phData Case Study

(Technical Version)

Michael Rowlands

Tax Company's Challenge

For years, Tax Company has been unable to identify leads that will result in sales of their software.

This has resulted in thousands of dollars in unnecessary labor and advertising costs.

Tax Company has created a dataset containing two years of customer information and if they were successful at selling to each customer.

Our Objective

Investigate the dataset, and determine if a machine learning approach is viable.

If so, create a model to predict if a lead will convert.

Preliminary Assumptions

The dataset is historical so we assume future data is subject to the same data generating process.

Customers only appear once in the dataset.

Client has no prior knowledge on what features are predictive of sale.

There is a fixed profit for sale (true positives) and fixed loss for attempted sales (false positives).

Exploratory Analysis

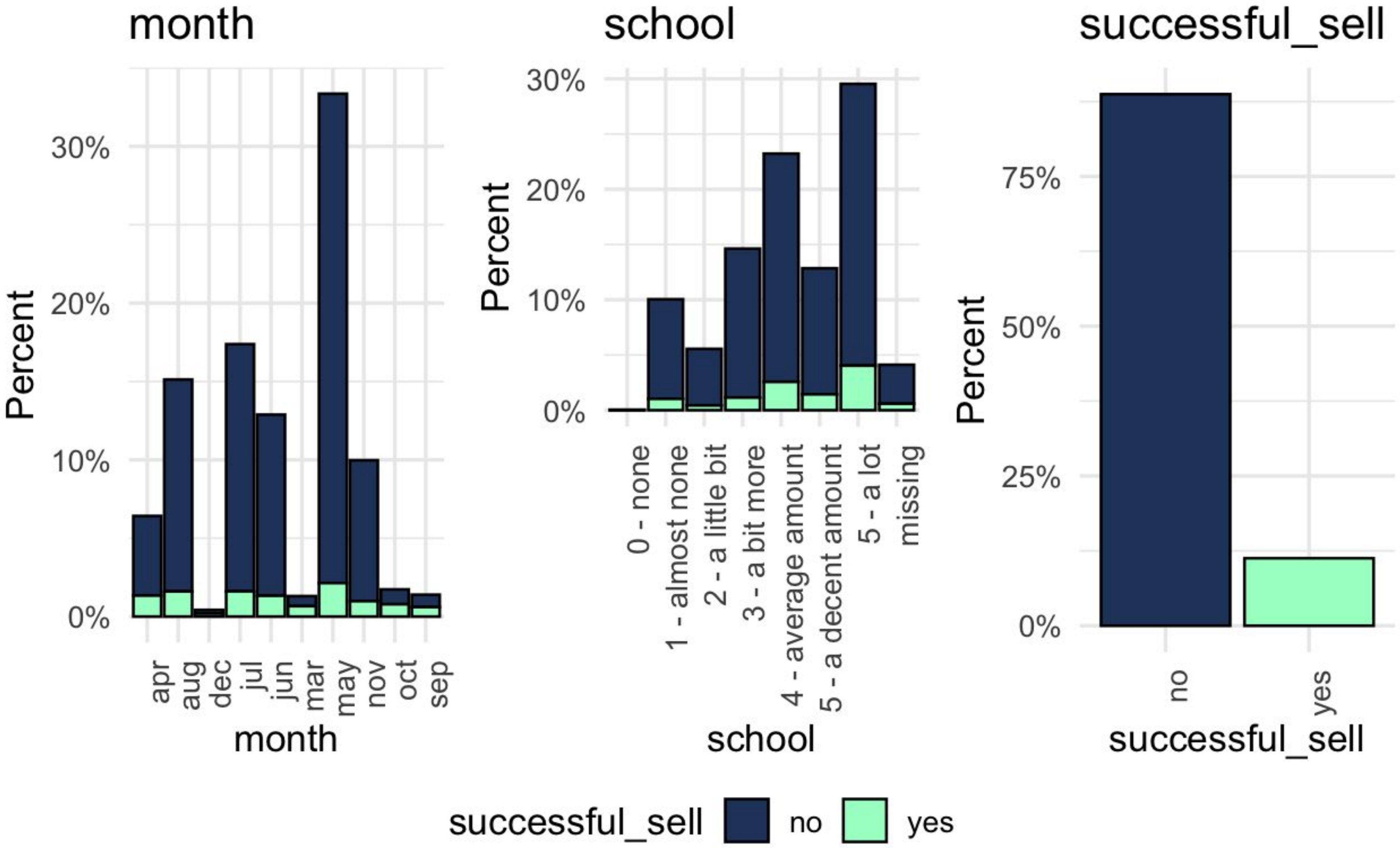
Basic Dataset Info

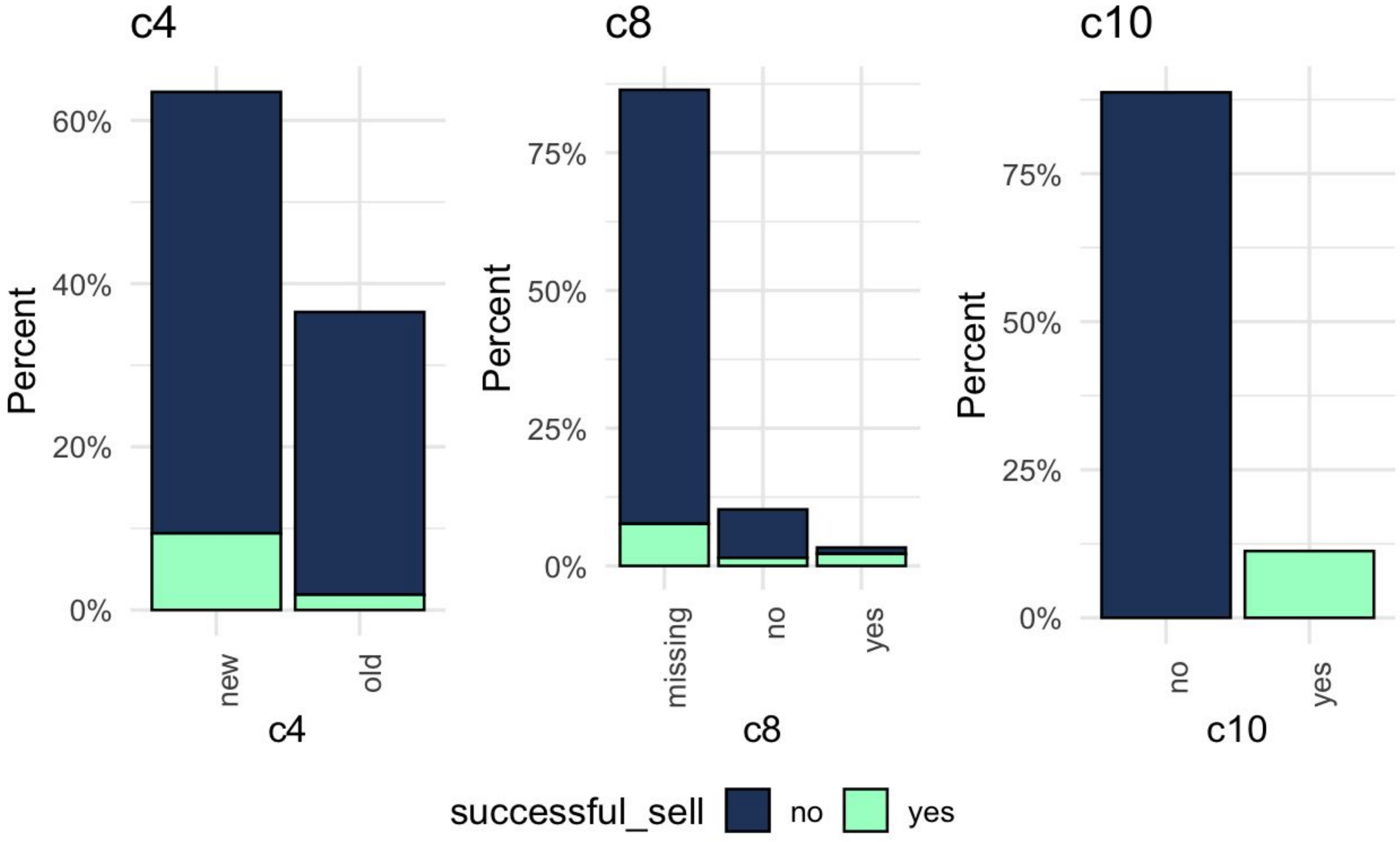
Full dataset contains 41,188 rows and 23 columns

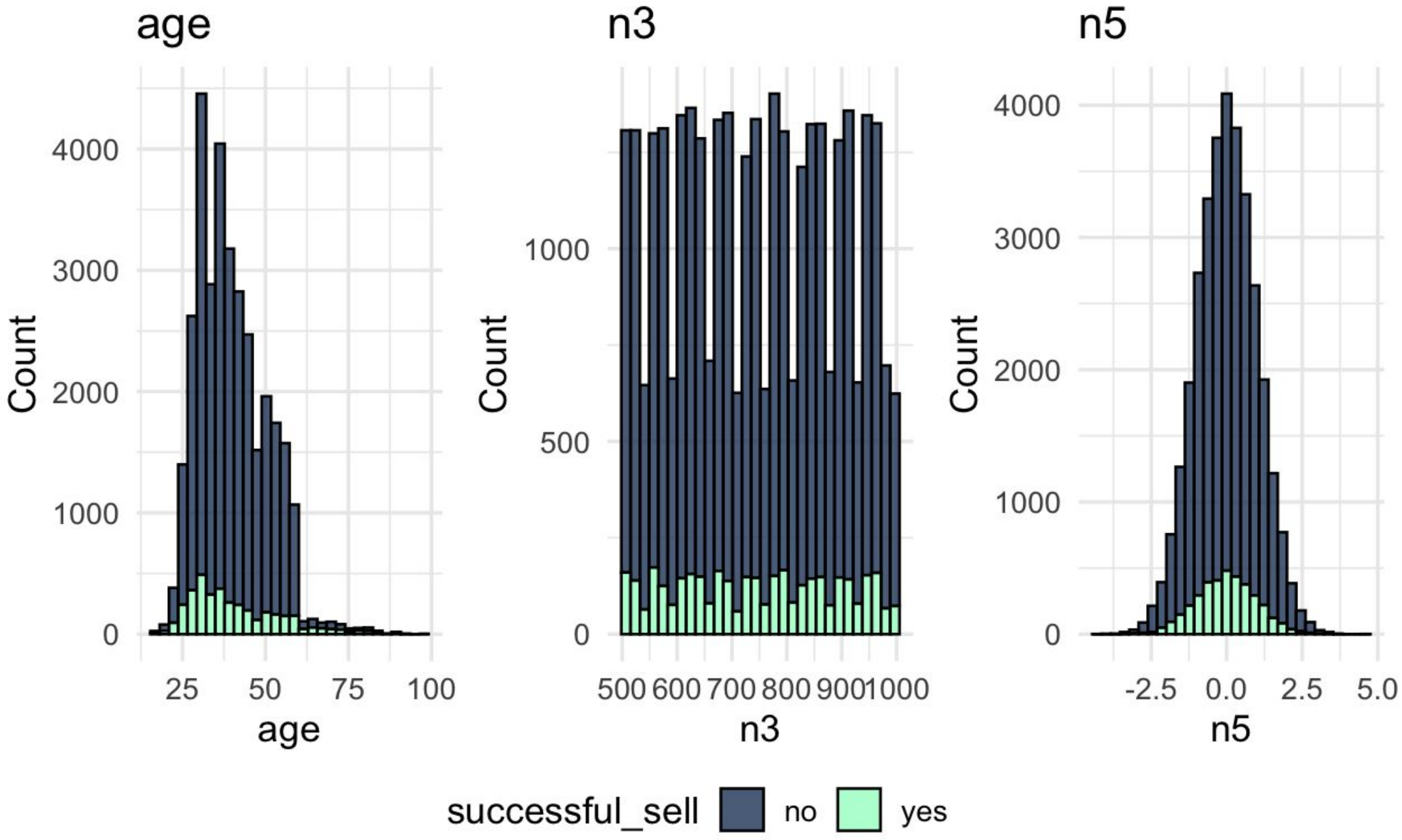
We pulled a test set of 20% and put it aside for later

