Predictive Analytics for Direct Marketing Campaign: A Banking Case Study

by Gretel Tan, Yan Zeng, Charles Xu & Riya E. Shaju 2023/11/19

Imports

In [1]:
```
from ucimlrepo import fetch_ucirepo
import altair as alt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import set_config
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEnco
from sklearn.compose import make_column_transformer, make_column_selector
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.impute import SimpleImputer
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

Summary

In this project, we aimed to use customer information from a phone-call based direct marketing campaign of a Portugese banking institution to predict whether customers would subscribe to the product offered, a term deposit. We applied several classification based models (k-NN, SVM, logistic regression and random forest) to our dataset to find the model which best fit our data, eventually settling on the random forest model, which performed the best among all the models tested, with an F-beta score with beta = 5 of 0.817, and an accuracy of 0.671 on the test data.

While this was the best performing model out of the models tested, its accuracy still left much to be desired. This indicates that perhaps more data is needed to accurately predict whether customers would subscribe to the term deposit. Future studies may also consider using more features, a different set of features which might be more relevant to whether customers will subscribe, or utilising feature engineering to obtain features which might be more useful in helping to predict whether customers would subscribe to the service.

Introduction

Direct marketing generally refers to the relational marketing process involving getting information on individual consumers, getting feedback on their responses to various measures like sales campaigns, and influencing their behaviours (Bauer & Miglautsch, 1992). Many companies utilise direct marketing strategies to target individual groups of customers, reaching out specifically to groups of customers who will allow companies to meet their sales or business objectives (Moro et al., 2014), such as targeting advertising for a particular product to a specific group of customers who will be most likely to purchase that product. With the advent of rapidly advancing computer and database technologies, as well as the growing field of data science, companies and direct marketers now have unprecedented access to individual-level consumer information, which can be used to develop detailed customer profiles. These profiles are valuable to companies, providing them with great insight to guide the formulation of direct marketing campaigns, among other business strategies (Nowak & Phelps, 1995). As such, companies are keen to utilise technology to revolutionise marketing, using the information and metrics available to them to maximise the value they can get from each consumer over their lifetimes (Moro et al., 2014).

Our project aims to predict whether individual customers will subscribe to a service provided by a company, based on demographic information collected about each customer. Should the model be good enough to predict whether customers are likely to subscribe to the service accurately, the company, a Portugese banking institution, would be able to target ads and marketing phone calls only at the new customers who are most likely to subscribe to this service, or similar services. This would result in huge savings in terms of company resources, freeing up campaign funds and human resources, which might have otherwise been wasted on calling reluctant customers, to be redirected to other services which might benefit the company more. It might also reduce annoyance in customers, as, ideally customers will only receive calls if they are likely to be interested in a product, and would not have to entertain calls or ads about products which they do not care about. This presents a win-win situation for both consumers and the company.

Methods

Data

In this project, a dataset about direct marketing campaigns of a Portugese banking institution, from Sérgio Moro, P. Rita, and P. Cortez was used (Moro, S., Rita, P., and Cortez, P. 2012). The data was downloaded from UC Irvine's Machine Learning Repository, and the link can be found here: https://archive.ics.uci.edu/dataset/222/bank+marketing. The dataset has 16 features and 45211 instances, with each row representing information about a single client of the Portugese bank. The aim of the authors in creating the data set was to predict whether the client will subscribe a term deposit, which is captured by the 'subscribed' column. We have also used this column as our target in our analysis.

Analysis

As our project is interested answering a classification problem, we decided to test different classification models to predict whether customers would subscribe to the term deposit. The models we chose to use are: the k-nearest neighbours (kNN), support vector machine (SVM), logistic regression, and random forest. We chose these models as they offer different benefits, and we were interested in finding out which model would work best for our data. We chose to include logistic regression as it offers both interpretability and potential to perform well in classification problems, while we chose the other models despite their lower interpretability as, in our case, it is not so critical that we understand why or how the model comes to its predictions as long as the model performs well. All variables from the original dataset except poutcome and contact were used to fit our models. 60% of the data was partitioned into the training set, and 40% of the data was partitioned into the test set, used for evaluating how well our best model would perform on unseen data. We used 5-fold cross-validation with the F-beta score (beta = 5) as the classification metric. Beta was chosen as 5 for the F-beta score as we would like to focus on making accurate predictions for the customers who might be interested in subscribing to the term deposit, corresponding to a higher recall. This is as because we would rather have false positives and annoy some customers who might not be interested in subscribing to our service, than miss out on customers who might want to subscribe to the service (false negatives), which would cause the bank to lose a potential opportunity. Furthermore, customers who fit this profile are more likely to subscribe to similar services, and if they are accurately identified, the bank will be able to target them more specifically in future campaigns. Numeric variables were standardised immediately before model testing and fitting, while categorical variables were encoded via one-hot encoding. The Python programming language (Van Rossum and Drake 2009) was used to perform the analysis, with the following Python packages being used as well: numpy(Harris et al. 2020), Pandas (McKinney 2010), altair (VanderPlas, 2018), scikit-learn (Pedregosa et al. 2011), matplotlib (Hunter, 2017).

Results

We started our analysis by reading in the data from the repository. After doing exploratory data analysis of our data, we decided to drop the 'poutcome' and 'contact' features from our data, as there were many NaN values in the two feature columns for them, limiting the usefulness of these features in our model training and predictions. Plotting histograms of the features, coloured by class (whether the customer subscribed or not) revealed that the features were sufficiently differently distributed for us to be confident that we should include all other features in training our models. We also identified that there was great class imbalance in our target. As such, we decided not to use accuracy as the metric used to evaluate our model, as it would not give us a good idea of whether the model is performing well or not, preferring to use the F-beta score (beta = 5) instead.

Reading in Data

In [2]:
```
# fetch dataset
bank_marketing = fetch_ucirepo(id=222)

# bank marketing data
X = bank_marketing.data.features
y = bank_marketing.data.targets

# write raw data "data/raw" directory
X.to_csv("data/raw/bank_marketing_train.csv")
y.to_csv("data/raw/bank_marketing_test.csv")

# concat features and targets
bank_marketing_data = pd.concat([X, y], axis=1)
```

In [3]:
```
# rename target 'y' as 'subscribed'
bank_marketing_data.rename(columns={'y': 'subscribed'}, inplace=True)
```

```
# create a preliminary split to explore data
eda_train_df, eda_test_df = train_test_split(
    bank_marketing_data, train_size=0.60, stratify=bank_marketing_data["subs
)
eda_train_df.head()
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact
4765	31	blue-collar	married	secondary	no	3311	yes	no	NaN
3560	57	admin.	divorced	primary	no	1	no	no	NaN
42934	34	technician	married	secondary	no	3000	yes	yes	cellular
14471	39	blue-collar	married	secondary	no	142	no	yes	cellular
16291	25	blue-collar	single	secondary	no	-247	yes	no	cellular

