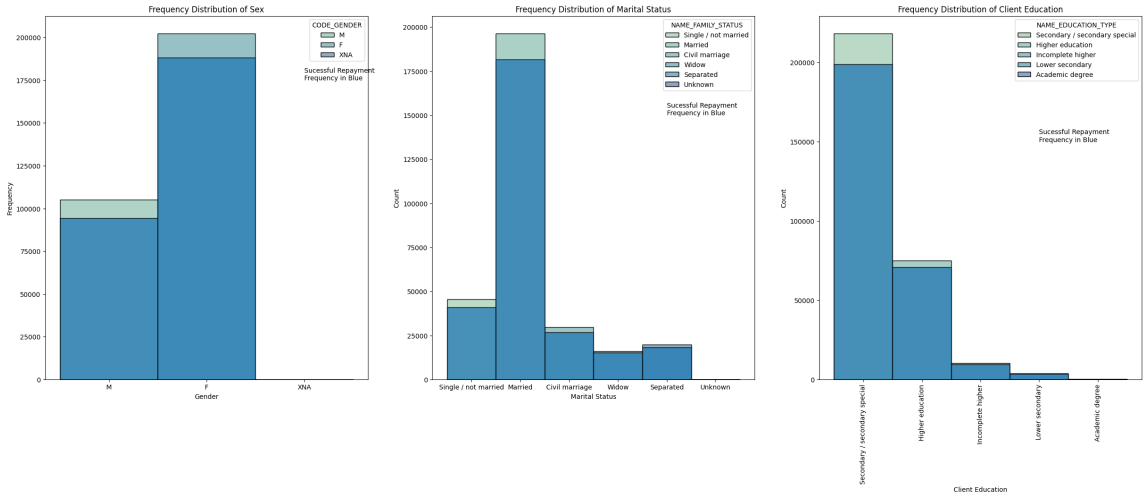


In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,3, sharex=False, figsize=(30,10))

# Set Figure Labels
ax[0].set_title('Frequency Distribution of Sex')
ax[1].set_title('Frequency Distribution of Marital Status')
ax[2].set_title('Frequency Distribution of Client Education')

# Set Lables
ax[0].set_ylabel('Frequncy')
ax[0].set_ylabel('Frequency')
ax[0].set_ylabel('Frequency')

# Set Lables
ax[0].set_xlabel('Gender')
ax[1].set_xlabel('Marital Status')
ax[2].set_xlabel('Client Education')

sns.histplot(ax = ax[0], data = df_app_train, palette="crest", x = "CODE_GENDER", hu
sns.histplot(ax = ax[1], data = df_app_train, palette="crest", x = "NAME_FAMILY_STAT
sns.histplot(ax = ax[2], data = df_app_train, palette="crest", x = "NAME_EDUCATION_T
ax1 = sns.histplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'CODE_GENDER'], lab
ax2 = sns.histplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'NAME_FAMILY_STATUS
ax3 = sns.histplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'NAME_EDUCATION_TYP
plt.xticks(rotation=90)

ax1.annotate("Sucessful Repayment\nFrequency in Blue", xy=('XNA', 175000))
ax2.annotate("Sucessful Repayment\nFrequency in Blue", xy=('Separated', 150000))
ax3.annotate("Sucessful Repayment\nFrequency in Blue", xy=('Lower secondary', 150000
```

Out[34]:

```
Text(Lower secondary, 150000, 'Sucessful Repayment\nFrequency in Blue')
```



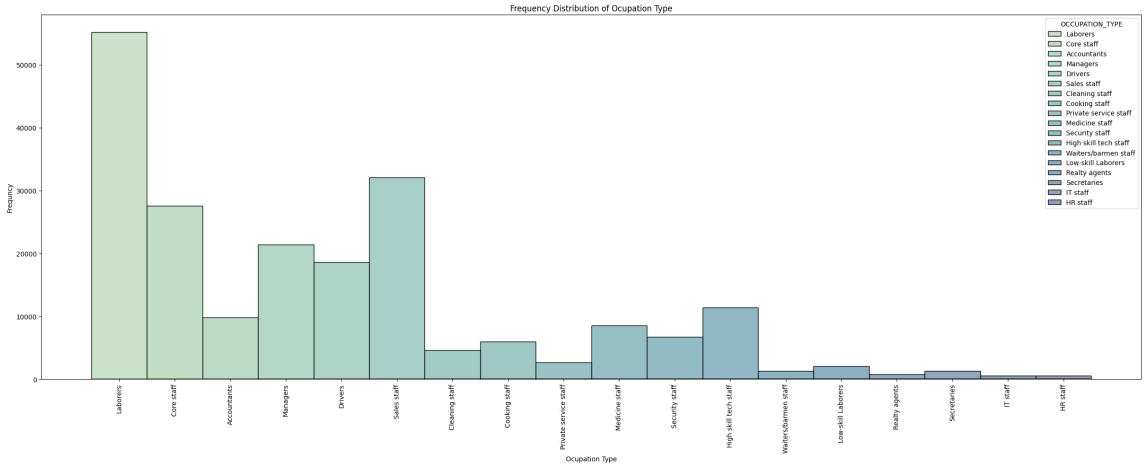
VEDA: Input Feature Visualiaztion: Occupation

In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,1, sharex=False, figsize=(30,10))

# Set Figure Labels
ax.set_title('Frequency Distribution of Ocupation Type')

# Set Lables
ax.set_ylabel('Frequncy')

# Set Lables
ax.set_xlabel('Ocupation Type')

sns.histplot(ax = ax, data = df_app_train, palette="crest", x = "OCCUPATION_TYPE", h
plt.xticks(rotation=90)
```

Out[35]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
 [Text(0, 0, 'Laborers'),
  Text(1, 0, 'Core staff'),
  Text(2, 0, 'Accountants'),
  Text(3, 0, 'Managers'),
  Text(4, 0, 'Drivers'),
  Text(5, 0, 'Sales staff'),
  Text(6, 0, 'Cleaning staff'),
  Text(7, 0, 'Cooking staff'),
  Text(8, 0, 'Private service staff'),
  Text(9, 0, 'Medicine staff'),
  Text(10, 0, 'Security staff'),
  Text(11, 0, 'High skill tech staff'),
  Text(12, 0, 'Waiters/barmen staff'),
  Text(13, 0, 'Low-skill Laborers'),
  Text(14, 0, 'Realty agents'),
  Text(15, 0, 'Secretaries'),
  Text(16, 0, 'IT staff'),
  Text(17, 0, 'HR staff')])
```

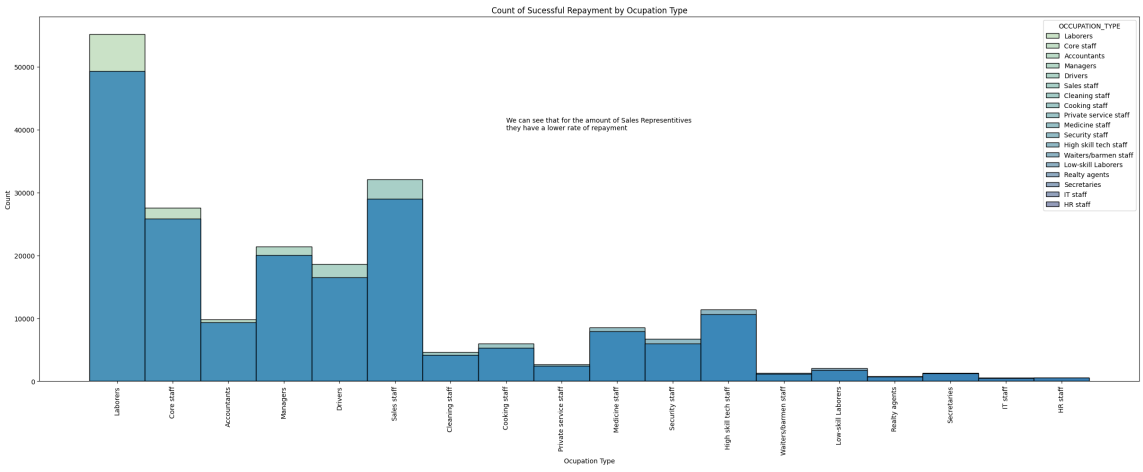


In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,1, sharex=False, figsize=(30,10))

# Set Figure Labels
ax.set_title('Count of Sucessful Repayment by Ocupation Type')

# Set Lables
ax.set_ylabel('Count')

# Set Lables
ax.set_xlabel('Ocupation Type')

ax = sns.histplot(ax = ax, data = df_app_train, palette="crest", x = "OCCUPATION_TYP
ax = sns.histplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'OCCUPATION_TYPE'],
ax.annotate("We can see that for the amount of Sales Representitives \nthey have a l
plt.xticks(rotation=90)
```

Out[36]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
 [Text(0, 0, 'Laborers'),
  Text(1, 0, 'Core staff'),
  Text(2, 0, 'Accountants'),
  Text(3, 0, 'Managers'),
  Text(4, 0, 'Drivers'),
  Text(5, 0, 'Sales staff'),
  Text(6, 0, 'Cleaning staff'),
  Text(7, 0, 'Cooking staff'),
  Text(8, 0, 'Private service staff'),
  Text(9, 0, 'Medicine staff'),
  Text(10, 0, 'Security staff'),
  Text(11, 0, 'High skill tech staff'),
  Text(12, 0, 'Waiters/barmen staff'),
  Text(13, 0, 'Low-skill Laborers'),
  Text(14, 0, 'Realty agents'),
  Text(15, 0, 'Secretaries'),
  Text(16, 0, 'IT staff'),
  Text(17, 0, 'HR staff')])
```



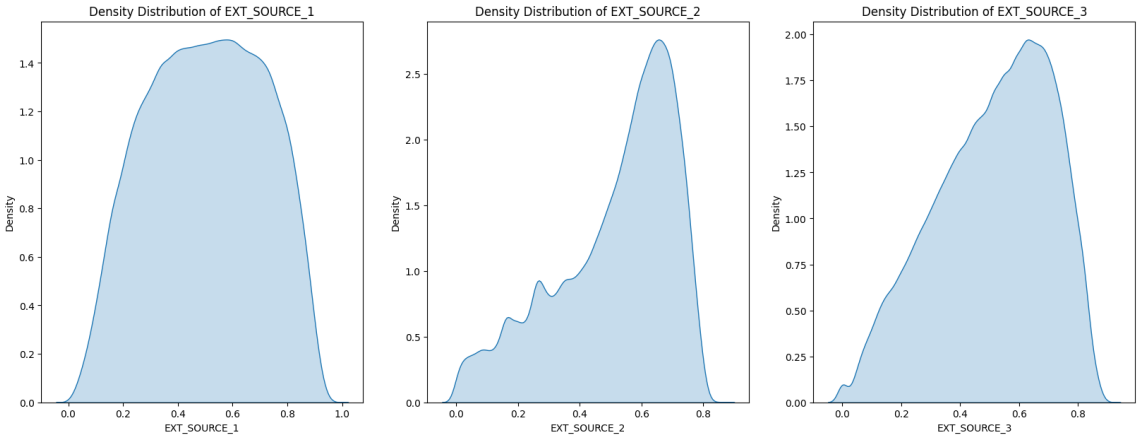
VEDA: Input Feature Visualiaztion: EXTERNAL SOURCE

In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,3, sharex=False, figsize=(20,7))

# Set Figure Labels
ax[0].set_title('Density Distribution of EXT_SOURCE_1')
ax[1].set_title('Density Distribution of EXT_SOURCE_2')
ax[2].set_title('Density Distribution of EXT_SOURCE_3')

# Set Lables
ax[0].set_ylabel('Density')
ax[1].set_ylabel('Density')
ax[2].set_ylabel('Density')

# Set Lables
ax[0].set_xlabel('EXT_SOURCE_1')
ax[1].set_xlabel('EXT_SOURCE_2')
ax[2].set_xlabel('EXT_SOURCE_3')

# Set histogram
sns.kdeplot(df_app_train["EXT_SOURCE_1"], ax=ax[0],fill=True)
sns.kdeplot(df_app_train["EXT_SOURCE_2"], ax=ax[1],fill=True)
sns.kdeplot(df_app_train["EXT_SOURCE_3"], ax=ax[2],fill=True)
```

Out[37]:

```
<Axes: title={'center': 'Density Distribution of EXT_SOURCE_3'}, xlabel='E
XT_SOURCE_3', ylabel='Density'>
```



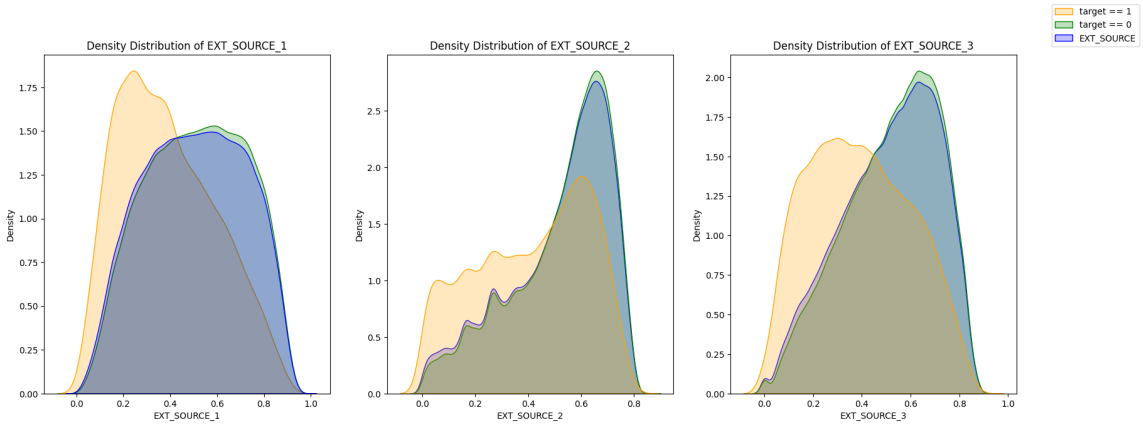
Lets now observe the target density over these external data sources to see if there may be any interesting distributions or trends.

In [ ]:

```
# set up fig
fig, ax = plt.subplots(1,3, sharex=False, figsize=(20,7))

# Set Figure Labels
ax[0].set_title('Density Distribution of EXT_SOURCE_1')
ax[1].set_title('Density Distribution of EXT_SOURCE_2')
ax[2].set_title('Density Distribution of EXT_SOURCE_3')

# Set Lables
ax[0].set_ylabel('Density')
ax[1].set_ylabel('Density')
ax[2].set_ylabel('Density')

# Set Lables
ax[0].set_xlabel('EXT_SOURCE_1')
ax[1].set_xlabel('EXT_SOURCE_2')
ax[2].set_xlabel('EXT_SOURCE_3')

# Set kdeplot of targets

sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 1, 'EXT_SOURCE_1'], label = '
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'EXT_SOURCE_1'], label = '
sns.kdeplot(df_app_train["EXT_SOURCE_1"],label="EXT_SOURCE", ax=ax[0], fill=True, cc
fig.legend()

sns.kdeplot(df_app_train["EXT_SOURCE_2"],label="EXT_SOURCE_2", ax=ax[1], fill=True,
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'EXT_SOURCE_2'], label = '
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 1, 'EXT_SOURCE_2'], label = '

sns.kdeplot(df_app_train["EXT_SOURCE_3"],label="EXT_SOURCE_3", ax=ax[2], fill=True,
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 0, 'EXT_SOURCE_3'], label = '
sns.kdeplot(df_app_train.loc[df_app_train['TARGET'] == 1, 'EXT_SOURCE_3'], label = '
```

Out[38]:

```
<Axes: title={'center': 'Density Distribution of EXT_SOURCE_3'}, xlabel='E
XT_SOURCE_3', ylabel='Density'>
```



DISCUSION

We can see that in each of teh EXT_SOURCE visualizations that EXT_SOURCE seems to follow the same density as target 0. However, the largest seperation between these two features and the target value of 1 is shown in EXT_SOURCE_3. Although slight, this may give some insight into how these data could be use later.

VEDA: Correlation Visualiaztion: Demographic Numerical

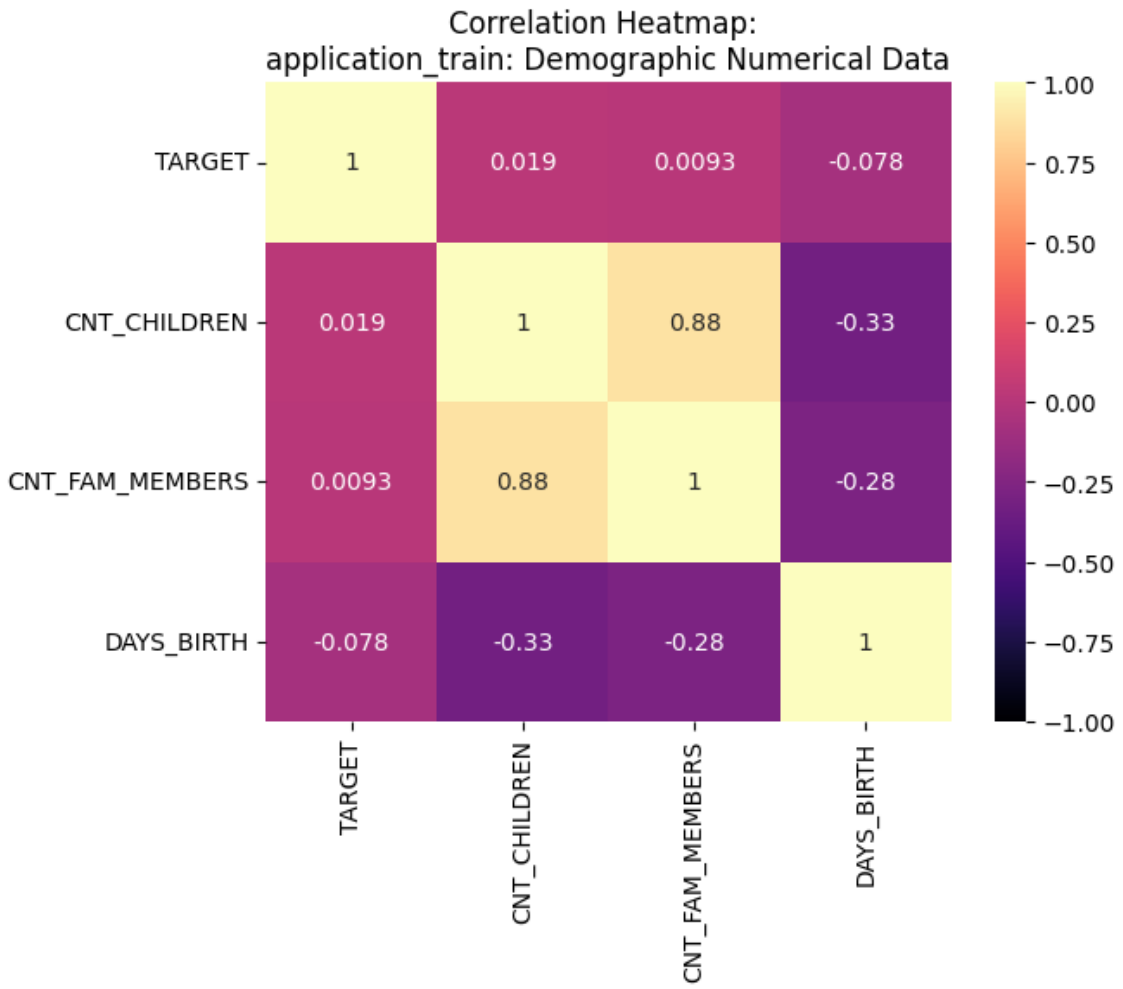
In [ ]:

```
# Lets see the correlation map for the numerical demographics
corr_occ_data = df_app_train[["TARGET", "CNT_CHILDREN", "CNT_FAM_MEMBERS", "DAYS_BIR
corr_occ_data["DAYS_BIRTH"] = abs(corr_occ_data["DAYS_BIRTH"])
corr_occ_data = corr_occ_data.corr()
sns.heatmap(corr_occ_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \napplication_train: Demographic Numerical Data")
```

```
<ipython-input-39-f80d2e77f988>:3: 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-doc
s/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)
  corr_occ_data["DAYS_BIRTH"] = abs(corr_occ_data["DAYS_BIRTH"])
```

Out[39]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \napplication_train: Demographic Nume
rical Data')
```



DISCUSSION

Unfortunatly there are not many insights we can pull from these data correlation coefficients. The largest correlation coefficent to show is DAYS_BIRTH. This negitive correlation shows that the older the client the more likely they are to succesfully repay since a positive correlation would be increasing with target == 1, failure to repay.

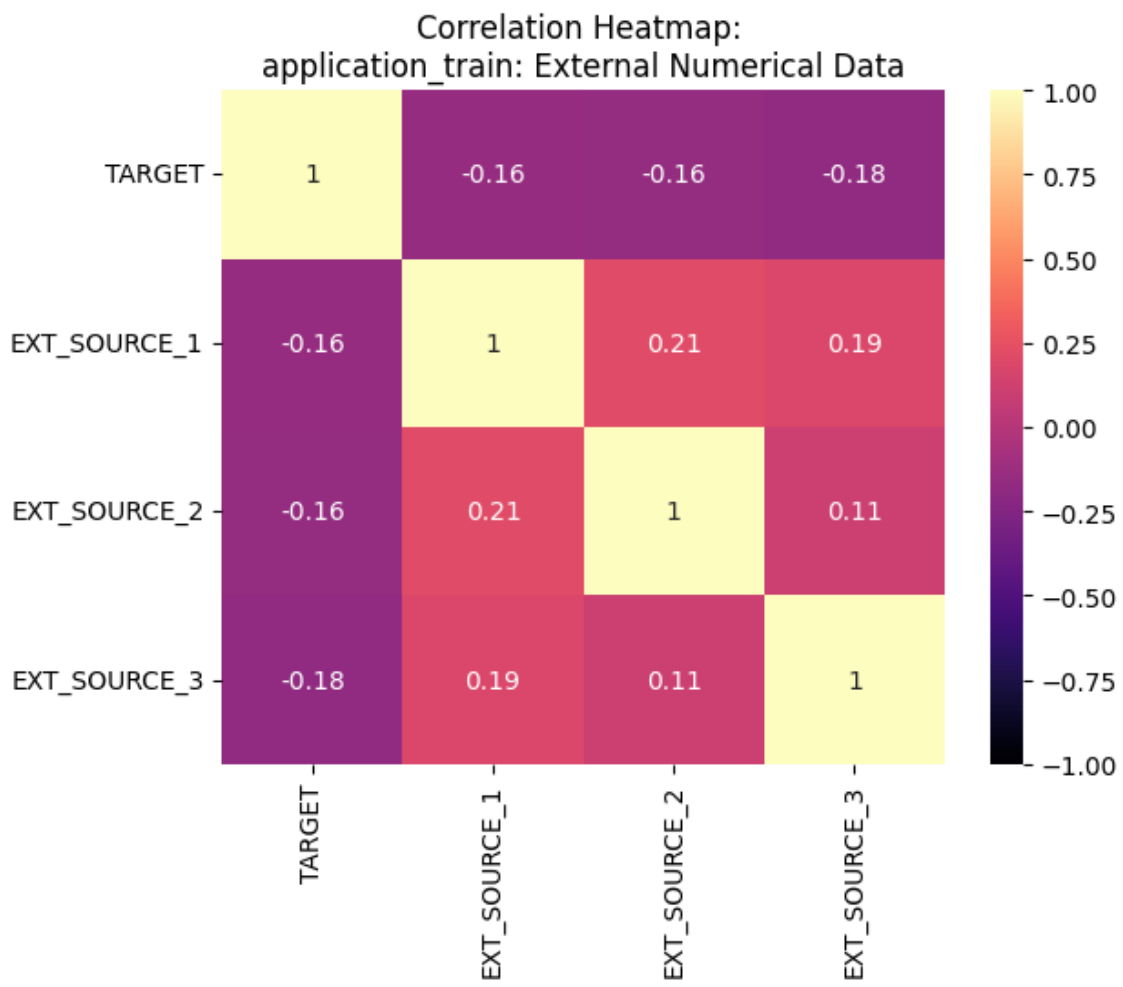
VEDA: Correlation Visualiaztion: EXTERNAL Numerical

In [ ]:

```
# Lets see the correlation map for the numerical external data
corr_extern_data = df_app_train[["TARGET", "EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOUR
corr_extern_data = corr_extern_data.corr()
sns.heatmap(corr_extern_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \napplication_train: External Numerical Data")
```

Out[40]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \napplication_train: External Numeric
al Data')
```



DISCUSSION

This heatmap provides some insight into the correlations of these data to the target value. The most evident insight about this visualization is that EXT_SOURCE_3 has the largest negitive correlation to the TARGET.

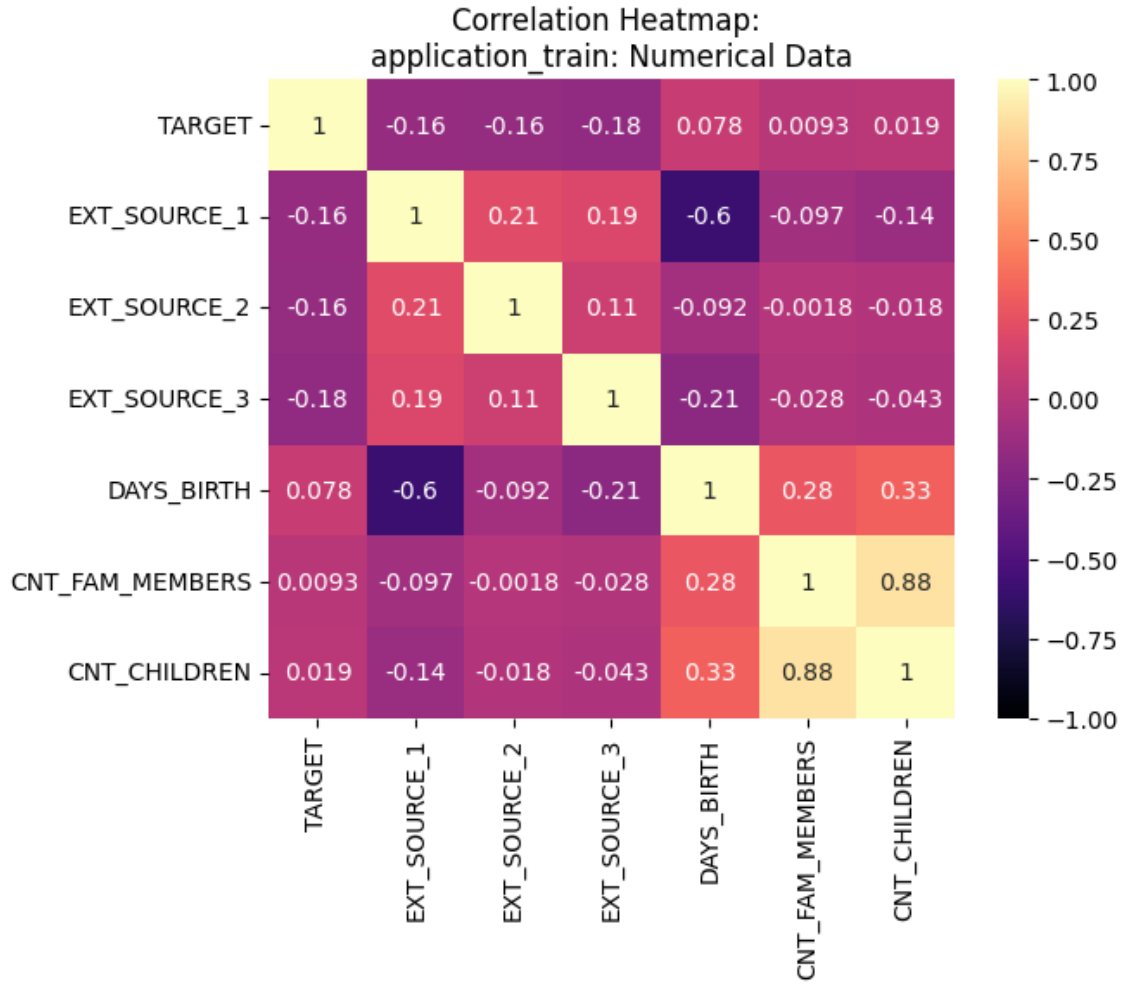
This means that this feature, has a positive correlation to repayment of the loan, since it has a negitive

In [ ]:

```
# Lets put these 2 heatmaps together for a better summary
corr_full_data = df_app_train[["TARGET", "EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOURCE
corr_full_data = corr_full_data.corr()
sns.heatmap(corr_full_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \napplication_train: Numerical Data")
```

Out[41]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \napplication_train: Numerical Data')
```



VEDA: Missing Value Analysis

In [ ]:

```
# Import Libraries
import missingno as msno
```

In [ ]:

```
# Numerical Analysis

# CITATION: Parts of code taken from HCDR_baseline_submission_phase2 starter code

missing_percentage = (df_app_train.isnull().sum() / df_app_train.isnull().count() *
missing_count = df_app_train.isna().sum().sort_values(ascending = False)
missing_app_table = pd.concat([missing_percentage, missing_count], axis=1, keys=["Mi
missing_app_table.head(20)
```

Out[43]:

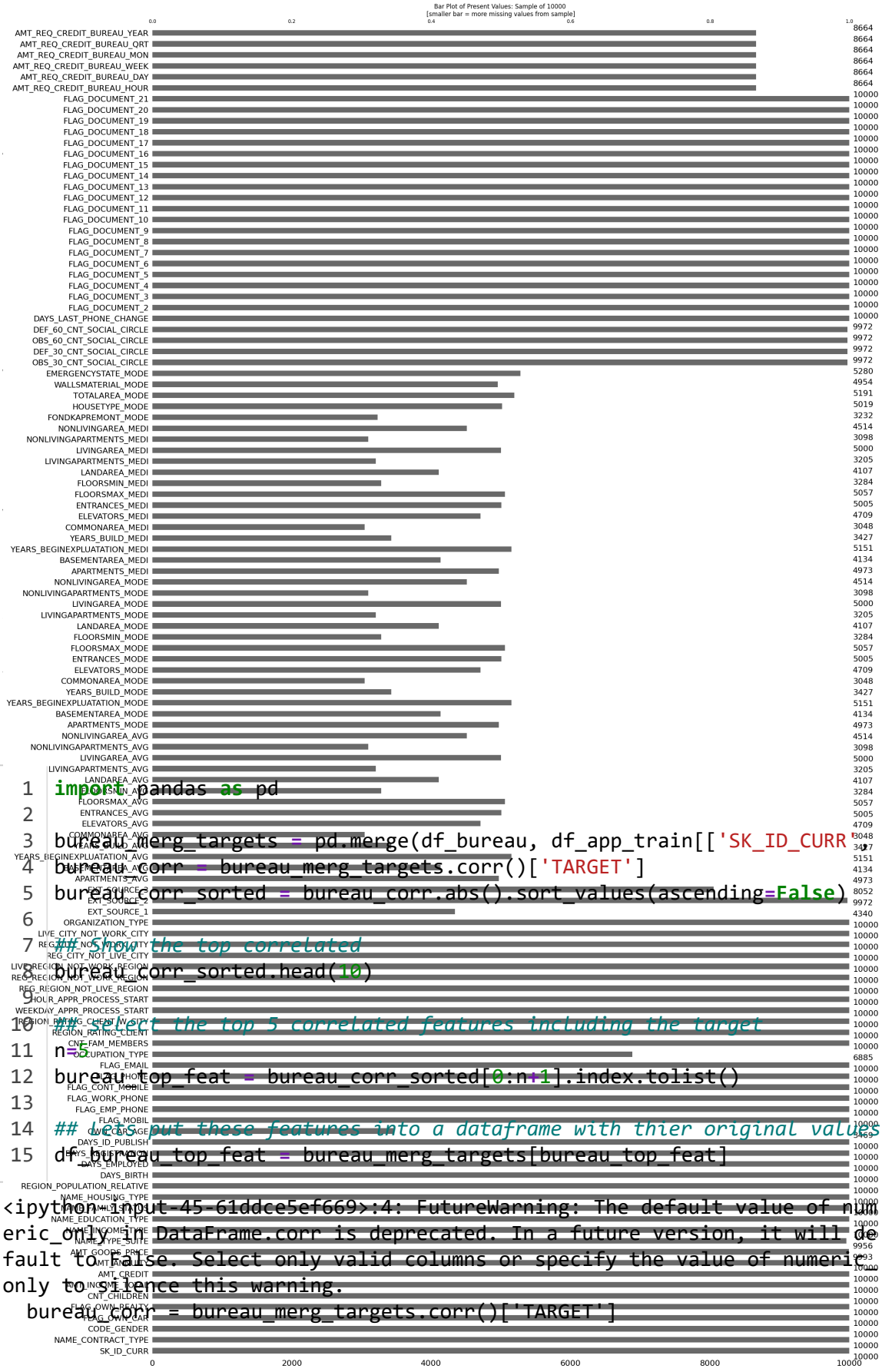
	Missing (%)	Missing (Count)
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

In [ ]:

```
# Visualization of these Misssing Values
fig, ax = plt.subplots(1,1, sharex=False, figsize=(20,10))
ax = msno.bar(df_app_train.drop("TARGET", axis='columns').sample(10000))
ax.set_title("Bar Plot of Present Values: Sample of 10000 \n[smaller bar = more miss
```

Out[44]:

```
Text(0.5, 1.0, 'Bar Plot of Present Values: Sample of 10000 \n[smaller bar
= more missing values from sample]')
```



t most of the
s that have a
This may be

features are
ons and other

```
import pandas as pd

bureau_merg_targets = pd.merge(df_bureau, df_app_train[['SK_ID_CURR', 'TARGET']], or
bureau_corr = bureau_merg_targets.corr()['TARGET']
bureau_corr_sorted = bureau_corr.abs().sort_values(ascending=False)

## Show the top correlated
bureau_corr_sorted.head(10)

## Select the top 5 correlated features including the target
n=5
bureau_top_feat = bureau_corr_sorted[0:n+1].index.tolist()

## Output these features into a dataframe with thier original values with the targ
df_bureau_top_feat = bureau_merg_targets[bureau_top_feat]
```

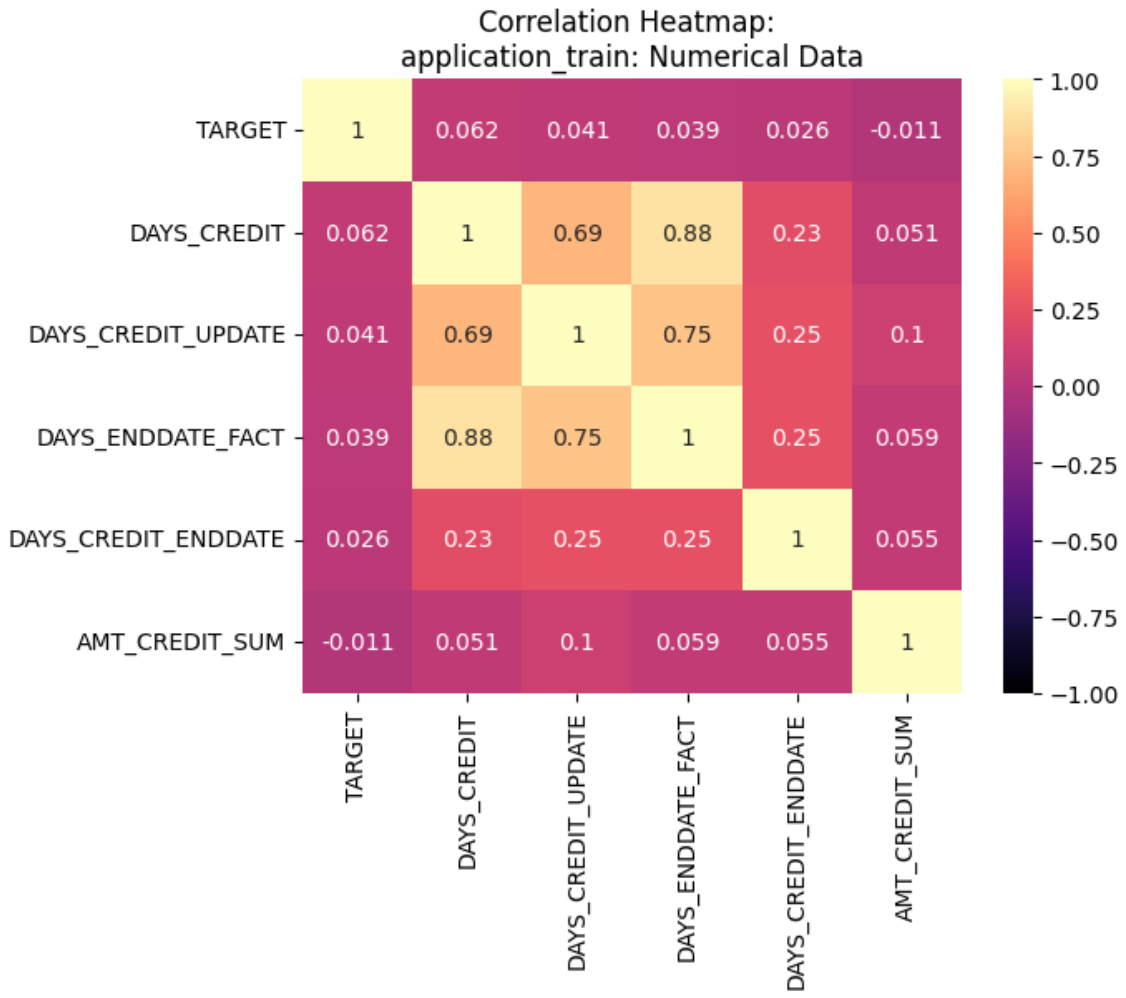
```
<ipython-input-45-61ddce5ef669>:4: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  bureau_corr = bureau_merg_targets.corr()['TARGET']
```

In [ ]:

```
corr_data = df_bureau_top_feat
corr_data = corr_data.corr()
sns.heatmap(corr_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \napplication_train: Numerical Data")
```

Out[46]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \napplication_train: Numerical Data')
```



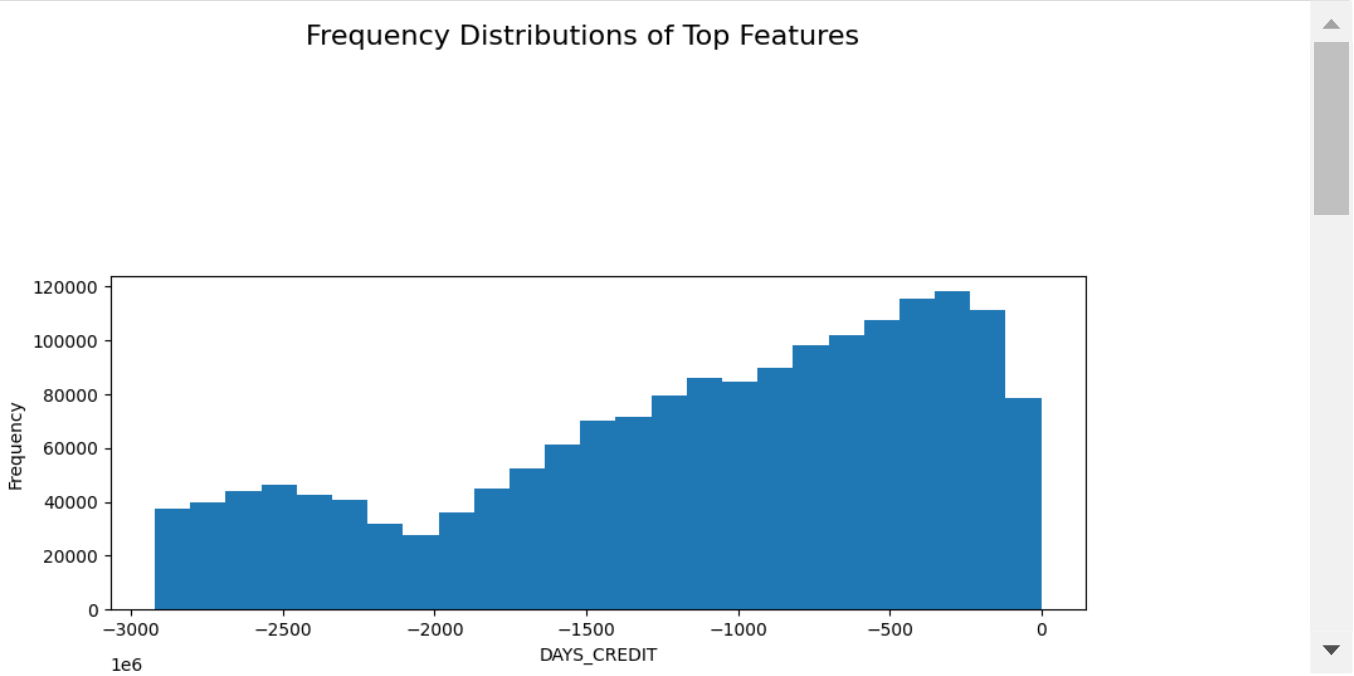
DISCUSSION

This heat map gives us a great sense of the inital features we will be interested in from the bureau.csv table, top amoung those are the DAYS_CREDIT, DAYS_CREDIT_UPDATE, DAYS_ENDATE_FACT, and DAYS_CREDIT_ENDATE. AMT_CREDIT_SUM has the lowest degree of correlation, so it could possibly be ignored.

VEDA: Distribution Anlaysis: beureau.csv

In [ ]:

```
# Lets look at the distributions of these data
fig, axs = plt.subplots(nrows=n, figsize=(10,20))
fig.suptitle('Frequency Distributions of Top Features', fontsize=16)
for i, feature in enumerate(bureau_top_feat):
  axs[i - 1].hist(df_bureau_top_feat[feature], bins=25)
  axs[i - 1].set_xlabel(feature)
  axs[i - 1].set_ylabel("Frequency")

plt.show()
```



VEDA: Missing Value Anlaysis: beureau.csv

In [ ]:

```
# Numerical Analysis

# CITATION: Parts of code taken from HCDR_baseline_submission_phase2 starter code

missing_percentage = (df_bureau.isnull().sum() / df_bureau.isnull().count() * 100).s
missing_count = df_bureau.isna().sum().sort_values(ascending = False)
missing_app_table = pd.concat([missing_percentage, missing_count], axis=1, keys=["Mi
missing_app_table.head(20)
```

Out[48]:

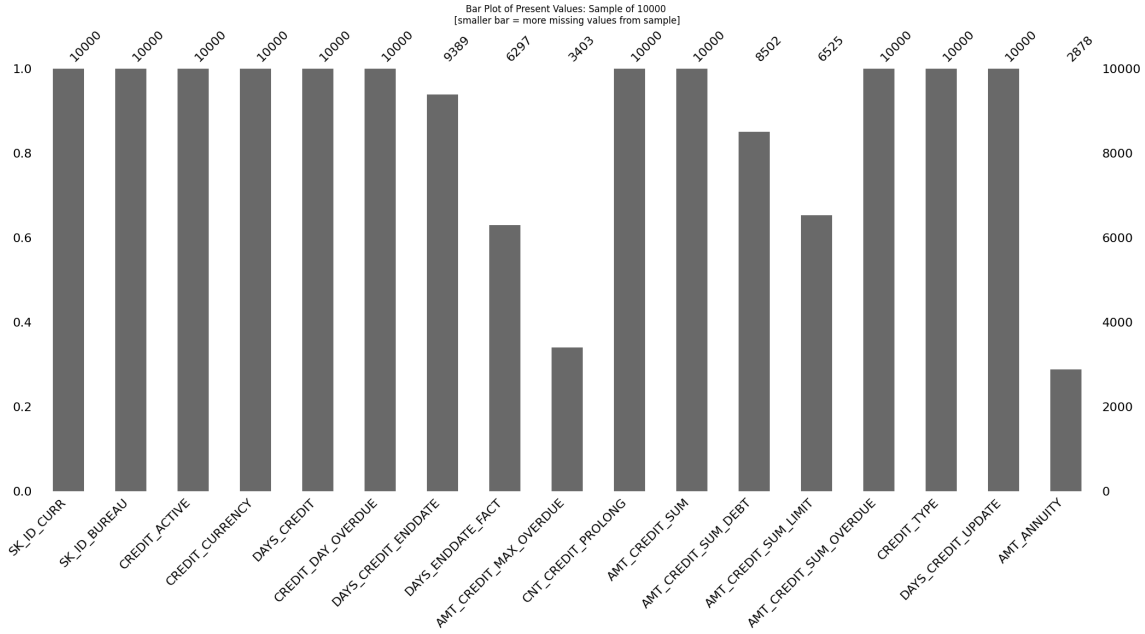
	Missing (%)	Missing (Count)
AMT_ANNUITY	71.47	1226791
AMT_CREDIT_MAX_OVERDUE	65.51	1124488
DAYS_ENDDATE_FACT	36.92	633653
AMT_CREDIT_SUM_LIMIT	34.48	591780
AMT_CREDIT_SUM_DEBT	15.01	257669
DAYS_CREDIT_ENDDATE	6.15	105553
AMT_CREDIT_SUM	0.00	13
CREDIT_ACTIVE	0.00	0
CREDIT_CURRENCY	0.00	0
DAYS_CREDIT	0.00	0
CREDIT_DAY_OVERDUE	0.00	0
SK_ID_BUREAU	0.00	0
CNT_CREDIT_PROLONG	0.00	0
AMT_CREDIT_SUM_OVERDUE	0.00	0
CREDIT_TYPE	0.00	0
DAYS_CREDIT_UPDATE	0.00	0
SK_ID_CURR	0.00	0

In [ ]:

```
# Visualization of these Misssing Values
fig, ax = plt.subplots(1,1, sharex=False, figsize=(20,10))
ax = msno.bar(df_bureau.sample(10000))
ax.set_title("Bar Plot of Present Values: Sample of 10000 \n[smaller bar = more miss
```

Out[49]:

```
Text(0.5, 1.0, 'Bar Plot of Present Values: Sample of 10000 \n[smaller bar
= more missing values from sample]')
```



DISCUSSION

We can see from the sample that AMT_CREDIT_MAX_OVERDUE and AMT_ANNUITY both have the most missing values from the data table.

VEDA: Input Feature Visualiaztion (bureau_balance.csv)

VEDA: Correlation Anlaysis: beureau_balance.csv

In [ ]:

```
import pandas as pd
bur_bal_id_merge = pd.merge(df_bureau_bal, df_bureau[['SK_ID_BUREAU', 'SK_ID_CURR']]
bur_bal_target = pd.merge(bur_bal_id_merge, df_app_train[['SK_ID_CURR', 'TARGET']],
bureau_bal_corr = bur_bal_target.corr()['TARGET']
bureau_bal_corr_sorted = bureau_bal_corr.abs().sort_values(ascending=False)

bureau_bal_top_feat = bureau_bal_corr_sorted[0:2].index.tolist()
bureau_bal_top_feat = bur_bal_target[bureau_bal_top_feat]
```

```
<ipython-input-50-9536630e7185>:4: FutureWarning: The default value of num
eric_only in DataFrame.corr is deprecated. In a future version, it will de
fault to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  bureau_bal_corr = bur_bal_target.corr()['TARGET']
```

In [ ]:

```
bureau_bal_top_feat
```

Out[51]:

	TARGET	MONTHS_BALANCE
0	0.0	0
1	0.0	-1
2	0.0	-2
3	0.0	-3
4	0.0	-4
...	...	...
27299920	1.0	-47
27299921	1.0	-48
27299922	1.0	-49
27299923	1.0	-50
27299924	1.0	-51

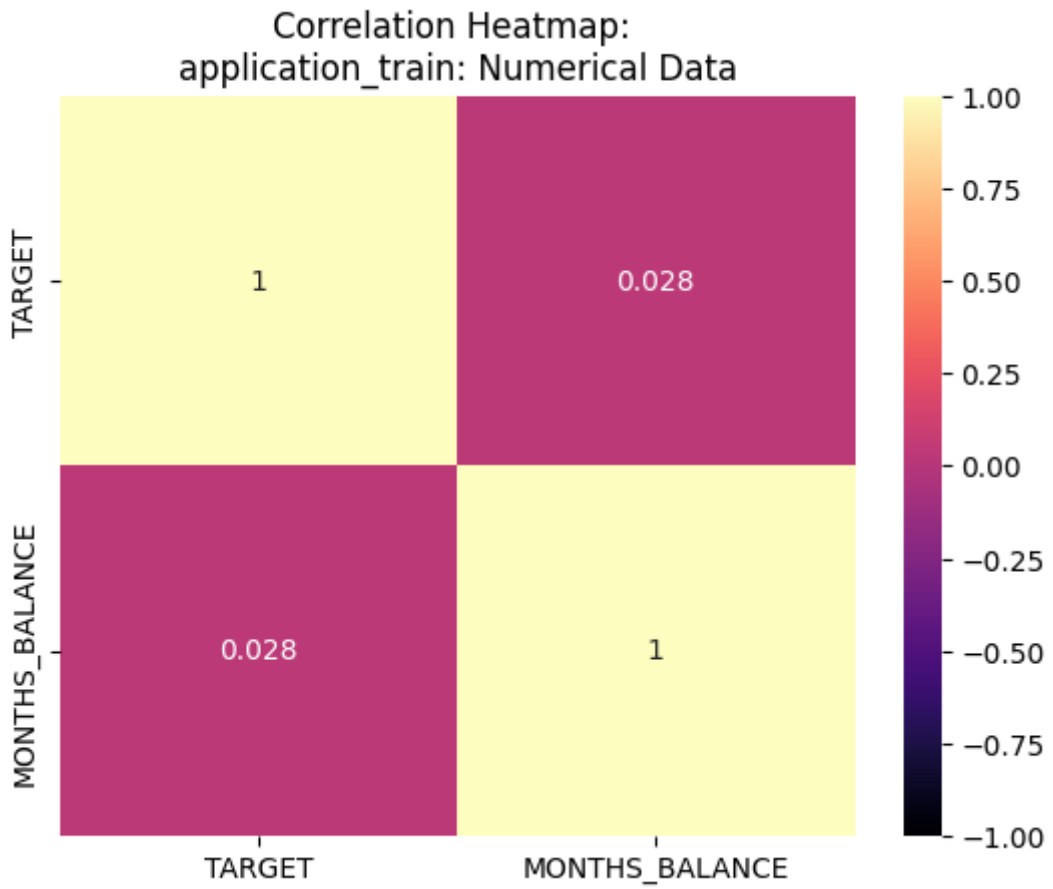
27299925 rows × 2 columns

In [ ]:

```
corr_data = bureau_bal_top_feat
corr_data = corr_data.corr()
sns.heatmap(corr_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \napplication_train: Numerical Data")
```

Out[52]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \napplication_train: Numerical Data')
```



DISCUSSION

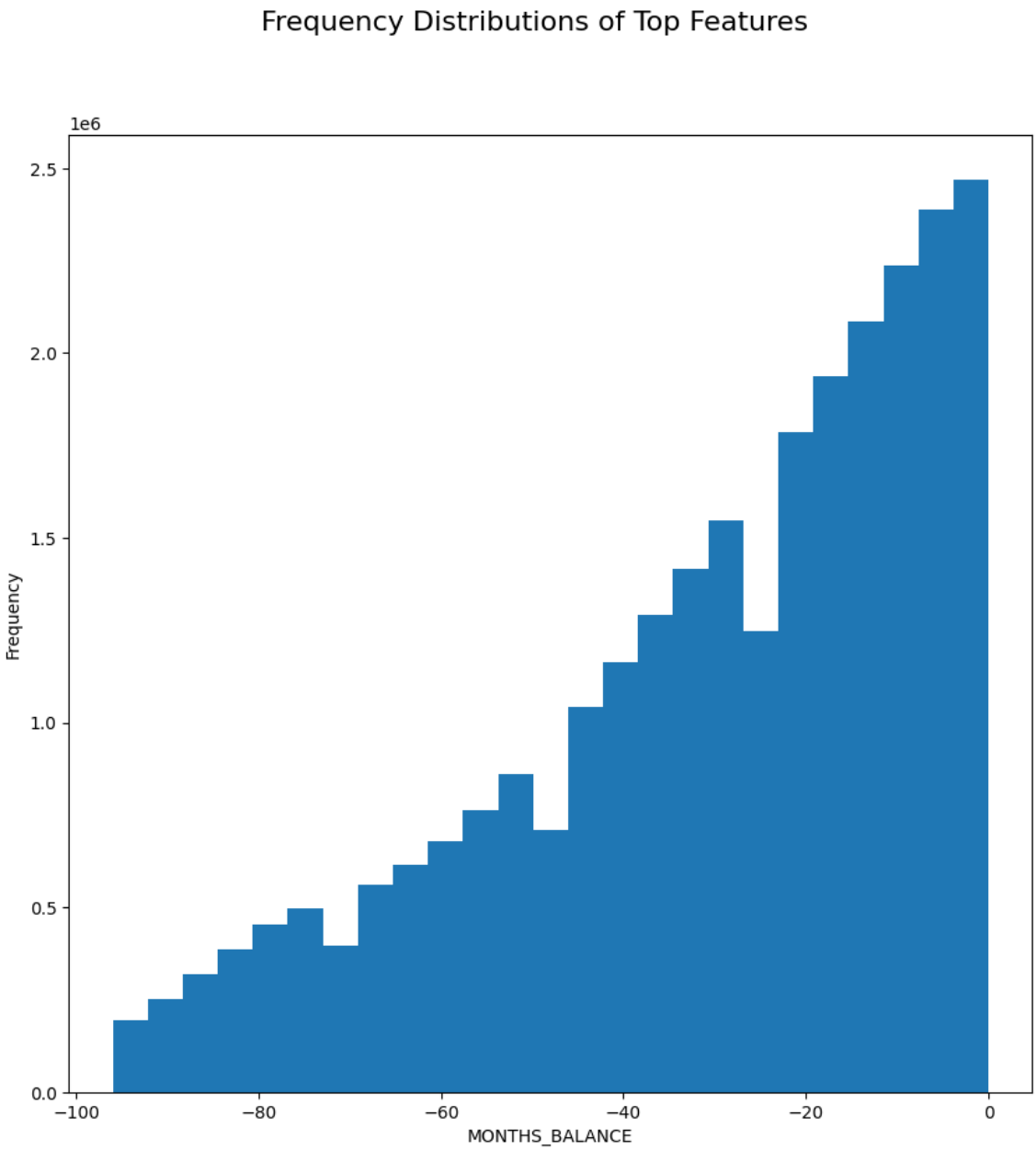
From this heat map and the previous correlation analysis, we can see that the only feasible feature from the buraeu_balance table is the MONTHS_BLANCE. After further analysis it can be said that this feature does have some correlation positivly with the target, meaning that as the MONTHS_BALANCE increases there is a decrease in the rate of repayment

---

VEDA: Distribution Anlaysis: beureau_balance.csv

In [ ]:

```
# Lets look at the distributions of these data
fig, axs = plt.subplots(1, figsize=(10,10))
fig.suptitle('Frequency Distributions of Top Features', fontsize=16)
axs.hist(df_bureau_bal['MONTHS_BALANCE'], bins=25)
axs.set_xlabel("MONTHS_BALANCE")
axs.set_ylabel("Frequency")

plt.show()
```



DISCUSSION

Observing the top feature of the buraeu_balance table, we can see that the distribution is unimodal, meaning that there seems to be a large grouping towards the 0 value. This imbalance in the distribution could help us later handle the values of this feauture upon implementation.

VEDA: Missing Value Anlaysis: beureau_balance.csv

There are no missing values in this table

VEDA: Input Feature Visualiaztion (POS_CASH_balance.csv)

VEDA: Correlation Anlaysis: POS_CASH_balance.csv

In [ ]:

```
import pandas as pd

pos_cash_target_merge = pd.merge(df_pos_cash_bal, df_app_train[['SK_ID_CURR', 'TARGE
pos_cash_corr = pos_cash_target_merge.corr()['TARGET']
pos_cash_corr_sorted = pos_cash_corr.abs().sort_values(ascending=False)

## Show the top correlated
pos_cash_corr_sorted.head(10)

## select the top 5 correlated features including the target
n=4
pos_cash_top_feat_list = pos_cash_corr_sorted[0:n+1].index.tolist()

## Lets put these features into a dataframe with thier original values with the targ
pos_cash_top_feat = pos_cash_target_merge[pos_cash_top_feat_list]
```

```
<ipython-input-54-56a417d1a489>:4: FutureWarning: The default value of num
eric_only in DataFrame.corr is deprecated. In a future version, it will de
fault to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  pos_cash_corr = pos_cash_target_merge.corr()['TARGET']
```

In [ ]:

```
print(pos_cash_top_feat_list)
```

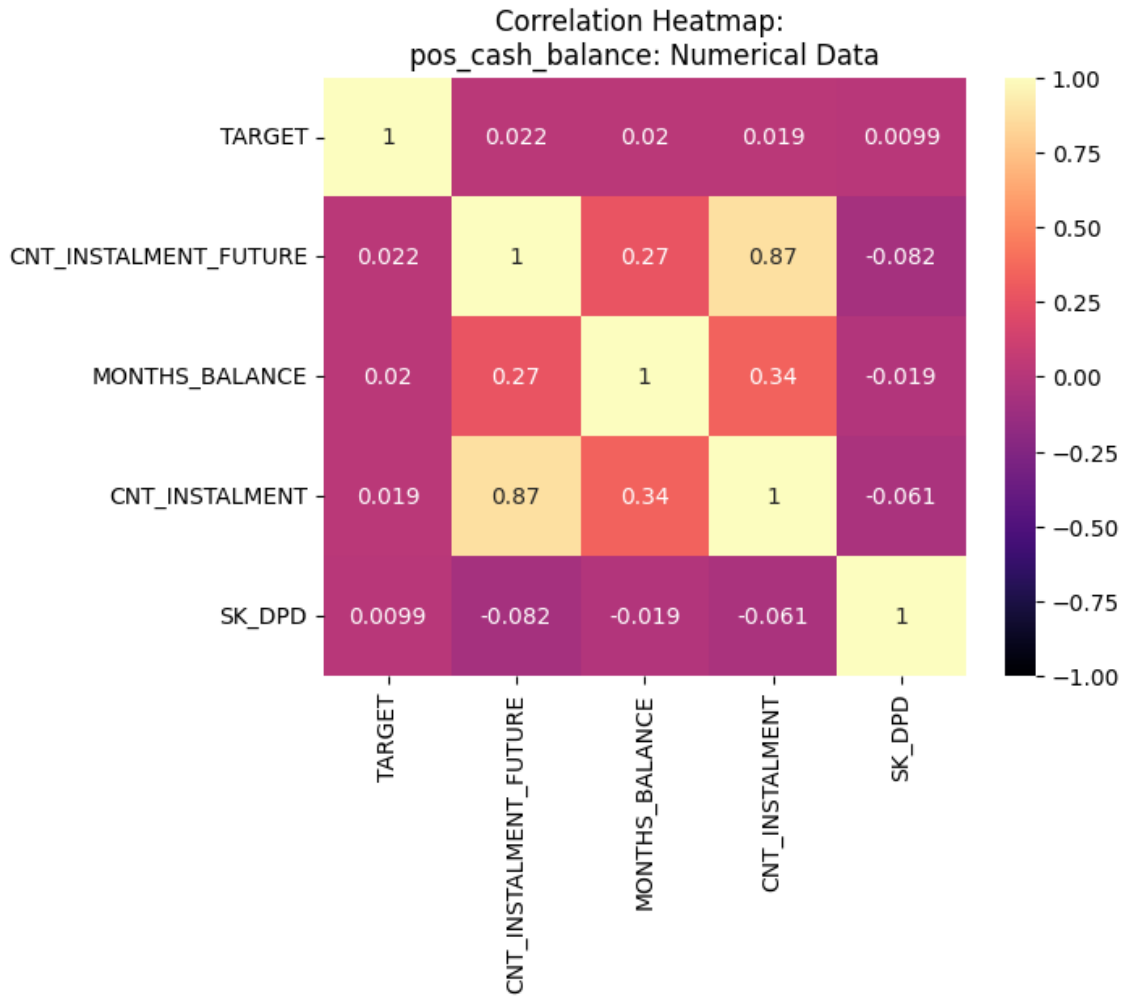
```
['TARGET', 'CNT_INSTALMENT_FUTURE', 'MONTHS_BALANCE', 'CNT_INSTALMENT', 'S
K_DPD']
```

In [ ]:

```
corr_data = pos_cash_top_feat
corr_data = corr_data.corr()
sns.heatmap(corr_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \npos_cash_balance: Numerical Data")
```

Out[56]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \npos_cash_balance: Numerical Data')
```



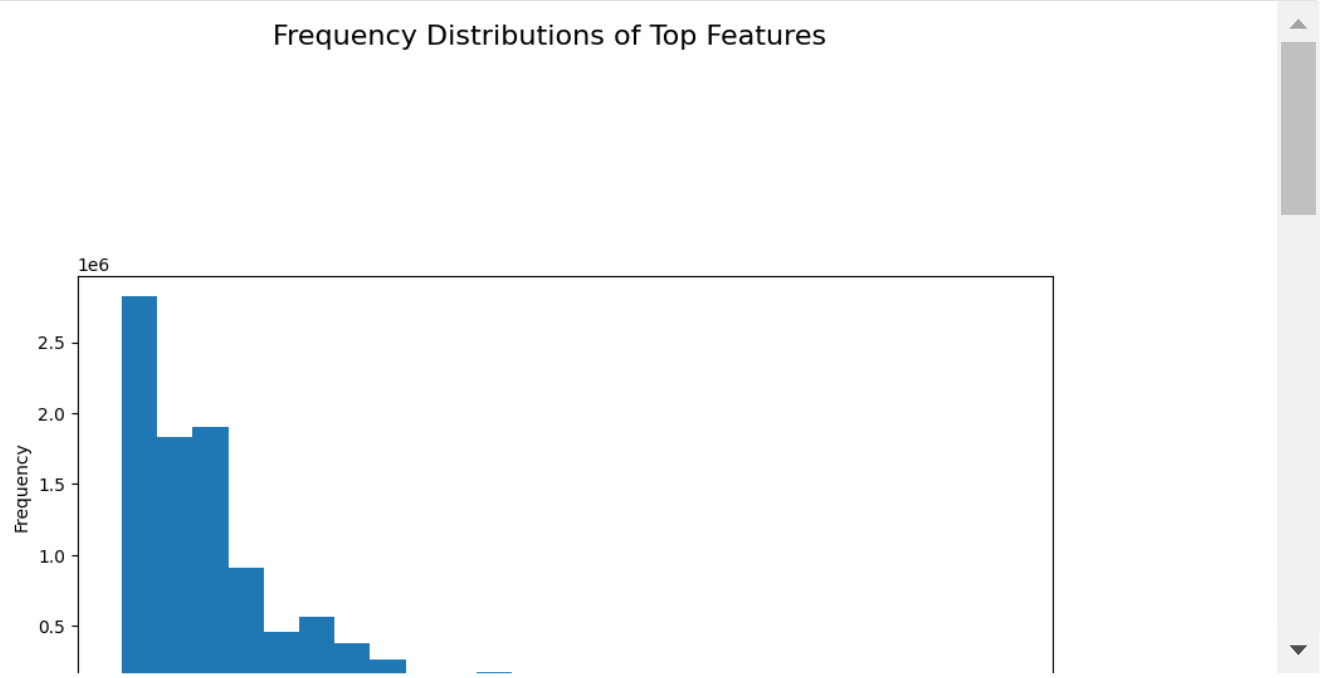
DISCUSSION

From this heat map we can see that CNT_INSTALMENT_FUTURE, MONTHS_BALANCE, and CNT_INSTALMENT are all features that have some correlation positivly to the target variable. This means that an increase in any one of these features should see a higher rate in the failure to repay the loan.