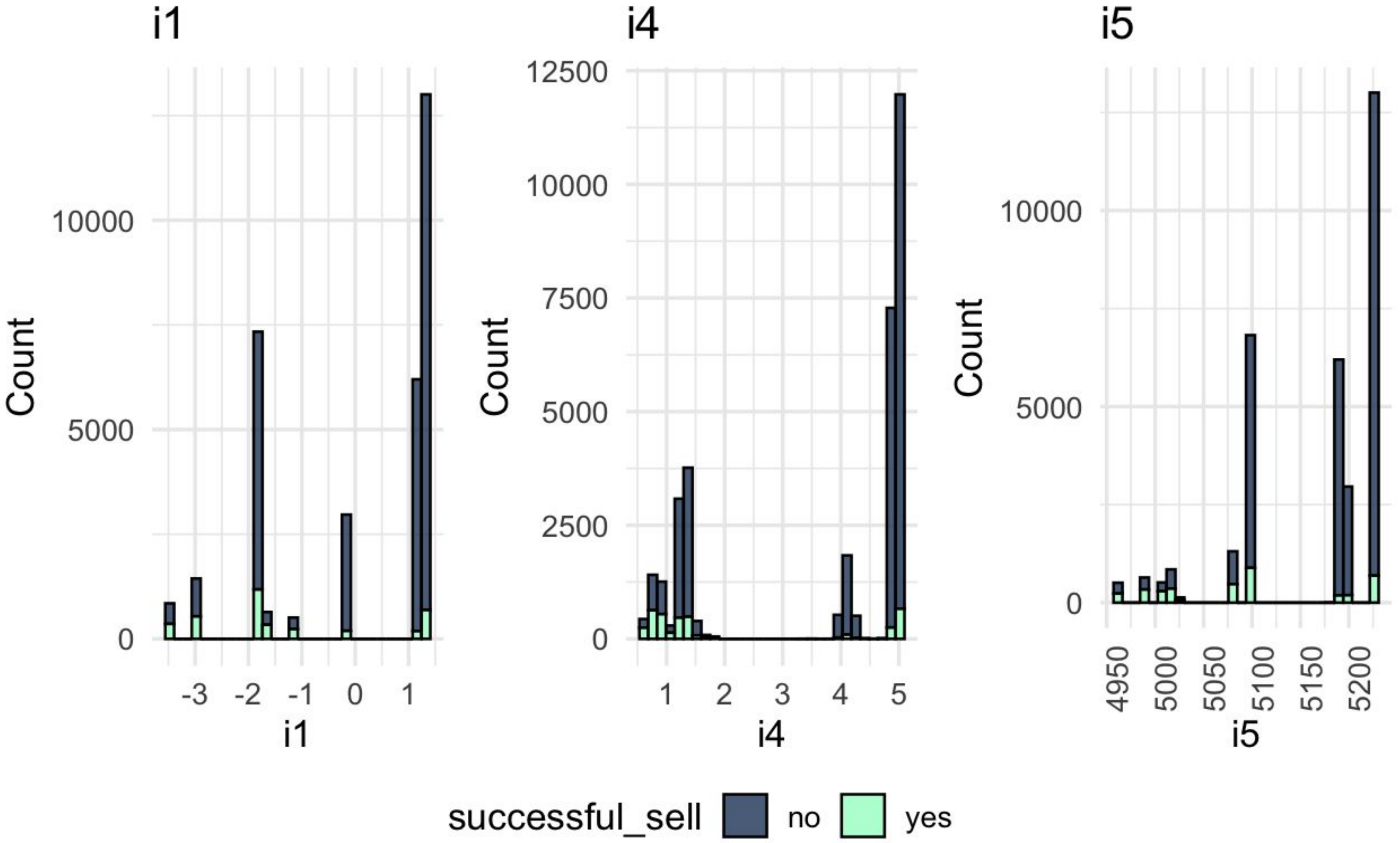
Target variable is “successful_sell”

About half the features are numeric and the other half are categorical

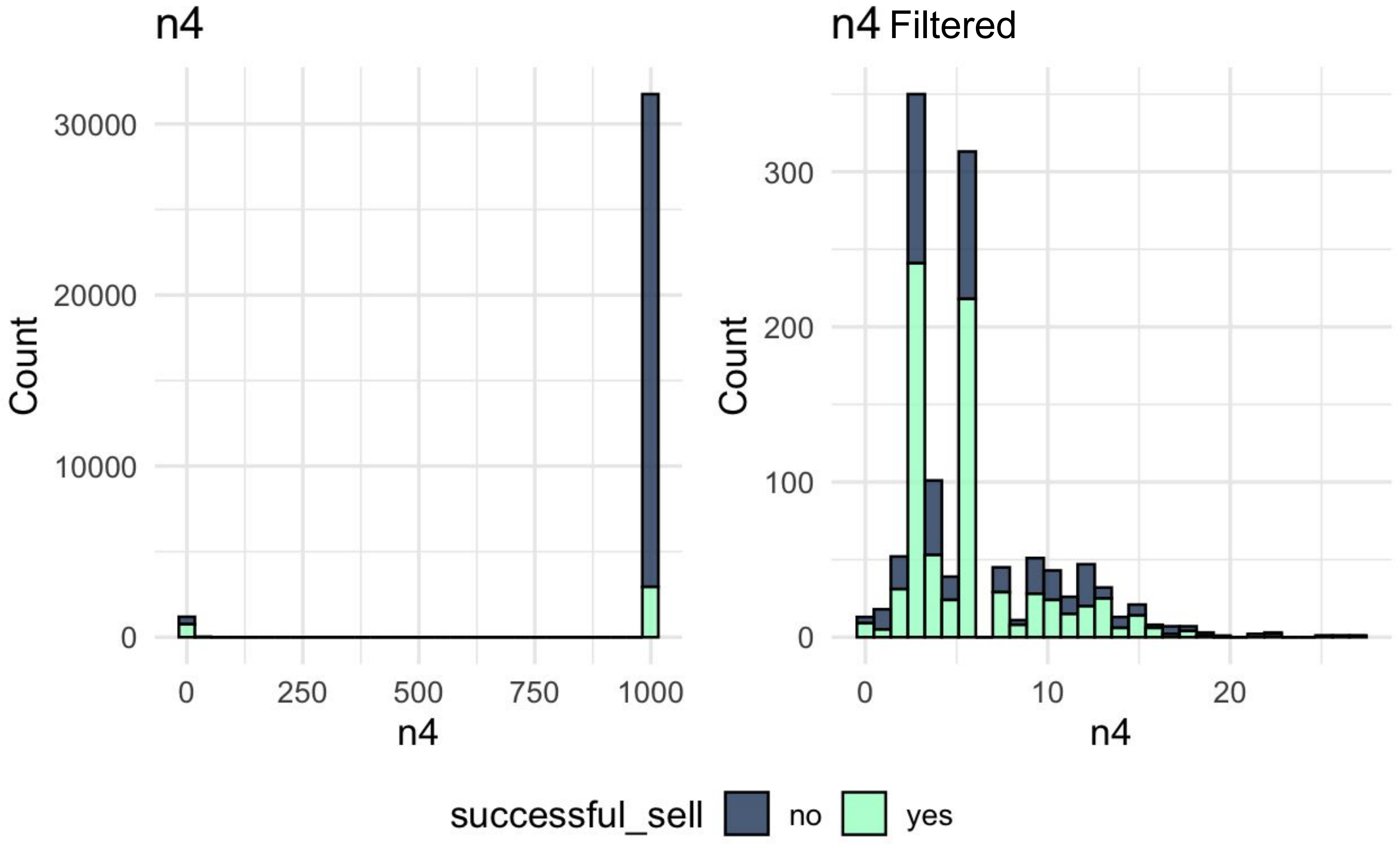

month
school
successful_sell
Percent
30%
20%
10%
0%
apr
aug
dec
jul
jun
mar
may
nov
oct
sep
month
0 - none
1 - almost none
2 - a little bit
3 - a bit more
4 - average amount
5 - a decent amount
5 - a lot
missing
school
75%
50%
25%
no
yes
successful_sell
successful_sell
no
yes


c4
c8
c10
60%
40%
20%
0%
75%
50%
25%
Percent
new
old
missing
no
yes
successful_sell
no
yes


age
n3
n5
Count
4000
3000
2000
1000
0
25
50
75
100
age
1000
500
0
500
600
700
800
900
1000
n3
4000
3000
2000
1000
0
-2.5
0.0
2.5
5.0
n5
successful_sell
no
yes

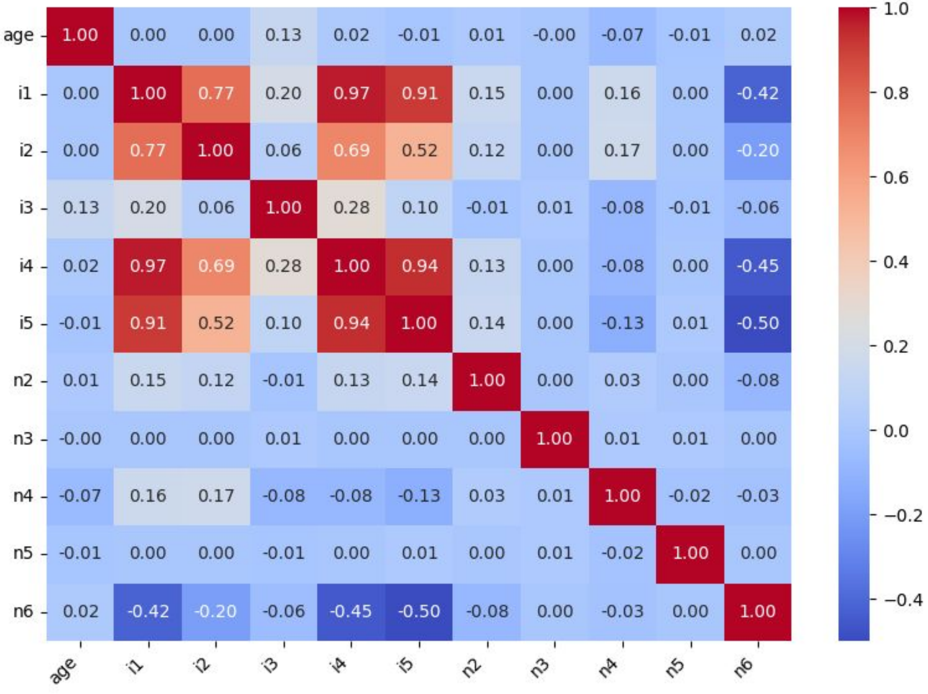


i1
i4
i5
Count
10000
5000
0
-3
-2
-1
0
1
12500
7500
2500
2
3
4
5
4950
5000
5050
5100
5150
5200
successful_sell
no
yes