Exploratory Data Analysis (EDA)

In [4]:

```
eda_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 27126 entries, 4765 to 1090
Data columns (total 17 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   age           27126 non-null  int64 
 1   job           26955 non-null  object
 2   marital       27126 non-null  object
 3   education     26010 non-null  object
 4   default       27126 non-null  object
 5   balance       27126 non-null  int64 
 6   housing       27126 non-null  object
 7   loan          27126 non-null  object
 8   contact       19280 non-null  object
 9   day_of_week   27126 non-null  int64 
 10  month         27126 non-null  object
 11  duration      27126 non-null  int64 
 12  campaign      27126 non-null  int64 
 13  pdays         27126 non-null  int64 
 14  previous      27126 non-null  int64 
 15  poutcome      4998 non-null   object
 16  subscribed    27126 non-null  object
dtypes: int64(7), object(10)
memory usage: 3.7+ MB
```

In [5]:

```
bank_marketing_summary = eda_train_df.describe(include = 'all')
bank_marketing_summary
```

Out[5]:

	age	job	marital	education	default	balance	housing	l
count	27126.000000	26955	27126	26010	27126	27126.000000	27126	27
unique	NaN	11	3	3	2	NaN	2	
top	NaN	blue-collar	married	secondary	no	NaN	yes	
freq	NaN	5847	16368	13948	26628	NaN	15137	22
mean	40.843692	NaN	NaN	NaN	NaN	1364.459670	NaN	
std	10.561813	NaN	NaN	NaN	NaN	3110.445058	NaN	
min	18.000000	NaN	NaN	NaN	NaN	-6847.000000	NaN	
25%	33.000000	NaN	NaN	NaN	NaN	69.000000	NaN	
50%	39.000000	NaN	NaN	NaN	NaN	451.000000	NaN	
75%	48.000000	NaN	NaN	NaN	NaN	1434.000000	NaN	
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	

In [6]:

```
# check for NAs
eda_train_df.isna().sum()
```

Out[6]:

```
age                0
job              171
marital            0
education       1116
default            0
balance            0
housing            0
loan               0
contact         7846
day_of_week        0
month              0
duration           0
campaign           0
pdays              0
previous           0
poutcome       22128
subscribed         0
dtype: int64
```

In [7]:

```
len(eda_train_df) # 60% of the dataset
```

Out[7]:

```
27126
```

In [8]:

```
len(bank_marketing_data) # all the observations in the data
```

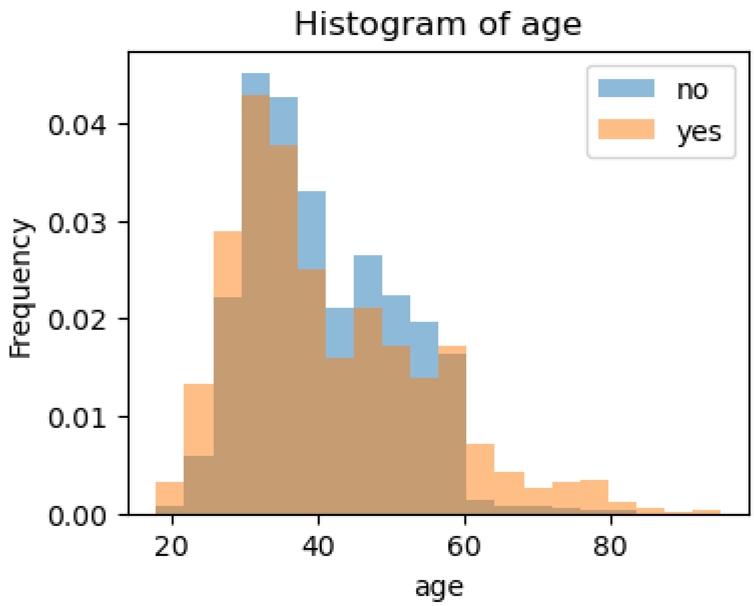
Out[8]:

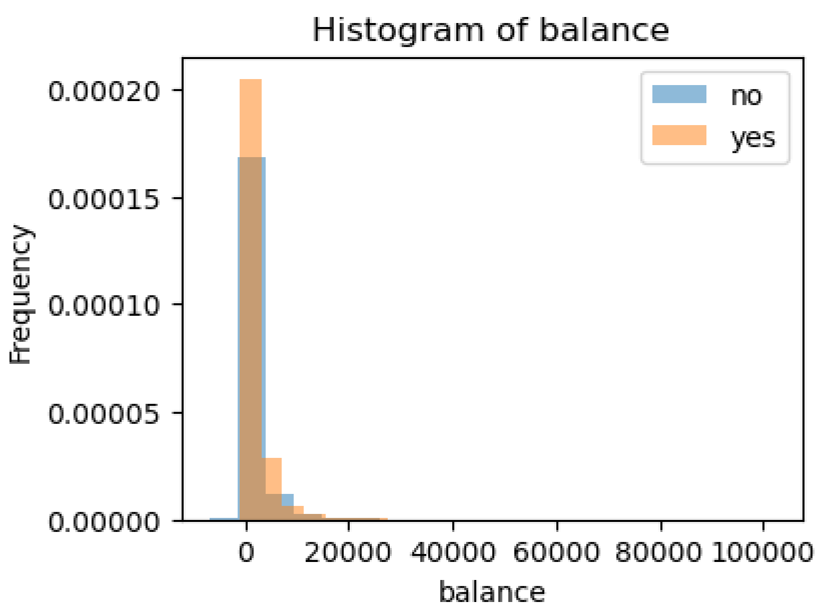
```
45211
```

In [9]:

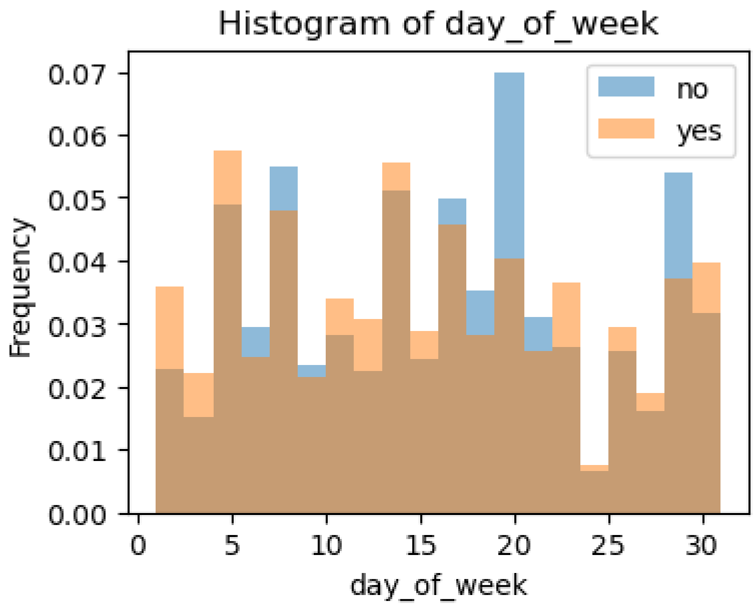
```
# distribution of numerical columns
numeric_cols = eda_train_df.select_dtypes(include=['number']).columns.to_lis
```

