Credit Risk Analysis Machine Learning - Hackathon

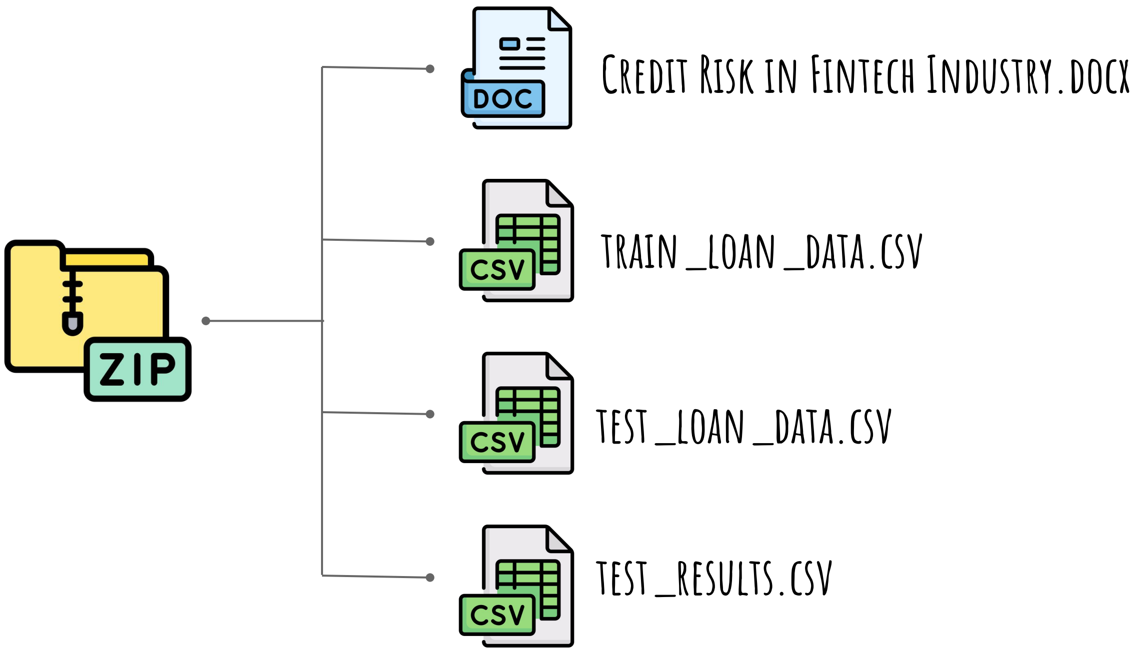
Meaga Varsha Ramakrishnan

TOC

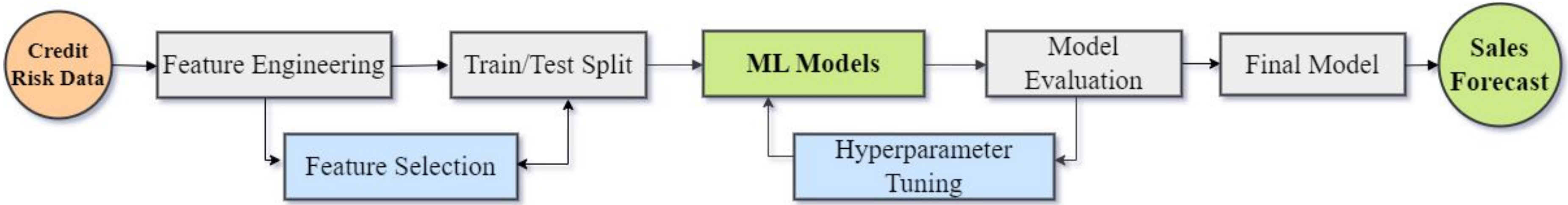
- Introduction
- Dataset Information
- Methodology
- Modeling Platform
- Exploratory Data Analysis and Pre-processing
- Feature Engineering
- Modeling and Discussions
- Model Summary - (attached google sheet link)
- Conclusion

