

er- Sudarshan Kumar

Sunil Mathur

Sally Cao

Amanda Morley

Drew McKnight

Project Title : Bank Marketing

The data is related to 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 assess if the product (bank term deposit) would be subscribed (or not).

The classification goal is to predict if the client will subscribe to a term deposit (variable y).

Attribute information

For more information, visit <https://www.openml.org/1461>.

Input variables:

Bank Client Data:

- 1- age (numeric)
- 2- job : type of job (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
- 3- marital : marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
- 4- education (categorical: "unknown", "secondary", "primary", "tertiary")
- 5- default: has credit in default? (binary: "yes", "no")
- 6- balance: average yearly balance, in euros (numeric)
- 7- housing: has housing loan? (binary: "yes", "no")
- 8- loan: has personal loan? (binary: "yes", "no")
 - related with the last contact of the current campaign:
- 9- contact: contact communication type (categorical: "unknown", "telephone", "cellular")
- 10- day: last contact day of the month (numeric)
- 11- month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
- 12- duration: last contact duration, in seconds (numeric)
- other attributes:
- 13- campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
- 14- pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
- 15- previous: number of contacts performed before this campaign and for this client (numeric)
- 16- poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success")
 - output variable (desired target):
- 17- y - has the client subscribed a term deposit? (binary: "yes", "no")

2 - yes; 1 = no (We are changing this variable so that 1 simply means subscribed)

EDA Observations:

- The Bank phone marketing campaign data set has 45,211 rows and 17 columns including the target or Y variable.
- The dataset is very clean with no null values found.
- Some key features that may be potentially important:
 - Age:** Mean age is 41 years. Range is from 18 - 95, mid 50% quartile is 33-48
 - **Hypothesis:** Those under 30 and greater than 60 may have more money to be able to open a deposit account
 - Average Balance:** Mean balance is 1.3 K Range is -8k to 102K
 - **Hypothesis:** Those with more average balance will likely have money to open a deposit account
 - Housing Loan/Personal Loan:** Either of these increases the liability.
 - **Hypothesis:** People with loans may not have enough money to open a deposit account
 - Education:** People with higher education are likely to be high earners and may have enough spare cash to open a deposit account.
 - Job:** People in management / technical jobs may be good candidates as students / entrepreneurs / self-employed may have more money to open a deposit account. Retired folks may be good candidates as well.
 - Reasons we find it interesting: It has the most categories in a feature compared to the other features and it is something we can leverage in focusing on certain jobs or removing ones that do not seem significant
 - Marital Status:** Singles may have more money versus married with family responsibilities
 - Contact:** Cellular seems to be resulting in higher conversion rates
 - **Hypothesis:** Those with cellular phones may have more disposable income or calling someone on a cellular phone yields a better chance of response.
 - Credit Default:** people with credit default may not be the best candidates to open a deposit account
- Previous campaigns - may have a role in someone deciding to open a deposit account
- Potential Issues:** Call Duration appears to be highly correlated. However, it is unknown till the end of the call. We are unsure if we should keep or drop this variable as it is highly correlated, however we won't know the result unless a marketing call is accepted.
- Potential Outliers:** In the box plot for the feature 'previous' (which is the number of times the client was contacted before the campaign) has an outlier where there is one client who was contacted 273 times would be a feature; we would want to remove outliers for so that our dataset would not be skewed if we wanted to keep the 'previous' feature for our data processing.

Recommended variables to focus on:

- Duration: highly correlated but we are unsure if this is a fair variable to use given it is necessary for a contact to be contacted definitely. If we find that this is a valuable variable this could be something that we consider to keep the customer on the phone longer as that seems to lead to more successful conversions.
- Job type is also an interesting variable to us, we see that students and retired workers are highly correlated so we would want to further investigate this. Perhaps the students are more educated and willing to subscribe and the retirees have more disposable income.
- Marital type is an interesting variable as those that are single seem to be highly correlated, perhaps those that are married will invest in something else like college funds.
- The type of contact also seems to make a difference, those that had cellular phones were more likely to subscribe and this could be for reasons like those that had cellular phones have more money versus married with family responsibilities
- Previous outcome success is also an interesting variable and could be used as a top point where we make our calls to. If we call those who had previous success with first this might increase our marketing power.

Full Pipeline and Model Deployment Summary:

- We created 7 models with accuracies ranging from ~85-90%, the highest being from the Random Forest.
- Decision Tree
 - Random Forest
 - SVM
 - XGBoost
 - Naive Bayes
- Features we used for the models:
 - age
 - balance
 - day
 - duration
 - pdays
 - housing
 - month
 - poutcome
 - contact
 - marital

Conclusion

Baseline Accuracy for predicting successful Customer subscription using feature "poutcome" (Previous campaign success) is 88.3 %. Our best performing model is "Random Forest" with 90.63% accuracy. Based on the random forest, duration seems to be the strongest predictor for whether a customer will subscribe or not with a feature importance score of ~34%. This could be due to the fact that customers who are willing to stay longer on the phone with a bank representative have a higher chance of being persuaded to subscribe to a bank deposit.

Import Library

```
In [1]: # Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn') # change the default style
cbold = '\033[01m'
cEnd = '\033[0m'
```

READ DATA

```
In [2]: # read csv data into pandas dataframe
df = pd.read_csv('projectdataset-1.csv')
```

```
In [3]: # basic shape, data type, null values: dataset looks pretty good.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column        Non-Null Count  Dtype
---  --
 0   age           45211 non-null    int64
 1   job           45211 non-null    object
 2   marital       45211 non-null    object
 3   education     45211 non-null    object
 4   default       45211 non-null    object
 5   balance       45211 non-null    int64
 6   housing       45211 non-null    object
 7   loan         45211 non-null    object
 8   contact       45211 non-null    object
 9   day           45211 non-null    int64
10   month        45211 non-null    object
11   duration      45211 non-null    int64
12   campaign     45211 non-null    int64
13   pdays        45211 non-null    int64
14   previous     45211 non-null    int64
15   poutcome     45211 non-null    object
16   class        45211 non-null    int64
dtypes: int64(11), object(6)
memory usage: 5.3+ MB
```

```
In [4]: # First 5 lines of data
df.head()
```

```
Out[4]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	p
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	
1	44	technician	single	secondary	no	29	yes	yes	unknown	5	may	151	1	-1	0	
2	43	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	
3	47	blue-collar	single	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	
4	33	unknown	married	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	

```
In [5]: # Prepare the data by separating X and y
# dropping y variable
```

```
# axis = 1 below means dropping by columns, 0 means by rows
df.Class.replace([1,2],0,1, inplace=True) # 0 - Not Subscribed, 1 - Subscribed
X = df.drop(['Class'], axis=1)
y = df['Class']
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype
---  --
 0   age           45211 non-null    int64
 1   job           45211 non-null    object
 2   marital       45211 non-null    object
 3   education     45211 non-null    object
 4   default       45211 non-null    object
 5   balance       45211 non-null    int64
 6   housing       45211 non-null    object
 7   loan         45211 non-null    object
 8   contact       45211 non-null    object
 9   day           45211 non-null    int64
10   month        45211 non-null    object
11   duration      45211 non-null    int64
12   campaign     45211 non-null    int64
13   pdays        45211 non-null    int64
14   previous     45211 non-null    int64
15   poutcome     45211 non-null    object
dtypes: int64(11), object(5)
memory usage: 5.5+ MB
```

```
In [6]: # basic stats
df.describe(include='all')
```

```
Out[6]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	
count	45211.000000	45211	45211	45211	45211	45211.000000	45211	45211	45211	45211.000000	45211	45211.000000	45
unique	NaN	NaN	12	3	4	2	NaN	yes	no	3	NaN	12	NaN
top	NaN	blue-collar	married	secondary	no	NaN	yes	no	cellular	NaN	may	NaN	
freq	NaN	9732	27214	23202	44396	NaN	25130	37967	29285	NaN	13766	NaN	
mean	40.936270	NaN	NaN	NaN	NaN	1362.272058	NaN	NaN	NaN	NaN	15.806419	NaN	258.163080
std	10.618762	NaN	NaN	NaN	NaN	3044.765829	NaN	NaN	NaN	NaN	8.322476	NaN	257.527812
min	18.000000	NaN	NaN	NaN	NaN	-8019.000000	NaN	NaN	NaN	NaN	8.000000	NaN	0.000000
25%	33.000000	NaN	NaN	NaN	NaN	72.000000	NaN	NaN	NaN	NaN	8.000000	NaN	103.000000
50%	39.000000	NaN	NaN	NaN	NaN	448.000000	NaN	NaN	NaN	NaN	16.000000	NaN	180.000000
75%	48.000000	NaN	NaN	NaN	NaN	1428.000000	NaN	NaN	NaN	NaN	21.000000	NaN	319.000000
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	NaN	NaN	NaN	31.000000	NaN	4918.000000