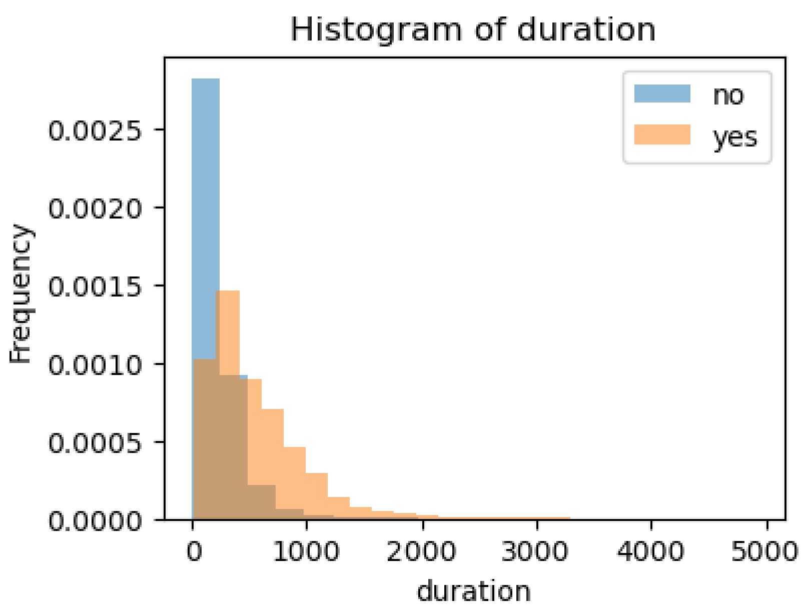
```
for i in numeric_cols:
    feature = i
    plt.figure(figsize=(4, 3))
    plot = eda_train_df.groupby("subscribed")[feature].plot.hist(bins=20, al
    plt.xlabel(feature)
    plt.show()
```

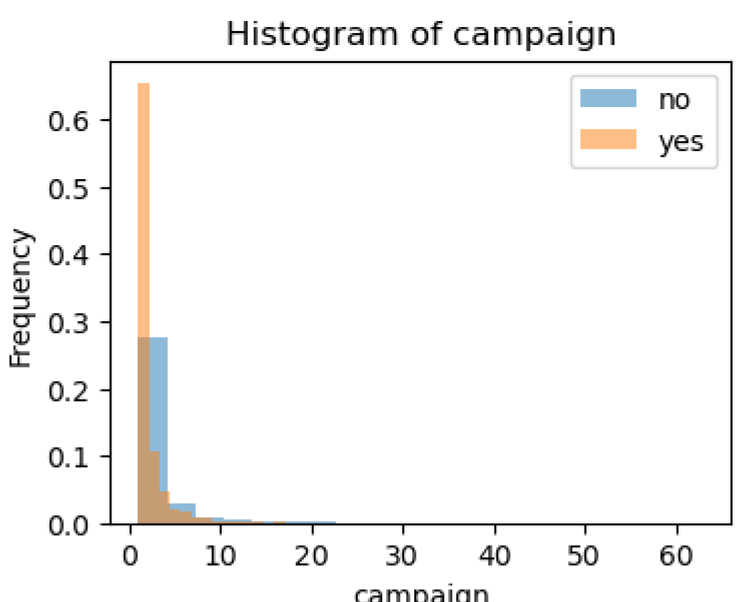




Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js


Histogram of day_of_week
no
yes
Frequency
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
5
10
15
20
25
30
day_of_week


Histogram of duration
no
yes
Frequency
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
1000
2000
3000
4000
5000
duration


Histogram of campaign
no
yes
Frequency
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
10
20
30
40
50
60
campaign

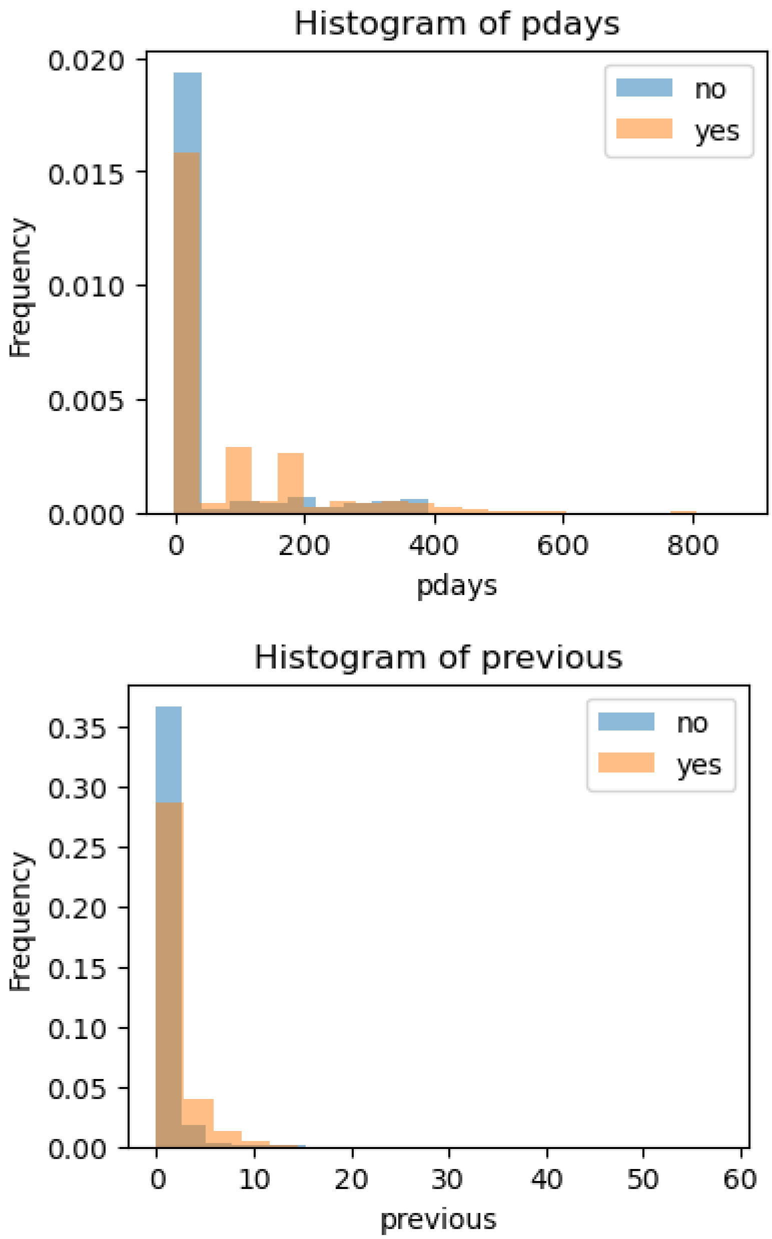


Figure 1. Comparison of the empirical distributions of training data numerical columns.

