Bank Marketing UCI

Classification-Imbalance-Outliers

Index

Data Description*

This data as was fetched from the open source online database repository of
Donald Bren School of Information and Computer Sciences (ICS) which is a part of University of Californi a Irvine (UCI).

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.

Data Set Characteristics:	Multivariate	**Number of Instances:**	45211	**Area:**	Business
Attribute Characteristics:	Real	**Number of Attributes:**	17	**Date Donated**	2012-02-14
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	1335982

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

Data Set Information:
There are four datasets available of which we will be using the latest one with more features and instances:

1) bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analyzed in [Moro et al., 2014] **[Used for Training and Testing]**

The other three datasets are as follows:

2) bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.
3) bank-full.csv with all examples and 17 inputs, ordered by date (older version of this dataset with less inputs).
4) bank.csv with 10% of the examples and 17 inputs, randomly selected from 3 (older version of this dataset with less inputs).

Missing Attribute Values: There are several missing values in some categorical attributes, all coded with the "unknown" label. These missing values can be treated as a possible class label or using deletion or imputation techniques.

Citation: [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

Source Hyperlink: [https://archive.ics.uci.edu/ml/datasets/Bank+Marketing]

*copied from source

Technology Stack

We used the following technology stack for this project.The Python version used is 3.6.9 .

S No	Component	Version	Remark
1	Pandas	1.1.4	--
2	Numpy	1.18.5	--
3	Scikit learn	0.22.2.post1	Machine Learning
4	Imblearn	0.4.3	Balancing Dataset
5.	Light GBM	2.2.3	Machine Learning
6.	Matplotlib	3.2.2	Visulaization
7.	Seaborn	0.11.0	Visulaization

```
[144]  1 #Primary Modules
       2
       3 import pandas as pd
       4 import numpy as np
       5
       6 #Visualization Modules
       7
       8 import seaborn as sns
       9 import matplotlib.pyplot as plt
      10
      11 #Feature Selection
      12 from sklearn.feature_selection import GenericUnivariateSelect
      13 from sklearn.feature_selection import mutual_info_classif,f_classif
      14 from lightgbm import plot_importance
      15 from sklearn.feature_selection import RFECV
      16 from sklearn.svm import SVC
      17
      18 #Model_selection Tools
      19 from sklearn.model_selection import train_test_split
      20 from sklearn.metrics import classification_report
      21 from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
      22
      23 #Data Preprocessing
      24
      25 from sklearn.preprocessing import Normalizer
      26 from imblearn.combine import SMOTEENN,SMOTETomek
      27
      28 #Models
      29 from sklearn.tree import DecisionTreeClassifier
      30 from lightgbm import LGBMClassifier as lgb
      31
      32 %matplotlib inline
```

Importing Dataset

```
[ ]  1 # Import data csv file as a Data Frame
     2 main_data=pd.read_csv(r"/content/drive/MyDrive/Colab Data/IDS_Project/bank-additional-full.csv",delimiter=';')
```

Importing Training dataset as Pandas Dataframe named main_data.

.

```
[ ]  1 # Overview of Five top most data instances
     2
     3 main_data.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nonexistent	1.1	93.994
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nonexistent	1.1	93.994
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nonexistent	1.1	93.994
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nonexistent	1.1	93.994
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0	nonexistent	1.1	93.994

Overview of top 5 instances of main_data

```
[ ]  1 # Overview of data types of columns and number of data instances
     2
     3 main_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

Inspecting data types of features we conclude that we have 10 categorical and numerical features each

Features : Description*

Features of the dataset are as follows:

A. Attributes having Client Information

1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5 - default: has credit in default? (categorical: 'no','yes','unknown')
6 - housing: has housing loan? (categorical: 'no','yes','unknown')
7 - loan: has personal loan? (categorical: 'no','yes','unknown')
related with the last contact of the current campaign:
8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric).

B. Attributes with information about previous contacts with client

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

*copied from source

C. Attributes with Social and Economic Context

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

D. Output variable (desired target):

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

The main question we would be asking will be→

How does these 20 attributes contribute to decide our target variable?

Features : Overview

A. Overview of Categorical Features

```
[ ]  1 # Overview of Categorical Features
     2
     3 columns=['job', 'marital', 'education', 'default', 'housing',
     4          'loan', 'contact', 'month', 'day_of_week', 'poutcome']
     5
     6 for column in columns:
     7   print("Unique values for {} column are {}".format(column,main_data[column].unique()))
```

```
Unique values for job column are ['housemaid' 'services' 'admin.' 'blue-collar' 'technician' 'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']
Unique values for marital column are ['married' 'single' 'divorced' 'unknown']
Unique values for education column are ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']
Unique values for default column are ['no' 'unknown' 'yes']
Unique values for housing column are ['no' 'yes' 'unknown']
Unique values for loan column are ['no' 'yes' 'unknown']
Unique values for contact column are ['telephone' 'cellular']
Unique values for month column are ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']
Unique values for day_of_week column are ['mon' 'tue' 'wed' 'thu' 'fri']
Unique values for poutcome column are ['nonexistent' 'failure' 'success']
```

