CS3244 Project

# Credit Card Approval Prediction

RODERICK KONG ZHANG
SHEN TIANWEI
OLIVIA (ALEX) XIAO
JOSHUA HARSHA DASS
RAYAN MAKNOJIA

Group 35

2025 0004 0020 3244

# TABLE OF CONTENTS

# 01

# About the Project

Motivation, Data Overview

# Motivation: Enhancing Credit Approvals

### Why This Project Matters

**Addresses a critical challenge:** Improving credit card approval processes in modern financial risk management

### Benefits for Financial Institutions

**Better risk management:**

Via data-driven insights

**Outcomes:**

- Lower default rates
- Optimised portfolios
- Reduced costs

### Benefits for Individuals

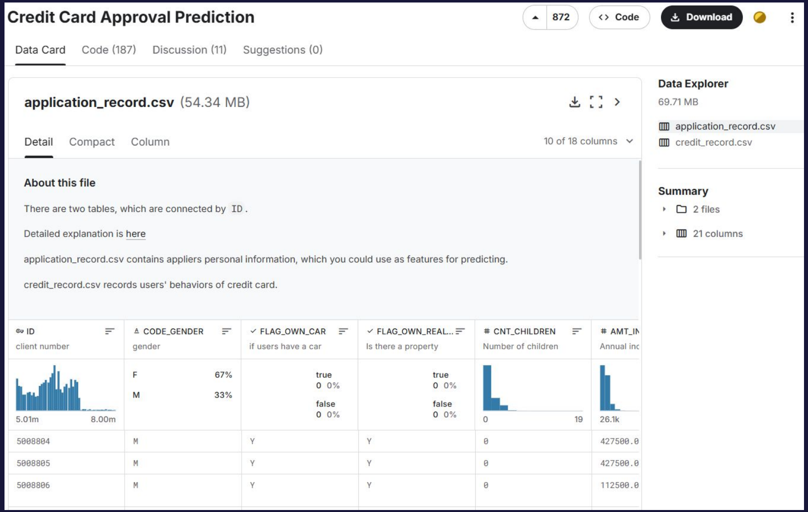
**Fairer process:** Less bias

**Outcomes:**

- Wider credit access
- Transparent criteria
- Empowered financial decisions

# Data Overview

- **Source:** "Credit Card Approval" dataset on Kaggle (uploaded by user "rikdifos")
- **Type:** Unlabelled data
- **Content:** Each row corresponds to a credit card applicant, with features relating to personal and financial profiles



# 02

# Data Inspection

Reviewing Datasets

# Data Inspection

- We must first have a certain understanding of the characteristics and structure of the data
- Initially, there are two .csv files (application_record.csv and credit_record.csv)
- We need to get the labels after preprocessing (since these are unlabelled datasets)
- Description of the datasets:

| | Feature name | Explanation | Remarks |
|---|---|---|---|
| 0 | ID | Client number | |
| 1 | CODE_GENDER | Gender | |
| 2 | FLAG_OWN_CAR | Is there a car | |
| 3 | FLAG_OWN_REALTY | Is there a property | |
| 4 | CNT_CHILDREN | Number of children | |
| 5 | AMT_INCOME_TOTAL | Annual income | |
| 6 | NAME_INCOME_TYPE | Income category | |
| 7 | NAME_EDUCATION_TYPE | Education level | |
| 8 | NAME_FAMILY_STATUS | Marital status | |
| 9 | NAME_HOUSING_TYPE | Way of living | |
| 10 | DAYS_BIRTH | Birthday | Count backwards from current day (0), -1 means yesterday |
| 11 | DAYS_EMPLOYED | Start date of employment | Count backwards from current day(0). If positive, it means the person currently unemployed. |
| 12 | FLAG_MOBIL | Is there a mobile phone | |
| 13 | FLAG_WORK_PHONE | Is there a work phone | |
| 14 | FLAG_PHONE | Is there a phone | |
| 15 | FLAG_EMAIL | Is there an email | |
| 16 | OCCUPATION_TYPE | Occupation | |
| 17 | CNT_FAM_MEMBERS | Family size | Number of family members |

| | Feature name | Explanation | Remarks |
|---|---|---|---|
| 0 | ID | Client number | |
| 1 | MONTHS_BALANCE | Record month | The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on |
| 2 | STATUS | Status | 0: 1-29 days past due; 1: 30-59 days past due; 2: 60-89 days overdue; 3: 90-119 days overdue; 4: 120-149 days overdue; 5: Overdue or bad debts, write-offs for more than 150 days; C: paid off that month; X: No loan for the month |

# 03
# Data Cleaning

Formatting Data, Creating Labels

# Data Cleaning

- After observing all the feature variables that have emerged, we need to clean up the samples with invalid values

```python
# Convert jobs and births to years and handle outliers
df_new = df.copy() # Copy the dataframe do not change the original dataframe
df_new = df_new[df_new["DAYS_BIRTH"] < 0]
df_new["AGE"] = (-df_new["DAYS_BIRTH"]) / 365
df_new["AGE"] = df_new["AGE"].round(0).astype(int)
df_new["DAYS_EMPLOYED_CLEAN"] = df_new["DAYS_EMPLOYED"].apply(lambda x: 0 if x > 0 else -x / 365) # Convert to years
df_new["DAYS_EMPLOYED_CLEAN"] = (df_new["DAYS_EMPLOYED_CLEAN"] * 2).round() / 2 # Round to the nearest 0.5
display(df_new.head())
print(df_new.shape)
```

5] Python

- Change some ‘yes’ and ‘no’ labels to boolean values of ‘0’ and ‘1’

```python
# Convert the binary columns to 0 and 1

df_new['FLAG_OWN_CAR'] = df_new['FLAG_OWN_CAR'].replace({'Y': 1, 'N': 0})
df_new['FLAG_OWN_REALTY'] = df_new['FLAG_OWN_REALTY'].replace({'Y': 1, 'N': 0})

display(df_new.head())
```

# Data Cleaning

- We remove non-adults (under 21 years old)

```
# Remove the young people
df_new = df_new[df_new["AGE"] > 21]
display(df_new.head())
print(df_new.shape)
```

- Find the samples where the eigenvalue is NaN and delete them

```
# Remove the Nan values
df_new.replace('', np.nan, inplace=True)
df_new.dropna(inplace=True)
print(df_new.shape)
```

- application_record.csv data cleaning is complete after sorting

```
# sort the dataframe by ID
df_new = df_new.sort_values(by='ID', ascending=True).reset_index(drop=True)
display(df_new.head())
```

# Data Cleaning

- We count the proportion of people with different overdue periods in credit_record.csv and obtained the following table

```python
credit0 = credit.copy()
def calculate_rate(pivot_tb, command):
    '''calculate bad customer rate'''
    credit0['status'] = 0
    exec(command)
    sumagg = credit0.groupby('ID')['status'].agg('sum').reset_index()
    pivot_tb_merge = pd.merge(pivot_tb[['ID']], sumagg, on='ID', how='left')
    pivot_tb_merge['status'] = pivot_tb_merge['status'].fillna(0)
    pivot_tb_merge.loc[pivot_tb_merge['status'] > 1, 'status'] = 1
    rate = pivot_tb_merge['status'].sum() / len(pivot_tb_merge)
    return round(rate, 5)


commands = {
    'past due more than 1 day': "credit0.loc[credit0['STATUS'].isin(['0','1','2','3','4','5']), 'status'] = 1",
    'past due more than 30 days': "credit0.loc[credit0['STATUS'].isin(['1','2','3','4','5']), 'status'] = 1",
    'past due more than 60 days': "credit0.loc[credit0['STATUS'].isin(['2','3','4','5']), 'status'] = 1",
    'past due more than 90 days': "credit0.loc[credit0['STATUS'].isin(['3','4','5']), 'status'] = 1",
    'past due more than 120 days': "credit0.loc[credit0['STATUS'].isin(['4','5']), 'status'] = 1",
    'past due more than 150 days': "credit0.loc[credit0['STATUS']=='5', 'status'] = 1"
}


summary_list = []
for situation, cmd in commands.items():
    rate = calculate_rate(pivot_tb, cmd)
    summary_list.append((situation, rate))


summary_dt = pd.DataFrame(summary_list, columns=['situation', 'bad customer ratio'])


display(summary_dt)
```

Python

| | situation | bad customer ratio |
|---|---|---|
| 0 | past due more than 1 day | 0.39432 |
| 1 | past due more than 30 days | 0.06443 |
| 2 | past due more than 60 days | 0.01070 |
| 3 | past due more than 90 days | 0.00537 |
| 4 | past due more than 120 days | 0.00391 |
| 5 | past due more than 150 days | 0.00311 |

# Data Cleaning

- Here we can see that 60 days is a suitable choice for judging whether it is a bad account (with a probability of about 6%), that is, if the value is greater than or equal to 1, we can judge it as a bad user

```
# label the status
df2_new['label'] = np.where(df2_new['STATUS'].isin(['1','2', '3', '4', '5']), 1, 0)
customer_label = df2_new.groupby('ID')['label'].max().reset_index()
display(customer_label.head(30))
```

- We re-label it and then label the application_record.csv according to the ID. We only consider the intersection of two IDs

```
final_data = pd.merge(customer_label, df_new, on='ID', how='inner')
display(final_data)
```

