Bank Marketing Analysis

by Runtian Li, Rafe Chang, Sid Grover, Anu Banga

Repo Link: https://github.com/UBC-MDS/dsci_522_group_8.git

In [1]:
```
## Import necessary Packages
import altair as alt
import altair_viewer
alt.data_transformers.enable("vegafusion")

import pandas as pd
import numpy as np
import statistics
import os
import sys

import warnings
warnings.filterwarnings("ignore")

sys.path.append("code/.")

# Data
from ucimlrepo import fetch_ucirepo

# Machine Learning
import IPython
import matplotlib.pyplot as plt
import mglearn
from IPython.display import HTML, display
# from plotting_functions import *
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, cross_validate, train_t
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.metrics import make_scorer, f1_score, recall_score, precision_s
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.metrics import PrecisionRecallDisplay

# %matplotlib inline
pd.set_option("display.max_colwidth", 200)

from IPython.display import Image
```

Summary

Here we build a model of balanced SVC to try to predict if a new client will subscribe to a term deposit. We tested five different classification models, including dummy classifier, unbalanced/balanced logistic regression, and unbalanced/balanced SVC, and chose the optimal model of balanced SVC based on how the model scored on the test data; the model has the highest test recall score of 0.82, which indicates that the model makes the least false negative predictions among all five models.

The balanced support vector machines model considers 13 different numerical/categorical features of customers. After hyperparameter optimization, the model's test accuracy increased from 0.82 to 0.875. The results were somewhat expected, given SVC's known efficacy in classification tasks, particularly when there's a clear margin of separation. The high recall score of 0.875 indicates that the model is particularly adept at identifying clients likely to subscribe, which was the primary goal. It's noteworthy that such a high recall was achieved, as it suggests the model is highly sensitive to true positive cases.

Introduction

Background

The data set Bank Marketing was created by Sérgio Moro and Paulo Rita at the University Institute of Lisbon, and Paulo Cortez at the University of Minhom. It is sourced from the UCI Machine Learning Repository. Each row in this data set is an observation related to direct marketing campaigns (phone calls) of a Portuguese banking institution.

Research Question

We are working on a binary classification model. The classification goal is to predict if the client will subscribe a term deposit: "yes" for will subscribe and "no" for won't subscribe.

Data Description

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. It was sourced from the UCI Machine Learning Repository and can be found here. We will be using bank-full.csv with all examples and 17 inputs, ordered by date (older version of this dataset with less inputs).

These are the detail of all inputs:

Feature Name	Type	Description	Classes
age	Numeric		
job	Categorical	Type of job	'admin.','blue-collar','entrepreneur','housemaid','management','retire employed','services','student','technician','unemployec
marital	Categorical	Marital status	'divorced','married','single','unknown'
education	Categorical		'primary', 'secondary', 'tertiary', 'unknown'
default	Categorical	Has credit in default?	'no', 'yes', 'unknown'
housing	Categorical	Has housing loan?	'no', 'yes', 'unknown'
loan	Categorical	Has personal loan?	'no', 'yes', 'unknown'
balance	Numeric	Balance of the individual	
contact	Categorical	Contact communication type	'cellular', 'telephone'
month	Categorical	Last contact month of year	'jan', 'feb', 'mar', ..., 'nov', 'dec'
day	Categorical	Last contact day of the week	'mon', 'tue', 'wed', 'thu', 'fri'
duration	Numeric	Last contact duration, in seconds	
campaign	Numeric	Number of contacts performed during this campaign and for this client	
pdays	Numeric	Number of days that passed by after the client was last contacted from a previous campaign	
previous	Numeric	Number of contacts performed before this	

Feature Name	Type	Description	Classes
		campaign and for this client	
poutcome	Categorical	Outcome of the previous marketing campaign	'failure', 'nonexistent', 'success'
y	Binary	Has the client subscribed to a term deposit?	'yes', 'no'

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Results and Discussion

Exploratory Data Analysis

In [2]:
```
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

# Import the uniques function from the src folder
sys.path.append('..')
from src.uniques import get_uniques

df = pd.read_csv("../data/bank-full.csv", delimiter=";")
df.rename(columns={"y": "target"}, inplace=True)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=123)

get_uniques(df);
```

In [3]:
```
# Import the eda_plotting functions function from the src folder
sys.path.append('..')
from src.eda_plotting import (
                                EDA_plot,
                                spearman_correlation_matrix,
                                text_EDA
                            )
```

In [4]:
```
numeric_cols = train_df.select_dtypes(include=['int64', 'float64']).columns.
categorical_cols = ["job", "marital", "education", "default", "housing", "lo
numerical_cols = numeric_cols
```