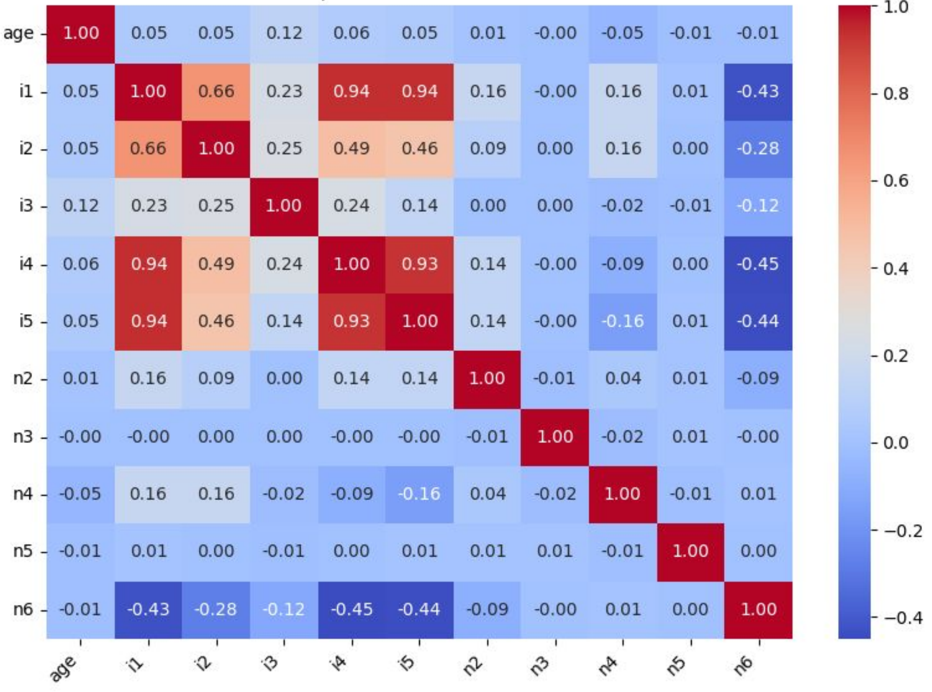


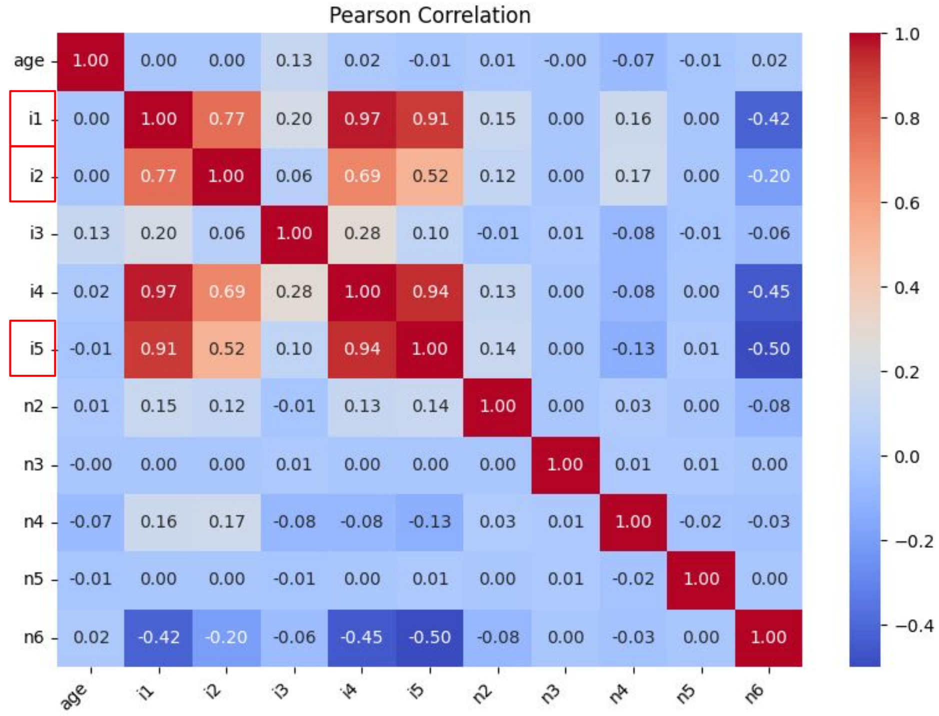
n4
n4 Filtered
Count
30000
20000
10000
0
0
250
500
750
1000
n4
Count
300
200
100
0
0
10
20
n4
successful_sell
no
yes