EDA

```
In [7]: # histograms for all numerical features
X.hist(figsize=(15,15))
```

```
Out[7]: array([[<AxesSubplot:Figure(1,1)>],
[<AxesSubplot:Figure(1,2)>],
[<AxesSubplot:Figure(1,3)>],
[<AxesSubplot:Figure(1,4)>],
[<AxesSubplot:Figure(1,5)>],
[<AxesSubplot:Figure(1,6)>],
[<AxesSubplot:Figure(1,7)>],
[<AxesSubplot:Figure(1,8)>],
[<AxesSubplot:Figure(1,9)>],
[<AxesSubplot:Figure(1,10)>],
[<AxesSubplot:Figure(1,11)>],
[<AxesSubplot:Figure(1,12)>],
[<AxesSubplot:Figure(1,13)>],
[<AxesSubplot:Figure(1,14)>],
[<AxesSubplot:Figure(1,15)>],
[<AxesSubplot:Figure(1,16)>],
[<AxesSubplot:Figure(1,17)>],
[<AxesSubplot:Figure(1,18)>],
[<AxesSubplot:Figure(1,19)>],
[<AxesSubplot:Figure(1,20)>],
[<AxesSubplot:Figure(1,21)>],
[<AxesSubplot:Figure(1,22)>],
[<AxesSubplot:Figure(1,23)>],
[<AxesSubplot:Figure(1,24)>],
[<AxesSubplot:Figure(1,25)>],
[<AxesSubplot:Figure(1,26)>],
[<AxesSubplot:Figure(1,27)>],
[<AxesSubplot:Figure(1,28)>],
[<AxesSubplot:Figure(1,29)>],
[<AxesSubplot:Figure(1,30)>],
[<AxesSubplot:Figure(1,31)>],
[<AxesSubplot:Figure(1,32)>],
[<AxesSubplot:Figure(1,33)>],
[<AxesSubplot:Figure(1,34)>],
[<AxesSubplot:Figure(1,35)>],
[<AxesSubplot:Figure(1,36)>],
[<AxesSubplot:Figure(1,37)>],
[<AxesSubplot:Figure(1,38)>],
[<AxesSubplot:Figure(1,39)>],
[<AxesSubplot:Figure(1,40)>],
[<AxesSubplot:Figure(1,41)>],
[<AxesSubplot:Figure(1,42)>],
[<AxesSubplot:Figure(1,43)>],
[<AxesSubplot:Figure(1,44)>],
[<AxesSubplot:Figure(1,45)>],
[<AxesSubplot:Figure(1,46)>],
[<AxesSubplot:Figure(1,47)>],
[<AxesSubplot:Figure(1,48)>],
[<AxesSubplot:Figure(1,49)>],
[<AxesSubplot:Figure(1,50)>],
[<AxesSubplot:Figure(1,51)>],
[<AxesSubplot:Figure(1,52)>],
[<AxesSubplot:Figure(1,53)>],
[<AxesSubplot:Figure(1,54)>],
[<AxesSubplot:Figure(1,55)>],
[<AxesSubplot:Figure(1,56)>],
[<AxesSubplot:Figure(1,57)>],
[<AxesSubplot:Figure(1,58)>],
[<AxesSubplot:Figure(1,59)>],
[<AxesSubplot:Figure(1,60)>],
[<AxesSubplot:Figure(1,61)>],
[<AxesSubplot:Figure(1,62)>],
[<AxesSubplot:Figure(1,63)>],
[<AxesSubplot:Figure(1,64)>],
[<AxesSubplot:Figure(1,65)>],
[<AxesSubplot:Figure(1,66)>],
[<AxesSubplot:Figure(1,67)>],
[<AxesSubplot:Figure(1,68)>],
[<AxesSubplot:Figure(1,69)>],
[<AxesSubplot:Figure(1,70)>],
[<AxesSubplot:Figure(1,71)>],
[<AxesSubplot:Figure(1,72)>],
[<AxesSubplot:Figure(1,73)>],
[<AxesSubplot:Figure(1,74)>],
[<AxesSubplot:Figure(1,75)>],
[<AxesSubplot:Figure(1,76)>],
[<AxesSubplot:Figure(1,77)>],
[<AxesSubplot:Figure(1,78)>],
[<AxesSubplot:Figure(1,79)>],
[<AxesSubplot:Figure(1,80)>],
[<AxesSubplot:Figure(1,81)>],
[<AxesSubplot:Figure(1,82)>],
[<AxesSubplot:Figure(1,83)>],
[<AxesSubplot:Figure(1,84)>],
[<AxesSubplot:Figure(1,85)>],
[<AxesSubplot:Figure(1,86)>],
[<AxesSubplot:Figure(1,87)>],
[<AxesSubplot:Figure(1,88)>],
[<AxesSubplot:Figure(1,89)>],
[<AxesSubplot:Figure(1,90)>],
[<AxesSubplot:Figure(1,91)>],
[<AxesSubplot:Figure(1,92)>],
[<AxesSubplot:Figure(1,93)>],
[<AxesSubplot:Figure(1,94)>],
[<AxesSubplot:Figure(1,95)>],
[<AxesSubplot:Figure(1,96)>],
[<AxesSubplot:Figure(1,97)>],
[<AxesSubplot:Figure(1,98)>],
[<AxesSubplot:Figure(1,99)>],
[<AxesSubplot:Figure(1,100)>],
[<AxesSubplot:Figure(1,101)>],
[<AxesSubplot:Figure(1,102)>],
[<AxesSubplot:Figure(1,103)>],
[<AxesSubplot:Figure(1,104)>],
[<AxesSubplot:Figure(1,105)>],
[<AxesSubplot:Figure(1,106)>],
[<AxesSubplot:Figure(1,107)>],
[<AxesSubplot:Figure(1,108)>],
[<AxesSubplot:Figure(1,109)>],
[<AxesSubplot:Figure(1,110)>],
[<AxesSubplot:Figure(1,111)>],
[<AxesSubplot:Figure(1,112)>],
[<AxesSubplot:Figure(1,113)>],
[<AxesSubplot:Figure(1,114)>],
[<AxesSubplot:Figure(1,115)>],
[<AxesSubplot:Figure(1,116)>],
[<AxesSubplot:Figure(1,117)>],
[<AxesSubplot:Figure(1,118)>],
[<AxesSubplot:Figure(1,119)>],
[<AxesSubplot:Figure(1,120)>],
[<AxesSubplot:Figure(1,121)>],
[<AxesSubplot:Figure(1,122)>],
[<AxesSubplot:Figure(1,123)>],
[<AxesSubplot:Figure(1,124)>],
[<AxesSubplot:Figure(1,125)>],
[<AxesSubplot:Figure(1,126)>],
[<AxesSubplot:Figure(1,127)>],
[<AxesSubplot:Figure(1,128)>],
[<AxesSubplot:Figure(1,129)>],
[<AxesSubplot:Figure(1,130)>],
[<AxesSubplot:Figure(1,131)>],
[<AxesSubplot:Figure(1,132)>],
[<AxesSubplot:Figure(1,133)>],
[<AxesSubplot:Figure(1,134)>],
[<AxesSubplot:Figure(1,135)>],
[<AxesSubplot:Figure(1,136)>],
[<AxesSubplot:Figure(1,137)>],
[<AxesSubplot:Figure(1,138)>],
[<AxesSubplot:Figure(1,139)>],
[<AxesSubplot:Figure(1,140)>],
[<AxesSubplot:Figure(1,141)>],
[<AxesSubplot:Figure(1,142)>],
[<AxesSubplot:Figure(1,143)>],
[<AxesSubplot:Figure(1,144)>],
[<AxesSubplot:Figure(1,145)>],
[<AxesSubplot:Figure(1,146)>],
[<AxesSubplot:Figure(1,147)>],
[<AxesSubplot:Figure(1,148)>],
[<AxesSubplot:Figure(1,149)>],
[<AxesSubplot:Figure(1,150)>],
[<AxesSubplot:Figure(1,151)>],
[<AxesSubplot:Figure(1,152)>],
[<AxesSubplot:Figure(1,153)>],
[<AxesSubplot:Figure(1,154)>],
[<AxesSubplot:Figure(1,155)>],
[<AxesSubplot:Figure(1,156)>],
[<AxesSubplot:Figure(1,157)>],
[<AxesSubplot:Figure(1,158)>],
[<AxesSubplot:Figure(1,159)>],
[<AxesSubplot:Figure(1,160)>],
[<AxesSubplot:Figure(1,161)>],
[<AxesSubplot:Figure(1,162)>],
[<AxesSubplot:Figure(1,163)>],
[<AxesSubplot:Figure(1,164)>],
