Preprocess_Modeling

August 4, 2025

1 Pre-Processing of Bank Dataset

[1]:
```python
# Environment setup
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

from pathlib import Path

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer, StandardScaler,␣
 ↪OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_validate
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
```

[2]:
```python
# install custom library
try:
    # check if jcds is installed
    from jcds import reports as jrep
except ImportError:
    print("Installing custom library: jcds")
    !pip install git+https://github.com/junclemente/jcds.git


from jcds import reports as jrep
```

```
from jcds import eda as jeda
```

[19]:
```
# Retrieve dataset
# Code developed with assistance from Generative AI

# check development environment
try:
    import google.colab
    IN_COLAB = True
except:
    IN_COLAB = False

# connect to drive if IN_COLAB = True
if IN_COLAB:
    from google.colab import drive
    drive.mount('/content/drive')

# set datapath
data_path = "../datasets/bank-additional-full.csv"

# download dataset if IN_COLAB = True
if IN_COLAB:
    print("Running in Colab... downloading dataset.")
    data_url = "https://raw.githubusercontent.com/junclemente/
 ↪ads504-final_project/main/datasets/bank-additional-full.csv"
    if not os.path.exists("../datasets"):
        os.makedirs("../datasets")
    if not os.path.exists(data_path):
        !wget -q {data_url} -O {data_path}
else:
    print("Running locally... using local dataset.")

# Load the dataset
df = pd.read_csv(data_path, sep=";")
df.head()
```

```
Running locally… using local dataset.
```

[19]:
```
   age        job  marital    education  default housing loan    contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  month day_of_week  …  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  …         1    999         0  nonexistent          1.1
```

```
1   may         mon  ...         1    999         0  nonexistent           1.1
2   may         mon  ...         1    999         0  nonexistent           1.1
3   may         mon  ...         1    999         0  nonexistent           1.1
4   may         mon  ...         1    999         0  nonexistent           1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
```

1.1 Numerical Features

1.1.1 Cardinality Report

[4]:
```python
jrep.data_cardinality(df, show_columns=True)
```

```
CARDINALITY REPORT

Total columns analyzed: 21

[BINARY COLUMNS]
There are 2 binary columns.
 * Columns: ['contact', 'y']
There are 0 binary with nan.

[CONSTANT/NEAR CONSTANT COLUMNS]
There are 0 constant columns.
There are 1 near-constant columns with >= 95% of values being the same.
 * Columns: ['pdays']

[LOW CARDINALITY CATEGORICAL COLUMNS]
 * There are 10 low cardinality columns with <= 10 unique values.
Columns:
 * marital: 4 unique values
 * education: 8 unique values
 * default: 3 unique values
 * housing: 3 unique values
 * loan: 3 unique values
 * contact: 2 unique values
 * month: 10 unique values
 * day_of_week: 5 unique values
 * poutcome: 3 unique values
 * y: 2 unique values
```

```
[HIGH CARDINALITY CATEGORICAL COLUMNS]
 * There are 0 high cardinality variables with >=90% unique values.
```

