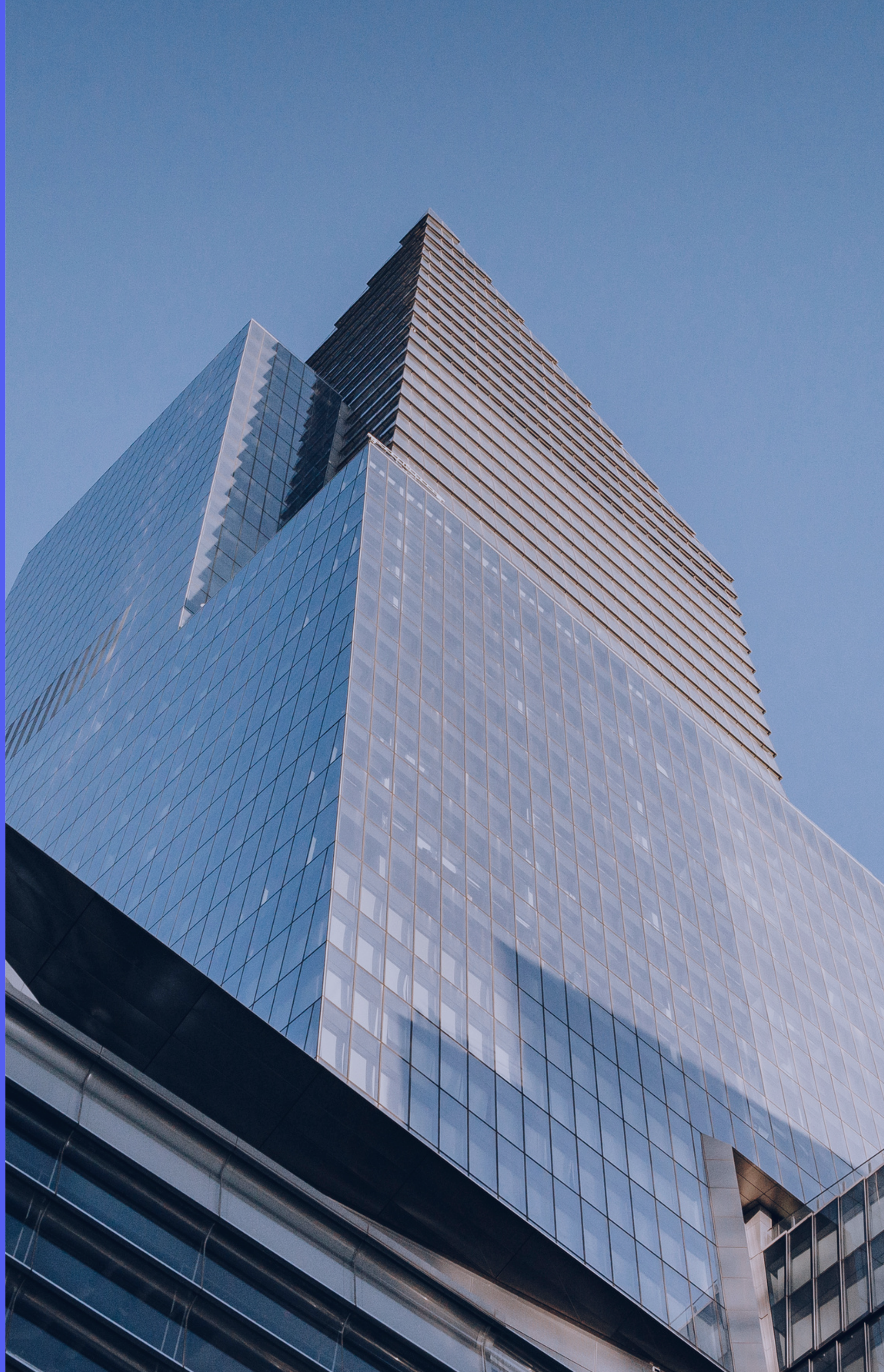
CREDIT RISK
PREDICTION

ID/X PARTNERS DATA SCIENTIST
PROJECT BASED INTERNSHIP
PROGRAM

BY:
Saskia Dwi Ulfah



PRESENTATION CONTENT

#1
Data Understanding

#2
Data Preparation

#3
Data Modeling

#4
Model Evaluation and Saving Final Model

DATA UNDERSTANDING

DATA UNDERSTANDING

```
[7] df = pd.read_csv(os.path.join(ROOT_PATH, 'loan_data_2007_2014.csv'))
```

```
df.head()
```