[<AxesSubplot:Figure(1,165)>],
[<AxesSubplot:Figure(1,166)>],
[<AxesSubplot:Figure(1,167)>],
[<AxesSubplot:Figure(1,168)>],
[<AxesSubplot:Figure(1,169)>],
[<AxesSubplot:Figure(1,170)>],
[<AxesSubplot:Figure(1,171)>],
[<AxesSubplot:Figure(1,172)>],
[<AxesSubplot:Figure(1,173)>],
[<AxesSubplot:Figure(1,174)>],
[<AxesSubplot:Figure(1,175)>],
[<AxesSubplot:Figure(1,176)>],
[<AxesSubplot:Figure(1,177)>],
[<AxesSubplot:Figure(1,178)>],
[<AxesSubplot:Figure(1,179)>],
[<AxesSubplot:Figure(1,180)>],
[<AxesSubplot:Figure(1,181)>],
[<AxesSubplot:Figure(1,182)>],
[<AxesSubplot:Figure(1,183)>],
[<AxesSubplot:Figure(1,184)>],
[<AxesSubplot:Figure(1,185)>],
[<AxesSubplot:Figure(1,186)>],
[<AxesSubplot:Figure(1,187)>],
[<AxesSubplot:Figure(1,188)>],
[<AxesSubplot:Figure(1,189)>],
[<AxesSubplot:Figure(1,190)>],
[<AxesSubplot:Figure(1,191)>],
[<AxesSubplot:Figure(1,192)>],
[<AxesSubplot:Figure(1,193)>],
[<AxesSubplot:Figure(1,194)>],
[<AxesSubplot:Figure(1,195)>],
[<AxesSubplot:Figure(1,196)>],
[<AxesSubplot:Figure(1,197)>],
[<AxesSubplot:Figure(1,198)>],
[<AxesSubplot:Figure(1,199)>],
[<AxesSubplot:Figure(1,200)>],
[<AxesSubplot:Figure(1,201)>],
[<AxesSubplot:Figure(1,202)>],
[<AxesSubplot:Figure(1,203)>],
[<AxesSubplot:Figure(1,204)>],
[<AxesSubplot:Figure(1,205)>],
[<AxesSubplot:Figure(1,206)>],
[<AxesSubplot:Figure(1,207)>],
[<AxesSubplot:Figure(1,208)>],
[<AxesSubplot:Figure(1,209)>],
[<AxesSubplot:Figure(1,210)>],
[<AxesSubplot:Figure(1,211)>],
[<AxesSubplot:Figure(1,212)>],
[<AxesSubplot:Figure(1,213)>],
[<AxesSubplot:Figure(1,214)>],
[<AxesSubplot:Figure(1,215)>],
[<AxesSubplot:Figure(1,216)>],
[<AxesSubplot:Figure(1,217)>],
[<AxesSubplot:Figure(1,218)>],
[<AxesSubplot:Figure(1,219)>],
[<AxesSubplot:Figure(1,220)>],
[<AxesSubplot:Figure(1,221)>],
[<AxesSubplot:Figure(1,222)>],
[<AxesSubplot:Figure(1,223)>],
[<AxesSubplot:Figure(1,224)>],
[<AxesSubplot:Figure(1,225)>],
[<AxesSubplot:Figure(1,226)>],
[<AxesSubplot:Figure(1,227)>],
[<AxesSubplot:Figure(1,228)>],
[<AxesSubplot:Figure(1,229)>],
[<AxesSubplot:Figure(1,230)>],
[<AxesSubplot:Figure(1,231)>],
[<AxesSubplot:Figure(1,232)>],
[<AxesSubplot:Figure(1,233)>],
[<AxesSubplot:Figure(1,234)>],
[<AxesSubplot:Figure(1,235)>],
[<AxesSubplot:Figure(1,236)>],
[<AxesSubplot:Figure(1,237)>],
[<AxesSubplot:Figure(1,238)>],
[<AxesSubplot:Figure(1,239)>],
[<AxesSubplot:Figure(1,240)>],
[<AxesSubplot:Figure(1,241)>],
[<AxesSubplot:Figure(1,242)>],
[<AxesSubplot:Figure(1,243)>],
[<AxesSubplot:Figure(1,244)>],
[<AxesSubplot:Figure(1,245)>],
[<AxesSubplot:Figure(1,246)>],
[<AxesSubplot:Figure(1,247)>],
[<AxesSubplot:Figure(1,248)>],
[<AxesSubplot:Figure(1,249)>],
[<AxesSubplot:Figure(1,250)>],
[<AxesSubplot:Figure(1,251)>],
[<AxesSubplot:Figure(1,252)>],
[<AxesSubplot:Figure(1,253)>],
[<AxesSubplot:Figure(1,254)>],
[<AxesSubplot:Figure(1,255)>],
[<AxesSubplot:Figure(1,256)>],
[<AxesSubplot:Figure(1,257)>],
[<AxesSubplot:Figure(1,258)>],
[<AxesSubplot:Figure(1,259)>],
[<AxesSubplot:Figure(1,260)>],
[<AxesSubplot:Figure(1,261)>],
[<AxesSubplot:Figure(1,262)>],
[<AxesSubplot:Figure(1,263)>],
[<AxesSubplot:Figure(1,264)>],
[<AxesSubplot:Figure(1,265)>],
[<AxesSubplot:Figure(1,266)>],
[<AxesSubplot:Figure(1,267)>],
[<AxesSubplot:Figure(1,268)>],
[<AxesSubplot:Figure(1,269)>],
[<AxesSubplot:Figure(1,270)>],
[<AxesSubplot:Figure(1,271)>],
[<AxesSubplot:Figure(1,272)>],
[<AxesSubplot:Figure(1,273)>],
[<AxesSubplot:Figure(1,274)>],
[<AxesSubplot:Figure(1,275)>],
[<AxesSubplot:Figure(1,276)>],
[<AxesSubplot:Figure(1,277)>],
[<AxesSubplot:Figure(1,278)>],
[<AxesSubplot:Figure(1,279)>],
[<AxesSubplot:Figure(1,280)>],
[<AxesSubplot:Figure(1,281)>],
[<AxesSubplot:Figure(1,282)>],
[<AxesSubplot:Figure(1,283)>],
[<AxesSubplot:Figure(1,284)>],
[<AxesSubplot:Figure(1,285)>],
[<AxesSubplot:Figure(1,286)>],
[<AxesSubplot:Figure(1,287)>],
[<AxesSubplot:Figure(1,288)>],
[<AxesSubplot:Figure(1,289)>],
[<AxesSubplot:Figure(1,290)>],
[<AxesSubplot:Figure(1,291)>],
[<AxesSubplot:Figure(1,292)>],
[<AxesSubplot:Figure(1,293)>],
[<AxesSubplot:Figure(1,294)>],
[<AxesSubplot:Figure(1,295)>],
[<AxesSubplot:Figure(1,296)>],
[<AxesSubplot:Figure(1,297)>],
[<AxesSubplot:Figure(1,298)>],
[<AxesSubplot:Figure(1,299)>],
[<AxesSubplot:Figure(1,300)>],
[<AxesSubplot:Figure(1,301)>],
[<AxesSubplot:Figure(1,302)>],
[<AxesSubplot:Figure(1,303)>],
[<AxesSubplot:Figure(1,304)>],
[<AxesSubplot:Figure(1,305)>],
[<AxesSubplot:Figure(1,306)>],
[<AxesSubplot:Figure(1,307)>],
[<AxesSubplot:Figure(1,308)>],
[<AxesSubplot:Figure(1,309)>],
[<AxesSubplot:Figure(1,310)>],
[&lt
```



```

In [23]: X.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  --
 0   age         45211 non-null    int64  
 1   balance     45211 non-null    int64  
 2   housing     45211 non-null    object  
 3   contact     45211 non-null    object  
 4   day         45211 non-null    int64  
 5   month       45211 non-null    object  
 6   duration    45211 non-null    int64  
 7   pdays       45211 non-null    int64  
 8   poutcome    45211 non-null    object  
dtypes: int64(5), object(4)
memory usage: 3.1+ MB