INTRODUCTION

Dataset information - SOURCE



METHODOLOGY



MODELING PLATFORM

- Platform : Google Colab
- Runtime Type : TPU

EXPLORATORY DATA ANALYSIS

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
addr_state	80000	51	CA	11744	NaN	NaN	NaN	NaN	NaN	NaN	NaN
annual_inc	80000.0	NaN	NaN	NaN	76046.143138	69020.055377	0.0	46000.0	65000.0	90000.0	7141778.0
earliest_cr_line	80000	640	Sep-2003	547	NaN	NaN	NaN	NaN	NaN	NaN	NaN
emp_length	75412	11	10+ years	26278	NaN	NaN	NaN	NaN	NaN	NaN	NaN
emp_title	74982	36661	Teacher	1278	NaN	NaN	NaN	NaN	NaN	NaN	NaN
fico_range_high	80000.0	NaN	NaN	NaN	699.987975	31.73484	664.0	674.0	694.0	714.0	850.0
fico_range_low	80000.0	NaN	NaN	NaN	695.987813	31.734075	660.0	670.0	690.0	710.0	845.0
grade	80000	7	B	23502	NaN	NaN	NaN	NaN	NaN	NaN	NaN
home_ownership	80000	6	MORTGAGE	39628	NaN	NaN	NaN	NaN	NaN	NaN	NaN
application_type	80000	2	Individual	78446	NaN	NaN	NaN	NaN	NaN	NaN	NaN
initial_list_status	80000	2	w	46745	NaN	NaN	NaN	NaN	NaN	NaN	NaN
int_rate	80000.0	NaN	NaN	NaN	13.232898	4.771705	5.31	9.75	12.74	15.99	30.99
loan_amnt	80000.0	NaN	NaN	NaN	14403.867813	8703.826298	750.0	7925.0	12000.0	20000.0	40000.0
num_actv_bc_tl	76052.0	NaN	NaN	NaN	3.63379	2.262505	0.0	2.0	3.0	5.0	32.0
mort_acc	77229.0	NaN	NaN	NaN	1.674759	2.005104	0.0	0.0	1.0	3.0	32.0
tot_cur_bal	76052.0	NaN	NaN	NaN	141586.358991	159371.366632	0.0	29642.0	81000.5	211027.25	5172185.0
open_acc	80000.0	NaN	NaN	NaN	11.605675	5.483362	1.0	8.0	11.0	14.0	80.0
pub_rec	80000.0	NaN	NaN	NaN	0.216675	0.579854	0.0	0.0	0.0	0.0	24.0
pub_rec_bankruptcies	79969.0	NaN	NaN	NaN	0.137103	0.383202	0.0	0.0	0.0	0.0	7.0
purpose	80000	14	debt_consolidation	46418	NaN	NaN	NaN	NaN	NaN	NaN	NaN
revol_bal	80000.0	NaN	NaN	NaN	16289.340975	22649.147472	0.0	5965.75	11111.0	19635.0	1023940.0
revol_util	79947.0	NaN	NaN	NaN	51.899142	24.504836	0.0	33.5	52.2	70.8	152.6
sub_grade	80000	35	C1	4982	NaN	NaN	NaN	NaN	NaN	NaN	NaN
term	80000	2	36 months	60750	NaN	NaN	NaN	NaN	NaN	NaN	NaN
title	79030	5349	Debt consolidation	39396	NaN	NaN	NaN	NaN	NaN	NaN	NaN
total_acc	80000.0	NaN	NaN	NaN	25.036875	12.009194	2.0	16.0	23.0	32.0	162.0