In [10]:
```
# target class imbalance
counts = eda_train_df["subscribed"].value_counts()
fig, ax = plt.subplots(figsize=(6, 4))
counts.plot(kind='bar', ax=ax)
ax.set_title('Distribution of Target Values')
ax.set_xlabel('Target')
ax.set_ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

Figure 2. Comparison of the empirical distributions of target values.

In [11]:
```
# distribution of NA values in all columns
alt.data_transformers.enable('vegafusion')

alt.Chart(
    eda_train_df.isna().reset_index().melt(
        id_vars='index'
    )
).mark_rect().encode(
    alt.X('index:O').axis(None),
    alt.Y('variable').title(None),
    alt.Color('value').title('NaN'),
    alt.Stroke('value')
).properties(
    width=eda_train_df.shape[0]
)
```

Out[11]:

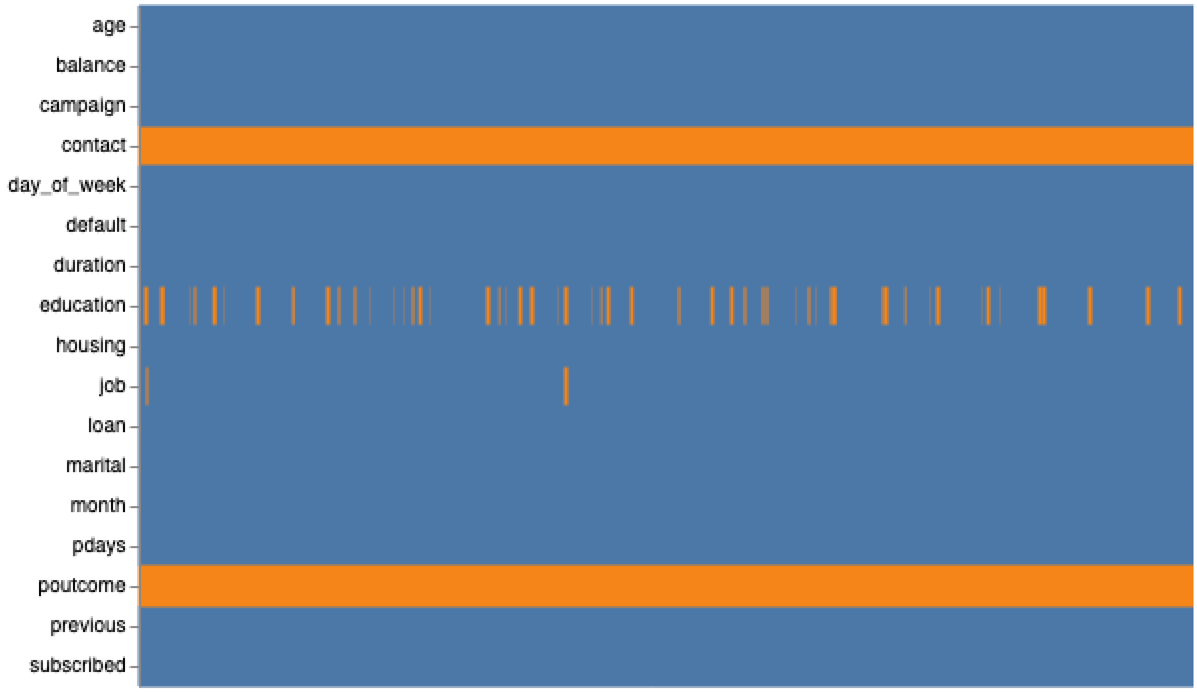


Figure 3. Distribution of NA values in all columns.

In [12]:

```
# unique values in categorical columns
cat_cols = ["job", "marital", "education"]
for column in cat_cols:
    unique_values = list(eda_train_df[column].unique())
    print(f"{column}: {unique_values}")
```

```
job: ['blue-collar', 'admin.', 'technician', 'services', 'student', 'managem
ent', 'entrepreneur', 'retired', 'unemployed', nan, 'self-employed', 'housem
aid']
marital: ['married', 'divorced', 'single']
education: ['secondary', 'primary', 'tertiary', nan]
```

Preprocessing

In [13]:

```
# drop poutcome, contact - too many na values
bank_marketing_data = bank_marketing_data.drop(["poutcome", "contact"], axis
bank_marketing_data = bank_marketing_data.dropna()
bank_marketing_data
```

Out[13]:

	age	job	marital	education	default	balance	housing	loan	day_o
0	58	management	married	tertiary	no	2143	yes	no	
1	44	technician	single	secondary	no	29	yes	no	
2	33	entrepreneur	married	secondary	no	2	yes	yes	
5	35	management	married	tertiary	no	231	yes	no	
6	28	management	single	tertiary	no	447	yes	yes	
...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	no	825	no	no	
45207	71	retired	divorced	primary	no	1729	no	no	
45208	72	retired	married	secondary	no	5715	no	no	
45209	57	blue-collar	married	secondary	no	668	no	no	
45210	37	entrepreneur	married	secondary	no	2971	no	no	

43193 rows × 15 columns

In [14]:
```
# pre-process data (e.g., scale and split into train & test)
np.random.seed(522)
set_config(transform_output="pandas")

# create the split
bank_marketing_train, bank_marketing_test = train_test_split(
    bank_marketing_data, train_size=0.60, stratify=bank_marketing_data["subs
)

bank_marketing_train.to_csv("data/processed/bank_marketing_train.csv")
bank_marketing_test.to_csv("data/processed/bank_marketing_test.csv")
```

In [15]:
```
numeric_features = ['age', 'balance', 'duration', 'campaign', 'pdays', 'prev
categorical_features = ['job', 'marital']
ordinal_features = ['education']
binary_features = ['default', 'housing', 'loan']
drop_features = ['day_of_week', 'month']
target = "subscribed"
```

In [16]:
```
X_train = bank_marketing_train.drop(columns=target)
y_train = bank_marketing_train[target]
X_test = bank_marketing_test.drop(columns=target)
y_test = bank_marketing_test[target]
```

In [17]:
```
numeric_transformer = StandardScaler()

categorical_transformer = OneHotEncoder(handle_unknown="ignore", sparse_outp

education_order = ['primary', 'secondary', 'tertiary']
ordinal_transformer = OrdinalEncoder(categories=[education_order], dtype=int
```