	Unnamed: 0	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_statu
0	0	1077501	1296599	5000	5000	4975.0	36 months	10.65	162.87	B	B2	NaN	10+ years	RENT	24000.0	Verifie
1	1	1077430	1314167	2500	2500	2500.0	60 months	15.27	59.83	C	C4	Ryder	< 1 year	RENT	30000.0	Source Verifie
2	2	1077175	1313524	2400	2400	2400.0	36 months	15.96	84.33	C	C5	NaN	10+ years	RENT	12252.0	Not Verifie
3	3	1076863	1277178	10000	10000	10000.0	36 months	13.49	339.31	C	C1	AIR RESOURCES BOARD	10+ years	RENT	49200.0	Source Verifie

In this task, we were given data. The data itself contained **75 columns and 466285 entries**. The features related to the lendee's information and their credit activities, such as employment, home ownership, loan amount, purpose of credit activity, payment plan, etc. Based on those features, we were asked to **build a model for credit risk prediction**.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 466285 entries, 0 to 466284
Data columns (total 75 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Unnamed: 0                   466285 non-null  int64
 1   id                           466285 non-null  int64
 2   member_id                    466285 non-null  int64
 3   loan_amnt                    466285 non-null  int64
 4   funded_amnt                  466285 non-null  int64
 5   funded_amnt_inv              466285 non-null  float64
 6   term                         466285 non-null  object
 7   int_rate                     466285 non-null  float64
 8   installment                  466285 non-null  float64
 9   grade                        466285 non-null  object
 10  sub_grade                    466285 non-null  object
 11  emp_title                    438697 non-null  object
 12  emp_length                   445277 non-null  object
 13  home_ownership               466285 non-null  object
 14  annual_inc                   466281 non-null  float64
 15  verification_status          466285 non-null  object
 16  issue_d                      466285 non-null  object
 17  loan_status                  466285 non-null  object
 18  pymnt_plan                   466285 non-null  object
 19  url                          466285 non-null  object
 20  desc                         125983 non-null  object
 21  purpose                      466285 non-null  object
 22  title                        466265 non-null  object
 23  zip_code                     466285 non-null  object
 24  addr_state                   466285 non-null  object
 25  dti                          466285 non-null  float64
 26  delinq_2yrs                  466256 non-null  float64
 27  earliest_cr_line             466256 non-null  object
 28  inq_last_6mths               466256 non-null  float64
 29  mths_since_last_delinq       215934 non-null  float64
 30  mths_since_last_record       62638 non-null   float64
 31  open_acc                     466256 non-null  float64
 32  pub_rec                      466256 non-null  float64
```

DATA PREPARATION

CHECK DUPLICATED ROWS

Check for duplicated records.

```
[10] df[df.duplicated()]
```

Unnamed: 0	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title

No duplicated rows.

This data **contained no duplicated rows**. Hence, every record in this data corresponded to one credit activity.

CHECK AND DROP NULL VALUES

```
all_null_columns = df.columns[df.isnull().all()]

print("Columns with all null values:")
print(all_null_columns)
```

```
Columns with all null values:
Index(['annual_inc_joint', 'dti_joint', 'verification_status_joint',
       'open_acc_6m', 'open_il_6m', 'open_il_12m', 'open_il_24m',
       'mths_since_rcnt_il', 'total_bal_il', 'il_util', 'open_rv_12m',
       'open_rv_24m', 'max_bal_bc', 'all_util', 'inq_fi', 'total_cu_tl',
       'inq_last_12m'],
      dtype='object')
```

```
[13]
# drop

df = df.drop(columns=all_null_columns)
```

```
[17]
partial_null_columns = df.columns[(df.isnull().sum()/len(df))*100 > 40]
```

```
partial_null_columns
```

```
Index(['desc', 'mths_since_last_delinq', 'mths_since_last_record',
       'next_pymnt_d', 'mths_since_last_major_derog'],
      dtype='object')
```

```
[18]
# drop

df = df.drop(columns=partial_null_columns)
```

In this phase, we checked columns whose percentage of null values was 100% or less. We **dropped columns whose percentage of null values was > 40%**.

FILL-IN NULL VALUES