Unique labels of each categorical feature.

```
[ ]  1 # Overview of data instances present in Categorical Features
     2
     3 for column in columns:
     4   print("Value count for",column)
     5   for label in main_data[column].unique():
     6     print("Value count for {} column for label {} is {}".format(column,label,sum(main_data[column] == label)))
     7   print("\n")
```

```
Value count for job
Value count for job column for label housemaid is 1060
Value count for job column for label services is 3967
Value count for job column for label admin. is 10419
Value count for job column for label blue-collar is 9253
Value count for job column for label technician is 6739
Value count for job column for label retired is 1718
Value count for job column for label management is 2924
Value count for job column for label unemployed is 1014
Value count for job column for label self-employed is 1421
Value count for job column for label unknown is 330
Value count for job column for label entrepreneur is 1456
Value count for job column for label student is 875


Value count for marital
Value count for marital column for label married is 24921
Value count for marital column for label single is 11564
Value count for marital column for label divorced is 4611
Value count for marital column for label unknown is 80
```

```
Value count for education
Value count for education column for label basic.4y is 4176
Value count for education column for label high.school is 9512
Value count for education column for label basic.6y is 2291
Value count for education column for label basic.9y is 6045
Value count for education column for label professional.course is 5240
Value count for education column for label unknown is 1730
Value count for education column for label university.degree is 12164
Value count for education column for label illiterate is 18


Value count for default
Value count for default column for label no is 32577
Value count for default column for label unknown is 8596
Value count for default column for label yes is 3


Value count for housing
Value count for housing column for label no is 18615
Value count for housing column for label yes is 21571
Value count for housing column for label unknown is 990


Value count for loan
Value count for loan column for label no is 33938
Value count for loan column for label yes is 6248
Value count for loan column for label unknown is 990


Value count for contact
Value count for contact column for label telephone is 15041
Value count for contact column for label cellular is 26135


Value count for month
Value count for month column for label may is 13767
Value count for month column for label jun is 5318
Value count for month column for label jul is 7169
Value count for month column for label aug is 6176
Value count for month column for label oct is 717
Value count for month column for label nov is 4100
Value count for month column for label dec is 182
Value count for month column for label mar is 546
Value count for month column for label apr is 2631
Value count for month column for label sep is 570


Value count for day_of_week
Value count for day_of_week column for label mon is 8512
Value count for day_of_week column for label tue is 8086
Value count for day_of_week column for label wed is 8134
Value count for day_of_week column for label thu is 8618
Value count for day_of_week column for label fri is 7826


Value count for poutcome
Value count for poutcome column for label nonexistent is 35551
Value count for poutcome column for label failure is 4252
Value count for poutcome column for label success is 1373
```

Value count for unique labels of each categorical feature.

```
[ ]  1 #Counting Null Instances present in Categorical Features
     2
     3 for column in columns:
     4   print("Value count for",column)
     5   if 'unknown' not in main_data[column].unique():
     6       print("No Null values present in {} column".format(column))
     7       print("\n")
     8       continue
     9   for label in main_data[column].unique():
    10     if label=='unknown':
    11       print("Value count for {} column for null values is {}".format(column,sum(main_data[column] == label)))
    12   print("\n")
```

```
Value count for job
Value count for job column for null values is 330


Value count for marital
Value count for marital column for null values is 80


Value count for education
Value count for education column for null values is 1730


Value count for default
Value count for default column for null values is 8596


Value count for housing
Value count for housing column for null values is 990


Value count for loan
Value count for loan column for null values is 990


Value count for contact
No Null values present in contact column


Value count for month
No Null values present in month column


Value count for day_of_week
No Null values present in day_of_week column


Value count for poutcome
No Null values present in poutcome column
```

The null values in categorical columns are represented by label “unknown”

B. Overview of Numerical Features

```
1 # Overview of Numerical Features
2 columns=['age', 'duration', 'campaign',
3          'pdays', 'previous', 'emp.var.rate', 'cons.price.idx',
4          'cons.conf.idx', 'euribor3m', 'nr.employed']
5 main_data[columns].describe()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41176.00000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000
mean	40.02380	258.315815	2.567879	962.464810	0.173013	0.081922	93.575720	-40.502863	3.621293	5167.034870
std	10.42068	259.305321	2.770318	186.937102	0.494964	1.570883	0.578839	4.627860	1.734437	72.251364
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

Overview of numerical features using describe function.
[25% ,50% and 75% represent Q1,Q2 and Q3 quartiles]
[std is Standerd Deviation]

Preliminary Data Visualization