In [5]:
```
text_EDA(train_df)
```

```
DataFrame Information:
<class 'pandas.core.frame.DataFrame'>
Index: 36168 entries, 28686 to 15725
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        36168 non-null  int64 
 1   job        36168 non-null  object
 2   marital    36168 non-null  object
 3   education  36168 non-null  object
 4   default    36168 non-null  object
 5   balance    36168 non-null  int64 
 6   housing    36168 non-null  object
 7   loan       36168 non-null  object
 8   contact    36168 non-null  object
 9   day        36168 non-null  int64 
 10  month      36168 non-null  object
 11  duration   36168 non-null  int64 
 12  campaign   36168 non-null  int64 
 13  pdays      36168 non-null  int64 
 14  previous   36168 non-null  int64 
 15  poutcome   36168 non-null  object
 16  target     36168 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.0+ MB
```

```
Descriptive Statistics:
```

	count	mean	std	min	25%	50%	75%	max
age	36168.0	40.944979	10.609908	18.0	33.0	39.0	48.00	95.0
balance	36168.0	1371.354208	2999.155128	-8019.0	73.0	448.5	1448.00	98417.0
day	36168.0	15.801095	8.309679	1.0	8.0	16.0	21.00	31.0
duration	36168.0	258.955403	259.218884	0.0	103.0	180.0	319.25	4918.0
campaign	36168.0	2.759013	3.095290	1.0	1.0	2.0	3.00	58.0
pdays	36168.0	40.199762	100.114274	-1.0	-1.0	-1.0	-1.00	871.0
previous	36168.0	0.580596	2.364362	0.0	0.0	0.0	0.00	275.0

```
First 5 Rows:
```

	age	job	marital	education	default	balance	housing	loan	contact
28686	29	services	single	secondary	no	-205	no	no	cellular
9304	53	blue-collar	married	primary	no	0	yes	no	unknown
41425	55	management	married	primary	no	2587	no	no	cellular
44803	30	technician	single	tertiary	no	0	no	no	cellular
5878	30	unemployed	married	secondary	no	529	yes	yes	unknown

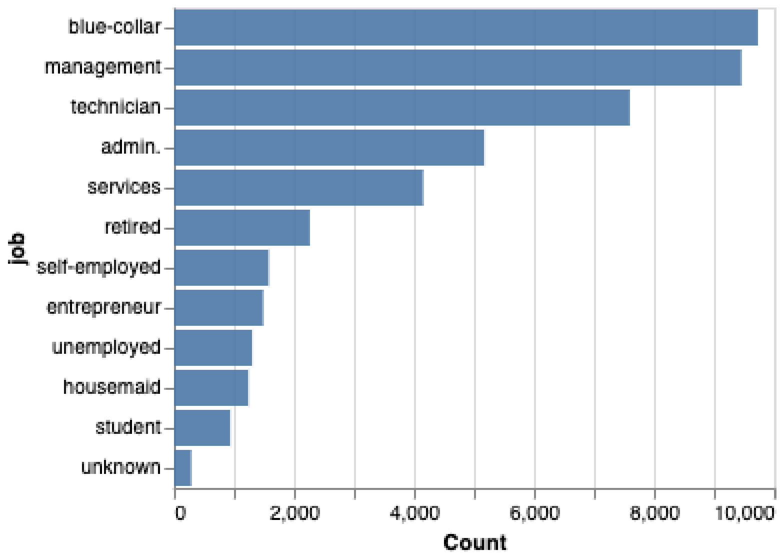
```
Last 5 Rows:
```

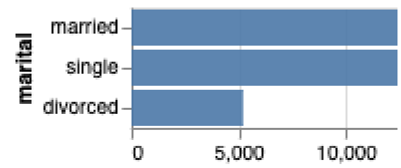
	age	job	marital	education	default	balance	housing	loan	contac
7763	50	unemployed	married	secondary	no	3674	yes	no	unknow
15377	36	management	married	tertiary	no	635	yes	no	cellula
17730	43	blue-collar	married	primary	no	3664	no	no	telephon
28030	55	unemployed	married	primary	no	8585	no	no	telephon
15725	46	management	single	tertiary	no	2154	yes	no	cellula

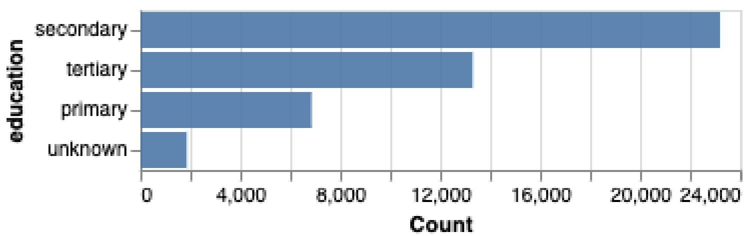
In [6]:
```
display(spearman_correlation_matrix(df, numerical_cols))
```

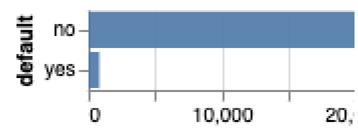
	age	balance	day	duration	campaign	pdays	previou
age	1.000000	0.096380	-0.008948	-0.033257	0.037136	-0.017468	-0.01190
balance	0.096380	1.000000	0.001329	0.042651	-0.030959	0.069676	0.07953
day	-0.008948	0.001329	1.000000	-0.058142	0.139581	-0.092226	-0.08778
duration	-0.033257	0.042651	-0.058142	1.000000	-0.107962	0.028698	0.03117
campaign	0.037136	-0.030959	0.139581	-0.107962	1.000000	-0.112284	-0.10844
pdays	-0.017468	0.069676	-0.092226	0.028698	-0.112284	1.000000	0.98564
previous	-0.011900	0.079536	-0.087780	0.031175	-0.108448	0.985645	1.00000