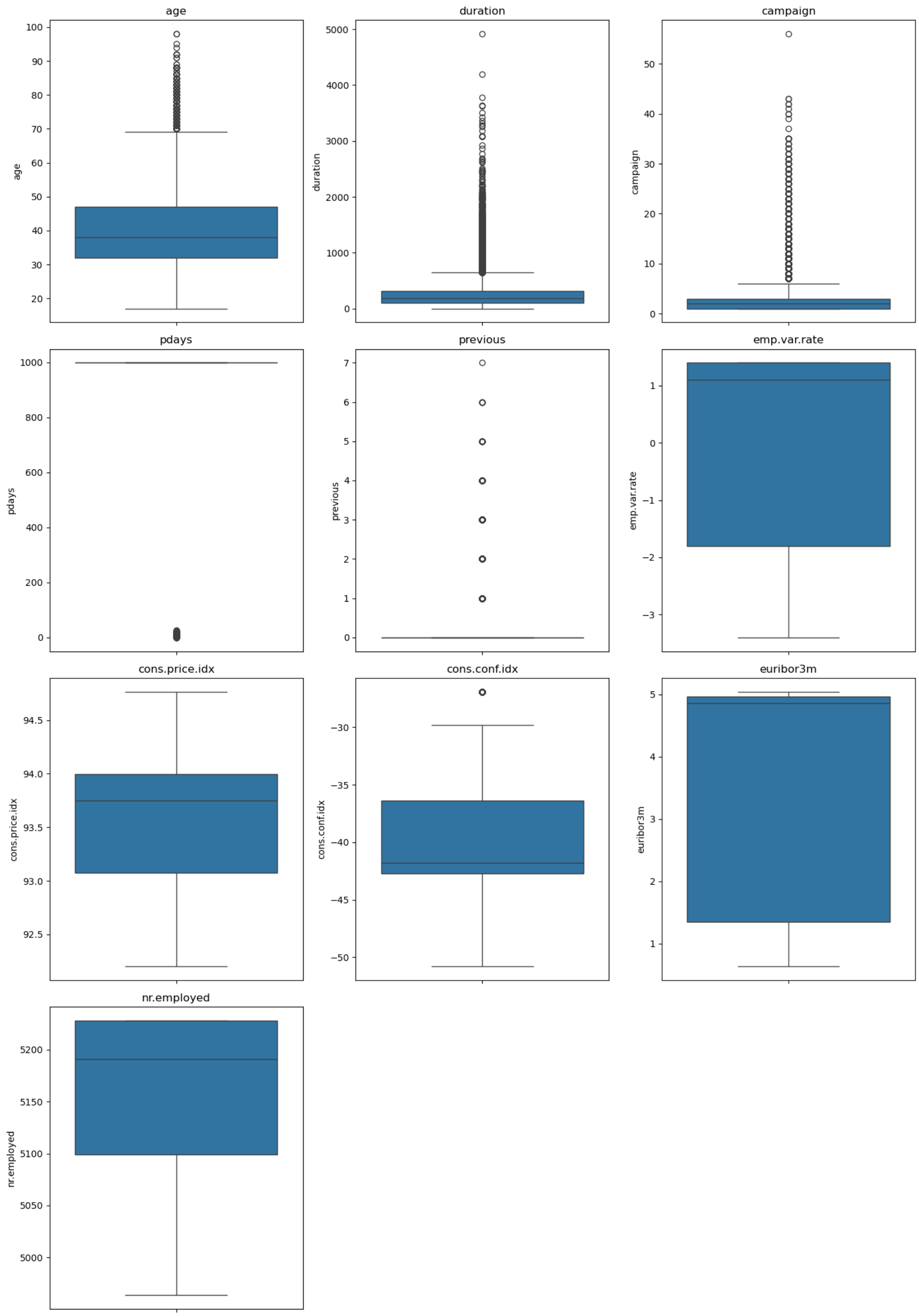
1.1.2 Outliers

[5]:
```python
# get total count of outliers
outliers = jeda.detect_outliers_iqr(df)

# create dataframe
outliers_df = pd.DataFrame(list(outliers.items()), columns=["feature", "count"])
display(outliers_df)
```

	feature	count
0	age	469
1	duration	2963
2	campaign	2406
3	pdays	1515
4	previous	5625
5	emp.var.rate	0
6	cons.price.idx	0
7	cons.conf.idx	447
8	euribor3m	0
9	nr.employed	0

[6]:
```python
jeda.plot_outlier_boxplots(df)
```


age
duration
campaign
pdays
previous
emp.var.rate
cons.price.idx
cons.conf.idx
euribor3m
nr.employed

1.1.3 Observation

Six of the numerical variables have outliers.

