This document provides an overview of the methodology we employed when answering Question 5 of the assignment. For additional details, please consult the corresponding python code contained in the submitted Jupyter notebook.

OVERVIEW OF DATASET

The supplied training dataset contains 30,000 examples (rows) representing online purchase orders. Each example has 44 attributes that describe the purchase (e.g., amount of the order, day of the week, payment method, etc.). The class (target) attribute is categorical and has two values:

- "yes" meaning high risk of payment default; and
- "no" meaning low risk.

The task is to use these features to predict the risk of payment default in the test dataset, which contains 20,000 examples with the same attributes. However, the class labels for the test dataset are not supplied.

A few additional aspects of the datasets are worth mentioning:

- The classes are quite imbalanced in the training dataset (i.e., “no” = 94.18% of training examples).

- Several features (e.g., 'ANUMMER_02', 'ANUMMER_03', 'ANUMMER_04', 'ANUMMER_05') are missing most of their data (i.e., >70% of the values are missing for that feature).

- Both the train and test datasets contain an ‘ORDER_ID’ feature which needs to be excluded from the model training but which must be included in the final output/prediction.

METHODOLOGY

Preprocessing Steps

Several of the preprocessing steps we employed are trivial and require little discussion. These include:

- Replacement of ‘?’ values with *np.NaN* which is the expected missing data value used by the *sklearn SimpleImputer*.

- Dropping the ‘ORDER_ID’ feature from the training dataset so that the model does not overfit to this feature. This feature is also removed from the test dataset but saved so that it can be combined with the predicted output.

- Splitting the training dataset in two: *X_train* (containing all the features) and *y_train* (the target classes).

There are also several other preprocessing steps that require more fulsome discussion:

- As noted, there are a number of features that are missing >70% of their data. Using what little data exists in these columns to fill the missing data would be spurious. Thus, we identify such features by examining the count of *np.NaN* (after replacement of '?' values) and dropping those features that have >70% missing data.

- Several ('DATE_LORDER', 'TIME_ORDER', 'B_BIRTHDATE') features have date/time time data that need to be converted to the correct format to be understood by Pandas. The 'TIME_ORDER' feature represents the time (hour:minutes) that an order was placed. We simplify this attribute to consider only the hour in which the order was placed.

After these steps are completed, we employ the preprocessing pipeline, which is available from *sklearn*. This allows us to create *SimpleImputer, StandardScaler, and OneHotEncoder* objects that can be passed to a *ColumnTransformer*, which does the filling of missing data, scaling of numerical features and one-hot encoding all together. As in Questions 1-4, we use pipeline to complete the following final preprocessing steps:

- We use the mean to fill missing numerical data and the mode (most frequent value) to fill missing categorical data.

- Numerical data are then rescaled by subtracting the mean and dividing by the standard deviation.

- Categorical data are converted to numerical/binary data using one-hot encoding.

Addressing Imbalance

We used two complementary strategies for addressing the imbalance classes in the dataset:

- First, as noted in the Question 5 description, the cost of misclassifying a 'yes' example is 10 times the cost of misclassifying a 'no' example (i.e., cost 50 vs. cost 5). Therefore, we applied class weights of 10 and 1 to the 'yes' and 'no' classes during model fitting. This encourages the model to fit preferentially to the 'yes' examples in the training dataset.

- Second, we resampled the training dataset to boost the relative proportion of the 'yes' examples. We did this by down-sampling the 'no' examples and keeping all of the 'yes' examples.

Feature Selection

Before selecting and assessing the performance of the various models, we trained a simple decision tree model on the training dataset. The purpose of this exercise was to select a reduced set of useful features that could be used in the training of the other models. Since the decision tree trains fairly quickly and provides a natural subset of the total set of features (i.e., only those features necessary to build the full tree), this was preferable to some other feature selection method (e.g., rank by mutual information, Chi-squared, etc.).

Algorithm Selection

As a starting point, we decided to consider the faster algorithms we explored in Questions 1-4 of this assignment. These included: random forest, decision tree, and logistic regression. We also included a simple k-nearest neighbours algorithm, in order to set a baseline level of performance to compare the other models against.

Each of these algorithms was trained on the training dataset, employing a k-folds technique (k=10) to refine the model fits and evaluate the performance. We also trained each model both with and without the class weights (i.e., ‘yes’ = 50, ‘no’ = 5). We then evaluated the models by examining the total confusion matrix (across all 10 folds) and the overall F1 score. Against these criteria, the best performing model was random forest with the class weights.

Confusion Matrix for Final Model

		Predicted	
		Yes	**No**
Actual	**Yes**	28254	0
	No	154	28100

F1 Score: 0.99728

With this model selected, we then used that single model to predict the target classes for the test dataset. These predictions are included in the "prediction.txt" file we submitted.