VEDA: Distribution Anlaysis: POS_CASH_balance.csv

In [ ]:

```
# Lets look at the distributions of these data
fig, axs = plt.subplots(nrows=n, figsize=(10,20))
fig.suptitle('Frequency Distributions of Top Features', fontsize=16)
for i, feature in enumerate(pos_cash_top_feat_list):
  axs[i - 1].hist(pos_cash_top_feat[feature], bins=25)
  axs[i - 1].set_xlabel(feature)
  axs[i - 1].set_ylabel("Frequency")

plt.show()
```



DISCUSSION

We can see from these distributions that all of these are imbalanced unimodal distributions. This is important to take into account when we move to the stage of handling these features in feature selection and preprocessing.

VEDA: Missing Value Anlaysis: POS_CASH_balance.csv

In [ ]:

```
# Numerical Analysis

# CITATION: Parts of code taken from HCDR_baseline_submission_phase2 starter code

missing_percentage = (df_pos_cash_bal.isnull().sum() / df_pos_cash_bal.isnull().cour
missing_count = df_pos_cash_bal.isna().sum().sort_values(ascending = False)
missing_app_table = pd.concat([missing_percentage, missing_count], axis=1, keys=["Mi
missing_app_table.head(20)
```

Out[58]:

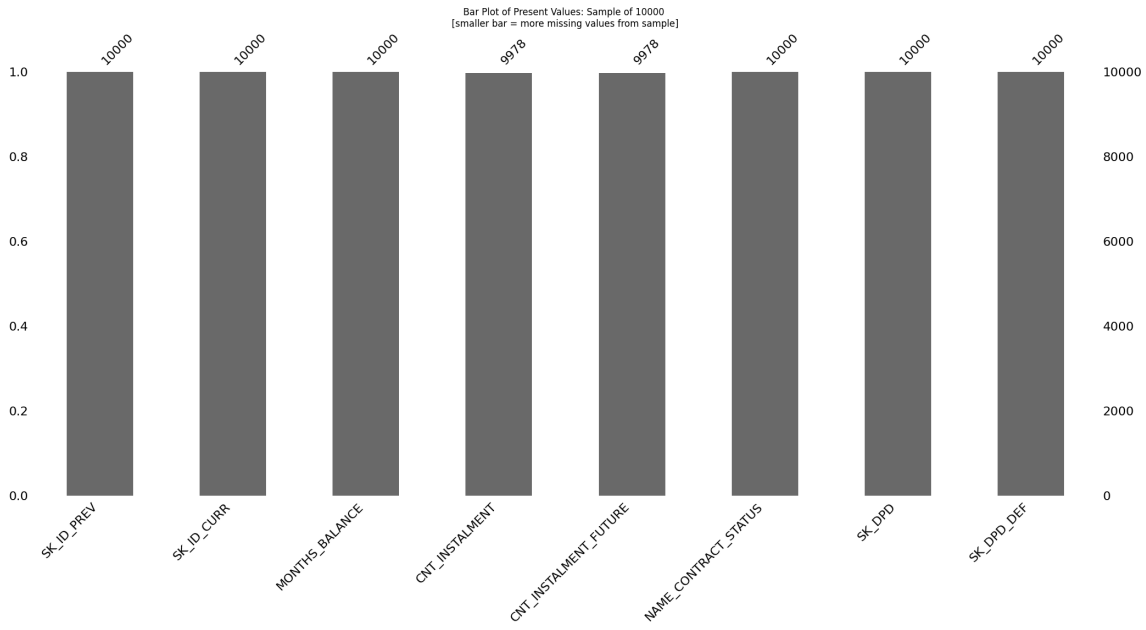
	Missing (%)	Missing (Count)
CNT_INSTALMENT_FUTURE	0.26	26087
CNT_INSTALMENT	0.26	26071
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
MONTHS_BALANCE	0.00	0
NAME_CONTRACT_STATUS	0.00	0
SK_DPD	0.00	0
SK_DPD_DEF	0.00	0

In [ ]:

```
# Visualization of these Misssing Values
fig, ax = plt.subplots(1,1, sharex=False, figsize=(20,10))
ax = msno.bar(df_pos_cash_bal.sample(10000))
ax.set_title("Bar Plot of Present Values: Sample of 10000 \n[smaller bar = more miss
```

Out[59]:

```
Text(0.5, 1.0, 'Bar Plot of Present Values: Sample of 10000 \n[smaller bar
= more missing values from sample]')
```



DISCUSSION

From this graph and the previous numerical analysis, we can see the only features missing values are the CNT_INSTALMETNS_FUTURE and CNT_INSTALMENTS which are both features we are interested in, so imputing this later will be a worth while task if the percentage was higher.

VEDA: Input Feature Visualiaztion (credit_card_balance.csv)

VEDA: Correlation Analysis: credict_card_balance.csv

In [ ]:

```
import pandas as pd

pos_credit_target_merge = pd.merge(df_credit_card_bal, df_app_train[['SK_ID_CURR', '
pos_credit_corr = pos_credit_target_merge.corr()['TARGET']
pos_credit_corr_sorted = pos_credit_corr.abs().sort_values(ascending=False)

## Show the top correlated
pos_credit_corr_sorted.head(10)

## select the top 5 correlated features including the target
n=4
pos_credit_top_feat_list = pos_credit_corr_sorted[0:n+1].index.tolist()

## Lets put these features into a dataframe with thier original values with the targ
pos_credit_top_feat = pos_credit_target_merge[pos_credit_top_feat_list]
```

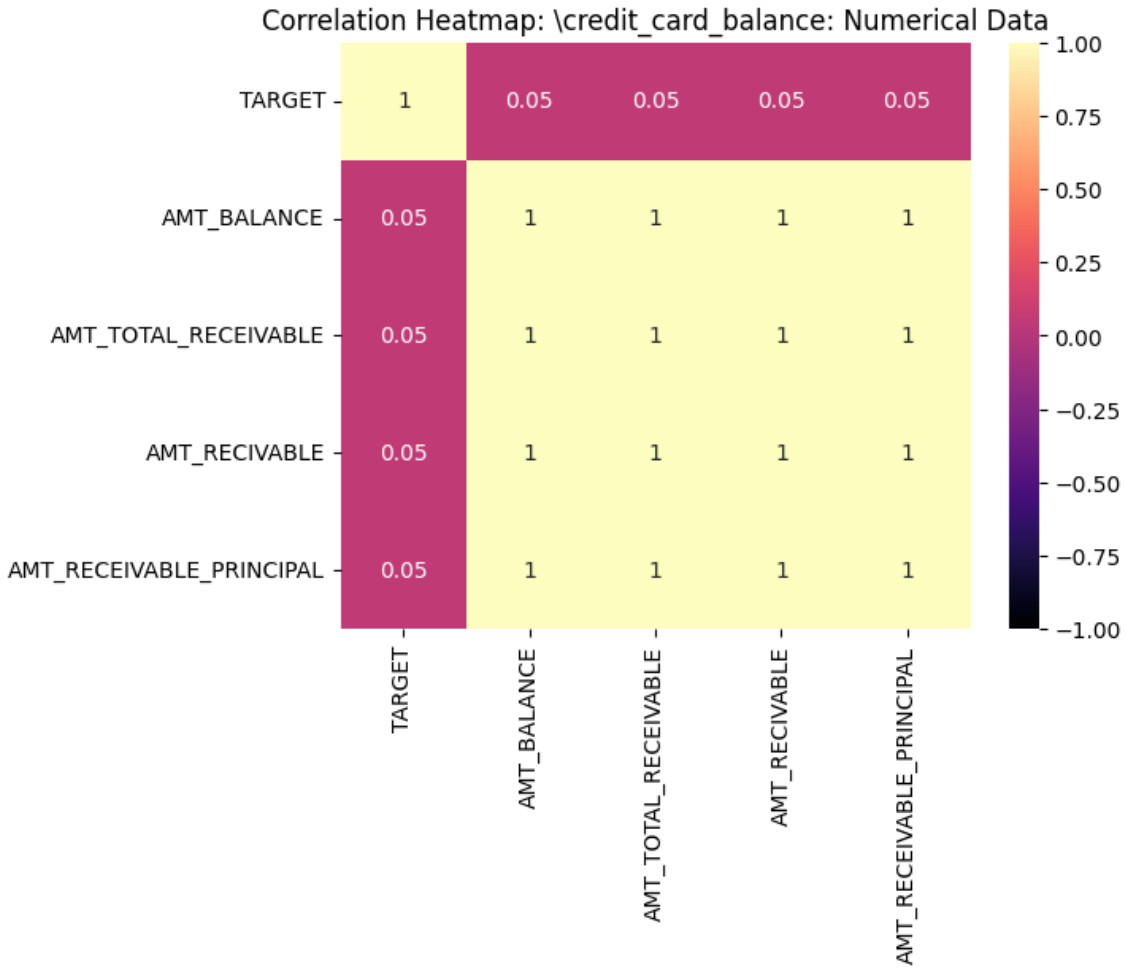
```
<ipython-input-60-c9b13e518754>:4: FutureWarning: The default value of num
eric_only in DataFrame.corr is deprecated. In a future version, it will de
fault to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  pos_credit_corr = pos_credit_target_merge.corr()['TARGET']
```

In [ ]:

```
corr_data = pos_credit_top_feat
corr_data = corr_data.corr()
sns.heatmap(corr_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \credit_card_balance: Numerical Data")
```

Out[61]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \\credit_card_balance: Numerical Dat
a')
```



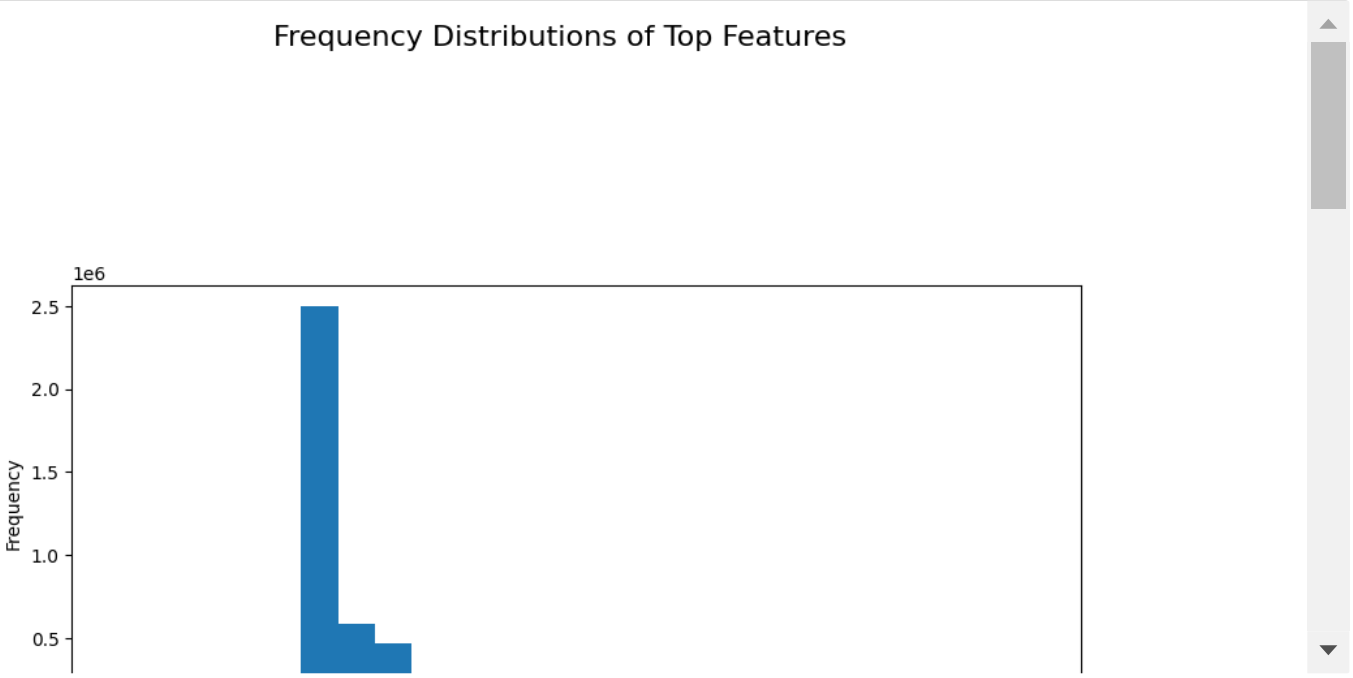
DISCUSSION

It seems that these data must have some artifacts or characteristics about them that throw off the heat map. Possibly they are having an interesting interaction with target variable.

VEDA: Distribution Analysis: credict_card_balance.csv

In [ ]:

```
# Lets look at the distributions of these data
fig, axs = plt.subplots(nrows=n, figsize=(10,20))
fig.suptitle('Frequency Distributions of Top Features', fontsize=16)
for i, feature in enumerate(pos_credit_top_feat_list):
  axs[i - 1].hist(pos_credit_top_feat[feature], bins=25)
  axs[i - 1].set_xlabel(feature)
  axs[i - 1].set_ylabel("Frequency")

plt.show()
```



DISCUSSION This exaplains some of the artifacts that we saw on the heat map. It would seem that this data is normalized, having a sqewed distribution towards the mode at 0.

VEDA: Missing Data Analysis: credict_card_balance.csv

In [ ]:

```
# Numerical Analysis

# CITATION: Parts of code taken from HCDR_baseline_submission_phase2 starter code

missing_percentage = (df_credit_card_bal.isnull().sum() / df_credit_card_bal.isnull(
missing_count = df_credit_card_bal.isna().sum().sort_values(ascending = False)
missing_app_table = pd.concat([missing_percentage, missing_count], axis=1, keys=["Mi
missing_app_table.head(20)
```

Out[63]:

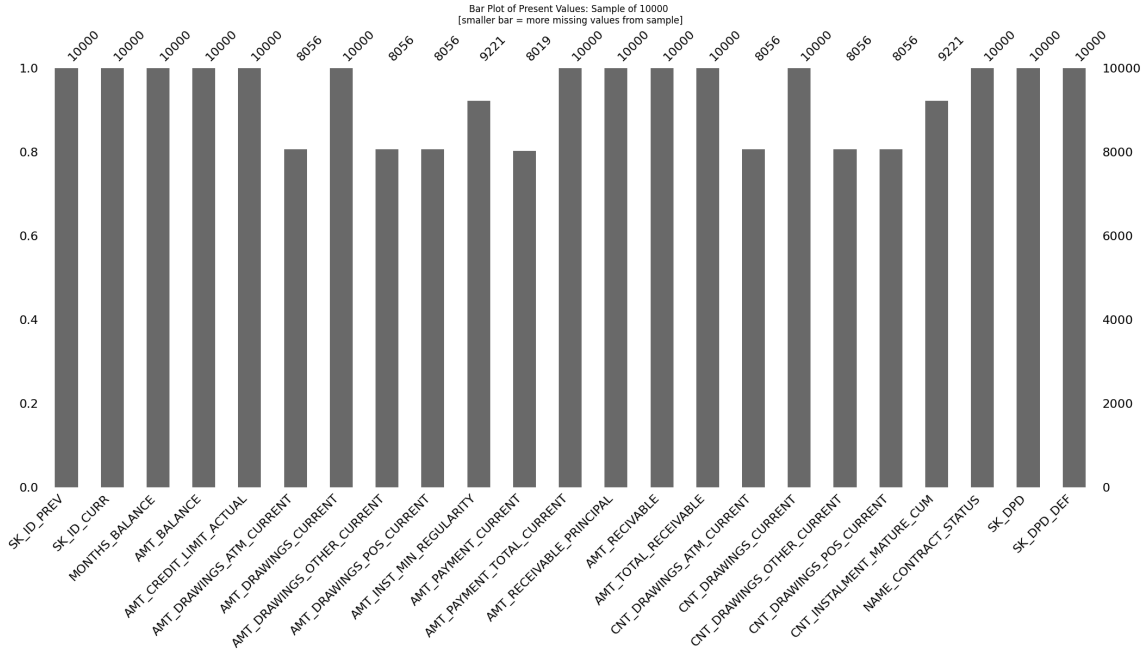
	Missing (%)	Missing (Count)
AMT_PAYMENT_CURRENT	20.00	767988
AMT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_DRAWINGS_POS_CURRENT	19.52	749816
AMT_DRAWINGS_OTHER_CURRENT	19.52	749816
AMT_DRAWINGS_POS_CURRENT	19.52	749816
CNT_DRAWINGS_OTHER_CURRENT	19.52	749816
CNT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_INSTALMENT_MATURE_CUM	7.95	305236
AMT_INST_MIN_REGULARITY	7.95	305236
SK_ID_PREV	0.00	0
AMT_TOTAL_RECEIVABLE	0.00	0
SK_DPD	0.00	0
NAME_CONTRACT_STATUS	0.00	0
CNT_DRAWINGS_CURRENT	0.00	0
AMT_PAYMENT_TOTAL_CURRENT	0.00	0
AMT_RECIVABLE	0.00	0
AMT_RECEIVABLE_PRINCIPAL	0.00	0
SK_ID_CURR	0.00	0
AMT_DRAWINGS_CURRENT	0.00	0
AMT_CREDIT_LIMIT_ACTUAL	0.00	0

In [ ]:

```
# Visualization of these Misssing Values
fig, ax = plt.subplots(1,1, sharex=False, figsize=(20,10))
ax = msno.bar(df_credit_card_bal.sample(10000))
ax.set_title("Bar Plot of Present Values: Sample of 10000 \n[smaller bar = more miss
```

Out[64]:

```
Text(0.5, 1.0, 'Bar Plot of Present Values: Sample of 10000 \n[smaller bar
= more missing values from sample]')
```



DISCUSSION

From these visualizations and numerical based analysis, we can see that there is a high concentration of missing values around the AMT_____CURRENT Features. This is definitly and insight into the context of the data that may be helpful.

VEDA: Input Feature Visualiaztion (previous_application.csv)

VEDA: Correlation Analysis: previous_application.csv

In [ ]:

```
import pandas as pd

pre_app_target_merge = pd.merge(df_pre_app, df_app_train[['SK_ID_CURR', 'TARGET']],
pre_app_corr = pre_app_target_merge.corr()['TARGET']
pre_app_corr_sorted = pre_app_corr.abs().sort_values(ascending=False)

## Show the top correlated
pre_app_corr_sorted.head(10)

## select the top 5 correlated features including the target
n=4
pre_app_top_feat_list = pre_app_corr_sorted[0:n+1].index.tolist()

## Lets put these features into a dataframe with thier original values with the targ
pre_app_top_feat = pre_app_target_merge[pre_app_top_feat_list]
```

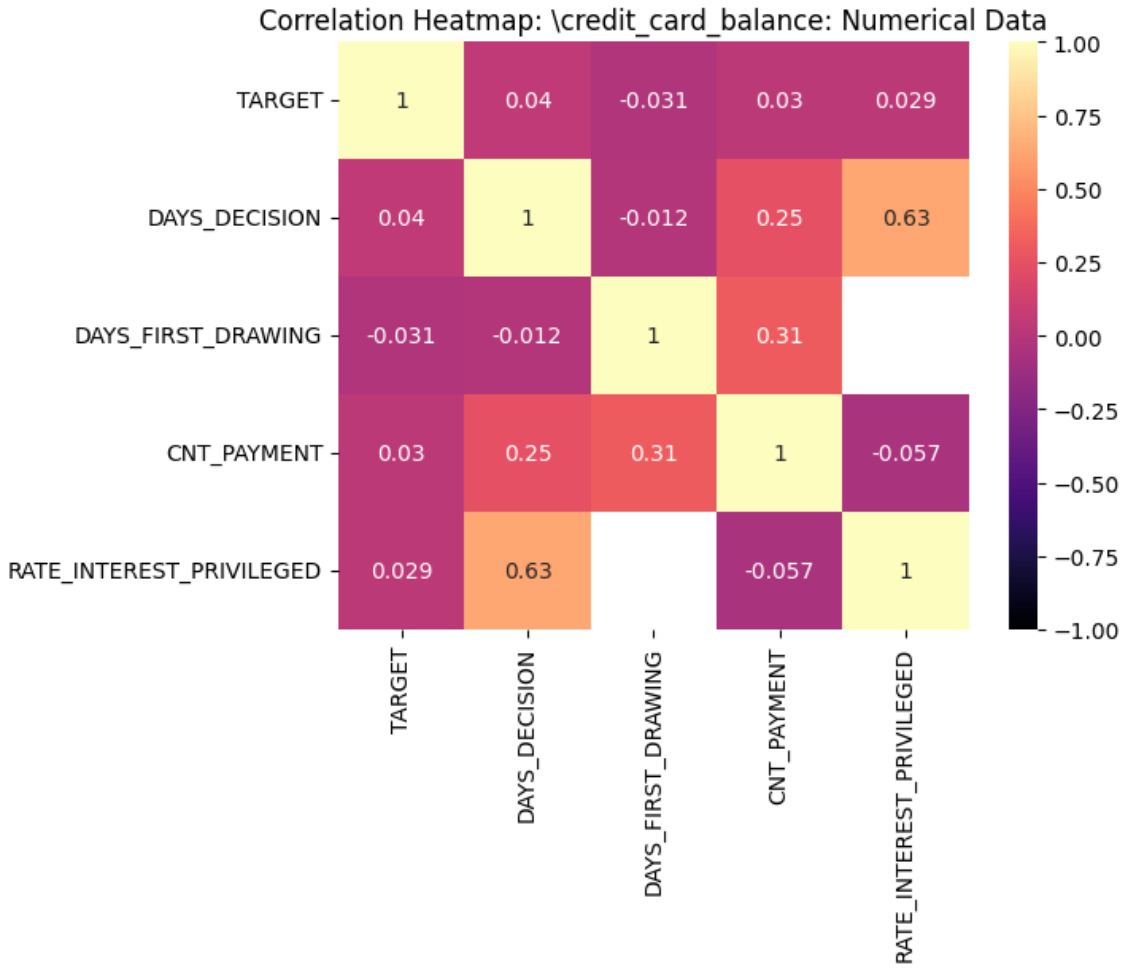
```
<ipython-input-65-beabb716ec20>:4: FutureWarning: The default value of num
eric_only in DataFrame.corr is deprecated. In a future version, it will de
fault to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  pre_app_corr = pre_app_target_merge.corr()['TARGET']
```

In [ ]:

```
corr_data = pre_app_top_feat
corr_data = corr_data.corr()
sns.heatmap(corr_data, cmap="magma", annot=True, vmin= -1.0, vmax=1.0)
plt.title("Correlation Heatmap: \credit_card_balance: Numerical Data")
```

Out[66]:

```
Text(0.5, 1.0, 'Correlation Heatmap: \\credit_card_balance: Numerical Dat
a')
```



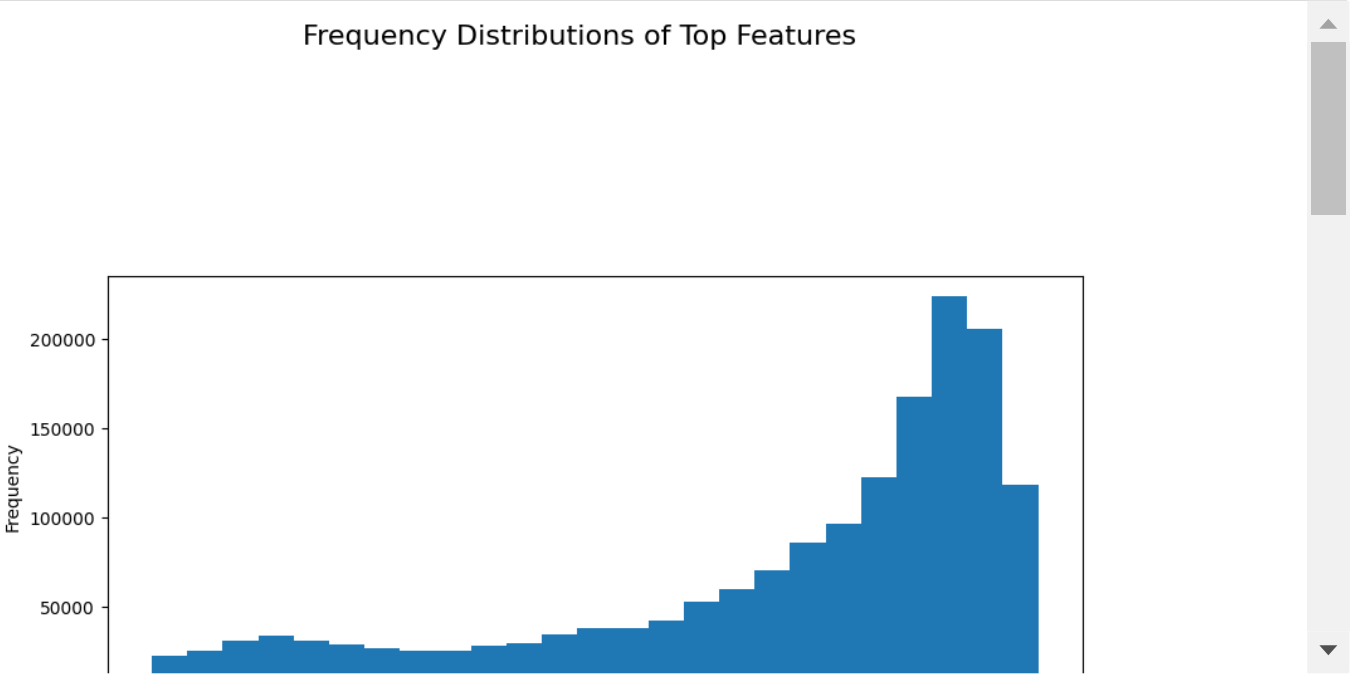
DISCUSSION

From this heat map of the correlations we can see that the highest correlation is between the DAYS_DECISION and DAYS_FIRST_DRAWING. Overall this data set seems to be more consistently positivly correlated with the target value being equal to 1 than most of the others.

VEDA: Distribution Analysis: previous_application.csv

In [ ]:

```
# Lets look at the distributions of these data
fig, axs = plt.subplots(nrows=n, figsize=(10,20))
fig.suptitle('Frequency Distributions of Top Features', fontsize=16)
for i, feature in enumerate(pre_app_top_feat_list):
  axs[i - 1].hist(pre_app_top_feat[feature], bins=25)
  axs[i - 1].set_xlabel(feature)
  axs[i - 1].set_ylabel("Frequency")

plt.show()
```



DISCUSSION

We can see from the distributions that there seems to be an interesting distribution with DAYS_DECISION and CNT_PAYMENT, however the other features seem to be categorical in nature.

VEDA: Missing Data Analysis: previous_application.csv

In [ ]:

```
# Numerical Analysis

# CITATION: Parts of code taken from HCDR_baseline_submission_phase2 starter code

missing_percentage = (df_pre_app.isnull().sum() / df_pre_app.isnull().count() * 100)
missing_count = df_pre_app.isna().sum().sort_values(ascending = False)
missing_app_table = pd.concat([missing_percentage, missing_count], axis=1, keys=["Mi
missing_app_table.head(20)
```

Out[68]:

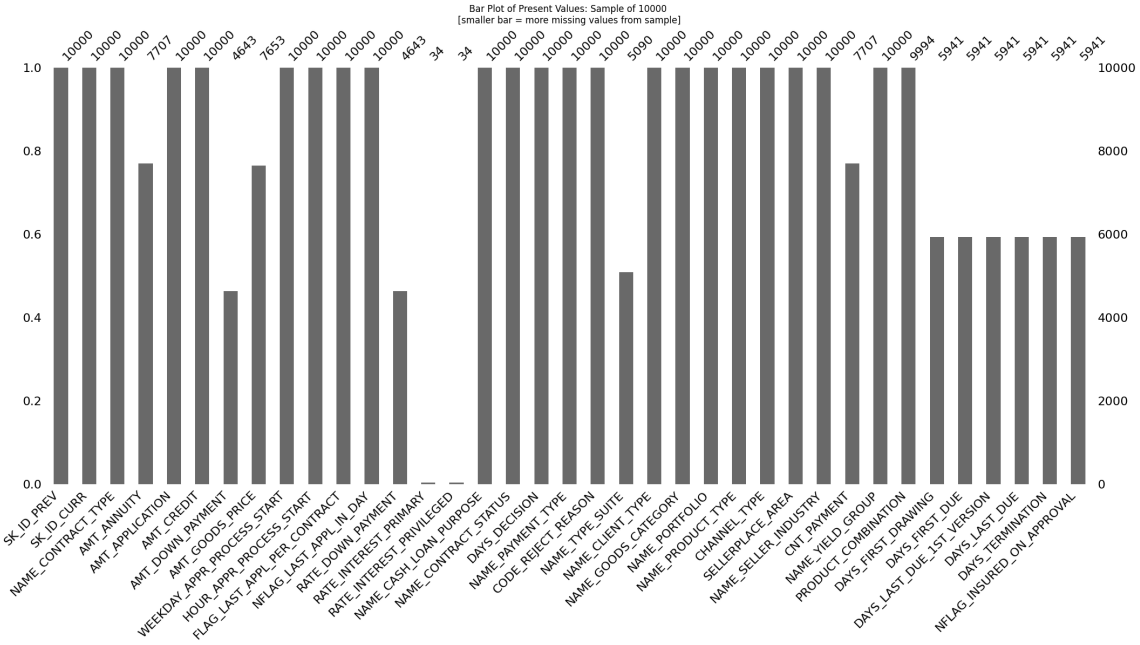
	Missing (%)	Missing (Count)
RATE_INTEREST_PRIVILEGED	99.64	1664263
RATE_INTEREST_PRIMARY	99.64	1664263
AMT_DOWN_PAYMENT	53.64	895844
RATE_DOWN_PAYMENT	53.64	895844
NAME_TYPE_SUITE	49.12	820405
NFLAG_INSURED_ON_APPROVAL	40.30	673065
DAYS_TERMINATION	40.30	673065
DAYS_LAST_DUE	40.30	673065
DAYS_LAST_DUE_1ST_VERSION	40.30	673065
DAYS_FIRST_DUE	40.30	673065
DAYS_FIRST_DRAWING	40.30	673065
AMT_GOODS_PRICE	23.08	385515
AMT_ANNUITY	22.29	372235
CNT_PAYMENT	22.29	372230
PRODUCT_COMBINATION	0.02	346
AMT_CREDIT	0.00	1
NAME_YIELD_GROUP	0.00	0
NAME_PORTFOLIO	0.00	0
NAME_SELLER_INDUSTRY	0.00	0
SELLERPLACE_AREA	0.00	0

In [ ]:

```
# Visualization of these Misssing Values
fig, ax = plt.subplots(1,1, sharex=False, figsize=(20,10))
ax = msno.bar(df_pre_app.sample(10000))
ax.set_title("Bar Plot of Present Values: Sample of 10000 \n[smaller bar = more miss
```

Out[69]:

```
Text(0.5, 1.0, 'Bar Plot of Present Values: Sample of 10000 \n[smaller bar
= more missing values from sample]')
```



DISCUSSION

From these figures it is quite obvious that both RATE_INTEREST_PRIMAR and RATE_INTEREST_PRIVLEGED are outliers in the amoun of data that is missing. This could aid in our feature selection later in the project.

Phase 2: Inital Baseline Modeling

Phase 2: Preprocessing

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:

```
from sklearn.base import BaseEstimator, TransformerMixin
# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [ ]:

```
# Establish X and y
y = df_app_train['TARGET'].copy()
X = df_app_train.copy().drop(["TARGET"],axis=1)

# Split X & y into train & test sets
# Subsequently split train into train & validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_sta
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.
X_kaggle_test = df_app_test

# Identify the numeric features we wish to consider.
num_attribs = X.select_dtypes(include = ['int64','float64']).columns

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler()),
    ])
# Identify the categorical features we wish to consider.
cat_attribs =  X.select_dtypes(include = ['object']).columns

# Notice handle_unknown="ignore" in OHE which ignore values from the validation/test
# do NOT occur in the training set
cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        #('imputer', SimpleImputer(strategy='most_frequent')),
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

data_prep_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])

```

Modeling Pipelines

Modeling Pipelines : Loss Functions

- L1 Loss (Mean Absolute Error):

$$L_{1}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \left| x_{i} - y_{i} \right|$$

where:
x and y are the predicted and actual values, respectively n is the number of samples in the dataset i is the index of each sample in the dataset
$\left| \cdot \right|$ denotes the absolute value
The L1 loss function measures the absolute difference between the predicted values and actual values, and then takes the mean of those differences. It is less sensitive to outliers than the L2 loss function.

- L2 Loss (Mean Squared Error):

$L_{2}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \left(x_{i} - y_{i} \right)^{2}$

where:
x and y are the predicted and actual values, respectively n is the number of samples in the dataset i is the index of each sample in the dataset
This loss function is commonly used in regression problems, where the goal is to predict continuous values. It measures the average of the squared differences between the predicted and actual values. The L2 loss function measures the squared difference between the predicted values and actual values, and then takes the mean of those differences. It is more sensitive to outliers than the L1 loss function.

Modeling Pipelines : Metrics

- F1 Score:

The F1 score is a metric that combines precision and recall. It is useful in situations where both precision and recall are important, such as in binary classification problems where the classes are imbalanced. The F1 score ranges from 0 to 1, where 1 represents perfect precision and recall. It is calculated as:

$$F1 = 2\frac{precision * recall}{precision + recall}$$

- Accuracy Score: The accuracy score is a metric that measures the proportion of correctly classified samples out of all samples. It is useful when the classes in a dataset are balanced. However, it can be misleading in situations where the classes are imbalanced. The accuracy score ranges from 0 to 1, where 1 represents perfect classification. It is calculated as:

$$accuracy = \frac{number\ of\ correctly\ classified\ samples}{total\ number\ of\ samples}$$

- AUC (Area Under the ROC Curve): The AUC is a metric that measures the performance of a binary classification model by calculating the area under the receiver operating characteristic (ROC) curve. The ROC curve is a graph that shows the true positive rate (sensitivity) against the false positive rate (1 - specificity) at different classification thresholds. The AUC ranges from 0 to 1, where 1 represents perfect classification. It is useful in situations where the classes are imbalanced and where the model's output is a probability. The AUC can be calculated using the trapezoidal rule or other numerical integration methods.

In summary, F1 score is useful in situations where both precision and recall are important, accuracy score is useful when the classes in a dataset are balanced, and AUC is useful in situations where the classes are

In [ ]:

```
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC"
                                  ])

```

Phase 2: Logistic Regression

In [ ]:

```
1 X_train.head(10)
```

Out[75]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OV
35339	140933	Cash loans	F	Y	
82049	195150	Cash loans	F	Y	
226288	362102	Cash loans	F	Y	
265467	407465	Cash loans	M	N	
175195	303015	Cash loans	F	Y	
92993	207984	Cash loans	M	N	
7206	108388	Cash loans	F	N	
164322	290486	Cash loans	F	N	
305651	454126	Cash loans	F	N	
137245	259172	Cash loans	F	N	

10 rows × 121 columns

In [ ]:

```
1 y_train.head(10)
```

Out[76]:

```
35339     0
82049     0
226288    0
265467    0
175195    0
92993     1
7206      0
164322    1
305651    0
137245    0
Name: TARGET, dtype: int64
```

In [ ]:

```
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn import metrics

#Create the Logistic Regression Pipeline
lr_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("lr", LogisticRegression())
    ])

#Fit the data to the pipeline
model = lr_pipeline.fit(X_train, y_train)

#Log the results of Accuracy and AUC for Train,Valid and Test datasets
exp_name = f"Baseline_Logistic_Regression"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, model.predict(X_train)),
                accuracy_score(y_valid, model.predict(X_valid)),
                accuracy_score(y_test, model.predict(X_test)),
                roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:4
58: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[77]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.92	0.9162	0.9194	0.7485	0.7475	0.7438

In [ ]:

```
#Create the AUC graph
#metrics.plot_roc_curve(lr_pipeline, X_valid, y_valid)

```

Phase 2: Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

#Create the Random Forest Pipeline
rf_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("rf",RandomForestClassifier(random_state=42))
        ])

#Fit the data to the pipeline
model = rf_pipeline.fit(X_train, y_train)

#Log the results of Accuracy and AUC for Train,Valid and Test datasets
exp_name = f"Baseline_Random_Forest"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, model.predict(X_train)),
                accuracy_score(y_valid, model.predict(X_valid)),
                accuracy_score(y_test, model.predict(X_test)),
                roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

Out[79]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.9200	0.9162	0.9194	0.7485	0.7475	0.7438
1	Baseline_Random_Forest	0.9999	0.9165	0.9194	1.0000	0.7102	0.7109

In [ ]:

```
#Create the AUC graph
#metrics.plot_roc_curve(rf_pipeline, X_valid, y_valid)

```

Phase 2: Decision Tree

In [ ]:

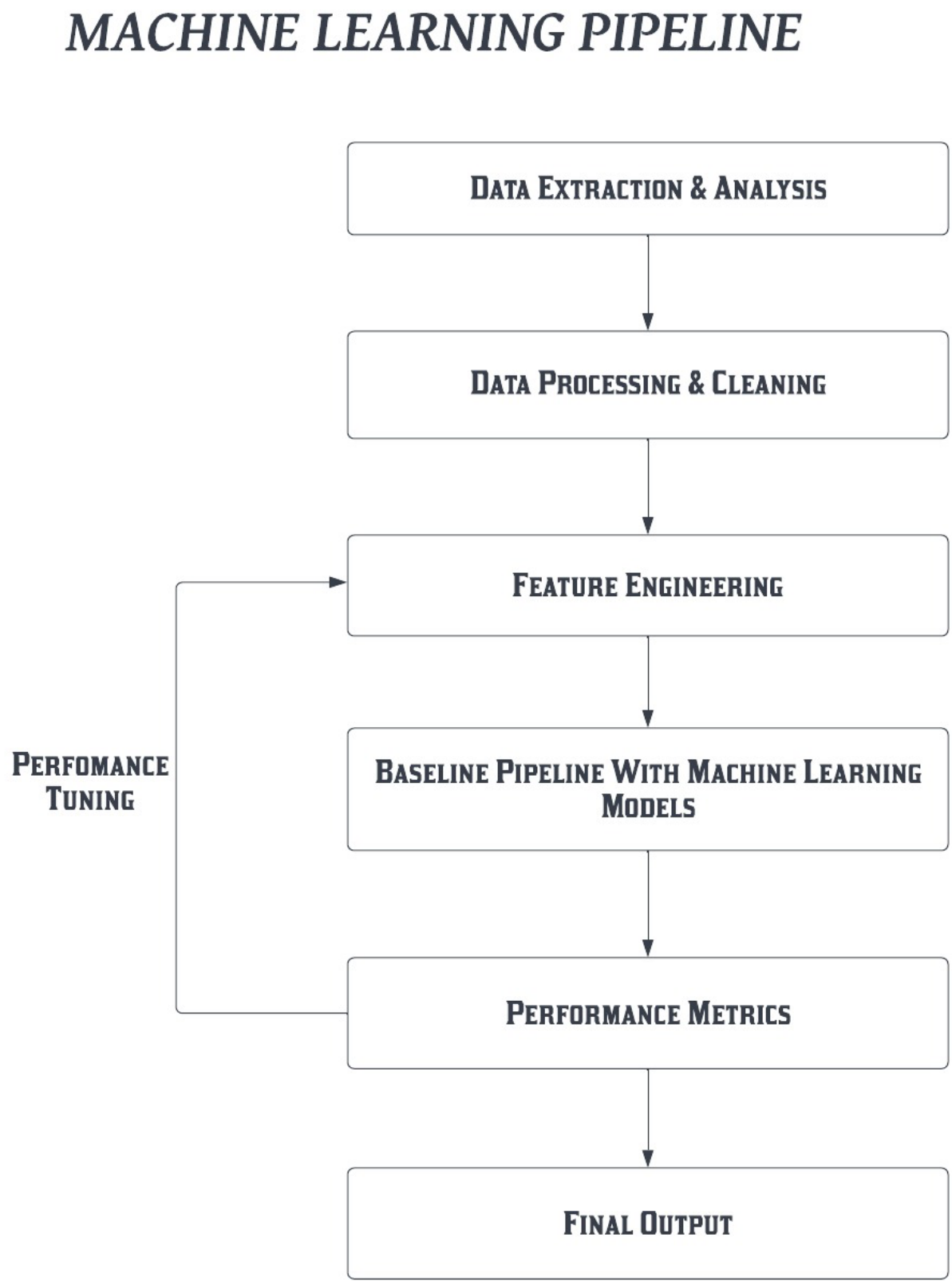
```
from sklearn.tree import DecisionTreeClassifier

#Create the Decision Tree Pipeline
dt_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("dt",DecisionTreeClassifier(random_state=42))
        ])

#Fit the data to the pipeline
model = dt_pipeline.fit(X_train, y_train)

#Log the results of Accuracy and AUC for Train,Valid and Test datasets
exp_name = f"Baseline_Decision_Tree"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, model.predict(X_train)),
                accuracy_score(y_valid, model.predict(X_valid)),
                accuracy_score(y_test, model.predict(X_test)),
                roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

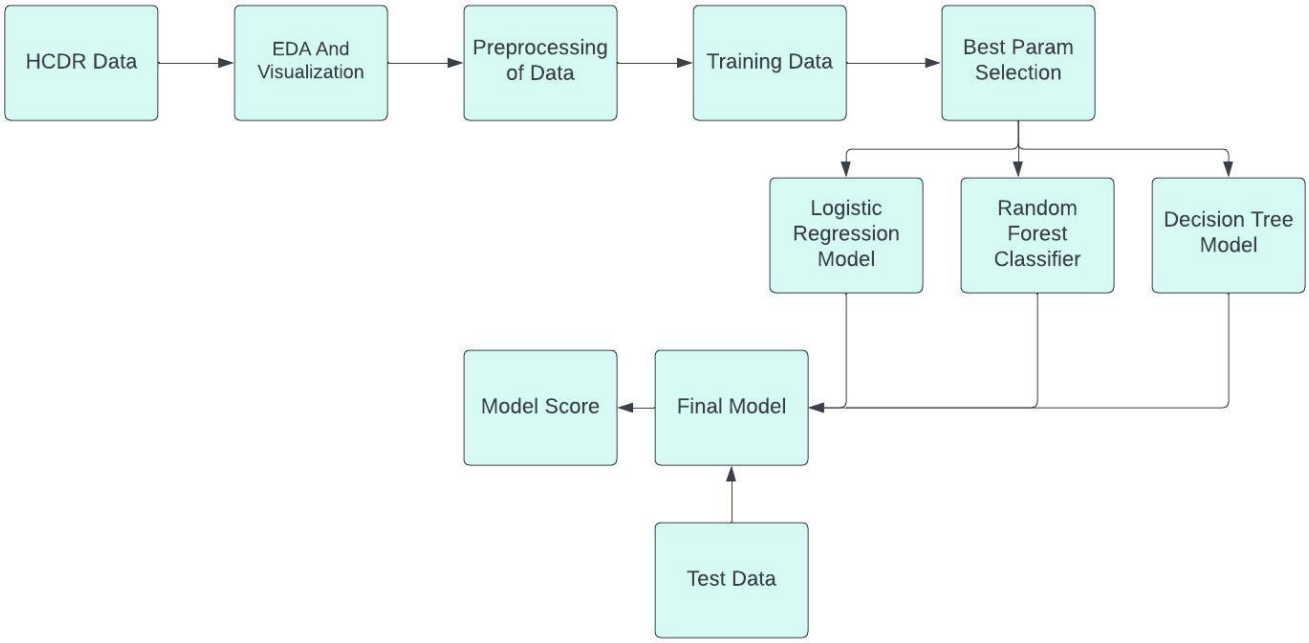
Out[81]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.9200	0.9162	0.9194	0.7485	0.7475	0.7438
1	Baseline_Random_Forest	0.9999	0.9165	0.9194	1.0000	0.7102	0.7109
2	Baseline_Decision_Tree	1.0000	0.8528	0.8529	1.0000	0.5427	0.5367

Phase 2: Basic ML Pipline Outline Diagram



Phase 2: Process Diagram



Phase 2: Process Diagram + Tuning Step

Phase 2: Kaggle Submission

For each SK_ID_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```
SK_ID_CURR,TARGET
100001,0.1
100005,0.9
100013,0.2
etc.
```

In [ ]:

```
model = lr_pipeline.fit(X_train, y_train)
X_kaggle_test = df_app_test.copy()
test_class_scores = model.predict_proba(X_kaggle_test)[:, 1]
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:4
58: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [ ]:

```
test_class_scores[0:10]
```

Out[83]:

```
array([0.06095341, 0.23342843, 0.055663  , 0.02879285, 0.12086613,
       0.03513475, 0.02080647, 0.09970679, 0.01536396, 0.11598527])
```

In [ ]:

```
# Submission dataframe
submit_df = df_app_test[['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

submit_df.head()
```

```
<ipython-input-84-e10dd3e85e77>:3: 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-doc
s/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)
  submit_df['TARGET'] = test_class_scores
```

Out[84]:

	SK_ID_CURR	TARGET
0	100001	0.060953
1	100005	0.233428
2	100013	0.055663
3	100028	0.028793
4	100038	0.120866

In [ ]:

```
submit_df.to_csv("submission.csv",index=False)
```

Phase 2: Kaggle submission via the command line API

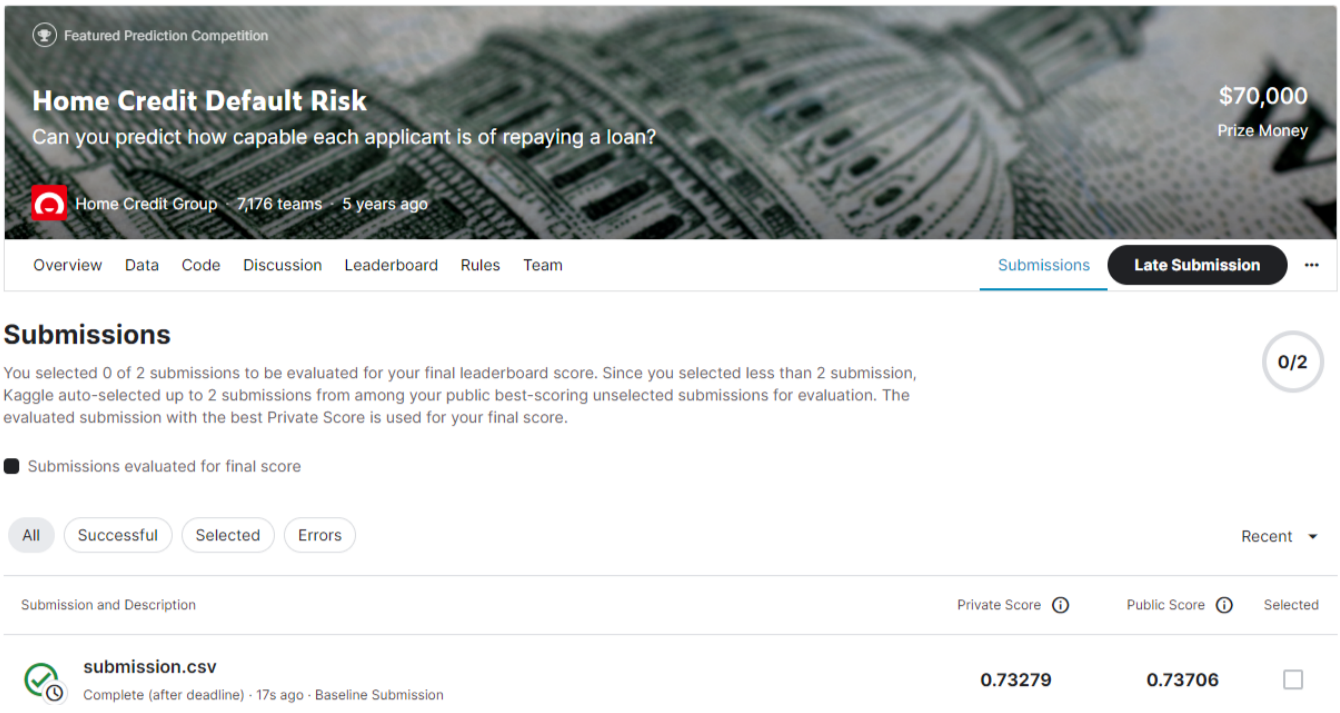
In [ ]:

```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "basel
```

```
100% 1.26M/1.26M [00:00<00:00, 3.68MB/s]
Successfully submitted to Home Credit Default Risk
```

Phase 2: Submission Report

Click on this link (https://www.kaggle.com/c/home-credit-default-risk/submissions?sortBy=date&group=all&page=1)



Phase 2: Hyper Parameter Tuning (Intial Planning)

We have selected the following Hyperparamters with respect to the different Machine Learning Algorithms we will try:

1. Logisitc Regresssion: For our LR model the parameters chosen for hyperparameter tuning are:
   A. C parameter which controls the penalty strength.
   B. Penalty parameter which imposes a penalty to the logistic model for having too many variables. We will try with ['none', l1','l2']
   C. Solver which allows us to see useful difference in performance or convergence with different solvers
2. Random Forest

   For our RF model we have chosen the following hyperparamters:
   A. bootstrap - this parameter means that each tree in the random forest runs on a subset of the observations.
   B. max_depth - maximum number of levels allowed in each decision tree
   C. forest__max_features - number of features in consideration at every split
   D. forest__n_estimators - number of trees in the random forest

3. Decision Tree

   For our DT model we have chosen the following parameters:
   A. criterion - The function to measure the quality of a split.
   B. max_depth - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
   C. min_samples_leaf - The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

Phase 2: Results & Discussion

As per the Project pipeline we first downloaded the data from Kaggle. Then performed EDA on the data to get a better understanding of what all features are present and how they are correlated to the 'Target' variable present in the application_trai datatset. In all we have 9 tables which we had to merge together to get our training and test datasets. We chose 3 Machine Learing models to be run on our given dataset of HCDR. The models are as follows:

1. Logistic Regression
2. Random Forest
3. Decision Tree

We divided the application_train data into 3 subsets of train,valid and test with a random seed of 42 and test size = 0.15. We used 2 metrics : Accuracy and Area Under Curve

Upon running these models using the above mentioned metrics we found out the results of each of the metric for every train,valid and test datasets. We found out that without any Hyperparameter tuning, as a Baseline model, Logistic Regression performed the best with each of the metric. The experiment log table is :

In [ ]:

```
1 expLog
```

Out[87]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.9200	0.9162	0.9194	0.7485	0.7475	0.7438
1	Baseline_Random_Forest	0.9999	0.9165	0.9194	1.0000	0.7102	0.7109
2	Baseline_Decision_Tree	1.0000	0.8528	0.8529	1.0000	0.5427	0.5367

In [ ]:

```
1
```

Phase 2: Conclusion

The aim of the HCDR study is to predict the population's capacity for repayment among those who are economically neglected. This project is important because both the lender and the borrower want accurate estimates. The ML pipelines used by Real-time Home Credit allow them to present their customers with loan offers that have the highest amount and APR because they use EDA to fit the data to the model and generate scores. A user's average, minimum, and maximum balances as well as reported Bureau scores, salary, and other factors are used to generate a credit history, which serves as a gauge of their reliability. The user's timely defaults and repayments can be used to assess repayment habits. Alternative data also includes criteria like location data, social media data, calling/SMS data, etc. In order to complete this project, we would build machine learning pipelines, do exploratory data analysis using the Kaggle datasets, and test many models before deploying one. The estimation of many models was a part of phase 2. When we dug into the data we were able to create a pipline that acuratly predicts the target with an AUC score of 74. This

is significant becuase, both feature selection and data imputation were performed. We chose characteristics and imputed values in the beginning. The values of a few lacking features were filled in. Finally, based on our prior knowledge, we decided which features to incorporate. To find the most effective model, we trained and evaluated a number of them, including Random Forest, Decision tree Model, and Logistic Regression . Out of all the models, the logistic regression model performs the best. We intend to put all models into practice in phase 3 by fine-tuning their individual parameters. In the future we would like to preform hyperparmeter tuning with more compute power, allowing us to accurately merge and estimate the target class with data that we have deemed significant

Phase 3: Feature Engineering, Hyperparameter Tuning, & Improved Model

Phase 3: Feature Engineering

Feature Engineering: General Functions

In [ ]:

```
# One Hot Encoder Implementation for the correlation analysis for categorical featur
def OneHotCorr(df):
  cat_columns = df.select_dtypes(include='object').columns
  df = pd.get_dummies(df, columns = cat_columns, dummy_na = False)
  return df

```

In [ ]:

```
# Correlation analysis for multiple dataframes
def TargetCorr(df_1, df_2):
  df__id = df_1[["SK_ID_CURR", "TARGET"]].copy()
  df__tar = df__id.merge(df_2, how='left', on='SK_ID_CURR')
  df__corr = df__tar.corr()['TARGET'].abs().sort_values(ascending=False)
  return df__corr
```

Feature Engineering: bureau & bureau_balance

Feature Engineering: Secondary Table Merge

In [ ]:

```
bur_merge = df_bureau.merge(df_bureau_bal, how="left", on=["SK_ID_BUREAU"])
bur_merge.head(10)
```

Out[13]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRE
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	
5	215354	5714467	Active	currency 1	-273	
6	215354	5714468	Active	currency 1	-43	
7	162297	5714469	Closed	currency 1	-1896	
8	162297	5714470	Closed	currency 1	-1146	
9	162297	5714471	Active	currency 1	-1146	

Feature Engineering: Feature Creation

In [ ]:

```
# Create Features for the Bureau and Bureau_balance
#-------------------------------------------------

## term of credit granted to the individual with the loan
bur_merge['BUR_END_DAY_RATIO'] = bur_merge['DAYS_CREDIT_ENDDATE'] / bur_merge['DAYS_
bur_merge['BUR_END_DAY_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
bur_merge['BUR_END_DAY_RATIO'] = bur_merge['BUR_END_DAY_RATIO'].fillna(bur_merge['BU

## amount repaid per year
bur_merge['BUR_DEBT_ANNUITY_RATIO'] = bur_merge['AMT_CREDIT_SUM_DEBT'] / bur_merge['
bur_merge['BUR_DEBT_ANNUITY_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
bur_merge['BUR_DEBT_ANNUITY_RATIO'] = bur_merge['BUR_DEBT_ANNUITY_RATIO'].fillna(bur

# debt to limit ratio - responsibility with credit
bur_merge['BUR_DEBT_LIMIT_RATIO'] = bur_merge['AMT_CREDIT_SUM_DEBT'] / bur_merge['AM
bur_merge['BUR_DEBT_LIMIT_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
bur_merge['BUR_DEBT_LIMIT_RATIO'] = bur_merge['BUR_DEBT_LIMIT_RATIO'].fillna(bur_mer

# proportion of the borrower's income that is dedicated to repaying the loan.
bur_merge['BUR_CREDIT_ANNUITY_RATIO'] = bur_merge['AMT_CREDIT_SUM'] / bur_merge['AMT
bur_merge['BUR_CREDIT_ANNUITY_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=Tru
bur_merge['BUR_CREDIT_ANNUITY_RATIO'] = bur_merge['BUR_CREDIT_ANNUITY_RATIO'].fillna

# total debt for each loan reported in the bureau data.
bur_merge['BUR_CREDIT_DEBT_RATIO'] = bur_merge['AMT_CREDIT_SUM'] / bur_merge['AMT_CR
bur_merge['BUR_CREDIT_DEBT_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
bur_merge['BUR_CREDIT_DEBT_RATIO'] = bur_merge['BUR_CREDIT_DEBT_RATIO'].fillna(bur_m

# difference between credit record date and update
bur_merge['BUR_DAY_UPDATE_DIFF'] = bur_merge['DAYS_CREDIT'] - bur_merge['DAYS_CREDIT
```

In [ ]:

```
# Check that all columns have been added to the secondary table
bur_merge.columns
bur_merge.head(10)
```

Out[15]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRE
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	
5	215354	5714467	Active	currency 1	-273	
6	215354	5714468	Active	currency 1	-43	
7	162297	5714469	Closed	currency 1	-1896	
8	162297	5714470	Closed	currency 1	-1146	
9	162297	5714471	Active	currency 1	-1146	

10 rows × 25 columns

Feature Engineering: Correlation Analysis

In [ ]:

```
# Prepare the categorical features of bur_merge
bur_merge_ohe = OneHotCorr(bur_merge)
```

In [ ]:

```
# Show the correlations to the target
bur_merge_corr = TargetCorr(df_app_train, bur_merge_ohe)
```

In [ ]:

```
1 print(bur_merge_corr)
```

```
TARGET                                                  1.000000
DAYS_CREDIT                                             0.048601
DAYS_CREDIT_UPDATE                                      0.033850
DAYS_ENDDATE_FACT                                       0.031851
CREDIT_ACTIVE_Closed                                    0.030775
CREDIT_ACTIVE_Active                                    0.029832
MONTHS_BALANCE                                          0.027773
BUR_DAY_UPDATE_DIFF                                     0.021732
STATUS_C                                                0.020756
CREDIT_TYPE_Credit card                                 0.017873
BUR_END_DAY_RATIO                                       0.016490
CREDIT_TYPE_Microloan                                   0.016131
DAYS_CREDIT_ENDDATE                                     0.015737
STATUS_0                                                0.015521
STATUS_1                                                0.012811
CREDIT_TYPE_Consumer credit                             0.012619
CREDIT_TYPE_Car loan                                    0.012312
AMT_CREDIT_SUM                                          0.010644
CREDIT_TYPE_Mortgage                                    0.008863
STATUS_5                                                0.008822
SK_ID_BUREAU                                            0.008195
CREDIT_TYPE_Loan for the purchase of equipment          0.007505
CREDIT_ACTIVE_Sold                                      0.006963
AMT_CREDIT_SUM_LIMIT                                    0.006044
AMT_CREDIT_SUM_OVERDUE                                  0.004549
STATUS_2                                                0.004237
STATUS_3                                                0.003339
CREDIT_CURRENCY_currency 2                              0.003322
CREDIT_CURRENCY_currency 1                              0.003235
STATUS_4                                                0.003102
CREDIT_TYPE_Another type of loan                        0.002121
AMT_CREDIT_SUM_DEBT                                     0.001939
CREDIT_TYPE_Unknown type of loan                        0.001648
CREDIT_TYPE_Loan for working capital replenishment      0.001273
BUR_CREDIT_ANNUITY_RATIO                                0.001036
CNT_CREDIT_PROLONG                                      0.000898
CREDIT_TYPE_Real estate loan                            0.000810
CREDIT_TYPE_Loan for business development               0.000765
CREDIT_CURRENCY_currency 4                              0.000749
CREDIT_TYPE_Cash loan (non-earmarked)                   0.000641
SK_ID_CURR                                              0.000613
CREDIT_TYPE_Loan for purchase of shares (margin lending)  0.000427
AMT_ANNUITY                                             0.000388
CREDIT_TYPE_Mobile operator loan                        0.000353
BUR_CREDIT_DEBT_RATIO                                   0.000260
STATUS_X                                                0.000259
BUR_DEBT_LIMIT_RATIO                                    0.000145
CREDIT_ACTIVE_Bad debt                                  0.000141
CREDIT_CURRENCY_currency 3                              0.000133
BUR_DEBT_ANNUITY_RATIO                                  0.000132
AMT_CREDIT_MAX_OVERDUE                                  0.000104
CREDIT_DAY_OVERDUE                                      0.000096
CREDIT_TYPE_Interbank credit                            0.000069
Name: TARGET, dtype: float64
```