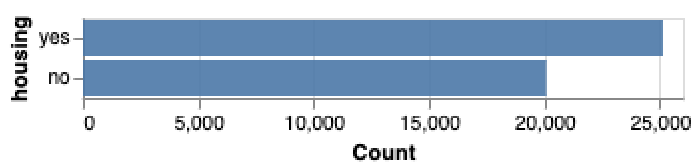
In [7]:
```
display(EDA_plot(df, numeric_cols, categorical_cols))
```

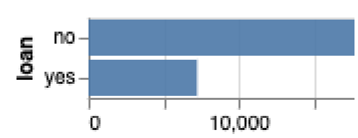

job
blue-collar
management
technician
admin.
services
retired
self-employed
entrepreneur
unemployed
housemaid
student
unknown
0
2,000
4,000
6,000
8,000
10,000
Count

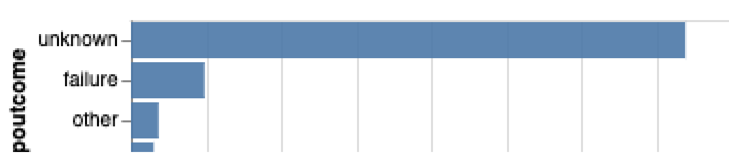

marital
married
single
divorced
0
5,000
10,000

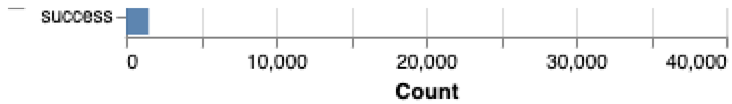

education
secondary
tertiary
primary
unknown
0
4,000
8,000
12,000
16,000
20,000
24,000
Count