✓ 0.0s

# 04

# Exploratory Data Analysis

Data Visualisation, Handling Imbalanced Data,
K-means Clustering

# Data Visualisation

- Next, we need to visualise the distribution of the data
- Firstly, we observe that there are numerical and categorical features in the data

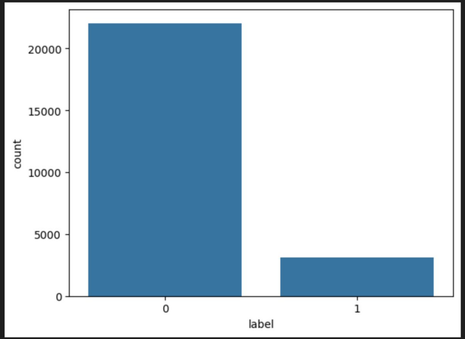
```python
file_path = './cleaned_data/final_data.csv'
data = pd.read_csv(file_path)
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25126 entries, 0 to 25125
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   25126 non-null  int64  
 1   label                25126 non-null  int64  
 2   CODE_GENDER          25126 non-null  object 
 3   FLAG_OWN_CAR         25126 non-null  int64  
 4   FLAG_OWN_REALTY      25126 non-null  int64  
 5   CNT_CHILDREN         25126 non-null  int64  
 6   AMT_INCOME_TOTAL     25126 non-null  float64
 7   NAME_INCOME_TYPE     25126 non-null  object 
 8   NAME_EDUCATION_TYPE  25126 non-null  object 
 9   NAME_FAMILY_STATUS   25126 non-null  object 
 10  NAME_HOUSING_TYPE    25126 non-null  object 
 11  DAYS_BIRTH           25126 non-null  int64  
 12  DAYS_EMPLOYED        25126 non-null  int64  
 13  FLAG_MOBIL           25126 non-null  int64  
 14  FLAG_WORK_PHONE      25126 non-null  int64  
 15  FLAG_PHONE           25126 non-null  int64  
 16  FLAG_EMAIL           25126 non-null  int64  
 17  OCCUPATION_TYPE      25126 non-null  object 
 18  CNT_FAM_MEMBERS      25126 non-null  float64
 19  AGE                  25126 non-null  int64  
 20  DAYS_EMPLOYED_CLEAN  25126 non-null  float64
dtypes: float64(3), int64(12), object(6)
memory usage: 4.0+ MB
None
```

# Data Visualisation

- We observe the distribution of our labels and distinguish between columns of these different data types

```
Proportions:
label
0    0.877099
1    0.122901
Name: count, dtype: float64
```



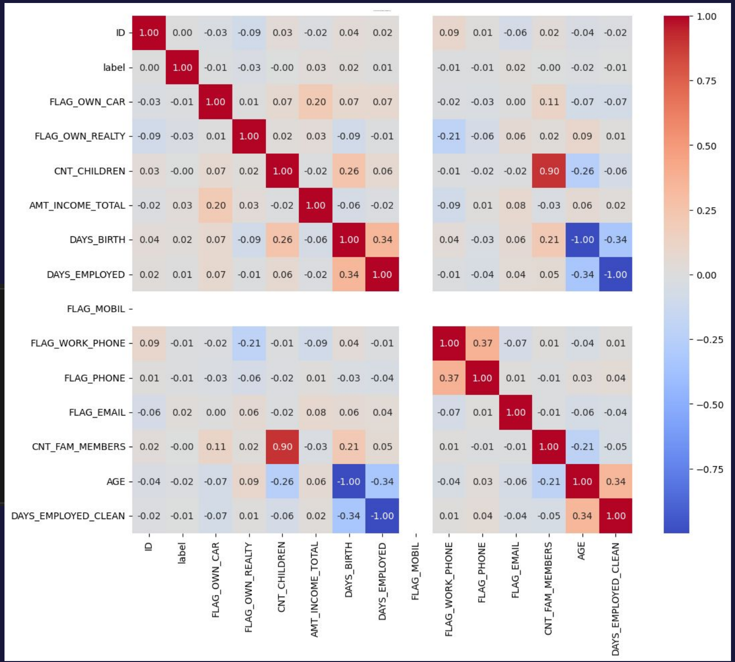

```python
data_types = data.dtypes.value_counts()
numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
print(f"Numerical columns: {numerical_cols}")
print(f"Categorical columns: {categorical_cols}")
print(f"Data types:\n{data_types}")
```

```
Numerical columns: ['ID', 'label', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'FLAG_MOBIL',
Categorical columns: ['CODE_GENDER', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE']
Data types:
int64      12
object      6
float64     3
Name: count, dtype: int64
```

# Data Visualisation

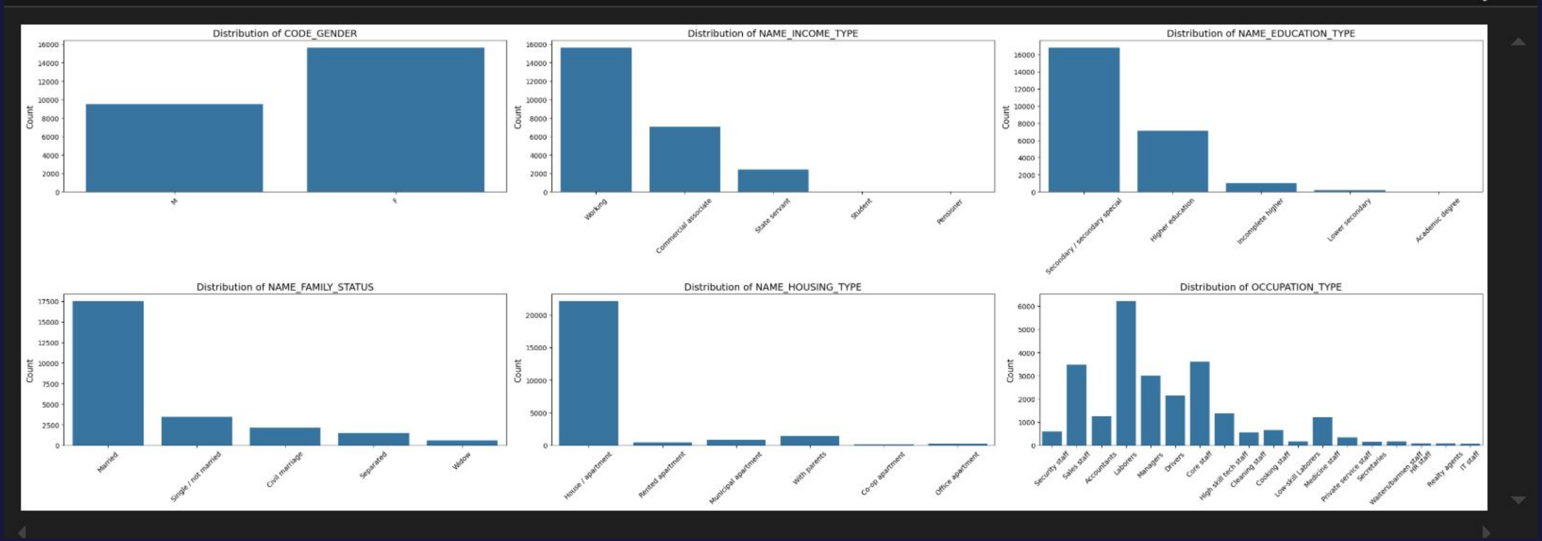
- At the same time, we start to analyse other feature data and their correlation with the labels
- We found that the correlation between numerical data and labels is very small, perhaps the linear relationship is not very strong, or the data is too noisy

```python
correlation_matrix = data[numerical_cols].corr()


plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.title('Correlation Heatmap (Adjusted Size)', fontsize=16)
plt.show()
```



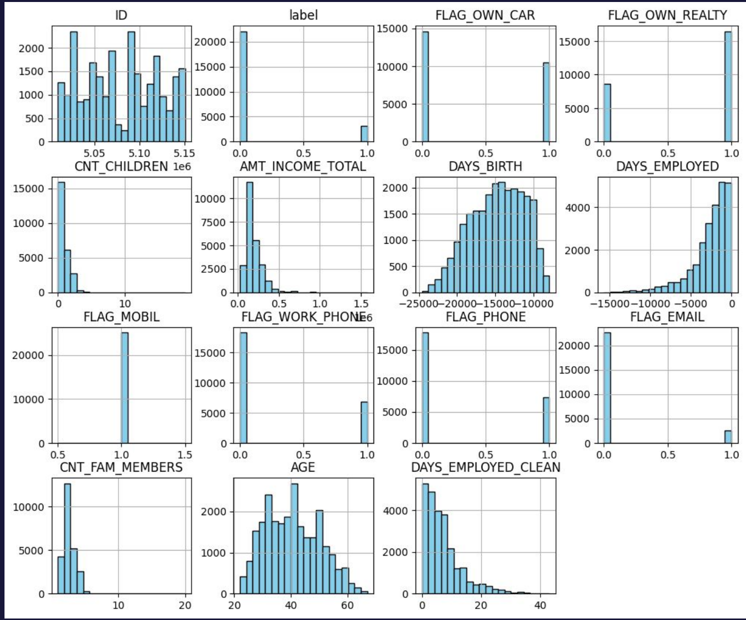
# Data Visualisation

- Visualisation of numerical and categorical data distribution



# Data Visualisation

- Visualisation of numerical and categorical data distribution



# Handling Imbalanced Data

- We found that the rejection rate was too small and the data was unbalanced
- We use SMOTE (Synthetic Minority Over-sampling Technique), hence, we first need to mark the categorical data to prepare for the next step of oversampling both numerical and categorical data

```
ain, test = train_test_split(
  data2,
  test_size=0.3,
  random_state=42,
  stratify=data2['label']

tegorical_columns = ['CODE_GENDER','NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'Grouped_Housing_Type', 'OCCUPATION_TYPE']
splay(train.shape)
splay(test.shape)
```