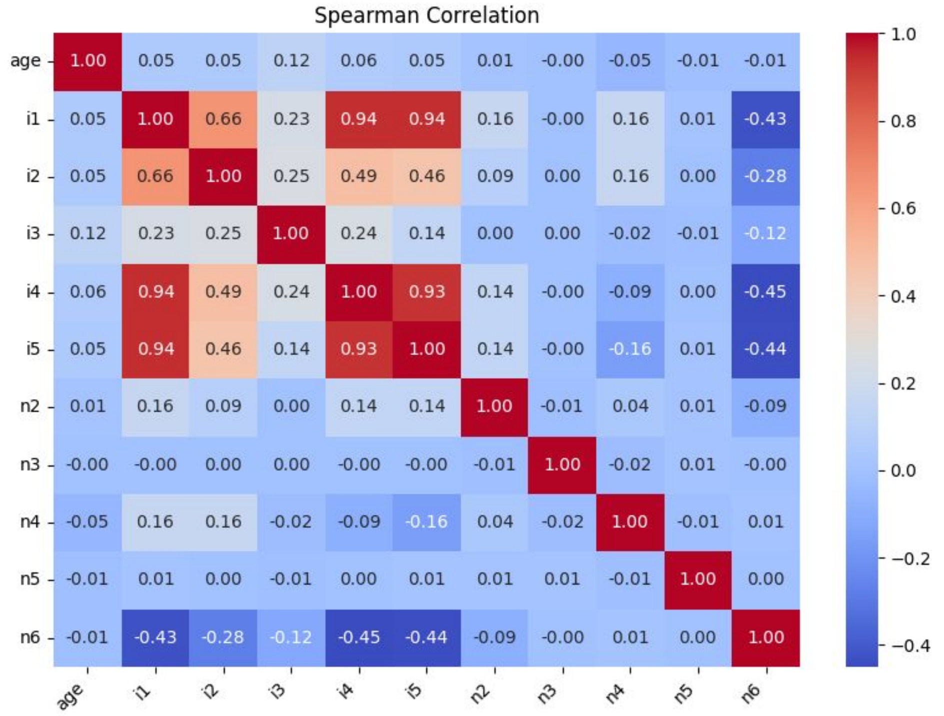
Pearson Correlation



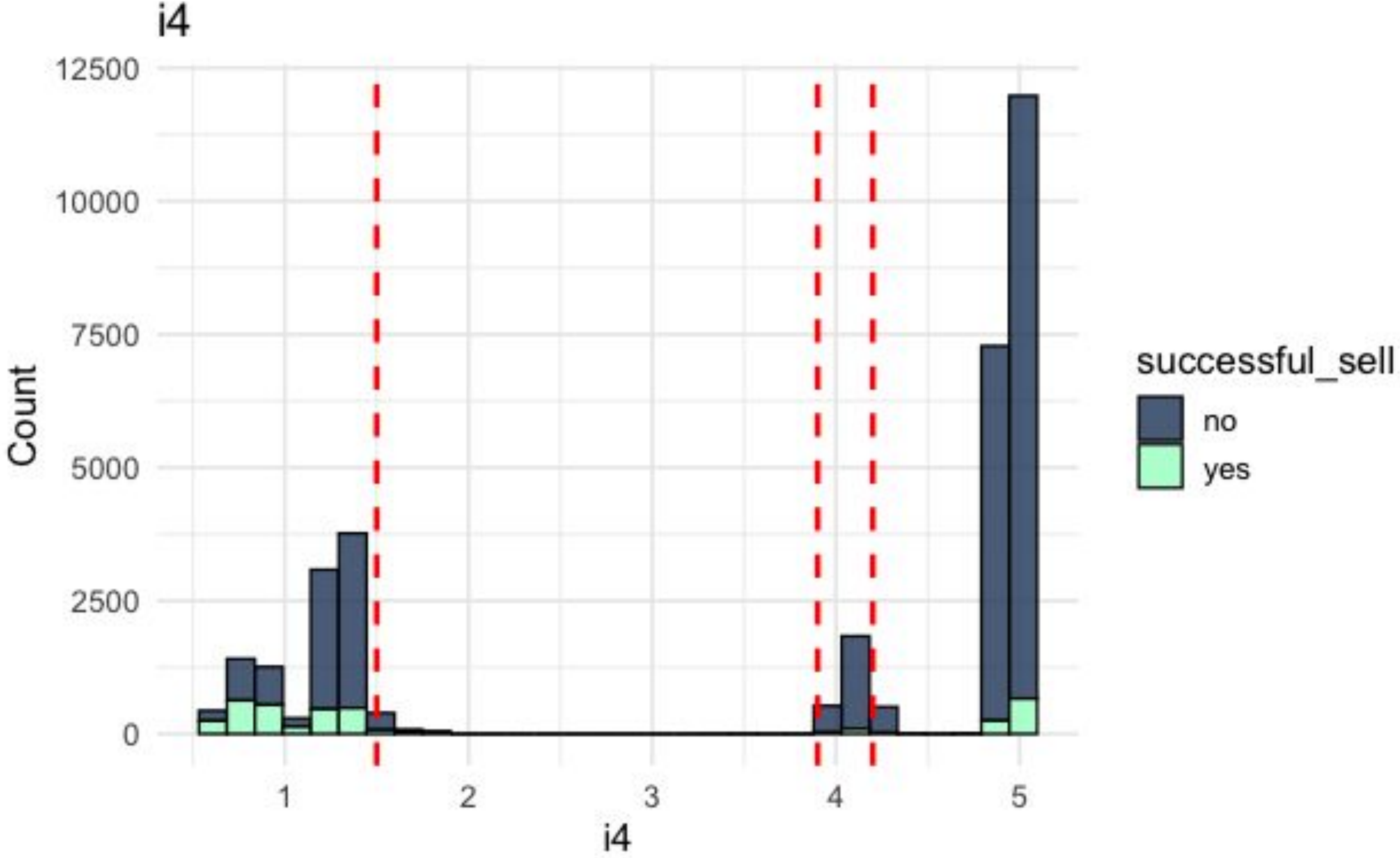
Spearman Correlation



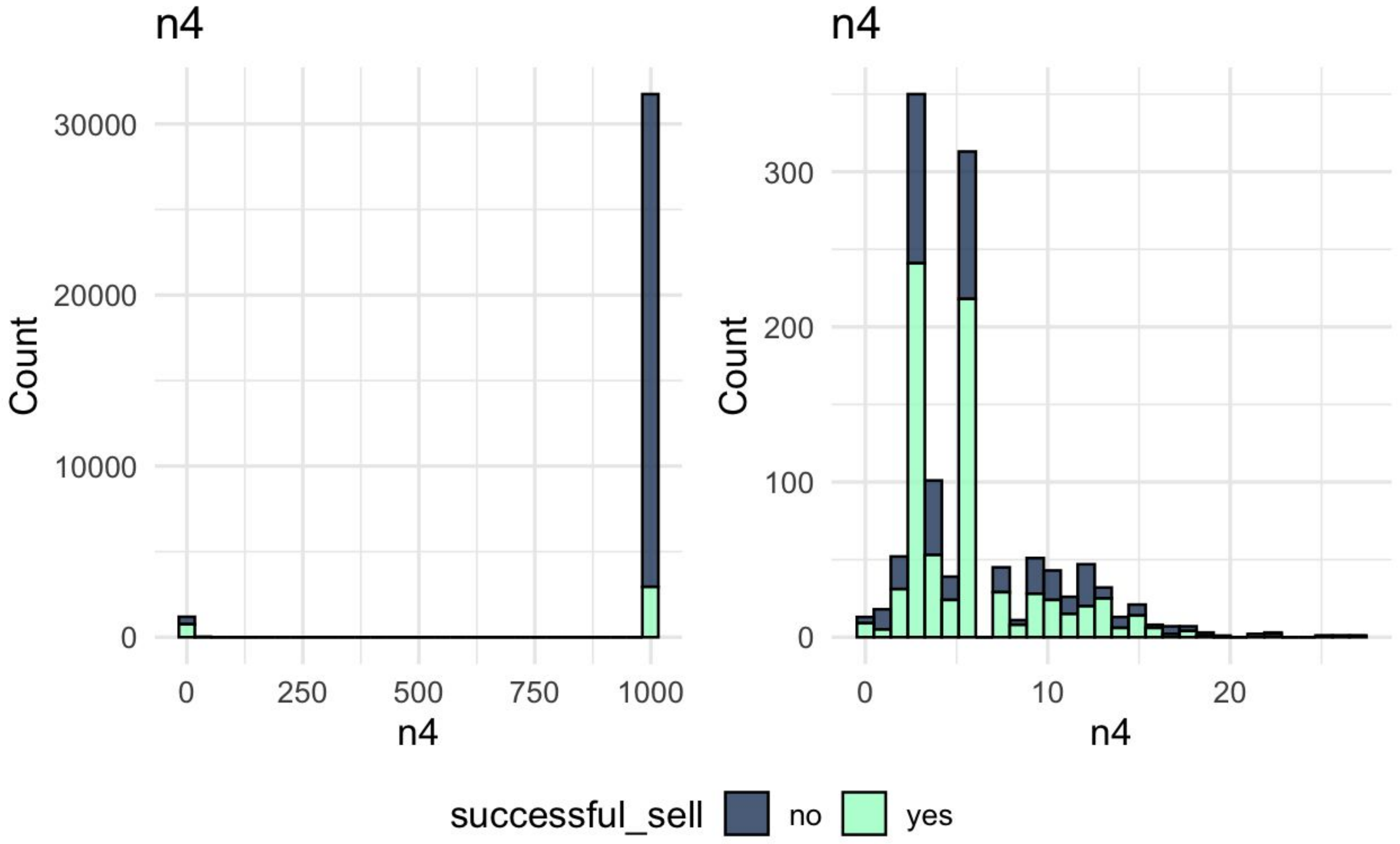

Pearson Correlation
age 1.00 0.00 0.00 0.13 0.02 -0.01 0.01 -0.00 -0.07 -0.01 0.02
i1 0.00 1.00 0.77 0.20 0.97 0.91 0.15 0.00 0.16 0.00 -0.42
i2 0.00 0.77 1.00 0.06 0.69 0.52 0.12 0.00 0.17 0.00 -0.20
i3 0.13 0.20 0.06 1.00 0.28 0.10 -0.01 0.01 -0.08 -0.01 -0.06
i4 0.02 0.97 0.69 0.28 1.00 0.94 0.13 0.00 -0.08 0.00 -0.45
i5 -0.01 0.91 0.52 0.10 0.94 1.00 0.14 0.00 -0.13 0.01 -0.50
n2 0.01 0.15 0.12 -0.01 0.13 0.14 1.00 0.00 0.03 0.00 -0.08
n3 -0.00 0.00 0.00 0.01 0.00 0.00 0.00 1.00 0.01 0.01 0.00
n4 -0.07 0.16 0.17 -0.08 -0.08 -0.13 0.03 0.01 1.00 -0.02 -0.03
n5 -0.01 0.00 0.00 -0.01 0.00 0.01 0.00 0.01 -0.02 1.00 0.00
n6 0.02 -0.42 -0.20 -0.06 -0.45 -0.50 -0.08 0.00 -0.03 0.00 1.00
age i1 i2 i3 i4 i5 n2 n3 n4 n5 n6
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4


Spearman Correlation
age 1.00 0.05 0.05 0.12 0.06 0.05 0.01 -0.00 -0.05 -0.01 -0.01
i1 0.05 1.00 0.66 0.23 0.94 0.94 0.16 -0.00 0.16 0.01 -0.43
i2 0.05 0.66 1.00 0.25 0.49 0.46 0.09 0.00 0.16 0.00 -0.28
i3 0.12 0.23 0.25 1.00 0.24 0.14 0.00 0.00 -0.02 -0.01 -0.12
i4 0.06 0.94 0.49 0.24 1.00 0.93 0.14 -0.00 -0.09 0.00 -0.45
i5 0.05 0.94 0.46 0.14 0.93 1.00 0.14 -0.00 -0.16 0.01 -0.44
n2 0.01 0.16 0.09 0.00 0.14 0.14 1.00 -0.01 0.04 0.01 -0.09
n3 -0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.01 1.00 -0.02 0.01 -0.00
n4 -0.05 0.16 0.16 -0.02 -0.09 -0.16 0.04 -0.02 1.00 -0.01 0.01
n5 -0.01 0.01 0.00 -0.01 0.00 0.01 0.01 0.01 -0.01 1.00 0.00
n6 -0.01 -0.43 -0.28 -0.12 -0.45 -0.44 -0.09 -0.00 0.01 0.00 1.00
age i1 i2 i3 i4 i5 n2 n3 n4 n5 n6
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4

Feature Engineering



i4
12500
10000
7500
5000
2500
0
Count
1
2
3
4
5
i4
successful_sell
no
yes


n4
n4
30000
20000
10000
0
Count
0
250
500
750
1000
n4
300
200
100
0
Count
0
10
20
n4
successful_sell
no
yes

Imputation

Almost all of the missing data is in the categorical columns (with the exception of possibly n4)

We will leave these values as a “missing” level

While this can introduce bias (since we are assuming all data is missing for the same reason), we will leave more sophisticated methods for imputation such as MICE for future analysis