```
binary_transformer = OneHotEncoder(drop = 'if_binary', dtype=int, handle_unk
```

In [18]:
```
preprocessor = make_column_transformer(
    (numeric_transformer, numeric_features),
    (ordinal_transformer, ordinal_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
    ("drop", drop_features),
)

preprocessor.fit(X_train)
preprocessed_X_train = preprocessor.transform(X_train)
preprocessed_X_test = preprocessor.transform(X_test)

# write raw data "data/processed" directory
preprocessed_X_train.to_csv("data/processed/preprocessed_X_train.csv")
preprocessed_X_test.to_csv("data/processed/preprocessed_X_test.csv")

preprocessed_X_train
```

Out[18]:

	standardscaler__age	standardscaler__balance	standardscaler__duration	stan
30605	-0.930271	-0.450231	0.103187	
26394	-0.930271	-0.300735	-0.567378	
36950	-0.261847	3.579522	0.743273	
14150	1.838917	0.287948	3.276944	
38910	0.311089	-0.336282	-0.917901	
...	...	...	...	
41344	1.361471	0.171673	-0.399737	
10384	-0.261847	-0.282131	-0.525468	
42552	1.743428	0.468008	-0.087314	
33759	-0.930271	-0.225654	-0.517848	
43515	-1.121250	-0.299074	0.545151	

25915 rows × 24 columns

Model training

We did hyperparameter optimisation for the following classification models: k-nearest neighbouts classifier, support vector machine, logistic regression, and random forest model. To find the best model, we performed 5-fold cross validation within GridSearch

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js s our metric of model prediction performance.

```
In [19]: model_comparison = { "model_name": [], "mean_train_score": [], "mean_test_sc
```

K-Nearest Neighbors

```
In [20]: # tune model (here, find k for k-Nearest Neighbors classification using 5 fo
knn_pipe = make_pipeline(preprocessor, KNeighborsClassifier(n_jobs=-1))
parameter_grid = {
    "kneighborsclassifier__n_neighbors": np.arange(1, 20, 2),
}

cv = 5
knn_search = GridSearchCV(
    estimator=knn_pipe,
    param_grid=parameter_grid,
    cv=cv,
    scoring=make_scorer(fbeta_score, pos_label='yes', beta=5),
    n_jobs=-1,
    return_train_score=True,
)

bank_marketing_fit_knn = knn_search.fit(X_train, y_train)
```

```
In [21]: best_model_scores_knn = pd.DataFrame(knn_search.cv_results_)[ [
                 "mean_test_score",
                 "mean_train_score",
                 "param_kneighborsclassifier__n_neighbors",
                 "rank_test_score",
] ].set_index("rank_test_score").sort_index().iloc[1, :]
model_comparison["model_name"].append("K-Nearest Neighbors")
model_comparison["mean_train_score"].append(best_model_scores_knn["mean_trai
model_comparison["mean_test_score"].append(best_model_scores_knn["mean_test_
```

```
In [22]: accuracies_grid_knn = pd.DataFrame(bank_marketing_fit_knn.cv_results_)
accuracies_grid_knn = (
    accuracies_grid_knn[[
        "param_kneighborsclassifier__n_neighbors",
        "mean_test_score",
        "std_test_score"
    ]]
    .assign(
        sem_test_score=accuracies_grid_knn["std_test_score"] / cv**(1/2),
        sem_test_score_lower=lambda df: df["mean_test_score"] - (df["sem_tes
        sem_test_score_upper=lambda df: df["mean_test_score"] + (df["sem_tes
    )
    .rename(columns={"param_kneighborsclassifier__n_neighbors": "k"})
    .drop(columns=["std_test_score"])
)

accuracies_grid_knn.sort_values("mean_test_score", ascending=False)
```

Out[22]:

	k	mean_test_score	sem_test_score	sem_test_score_lower	sem_test_score_upp
0	1	0.361408	0.007098	0.357859	0.36495
1	3	0.302972	0.010859	0.297543	0.30840
2	5	0.290297	0.007882	0.286356	0.29423
3	7	0.281309	0.008024	0.277297	0.28532
4	9	0.262828	0.007445	0.259106	0.26655
5	11	0.252995	0.004642	0.250674	0.25531
6	13	0.238278	0.004567	0.235994	0.24056
7	15	0.230632	0.003704	0.228781	0.23248
8	17	0.226997	0.004469	0.224762	0.22923
9	19	0.222738	0.004967	0.220254	0.22522

In [23]:

```
line_n_point_knn = alt.Chart(accuracies_grid_knn, width=600).mark_line(color
    x=alt.X("k", scale=alt.Scale(type='log'), title="k"),
    y=alt.Y("mean_test_score")
        .scale(zero=False)
        .title("F-beta score (beta = 5)")
)

line_n_point_knn + line_n_point_knn.mark_circle(color='black')
```

Out[23]:

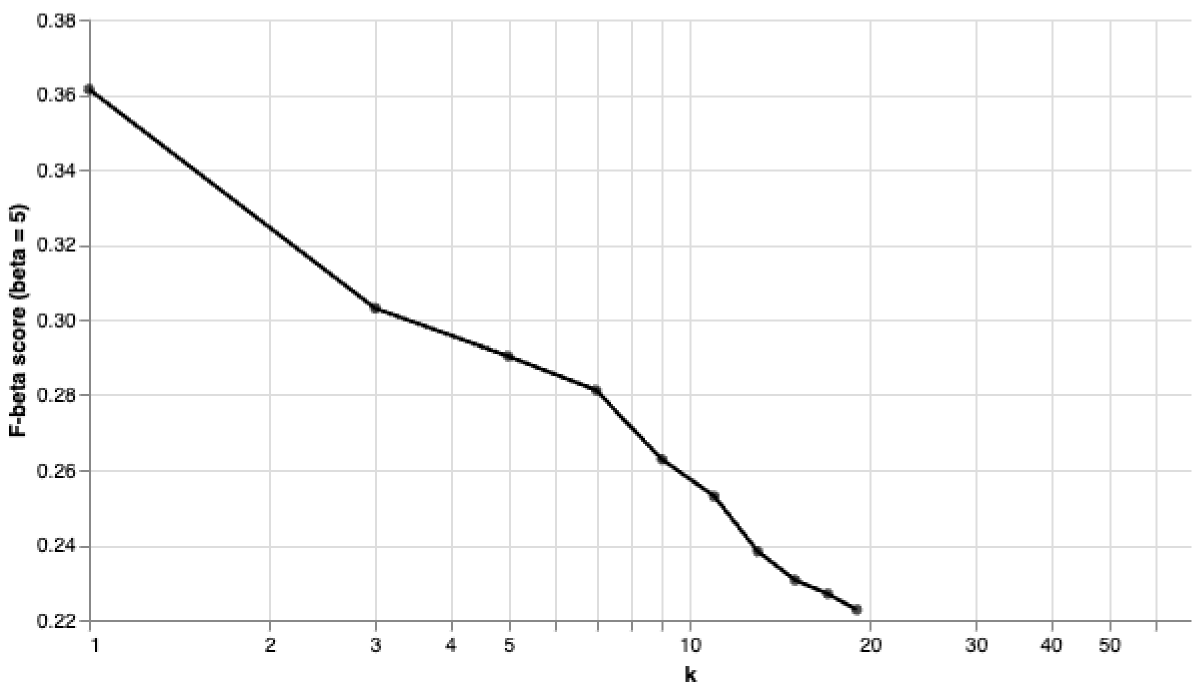


Figure 4. Results from 5-fold cross validation of k-NN model to choose K. F-beta score (with beta = 5) was used as the classification metric as K was varied.

SVM

In [24]:
```
# tune model (here, find C for SVM model using 5 fold cv)
svc_pipe = make_pipeline(preprocessor, SVC(class_weight={'no': 1, 'yes': 10}
parameter_grid = {
    "svc__C": 10.0 ** np.arange(-3, 3, 1),
}