1. **age outliers:** 70 - 100. These appear to be legitimate ages. No processing needed, maybe center/scale transform.
2. **duration:** based on data dictionary, this feature is highly correlated with target and should be removed.
3. **campaign:** There are 2406 detected outlies and showing a right skew. Transform to center/scale should be considered.
4. **pdays:** This could be converted to binary.
5. **previous:** Represents the number of times a customer was contacted prior to the campaign. This could be converted to binary: contacted vs not contacted.
6. **cons.conf.idx:** Total 447 outliers detected. Based on distribution from EDA, maybe this can be binned or turned to categorical.

Possible multicolinearity:

- euribor3m and emp.var.rate
- nr.employed and emp.var.rate
- cons.price.idx and emp.var.rate
- pdays and previous

1.2 Categorical Features

1.2.1 Observations

- marital: does not show much variability; may not be a good predictor
- housing: does not show much variability; not a good predictor unless `unknown` could have information
- loan: same as housing, does not show much variability. Also has a `unknown`
- day_of_week: does not show much variability; may not be a good predictor

Based on chi_2 stat, `housing` and `loan` are not statistically significant with p-value > 0.05.

With the Cramer's V statistic, `marital` and `day_of_week` do not show strong association with the target, `0.5`, and `0.3` respectively. These may not be good predictors and could be candidates for removal from the final dataset for modeling.

1.3 Variables to Drop

Due to multicollinearity with `emp.var.rate`, the following numerical features will be dropped:

- euribor3m
- nr.employed
- cons.price.idx

Out of these four numerical features, without someone with domain knowledge, generative AI was asked which feature would be best to keep. ChatGPT recommended keeping `emp.var.rate` due to "strong economic indicator, lower redundancy, and most interpretable."

`pdays` and `previous` also show colinearity. Consder converting `pdays` to binary and dropping `previous` as it is not a strong predictor of the target.

Due to low statistical significance and low strength of association, the following categorical features will be dropped:

- housing
- loan

2 Transformation Pipelines

[7]:
```
jrep.data_info(df, show_columns=True)
```

```
SHAPE:
There are 41188 rows and 21 columns (30.26 MB).

DUPLICATES:
There are 12 duplicated rows.

COLUMNS/VARIABLES:
Column dType Summary:
 * object: 11
 * int: 5
 * float: 5
There are 10 numerical (int/float/bool) variables.
 * Columns: ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
There are 11 categorical (nominal/ordinal) variables.
 * Columns: ['job', 'marital', 'education', 'default', 'housing', 'loan',
'contact', 'month', 'day_of_week', 'poutcome', 'y']

DATETIME COLUMNS:
There are 0 datetime variables and 0 possible datetime variables.

OTHER COLUMN/VARIABLE INFO:
ID Like Columns (threshold = 95.0%): 0
Columns with mixed datatypes: 0
 * Columns: []
```

[8]:
```
# get feature list by type
cat_var = jeda.show_catvar(df)
con_var = jeda.show_convar(df)
target = "y"

# remove target from categorical list
cat_var = [col for col in cat_var if col != target]
```

```python
print(f"Categorical variables:", cat_var)
print(f"Continuous variables:", con_var)
```

```
Categorical variables: ['job', 'marital', 'education', 'default', 'housing',
'loan', 'contact', 'month', 'day_of_week', 'poutcome']
Continuous variables: ['age', 'duration', 'campaign', 'pdays', 'previous',
'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
```

[9]:
```python
cols_to_drop = [
    "housing",
    "loan",
    "previous",
    "pdays",  # after converting to binary
    "euribor3m",
    "nr.employed",
    "cons.price.idx",
]

con_var = [c for c in con_var if c not in cols_to_drop]
cat_var = [c for c in cat_var if c not in cols_to_drop]
```