Modeling and Results

```python
def evaluate_models_with_thresholds(
    models, X, y, preprocessor, n_splits=5, random_state=42,
    cost_params=None, thresholds=np.linspace(0.1, 0.9, 9), sampling_strategies=None
):
    """
    Evaluate multiple models with hyperparameter tuning using cross-validation,
    optimize classification thresholds for custom cost function, and calculate costs.

    Returns:
    - results: dict, evaluation metrics, costs, optimal thresholds, and predictions for each model
    """

    # Set default values of cost parameters and sampling strategies
    if cost_params is None:
        cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -5, "fn_cost": 0}

    if sampling_strategies is None:

        sampling_strategies = [
            None,
            RandomOverSampler(random_state=random_state),
            RandomUnderSampler(random_state=random_state)
        ]

    # Initialize cross-validation strategy and results dictionary
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    results = {}

    # Iterate over models and sampling strategies with hyperparameter tuning using GridSearchCV
    for model_name, model, param_grid in models:

        for sampler in sampling_strategies:
            sampler_name = sampler.__class__.__name__ if sampler else "NoSampler"
            pipeline = ImbPipeline([
                ("preprocessor", preprocessor),
                ("sampler", sampler if sampler else "passthrough"),
                ("model", model)
            ])

            grid_search = GridSearchCV(
                pipeline,
                param_grid={"model__" + key: value for key, value in param_grid.items()},
                cv=cv,
                scoring='average_precision',  # Use average precision for hyperparameter tuning
                n_jobs=-1,
                return_train_score=True
            )

            # Fit the GridSearchCV object
            grid_search.fit(X, y)

            # Pull best model from GridSearchCV using average precision as metric
            best_model = grid_search.best_estimator_

            # Calculate cross-validation score variation for the best model
            best_index = grid_search.best_index_
            cv_scores = grid_search.cv_results_['mean_test_score']
            cv_std = grid_search.cv_results_['std_test_score'][best_index]

            # Get cross-validated probabilities
            probs = cross_val_predict(best_model, X, y, cv=cv, method="predict_proba", n_jobs=-1)[:, 1]

            # Optimize threshold based on custom cost
            best_threshold = None
            best_cost = float('-inf')
            best_preds = None


            for threshold in thresholds:

                # If the probability is greater than the threshold, predict as positive
                thresholded_preds = (probs >= threshold).astype(int)
                cost = custom_cost(y, thresholded_preds, **cost_params)

                # If threshold increases profit, update best cost, threshold, and predictions
                if cost > best_cost:
                    best_cost = cost
                    best_threshold = threshold
                    best_preds = thresholded_preds

            # Pull feature importance from the best model if available
            feature_importance = None
            if hasattr(best_model.named_steps["model"], "feature_importances_"):
                feature_importance = best_model.named_steps["model"].feature_importances_
            elif hasattr(best_model.named_steps["model"], "coef_"):
                feature_importance = np.abs(best_model.named_steps["model"].coef_).flatten()

            # Confusion matrix
            confusion_mat = confusion_matrix(y, best_preds)

            # F1 score
            f1 = f1_score(y, best_preds)

            # Average precision
            average_precision = average_precision_score(y, probs)

            # Store results
            results[f"{model_name}_{sampler_name}"] = {
                "best_params": grid_search.best_params_,
                "cv_score_variation": cv_std,  # Standard deviation of CV scores
                "best_threshold": best_threshold,
                "total_profit": best_cost,
                "average_profit_per_sale_attempt": best_cost / (confusion_mat[1,1] + confusion_mat[0,1]),
                "feature_importance": feature_importance,
                "model": best_model,
                "confusion_matrix": confusion_mat,
                "F1": f1,
                "average_precision_score": average_precision,
                "probs": probs,  # Store probabilities
                "best_preds": best_preds,  # Store thresholded predictions
            }

    return results
```

```python
def evaluate_models_with_thresholds(
    models, X, y, preprocessor, n_splits=5, random_state=42,
    cost_params=None, thresholds=np.linspace(0.1, 0.9, 9), sampling_strategies=None
):
    """
    Evaluate multiple models with hyperparameter tuning using cross-validation,
    optimize classification thresholds for custom cost function, and calculate costs.

    Returns:
    - results: dict, evaluation metrics, costs, optimal thresholds, and predictions for each model
    """

    # Set default values of cost parameters and sampling strategies
    if cost_params is None:
        cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -5, "fn_cost": 0}

    if sampling_strategies is None:

        sampling_strategies = [
            None,
            RandomOverSampler(random_state=random_state),
            RandomUnderSampler(random_state=random_state)
        ]

    # Initialize cross-validation strategy and results dictionary
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    results = {}

    # Iterate over models and sampling strategies with hyperparameter tuning using GridSearchCV
    for model_name, model, param_grid in models:

        for sampler in sampling_strategies:
            sampler_name = sampler.__class__.__name__ if sampler else "NoSampler"
            pipeline = ImbPipeline([
                ("preprocessor", preprocessor),
                ("sampler", sampler if sampler else "passthrough"),
                ("model", model)
            ])

            grid_search = GridSearchCV(
                pipeline,
                param_grid={"model__" + key: value for key, value in param_grid.items()},
                cv=cv,
                scoring='average_precision',  # Use average precision for hyperparameter tuning
                n_jobs=-1,
                return_train_score=True
            )

            # Fit the GridSearchCV object
            grid_search.fit(X, y)

            # Pull best model from GridSearchCV using average precision as metric
            best_model = grid_search.best_estimator_

            # Calculate cross-validation score variation for the best model
            best_index = grid_search.best_index_
            cv_scores = grid_search.cv_results_['mean_test_score']
            cv_std = grid_search.cv_results_['std_test_score'][best_index]

            # Get cross-validated probabilities
            probs = cross_val_predict(best_model, X, y, cv=cv, method="predict_proba", n_jobs=-1)[:, 1]

            # Optimize threshold based on custom cost
            best_threshold = None
            best_cost = float('-inf')
            best_preds = None


            for threshold in thresholds:

                # If the probability is greater than the threshold, predict as positive
                thresholded_preds = (probs >= threshold).astype(int)
                cost = custom_cost(y, thresholded_preds, **cost_params)

                # If threshold increases profit, update best cost, threshold, and predictions
                if cost > best_cost:
                    best_cost = cost
                    best_threshold = threshold
                    best_preds = thresholded_preds

            # Pull feature importance from the best model if available
            feature_importance = None
            if hasattr(best_model.named_steps["model"], "feature_importances_"):
                feature_importance = best_model.named_steps["model"].feature_importances_
            elif hasattr(best_model.named_steps["model"], "coef_"):
                feature_importance = np.abs(best_model.named_steps["model"].coef_).flatten()

            # Confusion matrix
            confusion_mat = confusion_matrix(y, best_preds)

            # F1 score
            f1 = f1_score(y, best_preds)

            # Average precision
            average_precision = average_precision_score(y, probs)

            # Store results
            results[f"{model_name}_{sampler_name}"] = {
                "best_params": grid_search.best_params_,
                "cv_score_variation": cv_std,  # Standard deviation of CV scores
                "best_threshold": best_threshold,
                "total_profit": best_cost,
                "average_profit_per_sale_attempt": best_cost / (confusion_mat[1,1] + confusion_mat[0,1]),
                "feature_importance": feature_importance,
                "model": best_model,
                "confusion_matrix": confusion_mat,
                "F1": f1,
                "average_precision_score": average_precision,
                "probs": probs,  # Store probabilities
                "best_preds": best_preds,  # Store thresholded predictions
            }

    return results
```

```python
def evaluate_models_with_thresholds(
    models, X, y, preprocessor, n_splits=5, random_state=42,
    cost_params=None, thresholds=np.linspace(0.1, 0.9, 9), sampling_strategies=None
):
    """
    Evaluate multiple models with hyperparameter tuning using cross-validation,
    optimize classification thresholds for custom cost function, and calculate costs.

    Returns:
    - results: dict, evaluation metrics, costs, optimal thresholds, and predictions for each model
    """

    # Set default values of cost parameters and sampling strategies
    if cost_params is None:
        cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -5, "fn_cost": 0}

    if sampling_strategies is None:

        sampling_strategies = [
            None,
            RandomOverSampler(random_state=random_state),
            RandomUnderSampler(random_state=random_state)
        ]

    # Initialize cross-validation strategy and results dictionary
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    results = {}

    # Iterate over models and sampling strategies with hyperparameter tuning using GridSearchCV
    for model_name, model, param_grid in models:

        for sampler in sampling_strategies:
            sampler_name = sampler.__class__.__name__ if sampler else "NoSampler"
            pipeline = ImbPipeline([
                ("preprocessor", preprocessor),
                ("sampler", sampler if sampler else "passthrough"),
                ("model", model)
            ])

            grid_search = GridSearchCV(
                pipeline,
                param_grid={"model__" + key: value for key, value in param_grid.items()},
                cv=cv,
                scoring='average_precision',  # Use average precision for hyperparameter tuning
                n_jobs=-1,
                return_train_score=True
            )

            # Fit the GridSearchCV object
            grid_search.fit(X, y)

            # Pull best model from GridSearchCV using average precision as metric
            best_model = grid_search.best_estimator_

            # Calculate cross-validation score variation for the best model
            best_index = grid_search.best_index_
            cv_scores = grid_search.cv_results_['mean_test_score']
            cv_std = grid_search.cv_results_['std_test_score'][best_index]

            # Get cross-validated probabilities
            probs = cross_val_predict(best_model, X, y, cv=cv, method="predict_proba", n_jobs=-1)[:, 1]

            # Optimize threshold based on custom cost
            best_threshold = None
            best_cost = float('-inf')
            best_preds = None


            for threshold in thresholds:

                # If the probability is greater than the threshold, predict as positive
                thresholded_preds = (probs >= threshold).astype(int)
                cost = custom_cost(y, thresholded_preds, **cost_params)

                # If threshold increases profit, update best cost, threshold, and predictions
                if cost > best_cost:
                    best_cost = cost
                    best_threshold = threshold
                    best_preds = thresholded_preds

            # Pull feature importance from the best model if available
            feature_importance = None
            if hasattr(best_model.named_steps["model"], "feature_importances_"):
                feature_importance = best_model.named_steps["model"].feature_importances_
            elif hasattr(best_model.named_steps["model"], "coef_"):
                feature_importance = np.abs(best_model.named_steps["model"].coef_).flatten()

            # Confusion matrix
            confusion_mat = confusion_matrix(y, best_preds)

            # F1 score
            f1 = f1_score(y, best_preds)

            # Average precision
            average_precision = average_precision_score(y, probs)

            # Store results
            results[f"{model_name}_{sampler_name}"] = {
                "best_params": grid_search.best_params_,
                "cv_score_variation": cv_std,  # Standard deviation of CV scores
                "best_threshold": best_threshold,
                "total_profit": best_cost,
                "average_profit_per_sale_attempt": best_cost / (confusion_mat[1,1] + confusion_mat[0,1]),
                "feature_importance": feature_importance,
                "model": best_model,
                "confusion_matrix": confusion_mat,
                "F1": f1,
                "average_precision_score": average_precision,
                "probs": probs,  # Store probabilities
                "best_preds": best_preds,  # Store thresholded predictions
            }

    return results
```

```python
def evaluate_models_with_thresholds(
    models, X, y, preprocessor, n_splits=5, random_state=42,
    cost_params=None, thresholds=np.linspace(0.1, 0.9, 9), sampling_strategies=None
):
    """
    Evaluate multiple models with hyperparameter tuning using cross-validation,
    optimize classification thresholds for custom cost function, and calculate costs.

    Returns:
    - results: dict, evaluation metrics, costs, optimal thresholds, and predictions for each model
    """

    # Set default values of cost parameters and sampling strategies
    if cost_params is None:
        cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -5, "fn_cost": 0}

    if sampling_strategies is None:

        sampling_strategies = [
            None,
            RandomOverSampler(random_state=random_state),
            RandomUnderSampler(random_state=random_state)
        ]

    # Initialize cross-validation strategy and results dictionary
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    results = {}

    # Iterate over models and sampling strategies with hyperparameter tuning using GridSearchCV
    for model_name, model, param_grid in models:

        for sampler in sampling_strategies:
            sampler_name = sampler.__class__.__name__ if sampler else "NoSampler"
            pipeline = ImbPipeline([
                ("preprocessor", preprocessor),
                ("sampler", sampler if sampler else "passthrough"),
                ("model", model)
            ])

            grid_search = GridSearchCV(
                pipeline,
                param_grid={"model__" + key: value for key, value in param_grid.items()},
                cv=cv,
                scoring='average_precision',  # Use average precision for hyperparameter tuning
                n_jobs=-1,
                return_train_score=True
            )

            # Fit the GridSearchCV object
            grid_search.fit(X, y)

            # Pull best model from GridSearchCV using average precision as metric
            best_model = grid_search.best_estimator_

            # Calculate cross-validation score variation for the best model
            best_index = grid_search.best_index_
            cv_scores = grid_search.cv_results_['mean_test_score']
            cv_std = grid_search.cv_results_['std_test_score'][best_index]

            # Get cross-validated probabilities
            probs = cross_val_predict(best_model, X, y, cv=cv, method="predict_proba", n_jobs=-1)[:, 1]

            # Optimize threshold based on custom cost
            best_threshold = None
            best_cost = float('-inf')
            best_preds = None


            for threshold in thresholds:

                # If the probability is greater than the threshold, predict as positive
                thresholded_preds = (probs >= threshold).astype(int)
                cost = custom_cost(y, thresholded_preds, **cost_params)

                # If threshold increases profit, update best cost, threshold, and predictions
                if cost > best_cost:
                    best_cost = cost
                    best_threshold = threshold
                    best_preds = thresholded_preds

            # Pull feature importance from the best model if available
            feature_importance = None
            if hasattr(best_model.named_steps["model"], "feature_importances_"):
                feature_importance = best_model.named_steps["model"].feature_importances_
            elif hasattr(best_model.named_steps["model"], "coef_"):
                feature_importance = np.abs(best_model.named_steps["model"].coef_).flatten()

            # Confusion matrix
            confusion_mat = confusion_matrix(y, best_preds)

            # F1 score
            f1 = f1_score(y, best_preds)

            # Average precision
            average_precision = average_precision_score(y, probs)

            # Store results
            results[f"{model_name}_{sampler_name}"] = {
                "best_params": grid_search.best_params_,
                "cv_score_variation": cv_std,  # Standard deviation of CV scores
                "best_threshold": best_threshold,
                "total_profit": best_cost,
                "average_profit_per_sale_attempt": best_cost / (confusion_mat[1,1] + confusion_mat[0,1]),
                "feature_importance": feature_importance,
                "model": best_model,
                "confusion_matrix": confusion_mat,
                "F1": f1,
                "average_precision_score": average_precision,
                "probs": probs,  # Store probabilities
                "best_preds": best_preds,  # Store thresholded predictions
            }

    return results
```

```python
def evaluate_models_with_thresholds(
    models, X, y, preprocessor, n_splits=5, random_state=42,
    cost_params=None, thresholds=np.linspace(0.1, 0.9, 9), sampling_strategies=None
):
    """
    Evaluate multiple models with hyperparameter tuning using cross-validation,
    optimize classification thresholds for custom cost function, and calculate costs.

    Returns:
    - results: dict, evaluation metrics, costs, optimal thresholds, and predictions for each model
    """

    # Set default values of cost parameters and sampling strategies
    if cost_params is None:
        cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -5, "fn_cost": 0}

    if sampling_strategies is None:

        sampling_strategies = [
            None,
            RandomOverSampler(random_state=random_state),
            RandomUnderSampler(random_state=random_state)
        ]

    # Initialize cross-validation strategy and results dictionary
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    results = {}

    # Iterate over models and sampling strategies with hyperparameter tuning using GridSearchCV
    for model_name, model, param_grid in models:

        for sampler in sampling_strategies:
            sampler_name = sampler.__class__.__name__ if sampler else "NoSampler"
            pipeline = ImbPipeline([
                ("preprocessor", preprocessor),
                ("sampler", sampler if sampler else "passthrough"),
                ("model", model)
            ])

            grid_search = GridSearchCV(
                pipeline,
                param_grid={"model__" + key: value for key, value in param_grid.items()},
                cv=cv,
                scoring='average_precision',  # Use average precision for hyperparameter tuning
                n_jobs=-1,
                return_train_score=True
            )

            # Fit the GridSearchCV object
            grid_search.fit(X, y)

            # Pull best model from GridSearchCV using average precision as metric
            best_model = grid_search.best_estimator_

            # Calculate cross-validation score variation for the best model
            best_index = grid_search.best_index_
            cv_scores = grid_search.cv_results_['mean_test_score']
            cv_std = grid_search.cv_results_['std_test_score'][best_index]

            # Get cross-validated probabilities
            probs = cross_val_predict(best_model, X, y, cv=cv, method="predict_proba", n_jobs=-1)[:, 1]

            # Optimize threshold based on custom cost
            best_threshold = None
            best_cost = float('-inf')
            best_preds = None

            for threshold in thresholds:

                # If the probability is greater than the threshold, predict as positive
                thresholded_preds = (probs >= threshold).astype(int)
                cost = custom_cost(y, thresholded_preds, **cost_params)

                # If threshold increases profit, update best cost, threshold, and predictions
                if cost > best_cost:
                    best_cost = cost
                    best_threshold = threshold
                    best_preds = thresholded_preds
            # Pull feature importance from the best model if available
            feature_importance = None
            if hasattr(best_model.named_steps["model"], "feature_importances_"):
                feature_importance = best_model.named_steps["model"].feature_importances_
            elif hasattr(best_model.named_steps["model"], "coef_"):
                feature_importance = np.abs(best_model.named_steps["model"].coef_).flatten()

            # Confusion matrix
            confusion_mat = confusion_matrix(y, best_preds)

            # F1 score
            f1 = f1_score(y, best_preds)

            # Average precision
            average_precision = average_precision_score(y, probs)

            # Store results
            results[f"{model_name}_{sampler_name}"] = {
                "best_params": grid_search.best_params_,
                "cv_score_variation": cv_std,  # Standard deviation of CV scores
                "best_threshold": best_threshold,
                "total_profit": best_cost,
                "average_profit_per_sale_attempt": best_cost / (confusion_mat[1,1] + confusion_mat[0,1]),
                "feature_importance": feature_importance,
                "model": best_model,
                "confusion_matrix": confusion_mat,
                "F1": f1,
                "average_precision_score": average_precision,
                "probs": probs,  # Store probabilities
                "best_preds": best_preds,  # Store thresholded predictions
            }

    return results
```