Python

# Handling Imbalanced Data

- Finally, we integrate the oversampled data into the original dataset (oversampling will cause the training set ratio to increase from 70% to 73%, but the change is not significant)

```python
X_resampled_categorical = X_resampled.iloc[:, :len(categorical_columns)]
X_resampled_numerical = X_resampled.iloc[:, len(categorical_columns):]


df_categorical = pd.DataFrame(X_resampled_categorical, columns=categorical_columns)
df_numerical = pd.DataFrame(X_resampled_numerical.values, columns=numerical_columns)


X_resampled_df = pd.concat([df_categorical, df_numerical], axis=1)


y_resampled_df = pd.DataFrame(y_resampled, columns=['label'])


resampled_data = pd.concat([X_resampled_df, y_resampled_df], axis=1)

display(resampled_data.head())
display(resampled_data.shape)

resampled_data.to_csv("cleaned_data/resampled_data_smote_73percent.csv", index=False, encoding="utf-8")
test.to_csv("cleaned_data/test_set_percent73_smote.csv", index=False, encoding="utf-8")
```

Python

# Handling Imbalanced Data

- We delete the data that will affect the sampling, such as meaningless numbers, and then oversample, setting the small category to oversample to 30%

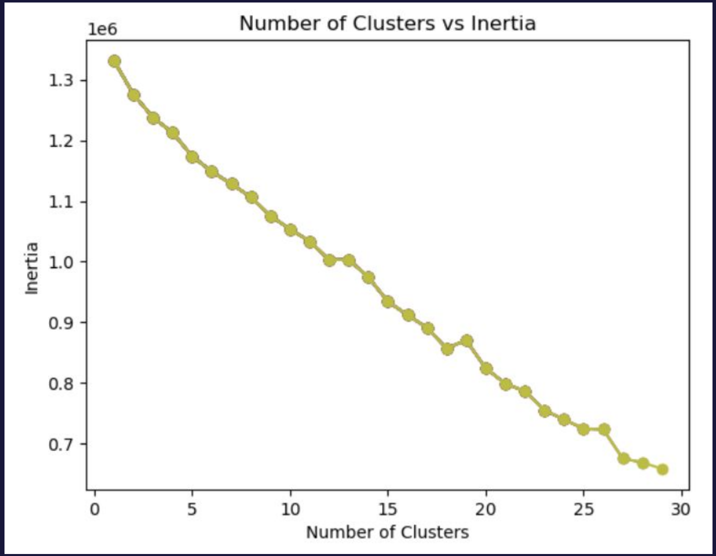
```
X = train.drop(columns=['label','ID','NAME_HOUSING_TYPE_GROUPED','combined','count'])
display(X.shape)
y = train['label']
print(X.dtypes)
X_categorical = X[categorical_columns]


numerical_columns = [col for col in X.columns if col not in categorical_columns]
X_numerical = X[numerical_columns]

X_combined = pd.concat([X_categorical, X_numerical], axis=1)


categorical_features = [X_combined.columns.get_loc(col) for col in categorical_columns]


smotenc = SMOTENC(categorical_features=categorical_features, sampling_strategy=0.3,random_state=42)
X_resampled, y_resampled = smotenc.fit_resample(X_combined, y)

print("Target variable distribution after resampling:")
print(y_resampled.value_counts())
print("Resampled data shape:", X_resampled.shape)
print("Resampled target variable shape:", y_resampled.shape)
```

Python

# K-Means Clustering

**Preprocessing Steps**

- Non-numeric features encoded and data standardised




Optimising Cluster Count

# K-Means Clustering

**Attempt 1 Results**

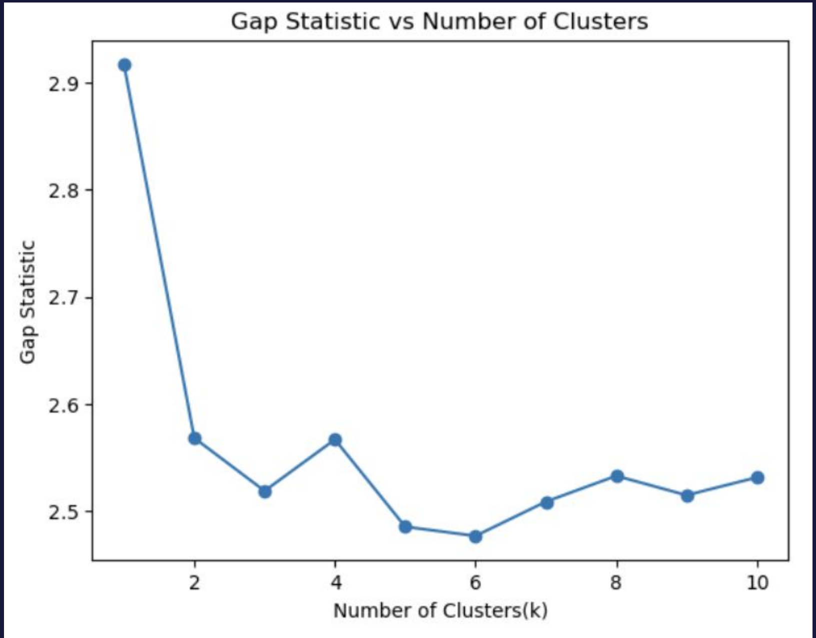
- Based on the graph shown previously, around 5 clusters the inertia starts to decrease more slowly. We will use k = 5.

```
kmeans = KMeans(n_clusters = 5)
kmeans.fit(df)
df["K Means Cluster"] = kmeans.labels_
```

**Evaluation** - Silhouette Score: 0.09662359346975798

# K-Means Clustering

## Iteration with Feature Selection



# K-Means Clustering

**Takeaways**

- No clear clusters within data
  - Silhouette score for initial iteration near 0
  - Gap statistic for final iteration 1

# 05

# Feature Engineering

PCA, LDA, Lasso Regression

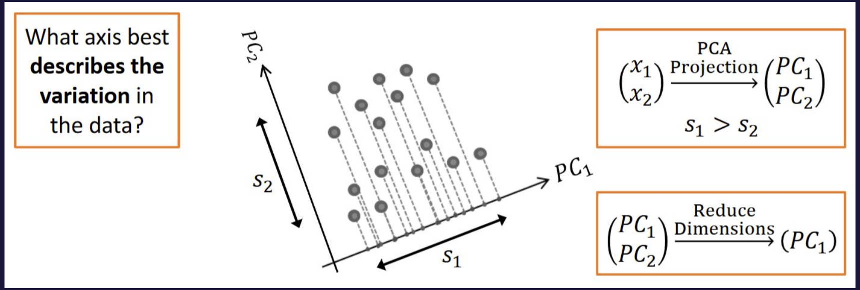
# Feature Extraction: PCA

**Why use Principal Component Analysis?**

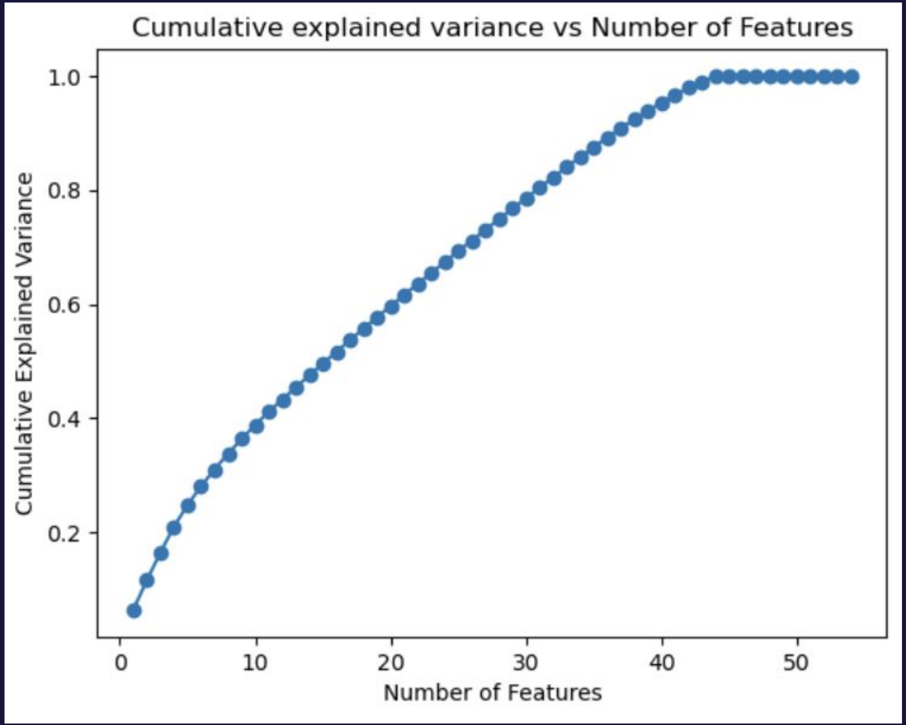
- Reduce dimensionality and noise
- Uncover latent structure and correlations
- Speed up training and improve generalisation