2.1 Train / Test Split

[10]:
```python
# Encode target variable to 0, 1 prior to train_test_split
df[target] = df[target].replace({"no": 0, "yes": 1}).astype(int)
# show target unique values
jeda.list_unique_values(df, column=target)
```

```
>>> EXECUTING DataFrame["y"].unique().tolist()
Unique values in 'y':
[0, 1]
```

```
/tmp/ipykernel_265047/570709414.py:2: FutureWarning: Downcasting behavior in
`replace` is deprecated and will be removed in a future version. To retain the
old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to
the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df[target] = df[target].replace({"no": 0, "yes": 1}).astype(int)
```

[11]:
```python
RANDOM_STATE = 42
X = df.drop(columns=target)
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪stratify = y, random_state = RANDOM_STATE)
```

2.2 Define Pipeline Tranformers

[12]:
```
# columns to drop
def drop_cols(X):
    X = X.copy()
    return X.drop(columns=cols_to_drop)

drop_columns = FunctionTransformer(drop_cols, validate=False)

# convert pdays to binary
def pdays_to_binary(X):
    X = X.copy()
    X["pdays_binary"] = (X["pdays"] != 999).astype(int)
    return X


convert_to_binary = FunctionTransformer(pdays_to_binary, validate=False)

# scaler and encoder
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), con_var),
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_var),
    ]
)
```

2.3 Modeling Section

[13]:
```
# define function for modeling generator
def make_pipeline(clf):
    return ImbPipeline(steps=[
        ("bin", convert_to_binary),
        ("drop", drop_columns),
        ("encoder", preprocessor),
        ("smote", SMOTE(random_state=RANDOM_STATE)),
        ("classifier", clf)
    ])
```

[14]:
```
# initialize the models (classifiers)
models = {
    "Perceptron": Perceptron(random_state=RANDOM_STATE),
    "Logistic Regression": LogisticRegression(random_state=RANDOM_STATE),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Support Vector Machine": SVC(random_state=RANDOM_STATE, probability=True),
    "Neural Network": MLPClassifier(random_state=RANDOM_STATE, max_iter=1000),
    "XGBoost": XGBClassifier(random_state=RANDOM_STATE,␣
 ↪use_label_encoder=False, eval_metric='logloss'),
```

```
    "AdaBoost": AdaBoostClassifier(random_state=RANDOM_STATE),
    "Random Forest": RandomForestClassifier(random_state=RANDOM_STATE)
}
```

[15]:
```
# set scoring metrics
scoring = ["accuracy", "precision", "recall", "f1", "roc_auc"]
```

[16]:
```
# cross-validate each model
# initialize results dictionary
avg_results = {}
full_results = {}
for name, clf in models.items():
    pipe = make_pipeline(clf)
    cv_results = cross_validate(pipe, X_train, y_train, cv=5, scoring=scoring,␣
 ↪n_jobs=-1)
    full_results[name] = {
        "fit_time": cv_results["fit_time"],
        "score_time": cv_results["score_time"],
        "test_accuracy": cv_results["test_accuracy"],
        "test_precision": cv_results["test_precision"],
        "test_recall": cv_results["test_recall"],
        "test_f1": cv_results["test_f1"],
        "test_roc_auc": cv_results["test_roc_auc"]}
    avg_results[name] = {metric: np.mean(cv_results[f"test_{metric}"]) for␣
 ↪metric in scoring}
    avg_results[name].update({f"std_{metric}": np.
 ↪std(cv_results[f"test_{metric}"]) for metric in scoring})
```

```
/home/junc/miniconda3/envs/ads504_project/lib/python3.10/site-
packages/xgboost/training.py:183: UserWarning: [00:22:16] WARNING:
/home/conda/feedstock_root/build_artifacts/xgboost-
split_1748293041487/work/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/home/junc/miniconda3/envs/ads504_project/lib/python3.10/site-
packages/xgboost/training.py:183: UserWarning: [00:22:16] WARNING:
/home/conda/feedstock_root/build_artifacts/xgboost-
split_1748293041487/work/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/home/junc/miniconda3/envs/ads504_project/lib/python3.10/site-
packages/xgboost/training.py:183: UserWarning: [00:22:17] WARNING:
/home/conda/feedstock_root/build_artifacts/xgboost-
split_1748293041487/work/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
  bst.update(dtrain, iteration=i, fobj=obj)
/home/junc/miniconda3/envs/ads504_project/lib/python3.10/site-
packages/xgboost/training.py:183: UserWarning: [00:22:17] WARNING:
/home/conda/feedstock_root/build_artifacts/xgboost-
split_1748293041487/work/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/home/junc/miniconda3/envs/ads504_project/lib/python3.10/site-
packages/xgboost/training.py:183: UserWarning: [00:22:17] WARNING:
/home/conda/feedstock_root/build_artifacts/xgboost-
split_1748293041487/work/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

[17]:

```python
# create dataframe of average results
avg_results_df = pd.DataFrame(avg_results).T
# round results to 3 decimal places
avg_results_df = avg_results_df.round(3)
# display average results
display(avg_results_df)
```

```
                        accuracy  precision  recall     f1  roc_auc  \
Perceptron                 0.815      0.355   0.751  0.479    0.869
Logistic Regression        0.861      0.440   0.866  0.583    0.932
K-Nearest Neighbors        0.851      0.415   0.791  0.544    0.880
Support Vector Machine     0.876      0.470   0.826  0.599    0.934
Neural Network             0.888      0.502   0.543  0.521    0.905
XGBoost                    0.909      0.600   0.576  0.588    0.941
AdaBoost                   0.877      0.474   0.806  0.596    0.928
Random Forest              0.904      0.577   0.567  0.572    0.935

                        std_accuracy  std_precision  std_recall  std_f1  \
Perceptron                     0.027          0.042       0.077   0.042
Logistic Regression            0.003          0.007       0.011   0.006
K-Nearest Neighbors            0.006          0.012       0.010   0.010
Support Vector Machine         0.006          0.014       0.019   0.013
Neural Network                 0.003          0.012       0.029   0.012
XGBoost                        0.003          0.018       0.018   0.012
AdaBoost                       0.004          0.010       0.028   0.010
Random Forest                  0.004          0.020       0.014   0.013

                        std_roc_auc
Perceptron                    0.028
Logistic Regression           0.004
K-Nearest Neighbors           0.005
Support Vector Machine        0.005
```

```
Neural Network               0.005
XGBoost                      0.003
AdaBoost                     0.004
Random Forest                0.002
```

[18]:

```
# create a for loop that sorts the results by each metric for the first 5␣
↪columns after the model names
for metric in scoring:
    sorted_results = avg_results_df.sort_values(by=metric, ascending=False)
    print(f"Models by {metric}:")
    # display every model and its score
    display(sorted_results[[metric]])
```

Models by accuracy:

	accuracy
XGBoost	0.909
Random Forest	0.904
Neural Network	0.888
AdaBoost	0.877
Support Vector Machine	0.876
Logistic Regression	0.861
K-Nearest Neighbors	0.851
Perceptron	0.815

Models by precision:

	precision
XGBoost	0.600
Random Forest	0.577
Neural Network	0.502
AdaBoost	0.474
Support Vector Machine	0.470
Logistic Regression	0.440
K-Nearest Neighbors	0.415
Perceptron	0.355

Models by recall:

	recall
Logistic Regression	0.866
Support Vector Machine	0.826
AdaBoost	0.806
K-Nearest Neighbors	0.791
Perceptron	0.751
XGBoost	0.576
Random Forest	0.567
Neural Network	0.543

Models by f1:

	f1
Support Vector Machine	0.599
AdaBoost	0.596
XGBoost	0.588
Logistic Regression	0.583
Random Forest	0.572
K-Nearest Neighbors	0.544
Neural Network	0.521
Perceptron	0.479

Models by roc_auc:

	roc_auc
XGBoost	0.941
Random Forest	0.935
Support Vector Machine	0.934
Logistic Regression	0.932
AdaBoost	0.928
Neural Network	0.905
K-Nearest Neighbors	0.880
Perceptron	0.869