verification_status	80000	3	Source Verified	30855	NaN	NaN	NaN	NaN	NaN	NaN	NaN
loan_status	80000	2	Fully Paid	64030	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
addr_state	20000	50	CA	2865	NaN	NaN	NaN	NaN	NaN	NaN	NaN
annual_inc	20000.0	NaN	NaN	NaN	76497.649333	85680.966779	0.0	45000.0	65000.0	90000.0	9522972.0
earliest_cr_line	20000	568	Oct-2001	160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
emp_length	18742	11	10+ years	6579	NaN	NaN	NaN	NaN	NaN	NaN	NaN
emp_title	18622	11180	Teacher	357	NaN	NaN	NaN	NaN	NaN	NaN	NaN
fico_range_high	20000.0	NaN	NaN	NaN	700.2044	31.768558	664.0	674.0	694.0	714.0	850.0
fico_range_low	20000.0	NaN	NaN	NaN	696.20425	31.767853	660.0	670.0	690.0	710.0	845.0
grade	20000	7	B	5756	NaN	NaN	NaN	NaN	NaN	NaN	NaN
home_ownership	20000	4	MORTGAGE	9900	NaN	NaN	NaN	NaN	NaN	NaN	NaN
application_type	20000	2	Individual	19610	NaN	NaN	NaN	NaN	NaN	NaN	NaN
initial_list_status	20000	2	w	11582	NaN	NaN	NaN	NaN	NaN	NaN	NaN
int_rate	20000.0	NaN	NaN	NaN	13.259451	4.772028	5.31	9.75	12.79	16.02	30.99
loan_amnt	20000.0	NaN	NaN	NaN	14426.67125	8811.38736	1000.0	7800.0	12000.0	20000.0	40000.0
num_actv_bc_tl	18989.0	NaN	NaN	NaN	3.61741	2.220795	0.0	2.0	3.0	5.0	20.0
mort_acc	19296.0	NaN	NaN	NaN	1.66931	1.981554	0.0	0.0	1.0	3.0	19.0
tot_cur_bal	18989.0	NaN	NaN	NaN	141200.889673	155848.258592	0.0	29596.0	80707.0	210215.0	2210119.0
open_acc	20000.0	NaN	NaN	NaN	11.59345	5.507847	1.0	8.0	11.0	14.0	56.0
pub_rec	20000.0	NaN	NaN	NaN	0.208	0.568816	0.0	0.0	0.0	0.0	15.0
pub_rec_bankruptcies	19989.0	NaN	NaN	NaN	0.130722	0.374106	0.0	0.0	0.0	0.0	8.0
purpose	20000	14	debt_consolidation	11611	NaN	NaN	NaN	NaN	NaN	NaN	NaN
revol_bal	20000.0	NaN	NaN	NaN	16181.7775	21917.28208	0.0	5803.75	11051.5	19876.25	921464.0
revol_util	19987.0	NaN	NaN	NaN	51.709746	24.509718	0.0	33.2	52.2	70.6	127.6
sub_grade	20000	35	C1	1294	NaN	NaN	NaN	NaN	NaN	NaN	NaN
term	20000	2	36 months	15209	NaN	NaN	NaN	NaN	NaN	NaN	NaN
title	19753	1623	Debt consolidation	9855	NaN	NaN	NaN	NaN	NaN	NaN	NaN
total_acc	20000.0	NaN	NaN	NaN	25.0223	12.098794	2.0	16.0	23.0	32.0	107.0
verification_status	20000	3	Source Verified	7722	NaN	NaN	NaN	NaN	NaN	NaN	NaN

