DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of donations. Page visits, donations, and success stories are tracked to eventually create each customized website for organized for people on the DonorsChoose.org platform.

Here year, DonorsChoose.org wants requests to receive close to 500,000 project proposals. As a result, DonorsChoose.org has identified several problems they are solving:

- How to ensure current social processes and resources to screen 500,000 projects so that they can be passed as quickly and as accurately as possible
- How to ensure the consistency of project writing and to offer alternative suggestions for the customers to improve
- How to ensure the quality of the project proposals, so that the projects are more likely to be funded

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, taking into account the teacher's profile and the project they are requesting funding for. Kaggle and DonorsChoose.org have set up the DonorsChoose.org Data Science Challenge where teams can submit their models.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`teacher_id`	A unique identifier for the proposal's teacher. *Ex: tdd314b1cb5f8f7c8f694d279a190954*
`teacher_prefix`	Teacher's title. *Ex: Ms., Mr., Mrs., Mr., Teacher.*
`school_state`	State where school is located (Chicago USA). *Ex: WY*
`project_submitted_datetime`	Datetime when project was submitted. *Ex: 2016-04-26 17:43:53*
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: *Ex: Grades PreK-2* *Grades 3-5* *Grades 6-8* *Grades 9-12*
`project_subject_categories`	One or more (comma-separated) subject categories for the project. *Ex: Music & The Arts* *Ex: Literacy & Language, Math & Science*
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. *Ex: Gym & Fitness* *Ex: ESL, Mathematics*
`project_title`	Title of the project. *Ex: Art Supplies for Project Runway*
`project_essay_1`	First application essay* *Ex: My students are 21st century learners. They need to be exposed to a variety of ideas and be able to think critically about these ideas. Art is a way for students to express themselves and to see the world around them. Art is also a way for students to see themselves as artists and to see themselves succeed. Many of my students have never been to a museum or seen a piece of art in a museum. Many of my students have never had the opportunity to take a piece of art home to hang in their room. Many of my students have never had the opportunity to see their work displayed in a gallery. Many of my students have never had the opportunity to see their work displayed in a school hallway. Many of my students have never had the opportunity to see their work displayed in their classroom. Many of my students have never had the opportunity to see their work displayed in their school cafeteria. Many of my students have never had the opportunity to see their work displayed in their school library. Many of my students have never had the opportunity to see their work displayed in their school gym. Many of my students have never had the opportunity to see their work displayed in their school auditorium. Many of my students have never had the opportunity to see their work displayed in their school bathroom. Many of my students have never had the opportunity to see their work displayed in their school hallway

```
In [1]: import pandas
In [2]: pd.set_option('display.max_columns', 50)
```

2. Data Cleaning

```
In [3]: %matplotlib inline
```

2.1 Data Quality Assessment

Unfortunately real-world data is often rife with issues. Dirty data contains inaccurate or corrupt records, inconsistent formatting, and business rules that don't make sense. Before we can do any meaningful analysis, we need to take a hard look at our data to see what we're dealing with and to ensure that we don't let bad data slip through the cracks undetected. In this section, we'll perform a preliminary assessment of the quality of our data, looking for missing data, incorrect data, inconsistent formatting, and other---hopefully minor---issues that will require our attention later on. Let's begin by importing the Lending Club loan data into a pandas DataFrame and taking a look at the first 5 rows:

```
In [4]: loans = pd.read_csv('./data/loans/LoanStats3a.csv', skiprows=1, low_memory=False)
        loans.head(5)
```

| | 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_status | issue_d | loan_status | pymnt_plan | url | desc | purpose | title | zip_code | addr_state | dti | delinq_2yrs | earliest_cred_line | inq_last_6mths | mths_since_last_delinq | mths_since_last_record | open_acc | pub_rec | revol_bal | revol_util | total_acc | initial_list_status | out_princ | out_princ_inv | total_pymnt | total_pymnt_inv | total_rec_prin | total_rec_int | total_rec_late_fee | recoveries | collection_recovery_fee | last_pymnt_d | last_pymnt_amnt | next_pymnt_d | last_credit_pull_d | collections_12_mths_ex_med | mths_since_last_major_derog | policy_code | application_type | verification_status_joint | last_week_pymnt | acc_now_delinq | max_bal_bc | num_bc_sats | num_bc_tl | mortgage_acc | num_accts_ever_120_pd | num_actv_bc_tl | num_actv_rev_tl | num_bc_rl | num_il_tl | num_op_rev_tl | num_rev_accts | num_rev_tl | pct_tot_hi_pr | tot_hi_cred_lim | total_bal_ex | total_bc_limit | total_il_high_credit_limit |
|---|---|---

```
[1]: [ ]: [ ]: [ 
```

```
In [30]: # Average accuracies
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print()
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