```
[ ]  1 columns_num=['age', 'duration', 'campaign',
     2              'pdays', 'previous', 'emp.var.rate', 'cons.price.idx',
     3              'cons.conf.idx', 'euribor3m', 'nr.employed']
     4
     5 columns_cat=['job', 'marital', 'education', 'default', 'housing',
     6              'loan', 'contact', 'month', 'day_of_week', 'poutcome']
```

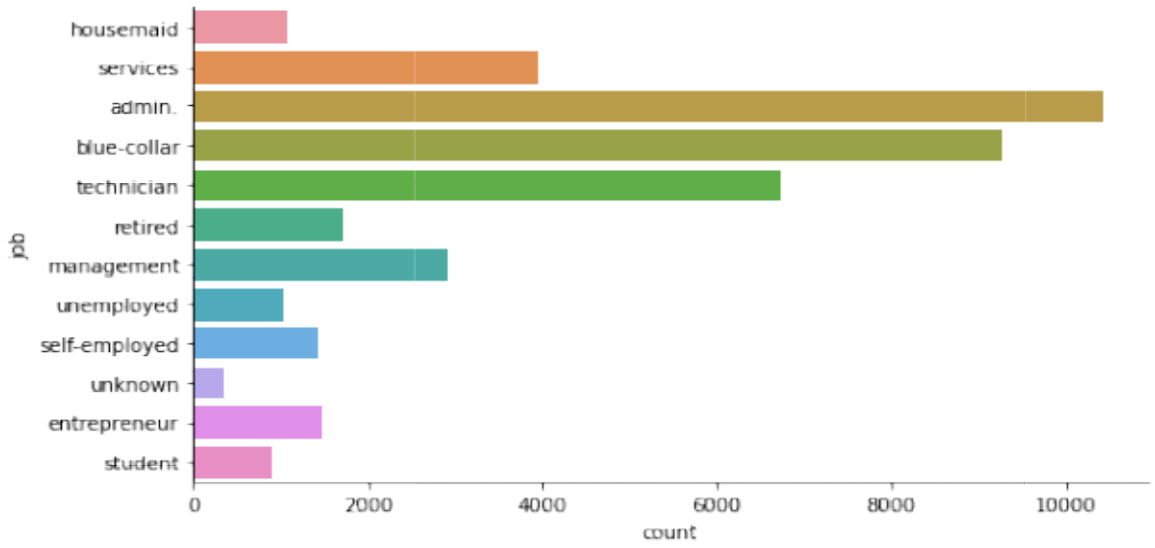
1. ***We created two lists constituting names of categorical variables and numerical variables respectively***

2. ***Then we visualized categorical features against the count of label specific instances to get a better understanding of composition of our dataset.***