cv = 5
svc_search =GridSearchCV(
    estimator=svc_pipe,
    param_grid=parameter_grid,
    cv=cv,
    scoring=make_scorer(fbeta_score, pos_label='yes', beta=5),
    n_jobs=-1,
    return_train_score=True,
)

bank_marketing_fit_svc = svc_search.fit(X_train, y_train)
```

In [25]:
```
best_model_scores_svc = pd.DataFrame(svc_search.cv_results_)[ [
                 "mean_test_score",
                 "mean_train_score",
                 "param_svc__C",
                 "rank_test_score",
] ].set_index("rank_test_score").sort_index().iloc[1, :]
model_comparison["model_name"].append("SVC RBF")
model_comparison["mean_train_score"].append(best_model_scores_svc["mean_trai
model_comparison["mean_test_score"].append(best_model_scores_svc["mean_test_
```

In [26]:
```
accuracies_grid_svc = pd.DataFrame(bank_marketing_fit_svc.cv_results_)
accuracies_grid_svc = (
    accuracies_grid_svc[[
        "param_svc__C",
        "mean_test_score",
        "std_test_score"
    ]]
    .assign(
        sem_test_score=accuracies_grid_svc["std_test_score"] / cv**(1/2),
        sem_test_score_lower=lambda df: df["mean_test_score"] - (df["sem_tes
        sem_test_score_upper=lambda df: df["mean_test_score"] + (df["sem_tes
    )
    .rename(columns={"param_svc__C": "C"})
    .drop(columns=["std_test_score"])
)

accuracies_grid_svc.sort_values("mean_test_score", ascending=False)
```

Out[26]:

	C	mean_test_score	sem_test_score	sem_test_score_lower	sem_test_score_u
2	0.1	0.813766	0.004250	0.811640	0.81
1	0.01	0.806599	0.005022	0.804088	0.80
3	1.0	0.797937	0.004704	0.795585	0.80
0	0.001	0.796062	0.005700	0.793212	0.79
4	10.0	0.721276	0.004015	0.719269	0.72
5	100.0	0.606725	0.010947	0.601251	0.61

In [27]:

```
line_n_point_svc = alt.Chart(accuracies_grid_svc, width=600).mark_line(color
    x=alt.X("C", scale=alt.Scale(type='log'), title="C"),
    y=alt.Y("mean_test_score")
        .scale(zero=False)
        .title("F-beta (beta = 5)")
)

line_n_point_svc + line_n_point_svc.mark_circle(color='black')
```

Out[27]:

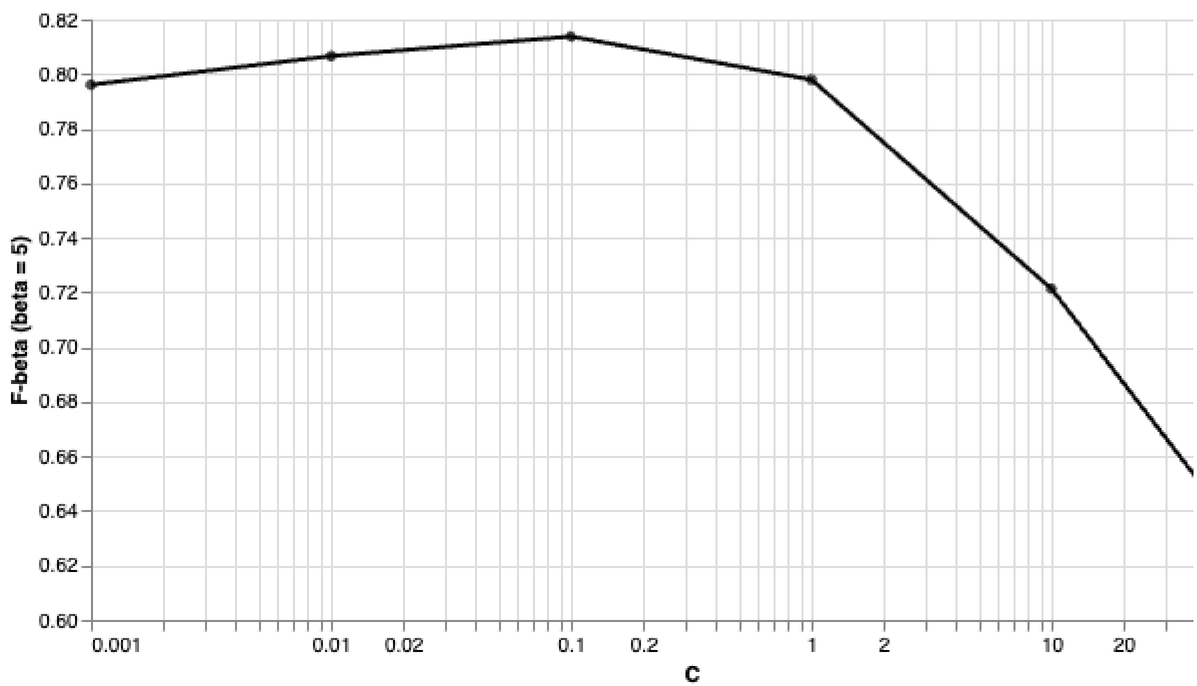


Figure 5. Results from 5-fold cross validation of SVM model to choose C. F-beta score (with beta = 5) was used as the classification metric as C was varied.

Logistic Regression

In [28]:

```
# tune model (here, find C for logistic regression using 5 fold cv)
lr_pipe = make_pipeline(preprocessor, LogisticRegression(max_iter=500, class

parameter_grid = {
    "logisticregression__C": 10.0 ** np.arange(-3, 5, 1),
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
}

cv = 5
lr_search = GridSearchCV(
    estimator=lr_pipe,
    param_grid=parameter_grid,
    cv=cv,
    scoring=make_scorer(fbeta_score, pos_label='yes', beta=5),
    n_jobs=-1,
    return_train_score=True,
)

bank_marketing_fit_lr = lr_search.fit(X_train, y_train)
```

In [29]:
```
best_model_scores_lr = pd.DataFrame(lr_search.cv_results_)[ [
                 "mean_test_score",
                 "mean_train_score",
                 "param_logisticregression__C",
                 "rank_test_score",
] ].set_index("rank_test_score").sort_index().iloc[1, :]
model_comparison["model_name"].append("Logistic Regression")
model_comparison["mean_train_score"].append(best_model_scores_lr["mean_train
model_comparison["mean_test_score"].append(best_model_scores_lr["mean_test_s
```

In [30]:
```
accuracies_grid_lr = pd.DataFrame(bank_marketing_fit_lr.cv_results_)
accuracies_grid_lr = (
    accuracies_grid_lr[[
        "param_logisticregression__C",
        "mean_test_score",
        "std_test_score"
    ]]
    .assign(
        sem_test_score=accuracies_grid_lr["std_test_score"] / cv**(1/2),
        sem_test_score_lower=lambda df: df["mean_test_score"] - (df["sem_tes
        sem_test_score_upper=lambda df: df["mean_test_score"] + (df["sem_tes
    )
    .rename(columns={"param_logisticregression__C": "C"})
    .drop(columns=["std_test_score"])
)

accuracies_grid_lr.sort_values("mean_test_score", ascending=False)
```

Out[30]:

	C	mean_test_score	sem_test_score	sem_test_score_lower	sem_test_score
0	0.001	0.777437	0.007850	0.773512	0.
3	1.0	0.777166	0.005560	0.774386	0.7
4	10.0	0.777137	0.005545	0.774365	0.
5	100.0	0.777137	0.005545	0.774365	0.
6	1000.0	0.777137	0.005545	0.774365	0.
7	10000.0	0.777137	0.005545	0.774365	0.
2	0.1	0.776632	0.005775	0.773745	0.
1	0.01	0.775215	0.006968	0.771731	0.7

In [31]:

```
line_n_point_lr = alt.Chart(accuracies_grid_lr, width=600).mark_line(color="
    x=alt.X("C", scale=alt.Scale(type='log'), title="C"),
    y=alt.Y("mean_test_score")
        .scale(zero=False)
        .title("F-beta score (beta = 5)")
)

line_n_point_lr + line_n_point_lr.mark_circle(color='black')
```

Out[31]:

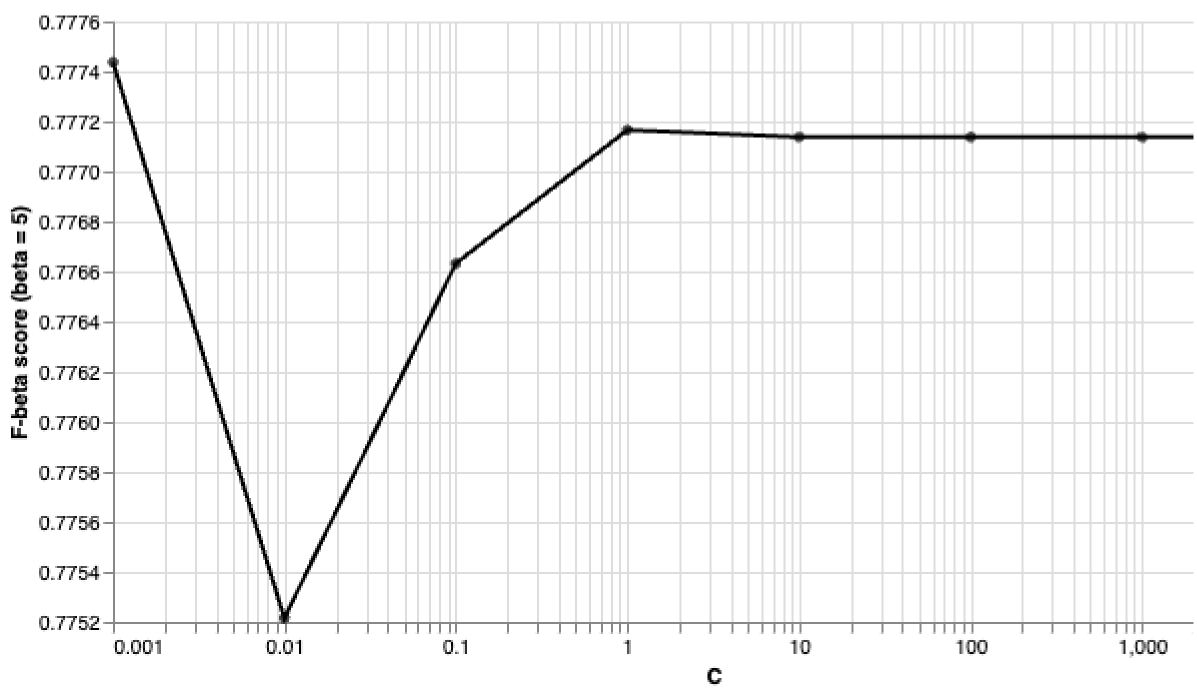


Figure 6. Results from 5-fold cross validation of logistic regression model to choose C. F-beta score (with beta = 5) was used as the classification metric as C was varied.

Random Forest

In [32]:

```
# tune model (here, find n_estimator and max_depth for random forest classif
                                      eprocessor, RandomForestClassifier(n_jobs=-1, clas
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
parameter_grid = {
    "randomforestclassifier__n_estimators": [100, 200, 300, 400, 500],
    "randomforestclassifier__max_depth": [3, 5, 7, 15, None],
}

cv = 5
rf_search = GridSearchCV(
    estimator=rf_pipe,
    param_grid=parameter_grid,
    cv=cv,
    scoring=make_scorer(fbeta_score, pos_label='yes', beta=5),
    n_jobs=-1,
    return_train_score=True,
)

bank_marketing_fit_rf = rf_search.fit(X_train, y_train)
```

In [33]:

```
best_model_scores_rf = pd.DataFrame(rf_search.cv_results_)[ [
                 "mean_test_score",
                 "mean_train_score",
                 "param_randomforestclassifier__n_estimators",
                 "param_randomforestclassifier__max_depth",
                 "rank_test_score",
] ].set_index("rank_test_score").sort_index().iloc[1, :]
model_comparison["model_name"].append("Random Forest")
model_comparison["mean_train_score"].append(best_model_scores_rf["mean_trair
model_comparison["mean_test_score"].append(best_model_scores_rf["mean_test_s
```

In [34]:

```
accuracies_grid_rf = pd.DataFrame(bank_marketing_fit_rf.cv_results_)
accuracies_grid_rf = (
    accuracies_grid_rf[[
        "param_randomforestclassifier__n_estimators",
        "param_randomforestclassifier__max_depth",
        "mean_test_score",
        "std_test_score"
    ]]
    .assign(
        sem_test_score=accuracies_grid_rf["std_test_score"] / cv**(1/2),
        sem_test_score_lower=lambda df: df["mean_test_score"] - (df["sem_tes
        sem_test_score_upper=lambda df: df["mean_test_score"] + (df["sem_tes
    )
    .rename(columns={"param_randomforestclassifier__n_estimators": "n_estima
    .drop(columns=["std_test_score"])
)

accuracies_grid_rf.sort_values("mean_test_score", ascending=False)
```

Out[34]:

	n_estimators	max_depth	mean_test_score	sem_test_score	sem_test_score_low
0	100	3	0.835923	0.004141	0.83385
4	500	3	0.833813	0.003660	0.83198
3	400	3	0.832944	0.003780	0.83105
1	200	3	0.832356	0.003539	0.83058
2	300	3	0.831250	0.004722	0.82888
5	100	5	0.824774	0.004523	0.8225
6	200	5	0.824121	0.002295	0.82297
8	400	5	0.823556	0.003602	0.82175
9	500	5	0.823294	0.003057	0.82176
7	300	5	0.823221	0.003825	0.82130
13	400	7	0.818038	0.005740	0.81516
12	300	7	0.814254	0.006290	0.81110
14	500	7	0.813344	0.007307	0.80969
11	200	7	0.812939	0.006301	0.80978
10	100	7	0.812818	0.006106	0.80976
17	300	15	0.626264	0.002869	0.62483
15	100	15	0.623903	0.003323	0.62224
18	400	15	0.622868	0.003306	0.6212
19	500	15	0.621905	0.001654	0.62107
16	200	15	0.616901	0.003942	0.61493
21	200	None	0.249829	0.005120	0.24726
23	400	None	0.247809	0.003823	0.24589
24	500	None	0.245479	0.005192	0.24288
22	300	None	0.245203	0.004713	0.24284
20	100	None	0.243830	0.003819	0.2419

Model comparison

Out of the above models, the random forest model performed the best, with its best, hyperparameter-optimised model having a mean test score of 0.833765, which was the highest mean test score for the optimised models. We thus decided to use the random forest model for our final predictions with the test data.

In [35]:
```
pd.DataFrame(model_comparison)
```

Out[35]:

	model_name	mean_train_score	mean_test_score
0	K-Nearest Neighbors	0.538419	0.302972
1	SVC RBF	0.810876	0.806599
2	Logistic Regression	0.778668	0.777166
3	Random Forest	0.838477	0.833813

Table 1. Performance comparison across all models.

Prediction

The random forest model performed similarly on the test data when compared to the training data, having an F-beta score (beta = 5) of 0.817092 on the test data. This was only slightly lower than the mean test score of the best model after cross validation using the training data, which was 0.833765. This relatively high F-beta score and the small gap between the scores indicates that the model is quite good at predicting whether customers will subscribe to the term deposit, and is likely to generalise well to unseen data. It had quite a low accuracy, with 5492 false positives out of the 1807 actual positives. This is expected as we heavily favoured recall, and acceptable as the high number of false positives is not of large consequence to the bank.

In [36]:
```
# Compute accuracy
y_pred = bank_marketing_fit_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Compute F-beta score (beta = 5)
bank_marketing_preds = X_test.assign(
    predicted=bank_marketing_fit_rf.predict(X_test)
)
f_beta_5_score = fbeta_score(
    y_test,
    bank_marketing_preds['predicted'],
    beta=5,
    pos_label='yes'
)

pd.DataFrame({'accuracy': [accuracy], 'F-beta score (beta = 5)': [f_beta_5_s
```

Out[36]:

	accuracy	F-beta score (beta = 5)
0	0.676294	0.815119

Table 2. Accuracy and F-beta score of model performance on test data.

In [37]:
```python
pd.crosstab(
    y_test,
    bank_marketing_preds['predicted'],
)
```

Out[37]:

predicted	no	yes
subscribed		
no	9886	5384
yes	209	1799

Table 3. Confusion matrix of model performance on test data.

Discussion

As the F-beta score (beta = 5) score of the model is quite high and the model does not seem to be overfit to the training data, it is probably safe to apply this model to new customers, and to predict whether they will be interested in subscribing to the term deposit. This means that the bank can target ads and direct marketing calls about this term deposit, and potentially, other related products, to this specific group of customers, and can expect that the success rate would be quite high compared to a random group of customers.

While the high number of false positives is acceptable given the low-stakes nature of having false positives, it would still be beneficial to the bank to improve the performance of our model, and to reduce the number of false positives. In the future, the model may be refined by including more data points, which might help to train the model better. More relevant features may also be included to train the model better, and feature engineering may be carried out to further refine the model.

References

- Bauer, C. L., & Miglautsch, J. (1992). A conceptual definition of direct marketing. Journal of Direct Marketing, 6(2), 7–17. https://doi.org/10.1002/dir.4000060204
- Harris, C.R. et al., 2020. Array programming with NumPy. Nature, 585, pp.357–362.
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.
- McKinney, Wes. 2010. “Data Structures for Statistical Computing in Python.” In Proceedings of the 9th Python in Science Conference, edited by Stéfan van der Walt and Jarrod Millman, 51–56.
- Moro, S., Cortez, P., & Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. Decision Support Systems, 62, 22–31.

https://doi.org/10.1016/j.dss.2014.03.001

- Moro,S., Rita,P., and Cortez,P.. (2012). Bank Marketing. UCI Machine Learning Repository. https://doi.org/10.24432/C5K306.
- Nowak, G. J., & Phelps, J. (1995). Direct marketing and the use of individual-level consumer information: Determining how and when "privacy" matters. Journal of Direct Marketing, 9(3), 46–60. https://doi.org/10.1002/dir.4000090307
- Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
- VanderPlas, J. et al., 2018. Altair: Interactive statistical visualizations for python. Journal of open source software, 3(32), p.1057.
- Van Rossum, Guido, and Fred L. Drake. 2009. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.