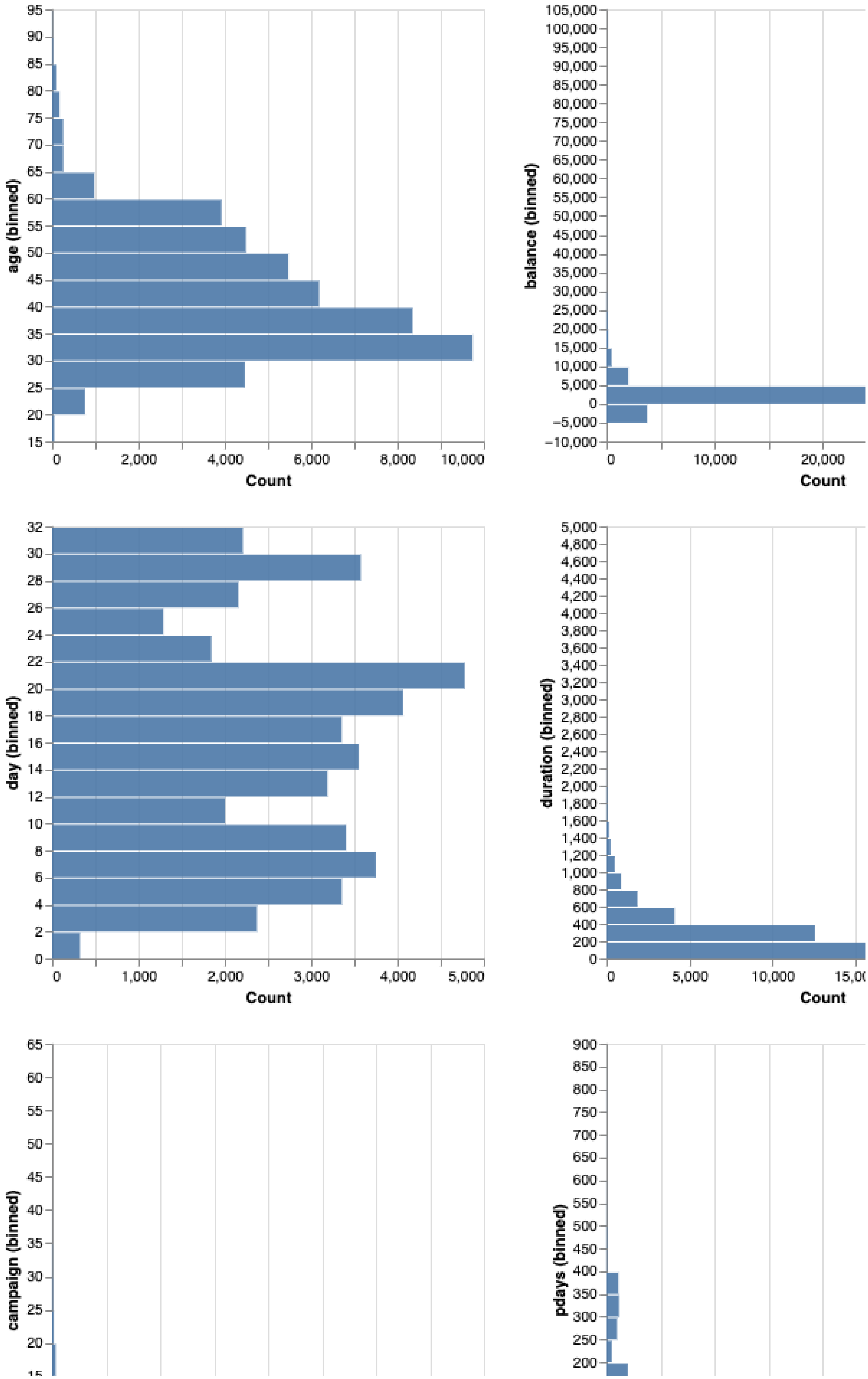

default
no
yes
0
10,000

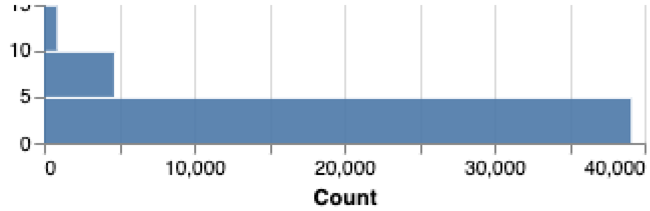

housing
yes
no
0
5,000
10,000
15,000
20,000
25,000
Count

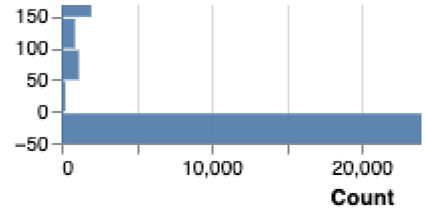

loan
no
yes
0
10,000

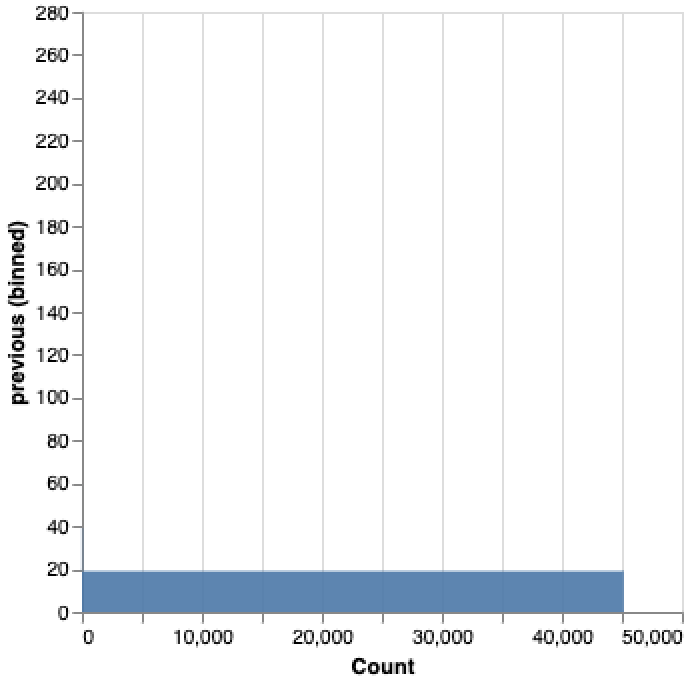

poutcome
unknown
failure
other


success
0
10,000
20,000
30,000
40,000
Count


age (binned)
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
0
2,000
4,000
6,000
8,000
10,000
Count
balance (binned)
105,000
100,000
95,000
90,000
85,000
80,000
75,000
70,000
65,000
60,000
55,000
50,000
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
−5,000
−10,000
0
10,000
20,000
Count
day (binned)
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
0
1,000
2,000
3,000
4,000
5,000
Count
duration (binned)
5,000
4,800
4,600
4,400
4,200
4,000
3,800
3,600
3,400
3,200
3,000
2,800
2,600
2,400
2,200
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
200
0
0
5,000
10,000
Count
campaign (binned)
65
60
55
50
45
40
35
30
25
20
pdays (binned)
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200








```
(None, None)
```

Preprocessing

- Since there is no missing values in our dataset, we don't need to do imputation or drop NAs.
- We are going to drop "contact", "day" and "month" column here since they are not helping us in identifying useful underlying pattern in the model.
- We take "age", "balance", "duration", "campaign", "pdays", "previous" as numerical features and we are doing StandardScaler transformation on them.
- We take "job", "marital", "education", "default", "housing", "loan", "poutcome" as categorical features and we are doing one hot encoding on them. We dropped columns only if the categorical is binary.