**How does it work?**

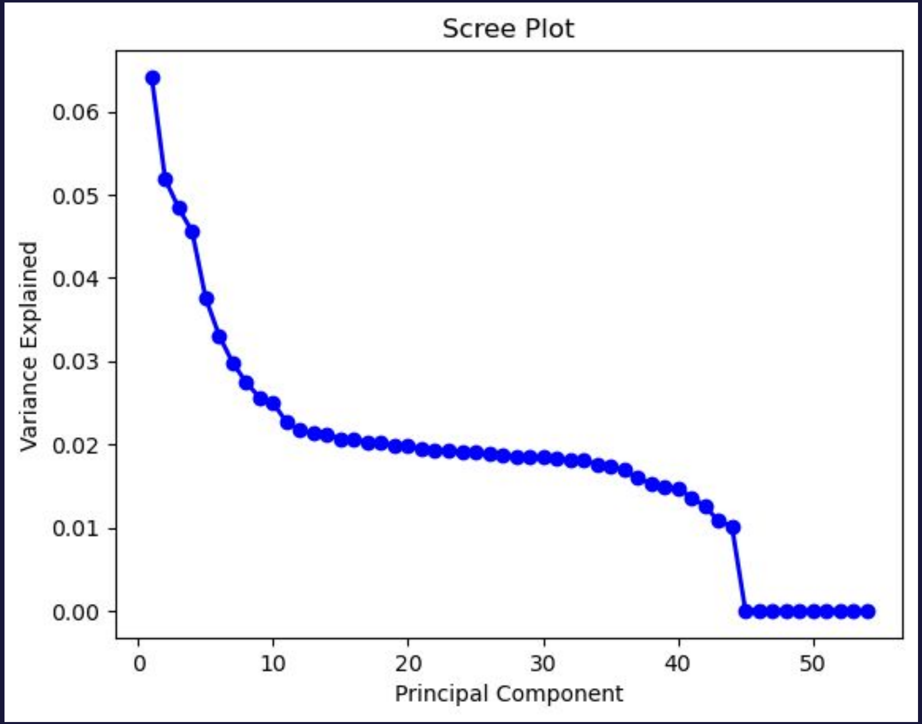
- Centre and optionally scale features
- Compute covariance matrix and its eigenvectors
- Project data onto the top k eigenvectors (principal components)



# Feature Extraction: PCA



# Feature Extraction: PCA



# Feature Extraction: LDA

**Why use Linear Discriminant Analysis?**

- Maximises class separability
- Reduces dimensionality for supervised tasks
- Improves classification performance

**How does it work?**

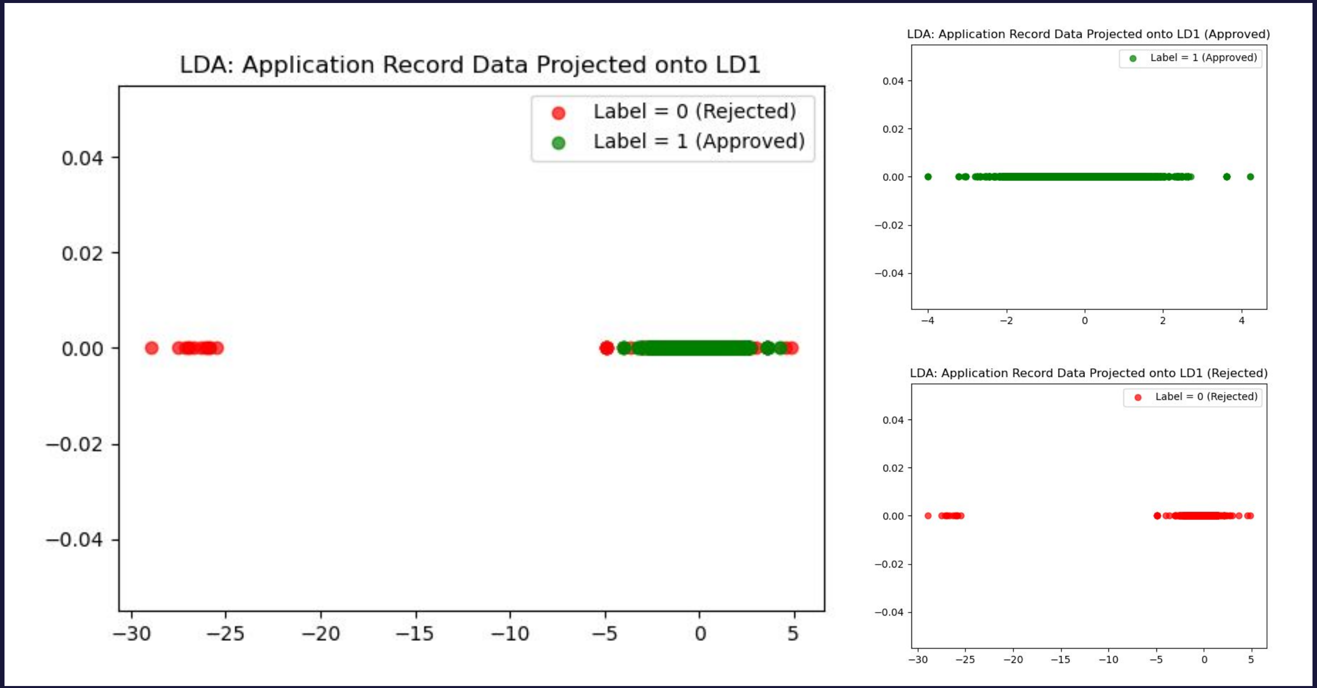
- Compute within-class and between-class scatter matrices and solve eigenproblem
- Project data onto top discriminant eigenvectors

$$J(\mathbf{W}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

*Between-class variance*

*Within-class variance*

# Feature Extraction: LDA



# Feature Selection: Lasso Regression

**Why use L1 (Lasso) Regularisation?**

- Lasso regression is used for feature selection and to improve model interpretability by shrinking coefficients of less important variables to zero
- Removes irrelevant variables from equation that do not improve prediction
- Reduce dimensionality

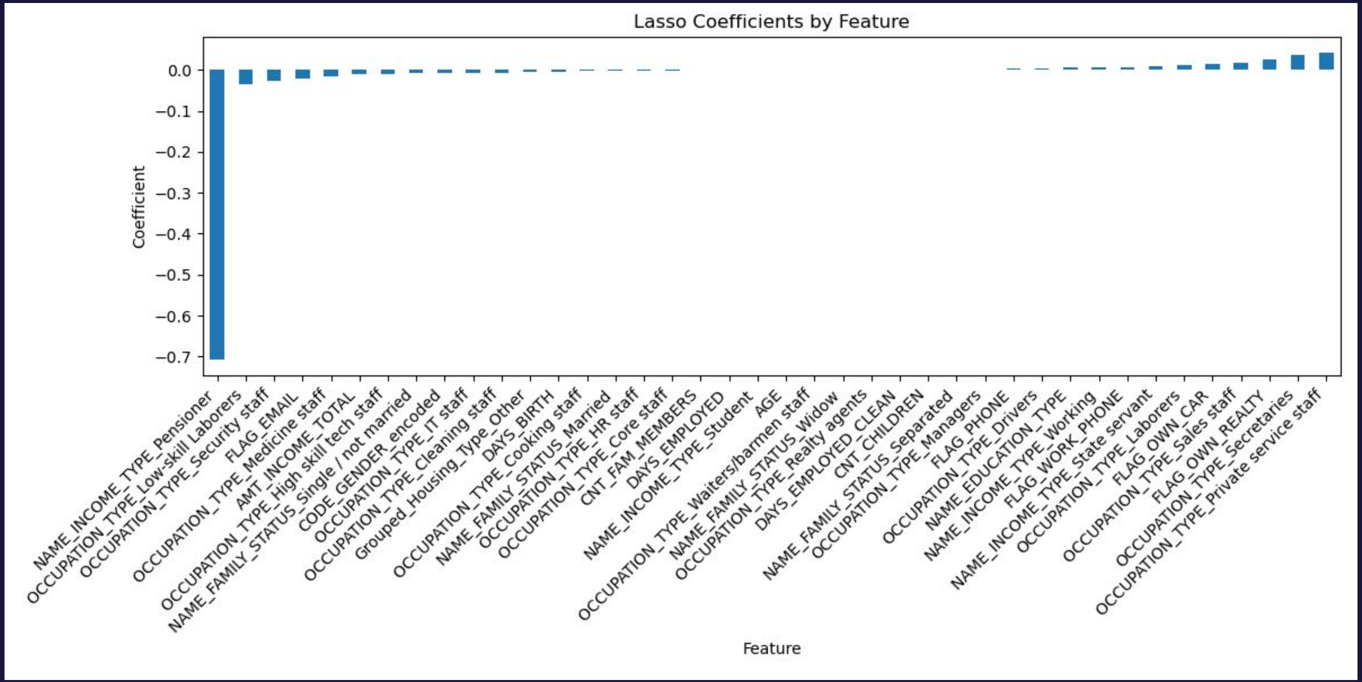
**How does it work?**

- **Penalty:** Uses L1 regularisation to shrink coefficients.
- **Sparsity:** Drives some *slope* exactly to zero ⇒ automatic feature selection.
- **Overfitting control:** Reduces variance by penalising large coefficients.
- **Tuning:** λ (alpha) chosen via cross-validation to balance fit vs sparsity.