```

In [24]: # We will train our decision tree classifier with the following features:

```

num_features = ['age', 'balance', 'day', 'duration', 'pdays']
cat_features = ['housing', 'contact', 'month', 'poutcome']

```

In [25]:

```

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from future_encoders import OneHotEncoder4 # requires future_encoders.py
# Create the preprocessing pipeline for numerical features
# There are two steps in this pipeline
# Pipeline(steps=(name1, transform1), (name2, transform2), ...)
# NOTE the step names can be arbitrary

# Step 1 is what we discussed before - filling the missing values if any using mean
# Step 2 is feature scaling via standardization - making features look like normal-distributed
# see standardization: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
num_pipeline = Pipeline(
    steps=[
        ('num_imputer', SimpleImputer()), # we will tune different strategies later
        ('scaler', StandardScaler()),
    ]
)

# Create the preprocessing pipelines for the categorical features
# There are two steps in this pipeline:
# Step 1: filling the missing values if any using the most frequent value
# Step 2: one hot encoding

cat_pipeline = Pipeline(
    steps=[
        ('cat_imputer', SimpleImputer(strategy="most_frequent")),
        ('onehot', OneHotEncoder(handle_unknown='ignore')),
    ]
)

# Assign features to the pipelines and Combine two pipelines to form the preprocessor
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer(
    transformers=[
        ('num_pipeline', num_pipeline, num_features),
        ('cat_pipeline', cat_pipeline, cat_features),
    ]
)

```

Models

Following Models are used.

1. Decision Tree
2. Random Forest
3. SVM
4. XGBoost
5. Naive Bayes

Decision Tree

In [26]: # Specify the model to use, which is DecisionTreeClassifier

```

# Make a full pipeline by combining preprocessor and the model
from sklearn.tree import DecisionTreeClassifier

pipeline_dt = Pipeline(
    steps=[
        ('preprocessor', preprocessor),
        ('clf_dt', DecisionTreeClassifier()),
    ]
)

```

In [27]: # we show how to use GridSearch with K-fold cross validation (K=10) to fine tune the model

```

# we use the accuracy as the scoring metric with training score return_train_score=True
from sklearn.model_selection import GridSearchCV

# set up the values of hyperparameters you want to evaluate
# here you must use the step names as the prefix followed by two under_scores to specify the parameter names and the "Full path" of the steps

# we are trying 2 different imputer strategies
# 2x5 different decision tree models with different parameters
# In total we are trying 2x2x5 = 20 different combinations

param_grid_dt = [
    {
        'preprocessor_num_pipeline_num_imputer_strategy': ['mean', 'median'],
        'clf_dt_criterion': ['gini', 'entropy'],
        'clf_dt_max_depth': [3, 4, 5, 6, 7],
    }
]

# set up the grid search
grid_search_dt = GridSearchCV(pipeline_dt, param_grid_dt, cv=10, scoring='accuracy')

```

```
# train
```

```
grid_search_dt.fit(
    x_train, y_train, x_val, y_val)
```

- ```
GridSearchCV(cv=
es

r',

er()),

ler()]]},
```

```
Pipeline(steps=[('cat_input',
 SimpleImputer(
 'onehot',
 oneHotEncoder=
 {'housing',
 'contact',
 'month',
 'outcome'})),
 ('clf_dt', DecisionTreeClassifier()),
 ('clf_dt_max_depth',
 {'clf_dt_max_depth': (3, 4, 5, 6, 7)},
 'preprocessor_num_pipeline_num_imputer_strategy': ['mean',
 'median'])),
 scoring='accuracy')]
```

```
In [29]: # check the best performing parameter combination
 grid_search_dt_best_params

Out[29]: {'clf_dt_criterion': 'gini',
 'clf_dt_max_depth': 7,
 'preprocessor_num_pipeline_num_imputer_strategy': 'median'}
```

```
In [30]: # build-in CV results keys
 sorted(grid_search_dt_cv_results.keys())

Out[30]: ['mean_fit_time',
 'mean_score_time',
 'mean_test_score',
 'std_fit_time',
 'std_score_time',
 'std_test_score']
```

```
'param_clf_dt_criterion',
'param_clf_dt_max_depth',
'param_preprocessor_num_pipeline_num_imputer_strategy',
'param',
'rank_test_score',
'split0_test_score',
'split1_test_score',
'split2_test_score',
'split3_test_score',
'split4_test_score',
'split5_test_score',
'split6_test_score',
'split7_test_score',
'split8_test_score',
'split9_test_score',
'std_fit_time',
'std_score_time',
'std_test_score']

In [31]: # test score for the 20 decision tree models
 grid_search_dt_cv_results['mean_test_score']

Out[31]: array([0.90054734, 0.90054734, 0.90035379, 0.90035379, 0.90022062,
```

```
In [31]: array([0.90054734, 0.90054734, 0.90035379, 0.90035379, 0.90220622,
0.90289816, 0.90101753, 0.90104518, 0.90289837, 0.90245525,
0.89939851, 0.89939851, 0.89902857, 0.89902857, 0.90121084,
0.90118329, 0.90104509, 0.90101744, 0.90140464, 0.90123404])

In [32]: # best test score
print('best dt score is: ', grid_search_dt.best_score_)

best dt score is: 0.902455206477523

In [33]: # select the best model
the best parameters are shown, note SimpleImputer() implies that mean strategy is used
clf_best = grid_search_dt.best_estimator_
clf_best

Out[33]: Pipeline(steps=[('preprocessor',
 ColumnTransformer(transformers=[('num_pipeline',
 Pipeline(steps=[('num_imputer',
 SimpleImputer(strategy='median')),
 'scaler'),
 StandardScaler()]),
 ['age', 'balance', 'day',
 'duration', 'pdays']),
)])
```

```

 'onehot',
 OneHotEncoder(handle_unknown='ignore')))),
 {'housing', 'contact',
 'month', 'outcome'}})),
 ('clf_dt', DecisionTreeClassifier(max_depth=7)))

In [34]: # final test on the testing set
 # To predict on new data: simply calling the predict method
 # the full pipeline steps will be applied to the testing set followed by the prediction
 y_pred = clf_step.predict(X_test)
 y_pred

Out[34]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

In [35]: clf_best.named_steps

Out[35]: {'preprocessor': ColumnTransformer(transformers=(('num_pipeline',

```

[illegible]

```

Pipeline(steps=[
 SimpleImputer(strategy='most_frequent'),
 'onehot',
 OneHotEncoder(handle_unknown='ignore'))],
 {'housing', 'contact', 'month', 'poutcome'}}))

In [37]: onehot_columns = list(clf_best.named_steps['preprocessor'].named_transformers['cat_pipeline'].named_steps['onehot'].get_feature_names(input_features=cat_features))

In [38]: i = clf_best.named_steps['clf_dt'].feature_importances_
 i
Out[38]: array([0.04138105, 0.01423975, 0.02157983, 0.49123509, 0.04750922,
 0.04124727, 0.00382411, 0.00375056, 0. , 0.00991195,
 0.02209235, 0.0008162 , 0. , 0.00159478, 0.00061306,
 0.00448686, 0.02231354, 0.00542562, 0.00164968,
 0.00871744, 0.00392178, 0. , 0. , 0.25458985,
 0. ,
])

In [39]: numeric_features_list = list(num_features)
 numeric_features_list.extend(onehot_columns)