NULL Contributions

	ColumnName	NULL_count	Contribution(in %)
4	emp_title	5018	6.27
3	emp_length	4588	5.74
13	num_actv_bc_tl	3948	4.93
15	tot_cur_bal	3948	4.93
14	mort_acc	2771	3.46
24	title	970	1.21
21	revol_util	53	0.07
18	pub_rec_bankruptcies	31	0.04

	ColumnName	NULL_count	Contribution(in %)
4	emp_title	1378	6.89
3	emp_length	1258	6.29
13	num_actv_bc_tl	1011	5.06
15	tot_cur_bal	1011	5.06
14	mort_acc	704	3.52
24	title	247	1.23
21	revol_util	13	0.06
18	pub_rec_bankruptcies	11	0.06

Row-wise NULL contributions are also evaluated

- Train - 76 rows with 5 null values in the same row
- Test - 76 rows with 5 null values in the same row

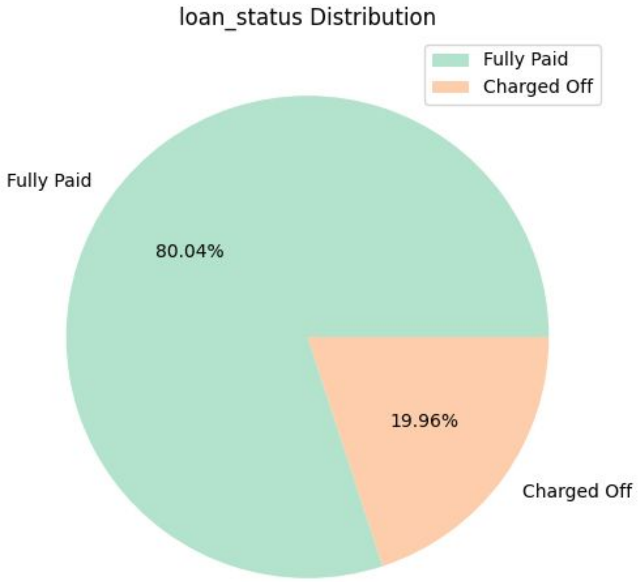
Unique Value Distribution - train

	ColumnName	sample_UniqueValues	UniqueValues_count	Unique%
3	emp_title	[Deputy, Department of Veterans Affairs, Marbl...	36662	45.83
11	title	[Debt consolidation, Credit Loan, Debt Connsol...	5350	6.69
1	earliest_cr_line	[Jul-1997, Apr-1987, Aug-2007, Sep-1980, Jul-1...	640	0.80
0	addr_state	[CO, CA, FL, IL, MD, NY, PA, WI, UT, TX, AL]	51	0.06
9	sub_grade	[E1, B1, B5, B2, F5, D3, C1, C4, B4, D4, A5]	35	0.04
8	purpose	[debt_consolidation, home_improvement, credit_...	14	0.02
2	emp_length	[10+ years, nan, 3 years, < 1 year, 1 year, 8 ...	12	0.01
4	grade	[E, B, F, D, C, A, G]	7	0.01
5	home_ownership	[MORTGAGE, RENT, OWN, ANY, NONE, OTHER]	6	0.01
12	verification_status	[Source Verified, Verified, Not Verified]	3	0.00
6	application_type	[Individual, Joint App]	2	0.00
7	initial_list_status	[w, f]	2	0.00
10	term	[60 months, 36 months]	2	0.00
13	loan_status	[Charged Off, Fully Paid]	2	0.00