In [8]:
```
numeric_looking_columns = train_df.select_dtypes(include=np.number).columns.
print(numeric_looking_columns)
```

```
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
```

In [9]:
```
# Lists of feature names
numerical_features = ["age", "balance", "duration", "campaign", "pdays", "pr
```

```
categorical_features = ["job", "marital", "education", "default", "housing",
drop_features = ["contact", "day", "month"]
```

In [10]:
```
# Import the count_classes function from the src folder
sys.path.append('..')
from src.preprocessor import preprocess_data
```

In [11]:
```
# Call the funciton preprocess_data
X_train_enc, X_train, y_train, X_test, y_test, preprocessor = preprocess_dat
    train_df, test_df, numerical_features, categorical_features, drop_featur
X_train_enc.head()
```

Out[11]:

	age	balance	duration	campaign	pdays	previous	job_admin.
28686	-1.125848	-0.525607	-0.250585	-0.568295	-0.411533	-0.245565	0.0
9304	1.136220	-0.457253	0.100475	-0.245219	-0.411533	-0.245565	0.0
41425	1.324725	0.405335	0.266360	-0.245219	0.537396	0.600341	0.0
44803	-1.031595	-0.457253	-0.173429	-0.245219	-0.411533	-0.245565	0.0
5878	-1.031595	-0.280868	-0.586213	0.077857	-0.411533	-0.245565	0.0

5 rows × 32 columns

Model Selection

In [12]:
```
# 1. Base Model: Dummy Classifier
classification_metrics = ["accuracy", "precision", "recall", "f1"]
dc = DummyClassifier(strategy="most_frequent")
pipe_dc = make_pipeline(preprocessor, dc)
# The mean and std of the cross validated scores for all metrics as a datafr
cross_val_results = {}
scoring = {
    "accuracy": 'accuracy',
    'precision': make_scorer(precision_score, pos_label="yes", zero_divisior
    'recall': make_scorer(recall_score, pos_label="yes"),
    'f1': make_scorer(f1_score, pos_label="yes")
}  # scoring can be a string, a list, or a dictionary

cross_val_results['dummy'] = pd.DataFrame(cross_validate(pipe_dc, X_train, y

# Show the train and validation scores
cross_val_results['dummy']
```

Out[12]:

	mean	std
fit_time	0.061	0.015
score_time	0.092	0.002
test_accuracy	0.883	0.000
train_accuracy	0.883	0.000
test_precision	0.000	0.000
train_precision	0.000	0.000
test_recall	0.000	0.000
train_recall	0.000	0.000
test_f1	0.000	0.000
train_f1	0.000	0.000

In [13]:

```
# 2. Logistic regression

# The logreg model pipeline
logreg = make_pipeline(preprocessor, LogisticRegression(max_iter=1000, rando

# The mean and std of the cross validated scores for all metrics as a datafr
cross_val_results['logreg'] = pd.DataFrame(cross_validate(logreg, X_train, y

# Show the train and validation scores
cross_val_results['logreg']
```

Out[13]:

	mean	std
fit_time	0.806	0.189
score_time	0.166	0.028
test_accuracy	0.900	0.003
train_accuracy	0.900	0.001
test_precision	0.652	0.029
train_precision	0.655	0.006
test_recall	0.313	0.019
train_recall	0.315	0.009
test_f1	0.423	0.023
train_f1	0.425	0.009

In [14]:

```
# 3. Support vector classifier

# The svc model pipeline
svc = make_pipeline(preprocessor, SVC(random_state=123))
```

```python
# The mean and std of the cross validated scores for all metrics as a datafr
cross_val_results['svc'] = pd.DataFrame(cross_validate(svc, X_train, y_trair
# Show the train and validation scores
cross_val_results['svc']
```

Out[14]:

	mean	std
fit_time	6.465	0.129
score_time	2.441	0.058
test_accuracy	0.899	0.002
train_accuracy	0.907	0.001
test_precision	0.655	0.016
train_precision	0.726	0.007
test_recall	0.288	0.008
train_recall	0.326	0.007
test_f1	0.400	0.010
train_f1	0.450	0.007

In [15]:

```python
# 4. Balanced logistic regression
logreg_bal = make_pipeline(preprocessor,
                           LogisticRegression(max_iter=1000,
                                              random_state=123,
                                              class_weight="balanced"))

# The mean and std of the cross validated scores for all metrics as a datafr
cross_val_results['logreg_bal'] = pd.DataFrame(cross_validate(logreg_bal, X_

# Show the train and validation scores
cross_val_results['logreg_bal']
```

Out[15]:

	mean	std
fit_time	0.894	0.197
score_time	0.195	0.050
test_accuracy	0.829	0.002
train_accuracy	0.829	0.001
test_precision	0.386	0.005
train_precision	0.386	0.003
test_recall	0.777	0.012
train_recall	0.778	0.002
test_f1	0.516	0.006
train_f1	0.516	0.003

In [16]:

```
# 5. Balanced support vector classifier
svc_bal = make_pipeline(preprocessor, SVC(random_state=123, class_weight="ba

# The mean and std of the cross validated scores for all metrics as a datafr
cross_val_results['svc_bal'] = pd.DataFrame(cross_validate(svc_bal, X_train,

# Show the train and validation scores
cross_val_results['svc_bal']
```