```

```
In [40]: print(numeric_features_list)

['age', 'balance', 'day', 'duration', 'pdays', 'housing_no', 'housing_yes', 'contact_cellular', 'contact_telephone', 'contact_unknown', 'month_apr', 'month_aug', 'month_dec', 'month_feb', 'month_jan', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep', 'poutcome_failure', 'poutcome_other', 'poutcome_success', 'poutcome_unknown']
```

```
In [41]: import eli5 as eli5
eli5.explain_weights(clf.best.named_steps['clf_dt'], top=50, feature_names=numeric_features_list, feature_names_getter=lambda x: x['<BETA3>'])
```

```
Out[41]:
```

| Weight | Feature          |
|--------|------------------|
| 0.4912 | duration         |
| 0.2546 | poutcome_success |
| 0.0475 | pdays            |
| 0.0414 | age              |
| 0.0412 | housing_no       |
| 0.0223 | month_mar        |
| 0.0221 | month_apr        |
| 0.0216 | day              |
| 0.0142 | balance          |
| 0.0099 | contact_unknown  |
| 0.0087 | month_oct        |
| 0.0054 | month_may        |
| 0.0045 | month_jan        |
| 0.0038 | housing_yes      |

```
0.0038 housing_yes
0.0038 contact_cellular
0.0030 month_sep
0.0016 month_nov
0.0016 month_feb
0.0008 month_aug
0.0006 month_jan
0 outcome_unknown
0 contact_telephone
0 month_jul
0 outcome_failure
0 outcome_other
0 month_dec

duration <= 1.015 (89.0%)
 outcome_success <= 0.500 (86.1%)
 duration <= -0.205 (54.9%)
 month_mar <= 0.500 (54.4%)
 month_oct <= 0.500 (53.7%)
 age <= 2.598 (53.3%)
 month_apr <= 0.500 (50.4%) -->> 0.017
 month_apr > 0.500 (2.9%) -->> 0.081
 age > 2.598 (0.4%)
 month_feb <= 0.500 (0.3%) -->> 0.183
```

```

month_feb <= 0.500 (0.3%) --> 0.183
month_feb > 0.500 (0.1%) --> 0.500
month_oct > 0.500 (0.7%)
duration <= -0.629 (0.2%)
duration <= -0.695 (0.2%) --> 0.000
duration > -0.695 (0.1%) --> 0.061
duration > -0.629 (0.5%)
day <= 0.562 (0.3%) --> 0.167
day > 0.562 (0.2%) --> 0.573
month_mar > 0.500 (0.6%)
duration <= -0.590 (0.2%)
duration <= -0.693 (0.1%)
balance <= -0.276 (0.0%) --> 0.111
balance > -0.276 (0.1%) --> 0.000
duration > -0.693 (0.1%)
day <= -0.218 (0.1%) --> 0.346
day > -0.218 (0.0%) --> 0.056
duration > -0.590 (0.3%)
duration <= -0.328 (0.2%)
age <= -0.696 (0.1%) --> 0.188

```

```

age > -0.696 (0.14) ---> 0.519
duration > -0.328 (0.14)
day <= -0.278 (0.04) ---> 0.875
day > -0.278 (0.14) ---> 0.421
duration > -0.205 (31.24)
housing_no <= 0.500 (17.94)
pdays <= 3.338 (17.84)
month_mar <= 0.500 (17.74)
age <= 2.033 (17.74) ---> 0.048
age > 2.033 (0.04) ---> 0.818
month_mar > 0.500 (0.14)
day <= 1.462 (0.04) ---> 0.889
day > 1.462 (0.04) ---> 0.250
pdays > 3.338 (0.14)
duration <= 0.025 (0.04)
balance <= -0.124 (0.04) ---> 0.222
balance > -0.124 (0.04) ---> 1.000
duration > 0.025 (0.14)
duration <= 0.878 (0.14) ---> 0.955
duration > 0.878 (0.04) ---> 0.500

```

```

duration > 0.678 (0.04) ----> 0.500
housing_no > 0.500 (13.3%)
age <= 1.845 (12.4%)
month_apr <= 0.500 (11.9%)
pdays <= -0.201 (10.5%) ----> 0.127
pdays > -0.201 (1.4%) ----> 0.369
month_apr > 0.500 (0.61)
pdays <= 1.536 (0.5%) ----> 0.568
pdays > 1.536 (0.14) ----> 0.172
age > 1.845 (0.8%)
pdays <= 0.533 (0.7%)
balance <= -0.329 (0.2%) ----> 0.383
balance > -0.329 (0.5%) ----> 0.574
pdays > 0.533 (0.14)
duration <= 0.498 (0.14) ----> 0.227
duration > 0.498 (0.04) ----> 0.667
putoutcome_success > 0.500 (2.94)
duration <= -0.486 (0.5%)
duration <= -0.679 (0.2%)
balance <= 4.353 (0.2%)

```

```

data <- data.frame(
 pdays <- 3.348 (0.1%) ---> 0.021
 month_feb <- 0.500 (0.1%) ---> 0.021
 month_feb > 0.500 (0.0%) ---> 0.333
 pdays > 3.348 (0.0%)
 pdays <- 3.787 (0.0%) ---> 0.667
 pdays > 3.787 (0.0%) ---> 0.000
 balance > 4.353 (0.0%) ---> 1.000
duration <- -0.679 (0.4%)
month_sep <- 0.500 (0.4%)
pdays <- 0.622 (0.2%)
pdays <- 0.448 (0.1%) ---> 0.158
pdays > 0.448 (0.1%) ---> 0.463
pdays > 0.622 (0.2%)
balance <- -0.364 (0.0%) ---> 0.462
balance > -0.364 (0.2%) ---> 0.073
month_sep > 0.500 (0.0%)
pdays <- 1.127 (0.0%)
age <- 0.602 (0.0%) ---> 0.000
age > 0.602 (0.0%) ---> 1.000
notvis_x > 127.0 (0.0%) ---> 0.000

```

[illegible][illegible]

```

day <- -1.118 (0.0%) ---> 1.000
day > -1.118 (0.0%) ---> 0.500
duration > 1.447 (2.3%)
day <- -0.038 (1.1%)
balance <- 0.317 (0.9%) ---> 0.476
balance > 0.317 (0.3%) ---> 0.648
day > -0.038 (1.2%)
housing_no <- 0.500 (0.6%) ---> 0.322
housing_no > 0.500 (0.6%) ---> 0.486
contact_unknown > 0.500 (1.9%)
duration <= 1.505 (1.0%)
age <- 0.434 (0.7%)
month_may <- 0.500 (0.3%) ---> 0.277
month_may > 0.500 (0.5%) ---> 0.136
age > 0.434 (0.3%)
duration <= 1.478 (0.3%) ---> 0.061
duration > 1.478 (0.0%) ---> 0.400
duration <= 1.505 (0.9%)
age <- -0.696 (0.3%)
duration <= 2.149 (0.3%) ---> 0.426

```

```

duration <= 2.149 (0.3%) ---> 0.426
duration > 2.149 (0.0%) ---> 0.000
age > -0.696 (0.6%)
balance <= -0.427 (0.1%) ---> 0.098
balance > -0.427 (0.5%) ---> 0.280
poutcome_success > 0.500 (0.3%)
housing_no <= 0.500 (0.1%)
duration <= 1.960 (0.1%)
day <= 0.322 (0.1%)
month_apr <= 0.500 (0.1%) ---> 0.679
month_apr > 0.500 (0.0%) ---> 0.250
day > 0.322 (0.0%)
balance <= -0.435 (0.0%) ---> 0.500
balance > -0.435 (0.0%) ---> 1.000
duration > 1.960 (0.0%) ---> 0.000
housing_no > 0.500 (0.2%)
day <= 1.762 (0.2%)
day <= -1.718 (0.0%)
month_jun <= 0.500 (0.0%) ---> 1.000
month_jun > 0.500 (0.0%) ---> 0.000

```

```

 day > -1.718 (0.24)
 month_may <= 0.500 (0.1%) ----> 1.000
 month_may > 0.500 (0.0%) ----> 0.818
 day > 1.762 (0.0%) ----> 0.000
duration > 2.196 (4.0%)
 contact_cellular <= 0.500 (1.3%)
 balance <= -0.179 (0.8%)
 day <= 1.642 (0.7%)
 month_jun <= 0.500 (0.5%) ----> 0.269
 duration <= 2.881 (0.2%) ----> 0.441
 duration > 2.881 (0.3%) ----> 0.441
 month_jun > 0.500 (0.2%)
 balance <= -0.456 (0.0%) ----> 0.909
 balance > -0.456 (0.2%) ----> 0.485
 day > 1.642 (0.0%)
 duration <= 2.394 (0.0%) ----> 0.000
 duration > 2.394 (0.0%)
 duration <= 4.920 (0.0%) ----> 1.000
 duration > 4.920 (0.0%) ----> 0.500
 balance > -0.179 (0.5%)