Unique Value Distribution - test

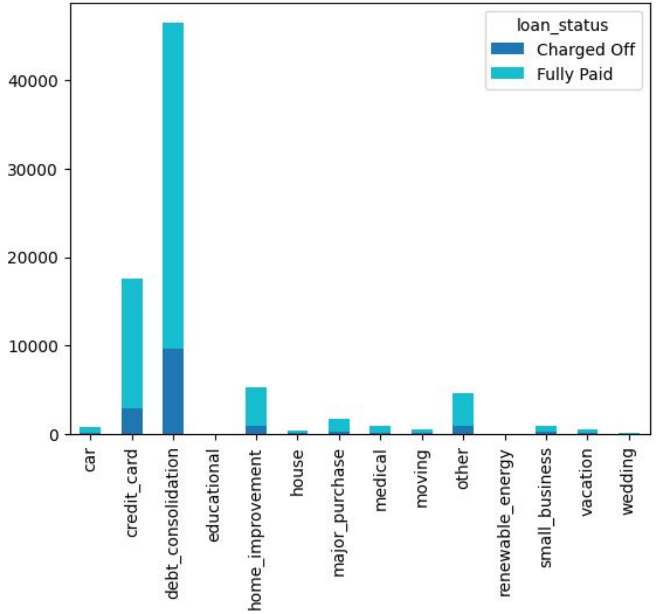
	ColumnName	sample_UniqueValues	UniqueValues_count	Unique%
3	emp_title	[Tower technician, Supervisor, APPLICATIONS PR...	11181	55.91
11	title	[Debt consolidation, Credit card refinancing, ...	1624	8.12
1	earliest_cr_line	[May-2012, Dec-2001, Mar-1989, Nov-2004, Feb-1...	568	2.84
0	addr_state	[MO, HI, TX, CA, MI, NJ, FL, GA, MD, AL, NC]	50	0.25
9	sub_grade	[C4, B2, C1, B5, A3, D3, B3, B1, E5, B4, A4]	35	0.18
8	purpose	[debt_consolidation, credit_card, home_improve...	14	0.07
2	emp_length	[1 year, 10+ years, 9 years, nan, < 1 year, 2 ...	12	0.06
4	grade	[C, B, A, D, E, F, G]	7	0.03
5	home_ownership	[OWN, RENT, MORTGAGE, ANY]	4	0.02
12	verification_status	[Source Verified, Not Verified, Verified]	3	0.01
6	application_type	[Individual, Joint App]	2	0.01
7	initial_list_status	[f, w]	2	0.01
10	term	[36 months, 60 months]	2	0.01

Unique Value Distribution - tARGET



Indicates the Dataset is IMBALANCED

Other Plots



loan_status
Charged Off
Fully Paid
40000
30000
20000
10000
0
car
credit_card
debt_consolidation
educational
home_improvement
house
major_purchase
medical
moving
other
renewable_energy
small_business
vacation
wedding

Feature Engineering

Dropping Columns - High Cardinality

The following columns are dropped as they are **Highly Cardinal** (has more Unique values)

- emp_title
- title

	ColumnName	sample_UniqueValues	UniqueValues_count	Unique%
3	emp_title	[Deputy, Department of Veterans Affairs, Marbl...	44317	44.32
11	title	[Debt consolidation, Credit Loan, Debt Connsol...	6507	6.51

	ColumnName	NULL_count	Contribution(in %)
4	emp_title	6396	6.40

KNN-IMPUTER CODE SNIPPET

```
def impute_null_values_using_knn_imputer (df, column_to_be_imputed, column_type, columns_to_be_considered, n_neighbors=3):
  knn_imputer = KNNImputer(n_neighbors=n_neighbors)
  data_to_impute = df[columns_to_be_considered+[column_to_be_imputed]]
  # print('Value Counts (before imputation) : \n',data_to_impute[columns_to_be_considered].value_counts(dropna=False))
  value_counts_df_before =
data_to_impute[column_to_be_imputed].value_counts(dropna= False).rename_axis('unique_values').reset_index(name='count_before_imputati
on').sort_values(by=['unique_values'])
  value_counts_df_before['unique_values'] = value_counts_df_before['unique_values'].fillna('NaN').astype('str')
  knn_imputer.fit(data_to_impute)
  imputed_data = knn_imputer.transform(data_to_impute)
  imputed_data = pd.DataFrame(imputed_data, columns= data_to_impute.columns)
  if column_type=='int':
    df[column_to_be_imputed] = imputed_data[column_to_be_imputed].astype( int)
  else:
    df[column_to_be_imputed] = imputed_data[column_to_be_imputed]
  # print('Null Counts (after imputation) : \n',df[columns_to_be_considered].value_counts(dropna=False))
  value_counts_df_after =
df[column_to_be_imputed].astype('float').astype('str').value_counts(dropna=False).rename_axis('unique_values').reset_index(name='cou
nt_after_imputation').sort_values(by=['unique_values'])
  value_counts_df = value_counts_df_before.merge(value_counts_df_after, on=[ 'unique_values'], how='left')
  value_counts_df['no_of_values_imputed'] = value_counts_df['count_after_imputation'] - value_counts_df['count_before_imputation']
  if column_type=='int':
    display(value_counts_df)
  return df
```