Out[16]:

	mean	std
fit_time	11.606	0.144
score_time	4.384	0.028
test_accuracy	0.814	0.006
train_accuracy	0.825	0.001
test_precision	0.368	0.010
train_precision	0.388	0.001
test_recall	0.821	0.011
train_recall	0.864	0.004
test_f1	0.508	0.011
train_f1	0.535	0.001

In [17]:

```
# Compare the average scores of all the models
pd.concat(
    cross_val_results,
    axis='columns'
).xs(
    'mean',
```

```
    axis='columns',

    level=1
).style.format(
    precision=2
).background_gradient(
    axis=None
)
```

Out[17]:

	dummy	logreg	svc	logreg_bal	svc_bal
fit_time	0.06	0.81	6.46	0.89	11.61
score_time	0.09	0.17	2.44	0.20	4.38
test_accuracy	0.88	0.90	0.90	0.83	0.81
train_accuracy	0.88	0.90	0.91	0.83	0.82
test_precision	0.00	0.65	0.66	0.39	0.37
train_precision	0.00	0.66	0.73	0.39	0.39
test_recall	0.00	0.31	0.29	0.78	0.82
train_recall	0.00	0.32	0.33	0.78	0.86
test_f1	0.00	0.42	0.40	0.52	0.51
train_f1	0.00	0.42	0.45	0.52	0.54

`Dummy Classifier` has low accuracy and zero precision, recall, and F1 scores, indicating it never predicts the positive class (in this case the client subscribed a term deposit). This is expected as it always predicts the most frequent class.

`logreg` shows improved accuracy over the dummy model. However, its recall is low, suggesting it misses a significant number of true positive cases. `svc` performed almost the same as logistic regression model among all metrics.

`logreg_bal` and `svc_bal` have lower accuracy compared to their unbalanced counterparts but significantly higher recall. This indicates they are better at identifying positive cases but at the cost of making more false positive errors.

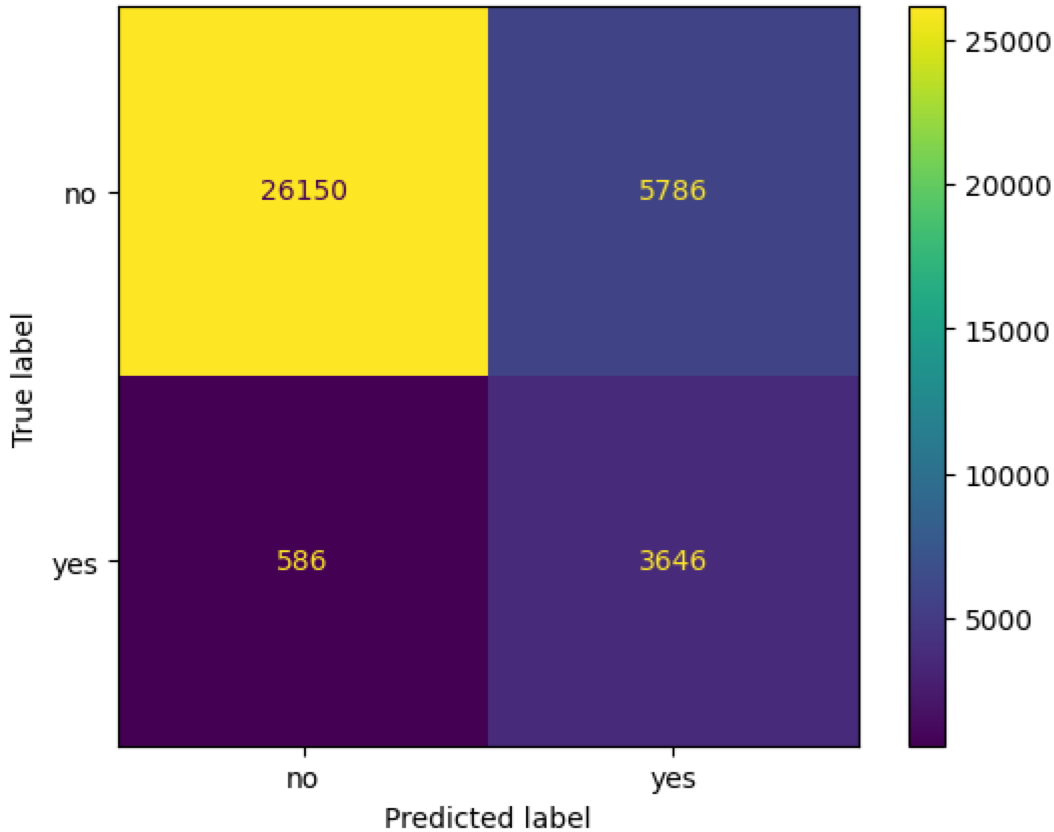
Given the context of our bank marketing data set, we aim to detect the clients who will subscribe a term deposit given the features. Missing a potential "yes" could be more costly than false positives, as it represents a lost opportunity for the sales team to transform this potential customer. Therefore, we chose `svc_bal` as the model has the highest `test_recall` score.