```

```

balance > 0.179 (0.5%)
 balance <= 2.489 (0.5%)
 age <= -1.072 (0.0%)
 duration <= 3.840 (0.0%) ---> 0.111
 duration > 3.840 (0.0%) ----> 0.750
 age > -1.072 (0.5%)
 month_jan <= 0.500 (0.5%) ---> 0.645
 month_jan > 0.500 (0.0%) ---> 0.000
 balance > 2.489 (0.0%)
 duration <= 3.840 (0.0%) ---> 0.000
 duration > 3.840 (0.0%)
 age <= 1.280 (0.0%) ---> 1.000
 age > 1.280 (0.0%) ---> 0.000
contact_cellular > 0.500 (2.7%)
 age <= 1.280 (2.3%)
 outcome_success <= 0.500 (2.2%)
 pdays <= 0.837 (1.9%)
 duration <= 3.716 (1.3%) ---> 0.625
 duration > 3.716 (0.5%) ----> 0.737
 pdays > 0.837 (0.3%)

```

```

 pdays <= 1.301 (0.1%) ---> 0.238
 pdays > 1.301 (0.3%) ---> 0.589
 poutcome_success > 0.500 (0.1%)
 day <= -0.938 (0.0%)
 balance <= -0.169 (0.0%) ---> 1.000
 balance > -0.169 (0.0%) ---> 0.455
 day > -0.938 (0.1%) ---> 1.000
age > 1.280 (0.4%)
 day <= 0.802 (0.3%)
 month nov <= 0.500 (0.3%)
 pdays <= 1.446 (0.2%) ---> 0.639
 pdays > 1.446 (0.0%) ---> 0.250
 month nov > 0.500 (0.0%)
 age <= 2.410 (0.0%) ---> 0.200
 age > 2.410 (0.0%) ---> 1.000
 day > 0.802 (0.1%)
 month aug <= 0.500 (0.1%)
 balance <= -0.457 (0.0%) ---> 0.667
 balance > -0.457 (0.1%) ---> 0.158
 month aug > 0.500 (0.0%) ---> 1.000

```

```

month_aug > 0.500 (0.08) ----> 1.000

In [42]: r = pd.DataFrame(i, index=numeric_features_list, columns=['importance'])
 r
 print(r.sort_values('importance', ascending = False))

 importance
duration 0.490235
poutcome_success 0.254590
pdays 0.047509
age 0.041381
housing_no 0.041247
month_mar 0.022314
month_apr 0.022092
day 0.021580
balance 0.014240
contact_unknown 0.009912
month_oct 0.008717
month_may 0.005426
month_jun 0.004487
housing_yes 0.003824
contact_cellular 0.003751

```

```

contact_cellular 0.001751
month_sep 0.001022
month_nov 0.001650
month_feb 0.001595
month_aug 0.000816
month_jan 0.000613
month_jul 0.000000
month_dec 0.000000
poutcome_failure 0.000000
poutcome_other 0.000000
contact_telephone 0.000000
poutcome_unknown 0.000000

```

### Random Forest

```

In [43]: # try random forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

rf pipeline
pipeline_rf = Pipeline([
 ('preprocessor', preprocessor),
 ('classifier', RandomForestClassifier())
])

```

```

 ('clf_rf', RandomForestClassifier()),
])

here we are trying 2x3 different rf models
param_grid_rf = [
 {
 'clf_rf_criterion': ['gini', 'entropy'],
 'clf_rf_n_estimators': [50, 100, 150],
 }
]

set up the grid search
GridSearchCV(pipeline_rf, param_grid_rf, cv=10, scoring='accuracy')

```

In [45]: X\_train.info()

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 36169 entries, 24001 to 44229
Data columns (total 9 columns):
 # Column Non-Null Count Dtype
--- -
 0 age 36168 non-null int64
 1 balance 36168 non-null int64

```

[illegible]

```
er()),
 ('scaler',
 StandardScaler),

 le(r{})),

 {'age',
 'balance',
 'day',
 'duration',
 'pdays'}},
 ('cat_pipeline',
 Pipeline(steps=[('cat_impute

z',
 SimpleInput

er(strategy='most_frequent'))],
 ('onehot',
 OneHotEncoder)

er(handle_unknown='ignore'))}),

 {'housing',
 'contact',
 'month',
 'poutcome'}}]),

 ('clf_rf', RandomForestClassifier()),
 param_grid=[{'clf_rf_criterion': ['gini', 'entropy'],
 'clf_rf_n_estimators': [50, 100, 150]}]
```

```

 'clf_rf_n_estimators': (50, 100, 150)}],
 scoring="accuracy")

In [46]: # select the best model
the best parameters are shown, note SimpleImputer() implies that mean strategy is used
clf_best = grid_search_rf.best_estimator_
clf_best

Out[46]: Pipeline(steps=[('preprocessor',
 ColumnTransformer(transformers=[('num_pipeline',
 Pipeline(steps=[('num_imputer',
 SimpleImputer()),
 ('scaler',
 StandardScaler()))],
 ['age', 'balance', 'day',
 'duration', 'dpdays']),
 ('cat_pipeline',
 Pipeline(steps=[('cat_imputer',
 SimpleImputer(strategy="most_frequ
ent"))],
 ('onehot',
 OneHotEncoder(handle_unknown='igno
re')))]),
 ...])

```

```

 'housing', 'contact',
 'month', 'putoutme'())))),
 ('clf_rf', RandomForestClassifier()))

In [47]: clf_best.named_steps

Out[47]: {'preprocessor': ColumnTransformer(transformers=[('num_pipeline',
 Pipeline(steps=[('num_imputer',
 SimpleImputer()),
 ('scaler', StandardScaler())]),
 ['age', 'balance', 'day', 'duration',
 'pdays']),
 ('cat_pipeline',
 Pipeline(steps=[('cat_imputer',
 SimpleImputer(strategy='most_frequent')),
 ('onehot',
 OneHotEncoder(handle_unknown='ignore'))]),
 ['housing', 'contact', 'month', 'putoutme'])]),
 'clf_rf': RandomForestClassifier()}

In [48]: clf_best.named_steps['preprocessor']

Out[48]: ColumnTransformer(transformers=[('num_pipeline',

```

```
Out[48]: ColumnTransformer(transformers=[('num_pipeline',
 Pipeline(steps=[('num_imputer',
 SimpleImputer()),
 ('scaler', StandardScaler())]),
 {'age', 'balance', 'day', 'duration',
 'pdays'}),
 ('cat_pipeline',
 Pipeline(steps=[('cat_imputer',
 SimpleImputer(strategy='most_frequent')),
 ('onehot',
 OneHotEncoder(handle_unknown='ignore'))]),
 {'housing', 'contact', 'month', 'poutcome'})])

In [49]: i = clf_best['clf_rf'].feature_importances_
i

Out[49]: array([0.1310236 , 0.14364337, 0.11345132, 0.32614233, 0.05066708,
 0.01422428, 0.01052063, 0.0908629, 0.00456229, 0.09197191,
 0.0136944 , 0.00842556, 0.00420847, 0.00818128, 0.00500035,
 0.00817405, 0.01235324, 0.0138611 , 0.00867396, 0.00737479,
 0.0120874 , 0.00789165, 0.00725822, 0.00446163, 0.03735843,
 0.01265551])
```

```
In [50]: clf_best['preprocessor'].transformers_

Out[50]: [('num_pipeline',
Pipeline(steps=[('num_imputer', SimpleImputer()), ('scaler', StandardScaler())],
'age', 'balance', 'day', 'duration', 'pdays')),
('cat_pipeline',
Pipeline(steps=[('cat_imputer', SimpleImputer(strategy='most_frequent')),
('onehot', OneHotEncoder(handle_unknown='ignore'))],
'housing', 'contact', 'month', 'poutcome'))]

In [51]: # get columnTransformer
clf_best[0]

Out[51]: ColumnTransformer(transformers=[('num_pipeline',
Pipeline(steps=[('num_imputer',
SimpleImputer()),
('scaler', StandardScaler())],
'age', 'balance', 'day', 'duration',
'pdays')),
('cat_pipeline',
Pipeline(steps=[('cat_imputer',
SimpleImputer(strategy='most_frequent')),
('onehot',
```

```

('onehot',
 OneHotEncoder(handle_unknown='ignore'))),
 {'housing', 'contact', 'month', 'poutcome'])]]

In [52]: clf_best[0].transformers_

Out[52]: [('num_pipeline',
 Pipeline(steps=(('num_imputer', SimpleImputer()), ('scaler', StandardScaler()))),
 'age', 'balance', 'day', 'duration', 'pdays'),
 ('cat_pipeline',
 Pipeline(steps=(('cat_imputer', SimpleImputer(strategy='most_frequent')),
 ('onehot', OneHotEncoder(handle_unknown='ignore'))),
 {'housing', 'contact', 'month', 'poutcome'})]]