```
In [31]: # Average accuracies
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print()
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

Switching activation:ReLU, loss:BCE, optimizer:Adam

```
In [32]: # Average accuracies
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print()
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Tanh, loss:BCE, optimizer:Adam

```
In [33]: # Average accuracies
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Sigmoid, loss:BCE, optimizer:Adam

```
In [34]: # Average accuracies
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='sigmoid'))
    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Preparing Early categorical, numerical features + project.ds2020 RDSO preprocessed Jessop.DHS RDSO

```
In [35]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print()
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

Switching activation:ReLU, loss:BCE, optimizer:Adam

```
In [36]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Tanh, loss:BCE, optimizer:Adam

```
In [37]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Sigmoid, loss:BCE, optimizer:Adam

```
In [38]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='sigmoid'))
    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Preparing Early categorical, numerical features + project.PRESTO RDSO preprocessed Jessop.DHS RDSO

```
In [39]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

Switching activation:ReLU, loss:BCE, optimizer:Adam

```
In [40]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Tanh, loss:BCE, optimizer:Adam

```
In [41]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Sigmoid, loss:BCE, optimizer:Adam

```
In [42]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='sigmoid'))
    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

6.4.4 Applying Logistic Regression on TFIDF W2V, SET 4

Switching preprocessed data using TFIDF W2V

```
In [43]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_train']
X_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_test']
y_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_train']
y_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

Switching activation:ReLU, loss:BCE, optimizer:Adam

```
In [44]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_train']
X_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_test']
y_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_train']
y_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Tanh, loss:BCE, optimizer:Adam

```
In [45]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_train']
X_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_test']
y_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_train']
y_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Sigmoid, loss:BCE, optimizer:Adam

```
In [46]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_train']
X_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['X_test']
y_train = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_train']
y_test = np.load('tfidf_w2v_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='sigmoid'))
    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Preparing Early categorical, numerical features + project.DHS2020 RDSO preprocessed Jessop.DHS RDSO

```
In [47]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

Switching activation:ReLU, loss:BCE, optimizer:Adam

```
In [48]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Tanh, loss:BCE, optimizer:Adam

```
In [49]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Sigmoid, loss:BCE, optimizer:Adam

```
In [50]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='sigmoid'))
    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Preparing Early categorical, numerical features + project.PRESTO RDSO preprocessed Jessop.DHS RDSO

```
In [51]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for optimizer in ['SGD', 'Adam', 'RMSprop']:
    print(f"Training with {optimizer}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{optimizer} - Train Accuracy: {train_acc:.4f}")
    print(f"{optimizer} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with SGD...
SGD - Train Accuracy: 0.5000
SGD - Test Accuracy: 0.5000

Training with Adam...
Adam - Train Accuracy: 0.5000
Adam - Test Accuracy: 0.5000

Training with RMSprop...
RMSprop - Train Accuracy: 0.5000
RMSprop - Test Accuracy: 0.5000
```

Switching activation:ReLU, loss:BCE, optimizer:Adam

```
In [52]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    train_acc = history.history['accuracy'][-1]
    test_acc = model.evaluate(X_test, y_test, verbose=0)[1]
    print(f"{activation} - Train Accuracy: {train_acc:.4f}")
    print(f"{activation} - Test Accuracy: {test_acc:.4f}")
    print
```

```
Training with relu...
relu - Train Accuracy: 0.5000
relu - Test Accuracy: 0.5000

Training with tanh...
tanh - Train Accuracy: 0.5000
tanh - Test Accuracy: 0.5000

Training with sigmoid...
sigmoid - Train Accuracy: 0.5000
sigmoid - Test Accuracy: 0.5000
```

Switching activation:Tanh, loss:BCE, optimizer:Adam