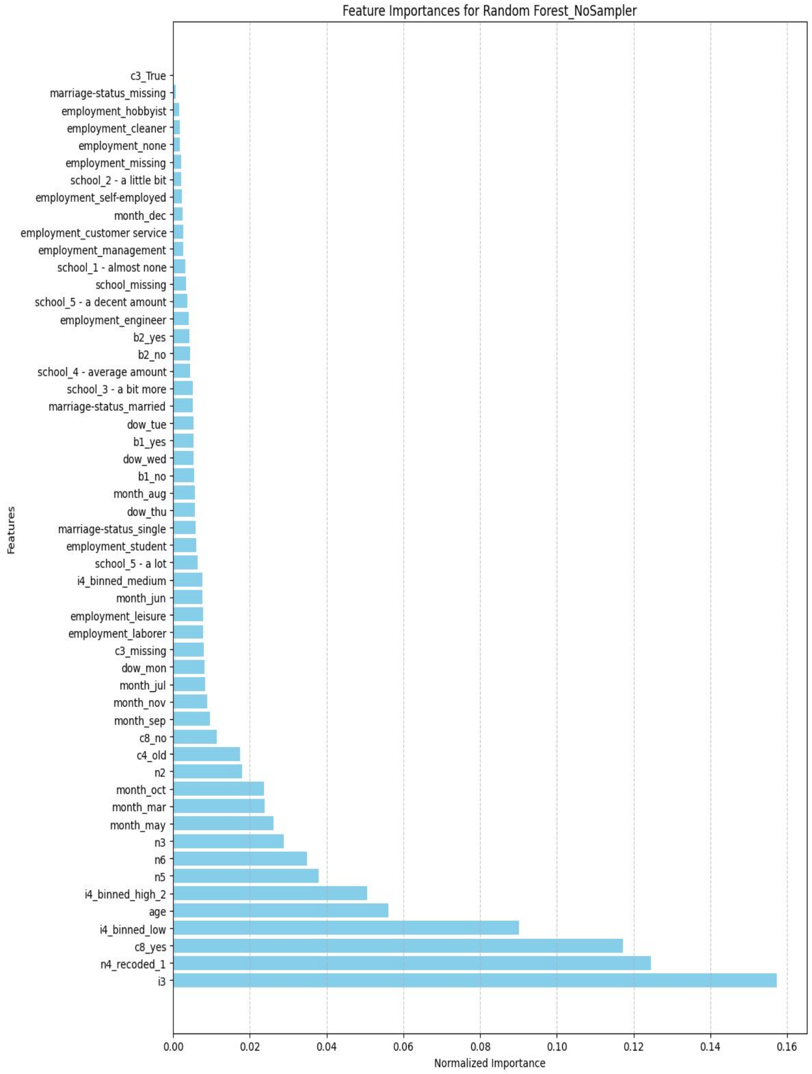
```python
def evaluate_models_with_thresholds(
    models, X, y, preprocessor, n_splits=5, random_state=42,
    cost_params=None, thresholds=np.linspace(0.1, 0.9, 9), sampling_strategies=None
):
    """
    Evaluate multiple models with hyperparameter tuning using cross-validation,
    optimize classification thresholds for custom cost function, and calculate costs.

    Returns:
    - results: dict, evaluation metrics, costs, optimal thresholds, and predictions for each model
    """

    # Set default values of cost parameters and sampling strategies
    if cost_params is None:
        cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -5, "fn_cost": 0}

    if sampling_strategies is None:

        sampling_strategies = [
            None,
            RandomOverSampler(random_state=random_state),
            RandomUnderSampler(random_state=random_state)
        ]

    # Initialize cross-validation strategy and results dictionary
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)
    results = {}

    # Iterate over models and sampling strategies with hyperparameter tuning using GridSearchCV
    for model_name, model, param_grid in models:

        for sampler in sampling_strategies:
            sampler_name = sampler.__class__.__name__ if sampler else "NoSampler"
            pipeline = ImbPipeline([
                ("preprocessor", preprocessor),
                ("sampler", sampler if sampler else "passthrough"),
                ("model", model)
            ])

            grid_search = GridSearchCV(
                pipeline,
                param_grid={"model__" + key: value for key, value in param_grid.items()},
                cv=cv,
                scoring='average_precision',  # Use average precision for hyperparameter tuning
                n_jobs=-1,
                return_train_score=True
            )

            # Fit the GridSearchCV object
            grid_search.fit(X, y)

            # Pull best model from GridSearchCV using average precision as metric
            best_model = grid_search.best_estimator_

            # Calculate cross-validation score variation for the best model
            best_index = grid_search.best_index_
            cv_scores = grid_search.cv_results_['mean_test_score']
            cv_std = grid_search.cv_results_['std_test_score'][best_index]

            # Get cross-validated probabilities
            probs = cross_val_predict(best_model, X, y, cv=cv, method="predict_proba", n_jobs=-1)[:, 1]

            # Optimize threshold based on custom cost
            best_threshold = None
            best_cost = float('-inf')
            best_preds = None


            for threshold in thresholds:

                # If the probability is greater than the threshold, predict as positive
                thresholded_preds = (probs >= threshold).astype(int)
                cost = custom_cost(y, thresholded_preds, **cost_params)

                # If threshold increases profit, update best cost, threshold, and predictions
                if cost > best_cost:
                    best_cost = cost
                    best_threshold = threshold
                    best_preds = thresholded_preds

            # Pull feature importance from the best model if available
            feature_importance = None
            if hasattr(best_model.named_steps["model"], "feature_importances_"):
                feature_importance = best_model.named_steps["model"].feature_importances_
            elif hasattr(best_model.named_steps["model"], "coef_"):
                feature_importance = np.abs(best_model.named_steps["model"].coef_).flatten()

            # Confusion matrix
            confusion_mat = confusion_matrix(y, best_preds)

            # F1 score
            f1 = f1_score(y, best_preds)

            # Average precision
            average_precision = average_precision_score(y, probs)

            # Store results
            results[f"{model_name}_{sampler_name}"] = {
                "best_params": grid_search.best_params_,
                "cv_score_variation": cv_std,  # Standard deviation of CV scores
                "best_threshold": best_threshold,
                "total_profit": best_cost,
                "average_profit_per_sale_attempt": best_cost / (confusion_mat[1,1] + confusion_mat[0,1]),
                "feature_importance": feature_importance,
                "model": best_model,
                "confusion_matrix": confusion_mat,
                "F1": f1,
                "average_precision_score": average_precision,
                "probs": probs,  # Store probabilities
                "best_preds": best_preds,  # Store thresholded predictions
            }

    return results
```

```python
models = [
    ("Dummy",
     DummyClassifier(strategy="constant", constant=1, random_state=42),
     {}),
    (
        "Random Forest",
        RandomForestClassifier(random_state=42),
        {"class_weight": ["balanced", None], "n_estimators": [50, 100, 500], "max_depth": [5, 10, None]}
    ),
    (
        "Penalized Logistic Regression",
        LogisticRegression(random_state=42, solver='liblinear'),
        {"C": [0.1, 1, 10], "penalty": ["l1", "l2"]}
    ),
    (
        "XGBoost",
        XGBClassifier(random_state=42, eval_metric="logloss"),
        {"n_estimators": [50, 100, 500], "max_depth": [3, 6, None], "learning_rate": [0.01, 0.1, 0.2], "reg_alpha": [0, .1, .5, 1, 10]}
    )
]


# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols_X),       # Scale numeric columns
        ("cat", OneHotEncoder(drop="first"), cat_cols_X)  # One-hot encode categorical columns
    ]
)


# Define profit for each type of prediction
cost_params = {"tp_cost": 100, "tn_cost": 0, "fp_cost": -15, "fn_cost": 0}


# Evaluate models with threshold optimization
results = evaluate_models_with_thresholds(models=models, X=X_train, y=y_train, preprocessor=preprocessor, cost_params=cost_params,
                                          thresholds=np.linspace(0.05, .95, 19),
                                          sampling_strategies=[None, RandomUnderSampler(random_state=42), RandomOverSampler(random_state=42)],
                                          n_splits=5, random_state=42)


# Display model diagnostics
model_diagnostics(results)
```

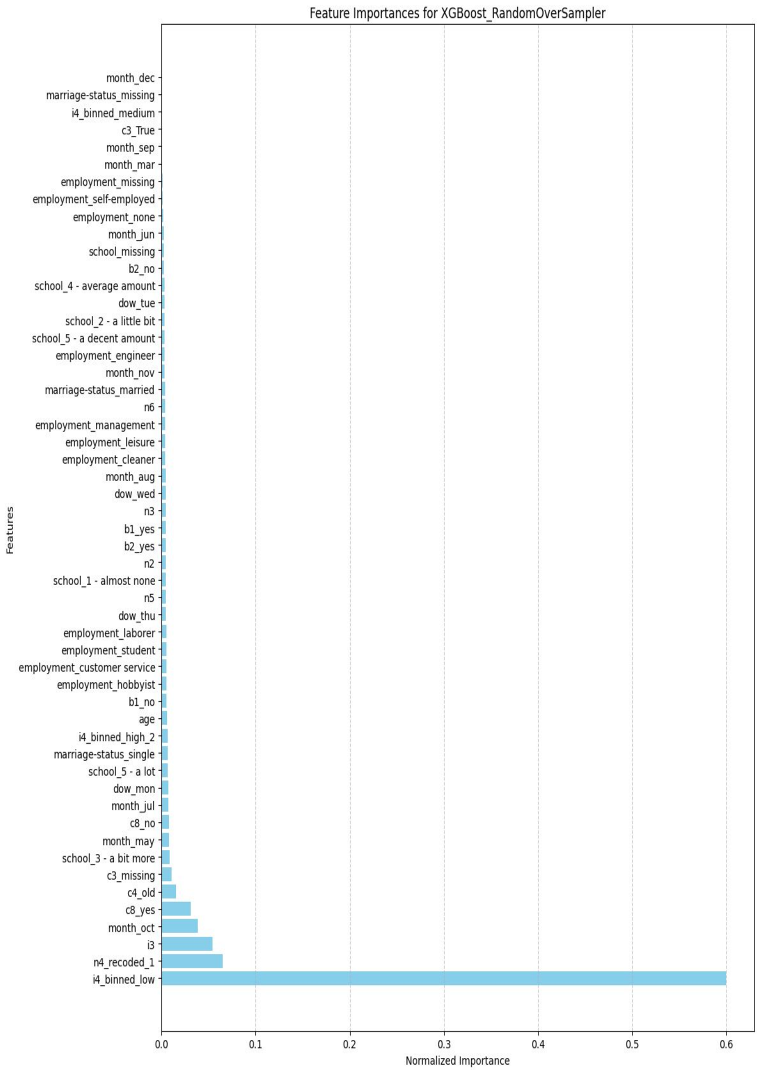
```
Model: XGBoost_RandomOverSampler
Best Parameters: {'model__learning_rate': 0.1, 'model__max_depth': 6, 'model__n_estimators': 50, 'model__reg_alpha': 10}
Total Profit: 175475.00
Average Profit per Sale Attempts: 31.39
Best Threshold: 0.55
Confusion Matrix:
[[25903  3335]
 [ 1457  2255]]
F1 Score: 0.48
Average Precision Score: 0.46
CV Average Precision Variation (std): 0.013
--------------------------------------------------
Model: Random Forest_NoSampler
Best Parameters: {'model__class_weight': None, 'model__max_depth': 10, 'model__n_estimators': 500}
Total Profit: 175395.00
Average Profit per Sale Attempts: 30.67
Best Threshold: 0.15
Confusion Matrix:
[[25791  3447]
 [ 1441  2271]]
F1 Score: 0.48
Average Precision Score: 0.46
CV Average Precision Variation (std): 0.015
--------------------------------------------------
--------------------------------------------------
Model: Dummy_RandomOverSampler
Best Parameters: {}
Total Profit: -67370.00
Average Profit per Sale Attempts: -2.04
Best Threshold: 0.05
Confusion Matrix:
[[    0 29238]
 [    0  3712]]
F1 Score: 0.20
Average Precision Score: 0.11
CV Average Precision Variation (std): 0.000
--------------------------------------------------
```

```
Model: XGBoost_RandomOverSampler
Best Parameters: {'model__learning_rate': 0.1, 'model__max_depth': 6, 'model__n_estimators': 50, 'model__reg_alpha': 10}
Total Profit: 175475.00
Average Profit per Sale Attempts: 31.39
Best Threshold: 0.55
Confusion Matrix:
[[25903  3335]
 [ 1457  2255]]
F1 Score: 0.48
Average Precision Score: 0.46
CV Average Precision Variation (std): 0.013
--------------------------------------------------
Model: Random Forest_NoSampler
Best Parameters: {'model__class_weight': None, 'model__max_depth': 10, 'model__n_estimators': 500}
Total Profit: 175395.00
Average Profit per Sale Attempts: 30.67
Best Threshold: 0.15
Confusion Matrix:
[[25791  3447]
 [ 1441  2271]]
F1 Score: 0.48
Average Precision Score: 0.46
CV Average Precision Variation (std): 0.015
--------------------------------------------------
--------------------------------------------------
Model: Dummy_RandomOverSampler
Best Parameters: {}
Total Profit: -67370.00
Average Profit per Sale Attempts: -2.04
Best Threshold: 0.05
Confusion Matrix:
[[    0 29238]
 [    0  3712]]
F1 Score: 0.20
Average Precision Score: 0.11
CV Average Precision Variation (std): 0.000
--------------------------------------------------
```

```
Model: XGBoost_RandomOverSampler
Best Parameters: {'model__learning_rate': 0.1, 'model__max_depth': 6, 'model__n_estimators': 50, 'model__reg_alpha': 10}
Total Profit: 175475.00
Average Profit per Sale Attempts: 31.39
Best Threshold: 0.55
Confusion Matrix:
[[25903  3335]
 [ 1457  2255]]
F1 Score: 0.48
Average Precision Score: 0.46
CV Average Precision Variation (std): 0.013
--------------------------------------------------
Model: Random Forest_NoSampler
Best Parameters: {'model__class_weight': None, 'model__max_depth': 10, 'model__n_estimators': 500}
Total Profit: 175395.00
Average Profit per Sale Attempts: 30.67
Best Threshold: 0.15
Confusion Matrix:
[[25791  3447]
 [ 1441  2271]]
F1 Score: 0.48
Average Precision Score: 0.46
CV Average Precision Variation (std): 0.015
--------------------------------------------------
--------------------------------------------------
Model: Dummy_RandomOverSampler
Best Parameters: {}
Total Profit: -67370.00
Average Profit per Sale Attempts: -2.04
Best Threshold: 0.05
Confusion Matrix:
[[    0 29238]
 [    0  3712]]
F1 Score: 0.20
Average Precision Score: 0.11
CV Average Precision Variation (std): 0.000
--------------------------------------------------
```