In [53]: num_original_feature_names = clf_best[0].transformers_[0][2]
num_original_feature_names

Out[53]: ['age', 'balance', 'day', 'duration', 'pdays']

In [54]: cat_original_feature_names = clf_best[0].transformers_[1][2]
cat_original_feature_names

Out[54]: ['housing', 'contact', 'month', 'poutcome']
```

```

In [54]: ['housing', 'contact', 'month', 'poutcome']

In [55]: cat_new_feature_names = list(clf_best[0].transformers_[1][1]['onehot'].get_feature_names(cat_original_f
feature_names))
cat_new_feature_names

Out[55]: ['housing_no',
'housing_yes',
'contact_cellular',
'contact_telephone',
'contact_unknown',
'month_apr',
'month_aug',
'month_dec',
'month_feb',
'month_jan',
'month_jul',
'month_jun',
'month_mar',
'month_may',
'month_nov',
'month_oct',
'month_sep',

```



```
[56]: feature_names = n_original_feature_names + cat_new_feature_names
 feature_names

Out[56]: ['age',
 'balance',
 'day',
 'duration',
 'pdays',
 'housing_no',
 'housing_yes',
 'contact_cellular',
 'contact_telephone',
 'contact_unknown',
 'month_apr',
 'month_aug',
 'month_dec',
 'month_feb',
 'month_jan',
 'month_jul',
 'month_jun',
 'month_mar',
 'month_may',
 'month_nov',
 'month_oct',
 'month_sep',
 'poutcome_failure',
 'poutcome_other',
 'poutcome_success',
 'poutcome_unknown']

In [57]: x = pd.DataFrame(i, index=feature_names, columns=['importance'])
 x

Out[57]:
```

|                   | importance |
|-------------------|------------|
| age               | 0.131024   |
| balance           | 0.143643   |
| day               | 0.113451   |
| duration          | 0.326142   |
| pdays             | 0.050667   |
| housing_no        | 0.011422   |
| housing_yes       | 0.010521   |
| contact_cellular  | 0.009086   |
| contact_telephone | 0.004562   |
| contact_unknown   | 0.009179   |
| month_apr         | 0.013694   |
| month_aug         | 0.008426   |
| month_dec         | 0.004208   |
| month_feb         | 0.008181   |
| month_jan         | 0.005000   |
| month_jul         | 0.008174   |
| month_jun         | 0.012353   |
| month_mar         | 0.013381   |
| month_may         | 0.008874   |
| month_nov         | 0.007375   |
| month_oct         | 0.011209   |
| month_sep         | 0.007258   |
| poutcome_failure  | 0.007258   |
| poutcome_other    | 0.004462   |
| poutcome_success  | 0.057358   |
| poutcome_unknown  | 0.012656   |

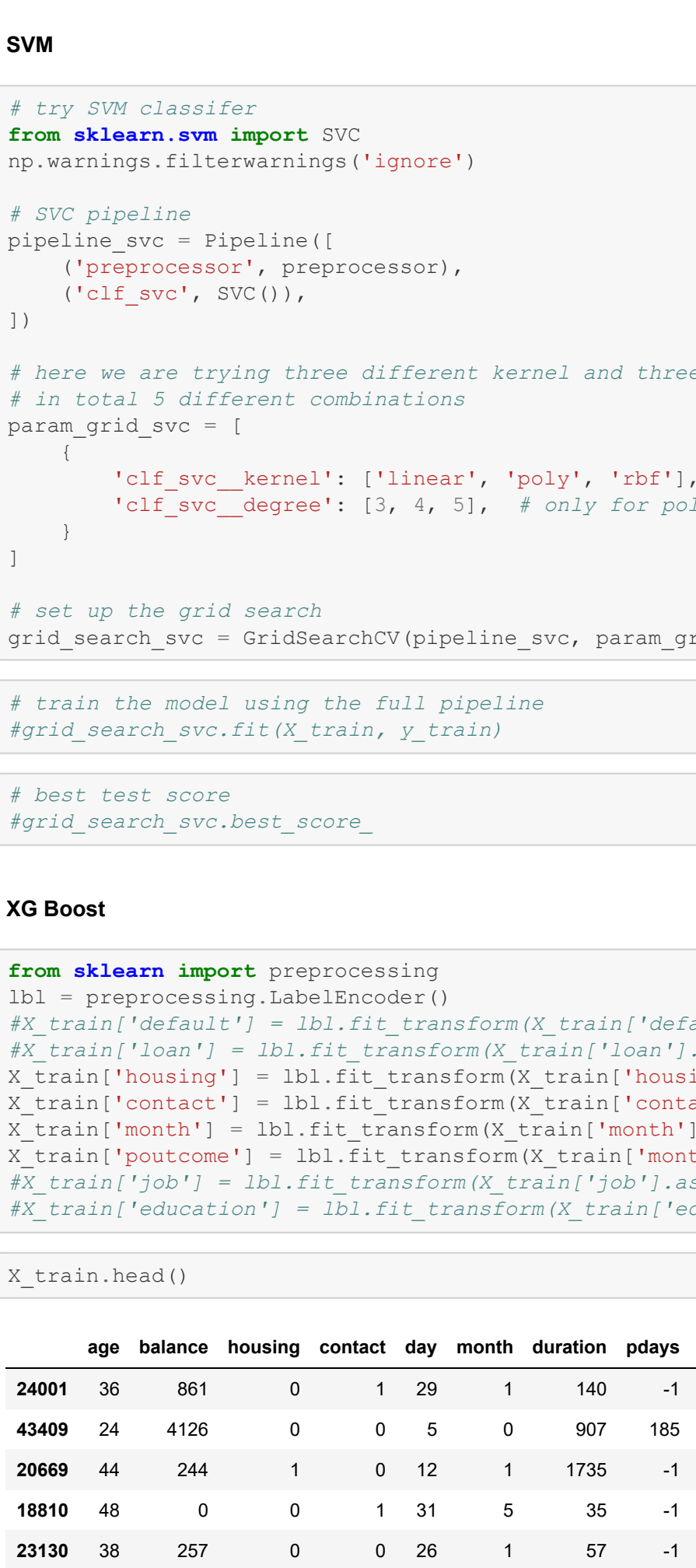
```
In [58]: x.sort_values('importance', ascending=False)

Out[58]:
```

|                   | importance |
|-------------------|------------|
| duration          | 0.326142   |
| balance           | 0.143643   |
| age               | 0.131024   |
| day               | 0.113451   |
| poutcome_success  | 0.057358   |
| pdays             | 0.050667   |
| month_apr         | 0.013694   |
| month_mar         | 0.013381   |
| poutcome_unknown  | 0.012656   |
| month_jun         | 0.012353   |
| housing_no        | 0.011422   |
| month_oct         | 0.011209   |
| housing_yes       | 0.010521   |
| contact_unknown   | 0.009179   |
| contact_cellular  | 0.009086   |
| month_may         | 0.008874   |
| month_aug         | 0.008426   |
| month_feb         | 0.008181   |
| month_jul         | 0.008174   |
| month_sep         | 0.007875   |
| month_nov         | 0.007375   |
| poutcome_failure  | 0.007258   |
| month_jan         | 0.005000   |
| contact_telephone | 0.004562   |
| poutcome_other    | 0.004462   |
| month_dec         | 0.004208   |

```
In [59]: x.sort_values('importance', ascending=False).plot.bar()

Out[59]: <AxesSubplot>
```



```
In [60]: # Save the model as a pickle file
import joblib
joblib.dump(clf_best, "clf-best.pickle")

Out[60]: ['clf-best.pickle']

In [61]: # Load the model from a pickle file
saved_tree_clf = joblib.load("clf-best.pickle")
saved_tree_clf

Out[61]: Pipeline(steps=[('preprocessor',
 ColumnTransformer(transformers=[('num_pipeline',
 Pipeline(steps=[('num_imputer',
 SimpleImputer()),
 ('scaler',
 StandardScaler())]),
 ('age', 'balance', 'day',
 'duration', 'pdays'),
 ('cat_pipeline',
 Pipeline(steps=[('cat_imputer',
 SimpleImputer(strategy='most_frequ
ent'))],
 ('onehot',
 OneHotEncoder(handle_unknown='igno
re'))]),
 ('housing', 'contact',
 'month', 'poutcome'))]),
 ('clf_rf', RandomForestClassifier()))])

In [62]: onehot_columns = list(clf_best.named_steps['preprocessor'].named_transformers_['cat_pipeline'].named_steps['onehot'].get_feature_names(input_features=cat_features))