EARLIEST_CR_LINE_YEAR

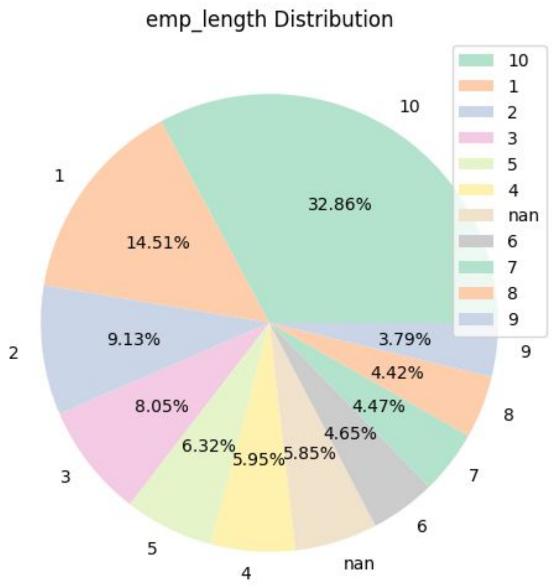
- Converting the *Month-Year* format ***earliest_cr_line*** column to ***earliest_cr_line_month*** (Month) and ***earliest_cr_line_year*** (Year) separately
- Dropping the ***earliest_cr_line*** column

```
data['earliest_cr_line_year'], data['earliest_cr_line_month'] =
pd.DatetimeIndex(data['earliest_cr_line']).year, pd.DatetimeIndex(data['earliest_cr_line']).month
data = data.drop(columns = 'earliest_cr_line')
data.head(3)
```

EMP_LENGTH

- The emp_length columns values are suffixed with ' years', so splitting the column based on this siffix.
- Replacing the following to maintain a standard format:
  - 10+ years : 10 years
  - 1 year : 1 years
  - < 1 year : 1 years

```
data['emp_length'] = data['emp_length'].replace({'10+ years': '10 years', '1 year':'1', '< 1 year':'1 years'}).str.split(" years",n=1, expand=True)[0]
```



EMP_LENGTH

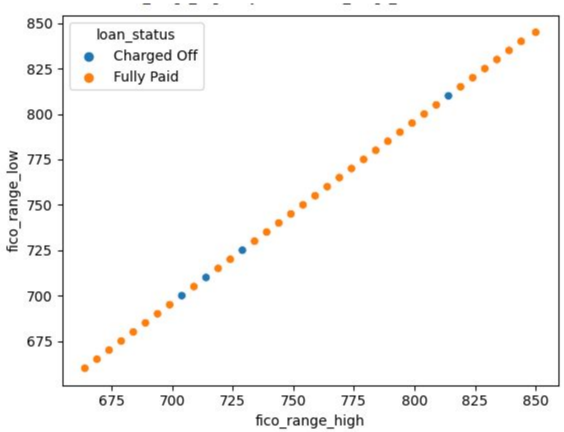
- NULL values are imputed using KNN-Imputer

```
emp_len_imputed = impute_null_values_using_knn_imputer(data,
                                                        column_to_be_imputed='emp_length',
                                                        column_type = 'int',
                                                       columns_to_be_considered=['annual_inc', 'earliest_cr_line_year', 'loan_amnt'],
                                                        n_neighbors=3)
```

FICO_RANGE_HIGH , FICO_RANGE_LOW

- There exists a straight-forward linear relationship between these two columns
- So, creating '***fico_range_average***', average column out of these and dropping these 2 columns