Feature Engineering: Feature Selection

In [ ]:

```
# Remove the ID
bur_merge_corr = bur_merge_corr[1:].copy()
```

In [ ]:

```
# Select all of the features that have greater than or equal to 2% correlation to th
bur_select = bur_merge_ohe[list(bur_merge_corr[bur_merge_corr>=0.02].index) + ['SK_I
```

In [ ]:

```
bur_select.shape
```

Out[21]:

```
(25121815, 10)
```

In [ ]:

```
bur_select.head(10)
```

Out[22]:

	DAYS_CREDIT	DAYS_CREDIT_UPDATE	DAYS_ENDDATE_FACT	CREDIT_ACTIVE_Closed	C
0	-497	-131	-153.0	1	
1	-208	-20	NaN	0	
2	-203	-16	NaN	0	
3	-203	-16	NaN	0	
4	-629	-21	NaN	0	
5	-273	-31	NaN	0	
6	-43	-22	NaN	0	
7	-1896	-1710	-1710.0	1	
8	-1146	-840	-840.0	1	
9	-1146	-690	NaN	0	

Feature Engineering: Feature Aggregation

In [ ]:

```
bur_final = bur_select.groupby(["SK_ID_CURR"], as_index = False).agg("mean")
```

In [ ]:

```
1 bur_final.head(10)
```

Out[24]:

	SK_ID_CURR	DAYS_CREDIT	DAYS_CREDIT_UPDATE	DAYS_ENDDATE_FACT	CREDIT_AC
0	100001	-1009.284884	-127.651163	-908.421429	
1	100002	-996.781818	-631.963636	-808.400000	
2	100003	-1400.750000	-816.000000	-1097.333333	
3	100004	-867.000000	-532.000000	-532.500000	
4	100005	-272.380952	-81.952381	-123.000000	
5	100007	-1149.000000	-783.000000	-783.000000	
6	100008	-757.333333	-611.000000	-909.000000	
7	100009	-1271.500000	-851.611111	-1108.500000	
8	100010	-1939.500000	-578.000000	-1138.000000	
9	100011	-1773.000000	-1454.750000	-1463.250000	

Feature Engineering: POS_CASH_balance

Feature Engineering: Feature Creation

In [ ]:

```
1 pos_cash_bal = df_pos_cash_bal.copy()
```

In [ ]:

```
1  # Create Features for the POS_CASH_balance
2  #--------------------------------------------------
3
4  # ratio of installments paid to future installments remaining for each loan.
5  pos_cash_bal['POS_INSTALL_FUTURE_RATIO'] = pos_cash_bal["CNT_INSTALMENT"] / pos_cash
6  pos_cash_bal['POS_INSTALL_FUTURE_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=
7  pos_cash_bal['POS_INSTALL_FUTURE_RATIO'] = pos_cash_bal['POS_INSTALL_FUTURE_RATIO'].
8
9  # number of days that a customer was overdue on a payment, considering both the regu
10 # delay ('SK_DPD') and the more severe delay ('SK_DPD_DEF')
11 pos_cash_bal['PYAMENT_BEHAVIOR'] = pos_cash_bal['SK_DPD'] - pos_cash_bal['SK_DPD_DEF
```

In [ ]:

```
1 pos_cash_bal.columns
```

Out[27]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'CNT_INSTALMENT',
       'CNT_INSTALMENT_FUTURE', 'NAME_CONTRACT_STATUS', 'SK_DPD', 'SK_DPD_
DEF',
       'POS_INSTALL_FUTURE_RATIO', 'PYAMENT_BEHAVIOR'],
      dtype='object')
```

In [ ]:

```
1 pos_cash_bal.head(10)
```

Out[28]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FU
0	1803195	182943	-31	48.0	
1	1715348	367990	-33	36.0	
2	1784872	397406	-32	12.0	
3	1903291	269225	-35	48.0	
4	2341044	334279	-35	36.0	
5	2207092	342166	-32	12.0	
6	1110516	204376	-38	48.0	
7	1387235	153211	-35	36.0	
8	1220500	112740	-31	12.0	
9	2371489	274851	-32	24.0	

Feature Engineering: Correlation Analysis

In [ ]:

```
1 # Prepare the categorical features of pos_cash_bal
2 pos_cash_ohe = OneHotCorr(pos_cash_bal)
```

In [ ]:

```
1 pos_cash_corr = TargetCorr(df_app_train, pos_cash_ohe)
```

In [ ]:

```
print(pos_cash_corr)
```

```
TARGET                                        1.000000
CNT_INSTALMENT_FUTURE                         0.021972
MONTHS_BALANCE                                0.020147
CNT_INSTALMENT                                0.018506
SK_DPD                                        0.009866
SK_DPD_DEF                                    0.008594
PYAMENT_BEHAVIOR                              0.008126
POS_INSTALL_FUTURE_RATIO                      0.008017
NAME_CONTRACT_STATUS_Amortized debt           0.006732
NAME_CONTRACT_STATUS_Demand                   0.006062
NAME_CONTRACT_STATUS_Returned to the store    0.002596
SK_ID_CURR                                    0.002244
NAME_CONTRACT_STATUS_Active                   0.001915
NAME_CONTRACT_STATUS_Signed                   0.001669
NAME_CONTRACT_STATUS_Canceled                 0.000423
NAME_CONTRACT_STATUS_Completed                0.000412
NAME_CONTRACT_STATUS_Approved                 0.000170
NAME_CONTRACT_STATUS_XNA                      0.000136
SK_ID_PREV                                    0.000056
Name: TARGET, dtype: float64
```

Feature Engineering: Feature Selection

In [ ]:

```
pos_cash_corr = pos_cash_corr[1:].copy()
pos_cash_bal_select = pos_cash_bal[list(pos_cash_corr[pos_cash_corr >= 0.02].index)
```

In [ ]:

```
pos_cash_bal_select.head(10)
```

Out[33]:

	CNT_INSTALMENT_FUTURE	MONTHS_BALANCE	SK_ID_CURR	SK_ID_PREV
0	45.0	-31	182943	1803195
1	35.0	-33	367990	1715348
2	9.0	-32	397406	1784872
3	42.0	-35	269225	1903291
4	35.0	-35	334279	2341044
5	12.0	-32	342166	2207092
6	43.0	-38	204376	1110516
7	36.0	-35	153211	1387235
8	12.0	-31	112740	1220500
9	16.0	-32	274851	2371489

Feature Engineering: Feature Aggregation

In [ ]:

```
pos_cash_final = pos_cash_bal_select.groupby(["SK_ID_CURR"], as_index = False).agg("
```

In [ ]:

```
pos_cash_final.head(10)
```

Out[35]:

	SK_ID_CURR	CNT_INSTALMENT_FUTURE	MONTHS_BALANCE	SK_ID_PREV
0	100001	1.444444	-72.555556	1.584045e+06
1	100002	15.000000	-10.000000	1.038818e+06
2	100003	5.785714	-43.785714	2.297665e+06
3	100004	2.250000	-25.500000	1.564014e+06
4	100005	7.200000	-20.000000	2.495675e+06
5	100006	8.650000	-9.619048	2.215853e+06
6	100007	8.969697	-33.636364	2.041993e+06
7	100008	4.108434	-43.662651	1.955675e+06
8	100009	3.781250	-33.062500	1.636626e+06
9	100010	5.000000	-30.000000	2.349489e+06

Feature Engineering: credit_card_balance

Feature Engineering: Feature Creation

In [ ]:

```
credit_card_bal = df_credit_card_bal.copy()
```

In [ ]:

```
# Create Features for the credit_card_bal
#-------------------------------------------------
credit_card_bal['CRD_TOTAL_AMT_WITHDRAWN'] = credit_card_bal['CNT_DRAWINGS_ATM_CURRE


credit_card_bal['CRD_COUNT_WITHDRAWLS'] = credit_card_bal['CNT_DRAWINGS_ATM_CURRENT'


credit_card_bal['CRD_AMT_PAID_MONTH_RATIO'] = credit_card_bal['CRD_TOTAL_AMT_WITHDRA
credit_card_bal['CRD_AMT_PAID_MONTH_RATIO'].replace([np.inf, -np.inf], np.nan, inpla
credit_card_bal['CRD_AMT_PAID_MONTH_RATIO'] = credit_card_bal['CRD_AMT_PAID_MONTH_RA

credit_card_bal['NO_INSTALLMENTS_MADE_RATIO'] = credit_card_bal['CRD_COUNT_WITHDRAWL
credit_card_bal['NO_INSTALLMENTS_MADE_RATIO'].replace([np.inf, -np.inf], np.nan, inp
credit_card_bal['NO_INSTALLMENTS_MADE_RATIO'] = credit_card_bal['NO_INSTALLMENTS_MAD

credit_card_bal['RATIO_CREDIT_BALANCE_RATIO'] = credit_card_bal['AMT_BALANCE'] / cre
credit_card_bal['RATIO_CREDIT_BALANCE_RATIO'].replace([np.inf, -np.inf], np.nan, inp
credit_card_bal['RATIO_CREDIT_BALANCE_RATIO'] = credit_card_bal['RATIO_CREDIT_BALANC
```

In [ ]:

```
credit_card_bal.columns
```

Out[38]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'AMT_BALANCE',
       'AMT_CREDIT_LIMIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT',
       'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT',
       'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY',
       'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT',
       'AMT_RECEIVABLE_PRINCIPAL', 'AMT_RECIVABLE', 'AMT_TOTAL_RECEIVABL
E',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
       'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',
       'CNT_INSTALMENT_MATURE_CUM', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF', 'CRD_TOTAL_AMT_WITHDRAWN', 'CRD_COUNT_WITHDRAWLS',
       'CRD_AMT_PAID_MONTH_RATIO', 'NO_INSTALLMENTS_MADE_RATIO',
       'RATIO_CREDIT_BALANCE_RATIO'],
      dtype='object')
```

Feature Engineering: Correlation Analysis

In [ ]:

```
# Prepare the categorical features of pos_cash_bal
credit_card_ohe = OneHotCorr(credit_card_bal)
```

In [ ]:

```
credit_card_corr = TargetCorr(df_app_train, credit_card_ohe)
```

In [ ]:

```
print(credit_card_corr)
```

```
TARGET                           1.000000
AMT_BALANCE                      0.050098
AMT_TOTAL_RECEIVABLE             0.049839
AMT_RECIVABLE                    0.049803
AMT_RECEIVABLE_PRINCIPAL         0.049692
RATIO_CREDIT_BALANCE_RATIO       0.046731
AMT_INST_MIN_REGULARITY          0.039798
CRD_TOTAL_AMT_WITHDRAWN          0.039247
CRD_COUNT_WITHDRAWLS             0.039247
CNT_DRAWINGS_ATM_CURRENT         0.038437
CNT_DRAWINGS_CURRENT             0.037793
MONTHS_BALANCE                   0.035695
CNT_DRAWINGS_POS_CURRENT         0.029536
AMT_DRAWINGS_ATM_CURRENT         0.024700
CNT_INSTALMENT_MATURE_CUM        0.023684
NO_INSTALLMENTS_MADE_RATIO       0.023527
AMT_DRAWINGS_CURRENT             0.022378
AMT_CREDIT_LIMIT_ACTUAL          0.013823
AMT_PAYMENT_CURRENT              0.012929
AMT_PAYMENT_TOTAL_CURRENT        0.012302
SK_DPD_DEF                       0.010538
NAME_CONTRACT_STATUS_Demand      0.008044
AMT_DRAWINGS_POS_CURRENT         0.005084
NAME_CONTRACT_STATUS_Completed   0.004917
SK_ID_CURR                       0.004412
NAME_CONTRACT_STATUS_Active      0.004362
AMT_DRAWINGS_OTHER_CURRENT       0.003843
CNT_DRAWINGS_OTHER_CURRENT       0.003044
SK_ID_PREV                       0.002571
SK_DPD                           0.001684
NAME_CONTRACT_STATUS_Sent proposal  0.001491
NAME_CONTRACT_STATUS_Signed      0.001026
CRD_AMT_PAID_MONTH_RATIO         0.000679
NAME_CONTRACT_STATUS_Refused     0.000580
NAME_CONTRACT_STATUS_Approved    0.000345
Name: TARGET, dtype: float64
```

Feature Engineering: Feature Selection

In [ ]:

```
credit_card_corr = credit_card_corr[1:].copy()
credit_card_bal = credit_card_bal[list(credit_card_corr[credit_card_corr >= 0.02].ir
```

In [ ]:

```
1 credit_card_bal.head(10)
```

Out[43]:

	AMT_BALANCE	AMT_TOTAL_RECEIVABLE	AMT_RECIVABLE	AMT_RECEIVABLE_PRINCIPA
0	56.970	0.000	0.000	0.0(
1	63975.555	64875.555	64875.555	60175.0{
2	31815.225	31460.085	31460.085	26926.4{
3	236572.110	233048.970	233048.970	224949.2{
4	453919.455	453919.455	453919.455	443044.3{
5	82903.815	82773.315	82773.315	80519.0{
6	353451.645	351881.145	351881.145	345433.8{
7	47962.125	47962.125	47962.125	44735.3{
8	291543.075	286831.575	286831.575	285376.4{
9	201261.195	197224.695	197224.695	192793.2{

Feature Engineering: Feature Aggregation

In [ ]:

```
1 credit_card_final = credit_card_bal.groupby(["SK_ID_CURR"],as_index = False).agg("me
```

In [ ]:

```
1 credit_card_final.head(10)
```

Out[45]:

	SK_ID_CURR	AMT_BALANCE	AMT_TOTAL_RECEIVABLE	AMT_RECIVABLE	AMT_RECEIV{
0	100006	0.000000	0.000000	0.000000	
1	100011	54482.111149	54433.179122	54433.179122	
2	100013	18159.919219	18101.079844	18101.079844	
3	100021	0.000000	0.000000	0.000000	
4	100023	0.000000	0.000000	0.000000	
5	100028	8085.058163	7968.609184	7968.609184	
6	100036	0.000000	0.000000	0.000000	
7	100042	33356.183036	33298.140000	33298.140000	
8	100043	208572.600000	208397.449091	208397.449091	
9	100047	0.000000	0.000000	0.000000	

Feature Engineering: previous_application

Feature Engineering: Feature Creation

In [ ]:

```
pre_app = df_pre_app.copy()
```

In [ ]:

```
# Create Features for the Bureau and previous_application
#-----------------------------------------------------
pre_app['PRE_APP_CREDIT_RATIO'] = pre_app['AMT_APPLICATION'] / pre_app['AMT_CREDIT']
pre_app['PRE_APP_CREDIT_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
pre_app['PRE_APP_CREDIT_RATIO'] = pre_app['PRE_APP_CREDIT_RATIO'].fillna(pre_app['PR

pre_app['PRE_DOWN_CREDIT_RATIO'] = pre_app['AMT_DOWN_PAYMENT'] / pre_app['AMT_CREDIT
pre_app['PRE_DOWN_CREDIT_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
pre_app['PRE_DOWN_CREDIT_RATIO'] = pre_app['PRE_DOWN_CREDIT_RATIO'].fillna(pre_app['

pre_app['PRE_DOWN_INT_RATIO'] = pre_app['RATE_DOWN_PAYMENT'] / pre_app['RATE_INTERES
pre_app['PRE_DOWN_INT_RATIO'].replace([np.inf, -np.inf], np.nan, inplace=True)
pre_app['PRE_DOWN_INT_RATIO'] = pre_app['PRE_DOWN_INT_RATIO'].fillna(pre_app['PRE_DO

pre_app['PRE_DUE_DATE_DIFF'] = pre_app['DAYS_LAST_DUE'] - pre_app['DAYS_FIRST_DUE']
```

In [ ]:

```
pre_app.columns
```

Out[48]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRI
CE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSIO
N',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL',
       'PRE_APP_CREDIT_RATIO', 'PRE_DOWN_CREDIT_RATIO', 'PRE_DOWN_INT_RATI
O',
       'PRE_DUE_DATE_DIFF'],
      dtype='object')
```

In [ ]:

```
pre_app.head(10)
```

Out[49]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
0	2030495	271877	Consumer loans	1730.430	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0
5	1383531	199383	Cash loans	23703.930	315000.0
6	2315218	175704	Cash loans	NaN	0.0
7	1656711	296299	Cash loans	NaN	0.0
8	2367563	342292	Cash loans	NaN	0.0
9	2579447	334349	Cash loans	NaN	0.0

10 rows × 41 columns

Feature Engineering: Correlation Analysis

In [ ]:

```
# Prepare the categorical features of previous_application
pre_app_ohe = OneHotCorr(pre_app)
```

In [ ]:

```
pre_app_corr = TargetCorr(df_app_train, pre_app_ohe)
print(pre_app_corr)
```

```
TARGET                                      1.000000
NAME_CONTRACT_STATUS_Refused                0.054458
CODE_REJECT_REASON_XAP                      0.052015
NAME_CONTRACT_STATUS_Approved               0.049161
NAME_PRODUCT_TYPE_walk-in                   0.042842
                                                ...
AMT_GOODS_PRICE                             0.000254
NAME_GOODS_CATEGORY_Weapon                  0.000232
NAME_TYPE_SUITE_Other_A                     0.000141
NAME_GOODS_CATEGORY_Mobile                  0.000117
NAME_GOODS_CATEGORY_House Construction           NaN
Name: TARGET, Length: 169, dtype: float64
```

Feature Engineering: Feature Selection

In [ ]:

```
pre_app_corr = pre_app_corr[1:].copy()
pre_app = pre_app_ohe[list(pre_app_corr[pre_app_corr >= 0.02].index) + ['SK_ID_CURR'
```

In [ ]:

```
pre_app.head(10)
```

Out[53]:

	NAME_CONTRACT_STATUS_Refused	CODE_REJECT_REASON_XAP	NAME_CONTRACT_ST/
0	0	1	
1	0	1	
2	0	1	
3	0	1	
4	1	0	
5	0	1	
6	0	1	
7	0	1	
8	0	1	
9	0	1	

10 rows × 31 columns

Feature Engineering: Feature Aggregation

In [ ]:

```
pre_app_final = pre_app.groupby(["SK_ID_CURR"], as_index = False).agg("mean")
```

In [ ]:

```
pre_app_final.head(10)
```

Out[55]:

	SK_ID_CURR	NAME_CONTRACT_STATUS_Refused	CODE_REJECT_REASON_XAP	NAME_(
0	100001	0.000000	1.000000	
1	100002	0.000000	1.000000	
2	100003	0.000000	1.000000	
3	100004	0.000000	1.000000	
4	100005	0.000000	1.000000	
5	100006	0.111111	0.888889	
6	100007	0.000000	1.000000	
7	100008	0.000000	1.000000	
8	100009	0.000000	1.000000	
9	100010	0.000000	1.000000	

10 rows × 31 columns

Feature Engineering: installments_payments

In [ ]:

```
installments_payments = df_installments_payments.copy()
```

Feature Engineering: Feature Creation

In [ ]:

```
# Create Features for the Bureau and installments_payments
#-------------------------------------------------

installments_payments['INST_PAYMENT_DELAY'] = installments_payments['DAYS_ENTRY_PAYM

installments_payments['INST_RATIO_AMT_PAID_DUE'] = installments_payments['AMT_PAYMEN
installments_payments['INST_RATIO_AMT_PAID_DUE'].replace([np.inf, -np.inf], np.nan,
installments_payments['INST_RATIO_AMT_PAID_DUE'] = installments_payments['INST_RATIC
```

In [ ]:

```
1 installments_payments.columns
```

Out[58]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT', 'INST_PAYMENT_DELAY',
       'INST_RATIO_AMT_PAID_DUE'],
      dtype='object')
```

In [ ]:

```
1 installments_payments.head(10)
```

Out[59]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DA
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	
5	1137312	164489	1.0	12	
6	2234264	184693	4.0	11	
7	1818599	111420	2.0	4	
8	2723183	112102	0.0	14	
9	1413990	109741	1.0	4	

Feature Engineering: Correlation Analysis

In [ ]:

```
1 installment_ohe = OneHotCorr(installments_payments)
```

In [ ]:

```
1 installment_corr = TargetCorr(df_app_train, installment_ohe)
```

In [ ]:

```
print(installment_corr)
```

```
TARGET                    1.000000
DAYS_ENTRY_PAYMENT        0.035122
DAYS_INSTALMENT           0.034974
NUM_INSTALMENT_NUMBER     0.016190
NUM_INSTALMENT_VERSION    0.009896
INST_PAYMENT_DELAY        0.008012
AMT_PAYMENT               0.003623
SK_ID_CURR                0.002533
AMT_INSTALMENT            0.001498
SK_ID_PREV                0.000212
INST_RATIO_AMT_PAID_DUE   0.000119
Name: TARGET, dtype: float64
```

Feature Engineering: Feature Selection

In [ ]:

```
installment_corr = installment_corr[1:].copy()
installment = installment_ohe[list(installment_corr[installment_corr >= 0.015].index
```

In [ ]:

```
installment.head(10)
```

Out[64]:

	DAYS_ENTRY_PAYMENT	DAYS_INSTALMENT	NUM_INSTALMENT_NUMBER	SK_ID_CURR
0	-1187.0	-1180.0	6	161674
1	-2156.0	-2156.0	34	151639
2	-63.0	-63.0	1	193053
3	-2426.0	-2418.0	3	199697
4	-1366.0	-1383.0	2	167756
5	-1417.0	-1384.0	12	164489
6	-352.0	-349.0	11	184693
7	-994.0	-968.0	4	111420
8	-197.0	-197.0	14	112102
9	-609.0	-570.0	4	109741

Feature Engineering: Feature Aggregation

In [ ]:

```
installment_final = installment.groupby(["SK_ID_CURR"], as_index=False).agg("mean")
```

Feature Engineering: Data Merging

In [ ]:

```
# Copy the application train and application test data
hcdr_train = df_app_train.copy()
hcdr_test = df_app_test.copy()
```

Data Merging: Labeled Data

In [ ]:

```
# merge all of the tables onto the application train set
hcdr_train = hcdr_train.merge(bur_final, how = 'left', on = 'SK_ID_CURR')
hcdr_train = hcdr_train.merge(pos_cash_final, how = 'left', on = 'SK_ID_CURR')
hcdr_train = hcdr_train.merge(credit_card_final, how = 'left', on = 'SK_ID_CURR')
hcdr_train = hcdr_train.merge(pre_app_final, how = 'left', on = 'SK_ID_CURR')
hcdr_train = hcdr_train.merge(installment_final, how = 'left', on = 'SK_ID_CURR')
```

Data Merging: Dropping columns with 80% missing values

In [ ]:

```
non_null_counts = hcdr_train.count()
```

In [ ]:

```
percent_non_nulls = non_null_counts / len(hcdr_train)
```

In [ ]:

```
cols_to_keep = percent_non_nulls[percent_non_nulls >= 0.2].index.tolist()
```

In [ ]:

```
hcdr_train = hcdr_train[cols_to_keep]
```

In [ ]:

```
hcdr_train.shape
```

Out[73]:

```
(307511, 179)
```

Data Merging: Unlabled Data

In [ ]:

```
hcdr_test = hcdr_test.merge(bur_final, how = 'left', on = 'SK_ID_CURR')
hcdr_test = hcdr_test.merge(pos_cash_final, how = 'left', on = 'SK_ID_CURR')
hcdr_test = hcdr_test.merge(credit_card_final, how = 'left', on = 'SK_ID_CURR')
hcdr_test = hcdr_test.merge(pre_app_final, how = 'left', on = 'SK_ID_CURR')
hcdr_test = hcdr_test.merge(installment_final, how = 'left', on = 'SK_ID_CURR')
```

In [ ]:

```
non_null_counts = hcdr_test.count()
```

In [ ]:

```
percent_non_nulls = non_null_counts / len(hcdr_test)
```

In [ ]:

```
cols_to_keep = percent_non_nulls[percent_non_nulls >= 0.2].index.tolist()
```

In [ ]:

```
hcdr_test = hcdr_test[cols_to_keep]
```

In [ ]:

```
hcdr_test.shape
```

Out[81]:

```
(48744, 183)
```

In [ ]:

```
hcdr_train.head(10)
```

Out[82]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLA(
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
5	100008	0	Cash loans	M	N	
6	100009	0	Cash loans	F	Y	
7	100010	0	Cash loans	M	Y	
8	100011	0	Cash loans	F	N	
9	100012	0	Revolving loans	M	N	

10 rows × 179 columns

In [ ]:

```
hcdr_test.to_csv("hcdr_test.csv", index=False)
hcdr_train.to_csv("hcdr_train.csv", index=False)
```

In [ ]:

```
# Reading from Downloaded HCDR Train and Test csv

#hcdr_train = pd.read_csv('./hcdr_train.csv')
#hcdr_test = pd.read_csv('./hcdr_test.csv')
```

Feature Engineering: Data Import & Final Selection

In [44]:

```
# Reload Data for Ram Conservation
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving hcdr_fe_data.zip to hcdr_fe_data (1).zip
```

In [48]:

```
!unzip hcdr_fe_data.zip
```

```
Archive:  hcdr_fe_data.zip
  inflating: hcdr_test.csv
  inflating: hcdr_train.csv
```

In [49]:

```
hcdr_train = pd.read_csv('hcdr_train.csv')
hcdr_test = pd.read_csv('hcdr_test.csv')
```

In [50]:

```
hcdr_train_nn = hcdr_train.copy()
hcdr_test_nn = hcdr_test.copy()
```

Feature Engineering: Data Pipeline

In [ ]:

```
import numpy as np
```

In [ ]:

```
# Split the provided training data into training and validationa and test
# The kaggle evaluation test set has no labels
from sklearn.model_selection import train_test_split


# Establish X and y
y = hcdr_train['TARGET'].copy()
X = hcdr_train.copy().drop(["TARGET"],axis=1)

# Seperate into categorical and numerical
cat_features = X.select_dtypes(include='object').columns
num_features = X.select_dtypes(include = ['int64','float64']).columns

X[num_features] = X[num_features].copy().replace(to_replace=(np.inf, -np.inf, np.nar
X[cat_features] = X[cat_features].replace(to_replace=(np.inf, -np.inf, np.nan), valu

# Split X & y into train & test sets
# Subsequently split train into train & validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_sta
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.
X_kaggle_test = hcdr_test

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
print(f"X X_kaggle_test   shape: {X_kaggle_test.shape}")
```

```
X train           shape: (209107, 178)
X validation      shape: (52277, 178)
X test            shape: (46127, 178)
X X_kaggle_test   shape: (48744, 183)
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.base import BaseEstimator, TransformerMixin

```

In [ ]:

```
# Create a class to select numerical or categorical columns
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [ ]:

```
# Seperate into categorical and numerical
cat_features_list = X.select_dtypes(include='object').columns.tolist()
#num_cat_cols = X.select_dtypes(include = ['int64','float64']).loc[:, X.nunique() <
num_features_list = X.select_dtypes(include = ['int64','float64']).columns.tolist()
```

In [ ]:

```
# number of categorical and numerical features
print("Number of Numerical Features: " + str(len(num_features_list)))
print("Number of Categorical Features: " + str(len(cat_features_list)))
```

```
Number of Numerical Features: 162
Number of Categorical Features: 16
```

In [ ]:

```
# Numerical Feature List
num_attribs = num_features_list

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler()),
    ])

# Categorical Feature List
cat_attribs = cat_features_list

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        #('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

# Final Data Pipeline
data_prep_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

Feature Engineering: Baseline Test Comparison

In [ ]:

```
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC"
                                  ])
```

In [ ]:

```
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn import metrics

#Create the Logistic Regression Pipeline
lr_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("lr", LogisticRegression())
    ])

#Fit the data to the pipeline
model = lr_pipeline.fit(X_train, y_train)


```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:4
58: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [ ]:

```
#Log the results of Accuracy and AUC for Train,Valid and Test datasets
exp_name = f"FE_Baseline_Logistic_Regression"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, model.predict(X_train)),
                accuracy_score(y_valid, model.predict(X_valid)),
                accuracy_score(y_test, model.predict(X_test)),
                roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog
```

Out[111]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Baseline_Logistic_Regression	0.92	0.9167	0.9195	0.754	0.7557	0.7509

Feature Engineering: Discussion - Approach

When preforming feature selection and feature engineering we took two main approaches. The first approach was specifically for feature selection. We obeserved which features of the data set were above a certaint correlation threshhold to the target value, ususally around 0.02 - 0.015. From there we selected those features and appended them to the current canidates for adding. This was combined with the step of feature engineering which used the RFM (Recency, Frequency, Monetary Value) ideas when thinking about possible constructions of features. We then added those to the selected features and again ran the correlation threshold against the target to select the final features to be considered and merged to the application_train.csv.

Feature Engineering: Discussion - Impact

After creating the feature engineered data set, we preformed Logistic Regression with the same baseline model that we used in phase two. Below you can see that in our inital phase we were preforming at best with a 0.7438 Test AUC score. Now we are preforming at 0.7296 Test AUC score, which is a good start to our improvments.