In [63]: numeric_features_list = list(num_features)
 onehot_features_list.extend(onehot_columns)

In [64]: print(numeric_features_list)

['age', 'balance', 'day', 'duration', 'pdays', 'housing_no', 'housing_yes', 'contact_cellular', 'contact_telephone', 'contact_unknown', 'month_apr', 'month_aug', 'month_dec', 'month_feb', 'month_jan', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep', 'poutcome_failure', 'poutcome_other', 'poutcome_success', 'poutcome_unknown']

SVM

In [65]: # try SVM classifier
from sklearn.svm import SVC
np.warnings.filterwarnings('ignore')

SVC Pipeline
pipeline_svc = Pipeline([
 ('preprocessor', preprocessor),
 ('clf_svc', SVC()),
])

here we are trying three different kernel and three degree values for polynomial kernel
in total 3 different combinations
param_grid_svc = [
 {
 'clf_svc_kernel': ['linear', 'poly', 'rbf'],
 'clf_svc_degree': [3, 4, 5], # only for poly kernel
 }
]

set up the grid search
grid_search_svc = GridSearchCV(pipeline_svc, param_grid_svc, cv=10, scoring='accuracy')

In [66]: # train the model using the full pipeline
#grid_search_svc.fit(X_train, y_train)

In [67]: # best test score
#grid_search_svc.best_score_

XG Boost

In [68]: from sklearn import preprocessing
 lbl = preprocessing.LabelEncoder()
 #X_train['default'] = lbl.fit_transform(X_train['default'].astype(str))
 #X_train['loan'] = lbl.fit_transform(X_train['loan'].astype(str))
 X_train['housing'] = lbl.fit_transform(X_train['housing'].astype(str))
 X_train['contact'] = lbl.fit_transform(X_train['contact'].astype(str))
 X_train['month'] = lbl.fit_transform(X_train['month'].astype(str))
 X_train['poutcome'] = lbl.fit_transform(X_train['month'].astype(str))
 #X_train['job'] = lbl.fit_transform(X_train['job'].astype(str))
 #X_train['education'] = lbl.fit_transform(X_train['education'].astype(str))

In [69]: X_train.head()

Out[69]:
```

|       | age | balance | housing | contact | day | month | duration | pdays | poutcome |
|-------|-----|---------|---------|---------|-----|-------|----------|-------|----------|
| 24001 | 36  | 861     | 0       | 1       | 29  | 1     | 140      | -1    | 1        |
| 43409 | 24  | 4126    | 0       | 0       | 5   | 0     | 907      | 185   | 0        |
| 20669 | 44  | 244     | 1       | 0       | 12  | 1     | 1735     | -1    | 1        |
| 18810 | 48  | 0       | 0       | 1       | 31  | 5     | 35       | -1    | 7        |
| 23130 | 38  | 257     | 0       | 0       | 26  | 1     | 57       | -1    | 1        |

```
In [70]: import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import Imputer

model = xgb.XGBClassifier()

pipeline = Pipeline([
 ('standard_scaler', StandardScaler()),
 ('pca', PCA()),
 ('model', model)
])

param_grid = [
 {'pca_n_components': [5, 10, 15, 20],
 'model_max_depth': [2, 3, 5, 7, 10],
 'model_n_estimators': [10, 100, 500]}
]

grid_search_xg = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1, scoring='roc_auc')

In [71]: #fitting
grid_search_xg.fit(X_train, y_train)

mean_score = grid_search_xg.cv_results_['mean_test_score'][grid_search_xg.best_index_]
std_score = grid_search_xg.cv_results_['std_test_score'][grid_search_xg.best_index_]

grid_search_xg.best_params_, mean_score, std_score

print(f"Best parameters: {grid_search_xg.best_params}")
print(f"Mean CV score: {mean_score:.6f}")
print(f"Standard deviation of CV score: {std_score:.6f}")
print(f"Best score: {grid_search_xg.best_score:.6f}")

[10:28:25] WARNING: \nacklearner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Best parameters: {'model_max_depth': 7, 'model_n_estimators': 100, 'pca_n_components': 5}
Mean CV score: 0.894718
Standard deviation of CV score: 0.004039
Best score: 0.894718

In [72]: clf_best = grid_search_xg.best_estimator_
 clf_best

Out[72]: Pipeline(steps=[('standard_scaler', StandardScaler()),
 ('pca', PCA(n_components=5)),
 ('model',
 XGBClassifier(base_score=0.5, booster='gbtree',
 colsample_bylevel=1, colsample_bynode=1,
 colsample_bynode=1, gamma=0, gpu_id=-1,
 importance_type='gain',
 interaction_constraints='',
 learning_rate=0.300000012, max_delta_step=0,
 max_depth=7, min_child_weight=1, missing=nan,
 monotone_constraints='()', n_estimators=100,
 n_jobs=9, num_parallel_tree=1, random_state=0,
 reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
 subsample=1, tree_method='exact',
 validate_parameters=1, verbosity=None))])

In [73]: # best test score
print('best dt score is: ', grid_search_dt.best_score_)
#print('best svc score is: ', grid_search_svc.best_score_)
print('best rf score is: ', grid_search_rf.best_score_)
print('best xgboost score is: ', grid_search_xg.best_score_)

best dt score is: 0.9024552506477523
best rf score is: 0.9063535929766269
best xgboost score is: 0.8947183340539521

Best performing Model is Random Forrest with accuracy 90.63 %

Naive Bayes

In [74]: NBdf = df
 NBdf.Class.replace((1, 2), ('no', 'yes'), inplace=True)
 NBdf.head()

Out[74]:
```

|   | age | job          | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign | pdays | previous | p |
|---|-----|--------------|---------|-----------|---------|---------|---------|------|---------|-----|-------|----------|----------|-------|----------|---|
| 0 | 58  | management   | married | tertiary  | no      | 2143    | yes     | no   | unknown | 5   | may   | 261      | 1        | -1    | 0        | 0 |
| 1 | 44  | technician   | married | secondary | no      | 29      | yes     | no   | unknown | 5   | may   | 151      | 1        | -1    | -1       | 0 |
| 2 | 33  | entrepreneur | married | secondary | no      | 2       | yes     | yes  | unknown | 5   | may   | 76       | 1        | -1    | -1       | 0 |
| 3 | 47  | blue-collar  | married | unknown   | no      | 1506    | yes     | yes  | unknown | 5   | may   | 92       | 1        | -1    | -1       | 0 |
| 4 | 33  | unknown      | single  | unknown   | no      | 1       | no      | no   | unknown | 5   | may   | 198      | 1        | -1    | -1       | 0 |

Features to focus on

- duration - numerical
- job - categorical
- marital - categorical
- contact - categorical
- poutcome - categorical

```
In [75]: NBdf = [var for var in NBdf.columns if NBdf[var].dtype=="O"]

In [76]: categorical

Out[76]:
```

|  | ['job',      |
|--|--------------|
|  | 'marital',   |
|  | 'education', |
|  | 'default',   |
|  | 'housing',   |
|  | 'loan',      |
|  | 'contact',   |
|  | 'month',     |
|  | 'poutcome',  |
|  | 'Class']     |

```
In [77]: numerical = [var for var in NBdf.columns if NBdf[var].dtype=="O"]

In [78]: numerical

Out[78]: ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']

In [79]: X = NBdf
 X = X.drop(['education', 'default', 'housing', 'loan', 'month', 'Class', 'age', 'balance', 'day', 'campaign', 'pdays', 'previous'], axis=1)
 X.columns

Out[79]: Index(['job', 'marital', 'contact', 'duration', 'poutcome'], dtype='object')

In [80]: y = NBdf['Class']

In [81]: from sklearn.model_selection import train_test_split
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

In [82]: import category_encoders as ce

oneHot = ce.OneHotEncoder(cols=X.columns)
X_train = oneHot.fit_transform(X_train)
X_test = oneHot.transform(X_test)

In [84]: from sklearn.naive_bayes import GaussianNB

bayes = GaussianNB()
#bayes.fit(X_train, y_train)

In []: y_pred = bayes.predict(X_test)

In []: from sklearn.metrics import accuracy_score
 print('Model Accuracy Score is {0:0.4f}'.format(accuracy_score(y_test, y_pred)))

In []: print('Training-set Accuracy Score is {0:0.4f}'.format(accuracy_score(y_train, bayes.predict(X_train))))

In []: y_test.value_counts()

In []:

In []:
```