$$Cost\ Function = \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^i - y^i\right)^2 + \lambda\sum_{i=1}^{n}|slope|$$

$$\lambda = Hyperparameter$$

# Feature Selection: Lasso Regression



# 06

# Models & Insights

Linear Regression, Logistic Regression,
Polynomial Ridge Regression, Decision Tree, Random Forest

# Model 1: Linear Regression

**Implementation**

1. Convert categorical features to one hot encoding
2. Scale the features
3. Train the model (80% training, 20% test)
4. Find the optimal threshold to classify. We used the ROC curve method - Youden's Index
    - Doesn't assume equal class distribution
    - Balances sensitivity and specificity
    - Widely used in practice
    - Works well for credit approval scenarios where you want to balance risk

# Model 1: Linear Regression

**Evaluation: How good the model is at prediction**

1. **Outcome:** Improvement in almost all evaluation scores after we balanced the dataset
    - Reason: Due to the imbalanced dataset (approved >> rejected), the model becomes biased towards the majority class (approved), leading to poor predictions for the minority class (rejected)
2. **Identifying approvals:** Reasonable ability
3. **Identifying rejections:** Weak ability
4. **Overall:** Not suitable. For credit approval, false positives are more costly than false negatives as institutions can lose the whole principal amount compared to just the interest

**Before Balancing Dataset**

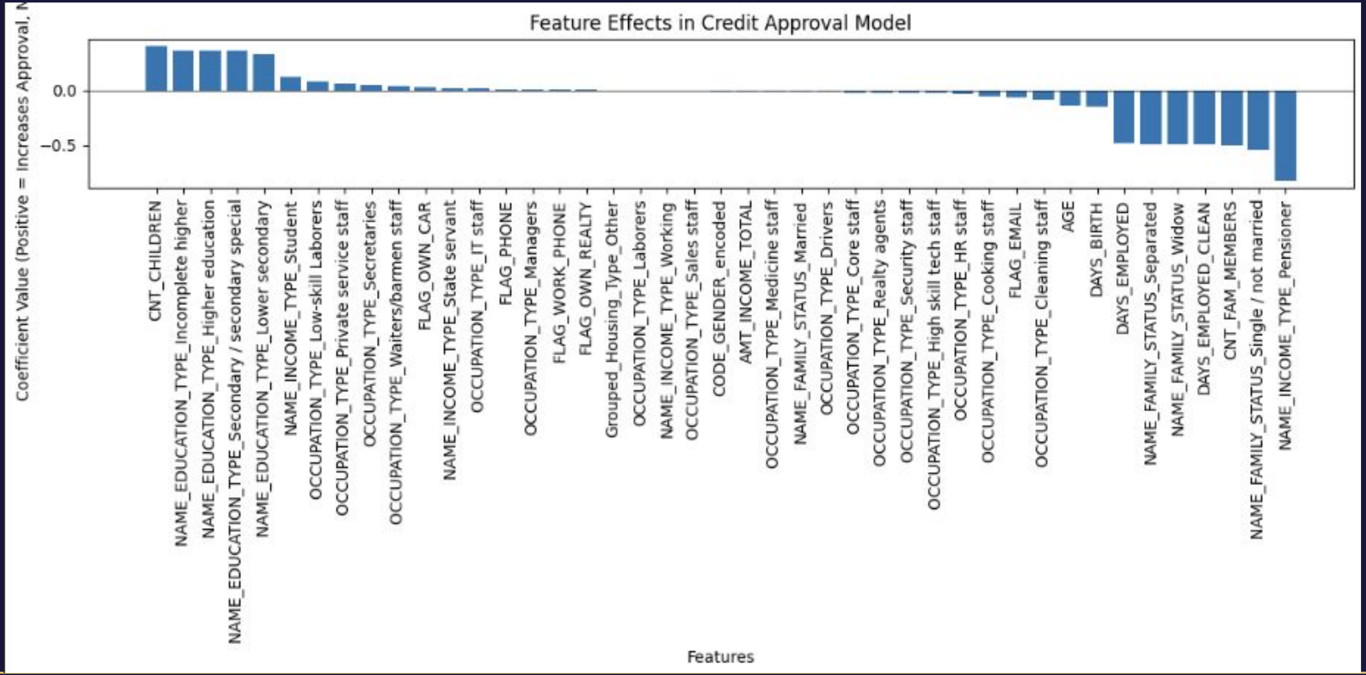
Accuracy: 0.513
Precision: 0.896
Recall: 0.502
Specificity: 0.586
F1 score: 0.644
ROC-AUC: 0.546
PR-AUC: 0.894

**After Balancing Dataset**

Accuracy: **0.721**
Precision: **0.907**
Recall: **0.750**
Specificity: **0.554**
F1 score: **0.812**
ROC-AUC: **0.699**
PR-AUC: **0.918**

# Model 1: Linear Regression

## Insight: How consumers can improve their approval chances



# Model 2: Logistic Regression

**Implementation**

**Data processing**

Categorical variables to one-hot encoding, handling missing values and scaled numerical features

**Dimensionality reduction with PCA**

Applied PCA to reduce feature space. Eliminated multicollinearity between features. Preserved 95% variance while reducing computational complexity

**Model Training: (70% training, 30% test)**

Employed GridSearchCV for hyperparameter tuning

**Threshold optimisation**

Used ROC curve to find optimal to find optimal classification threshold

# Model 2: Logistic Regression

**Evaluation**
**Key Findings**

- Original dataset was heavily biased towards approvals.
- Recall was perfect so model never falsely reject application.
- However because the false positive was so high, the specificity was very low.
- After Balancing Dataset to 70%. The Specificity and the accuracy was improved. (at default threshold).
- Alternatively I tried optimisation threshold at 0.8520 which increased the specificity but at the cost of accuracy and recall.
- ROC-AUC suggests better overall discriminative ability between classes.

**Before Balancing Dataset**
Accuracy: 0.877
Precision: 0.877
Recall: 1.000
Specificity: 0.002
F1 score: 0.935
ROC-AUC: 0.538
PR-AUC: 0.888

**Balanced -> Optimised Threshold**
Accuracy: 0.880 -> **0.716**
Precision: 0.882 -> **0.902**
Recall: 0.992 -> **0.748**
Specificity: 0.229 - > **0.527**
F1 score: 0.934 -> **0.818**
ROC-AUC: **0.694**
PR-AUC: **0.921**

# Model 2: Logistic Regression

**Evaluation**
**Limitations**

- Model is extremely liberal in approval prediction (high recall)
- PCA reduced interpretability
- Requires further optimisation

**Before Balancing Dataset**
Accuracy: 0.877
Precision: 0.877
Recall: 1.000
Specificity: 0.002
F1 score: 0.935
ROC-AUC: 0.538
PR-AUC: 0.888

**Balanced -> Optimised Threshold**
Accuracy: 0.880 -> **0.716**
Precision: 0.882 -> **0.902**
Recall: 0.992 -> **0.748**
Specificity: 0.229 - > **0.527**
F1 score: 0.934 -> **0.818**
ROC-AUC: **0.694**
PR-AUC: **0.921**

# Model 2: Logistic Regression

## Insights

### 4 Logistic Regression (baseline **interpretable classifier**)

| Insight | Why it works |
| --- | --- |
| Odds ratio = exp(β) | "Holding others constant, owning a car multiplies approval odds by × OR." |
| Confidence intervals | 95 % CI of OR highlights robust vs. noisy factors. |
| SHAP (LinearExplainer) | Handles one-hot encoded features efficiently. |
| Calibration curve / Brier score | Check that predicted probabilities are credible. |

# Model 3: Polynomial Ridge Regression

**Implementation: Why use Polynomial Ridge Regression?**

- Improve on linear model by capturing non-linear relationships and feature interactions (e.g. income × employment)
- Control model complexity and multicollinearity via L2 regularisation
- Boost classification accuracy on approval outcomes

**Implementation: How does it work?**

1. Expand original predictors into polynomial terms up to degree d

**Training set** $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{m}$

$$\{x_1 = 0, x_2 = 0\} \to \{y = -1\}$$
$$\{x_1 = 1, x_2 = 1\} \to \{y = -1\}$$
$$\{x_1 = 1, x_2 = 0\} \to \{y = +1\}$$
$$\{x_1 = 0, x_2 = 1\} \to \{y = +1\}$$