Baseline: No feature engineering

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.9200	0.9162	0.9194	0.7485	0.7475	0.7438
1	Baseline_Random_Forest	0.9999	0.9165	0.9194	1.0000	0.7102	0.7109
2	Baseline_Decision_Tree	1.0000	0.8528	0.8529	1.0000	0.5427	0.5367

Phase 3: Hyperparameter Tuning

Hyperparameter Tuning: Logistic Regression

In [ ]:

```
from time import time
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
#Create the Logistic Regression Pipeline
lr_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("lr", LogisticRegression(max_iter=100, random_state=42))
    ])

params = {'lr__C':[0.01, 0.1, 1.0, 10.0],
          'lr__penalty': ['l1','l2'],
          'lr__solver': ['saga']}




lr_clf_gridsearch_acc = GridSearchCV(lr_pipeline, param_grid=params, cv=3, scoring='
# For Accuracy
print("Performing grid search...")
print("pipeline:", [name for name, _ in lr_pipeline.steps])
print("parameters:")
print(params)
t0 = time()
lr_clf_gridsearch_acc.fit(X_train, y_train)
print("done in %0.3fs" % (time() - t0))
print()

print("Best parameters set found on development set:")
print()
print(lr_clf_gridsearch_acc.best_params_)
print()
print("Grid scores on development set:")
print()
means = lr_clf_gridsearch_acc.cv_results_['mean_test_score']
stds = lr_clf_gridsearch_acc.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, lr_clf_gridsearch_acc.cv_results_['params'
     print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
scoring='accuracy'
 # Print best accuracy score and best parameter combination
print("Best %s score: %0.3f" %(scoring, lr_clf_gridsearch_acc.best_score_))
print("Best parameters set:")
best_parameters = lr_clf_gridsearch_acc.best_estimator_.get_params()
for param_name in sorted(params.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#Sort the grid search results in decreasing order of average
sortedGridSearchResults = sorted(zip(lr_clf_gridsearch_acc.cv_results_["params"], lr
       key=lambda x: x[1], reverse=True)
print(f'Top 2 GridSearch results: ({scoring}, hyperparam Combo)\n {sortedGridSearchR
print()
```

```
Performing grid search...
pipeline: ['preparation', 'lr']
parameters:
{'lr__C': [0.01, 0.1, 1.0, 10.0], 'lr__penalty': ['l1', 'l2'], 'lr__solve
r': ['saga']}
Fitting 3 folds for each of 8 candidates, totalling 24 fits

/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(

done in 501.976s

Best parameters set found on development set:

{'lr__C': 0.01, 'lr__penalty': 'l1', 'lr__solver': 'saga'}

Grid scores on development set:

0.920 (+/-0.000) for {'lr__C': 0.01, 'lr__penalty': 'l1', 'lr__solver': 's
aga'}
0.920 (+/-0.000) for {'lr__C': 0.01, 'lr__penalty': 'l2', 'lr__solver': 's
aga'}
0.920 (+/-0.000) for {'lr__C': 0.1, 'lr__penalty': 'l1', 'lr__solver': 'sa
ga'}
0.920 (+/-0.000) for {'lr__C': 0.1, 'lr__penalty': 'l2', 'lr__solver': 'sa
ga'}
0.920 (+/-0.000) for {'lr__C': 1.0, 'lr__penalty': 'l1', 'lr__solver': 'sa
ga'}
0.920 (+/-0.000) for {'lr__C': 1.0, 'lr__penalty': 'l2', 'lr__solver': 'sa
ga'}
0.920 (+/-0.000) for {'lr__C': 10.0, 'lr__penalty': 'l1', 'lr__solver': 's
aga'}
0.920 (+/-0.000) for {'lr__C': 10.0, 'lr__penalty': 'l2', 'lr__solver': 's
aga'}

Best accuracy score: 0.920
Best parameters set:
        lr__C: 0.01
        lr__penalty: 'l1'
        lr__solver: 'saga'
Top 2 GridSearch results: (accuracy, hyperparam Combo)
 ({'lr__C': 0.01, 'lr__penalty': 'l1', 'lr__solver': 'saga'}, 0.9199739844
698708)
 ({'lr__C': 0.01, 'lr__penalty': 'l2', 'lr__solver': 'saga'}, 0.9199548560
331894)




/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_sag.py:350: C
onvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

In [ ]:

```
#Create the Logistic Regression Pipeline
lr_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("lr", LogisticRegression(max_iter=100, random_state=42))
    ])

params = {'lr__C':[0.01, 0.1, 1.0, 10.0],
          'lr__penalty': ['l1','l2'],
          'lr__solver': ['saga']}

# For AUC
lr_clf_gridsearch_auc = GridSearchCV(lr_pipeline, param_grid=params, cv=3, scoring='
print("Performing grid search...")
print("pipeline:", [name for name, _ in lr_pipeline.steps])
print("parameters:")
print(params)
t0 = time()
lr_clf_gridsearch_auc.fit(X_train, y_train)
print("done in %0.3fs" % (time() - t0))
print()

print("Best parameters set found on development set:")
print()
print(lr_clf_gridsearch_auc.best_params_)
print()
print("Grid scores on development set:")
print()
means = lr_clf_gridsearch_auc.cv_results_['mean_test_score']
stds = lr_clf_gridsearch_auc.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, lr_clf_gridsearch_auc.cv_results_['params'
     print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
scoring='roc_auc'
 # Print best accuracy score and best parameter combination
print("Best %s score: %0.3f" %(scoring, lr_clf_gridsearch_auc.best_score_))
print("Best parameters set:")
best_parameters = lr_clf_gridsearch_auc.best_estimator_.get_params()
for param_name in sorted(params.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#Sort the grid search results in decreasing order of average
sortedGridSearchResults = sorted(zip(lr_clf_gridsearch_auc.cv_results_["params"], lr
       key=lambda x: x[1], reverse=True)
print(f'Top 2 GridSearch results: ({scoring}, hyperparam Combo)\n {sortedGridSearchR
print()
```

```
Performing grid search...
pipeline: ['preparation', 'lr']
parameters:
{'lr__C': [0.01, 0.1, 1.0, 10.0], 'lr__penalty': ['l1', 'l2'], 'lr__solve
r': ['saga']}
Fitting 3 folds for each of 8 candidates, totalling 24 fits

/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

```
done in 484.026s

Best parameters set found on development set:

{'lr__C': 10.0, 'lr__penalty': 'l2', 'lr__solver': 'saga'}

Grid scores on development set:

0.743 (+/-0.003) for {'lr__C': 0.01, 'lr__penalty': 'l1', 'lr__solver': 's
aga'}
0.747 (+/-0.004) for {'lr__C': 0.01, 'lr__penalty': 'l2', 'lr__solver': 's
aga'}
0.747 (+/-0.004) for {'lr__C': 0.1, 'lr__penalty': 'l1', 'lr__solver': 'sa
ga'}
0.747 (+/-0.004) for {'lr__C': 0.1, 'lr__penalty': 'l2', 'lr__solver': 'sa
ga'}
0.747 (+/-0.004) for {'lr__C': 1.0, 'lr__penalty': 'l1', 'lr__solver': 'sa
ga'}
0.747 (+/-0.004) for {'lr__C': 1.0, 'lr__penalty': 'l2', 'lr__solver': 'sa
ga'}
0.747 (+/-0.004) for {'lr__C': 10.0, 'lr__penalty': 'l1', 'lr__solver': 's
aga'}
0.747 (+/-0.004) for {'lr__C': 10.0, 'lr__penalty': 'l2', 'lr__solver': 's
aga'}

Best roc_auc score: 0.747
Best parameters set:
        lr__C: 10.0
        lr__penalty: 'l2'
        lr__solver: 'saga'
Top 2 GridSearch results: (roc_auc, hyperparam Combo)
 ({'lr__C': 10.0, 'lr__penalty': 'l2', 'lr__solver': 'saga'}, 0.7468627515
459977)
 ({'lr__C': 1.0, 'lr__penalty': 'l2', 'lr__solver': 'saga'}, 0.74686181139
98566)




/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_sag.py:350: C
onvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

Hyperparameter Tuning: Random Forest

In [ ]:

```
rf_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("rf", RandomForestClassifier(random_state=42))
    ])


params = {
          'rf__max_depth': [5, 10, 15, 20, 50],
          'rf__max_features': ['log2', 'sqrt'],
          'rf__n_estimators' : [1, 10, 50, 100]}


# Using gridsearch here to determine the Accuracy percentage for the train, validati
rf_clf_gridsearch_acc = GridSearchCV(rf_pipeline, param_grid=params, cv=3, scoring='
# For Accuracy
print("Performing grid search...")
print("pipeline:", [name for name, _ in rf_pipeline.steps])
print("parameters:")
print(params)
t0 = time()
rf_clf_gridsearch_acc.fit(X_train, y_train)
print("done in %0.3fs" % (time() - t0))
print()

print("Best parameters set found on development set:")
print()
print(rf_clf_gridsearch_acc.best_params_)
print()
print("Grid scores on development set:")
print()
means = rf_clf_gridsearch_acc.cv_results_['mean_test_score']
stds = rf_clf_gridsearch_acc.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, rf_clf_gridsearch_acc.cv_results_['params'
     print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
scoring='accuracy'
 # Print best accuracy score and best parameter combination
print("Best %s score: %0.3f" %(scoring, rf_clf_gridsearch_acc.best_score_))
print("Best parameters set:")
best_parameters = rf_clf_gridsearch_acc.best_estimator_.get_params()
for param_name in sorted(params.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#Sort the grid search results in decreasing order of average
sortedGridSearchResults = sorted(zip(rf_clf_gridsearch_acc.cv_results_["params"], rf
       key=lambda x: x[1], reverse=True)
print(f'Top 2 GridSearch results: ({scoring}, hyperparam Combo)\n {sortedGridSearchR
print()
```

```
Performing grid search...
pipeline: ['preparation', 'rf']
parameters:
{'rf__max_depth': [5, 10, 15, 20, 50], 'rf__max_features': ['log2', 'sqr
t'], 'rf__n_estimators': [1, 10, 50, 100]}
Fitting 3 folds for each of 40 candidates, totalling 120 fits

/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

```
done in 760.421s

Best parameters set found on development set:

{'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf__n_estimators': 50}

Grid scores on development set:

0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 100}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 100}
0.917 (+/-0.001) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.916 (+/-0.001) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.905 (+/-0.001) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.900 (+/-0.003) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.885 (+/-0.001) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
__n_estimators': 1}
0.919 (+/-0.000) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
```

In [ ]:

```
rf_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("rf", RandomForestClassifier(random_state=42))
    ])
params = {'rf__max_depth': [5, 10, 15, 20, 50],
          'rf__max_features': ['log2', 'sqrt'],
          'rf__n_estimators' : [100, 200, 300]}
# Using gridsearch here to determine the ROC-AUC scores for the train, validation, a
rf_clf_gridsearch_auc = GridSearchCV(rf_pipeline, param_grid=params, cv=3, scoring='
# For AUC
print("Performing grid search...")
print("pipeline:", [name for name, _ in rf_pipeline.steps])
print("parameters:")
print(params)
t0 = time()
rf_clf_gridsearch_auc.fit(X_train, y_train)
print("done in %0.3fs" % (time() - t0))
print()

print("Best parameters set found on development set:")
print()
print("Best parameters set:")
print(rf_clf_gridsearch_auc.best_params_)
print()
print("Grid scores on development set:")
print()
means = rf_clf_gridsearch_auc.cv_results_['mean_test_score']
stds = rf_clf_gridsearch_auc.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, rf_clf_gridsearch_auc.cv_results_['params'
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
scoring='roc_auc'
 # Print best accuracy score and best parameter combination
print("Best %s score: %0.3f" %(scoring, rf_clf_gridsearch_auc.best_score_))
print("Best parameters set:")
best_parameters = rf_clf_gridsearch_auc.best_estimator_.get_params()
for param_name in sorted(params.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#Sort the grid search results in decreasing order of average
sortedGridSearchResults = sorted(zip(rf_clf_gridsearch_auc.cv_results_["params"], rf
       key=lambda x: x[1], reverse=True)
print(f'Top 2 GridSearch results: ({scoring}, hyperparam Combo)\n {sortedGridSearchR
print()
```

```
0.920 (+/-0.000) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
__n_estimators': 50}
0.881 (+/-0.001) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.919 (+/-0.000) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.853 (+/-0.003) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
__n_estimators': 1}
0.920 (+/-0.000) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.852 (+/-0.003) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 1}
0.919 (+/-0.000) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 10}
0.920 (+/-0.000) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 50}
0.920 (+/-0.000) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}

Best accuracy score: 0.920

Best parameters set:
    rf__max_depth: 50
    rf__max_features: 'sqrt'
    rf__n_estimators: 50

Top 2 GridSearch results: (accuracy, hyperparam Combo)
 ({'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf__n_estimators': 5
0}, 0.9200218068830844)
 ({'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf__n_estimators': 10
0}, 0.9200026779138984)
```

```
Performing grid search...
pipeline: ['preparation', 'rf']
parameters:
{'rf__max_depth': [5, 10, 15, 20, 50], 'rf__max_features': ['log2', 'sqr
t'], 'rf__n_estimators': [100, 200, 300]}
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

```
/usr/local/lib/python3.9/dist-packages/joblib/externals/loky/process_execu
tor.py:700: UserWarning: A worker stopped while some jobs were given to th
e executor. This can be caused by a too short worker timeout or by a memor
y leak.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

```
done in 2539.990s

Best parameters set found on development set:

{'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf__n_estimators': 300}

Grid scores on development set:

0.703 (+/-0.007) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 100}
0.706 (+/-0.005) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 200}
0.706 (+/-0.006) for {'rf__max_depth': 5, 'rf__max_features': 'log2', 'rf_
_n_estimators': 300}
0.717 (+/-0.005) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 100}
0.719 (+/-0.006) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 200}
0.719 (+/-0.006) for {'rf__max_depth': 5, 'rf__max_features': 'sqrt', 'rf_
_n_estimators': 300}
0.717 (+/-0.006) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.720 (+/-0.005) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 200}
0.720 (+/-0.004) for {'rf__max_depth': 10, 'rf__max_features': 'log2', 'rf
__n_estimators': 300}
0.730 (+/-0.001) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.732 (+/-0.002) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 200}
0.733 (+/-0.002) for {'rf__max_depth': 10, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 300}
0.715 (+/-0.003) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.722 (+/-0.003) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 200}
0.724 (+/-0.003) for {'rf__max_depth': 15, 'rf__max_features': 'log2', 'rf
__n_estimators': 300}
0.729 (+/-0.002) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.733 (+/-0.002) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 200}
0.734 (+/-0.002) for {'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 300}
0.709 (+/-0.004) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.719 (+/-0.004) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
__n_estimators': 200}
0.721 (+/-0.004) for {'rf__max_depth': 20, 'rf__max_features': 'log2', 'rf
__n_estimators': 300}
0.717 (+/-0.001) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.724 (+/-0.002) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 200}
0.726 (+/-0.002) for {'rf__max_depth': 20, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 300}
0.698 (+/-0.008) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
__n_estimators': 100}
0.709 (+/-0.009) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
__n_estimators': 200}
0.713 (+/-0.007) for {'rf__max_depth': 50, 'rf__max_features': 'log2', 'rf
```

```
__n_estimators': 300}
0.705 (+/-0.005) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 100}
0.715 (+/-0.004) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 200}
0.719 (+/-0.002) for {'rf__max_depth': 50, 'rf__max_features': 'sqrt', 'rf
__n_estimators': 300}

Best roc_auc score: 0.734
Best parameters set:
        rf__max_depth: 15
        rf__max_features: 'sqrt'
        rf__n_estimators: 300
Top 2 GridSearch results: (roc_auc, hyperparam Combo)
 ({'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf__n_estimators': 30
0}, 0.7343871139331194)
 ({'rf__max_depth': 15, 'rf__max_features': 'sqrt', 'rf__n_estimators': 20
0}, 0.7334145221400225)
```

Hyperparameter Tuning: Decision Tree

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier

dt_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("dt", DecisionTreeClassifier(random_state=42))
    ])


params = {'dt__criterion':['gini', 'entropy'],
          'dt__max_depth': [5, 10, 15, 20, 50],
          'dt__min_samples_leaf' : [1,2,3,4,5]}



# Using gridsearch here to determine the Accuracy percentage for the train, validati
dt_clf_gridsearch_acc = GridSearchCV(dt_pipeline, param_grid=params, cv=3, scoring='

# For Accuracy
print("Performing grid search...")
print("pipeline:", [name for name, _ in dt_pipeline.steps])
print("parameters:")
print(params)
t0 = time()
dt_clf_gridsearch_acc.fit(X_train, y_train)
print("done in %0.3fs" % (time() - t0))
print()

print("Best parameters set found on development set:")
print()
print(dt_clf_gridsearch_acc.best_params_)
print()
print("Grid scores on development set:")
print()
means = dt_clf_gridsearch_acc.cv_results_['mean_test_score']
stds = dt_clf_gridsearch_acc.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, dt_clf_gridsearch_acc.cv_results_['params'
     print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
scoring='accuracy'
 # Print best accuracy score and best parameter combination
print("Best %s score: %0.3f" %(scoring, dt_clf_gridsearch_acc.best_score_))
print("Best parameters set:")
best_parameters = dt_clf_gridsearch_acc.best_estimator_.get_params()
for param_name in sorted(params.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#Sort the grid search results in decreasing order of average
sortedGridSearchResults = sorted(zip(dt_clf_gridsearch_acc.cv_results_["params"], dt
       key=lambda x: x[1], reverse=True)
print(f'Top 2 GridSearch results: ({scoring}, hyperparam Combo)\n {sortedGridSearchR
print()
```

```
Performing grid search...
pipeline: ['preparation', 'dt']
parameters:
{'dt__criterion': ['gini', 'entropy'], 'dt__max_depth': [5, 10, 15, 20, 5
0], 'dt__min_samples_leaf': [1, 2, 3, 4, 5]}
Fitting 3 folds for each of 50 candidates, totalling 150 fits

/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

```
done in 657.871s

Best parameters set found on development set:

{'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt__min_samples_leaf':
1}

Grid scores on development set:

0.920 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 1}
0.920 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 2}
0.920 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 3}
0.920 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 4}
0.920 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 5}
0.914 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 1}
0.914 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 2}
0.914 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 3}
0.914 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 4}
0.913 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 5}
0.903 (+/-0.000) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 1}
0.902 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 2}
0.898 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 3}
0.900 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 4}
0.898 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 5}
0.889 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 1}
0.889 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 2}
0.882 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 3}
0.888 (+/-0.004) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 4}
0.884 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 5}
0.854 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 1}
0.872 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 2}
0.864 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 3}
0.878 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 4}
0.875 (+/-0.001) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 5}
0.920 (+/-0.000) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 1}
```

```
0.920 (+/-0.000) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 2}
0.920 (+/-0.000) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 3}
0.920 (+/-0.000) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 4}
0.920 (+/-0.000) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 5}
0.916 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 1}
0.916 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 2}
0.916 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 3}
0.916 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 4}
0.916 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 5}
0.897 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 1}
0.899 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 2}
0.897 (+/-0.000) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 3}
0.899 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 4}
0.898 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 5}
0.876 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 1}
0.879 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 2}
0.875 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 3}
0.879 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 4}
0.877 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 5}
0.858 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 1}
0.864 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 2}
0.859 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 3}
0.865 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 4}
0.864 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 5}

Best accuracy score: 0.920
Best parameters set:
        dt__criterion: 'entropy'
        dt__max_depth: 5
        dt__min_samples_leaf: 1
Top 2 GridSearch results: (accuracy, hyperparam Combo)
 ({'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt__min_samples_leaf':
1}, 0.9199261625205525)
 ({'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt__min_samples_leaf':
2}, 0.9199261625205525)
```

In [ ]:

```
dt_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("dt", DecisionTreeClassifier(random_state=42))
    ])

params = {'dt__criterion':['gini', 'entropy'],
          'dt__max_depth': [5, 10, 15, 20, 50],
          'dt__min_samples_leaf' : [1,2,3,4,5]}


# Using gridsearch here to determine the ROC-AUC scores for the train, validation, a
dt_clf_gridsearch_auc = GridSearchCV(dt_pipeline, param_grid=params, cv=3, scoring='

# For AUC
print("Performing grid search...")
print("pipeline:", [name for name, _ in dt_pipeline.steps])
print("parameters:")
print(params)
t0 = time()
dt_clf_gridsearch_auc.fit(X_train, y_train)
print("done in %0.3fs" % (time() - t0))
print()

print("Best parameters set found on development set:")
print()
print(dt_clf_gridsearch_auc.best_params_)
print()
print("Grid scores on development set:")
print()
means = dt_clf_gridsearch_auc.cv_results_['mean_test_score']
stds = dt_clf_gridsearch_auc.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, dt_clf_gridsearch_auc.cv_results_['params'
     print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
scoring='roc_auc'
 # Print best accuracy score and best parameter combination
print("Best %s score: %0.3f" %(scoring, dt_clf_gridsearch_auc.best_score_))
print("Best parameters set:")
best_parameters = dt_clf_gridsearch_auc.best_estimator_.get_params()
for param_name in sorted(params.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#Sort the grid search results in decreasing order of average
sortedGridSearchResults = sorted(zip(dt_clf_gridsearch_auc.cv_results_["params"], dt
       key=lambda x: x[1], reverse=True)
print(f'Top 2 GridSearch results: ({scoring}, hyperparam Combo)\n {sortedGridSearchR
print()
```

```
Performing grid search...
pipeline: ['preparation', 'dt']
parameters:
{'dt__criterion': ['gini', 'entropy'], 'dt__max_depth': [5, 10, 15, 20, 5
0], 'dt__min_samples_leaf': [1, 2, 3, 4, 5]}
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

```
done in 655.804s

Best parameters set found on development set:

{'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__min_samples_leaf': 2}

Grid scores on development set:

0.703 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 1}
0.703 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 2}
0.703 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 3}
0.703 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 4}
0.703 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__mi
n_samples_leaf': 5}
0.700 (+/-0.006) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 1}
0.699 (+/-0.009) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 2}
0.697 (+/-0.009) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 3}
0.695 (+/-0.012) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 4}
0.694 (+/-0.009) for {'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__m
in_samples_leaf': 5}
0.635 (+/-0.006) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 1}
0.628 (+/-0.006) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 2}
0.622 (+/-0.003) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 3}
0.618 (+/-0.002) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 4}
0.616 (+/-0.008) for {'dt__criterion': 'gini', 'dt__max_depth': 15, 'dt__m
in_samples_leaf': 5}
0.555 (+/-0.018) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 1}
0.545 (+/-0.017) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 2}
0.541 (+/-0.023) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 3}
0.543 (+/-0.023) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 4}
0.545 (+/-0.015) for {'dt__criterion': 'gini', 'dt__max_depth': 20, 'dt__m
in_samples_leaf': 5}
0.536 (+/-0.011) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 1}
0.541 (+/-0.009) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 2}
0.547 (+/-0.005) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 3}
0.551 (+/-0.004) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 4}
0.560 (+/-0.006) for {'dt__criterion': 'gini', 'dt__max_depth': 50, 'dt__m
in_samples_leaf': 5}
0.703 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 1}
0.703 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
```

```
_min_samples_leaf': 2}
0.703 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 3}
0.703 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 4}
0.703 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 5, 'dt_
_min_samples_leaf': 5}
0.679 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 1}
0.680 (+/-0.001) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 2}
0.679 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 3}
0.678 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 4}
0.678 (+/-0.002) for {'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt
__min_samples_leaf': 5}
0.605 (+/-0.011) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
_min_samples_leaf': 1}
0.608 (+/-0.012) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 2}
0.609 (+/-0.011) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 3}
0.610 (+/-0.009) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 4}
0.610 (+/-0.009) for {'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt
__min_samples_leaf': 5}
0.566 (+/-0.011) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 1}
0.568 (+/-0.009) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 2}
0.572 (+/-0.003) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 3}
0.573 (+/-0.006) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 4}
0.576 (+/-0.010) for {'dt__criterion': 'entropy', 'dt__max_depth': 20, 'dt
__min_samples_leaf': 5}
0.539 (+/-0.005) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 1}
0.544 (+/-0.004) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 2}
0.547 (+/-0.006) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 3}
0.554 (+/-0.006) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 4}
0.559 (+/-0.005) for {'dt__criterion': 'entropy', 'dt__max_depth': 50, 'dt
__min_samples_leaf': 5}

Best roc_auc score: 0.703
Best parameters set:
        dt__criterion: 'gini'
        dt__max_depth: 5
        dt__min_samples_leaf: 2
Top 2 GridSearch results: (roc_auc, hyperparam Combo)
 ({'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__min_samples_leaf':
2}, 0.7030256595845471)
 ({'dt__criterion': 'gini', 'dt__max_depth': 5, 'dt__min_samples_leaf':
3}, 0.7030256595845471)
```

Phase 3: Modeling Pipelines

```
try:
    bestPipeLog
except NameError:
    bestPipeLog = pd.DataFrame(columns=["exp_name",
                                        "Train Acc",
                                        "Valid Acc",
                                        "Test  Acc",
                                        "Train AUC",
                                        "Valid AUC",
                                        "Test  AUC"
                                       ])
```

Logistic Regression: Pipeline

```
#Create the Logistic Regression PipeLine
lr_pipeline = Pipeline([
    ("preparation", data_prep_pipeline),
    ("lr", LogisticRegression(C = 10.0, penalty="l2", solver="saga", random_stat
])
lr_model = lr_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_sag.py:350: C
onvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

In [ ]:

```
exp_name = f"FE_Best_Param_Logistic_Reg"
bestPipeLog.loc[len(bestPipeLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, lr_model.predict(X_train)),
                accuracy_score(y_valid, lr_model.predict(X_valid)),
                accuracy_score(y_test, lr_model.predict(X_test)),
                roc_auc_score(y_train, lr_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, lr_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, lr_model.predict_proba(X_test)[:, 1])],
    4))
bestPipeLog
```

Out[141]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.92	0.9166	0.9196	0.7468	0.7513	0.7478
1	FE_Best_Param_Logistic_Reg	0.92	0.9166	0.9195	0.7498	0.7535	0.7491

Random Forest: Pipeline

In [ ]:

```
rf_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("rf", RandomForestClassifier(max_depth = 15, max_features = "sqrt", n_estim
    ])

rf_model = rf_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

In [ ]:

```
exp_name = f"FE_Best_Param_Random_Forest"
bestPipeLog.loc[len(bestPipeLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, rf_model.predict(X_train)),
                accuracy_score(y_valid, rf_model.predict(X_valid)),
                accuracy_score(y_test, rf_model.predict(X_test)),
                roc_auc_score(y_train, rf_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, rf_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1])],
    4))
bestPipeLog
```

Out[147]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9196	0.7468	0.7513	0.7478
1	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9195	0.7498	0.7535	0.7491
2	Best_Param_Random_Forest	0.9230	0.9164	0.9194	0.9689	0.7415	0.7379
3	Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
4	FE_Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402

Decision Tree: Pipeline

In [ ]:

```
dt_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("dt", DecisionTreeClassifier(criterion="gini", max_depth = 5, min_samples_l
    ])

dt_model = dt_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

In [ ]:

```
exp_name = f"Best_Param_Decision_Tree"
bestPipeLog.loc[len(bestPipeLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, dt_model.predict(X_train)),
                accuracy_score(y_valid, dt_model.predict(X_valid)),
                accuracy_score(y_test, dt_model.predict(X_test)),
                roc_auc_score(y_train, dt_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, dt_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, dt_model.predict_proba(X_test)[:, 1])],
    4))
bestPipeLog
```

Out[149]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9196	0.7468	0.7513	0.7478
1	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9195	0.7498	0.7535	0.7491
2	Best_Param_Random_Forest	0.9230	0.9164	0.9194	0.9689	0.7415	0.7379
3	Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
4	FE_Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
5	Best_Param_Decision_Tree	0.9200	0.9164	0.9194	0.7116	0.7011	0.6999

Pipeline: Best Result

In [ ]:

```
bestPipeLog
```

Out[150]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9196	0.7468	0.7513	0.7478
1	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9195	0.7498	0.7535	0.7491
2	Best_Param_Random_Forest	0.9230	0.9164	0.9194	0.9689	0.7415	0.7379
3	Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
4	FE_Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
5	Best_Param_Decision_Tree	0.9200	0.9164	0.9194	0.7116	0.7011	0.6999

From the experiment log above we can see that the best preforming model is the Random Forest Pipeline with our tuned hyperparameters of **max_depth = 10, max_features = "sqrt", n_estimators = 100**

Input Feature: Analysis

In [ ]:

```
# Split the provided training data into training and validationa and test
# The kaggle evaluation test set has no labels
from sklearn.model_selection import train_test_split


# Establish X and y
y = hcdr_train['TARGET'].copy()
X = hcdr_train.copy().drop(["TARGET"],axis=1)

# Seperate into categorical and numerical
cat_cols = X.select_dtypes(include='object').columns
num_cat_cols = X.select_dtypes(include = ['int64','float64']).loc[:, X.nunique() < 1
num_features = X.select_dtypes(include = ['int64','float64']).loc[:, X.nunique() >=
cat_features = np.concatenate([cat_cols, num_cat_cols])

X[num_features] = X[num_features].copy().replace(to_replace=(np.inf, -np.inf, np.nar
X[cat_features] = X[cat_features].replace(to_replace=(np.inf, -np.inf, np.nan), valu

# Split X & y into train & test sets
# Subsequently split train into train & validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_sta
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.
X_kaggle_test = hcdr_test

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
print(f"X X_kaggle_test   shape: {X_kaggle_test.shape}")
```

```
X train           shape: (209107, 178)
X validation      shape: (52277, 178)
X test            shape: (46127, 178)
X X_kaggle_test   shape: (48744, 183)
```

In [ ]:

```
# number of categorical and numerical features
print("Number of Numerical Features: " + str(len(num_features_list)))
print("Number of Categorical Features: " + str(len(cat_features_list)))
```

```
Number of Numerical Features: 162
Number of Categorical Features: 16
```

Modeling Pipelines : Loss Functions

- L1 Loss (Mean Absolute Error):

$$L_{1}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \left| x_{i} - y_{i} \right|$$

where:
x and y are the predicted and actual values, respectively n is the number of samples in the dataset i is the index of each sample in the dataset
$\left| \cdot \right|$ denotes the absolute value
The L1 loss function measures the absolute difference between the predicted values and actual values, and then takes the mean of those differences. It is less sensitive to outliers than the L2 loss function.

- L2 Loss (Mean Squared Error):

$$L_{2}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \left(x_{i} - y_{i} \right)^{2}$$

where:
x and y are the predicted and actual values, respectively n is the number of samples in the dataset i is the index of each sample in the dataset
This loss function is commonly used in regression problems, where the goal is to predict continuous values. It measures the average of the squared differences between the predicted and actual values. The L2 loss function measures the squared difference between the predicted values and actual values, and then takes the mean of those differences. It is more sensitive to outliers than the L1 loss function.