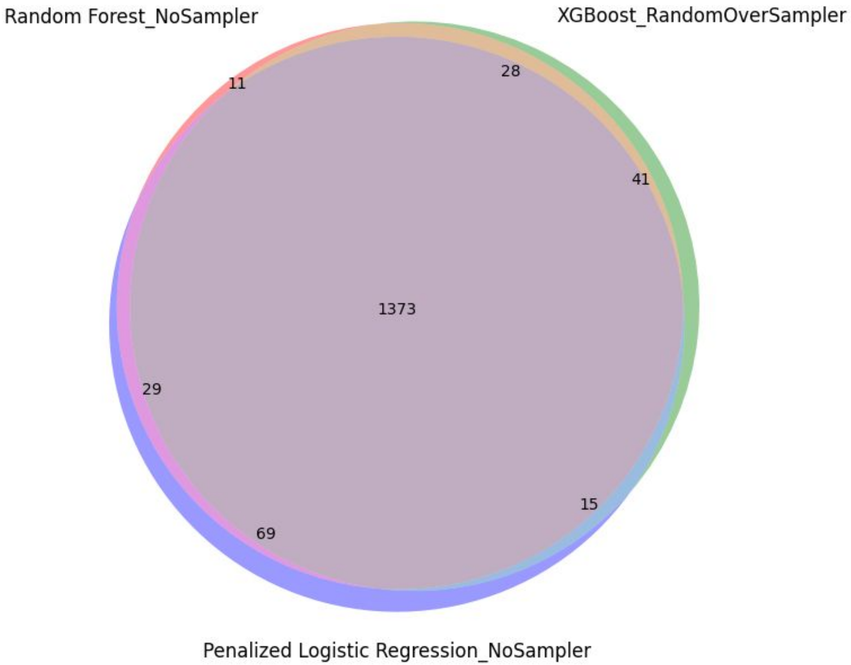


Feature Importances for Random Forest_NoSampler
Features
Normalized Importance
c3_True
marriage-status_missing
employment_hobbyist
employment_cleaner
employment_none
employment_missing
school_2 - a little bit
employment_self-employed
month_dec
employment_customer service
employment_management
school_1 - almost none
school_missing
school_5 - a decent amount
employment_engineer
b2_yes
b2_no
school_4 - average amount
school_3 - a bit more
marriage-status_married
dow_tue
b1_yes
dow_wed
b1_no
month_aug
dow_thu
marriage-status_single
employment_student
school_5 - a lot
i4_binned_medium
month_jun
employment_leisure
employment_laborer
c3_missing
dow_mon
month_jul
month_nov
month_sep
c8_no
c4_old
n2
month_oct
month_mar
month_may
n3
n6
n5
i4_binned_high_2
age
i4_binned_low
c8_yes
n4_recoded_1
i3
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16

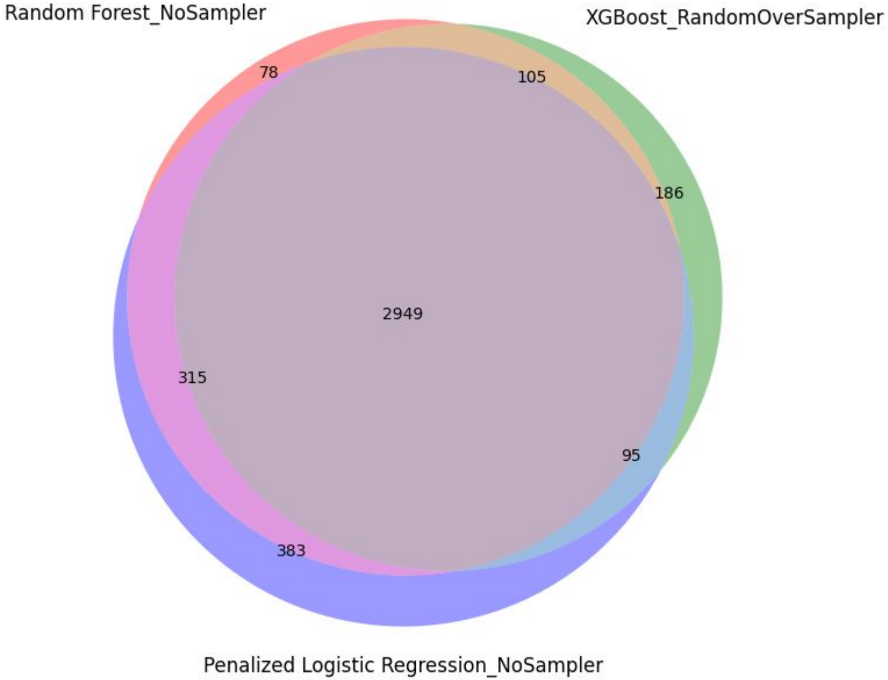


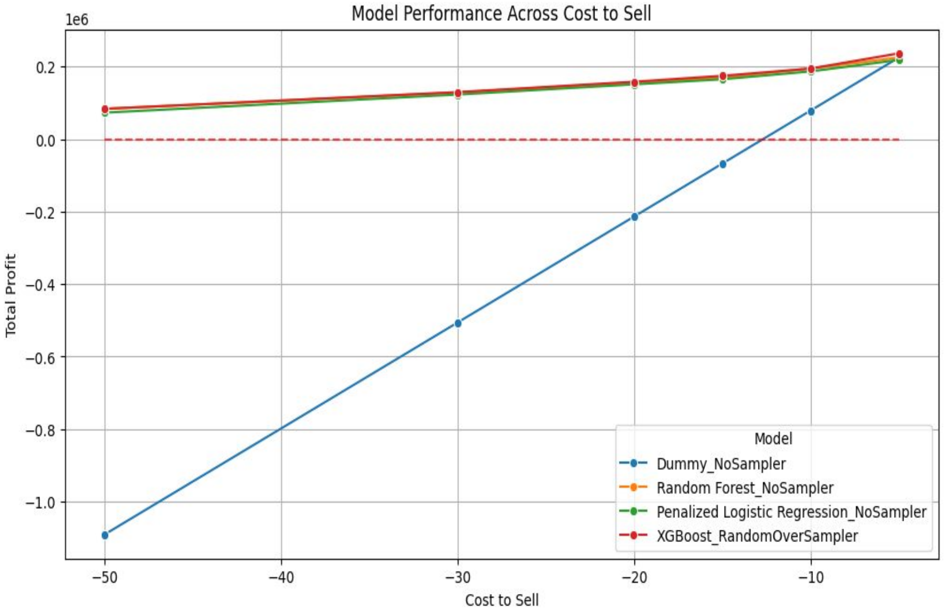
Feature Importances for XGBoost_RandomOverSampler
Features
Normalized Importance
month_dec
marriage-status_missing
i4_binned_medium
c3_True
month_sep
month_mar
employment_missing
employment_self-employed
employment_none
month_jun
school_missing
b2_no
school_4 - average amount
dow_tue
school_2 - a little bit
school_5 - a decent amount
employment_engineer
month_nov
marriage-status_married
n6
employment_management
employment_leisure
employment_cleaner
month_aug
dow_wed
n3
b1_yes
b2_yes
n2
school_1 - almost none
n5
dow_thu
employment_laborer
employment_student
employment_customer service
employment_hobbyist
b1_no
age
i4_binned_high_2
marriage-status_single
school_5 - a lot
dow_mon
month_jul
c8_no
month_may
school_3 - a bit more
c3_missing
c4_old
c8_yes
month_oct
i3
n4_recoded_1
i4_binned_low
0.0
0.1
0.2
0.3
0.4
0.5
0.6

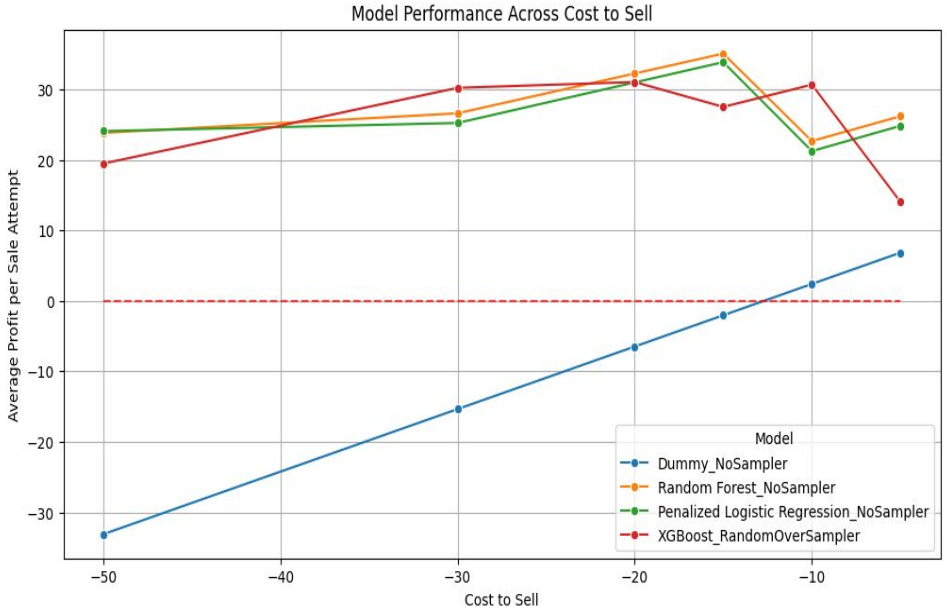
Overlap of Misclassified Instances for True Class 1