```
1 for column in columns_cat:
2   sns.catplot(y=column, data=main_data,kind='count',height=4, aspect=2)
```

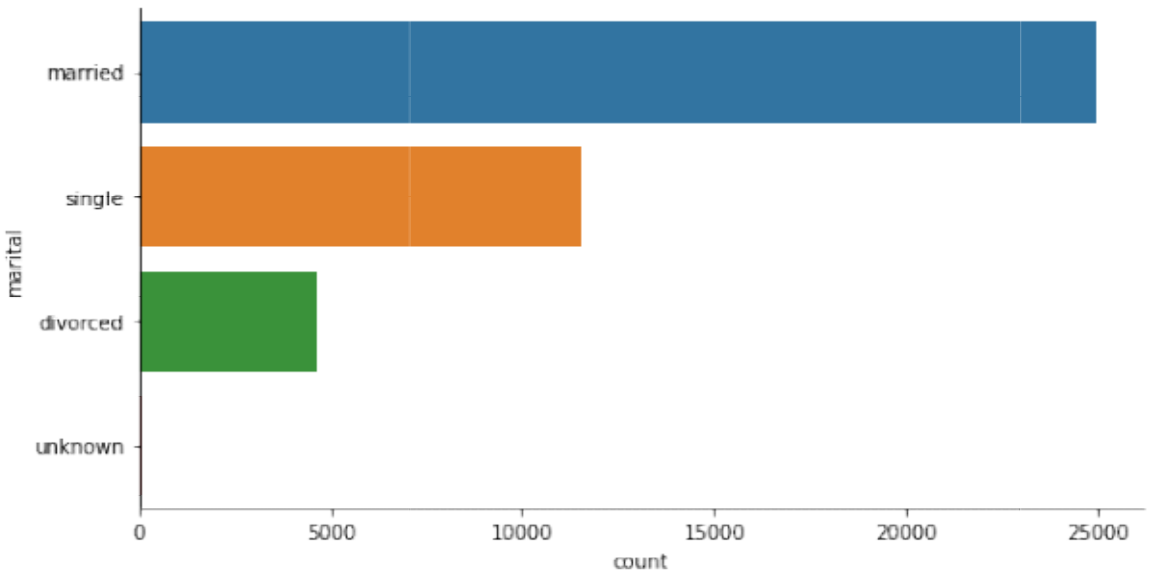
3. ***Feature wise inferences.--> The main question that our inference from these graphs answer is that how does the count of a unique label of a feature affects our overview of that attribute.***

1. **JOB**



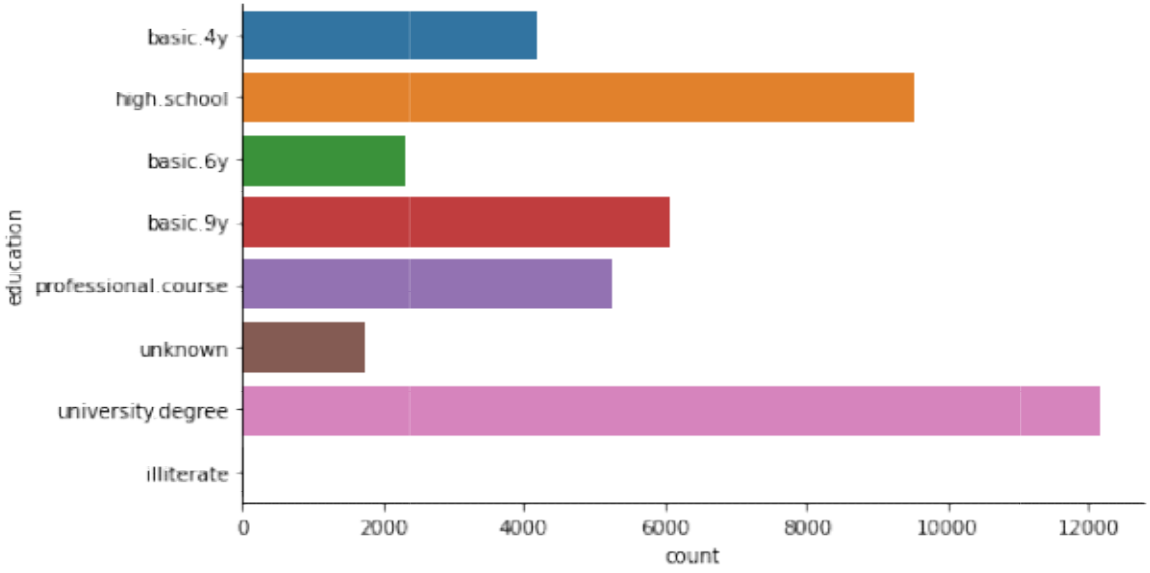
INFERENCE – Highest count in the dataset for the feature job is of label Admin followed by labels blue-collar and technician and minimum count corresponds to label student followed by label unemployed and label housemaid.

2. MARITAL STATUS