**2nd order** polynomial model

$$\mathbf{P} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & x_{1,1}x_{1,2} & x_{1,1}^2 & x_{1,2}^2 \\ 1 & x_{2,1} & x_{2,2} & x_{2,1}x_{2,2} & x_{2,1}^2 & x_{2,2}^2 \\ 1 & x_{3,1} & x_{3,2} & x_{3,1}x_{3,2} & x_{3,1}^2 & x_{3,2}^2 \\ 1 & x_{4,1} & x_{4,2} & x_{4,1}x_{4,2} & x_{4,1}^2 & x_{4,2}^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

# Model 3: Polynomial Ridge Regression

**Implementation: How does it work?**

2. Fit features to labels by minimising the loss function
3. Select regularisation strength λ (and polynomial degree) via k-fold cross-validation

$$\sum(y_i - \hat{y}_i)^2 + \lambda \sum \beta_j^2$$

Loss Function

# Model 3: Polynomial Ridge Regression

**Implementation: Tuning**

- Dataset used for best results: SMOTE
- Feature selection used: Lasso Regression and PCA
- Perform 5-fold cross-validation to tune hyperparameter λ
    - **Optimal λ after using Lasso for orders [1, 2, 3]:** [10000, 1, 1000]
    - **Optimal λ after using PCA for orders [1, 2, 3]:** [10000, 10000, 0.1]
- Find optimal thresholds by plotting ROC curve
    - **Optimal threshold after using Lasso for orders [1, 2, 3]:** [0.79, 0.74, 0.78]
    - **Optimal threshold after using PCA for orders [1, 2, 3]:** [0.77, 0.78, 0.78]

# Model 3: Polynomial Ridge Regression

**Evaluation: Predictive ability of model**

1. **Outcome:** Significant improvement in specificity, and some decrease in other evaluation metrics
   - **Reason:** Removing noisy/irrelevant features and reducing overfitting reduces false positives (original model was classifying almost all as approved)
2. **Identifying approvals:** Moderate ability
3. **Identifying rejections:** Moderate ability
4. **Best order:** 3
5. **Overall:** Moderately suitable
   - **Reason:** Moderate scores for TPR and TNR, and higher specificity compared to other models (significantly less false positives which are costly for banks)

**Before Balancing Dataset**

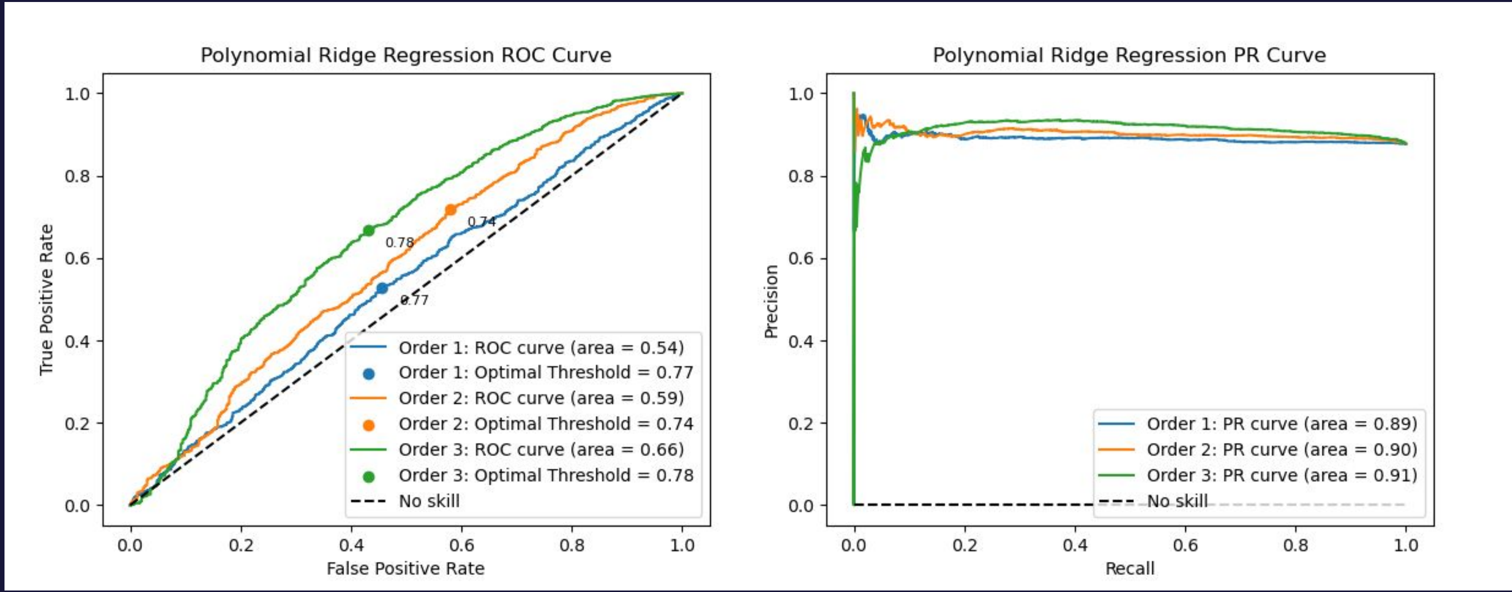
Accuracy: [0.877, 0.878, 0.876]
Precision: [0.877, 0.879, 0.893]
Recall: [1.000, 0.999, 0.976]
Specificity: [0.002, 0.016, 0.163]
F1 score: [0.935, 0.935, 0.932]
ROC-AUC: [0.542, 0.610, 0.714]
PR-AUC: [0.890, 0.911, 0.927]

**After SMOTE, Lasso, Optimised Thresholds**

Accuracy: [**0.488**, **0.585**, **0.660**]
Precision: [**0.892**, **0.899**, **0.915**]
Recall: [**0.474**, **0.593**, **0.674**]
Specificity: [**0.591**, **0.528**, **0.558**]
F1 score: [**0.619**, **0.715**, **0.777**]
ROC-AUC: [**0.538**, **0.589**, **0.656**]
PR-AUC: [**0.890**, **0.902**, **0.912**]

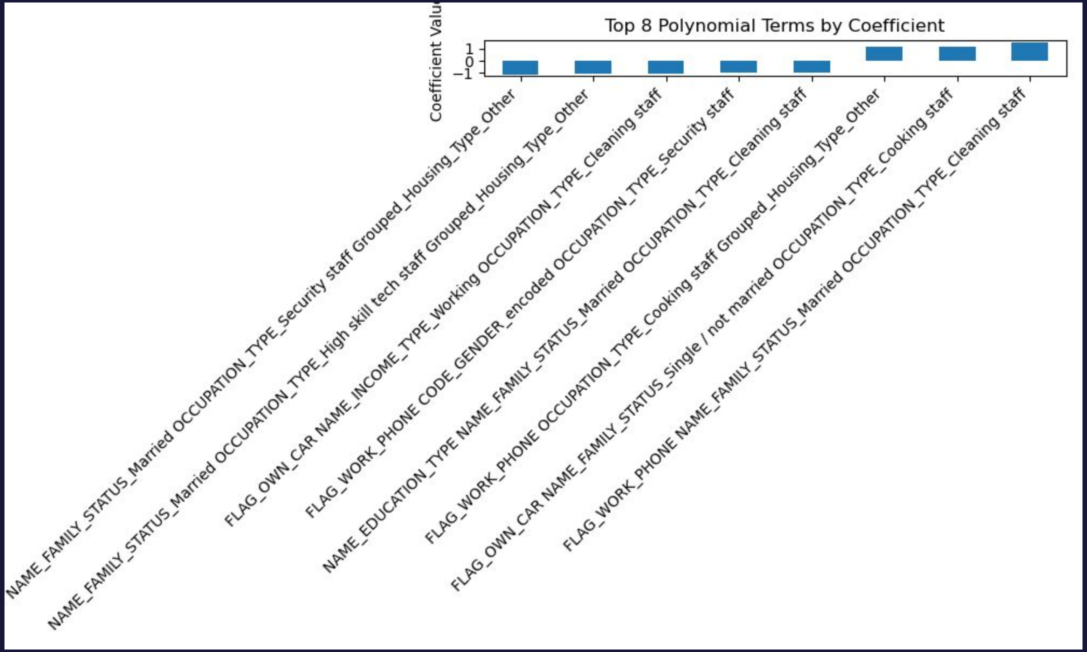
# Model 3: Polynomial Ridge Regression

## Evaluation: ROC Curve and PR Curve



# Model 3: Polynomial Ridge Regression

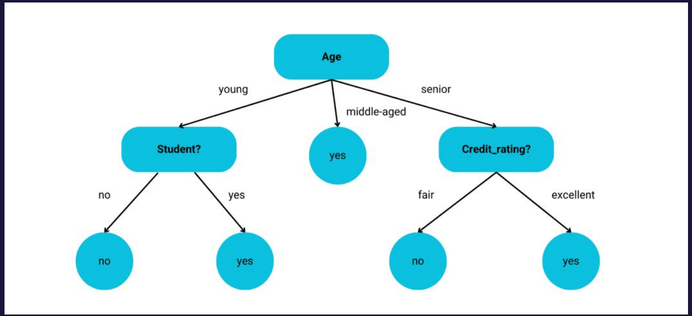
## Insights: Top 8 polynomial terms by |weight|