```
cols_with_null = df.columns[df.isnull().any()]
```

```
null_num_cols = df[cols_with_null].select_dtypes(exclude='object').columns
```

```
null_non_num_cols = df[cols_with_null].select_dtypes(include='object').columns
```

```
for column in null_num_cols:
    median = df[column].median()
    df[column].fillna(median, inplace=True)
```

```
for column in null_non_num_cols:
    mode_value = df[column].mode()[0]
    df[column].fillna(mode_value, inplace=True)
```

For the column whose percentage of null values was <40%:

- For **numerical data**, we filled in with **median value**.
- For **non-numerical data**, we filled in with **mode value**.

DROP UNNECESSARY COLUMNS

```
unncss_cols = ['Unnamed: 0', 'id', 'member_id', 'url', 'policy_code', 'application_type']
```

```
# drop

df = df.drop(columns=unncss_cols)
```

```
df = df.drop(columns=['emp_title','title'])
```

```
df = df.drop(columns=["out_prncp", "out_prncp_inv", "recoveries", "last_pymnt_d",
                      "last_pymnt_amnt", "total_pymnt", "total_pymnt_inv",
                      "total_rec_prncp", "total_rec_int", "total_rec_late_fee",
                      "collection_recovery_fee", "funded_amnt", "funded_amnt_inv"])
```

```
df = df.drop(columns=['issue_d','last_credit_pull_d'])
```

We also dropped columns that belong to one of these categories:

- Columns with high cardinality.
- Columns with one variation of value.
- Free text column.
- Columns that require value from the future.

ENCODE NON-NUMERICAL COLUMNS

```
le_cols = ['addr_state', 'earliest_cr_line', 'emp_length', 'grade',
           'home_ownership', 'initial_list_status', 'purpose',
           'pymnt_plan', 'sub_grade', 'verification_status', 'zip_code']
```

```
for col in le_cols:
    le = LabelEncoder()
    df[col] =  le.fit_transform(df[col])
    print(le.classes_)
```

```
# Term column
df["term"] = df["term"].str.replace(" 36 months", "36")
df["term"] = df["term"].str.replace(" 60 months", "60")
df["term"] = df["term"].astype(int)
```

The computer worked with numbers. So, we needed to transform text data into the form of numbers. To do this, we utilized LabelEncoder() from sklearn.

FILTERING

```python
df = df[(df['loan_status']=='Fully Paid') | (df['loan_status']=='Charged Off')]
```

```python
df.shape
```

```
(227214, 30)
```

```python
df['loan_status'].value_counts()
```

```
Fully Paid     184739
Charged Off     42475
Name: loan_status, dtype: int64
```

```python
# Map the risk

risk_mapping = {
    'Fully Paid' : 'Low Risk',
    'Charged Off': 'High Risk'
}
```

```python
df['loan_status'] = df['loan_status'].map(risk_mapping)
```

```python
df['loan_status'].value_counts()
```

```
Low Risk     184739
High Risk     42475
Name: loan_status, dtype: int64
```

```python
# Map the risk to numerical value

num_risk_mapping = {
    'Low Risk' : 0,
    'High Risk': 1
}
```

```python
df['loan_status'] = df['loan_status'].map(num_risk_mapping)
```

```python
df['loan_status'].value_counts()
```

```
0    184739
1     42475
Name: loan_status, dtype: int64
```

In this task, we only considered “Fully Paid” and “Charged Off” categories. Those two categories, we turned into its number representation:

- Fully Paid → Low Risk → 0
- Charged Off → High Risk → 1

ASSIGN LABEL, FEATURES, AND SMOTE

```
[68]
X = df.drop(columns=['loan_status'])
y = df['loan_status']
```

```
[69]
scaler= StandardScaler()
X= scaler.fit_transform(X)
```

```
[70]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[71]
y_train.value_counts()
```