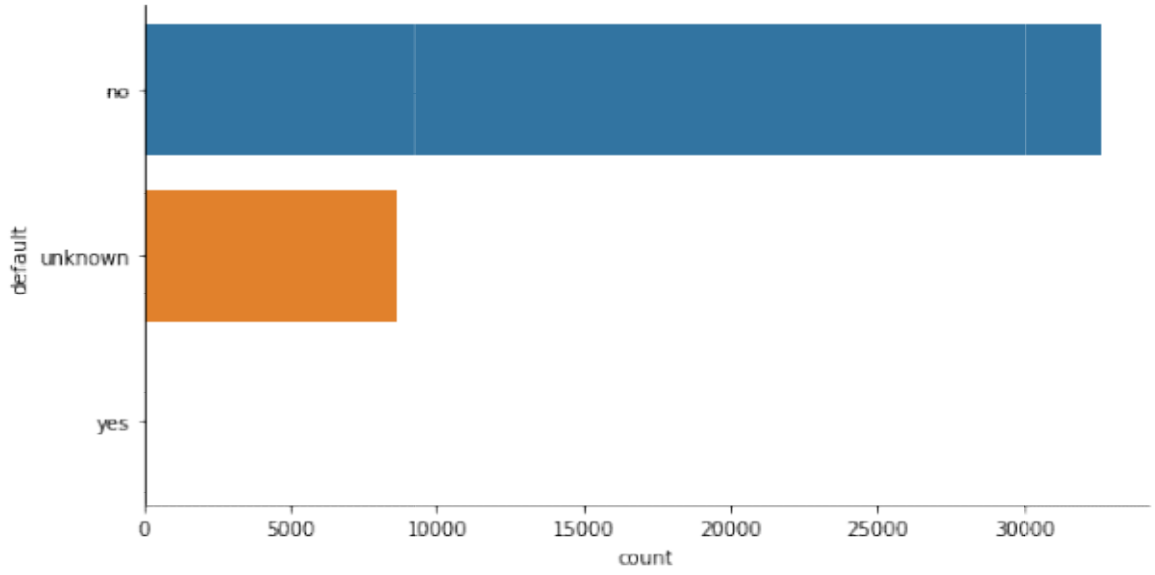
INFERENCE- Highest count in the dataset for marital feature corresponds to married people.

3. EDUCATION



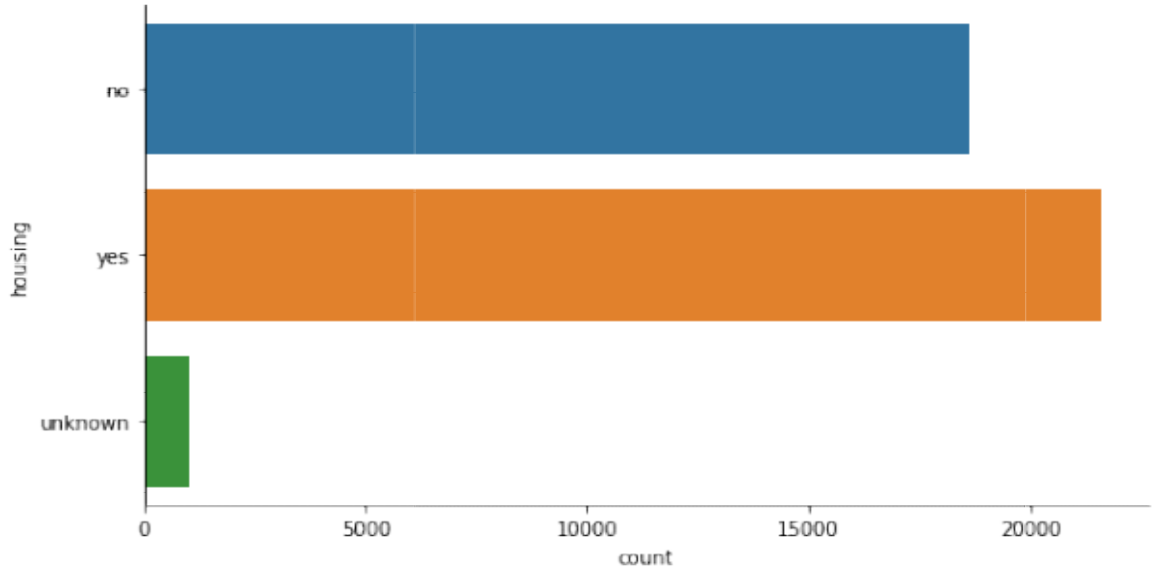
INFERENCE- The highest people in the dataset have a university degree followed closely by high school education.

4. DEFAULT



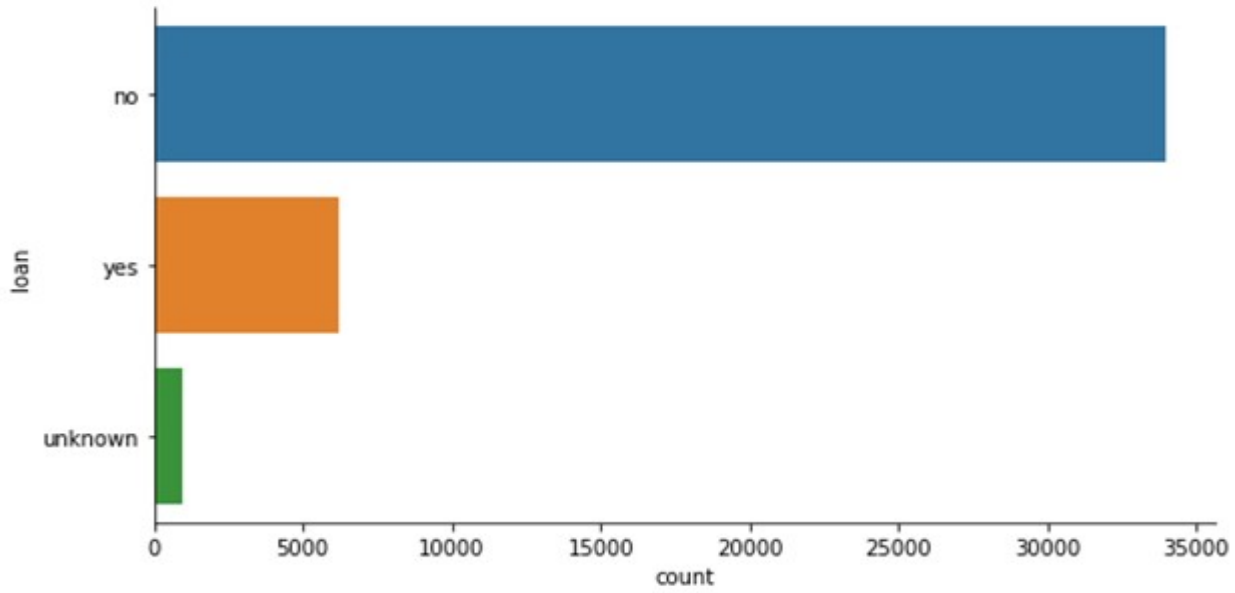
INFERENCE- Most people doesn't have their credit in default and status of many others being unknown is a significant insight.

5. HOUSING



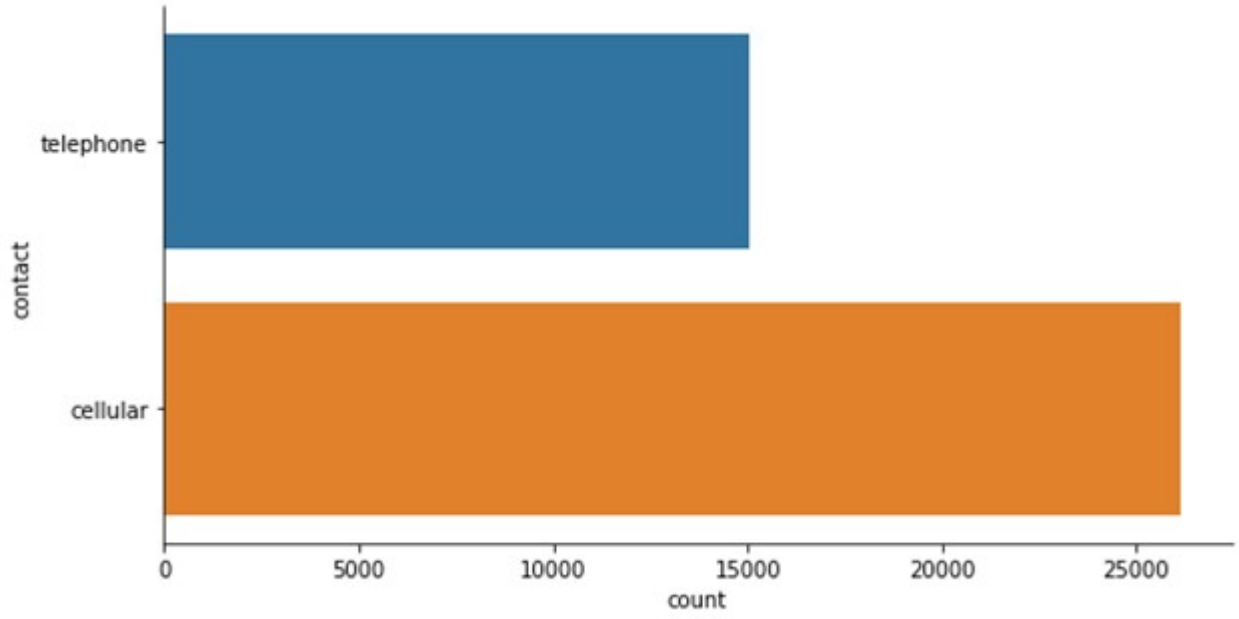
INFERENCE- Most of the people contacted already have sanctioned housing loans.

6. LOAN