# Model 4: Decision Tree

**Implementation: Why use Decision Trees?**

- Yields clear "if-then" rules (e.g., income < $30k ⇒ reject) that stakeholders can easily audit
- Captures non-linear effects and feature interactions (age × employment history) without manual engineering
- Handles mixed numeric/categorical inputs and missing values by default
- Visual insights into which applicant traits (e.g., pensioner status, home-ownership) drive approvals



# Model 4: Decision Tree

**Implementation: How does it work?**

1. **Select split:** At each node, pick the feature and threshold that best reduces impurity (Gini or entropy)
2. **Partition data:** Divide applicants into two groups based on that split
3. **Recurse:** Repeat on each subset until stopping criteria (e.g., max depth) is met
4. **Predict:** For a new applicant, follow the path of splits to a leaf; the majority class is the model's decision

**Implementation: Tuning**

- **Dataset used for best results:** Processed original dataset (imbalanced)
- **Feature selection used:** Lasso Regression
- **Decision Tree Pruning**
    - Maximum depth = 25
    - Minimum decrease in Gini impurity = 0.00001

# Model 4: Decision Tree

**Evaluation: How good the model is at prediction**

1. **Outcome:** Some improvement in precision and specificity, and negligible decrease in other evaluation metrics
   - **Reason:** Removing noisy/irrelevant features and reducing overfitting reduces false positives
2. **Identifying approvals:** Good ability
3. **Identifying rejections:** Weak ability
   - **Reason:** Decisions are skewed/biased towards approvals due to imbalanced dataset
4. **Overall:** Not suitable
   - **Reason:** For credit approval, false positives are more costly than false negatives as institutions can lose the whole principal amount compared to just the interest

**Before Lasso Regression**

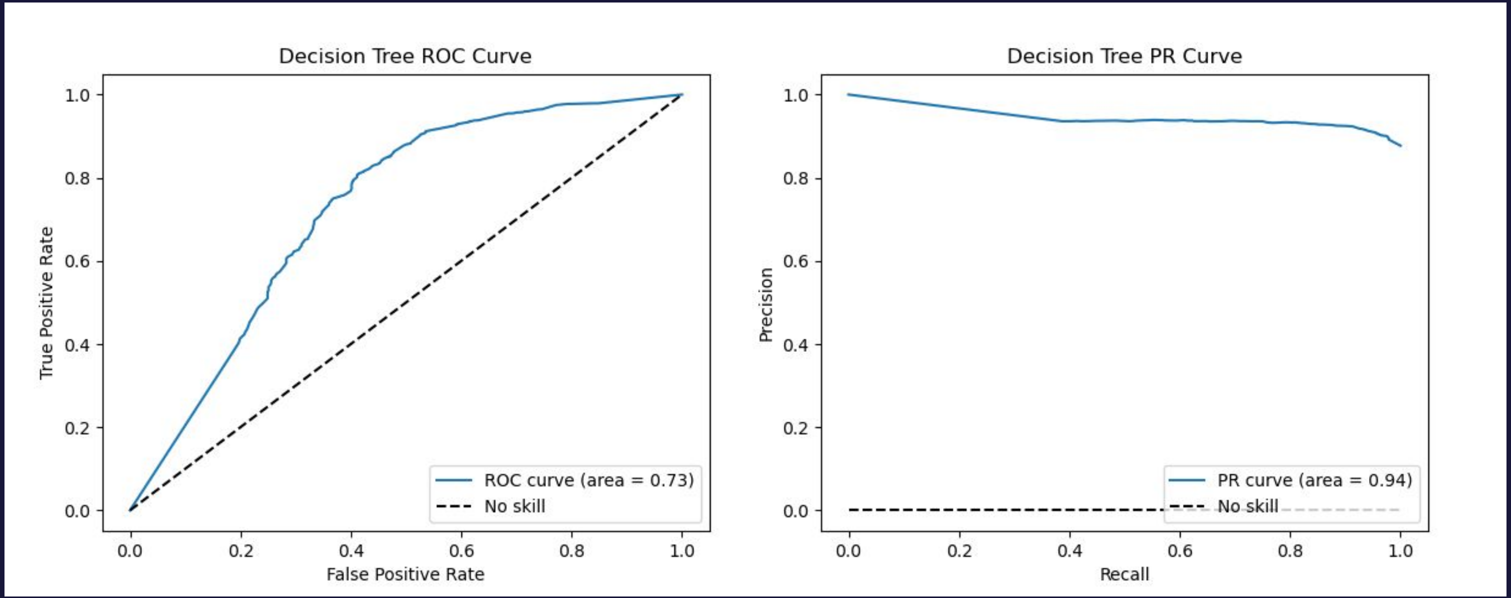
Accuracy: 0.870, 0.925 (Train)
Precision: 0.911
Recall: 0.944
Specificity: 0.343
F1 score: 0.927
ROC-AUC: 0.729
PR-AUC: 0.946

**After Lasso Regression**

Accuracy: **0.868**, **0.927** (Train)
Precision: **0.914**
Recall: **0.939**
Specificity: **0.367**
F1 score: **0.926**
ROC-AUC: **0.728**
PR-AUC: **0.945**

# Model 4: Decision Tree

## Evaluation: ROC Curve and PR Curve

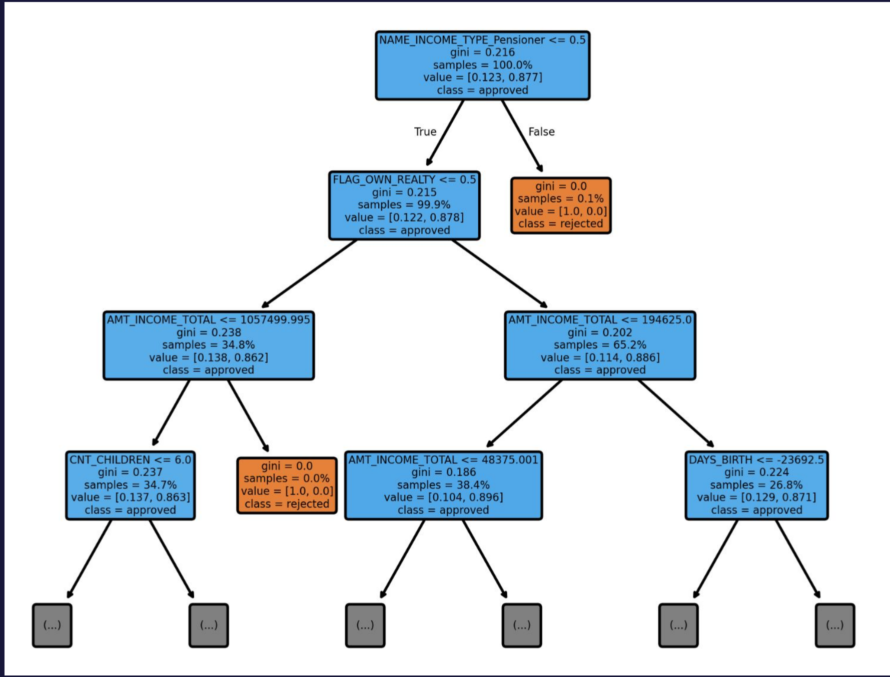


# Model 4: Decision Tree

**Insights: Transparency of application decisions**

**Decision trees are inherently interpretable**

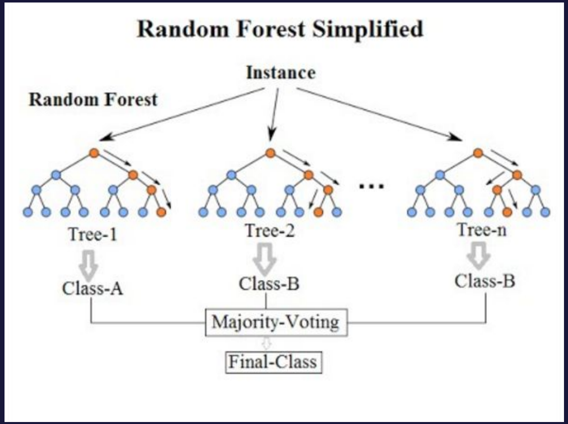
- Pensioner status is the most important (retirees are always rejected)
- Applicants with income > $1 057 500 are all rejected, despite high earnings
- Applicants with large families are likely to be rejected (> 6 children)



# Model 5: Random Forest

**Implementation: Why use Random Forests?**

- Drastically reduces overfitting and model variance on credit-approval data, improving predictive accuracy of single decision tree by averaging many models
- Handles non-linear patterns, feature interactions and mixed data types
- Provides built-in feature-importance scores for ranking applicant risk factors
- Robust to outliers and noisy measurements in financial profiles



# Model 5: Random Forest

**Implementation: How does it work?**

1. **Bootstrap sampling**: Build each tree on a random subset (with replacement) of applicants
2. **Random feature picks:** At each node, consider only a random subset of features to split
3. **Grow many trees:** Repeat to create an ensemble of diverse decision trees
4. **Aggregate votes:** For classification, each tree votes approve/reject and the forest takes the majority

**Implementation: Tuning**

- **Dataset used for best results:** SMOTE
- **Feature selection used:** Lasso Regression
- **Decision Tree Pruning**
  - Maximum depth = 25
  - Minimum decrease in Gini impurity = 0.00001
- **Number of trees:** 1000