```
0    147757
1     34014
Name: loan_status, dtype: int64
```

```
[72]
print("Before oversampling: ", Counter(y_train))

smote = SMOTE()
X_train,y_train= smote.fit_resample(X_train,y_train)

print("After oversampling: ",Counter(y_train))
```

```
Before oversampling:  Counter({0: 147757, 1: 34014})
After oversampling:  Counter({0: 147757, 1: 147757})
```

- Last step, we assigned features as X and loan_status as target (y).
- Next, we standardized X values with StandardScaler() from sklearn.
- Next, we split data into 2 sets: train set and test set.
- Because we have imbalanced classes, we used SMOTE from train set. SMOTE oversampled the minority class to have the same amount as majority class.

DATA MODELING

TRAINING

```
s = setup(X_train, target=y_train, session_id = 123, use_gpu=True)
```

```
best = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	0.9063	0.9649	0.9027	0.9093	0.9060	0.8127	0.8127	52.9780
rf	Random Forest Classifier	0.8840	0.9460	0.8296	0.9308	0.8773	0.7679	0.7725	88.3580
lightgbm	Light Gradient Boosting Machine	0.8829	0.9328	0.7845	0.9768	0.8701	0.7658	0.7811	6.4750
gbc	Gradient Boosting Classifier	0.8467	0.9161	0.7893	0.8916	0.8373	0.6933	0.6979	144.7530
dt	Decision Tree Classifier	0.8049	0.8049	0.8175	0.7974	0.8073	0.6098	0.6100	7.2290
ada	Ada Boost Classifier	0.7813	0.8654	0.7513	0.7993	0.7745	0.5625	0.5636	29.7240
knn	K Neighbors Classifier	0.7547	0.8713	0.9488	0.6835	0.7946	0.5094	0.5527	52.9510
lr	Logistic Regression	0.6544	0.7113	0.6589	0.6530	0.6559	0.3087	0.3088	2.7510
ridge	Ridge Classifier	0.6522	0.0000	0.6510	0.6526	0.6518	0.3044	0.3044	0.3940
lda	Linear Discriminant Analysis	0.6522	0.7086	0.6510	0.6526	0.6518	0.3044	0.3044	1.6900
svm	SVM - Linear Kernel	0.6495	0.0000	0.6403	0.6528	0.6458	0.2990	0.2996	2.1490
nb	Naive Bayes	0.5218	0.6979	0.9657	0.5161	0.6690	0.0437	0.0858	0.7180
qda	Quadratic Discriminant Analysis	0.5207	0.7107	0.9788	0.5123	0.6717	0.0413	0.0909	1.2470
dummy	Dummy Classifier	0.5000	0.5000	0.9000	0.4500	0.6000	0.0000	0.0000	0.2380

- For the sake of simplicity, we used PyCarer which automatically trained and compared models.
- Based on the experiment, the **Extra Trees Classifier reached the best performance** amongst models.

```
best
```

```
ExtraTreesClassifier
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=100, n_jobs=-1, oob_score=False,
                     random_state=123, verbose=0, warm_start=False)
```

TRAINING

```
xt = ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                          criterion='gini', max_depth=None, max_features='sqrt',
                          max_leaf_nodes=None, max_samples=None,
                          min_impurity_decrease=0.0, min_samples_leaf=1,
                          min_samples_split=2, min_weight_fraction_leaf=0.0,
                          n_estimators=100, n_jobs=-1, oob_score=False,
                          random_state=123, verbose=0, warm_start=False)
```

```
xt.fit(X_train, y_train)
```

```
ExtraTreesClassifier
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=100, n_jobs=-1, oob_score=False,
                     random_state=123, verbose=0, warm_start=False)
```

We trained Extra Tree Classifier by passing X_train and y_train as parameter.

MODEL EVALUATION AND SAVING FINAL MODEL

MODEL EVALUATION AND SAVING THE MODEL

```
print(classification_report(y_test, xt.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.83      0.94      0.89     36982
           1       0.42      0.18      0.25      8461

    accuracy                           0.80     45443
   macro avg       0.63      0.56      0.57     45443
weighted avg       0.76      0.80      0.77     45443
```

```
dump(xt, "/content/drive/MyDrive/Data Science/Rakamin/IDX Partners Virtual Internship/extra_tree.joblib")
```

```
['/content/drive/MyDrive/Data Science/Rakamin/IDX Partners Virtual Internship/extra_tree.joblib']
```

We **gained an accuracy of 0.80 in the test set**. It seemed like the model outperformed in synthetical training data and underperformed in real testing data. For future improvement, we could experiment with various sampling strategies and tune the model.

THANK YOU