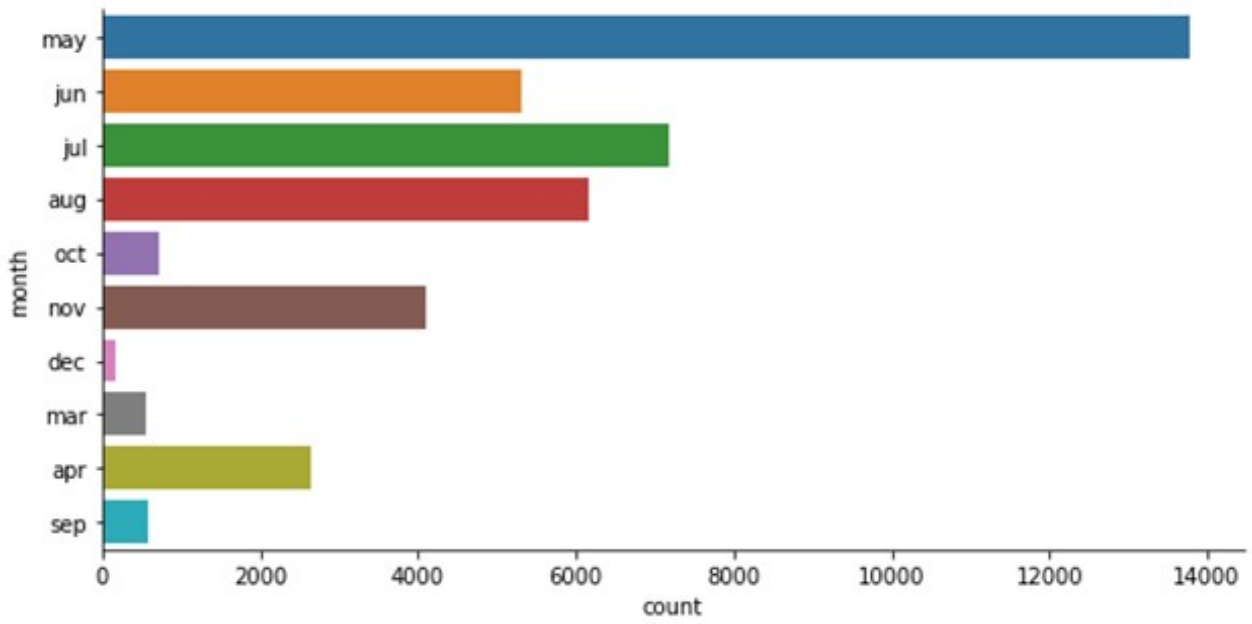
INFERENCE- Most of the people contacted don't have a previously sanctioned personal loan.

7. CONTACT



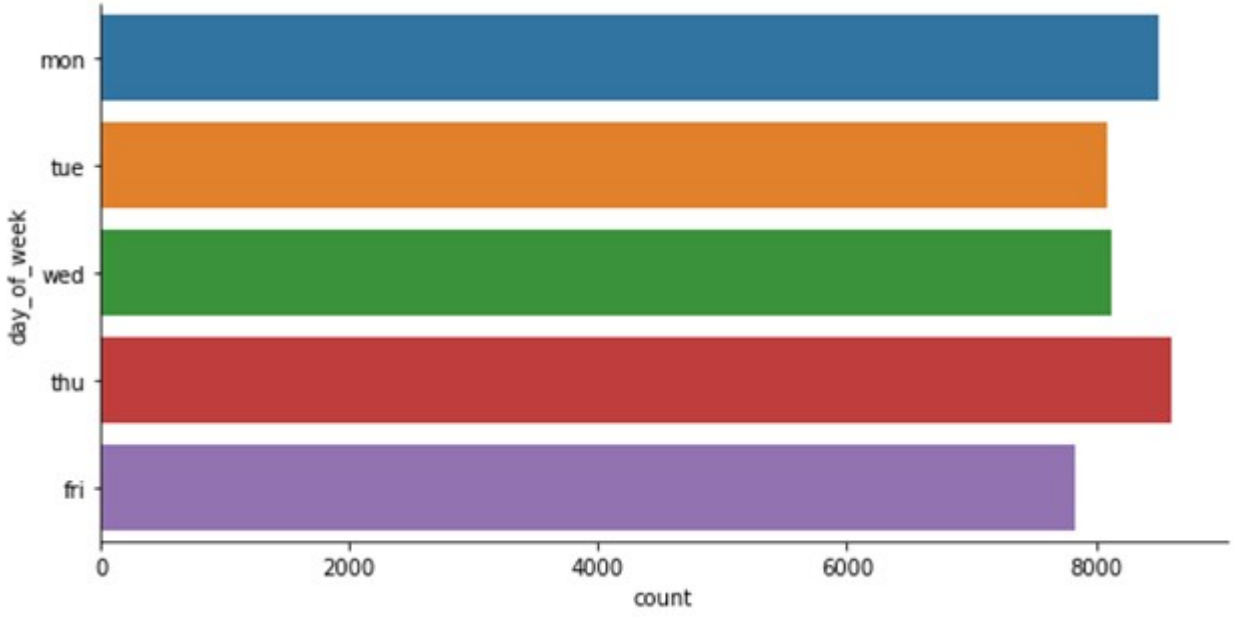
INFERENCE- Total count of people who were contacted on cellular contacts is higher than the people contacted on telephone. This may point to the target base being preferably new and younger customers.

8. MONTH



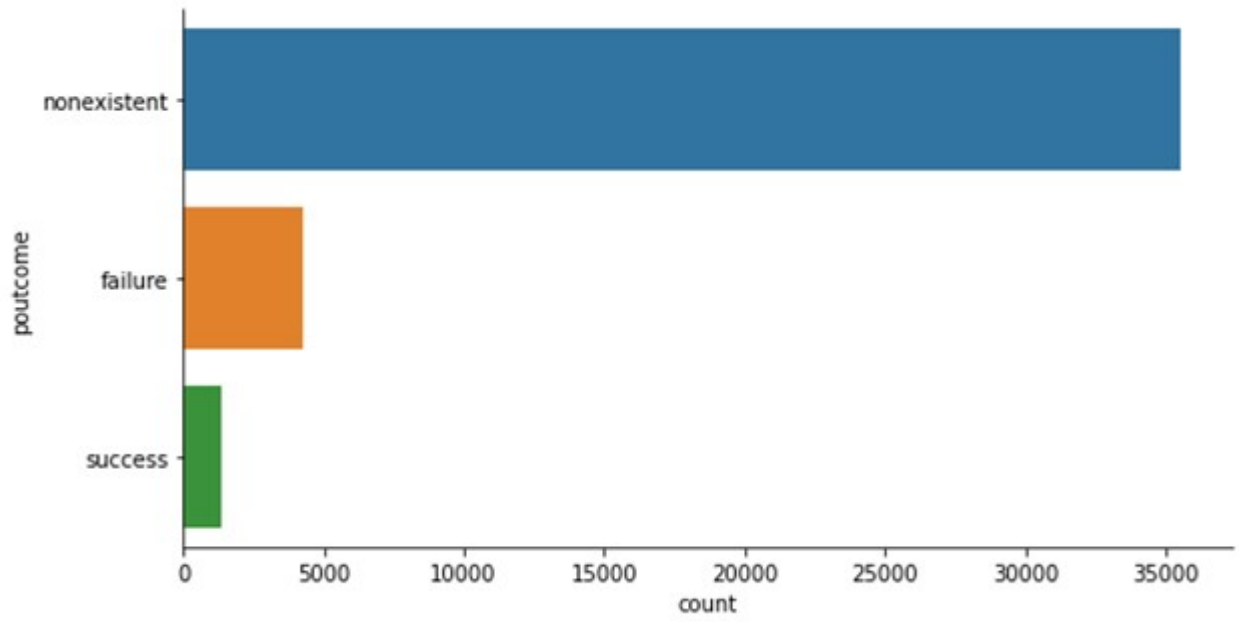
INFERENCE-The highest no. of people were contacted last in May. The least last contacted month is December probably owing to higher staff holidays in that month due to Christmas and New Year.