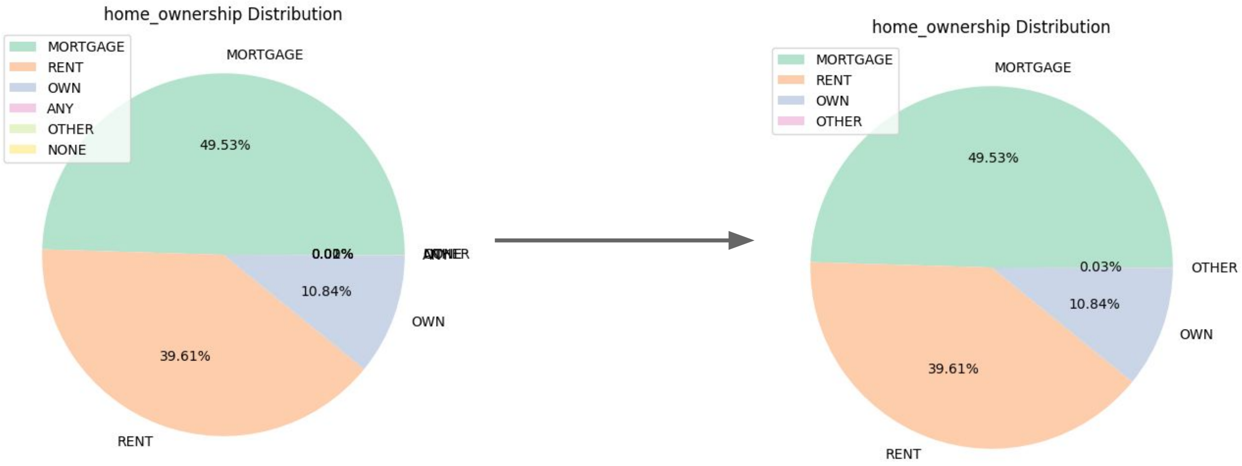
```
emp_len_imputed['fico_range_average'] =
(emp_len_imputed['fico_range_low'] +
emp_len_imputed['fico_range_high']) / 2
fico_range_avg =
emp_len_imputed.drop(columns=['fico_range_low',
'fico_range_high'])
```



HOME_OWNERSHIP

The following pre-process should be done:

- The ***NaN*** value is indicated as ***NONE***
- ***'ANY'*** and ***'OTHER'*** values are combined to ***'OTHER'***
- ***NONE*** is imputed with the *mode* value



Num_actv_bc_tl, mort_acc, tot_cur_bal, revol_bal, revol_util

- The NaN values in these columns are imputed using KNN-Imputer

```
actv_bc_tl = impute_null_values_using_knn_imputer(fico_range_avg,
                                                  column_to_be_imputed='num_actv_bc_tl',
                                                  column_type='int',
                                                  columns_to_be_considered=['loan_amnt',
'fico_range_average'],
                                                  n_neighbors=3)

mort_acc = impute_null_values_using_knn_imputer(actv_bc_tl,
                                                column_to_be_imputed='mort_acc',
                                                column_type='int',
                                                columns_to_be_considered=['loan_amnt',
'fico_range_average', 'num_actv_bc_tl'],
                                                n_neighbors=3)
```

TERM

- The following replacements are done:
    - ‘ 60 months’ - 60
    - ‘ 36 months’ - 36

```
na_filled['term'] = na_filled['term'].replace({' 60 months': 60, ' 36 months': 36}).astype(int)
na_filled['term'].dtype
```

Grade and sub_grade

- This table shows that the grade and sub_grade columns are related.
- So, updating the sub_grade column with

numeric value

```
na_filled['sub_grade'] = na_filled['sub_grade'].str[-1].astype(int)
na_filled['sub_grade'].unique()
```

grade	sub_grade
A	{A1, A3, A5, A2, A4}
B	{B4, B5, B3, B1, B2}
C	{C5, C1, C3, C4, C2}
D	{D4, D3, D2, D1, D5}
E	{E4, E3, E2, E5, E1}
F	{F2, F3, F1, F5, F4}
G	{G1, G5, G3, G4, G2}

ENCODING

- **Ordinal Encode:** *grade, initial_list_status*
- **One-Hot Encode:** *addr_state, home_ownership, application_type, purpose, verification_status*