# Model 5: Random Forest

**Evaluation: How good the model is at prediction**

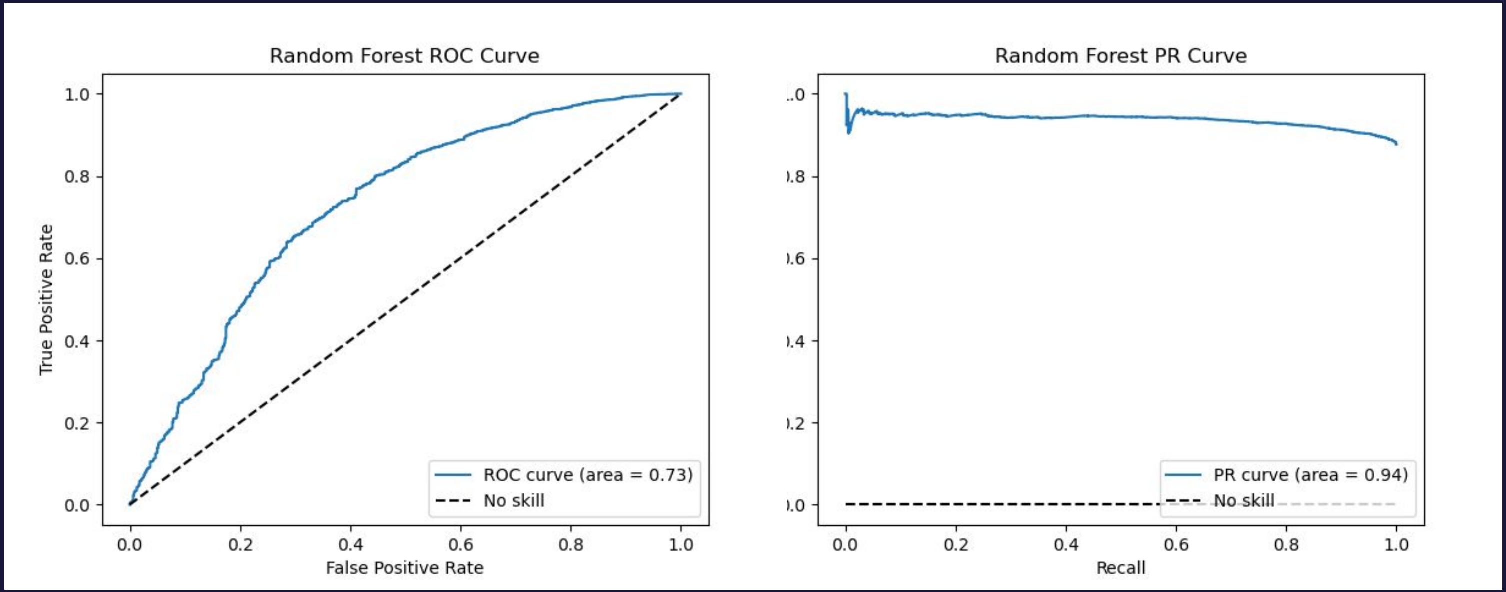
1. **Outcome:** Significant improvement in specificity after using Lasso regression and SMOTE, and some decrease in other evaluation metrics
   - **Reason:** Removing noisy/irrelevant features and reducing overfitting reduces false positives
   - **Reason:** Balancing the data makes the model less biased towards approval, increasing false negatives
2. **Identifying approvals:** Good ability
3. **Identifying rejections:** Weak ability
4. **Using Random Forest:** Decreased variance but increased bias, possibly due to underfitting
5. **Overall:** Not suitable
   - **Reason:** For credit approval, false positives are more costly than false negatives as institutions can lose the whole principal amount compared to just the interest

**Before Balancing Dataset**
Accuracy: 0.885, 0.901 (Train)
Precision: 0.891
Recall: 0.990
Specificity: 0.138
F1 score: 0.938
ROC-AUC: 0.790
PR-AUC: 0.954

**After SMOTE & Lasso**
Accuracy: **0.869**, **0.911** (Train)
Precision: **0.890**
Recall: **0.958**
Specificity: **0.239**
F1 score: **0.928**
ROC-AUC: **0.726**
PR-AUC: **0.937**

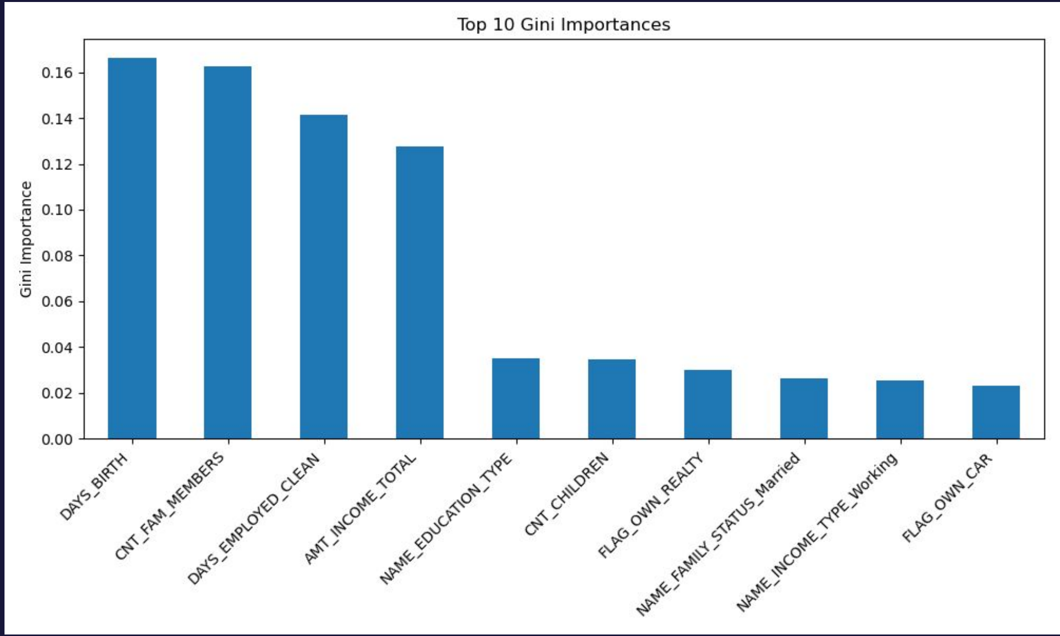
# Model 5: Random Forest

## Evaluation: ROC Curve and PR Curve



# Model 5: Random Forest

**Insights: Most important user statistics when considering credit card approval**



# 07

# Conclusion

Model Comparison, Conclusion

# Model Comparison

| Evaluation Metric | Linear Regression | Logistic Regression | Polynomial Ridge Regression | | | Decision Tree | Random Forest |
|---|---|---|---|---|---|---|---|
| | | | Order 1 | Order 2 | Order 3 | | |
| Accuracy | 0.721 | 0.716 | 0.488 | 0.585 | 0.660 | 0.868 | 0.869 |
| Precision | 0.907 | 0.902 | 0.915 | 0.899 | 0.915 | 0.914 | 0.890 |
| Recall/ Sensitivity | 0.750 | 0.748 | 0.474 | 0.593 | 0.674 | 0.939 | 0.958 |
| Specificity | 0.554 | 0.527 | 0.591 | 0.528 | 0.558 | 0.367 | 0.239 |
| F1 Score | 0.812 | 0.818 | 0.619 | 0.715 | 0.777 | 0.926 | 0.928 |
| ROC-AUC | 0.699 | 0.694 | 0.538 | 0.589 | 0.656 | 0.728 | 0.726 |
| PR-AUC | 0.918 | 0.921 | 0.890 | 0.902 | 0.912 | 0.945 | 0.937 |

# Conclusion

**Best Model?**
**None of the models trained predict credit card approval very well**

- **Curse of dimensionality:** after one-hot encoding and degree-d polynomial expansion, there are hundreds/thousands of features but only a few samples → the model fits noise, not true signal
- **Severe class imbalance:** One class dominates, models simply learn the majority and ignore the minority
- **Multicollinearity:** Correlated predictors (e.g., income and employment length) inflate variance and obscure true effects
- **Synthetic/anonymised data:** May not reflect real applicant distributions or include all predictive variables.

# Conclusion

**If we have to pick a model to use**
**Logistic Regression**

- Inherently probabilistic, essential for thresholding and credit decisions
- Average in all evaluation metrics

**Polynomial Ridge Regression**

- Highest precision and one of the highest specificities, relatively less false positives that are risky for banks

**Decision Tree**

- Highest ROC-AUC and PR-AUC

| Evaluation Metric | Logistic Regression | Polynomial Ridge Order 3 | Decision Tree |
|---|---|---|---|
| Accuracy | 0.716 | 0.660 | 0.868 |
| Precision | 0.902 | **0.915** | 0.914 |
| Recall/ Sensitivity | 0.748 | 0.674 | 0.939 |
| Specificity | 0.527 | 0.558 | 0.367 |
| F1 Score | 0.818 | 0.777 | 0.926 |
| ROC-AUC | 0.694 | 0.656 | **0.728** |
| PR-AUC | 0.921 | 0.912 | **0.945** |

# Thank You