In [18]:

```
svc_bal.fit(X_train, y_train)
confmat_svc_bal = ConfusionMatrixDisplay.from_estimator(
    svc_bal,
    X_train,
```

```
    y_train,
    values_format="d",)
confmat_svc_bal
```

Out[18]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1693c70
40>



In [19]:
```
# Import the scoring_metrics functions function from the src folder
sys.path.append('..')
from src.scoringmetrics import scoring_metrics
result=scoring_metrics(svc_bal, X_train, y_train, X_test, y_test, pos_label=
result
```

Out[19]:

	train_accuracy	test_accuracy	train_precision	test_precision	train_recall	test_re(
0	0.823822	0.815659	0.386556	0.369435	0.861531	0.816(

Hyperparameter Optimization

Optimizing hyperparameters in SVC with a smaller sample size of 10,000 instances is a strategy aimed at enhancing computational efficiency. This approach expedites the exploration of hyperparameter possibilities, aiding in the discovery of potential configurations. While the outcomes validate the concept, it's crucial to recognize and manage the constraints stemming from the smaller dataset size when interpreting the results.

In [20]:
```
# Creating a sample of 10000 observations
sample_data = df.sample(n=10000, random_state=123)
train_df_sampled, test_df_sampled = train_test_split(sample_data, test_size=

X_train_sampled = train_df_sampled.drop(columns=["target"])
X_test_sampled = test_df_sampled.drop(columns=["target"])
y_train_sampled = train_df_sampled["target"]
y_test_sampled = test_df_sampled["target"]

# Transformation on the sample training data
sample_preprocessor = make_column_transformer(
    (StandardScaler(), numerical_features),
    (OneHotEncoder(drop="if_binary"), categorical_features),
    ("drop", drop_features),
)

# X_train_sampled_enc = pd.DataFrame(sample_preprocessor.fit_transform(X_tra

svc_bal_sample = make_pipeline(sample_preprocessor, SVC(random_state=123, cl

param_dist = {
    'svc__C': uniform(0.1, 10),
    'svc__gamma': uniform(0.001, 0.1),
    'svc__kernel': ['rbf', 'sigmoid', 'linear']
}

# Perform RandomizedSearchCV for hyperparameter optimization
random_search = RandomizedSearchCV(svc_bal_sample, param_distributions=param
random_search.fit(X_train_sampled, y_train_sampled)

# Best hyperparameters
best_params_random = random_search.best_params_
print("Best Hyperparameters (Randomized Search):", best_params_random)
```

```
Best Hyperparameters (Randomized Search): {'svc__C': 4.331064601244609, 'svc
__gamma': 0.09907641983846155, 'svc__kernel': 'rbf'}
```

In [21]:
```
pd.DataFrame(random_search.cv_results_)[
    [
        "mean_test_score",
        "param_svc__gamma",
        "param_svc__C",
        "mean_fit_time",
        "rank_test_score",
    ]
].set_index("rank_test_score").sort_index().T
```

Out[21]:

rank_test_score	1	2	3	3	3	3
mean_test_score	0.831875	0.82775	0.8275	0.8275	0.8275	0.8275
param_svc__gamma	0.099076	0.008709	0.044086	0.073905	0.069326	0.018537
param_svc__C	4.331065	1.640822	4.437012	3.53178	5.073088	5.40062
mean_fit_time	0.677113	0.7783	1.51885	1.317613	1.537338	1.499696

4 rows × 25 columns

Test results after hyperparameter optimization

In [22]:

```
# Evaluate the best model on the test set
best_model_random = random_search.best_estimator_
accuracy_random = best_model_random.score(X_test, y_test)
print("Accuracy on Test Set:", accuracy_random)
```

```
Accuracy on Test Set: 0.8613292049098751
```

In [23]:

```
predictions = best_model_random.predict(X_test)

recall = recall_score(y_test, predictions, pos_label='yes')
print("Recall on Test Set:", recall)
```

```
Recall on Test Set: 0.8751182592242195
```