MODELING AND DISCUSSIONS

Functions Created - Scaling, Sampling

```
def scale_data(data, scaler=StandardScaler()):
  scaler.fit(data)
  data_scaled = scaler.transform(data)
  return data_scaled

def sampling_data(X_data, y_data, sampling_type='oversampling'):
  if sampling_type=='oversampling':
    sampler = RandomOverSampler()
  else:
    sampler = RandomUnderSampler()
  X_sampled, y_sampled = sampler.fit_resample(X_data, y_data)
  return X_sampled, y_sampled
```

Functions Created - Train-Test split

```
def split_data(df, target_col, train_size=0.75, random_state=0):
    print('Train Test Split...')
    print('Target Column :', target_col)
    print(f"Train-Test Size : {train_size},{round(1-train_size, 1)}")
    X = df.drop(columns=str(target_col))
    Y = df[target_col]
    # display(X)
    return train_test_split(X, Y, train_size=train_size, stratify=Y, random_state=random_state)
```

Functions Created - Get Data for Modeling

This function fetches the relevant modeling as per the parameters(scaling, sampling, features)

```
def get_data_for_modeling(data_and_hack_test, target_col, important_features, model,scaling, sampling, sampling_type):
  data, hack_test, hack_results = data_and_hack_test
  print(target_col in hack_test.columns.tolist())
  if len(important_features)==0:
    important_features = data.columns.tolist()
    important_features_for_test_data = data.drop(columns=target_col).columns.tolist()
  else:
    list(set(important_features)).append(target_col)
  data = data[list(set(important_features))]
  hack_test = hack_test[list(set(important_features_for_test_data))]
  X_train, X_test, y_train, y_test = split_data(data,target_col,train_size=0.7 )
  print("Scaling : ", scaling)
  if scaling==True:
    X_train, X_test, hack_test = scale_data(X_train), scale_data(X_test), scale_data(hack_test)
  else:
    X_train, X_test, hack_test = X_train, X_test, hack_test
```

Functions Created - Applying Models

```
def applying_model(data_and_hack_test, target_col, base_model_name, model=None, important_features=[], scaling=False, sampling=True,
sampling_type='oversampling'):
  """ The model is passed as parameter, for GridSearchCV """
  X_train, X_test, y_train, y_test, hack_test = get_data_for_modeling(data_and_hack_test, target_col, important_features,
model,scaling, sampling, sampling_type)

  if important_features==[]:
    feature_selection = False
    print('Training the model with the best parameter...' )
    model.fit(X_train, y_train)
    print('Best Estimator : ')
    print(model.best_estimator_)
  else:
    feature_selection = True

  print('Predicting results...')
  train_pred = model.predict(X_train)
  test_pred = model.predict(X_test)
  hack_test_pred = model.predict(hack_test)
  model_accuracy_info.append([base_model_name, f'Scaling-{scaling}, Sampling-{sampling}({sampling_type}),
FeatureSelection-{feature_selection}', round(accuracy_score(y_train, train_pred), 4), round(accuracy_score(y_test, test_pred), 4),
round(accuracy_score(hack_results, hack_test_pred), 4), round(f1_score(y_train, train_pred), 4), round(f1_score(y_test, test_pred), 4),
round(f1_score(hack_results, hack_test_pred), 4)  ])
  print(model_accuracy_info[-1])
```

MODEL SUMMARY

https://drive.google.com/file/d/1jOfqEZXHhgj2zV3X5cXuhaHvRHtFYnJm/view?usp=share_link

- The evaluation metric is **F1 score**

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \cdot \frac{Precision * Recall}{Precision + Recall}$$

$$\Rightarrow F1\ score = 2 \cdot \frac{Precision * Recall}{Precision + Recall}$$

CONCLUSION

BEST MODEL

```
[280] model_summary.sort_values(by='f1score_hack_test', ascending=False)['model_name'].iloc[0]
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

F1 Score of the best model : 0.4324

THANK YOU