```
In [53]: # Load the preprocessed data
# Assuming the preprocessed data is saved in variables X_train, X_test, y_train, y_test
X_train_num, X_train_cat, X_test_num, X_test_cat = np.load('early_data.npz', allow_pickle=True)
X_train = np.concatenate([X_train_num, X_train_cat], axis=1)
X_test = np.concatenate([X_test_num, X_test_cat], axis=1)
y_train = np.load('early_data.npz', allow_pickle=True)['y_train']
y_test = np.load('early_data.npz', allow_pickle=True)['y_test']
# Initialize lists to store the accuracies
for activation in ['relu', 'tanh', 'sigmoid']:
    print(f"Training with {activation}...")
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
In [54]: # pip install imbalanced-learn==0.10.1
         # pip install scikit-learn==1.2.2
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV
         model = SVC(kernel='linear', probability=True)
         param_grid = {'C': [0.01, 0.1, 1, 10]}
         grid = GridSearchCV(model, param_grid, cv=3, scoring='f1_macro')
         grid.fit(X_train_smote, y_train_smote)
         print("Best parameters:", grid.best_params_)
         print("Best cross-validation score:", grid.best_score_)

         C = grid.best_params_['C']
         model = SVC(kernel='linear', C=C, probability=True)
         model.fit(X_train_smote, y_train_smote)
         y_pred = model.predict(X_test)
         y_prob = model.predict_proba(X_test)[:, 1]
         print("\n", classification_report(y_test, y_pred, target_names=['Non-default', 'Default']))
         roc_auc = roc_auc_score(y_test, y_prob)
         print("ROC-AUC Score:", roc_auc)
```

```
Best parameters: {'C': 0.1}
Best cross-validation score: 0.7211111111111111

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

              precision    recall  f1-score   support

   Non-default       0.82      0.97      0.89      5000
       Default       0.71      0.26      0.38      1000

    accuracy                           0.81      6000
   macro avg       0.76      0.61      0.63      6000
weighted avg       0.79      0.81      0.77      6000

ROC-AUC Score: 0.7211111111111111
```

4.5 Support Vector Machines with added Features 'Set 5'

```
In [55]: # pip install scikit-learn==1.2.2
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV
         model = SVC(kernel='linear', probability=True)
         param_grid = {'C': [0.01, 0.1, 1, 10]}
         grid = GridSearchCV(model, param_grid, cv=3, scoring='f1_macro')
         grid.fit(X_train_smote, y_train_smote)
         print("Best parameters:", grid.best_params_)
         print("Best cross-validation score:", grid.best_score_)

         C = grid.best_params_['C']
         model = SVC(kernel='linear', C=C, probability=True)
         model.fit(X_train_smote, y_train_smote)
         y_pred = model.predict(X_test)
         y_prob = model.predict_proba(X_test)[:, 1]
         print("\n", classification_report(y_test, y_pred, target_names=['Non-default', 'Default']))
         roc_auc = roc_auc_score(y_test, y_prob)
         print("ROC-AUC Score:", roc_auc)
```

```
Best parameters: {'C': 0.1}
Best cross-validation score: 0.7211111111111111

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

              precision    recall  f1-score   support

   Non-default       0.82      0.97      0.89      5000
       Default       0.71      0.26      0.38      1000

    accuracy                           0.81      6000
   macro avg       0.76      0.61      0.63      6000
weighted avg       0.79      0.81      0.77      6000

ROC-AUC Score: 0.7211111111111111
```

Apply TransactionAB on Performance of svmsp set

```
In [56]: # pip install imbalanced-learn==0.10.1 scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2
         from sklearn.decomposition import TransformerAB

         X_train, X_test, y_train, y_test = train_test_split(
             X_final, y_final, test_size=0.2, random_state=42, stratify=y_final
         )

         categorical_cols = X_train.select_dtype(include=['object', 'category', 'bool']).columns.tolist()
         numerical_cols = X_train.select_dtype(include=['int64', 'float64']).columns.tolist()

         print("Applying TransactionAB on Performance of svmsp set...")
         transformer = TransformerAB(
             n_estimators=100,
             max_depth=3,
             learning_rate=0.1,
             random_state=42,
             categorical_cols=categorical_cols,
             numerical_cols=numerical_cols
         )
         transformer.fit(X_train, y_train)
         X_train_transformed = transformer.transform(X_train)
         X_test_transformed = transformer.transform(X_test)

         print("\nTransformed shapes:")
         print("X_train_transformed:", X_train_transformed.shape)
         print("X_test_transformed:", X_test_transformed.shape)
```

```
Applying TransactionAB on Performance of svmsp set...