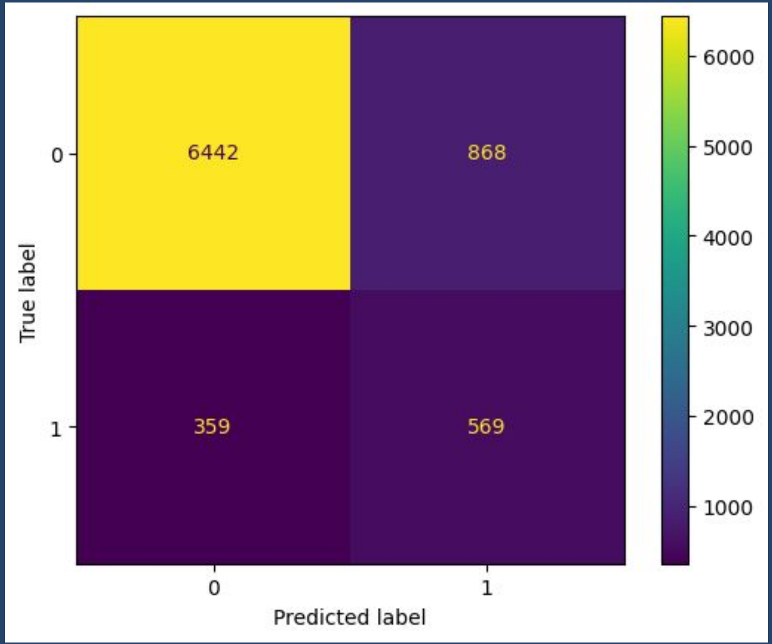
Overlap of Misclassified Instances for True Class 0




Model Performance Across Cost to Sell
1e6
Total Profit
Cost to Sell
Model
Dummy_NoSampler
Random Forest_NoSampler
Penalized Logistic Regression_NoSampler
XGBoost_RandomOverSampler


Model Performance Across Cost to Sell
Average Profit per Sale Attempt
Cost to Sell
Model
Dummy_NoSampler
Random Forest_NoSampler
Penalized Logistic Regression_NoSampler
XGBoost_RandomOverSampler

```python
X_test, y_test = preprocess_tax_df(df_test)



cost_params = {'tp_cost': 100, 'tn_cost': 0, 'fp_cost': -15, 'fn_cost': 0}

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols_X),       # Scale numeric columns
        ("cat", OneHotEncoder(drop="first"), cat_cols_X)  # One-hot encode categorical columns
    ]
)

best_model_name = 'Random Forest_NoSampler'
best_result = results[best_model_name]
best_params = best_result['best_params']
best_threshold = best_result['best_threshold']


# Extract model hyperparameters
model_params = {key.replace('model__', ''): value for key, value in best_params.items() if key.startswith('model__')}

# 2. Reconstruct the Pipeline
# Initialize the model with the best hyperparameters
best_model = RandomForestClassifier(random_state=42, **model_params)

# Build the pipeline steps
pipeline_steps = [
    ('preprocessor', preprocessor)
]

# Add the model to the pipeline
pipeline_steps.append(('model', best_model))

# Create the pipeline
pipeline = ImbPipeline(steps=pipeline_steps)

# 3. Train on the Full Training Set
pipeline.fit(X_train, y_train)

# 4. Evaluate on the Test Set
# Get predicted probabilities for the positive class
probs_test = pipeline.predict_proba(X_test)[:, 1]

# Apply the optimal threshold to get binary predictions
y_pred_test = (probs_test >= best_threshold).astype(int)

# Calculate the confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred_test)
ConfusionMatrixDisplay(confusion_mat).plot()
```



Random Forest:

Profit: $43,880

Profit per Attempted Sale: $30.54

Average Precision: 0.45

Current Business Practice:

Profit: -$16,850

Profit per Attempted Sale: -$2.05

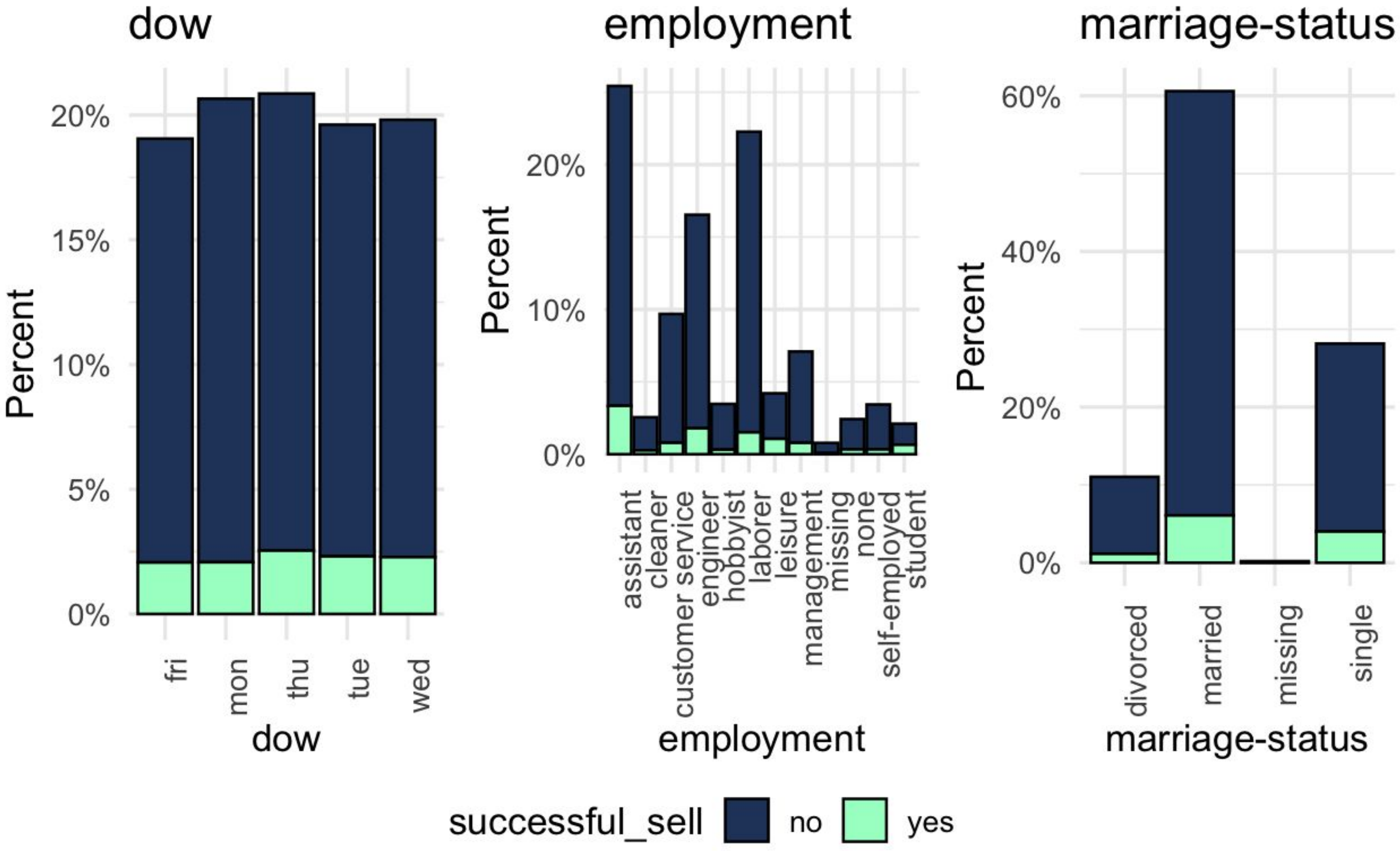
Next Steps and Places for Improvement

- Deploy the model so the customer can use it to generate likely leads
  - Watch features to ensure data generating process remains the same
  - Monitor model performance

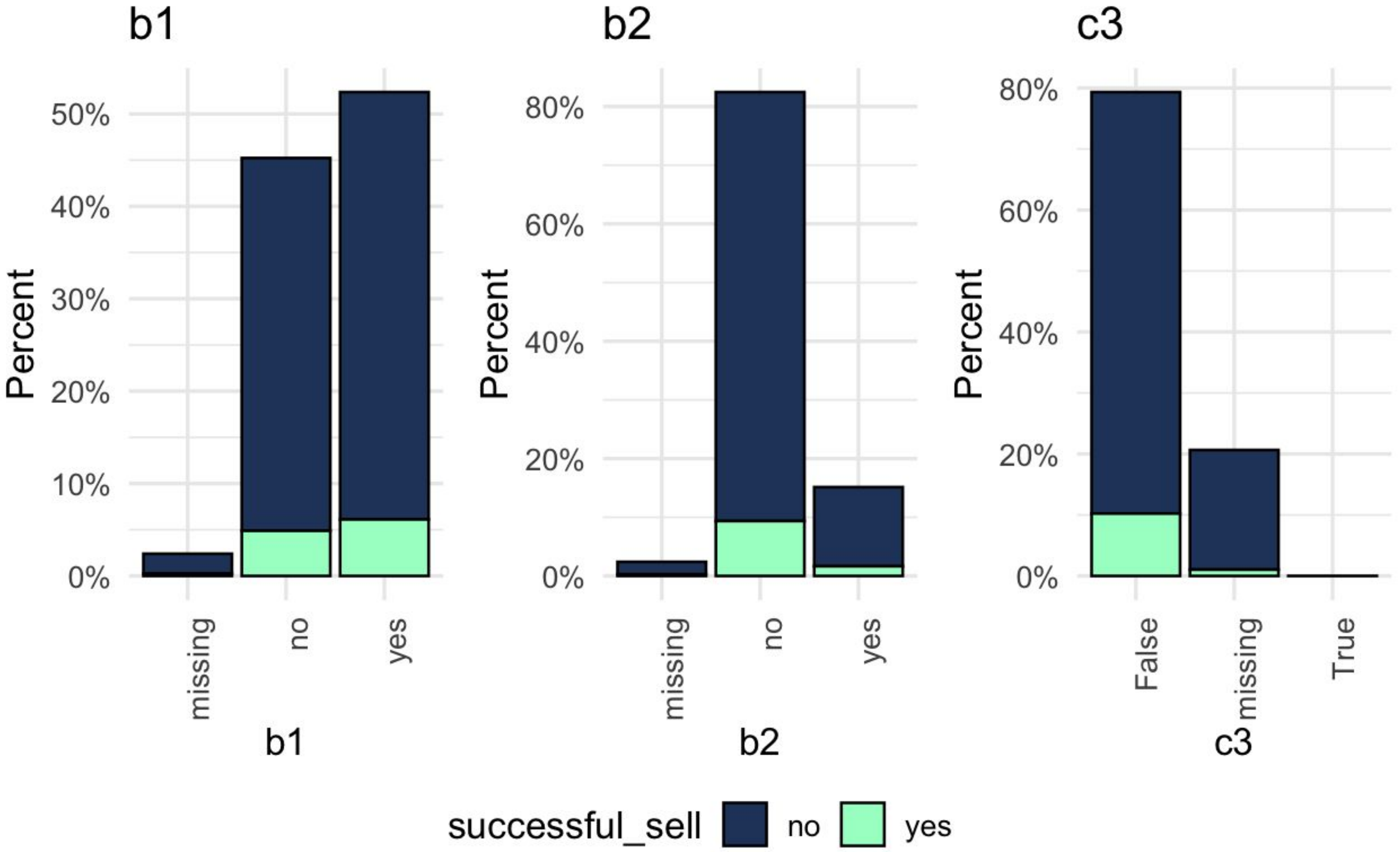
- Look to utilize domain knowledge and other data sources if available in future modeling
  - If we can leverage strong domain knowledge, consider a Bayesian approach which naturally lends itself to decision analysis

- Consider different choices for feature engineering and imputation methods and perform sensitivity analysis on those choices

Questions???



dow
Percent
20%
15%
10%
5%
0%
fri
mon
thu
tue
wed
dow
employment
Percent
20%
10%
0%
assistant
cleaner
customer service
engineer
hobbyist
laborer
leisure
management
missing
none
self-employed
student
employment
marriage-status
Percent
60%
40%
20%
0%
divorced
married
missing
single
marriage-status
successful_sell
no
yes



b1
b2
c3
Percent
50%
40%
30%
20%
10%
0%
80%
60%
missing
no
yes
False
True
successful_sell