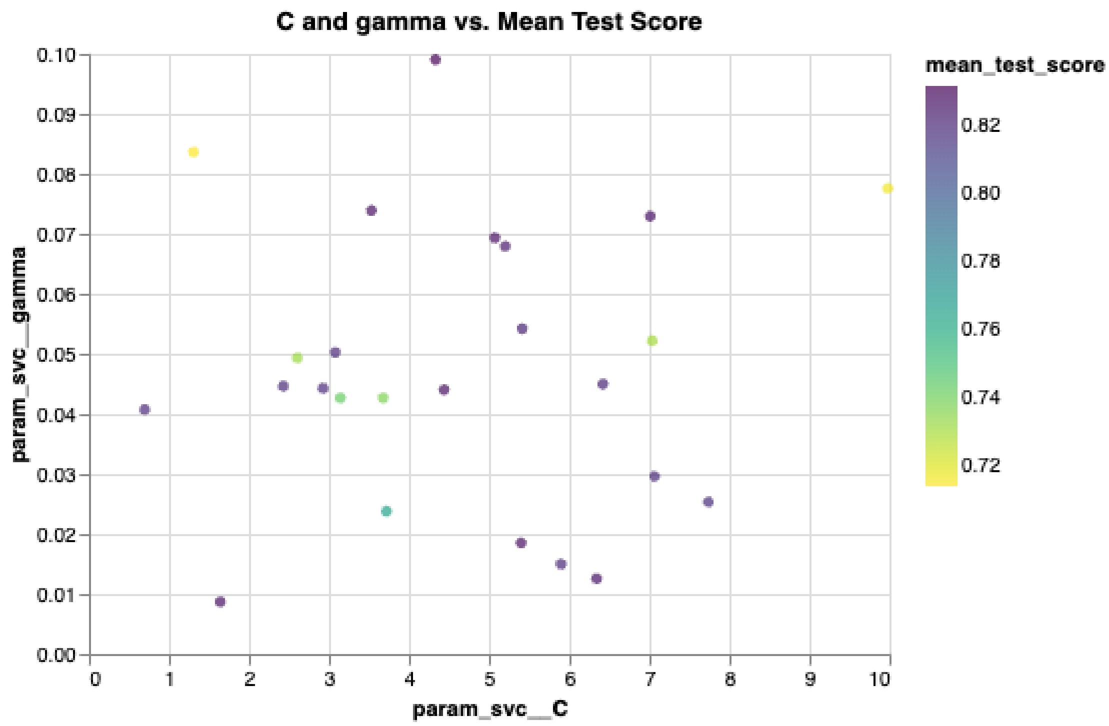
In [24]:

```
results = pd.DataFrame(random_search.cv_results_)

scatter = alt.Chart(results).mark_circle().encode(
    x='param_svc__C:Q',
    y='param_svc__gamma:Q',
    color=alt.Color('mean_test_score:Q',
                    scale=alt.Scale(scheme='viridis', reverse=True)
                   )
).properties(
    width=400,
    height=300,
    title='C and gamma vs. Mean Test Score'
)

scatter
```

Out[24]:



Discussions

Key Findings

In this bank marketing analysis project, we aimed to develop a binary classification model to predict client subscription to term deposits. We tested Logistic Regression and Support Vector Classifier (SVC) models, focusing on recall as a key performance metric. The SVC model outperformed Logistic Regression in recall, and after hyperparameter optimization, it achieved a recall score of 0.875 on the test dataset, which is quite promising!

Reflection on Expectations

The results were somewhat expected, given SVC's known efficacy in classification tasks, particularly when there's a clear margin of separation. The high recall score of 0.875 indicates that the model is particularly adept at identifying clients likely to subscribe, which was the primary goal. It's noteworthy that such a high recall was achieved, as it suggests the model is highly sensitive to true positive cases.

Impact of Finding

The high recall score of this model has significant implications for targeted marketing strategies. It suggests that the bank can confidently use the model's predictions to focus its marketing efforts on clients predicted to subscribe, potentially increasing the efficiency and effectiveness of its campaigns. This targeted approach could lead to higher conversion rates with lower marketing expenses. However, it's important to balance such a high recall with precision to ensure that the bank doesn't unnecessarily target unlikely prospects.

Future Improvements

The success of this model leads to several potential areas for further exploration:

- Balancing Precision and Recall: Investigating methods to enhance precision without substantially reducing recall.
- Feature Analysis: Identifying which features most significantly influence subscription predictions. Model Interpretability: Improving the model's interpretability to better understand the basis for its predictions.
- Temporal Adaptability: Assessing the model's adaptability to evolving trends and customer behaviors over time.
- Testing Alternative Models: Exploring whether ensemble methods or more advanced machine learning algorithms could yield better or comparable results.
- Customer Segmentation: Evaluating the model's performance across different customer segments to tailor more specific marketing strategies.

References

Moro,S., Rita,P., and Cortez,P., 2012. Bank Marketing. UCI Machine Learning Repository. https://doi.org/10.24432/C5K306"

Timbers,T. , Ostblom,J., and Lee,M., 2023. Breast Cancer Predictor Report. GitHub repository, https://github.com/ttimbers/breast_cancer_predictor_py/blob/0.0.1/src/breast_cancer_predic

Moro, S., Cortez, P., & Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. Decis. Support Syst., 62, 22-31.

Alsolami, F.J., Saleem, F., & Al-Ghamdi, A.S. (2020). Predicting the Accuracy for Telemarketing Process in Banks Using Data Mining.

Vajiramedhin, C., & Suebsing, A. (2014). Feature Selection with Data Balancing for Prediction of Bank Telemarketing. Applied mathematical sciences, 8, 5667-5672.

Moura, A.F., Pinho, C.M., Napolitano, D.M., Martins, F.S., & Fornari Junior, J.C. (2020). Optimization of operational costs of Call centers employing classification techniques. Research, Society and Development, 9.