Modeling Pipelines : Metrics

- Accuracy Score: The accuracy score is a metric that measures the proportion of correctly classified samples out of all samples. It is useful when the classes in a dataset are balanced. However, it can be misleading in situations where the classes are imbalanced. The accuracy score ranges from 0 to 1, where 1 represents perfect classification. It is calculated as:

$$accuracy = \frac{number\ of\ correctly\ classified\ samples}{total\ number\ of\ samples}$$

- AUC (Area Under the ROC Curve): The AUC is a metric that measures the performance of a binary classification model by calculating the area under the receiver operating characteristic (ROC) curve. The ROC curve is a graph that shows the true positive rate (sensitivity) against the false positive rate (1 - specificity) at different classification thresholds. The AUC ranges from 0 to 1, where 1 represents perfect classification. It is useful in situations where the classes are imbalanced and where the model's output is a probability. The AUC can be calculated using the trapezoidal rule or other numerical integration methods.

Pipeline Visualization of Steps

In [ ]:

```
logicModel = dt_clf_gridsearch_auc.best_estimator_
display(logicModel)
```

```
Pipeline(steps=[('preparation',
                 FeatureUnion(transformer_list=[('num_pipeline',
                                                 Pipeline(steps=[('selecto
r',
                                                                  DataFram
eSelector(attribute_names=['SK_ID_CURR',

'CNT_CHILDREN',

'AMT_INCOME_TOTAL',

'AMT_CREDIT',

'AMT_ANNUITY',

'AMT_GOODS_PRICE',

'REGION_POPULATION_RELATIVE',

'DAYS_BIRTH',

'DAYS_EMPLOYED',

'DAYS_REGISTRATION',

'DAYS_ID_PUBLISH',

'OWN_CAR_AGE',

'FLAG_M...

'WEEKDAY_APPR_PROCESS_START',

'ORGANIZATION_TYPE',

'FONDKAPREMONT_MODE',

'HOUSETYPE_MODE',

'WALLSMATERIAL_MODE',

'EMERGENCYSTATE_MODE'])),
                                                                 ('impute
r',
                                                                  SimpleIm
puter(strategy='most_frequent')),
                                                                 ('ohe',
                                                                  OneHotEn
coder(handle_unknown='ignore',

sparse=False,
```

```
sparse_output=False))]))]),
                ('dt',
                 DecisionTreeClassifier(max_depth=5, min_samples_leaf=2,
                                        random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
logicModel = lr_clf_gridsearch_auc.best_estimator_
display(logicModel)
```

```
Pipeline(steps=[('preparation',
                 FeatureUnion(transformer_list=[('num_pipeline',
                                                 Pipeline(steps=[('selecto
r',
                                                                  DataFram
eSelector(attribute_names=['SK_ID_CURR',
'CNT_CHILDREN',
'AMT_INCOME_TOTAL',
'AMT_CREDIT',
'AMT_ANNUITY',
'AMT_GOODS_PRICE',
'REGION_POPULATION_RELATIVE',
'DAYS_BIRTH',
'DAYS_EMPLOYED',
'DAYS_REGISTRATION',
'DAYS_ID_PUBLISH',
'OWN_CAR_AGE',
'FLAG_M...
'OCCUPATION_TYPE',
'WEEKDAY_APPR_PROCESS_START',
'ORGANIZATION_TYPE',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE'])),
                                                                  ('impute
r',
                                                                   SimpleIm
puter(strategy='most_frequent')),
                                                                  ('ohe',
                                                                   OneHotEn
coder(handle_unknown='ignore',
```

```
sparse=False,

sparse_output=False))]))]),
                ('lr',
                 LogisticRegression(C=10.0, random_state=42, solver='sag
a'))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
logicModel = rf_clf_gridsearch_auc.best_estimator_
display(logicModel)
```

```
Pipeline(steps=[('preparation',
                 FeatureUnion(transformer_list=[('num_pipeline',
                                                 Pipeline(steps=[('selecto
r',
                                                                  DataFram
eSelector(attribute_names=['SK_ID_CURR',
'CNT_CHILDREN',
'AMT_INCOME_TOTAL',
'AMT_CREDIT',
'AMT_ANNUITY',
'AMT_GOODS_PRICE',
'REGION_POPULATION_RELATIVE',
'DAYS_BIRTH',
'DAYS_EMPLOYED',
'DAYS_REGISTRATION',
'DAYS_ID_PUBLISH',
'OWN_CAR_AGE',
'FLAG_M...
'OCCUPATION_TYPE',
'WEEKDAY_APPR_PROCESS_START',
'ORGANIZATION_TYPE',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE'])),
                                                                  ('impute
r',
                                                                   SimpleIm
puter(strategy='most_frequent')),
                                                                  ('ohe',
                                                                   OneHotEn
coder(handle_unknown='ignore',
```

```
sparse=False,

sparse_output=False))]))])),
                ('rf',
                 RandomForestClassifier(max_depth=15, n_estimators=300,
                                        random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Phase 3: Results & Discussion of Results

BASELINE Experiments

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.9199	0.9165	0.9193	0.7534	0.7542	0.7511
1	Baseline_Logistic_Regression	0.9201	0.9167	0.9195	0.7556	0.7569	0.7521
2	Baseline_Logistic_Regression	0.9201	0.9167	0.9194	0.7556	0.7569	0.7521
3	FE_Baseline_Logistic_Regression	0.9200	0.9165	0.9195	0.7293	0.7318	0.7296

In [ ]:

```
1 bestPipeLog
```

Out[156]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9196	0.7468	0.7513	0.7478
1	FE_Best_Param_Logistic_Reg	0.9200	0.9166	0.9195	0.7498	0.7535	0.7491
2	Best_Param_Random_Forest	0.9230	0.9164	0.9194	0.9689	0.7415	0.7379
3	Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
4	FE_Best_Param_Random_Forest	0.9228	0.9164	0.9194	0.9707	0.7441	0.7402
5	Best_Param_Decision_Tree	0.9200	0.9164	0.9194	0.7116	0.7011	0.6999

Kaggle submission of Best Model with Hyperparameter Tuning

In [ ]:

```
1 model = rf_model
2 test_class_scores = model.predict_proba(X_kaggle_test)[:, 1]
```

In [ ]:

```
1 test_class_scores[0:10]
```

Out[158]:

```
array([0.06589143, 0.12248617, 0.04898443, 0.05315793, 0.11280577,
       0.06505853, 0.03768917, 0.08023548, 0.03728518, 0.15715383])
```

In [ ]:

```
1 X_kaggle_test.columns
```

Out[159]:

```
Index(['SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
       'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDI
T',
       'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       ...
       'PRE_DUE_DATE_DIFF', 'NAME_SELLER_INDUSTRY_Consumer electronics',
       'PRODUCT_COMBINATION_Cash X-Sell: low',
       'PRODUCT_COMBINATION_POS industry with interest', 'NAME_PORTFOLIO_X
NA',
       'PRE_APP_CREDIT_RATIO', 'DAYS_ENTRY_PAYMENT', 'DAYS_INSTALMENT',
       'NUM_INSTALMENT_NUMBER', 'SK_ID_PREV'],
      dtype='object', length=183)
```

In [ ]:

```
1 # Submission dataframe
2 submit_df = df_app_test[['SK_ID_CURR']].copy()
3 submit_df['TARGET'] = test_class_scores
4
5 submit_df.head()
```

Out[160]:

	SK_ID_CURR	TARGET
0	100001	0.065891
1	100005	0.122486
2	100013	0.048984
3	100028	0.053158
4	100038	0.112806

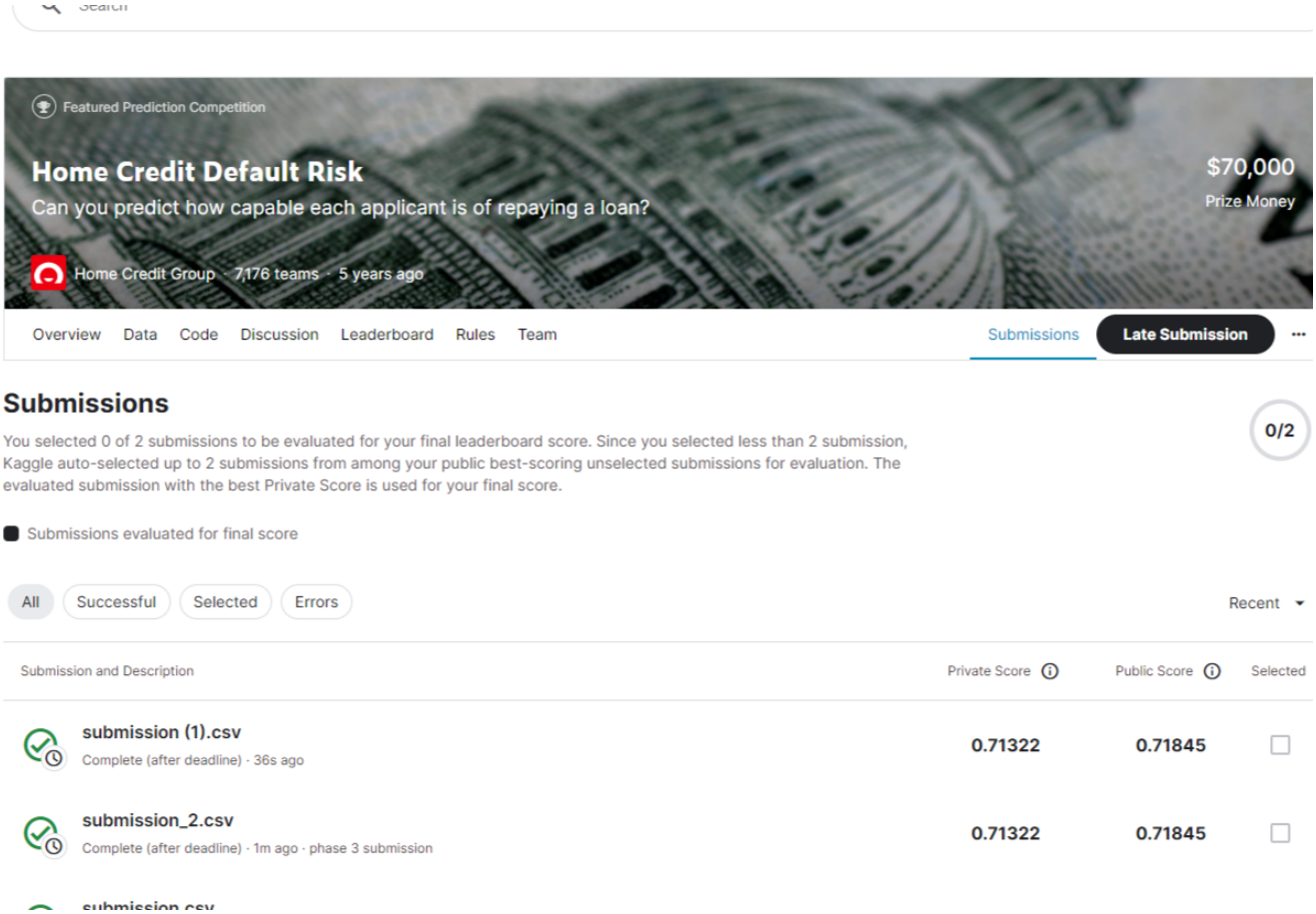
In [ ]:

```
1 submit_df.to_csv("submission_2.csv",index=False)
```

In [ ]:

```
1 ! kaggle competitions submit -c home-credit-default-risk -f submission_2.csv -m "pha
```

```
100% 1.26M/1.26M [00:00<00:00, 3.46MB/s]
Successfully submitted to Home Credit Default Risk
```



Phase 3: Results discussion

As part of Phase 3 our main goals were to perform Feature Engineering and Hyperparamter Tuning for our Models. Since the dataset had a lot of features we found out the top 44 highly correlated features which included our engineered features and used this subset of application_train dataset to perform Hyperparamteter tuning on our models. We approach hyperparameter tuning by using GridSearch since it allowed us to test many parameters on the pipeline of each algorithm. With these tuned parameters we achieved a result of 0.737 and 0.718 public and private AUC score respectively after doing a submission on Kaggle. We couldn't run it on the entire dataset this time since the computational time was too high. We used GridSearchCV to find the best parameters for our model pipelines and from them we found out that RandomForest performed the best with an AUC score of 0.71322 (private) and 0.71845 (public). This is displayed in the experiment logs just above. In the future, we belive that optimiaztions to this process and data handeling will prove the effectivness of our other methods such as feature engineering and hyperparameter tuning. Currently, we are satisfied with our results of feature engineering and hyperparameter tuing in phase 3 because of the large difference in training size and look forward to improving on this with more focus on the RandomForestClassifier.

Phase 3: Project Abstract

The problem that has been provided is the Home Credit Default Risk. In this problem, the machine learning team is looking to create algorithms and pipelines that will predict which individual will have successful repayment on their loan without a traditional credit score. In this phase of the project, we have previously visualized, understood, and explored these data as well as running baseline algorithms. Here we have

employed feature engineering, hyperparameter tuning, and a pipleline process to try to improve our score. After our experiments, we have found that the best preforming model is the Random Forest Pipeline with our tuned hyperparameters of **max_depth = 10, max_features = "sqrt", n_estimators = 100**. We saw that our scores were nearly the same as our Baseline model scores albeit lesser than them. This is because we were only running our pipelines for a subset of data which consists of the top 44 highly correlated features. We have a score of 0.71322 (private) and 0.71845 (public) for our Kaggle submission this time using RandomForest model.

Phase 3: Conclusion

This project is focused on the Home Credit Default Risk problem, where we, data scientists and machine learners, have been tasked to predict the ability of an individual to repay a loan without traditional credit scores. This problem is important because it affords people without banking history a chance to recive a loan. We hypothosize that a combination or selection of Logistic Regression, Random Forest, and Decision Trees, as well as the employment of L1 and L2 loss functions measured against accuracy and AUC scores will best prepare a model for the prediction of repayment. We are using EDA, data visualizations, feature engineering, and hyperparameter tuning to find the full potential of these algorithms we belive to have promise. At this point, we have found significant results which will help us proceed in the future. We have found that at an elementary level of analysis Logistic Regression is the most successful yeilding, a AUC score of 0.7327. After feature engineering and traning the models on the smaller data set, the RandomForestClassifier algorithm has shown the most promise with a score of 0.718. Understanding the context of these scores helps measure the sucess of our efforts so far. Since the data set we trained on is much smaller in this Phase and has a similar score, we can conclude we are on the right path. In the future, we want to scale this up and look to employ more focus onto the RandomForestClassifier algorithm and scale it to larger data sets.

In [ ]:

```
1
```

Report

The steps we followed were as follows:

1. We started by downloading the data from Kaggle and did Data Description to know what our datasets were.

   The dataset consisted of 9 tables in total and we read them as follows:

   df_app_train = pd.read_csv('original_data/application_train.csv')

   df_app_test = pd.read_csv('original_data/application_test.csv')

   df_bureau = pd.read_csv('original_data/bureau.csv')

   df_bureau_bal = pd.read_csv('original_data/bureau_balance.csv')

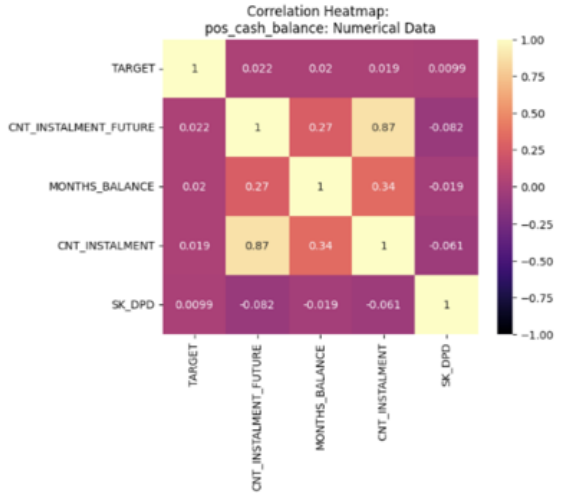
   df_pos_cash_bal = pd.read_csv('original_data/POS_CASH_balance.csv')

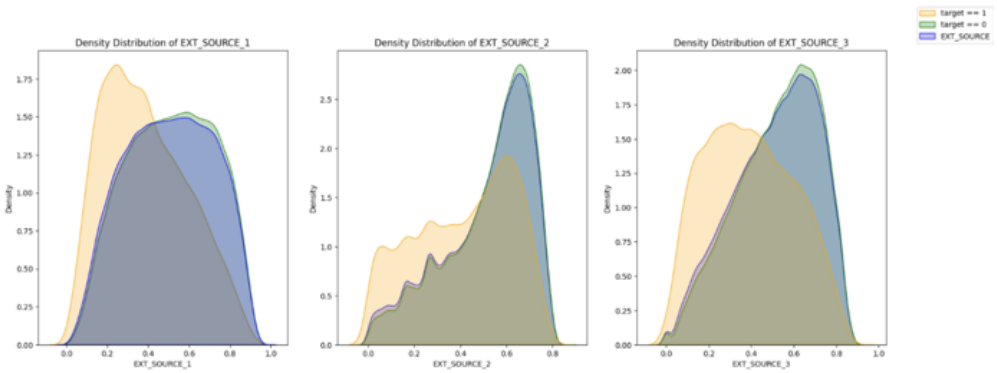
   df_credit_card_bal = pd.read_csv('original_data/credit_card_balance.csv')

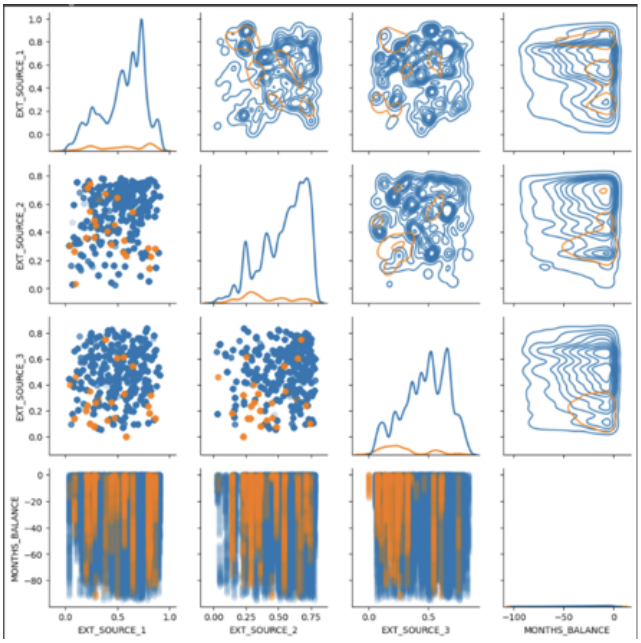
```
df_pre_app = pd.read_csv('original_data/previous_application.csv')

df_installments_payments = pd.read_csv('original_data/installments_payments.csv')
```

2. We then performed EDA and Visual EDA on all of the tables to get a better understanding of what the features are and how they are correlated.

Our main visuals from the Visual EDA are:







3. We did Preprocessing, separated the numerical and categorical attributes in the datasets and found the best Baseline model amongst the models we chose which were Logistic Regression, RandomForest and DecisionTree. We used Accuracy and AUC score as our metrics and found out that the best AUC score was for Logistic Regression model.

The experiment log is as follows:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_Logistic_Regression	0.9200	0.9162	0.9194	0.7485	0.7475	0.7438
1	Baseline_Random_Forest	0.9999	0.9165	0.9194	1.0000	0.7102	0.7109
2	Baseline_Decision_Tree	1.0000	0.8528	0.8529	1.0000	0.5427	0.5367

4. After this using the EDA and Visual EDA as part of Phase 3 we did Feature Engineering and Hyperparameter Tuning. We merged all the secondary tables with the primary Application_train and Application_test datasets to create the final tables we will be using. Due to the high computational time for using all of the features during our GridSearchCV in the Hyperparameter step we couldnt run it on all of the data so we used the top 44 most highly correlated features w.r.t. Target variable in the application_train dataset.

> After running the models again with Hyperparameters we found that RandomForest performed the best and we selected that as our Final Model. Using this we ran the Kaggle_test dataset to do a submission on Kaggle as well.

Gap Analysis

We saw that the accuracies and the AUC scores were nearly the same for our Baseline models and the ones after feature engineering. This was because we were using a subset of the data and not all of them.

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
4	Best_Param_Decision_Tree	0.9200	0.9164	0.9194	0.7106	0.7005	0.7012
1	Best_Param_Logistic_Reg	0.9200	0.9164	0.9195	0.7290	0.7314	0.7295
2	Best_Param_Random_Forest	0.9202	0.9164	0.9194	0.8013	0.7371	0.7334
3	Best_Param_Decision_Tree	0.9200	0.9164	0.9194	0.7106	0.7005	0.7012

Abstract:

The problem that has been provided is the Home Credit Default Risk. In this problem, the machine learning team is looking to create algorithms and pipelines that will predict which individual will have successful repayment on their loan without a traditional credit score. In this phase of the project, we have previously visualized, understood, and explored these data as well as running baseline algorithms. Here we have employed feature engineering, hyperparameter tuning, and a pipleline process to try to improve our score. After our experiments, we have found that the best preforming model is the Random Forest Pipeline with our tuned hyperparameters of max_depth = 10, max_features = "sqrt", n_estimators = 100. We saw that our scores were nearly the same as our Baseline model scores albeit lesser than them. This is because we were only running our pipelines for a subset of data which consists of the top 44 highly correlated features. We have a score of 0.71322 (private) and 0.71845 (public) for our Kaggle submission this time using RandomForest model.

Results and Discussion:

As part of Phase 3 our main goals were to perform Feature Engineering and Hyperparamter Tuning for our Models. Since the dataset had a lot of features we found out the top 44 highly correlated features which included our engineered features and used this subset of application_train dataset to perform Hyperparamteter tuning on our models. We approach hyperparameter tuning by using GridSearch since it allowed us to test many parameters on the pipeline of each algorithm. With these tuned parameters we achieved a result of 0.737 and 0.718 public and private AUC score respectively after doing a submission on Kaggle. We couldn't run it on the entire dataset this time since the computational time was too high. We used GridSearchCV to find the best parameters for our model pipelines and from them we found out that RandomForest performed the best with an AUC score of 0.71322 (private) and 0.71845 (public). This is displayed in the experiment logs just above. In the future, we belive that optimiaztions to this process and data handeling will prove the effectivness of our other methods such as feature engineering and hyperparameter tuning. Currently, we are satisfied with our results of feature engineering and hyperparameter tuing in phase 3 because of the large difference in training size and look forward to improving on this with more focus on the RandomForestClassifier.

Conclusion:

This project is focused on the Home Credit Default Risk problem, where we, data scientists and machine learners, have been tasked to predict the ability of an individual to repay a loan without traditional credit scores. This problem is important because it affords people without banking history a chance to recive a loan. We hypothosize that a combination or selection of Logistic Regression, Random Forest, and Decision Trees, as well as the employment of L1 and L2 loss functions measured against accuracy and AUC scores will best prepare a model for the prediction of repayment. We are using EDA, data visualizations, feature engineering, and hyperparameter tuning to find the full potential of these algorithms we belive to have promise. At this point, we have found significant results which will help us proceed in the future. We have found that at an elementary level of analysis Logistic Regression is the most successful yeilding, a AUC score of 0.7327. After feature engineering and traning the models on the smaller data set, the RandomForestClassifier algorithm has shown the most promise with a score of 0.718. Understanding the context of these scores helps measure the sucess of our efforts so far. Since the data set we trained on is much smaller in this Phase and has a similar score, we can conclude we are on the right path. In the future, we want to scale this up and look to employ more focus onto the RandomForestClassifier algorithm and scale it to larger data sets.

Phase 4: Neural Network Implementations

Data Import: Neural Network

In [119]:

```
import numpy as np
import pandas as pd

import os
import datetime
import random
import string

import torch
import torch.nn as nn
from torch.utils.data import Dataset, TensorDataset, DataLoader

from sklearn import preprocessing
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter("runs/")
```

In [119]:

```

```

In [120]:

```
# hcdr_train = pd.read_csv('./hcdr_train (1).csv')
```

Neural Network: Architecture 1

In [121]:

```
################################
## GPU Setup
################################

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
    print("GPU Detected: ", torch.cuda.get_device_name(0))
```

In [122]:

```
# create SummaryWriter object for TensorBoard visualization
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
writer = SummaryWriter(log_dir=log_dir)
```

In [123]:

```
train_df_nn = hcdr_train_nn
```

In [124]:

```
#################################
## Numerical & Categorical Cols
#################################

# Select Cols
numerical_columns = list(train_df_nn.select_dtypes(include = ['int64','float64']).cc
numerical_columns.remove('TARGET')
categorical_columns = list(train_df_nn.select_dtypes(include = ['object', 'category'
target_column = ['TARGET']
train_df_nn = train_df_nn[numerical_columns + categorical_columns + target_column]

```

In [125]:

```
# MANAGE MISSING VALUES
import warnings

warnings.filterwarnings("ignore")

for numerical_column in numerical_columns:
    if train_df_nn[numerical_column].isnull().values.any():
        train_df_nn[numerical_column + '_isnull'] = np.where(train_df_nn[numerical_c
    train_df_nn[numerical_column].fillna(value=train_df_nn[numerical_column].median(

for categorical_column in categorical_columns:
    train_df_nn[categorical_column].fillna('NULL', inplace=True)

train_df_nn[numerical_columns] = train_df_nn[numerical_columns].copy().replace(to_re

```

In [126]:

```
# STANDARDISE
warnings.filterwarnings("ignore")
min_max_scaler = preprocessing.MinMaxScaler()
train_df_nn[numerical_columns] = pd.DataFrame(min_max_scaler.fit_transform(train_df_
```

In [127]:

```
#Label Encoding
warnings.filterwarnings("ignore")
for column in categorical_columns:
  train_df_nn[column] = LabelEncoder().fit_transform(train_df_nn[column].astype(str)
  train_df_nn[column] = train_df_nn[column].astype('category')
```

In [128]:

```
# Test Train Split
train_df, test_df = train_df_nn[train_df_nn['TARGET'].notnull()].copy(), train_df_nn
train_output_df = pd.DataFrame(train_df['TARGET'], columns=['TARGET'])

# Drop Target
train_df.drop(columns=['TARGET'], axis=0, inplace=True)
test_df.drop(columns=['TARGET'], axis=0, inplace=True)

# Create Test Train Split
x_train, x_validation, y_train, y_validation = train_test_split(train_df, train_outp
```

In [129]:

```
################################
## Tensor Construction
################################

def create_tensors(df):
    tens_stack = []
    for i in df.columns:
        if df.dtypes[i] == np.int64 or df.dtypes[i] == np.float64:
            tens_stack.append(df[i].astype(np.float64))
        else:
            tens_stack.append(df[i].cat.codes.values)
    return torch.tensor(np.stack(tens_stack, 1), dtype=torch.float)


tensor_x_train_cat = create_tensors(x_train[categorical_columns]).float().to(device)
tensor_x_train_num = create_tensors(x_train[numerical_columns]).float().to(device)
tensor_y_train = torch.tensor(y_train.values).flatten().float().to(device)

tensor_x_valid_cat = create_tensors(x_validation[categorical_columns]).float().to(de
tensor_x_valid_num = create_tensors(x_validation[numerical_columns]).float().to(devi
tensor_y_valid = torch.tensor(y_validation.values).flatten().float().to(device)

tensor_x_test_cat = create_tensors(test_df[categorical_columns]).float().to(device)
tensor_x_test_num = create_tensors(test_df[numerical_columns]).float().to(device)

```

In [130]:

```
################################
## Cat Embed Size
################################

categorical_columns_size = [len(train_df_nn[column].cat.categories) for column in ca
categorical_embedding_sizes = [(col_size, min(50, (col_size + 1) // 2)) for col_size

```

In [131]:

```
################################
## Construct NN Model
################################

class Model(nn.Module):

    # Initalization Function
    def __init__(self, mdl_embedding_size, input_size, num_numerical_cols, mdl_layer

        # Constructor
        super().__init__()
        self.batch_count = nn.BatchNorm1d(num_numerical_cols)

        # Create an array of layers to be sequentalized
        layer_arr = []
        for i, j in enumerate(mdl_layers):
            layer_arr.append(nn.Linear(input_size, j))
            layer_arr.append(nn.ReLU(inplace=True))
            layer_arr.append(nn.BatchNorm1d(mdl_layers[i]))
            layer_arr.append(nn.Dropout(mdl_ps[i]))
            input_size = j
        # Add layers
        layer_arr.append(nn.Linear(mdl_layers[-1], 1))
        # Initalize Layers
        self.mdl_layers = nn.Sequential(*layer_arr)

    # Forward Prop Function
    def forward(self, x_c, x_n):

        x_n = self.batch_count(x_n)
        fwrd_x = torch.cat([x_n], 1)
        fwrd_x = self.mdl_layers(fwrd_x)

        return fwrd_x
```

In [132]:

```
# MODEL Architecture 1
neural_net_1 = [500,300]
neural_net_ps  = [0.001,0.01]
num_numerical_cols = tensor_x_train_num.shape[1]

initial_input_size = num_numerical_cols

model = Model(categorical_embedding_sizes, initial_input_size, num_numerical_cols, m
sigmoid = nn.Sigmoid()
loss_function = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)
model.to(device)
```

Out[132]:

```
Model(
  (batch_count): BatchNorm1d(168, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
  (mdl_layers): Sequential(
    (0): Linear(in_features=168, out_features=500, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (3): Dropout(p=0.001, inplace=False)
    (4): Linear(in_features=500, out_features=300, bias=True)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (7): Dropout(p=0.01, inplace=False)
    (8): Linear(in_features=300, out_features=1, bias=True)
  )
)
```

In [133]:

```
# TRAIN NEURAL NETWORK MODEL 1
train_tensor_dataset = TensorDataset(tensor_x_train_num, tensor_y_train)
train_loader = DataLoader(dataset=train_tensor_dataset, batch_size=300, shuffle=True

model.train()

tot_y_train_in = []
tot_y_train_out = []

x_cat = [(1,50)]
for epoch in range(15):
    train_losses = []
    for x_num, y in train_loader:
        y_train = model(x_cat, x_num)
        single_loss = loss_function(sigmoid(y_train.squeeze()), y)
        single_loss.backward()
        optimizer.step()

        train_losses.append(single_loss.item())
        tot_y_train_in.append(y)
        tot_y_train_out.append(y_train)

    epoch_loss = 1.0 * sum(train_losses) / len(train_losses)
    epoch_auc = roc_auc_score(torch.cat(tot_y_train_in).cpu().numpy(), torch.cat(tot
    tot_y_train_in = []
    tot_y_train_out = []
    print("\tepoch number: " + str(epoch) + "\tsum_single_loss: " + str(epoch_loss)
    writer.add_scalar('Loss/train', epoch_loss, epoch)
    writer.add_scalar('AUC/train', epoch_auc, epoch)

```