9. DAY OF WEEK



INFERENCE- The contact cycle is well spaced out over the five working days.

10. POUTCOME



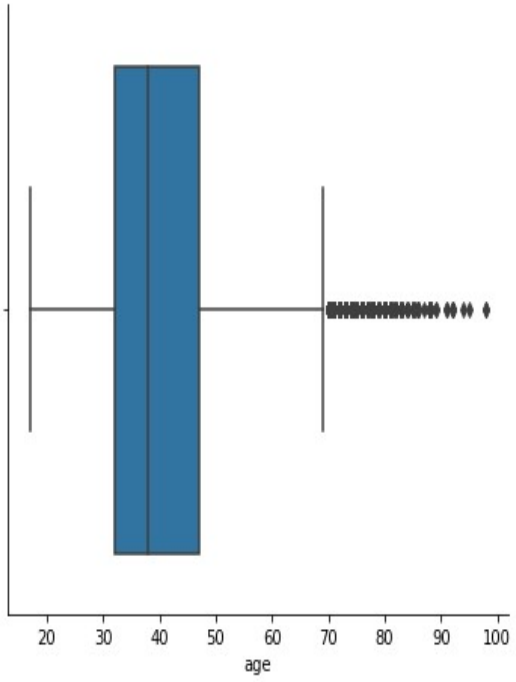
INFERENCE- Most of the customers contacted have nonexistent label for previous outcome. This points to the fact that as they were not contacted previously these are probably newest addition to the bank's customer base.

4. ***Then we visualized numerical features as box plot to gauge their composition and find out the outliers. .The main question that our inference from these graphs answer is that how does the count of a unique label of a feature affects our overview of that attribute.***