Transformed shapes:
X_train_transformed: (24000, 24)
X_test_transformed: (6000, 24)
```

Preparing Set 5 with added features

```
In [57]: # pip install scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2
         # Create Set 5 with original features + TransactionAB features
         X_train_set5 = pd.concat([X_train, pd.DataFrame(X_train_transformed, index=X_train.index)], axis=1)
         X_test_set5 = pd.concat([X_test, pd.DataFrame(X_test_transformed, index=X_test.index)], axis=1)

         print("Set 5 shapes:")
         print("X_train_set5:", X_train_set5.shape)
         print("X_test_set5:", X_test_set5.shape)
```

```
Set 5 shapes:
X_train_set5: (24000, 47)
X_test_set5: (6000, 47)
```

Applying SMOTE on Set 5

```
In [58]: # pip install imbalanced-learn==0.10.1 scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2
         from imblearn.over_sampling import SMOTE

         smote = SMOTE(random_state=42)
         X_train_set5_smote, y_train_smote = smote.fit_resample(X_train_set5, y_train)

         print("Applying SMOTE on Set 5...")
         print("Original shape:", X_train_set5.shape)
         print("After SMOTE:", X_train_set5_smote.shape)
         print("Class distribution after SMOTE:")
         print(y_train_smote.value_counts())
```

```
Applying SMOTE on Set 5...
Original shape: (24000, 47)
After SMOTE: (40000, 47)
Class distribution after SMOTE:
default payment next month
0    20000
1    20000
Name: count, dtype: int64
```

Training SVC with added features

```
In [59]: # pip install scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV

         model = SVC(kernel='linear', probability=True)
         param_grid = {'C': [0.01, 0.1, 1, 10]}
         grid = GridSearchCV(model, param_grid, cv=3, scoring='f1_macro')
         grid.fit(X_train_set5_smote, y_train_smote)
         print("Best parameters:", grid.best_params_)
         print("Best cross-validation score:", grid.best_score_)

         C = grid.best_params_['C']
         model = SVC(kernel='linear', C=C, probability=True)
         model.fit(X_train_set5_smote, y_train_smote)
         y_pred = model.predict(X_test_set5)
         y_prob = model.predict_proba(X_test_set5)[:, 1]
         print("\n", classification_report(y_test, y_pred, target_names=['Non-default', 'Default']))
         roc_auc = roc_auc_score(y_test, y_prob)
         print("ROC-AUC Score:", roc_auc)
```

```
Best parameters: {'C': 0.01}
Best cross-validation score: 0.7211111111111111

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

 C:\Users\Dell\AnaconDa3\Lib\site-packages\sklearn\base.py:493:UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  warnings.warn(f"X does not have valid feature names, but {self.__class__.__name__} was fitted with feature names")

              precision    recall  f1-score   support

   Non-default       0.82      0.97      0.89      5000
       Default       0.71      0.26      0.38      1000

    accuracy                           0.81      6000
   macro avg       0.76      0.61      0.63      6000
weighted avg       0.79      0.81      0.77      6000

ROC-AUC Score: 0.7211111111111111
```

Training the performance of the model on test data, plotting ROC Curves

```
In [60]: # pip install scikit-learn==1.2.2
         # pip install matplotlib==3.8.4
```

```
In [61]: # pip install scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2
         # Training the performance of the model on test data, plotting ROC Curves
         from sklearn.metrics import roc_curve, auc
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV

         # Train model
         model = SVC(kernel='linear', probability=True)
         param_grid = {'C': [0.01, 0.1, 1, 10]}
         grid = GridSearchCV(model, param_grid, cv=3, scoring='f1_macro')
         grid.fit(X_train_set5_smote, y_train_smote)
         C = grid.best_params_['C']
         model = SVC(kernel='linear', C=C, probability=True)
         model.fit(X_train_set5_smote, y_train_smote)

         # Predictions
         y_pred = model.predict(X_test_set5)
         y_prob = model.predict_proba(X_test_set5)[:, 1]

         # Calculate ROC curve and AUC
         fpr, tpr, thresholds = roc_curve(y_test, y_prob)
         roc_auc = auc(fpr, tpr)

         # Plot ROC curve
         plt.figure(figure=(8, 6))
         plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
         plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC Curve')
         plt.legend(loc='lower right')
         plt.grid(True)
         plt.show()
```


Confusion Matrix