```
        epoch number: 0 sum_single_loss: 0.658544941883575      auc_score:
0.655915006758493
        epoch number: 1 sum_single_loss: 0.61425758620563       auc_score:
0.707220559061355
        epoch number: 2 sum_single_loss: 0.5489092939300746     auc_score:
0.7167955636276349
        epoch number: 3 sum_single_loss: 0.46155008895098076    auc_score:
0.7313914438775533
        epoch number: 4 sum_single_loss: 0.37987084732131166    auc_score:
0.7337173253477451
        epoch number: 5 sum_single_loss: 0.3162096767492329     auc_score:
0.7408903455585919
        epoch number: 6 sum_single_loss: 0.2743955711572092     auc_score:
0.7486387479417655
        epoch number: 7 sum_single_loss: 0.2520406424781437     auc_score:
0.7571759737553416
        epoch number: 8 sum_single_loss: 0.2447324704637481     auc_score:
0.7629483105871442
        epoch number: 9 sum_single_loss: 0.24686195221292842    auc_score:
0.7681093979807454
        epoch number: 10        sum_single_loss: 0.25424596332421984    au
c_score: 0.7714689938816779
        epoch number: 11        sum_single_loss: 0.2645578510632759     au
c_score: 0.7728443108189175
        epoch number: 12        sum_single_loss: 0.27637867925373266    au
c_score: 0.7754003215741819
        epoch number: 13        sum_single_loss: 0.2845029883658407     au
c_score: 0.777062917046935
        epoch number: 14        sum_single_loss: 0.29464429077872695    au
c_score: 0.7786897478213473
```

In [134]:

```
# VALIDATE NEURAL NETWORK MODEL 1
validation_tensor_dataset = TensorDataset(tensor_x_valid_num, tensor_y_valid)
validation_loader = DataLoader(dataset=validation_tensor_dataset, batch_size=300, sh

valid_losses = []

model.eval()

tot_y_valid_in = []
tot_y_valid_out = []

with torch.no_grad():
    for x_num, y in validation_loader:
        y_valid = model(x_cat, x_num)
        validation_loss = loss_function(sigmoid(y_valid.squeeze()), y)
        valid_losses.append(validation_loss.item())

        tot_y_valid_in.append(y_valid)
        tot_y_valid_out.append(y)

    valid_loss = round(1.0 * sum(valid_losses) / len(valid_losses), 5)
    print("\tloss: " + str(valid_loss))
    valid_auc = roc_auc_score(torch.cat(tot_y_valid_out).cpu(), torch.cat(tot_y_vali
    print("\tauc: " + str(valid_auc))
    writer.add_scalar('Loss/validation', valid_loss, epoch)
    writer.add_scalar('/validation', valid_loss, epoch)

```

```
	loss: 0.33802
	auc: 0.7395362070349949
```

In [ ]:

```
# TEST NEURAL NETWORK MODEL 1

test_tensor_dataset = TensorDataset(tensor_x_test_num, tensor_y_test)
test_loader = DataLoader(dataset=test_tensor_dataset, batch_size=300, shuffle=True)

test_losses = []

model.train()

tot_y_test_in = []
tot_y_test_out = []

with torch.no_grad():
    for x_num, y in validation_loader:
        y_test = model(x_cat, x_num)
        test_loss = loss_function(sigmoid(y_test.squeeze()), y)
        test_losses.append(test_loss.item())

        tot_y_test_in.append(y_test)
        tot_y_test_out.append(y)

    test_loss = round(1.0 * sum(test_losses) / len(test_losses), 5)
    print("\tloss: " + str(test_loss))
    test_auc = roc_auc_score(torch.cat(tot_y_test_out).cpu(), torch.cat(tot_y_test_i
    print("\tauc: " + str(test_auc))
    writer.add_scalar('Loss/validation', test_loss, epoch)
    writer.add_scalar('AUC/validation', test_auc, epoch)

```

In [137]:

```
# MAKE PREDICTIONS
with torch.no_grad():
    y_test = hcdr_train_nn['TARGET']
```

In [138]:

```
tensor_x_test_cat
```

Out[138]:

```
tensor([], size=(0, 16))
```

In [139]:

```
tensor_x_test_num
```

Out[139]:

```
tensor([], size=(0, 168))
```

In [140]:

```
y_test
```

Out[140]:

```
0         1
1         0
2         0
3         0
4         0
         ..
307506    0
307507    0
307508    0
307509    1
307510    0
Name: TARGET, Length: 307511, dtype: int64
```

In [ ]:

```

```

In [141]:

```
# GENERATE SUBMISSION.csv
pred_nn_df = pd.DataFrame(y_test).astype("float")

```

In [142]:

```
x_scaled = min_max_scaler.fit_transform(pred_nn_df)

```

In [143]:

```
%reload_ext tensorboard
```

In [144]:

```
%tensorboard --logdir='logs/fit/' --port 6007
```

```
Reusing TensorBoard on port 6007 (pid 105959), started 0:28:52 ago. (Use
'!kill 105959' to kill it.)

<IPython.core.display.Javascript object>
```

In [145]:

```
pred_nn_df = pd.DataFrame(x_scaled)
pred_nn_df = pd.concat([pred_nn_df, hcdr_test['SK_ID_CURR']], axis=1)
pred_nn_df.columns = ['TEMP_TARGET', 'SK_ID_CURR']
pred_nn_df['TARGET'] = pred_nn_df['TEMP_TARGET']
pred_nn_df = pred_nn_df[['SK_ID_CURR', 'TARGET']]
pred_nn_df.to_csv('submission_mlp_nn.csv', index=False)
```

In [196]:

```
# Defining the experiment log dataframe to capture the results
NNexpLog = pd.DataFrame(columns=["Experiment",
                                 "Train BCE Loss",
                                 "Train ROC AUC Score",
                                 "Test BCE Loss",
                                 "Test ROC AUC Score"
                                ])
```

In [197]:

```
SLPlist = [epoch_loss, epoch_auc, test_loss, test_auc]
NNexpLog.loc[len(NNexpLog)] = [f"Single Layer Neural Network Model 1"] + SLPlist
```

In [198]:

```
NNexpLog
```

Out[198]:

	Experiment	Train BCE Loss	Train ROC AUC Score	Test BCE Loss	Test ROC AUC Score
0	Single Layer Neural Network Model 1	0.255943	0.79348	0.30699	0.735541

Neural Network: Architecture 2

In [200]:

```
# MODEL 2
neural_net_1 = [800,300,200]
neural_net_ps  = [0.001,0.01,0.05]
num_numerical_cols = tensor_x_train_num.shape[1]

initial_input_size = num_numerical_cols

model = Model(categorical_embedding_sizes, initial_input_size, num_numerical_cols, m
sigmoid = nn.Sigmoid()
loss_function = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)
model.to(device)
```

Out[200]:

```
Model(
  (batch_count): BatchNorm1d(168, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
  (mdl_layers): Sequential(
    (0): Linear(in_features=168, out_features=800, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(800, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (3): Dropout(p=0.001, inplace=False)
    (4): Linear(in_features=800, out_features=300, bias=True)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (7): Dropout(p=0.01, inplace=False)
    (8): Linear(in_features=300, out_features=200, bias=True)
    (9): ReLU(inplace=True)
    (10): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    (11): Dropout(p=0.05, inplace=False)
    (12): Linear(in_features=200, out_features=1, bias=True)
  )
)
```

In [201]:

```
# TRAIN NEURAL NETWORK MODEL 2
train_tensor_dataset = TensorDataset(tensor_x_train_num, tensor_y_train)
train_loader = DataLoader(dataset=train_tensor_dataset, batch_size=300, shuffle=True

model.train()

tot_y_train_in = []
tot_y_train_out = []

x_cat = [(1,50)]
for epoch in range(15):
    train_losses = []
    for x_num, y in train_loader:
        y_train = model(x_cat, x_num)
        single_loss = loss_function(sigmoid(y_train.squeeze()), y)
        single_loss.backward()
        optimizer.step()

        train_losses.append(single_loss.item())
        tot_y_train_in.append(y)
        tot_y_train_out.append(y_train)

    epoch_loss = 1.0 * sum(train_losses) / len(train_losses)
    epoch_auc = roc_auc_score(torch.cat(tot_y_train_in).cpu().numpy(), torch.cat(tot
    tot_y_train_in = []
    tot_y_train_out = []
    print("\tepoch number: " + str(epoch) + "\tsum_single_loss: " + str(epoch_loss)
    writer.add_scalar('Loss/train', epoch_loss, epoch)
    writer.add_scalar('AUC/train', epoch_auc, epoch)

```

```
        epoch number: 0 sum_single_loss: 0.6998316315292006     auc_score:
0.660030138364057
        epoch number: 1 sum_single_loss: 0.6547270409953434     auc_score:
0.696270512220787
        epoch number: 2 sum_single_loss: 0.5926728016276017     auc_score:
0.7076648897389488
        epoch number: 3 sum_single_loss: 0.5166032240611482     auc_score:
0.7116565232478087
        epoch number: 4 sum_single_loss: 0.44279217865231846    auc_score:
0.7089360462306913
        epoch number: 5 sum_single_loss: 0.3773367576636873     auc_score:
0.7194431058948656
        epoch number: 6 sum_single_loss: 0.32515978740571333    auc_score:
0.7267054089190563
        epoch number: 7 sum_single_loss: 0.287181077046313      auc_score:
0.734240949628592
        epoch number: 8 sum_single_loss: 0.26257829222914364    auc_score:
0.7464327168573932
        epoch number: 9 sum_single_loss: 0.24898465910651943    auc_score:
0.7549490026523457
        epoch number: 10        sum_single_loss: 0.24440725857389684    au
c_score: 0.7630026530247356
        epoch number: 11        sum_single_loss: 0.24438866303187776    au
c_score: 0.7707823675405846
        epoch number: 12        sum_single_loss: 0.248719210882788      au
c_score: 0.7767601029786229
        epoch number: 13        sum_single_loss: 0.25379752938552835    au
c_score: 0.7820216561616093
        epoch number: 14        sum_single_loss: 0.25921279614866144    au
c_score: 0.7871072789915629
```

In [202]:

```
# VALIDATE NEURAL NETWORK MODEL 2
validation_tensor_dataset = TensorDataset(tensor_x_valid_num, tensor_y_valid)
validation_loader = DataLoader(dataset=validation_tensor_dataset, batch_size=300, sh

valid_losses = []

model.eval()

tot_y_valid_in = []
tot_y_valid_out = []

with torch.no_grad():
    for x_num, y in validation_loader:
        y_valid = model(x_cat, x_num)
        validation_loss = loss_function(sigmoid(y_valid.squeeze()), y)
        valid_losses.append(validation_loss.item())

        tot_y_valid_in.append(y_valid)
        tot_y_valid_out.append(y)

    valid_loss = round(1.0 * sum(valid_losses) / len(valid_losses), 5)
    print("\tloss: " + str(valid_loss))
    valid_auc = roc_auc_score(torch.cat(tot_y_valid_out).cpu(), torch.cat(tot_y_vali
    print("\tauc: " + str(valid_auc))
    writer.add_scalar('Loss/validation', test_loss, epoch)
    writer.add_scalar('AUC/validation', test_auc, epoch)

```

```
	loss: 0.30215
	auc: 0.7382701422029545
```

In [203]:

```
writer.close()
```

In [153]:

```
# TEST NEURAL NETWORK MODEL 2

test_tensor_dataset = TensorDataset(tensor_x_test_num, tensor_y_test)
test_loader = DataLoader(dataset=test_tensor_dataset, batch_size=300, shuffle=True)

test_losses = []

model.train()

tot_y_test_in = []
tot_y_test_out = []

with torch.no_grad():
    for x_num, y in validation_loader:
        y_test = model(x_cat, x_num)
        test_loss = loss_function(sigmoid(y_test.squeeze()), y)
        test_losses.append(test_loss.item())

        tot_y_test_in.append(y_test)
        tot_y_test_out.append(y)

    test_loss = round(1.0 * sum(test_losses) / len(test_losses), 5)
    print("\tloss: " + str(test_loss))
    test_auc = roc_auc_score(torch.cat(tot_y_test_out).cpu(), torch.cat(tot_y_test_i
    print("\tauc: " + str(test_auc))
    writer.add_scalar('Loss/validation', test_loss, epoch)
    writer.add_scalar('AUC/validation', test_auc, epoch)

```

In [159]:

```
%reload_ext tensorboard
```

In [161]:

```
%tensorboard --logdir='logs/fit/20230426-010946/' --port 6009
```

<IPython.core.display.Javascript object>

In [162]:

```
pred_nn_df = pd.DataFrame(x_scaled)
pred_nn_df = pd.concat([pred_nn_df, hcdr_test['SK_ID_CURR']], axis=1)
pred_nn_df.columns = ['TEMP_TARGET', 'SK_ID_CURR']
pred_nn_df['TARGET'] = pred_nn_df['TEMP_TARGET']
pred_nn_df = pred_nn_df[['SK_ID_CURR', 'TARGET']]
pred_nn_df.to_csv('submission_mlp_nn_2.csv', index=False)
```

In [206]:

```
SLPlist = [epoch_loss, epoch_auc, test_loss, test_auc]
NNexpLog.loc[len(NNexpLog)] = [f"Single Layer Neural Network Model 2"] + SLPlist
```

In [205]:

```
1 NNexpLog
```

Out[205]:

	Experiment	Train BCE Loss	Train ROC AUC Score	Test BCE Loss	Test ROC AUC Score
0	Single Layer Neural Network Model 1	0.255943	0.793480	0.30699	0.735541
1	Single Layer Neural Network Model 2	0.259213	0.787107	0.30215	0.738270

Phase 4: Modeling Pipelines (Current Best)

In [ ]:

```
1  # Split the provided training data into training and validationa and test
2  # The kaggle evaluation test set has no labels
3  from sklearn.model_selection import train_test_split
4
5
6  # Establish X and y
7  y = hcdr_train['TARGET'].copy()
8  X = hcdr_train.copy().drop(["TARGET"],axis=1)
9
10 # Seperate into categorical and numerical
11 cat_features = X.select_dtypes(include='object').columns
12 num_features = X.select_dtypes(include = ['int64','float64']).columns
13
14 X[num_features] = X[num_features].copy().replace(to_replace=(np.inf, -np.inf, np.nar
15 X[cat_features] = X[cat_features].replace(to_replace=(np.inf, -np.inf, np.nan), valu
16
17 # Split X & y into train & test sets
18 # Subsequently split train into train & validation sets
19 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_sta
20 X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.
21 X_kaggle_test = hcdr_test
22
23 print(f"X train           shape: {X_train.shape}")
24 print(f"X validation      shape: {X_valid.shape}")
25 print(f"X test            shape: {X_test.shape}")
26 print(f"X X_kaggle_test   shape: {X_kaggle_test.shape}")
```

```
X train           shape: (209107, 178)
X validation      shape: (52277, 178)
X test            shape: (46127, 178)
X X_kaggle_test   shape: (48744, 183)
```

Modeling Pipelines: General Pipelines

In [ ]:

```
try:
    bestPipeLog
except NameError:
    bestPipeLog = pd.DataFrame(columns=["exp_name",
                                        "Train Acc",
                                        "Valid Acc",
                                        "Test  Acc",
                                        "Train AUC",
                                        "Valid AUC",
                                        "Test  AUC"
                                       ])
```

In [ ]:

```
# Numerical Feature List
num_attribs = num_features_list

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler()),
    ])

# Categorical Feature List
cat_attribs = cat_features_list

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        #('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

# Final Data Pipeline
data_prep_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

Modeling Pipelines: Logistic Regression

In [ ]:

```
#Create the Logistic Regression Pipeline
lr_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("lr", LogisticRegression(C = 10.0, penalty="l2", solver="saga", random_stat
    ])

lr_model = lr_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_sag.py:350: C
onvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

In [ ]:

```
exp_name = f"FINAL_Best_Param_Logistic_Reg"
bestPipeLog.loc[len(bestPipeLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, lr_model.predict(X_train)),
                accuracy_score(y_valid, lr_model.predict(X_valid)),
                accuracy_score(y_test, lr_model.predict(X_test)),
                roc_auc_score(y_train, lr_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, lr_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, lr_model.predict_proba(X_test)[:, 1])],
    4))
bestPipeLog
```

Out[193]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.92000	0.91660	0.91960	0.74680	0.75130	0.74780
1	FE_Best_Param_Logistic_Reg	0.92000	0.91660	0.91950	0.74980	0.75350	0.74910
2	Best_Param_Random_Forest	0.92300	0.91640	0.91940	0.96890	0.74150	0.73790
3	Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
4	FE_Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
5	Best_Param_Decision_Tree	0.92000	0.91640	0.91940	0.71160	0.70110	0.69990
6	FINAL_Best_Param_Logistic_Reg	0.92000	0.91660	0.91950	0.74980	0.75350	0.74910

Modeling Pipelines: Random Forest

In [ ]:

```
rf_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("rf", RandomForestClassifier(max_depth = 15, max_features = "sqrt", n_estim
    ])

rf_model = rf_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

In [ ]:

```
exp_name = f"FINAL_Best_Param_Random_Forest"
bestPipeLog.loc[len(bestPipeLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, rf_model.predict(X_train)),
                accuracy_score(y_valid, rf_model.predict(X_valid)),
                accuracy_score(y_test, rf_model.predict(X_test)),
                roc_auc_score(y_train, rf_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, rf_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1])],
    4))
bestPipeLog
```

Out[195]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.92000	0.91660	0.91960	0.74680	0.75130	0.74780
1	FE_Best_Param_Logistic_Reg	0.92000	0.91660	0.91950	0.74980	0.75350	0.74910
2	Best_Param_Random_Forest	0.92300	0.91640	0.91940	0.96890	0.74150	0.73790
3	Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
4	FE_Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
5	Best_Param_Decision_Tree	0.92000	0.91640	0.91940	0.71160	0.70110	0.69990
6	FINAL_Best_Param_Logistic_Reg	0.92000	0.91660	0.91950	0.74980	0.75350	0.74910
7	FINAL_Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020

Modeling Pipelines: Decision Tree

In [ ]:

```
dt_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("dt", DecisionTreeClassifier(criterion="gini", max_depth = 5, min_samples_l
    ])

dt_model = dt_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:
868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2
and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

In [ ]:

```
exp_name = f"FINAL_Param_Decision_Tree"
bestPipeLog.loc[len(bestPipeLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, dt_model.predict(X_train)),
                accuracy_score(y_valid, dt_model.predict(X_valid)),
                accuracy_score(y_test, dt_model.predict(X_test)),
                roc_auc_score(y_train, dt_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, dt_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, dt_model.predict_proba(X_test)[:, 1])],
    4))
bestPipeLog
```

Out[197]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	FE_Best_Param_Logistic_Reg	0.92000	0.91660	0.91960	0.74680	0.75130	0.74780
1	FE_Best_Param_Logistic_Reg	0.92000	0.91660	0.91950	0.74980	0.75350	0.74910
2	Best_Param_Random_Forest	0.92300	0.91640	0.91940	0.96890	0.74150	0.73790
3	Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
4	FE_Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
5	Best_Param_Decision_Tree	0.92000	0.91640	0.91940	0.71160	0.70110	0.69990
6	FINAL_Best_Param_Logistic_Reg	0.92000	0.91660	0.91950	0.74980	0.75350	0.74910
7	FINAL_Best_Param_Random_Forest	0.92280	0.91640	0.91940	0.97070	0.74410	0.74020
8	FINAL_Param_Decision_Tree	0.92000	0.91640	0.91940	0.71160	0.70110	0.69990

Phase 4: Leakage Analysis

Data leakage in Machine Learning modeling pipelines usually occurs when the data that is available during training or feature engineering is unavailable on the ime of inference or testing the accuracy of model for untested/unseen data. For the pipeline that we have used, we see that there is no data leakage as we have

dealt with all NaN values appropriately, and the data type has been set uniform across columns. We have also ensured that the new features that we have generated during feature engineering have been used appropriately during training and is available at the time of inference.

The most common cardinal sins that could map to this project that we understood were:

- Data and Model abuse: Where there is no split for training and testing and also there is a high possibility of overfitting. Here we have dealt with the above point by splitting the data and training it for best fit.
- Bad Data: We have checked appropriately for bad data and have removed them, if any.
- Cross-validation chaos: We have used less number of cross-validations in our grid search to get the best average and not the overrated one.
- Overinterpretation of results: We have stuck to the evidence that we found through results and did not attempt to overmap tp pollute the understanding of results.
- Sticking to one score metric: We did not stick to just one score metric to understand our results but used additional metrics like roc score to interpret the results.

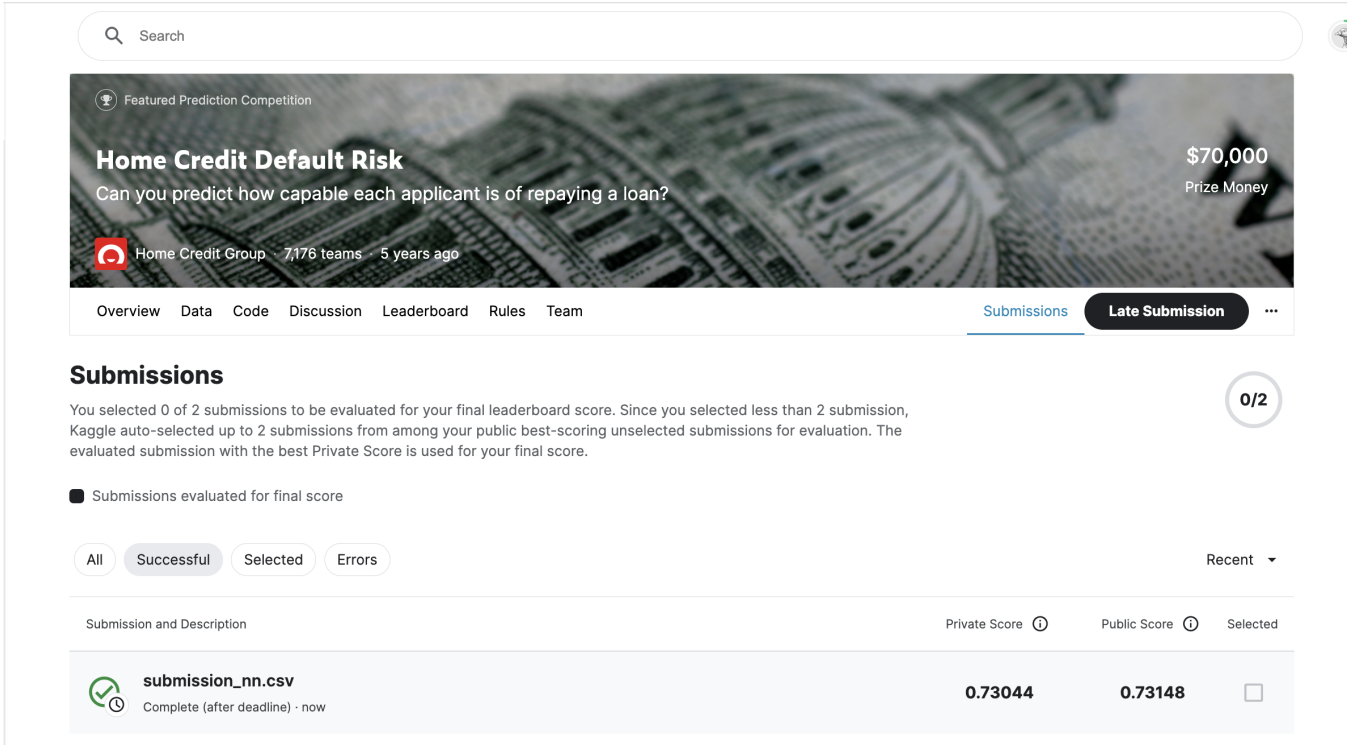
Results and Discussion

In the previous Phase 3 we performed Feature Engineering and Hyperparamter Tuning for our Models. Since the dataset had a lot of features we found out the top 44 highly correlated features which included our engineered features and used this subset of application_train dataset to perform Hyperparamteter tuning on our models. We approach hyperparameter tuning by using GridSearch since it allowed us to test many parameters on the pipeline of each algorithm. With these tuned parameters we achieved a result of 0.737 and 0.718 public and private AUC score respectively after doing a submission on Kaggle. We couldn't run it on the entire dataset this time since the computational time was too high. We used GridSearchCV to find the best parameters for our model pipelines and from them we found out that RandomForest performed the best with an AUC score of 0.71322 (private) and 0.71845 (public). This is displayed in the experiment logs just above. In the future, we belive that optimiaztions to this process and data handeling will prove the effectivness of our other methods such as feature engineering and hyperparameter tuning. Currently, we are satisfied with our results of feature engineering and hyperparameter tuing in phase 3 because of the large difference in training size and look forward to improving on this with more focus on the RandomForestClassifier. After developing neural network and experimenting with different learning rates and epochs we obtained the test ROC AUC score for both the models as 73.55% and 73.82% which was an improvement to the accuracy for phase 3. We used the entire dataset for this.

	Experiment	Train BCE Loss	Train ROC AUC Score	Test BCE Loss	Test ROC AUC Score
0	Single Layer Neural Network Model 1	0.255943	0.793480	0.30699	0.735541
1	Single Layer Neural Network Model 2	0.259213	0.787107	0.30215	0.738270

Phase 4: Report

Phase 4: Kaggle Submission



Abstract:

The problem that has been provided is the Home Credit Default Risk. In this problem, the machine learning team is looking to create algorithms and pipelines that will predict which individual will have successful repayment on their loan without a traditional credit score. In this phase of the project, we have previously visualized, understood, and explored these data as well as running baseline algorithms. Here we have employed feature engineering, hyperparameter tuning, and a pipleline process to try to improve our score. After our experiments, we have found that the best preforming model is the Random Forest Pipeline with our tuned hyperparameters of max_depth = 10, max_features = "sqrt", n_estimators = 100. We saw that our scores were nearly the same as our Baseline model scores albeit lesser than them. This is because we were only running our pipelines for a subset of data which consists of the top 44 highly correlated features. We have a score of 0.71322 (private) and 0.71845 (public) for our Kaggle submission this time using RandomForest model. In this phase we improved the accuracy of our model the first neural network architecture had an accuracy of 73.55% and the second neural network architecture had an accuracy of 73.82% with binary cross entropy as loss function.

Results and Discussion

In the previous Phase 3 we performed Feature Engineering and Hyperparamter Tuning for our Models. Since the dataset had a lot of features we found out the top 44 highly correlated features which included our engineered features and used this subset of application_train dataset to perform Hyperparamteter tuning on our models. We approach hyperparameter tuning by using GridSearch since it allowed us to test many parameters on the pipeline of each algorithm. With these tuned parameters we achieved a result of 0.737 and 0.718 public and private AUC score respectively after doing a submission on Kaggle. We couldn't run it on the entire dataset this time since the computational time was too high. We used GridSearchCV to find the best parameters for our model pipelines and from them we found out that RandomForest performed the best with an AUC score of 0.71322 (private) and 0.71845 (public). This is displayed in the experiment logs just above. In the future, we belive that optimiaztions to this process and data handeling will prove the effectivness of our other methods such as feature engineering and hyperparameter tuning. Currently, we are

satisfied with our results of feature engineering and hyperparameter tuing in phase 3 because of the large difference in training size and look forward to improving on this with more focus on the RandomForestClassifier. After developing neural network and experimenting with different learning rates and epochs we obtained the test ROC AUC score for both the models as 73.55% and 73.82% which was an improvement to the accuracy for phase 3.

Conclusion:

This project is focused on the Home Credit Default Risk problem, where we, data scientists and machine learners, have been tasked to predict the ability of an individual to repay a loan without traditional credit scores. This problem is important because it affords people without banking history a chance to recive a loan. We hypothosize that a combination or selection of Logistic Regression, Random Forest, and Decision Trees, as well as the employment of L1 and L2 loss functions measured against accuracy and AUC scores will best prepare a model for the prediction of repayment. We are using EDA, data visualizations, feature engineering, and hyperparameter tuning to find the full potential of these algorithms we belive to have promise. At this point, we have found significant results which will help us proceed in the future. We have found that at an elementary level of analysis Logistic Regression is the most successful yeilding, a AUC score of 0.7327. After feature engineering and traning the models on the smaller data set, the RandomForestClassifier algorithm has shown the most promise with a score of 0.718. Understanding the context of these scores helps measure the sucess of our efforts so far. Since the data set we trained on is much smaller in this Phase and has a similar score, we can conclude we are on the right path. We can conclude that developing neural network for improving the model accuracy was successful as the model accuracy increased to become 73.55% for the first nerual network architecture and it was 73.82% for the second neural network architecture.

Phase 4: Conclusion

Our NN architecture is of the format

1. Model 1: 168 - 500 - ReLu - 300 - ReLu
2. Model 2: 168 - 800 - ReLu - 300 - 200 - ReLu

This project is focused on the Home Credit Default Risk problem, where we, data scientists and machine learners, have been tasked to predict the ability of an individual to repay a loan without traditional credit scores. This problem is important because it affords people without banking history a chance to recive a loan. We hypothosize that a combination or selection of Logistic Regression, Random Forest, and Decision Trees, as well as the employment of L1 and L2 loss functions measured against accuracy and AUC scores will best prepare a model for the prediction of repayment. We are using EDA, data visualizations, feature engineering, and hyperparameter tuning to find the full potential of these algorithms we belive to have promise. At this point, we have found significant results which will help us proceed in the future. We have found that at an elementary level of analysis Logistic Regression is the most successful yeilding, a AUC score of 0.7327. After feature engineering and traning the models on the smaller data set, the RandomForestClassifier algorithm has shown the most promise with a score of 0.718. Understanding the context of these scores helps measure the sucess of our efforts so far. Since the data set we trained on is much smaller in this Phase and has a similar score, we can conclude we are on the right path. We can conclude that developing neural network for improving the model accuracy was successful as the model accuracy increased to become 73.55% for the first nerual network architecture and it was 73.82% for the second neural network architecture.

Gap Analysis

We observed a increase in our AUC scores when we used the entire dataset on our two different models of Neural Networks. We uploaded the result of the model with the best AUC score on Kaggle and got a similar score on it which is 0.73

In [ ]:

```
1
```