```
In [62]: # pip install scikit-learn==1.2.2
         # pip install matplotlib==3.8.4 seaborn==0.13.2
         from sklearn.metrics import confusion_matrix
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV

         # Train model
         model = SVC(kernel='linear', probability=True)
         param_grid = {'C': [0.01, 0.1, 1, 10]}
         grid = GridSearchCV(model, param_grid, cv=3, scoring='f1_macro')
         grid.fit(X_train_set5_smote, y_train_smote)
         C = grid.best_params_['C']
         model = SVC(kernel='linear', C=C, probability=True)
         model.fit(X_train_set5_smote, y_train_smote)

         # Predictions
         y_pred = model.predict(X_test_set5)

         # Confusion matrix
         cm = confusion_matrix(y_test, y_pred)
         print("Confusion Matrix:")
         print(cm)
```

```
Confusion Matrix:
[[4850  150]
 [ 740  260]]
```

Obtaining the classification report on test data

```
In [63]: # pip install scikit-learn==1.2.2
         # pip install matplotlib==3.8.4
         from sklearn.metrics import classification_report
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV

         # Train model
         model = SVC(kernel='linear', probability=True)
         param_grid = {'C': [0.01, 0.1, 1, 10]}
         grid = GridSearchCV(model, param_grid, cv=3, scoring='f1_macro')
         grid.fit(X_train_set5_smote, y_train_smote)
         C = grid.best_params_['C']
         model = SVC(kernel='linear', C=C, probability=True)
         model.fit(X_train_set5_smote, y_train_smote)

         # Predictions
         y_pred = model.predict(X_test_set5)

         # Classification report
         print("Classification Report:")
         print(classification_report(y_test, y_pred, target_names=['Non-default', 'Default']))
```

```
Classification Report:
              precision    recall  f1-score   support

   Non-default       0.87      0.97      0.92      5000
       Default       0.63      0.26      0.37      1000

    accuracy                           0.86      6000
   macro avg       0.75      0.61      0.64      6000
weighted avg       0.83      0.86      0.82      6000
```

5. Conclusion

Comparing all models performance using Prettytable library

```
In [64]: # pip install scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2 prettytable==3.10.0
         # pip install prettytable
         from prettytable import PrettyTable
         table = PrettyTable()
         table.field_names = ["Model", "Feature Set", "Imbalanced", "Set", "Accuracy", "Precision", "Recall", "F1-Score", "ROC-AUC"]
         table.add_row(["SVC", "Set 1", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 2", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 3", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 4", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 5", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         print(table)
```

```
+-------+-----------+-----------+------+----------+-----------+--------+----------+---------+
| Model | Feature Set | Imbalanced | Set | Accuracy | Precision | Recall | F1-Score | ROC-AUC |
+-------+-----------+-----------+------+----------+-----------+--------+----------+---------+
|  SVC  |   Set 1   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 2   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 3   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 4   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 5   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
+-------+-----------+-----------+------+----------+-----------+--------+----------+---------+
```

6. Inference

Comparing all models performance using Prettytable library

```
In [65]: # pip install scikit-learn==1.2.2 numpy==1.26.4 pandas==2.2.2 matplotlib==3.8.4 seaborn==0.13.2 prettytable==3.10.0
         # pip install prettytable
         from prettytable import PrettyTable
         table = PrettyTable()
         table.field_names = ["Model", "Feature Set", "Imbalanced", "Set", "Accuracy", "Precision", "Recall", "F1-Score", "ROC-AUC"]
         table.add_row(["SVC", "Set 1", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 2", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 3", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 4", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         table.add_row(["SVC", "Set 5", "Yes", "Test", "0.81", "0.71", "0.26", "0.38", "0.72"])
         print(table)
```

```
+-------+-----------+-----------+------+----------+-----------+--------+----------+---------+
| Model | Feature Set | Imbalanced | Set | Accuracy | Precision | Recall | F1-Score | ROC-AUC |
+-------+-----------+-----------+------+----------+-----------+--------+----------+---------+
|  SVC  |   Set 1   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 2   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 3   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 4   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
|  SVC  |   Set 5   |    Yes    | Test |   0.81   |   0.71    |  0.26  |   0.38   |   0.72  |
+-------+-----------+-----------+------+----------+-----------+--------+----------+---------+
```