```
1 for column in columns_num:
2   sns.catplot(x=column,data=main_data,kind='box')
```

5. ***Feature wise inferences*.**

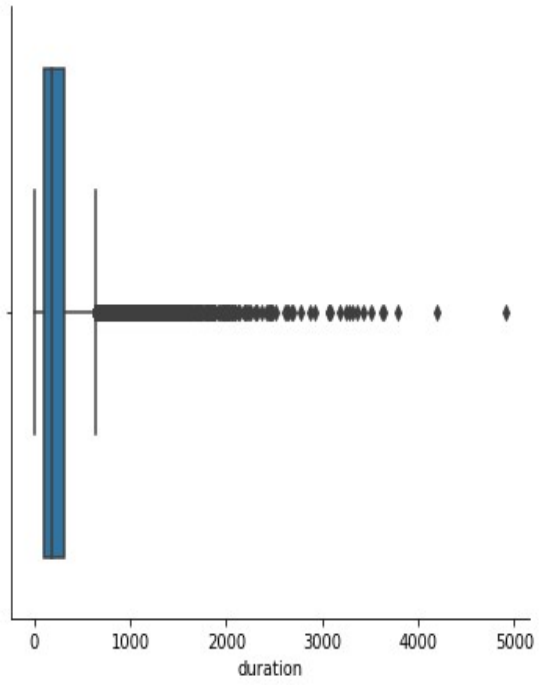
1. AGE



Count = 41176.000000
Mean = 40.023800
Standard Deviation = 10.420680
First Quartile (Q1) = 32.000000
Median = 50% = 38.000000
Third Quartile (Q3) = 47.000000
Minimum value = 17.000000
Maximum value = 98.000000

Outlier's are points that lies outside of this range [9.500000, 69.500000]

Inference: As apparent from the graph the target population was predominantly young with 3/4th being younger than 47.

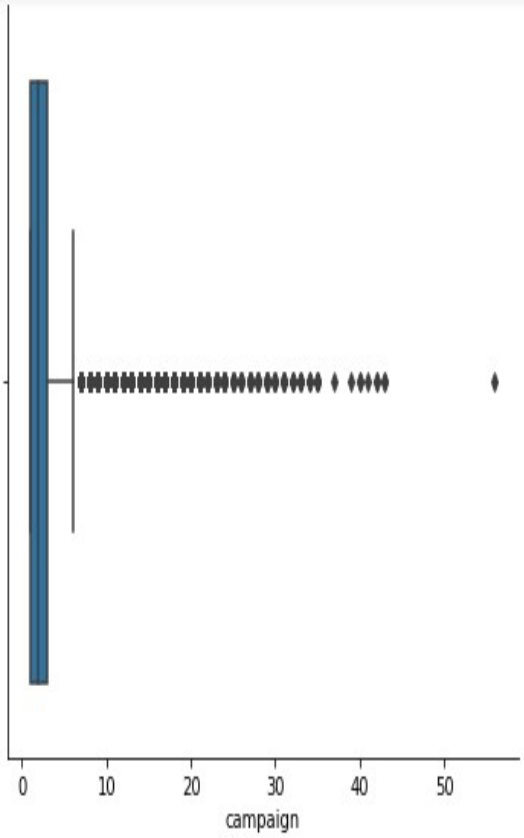


2. DURATION

Count = 41176.000000
Mean = 258.315826
Standard Deviation = 259.305328
First Quartile (Q1) = 102.000000
Median = 50% = 180.000000
Third Quartile (Q3) = 319.000000
Minimum value = 0.000000
Maximum value = 4918.000000

Outlier's are points that lies outside of this range [-223.500000, 644.500000]

Inference → Almost 50% of the contacts made finished well within 3 minutes thus are likely to ineffective. 25% of the contacts made were over 5 minutes and could have converted a client to say yes more comprehensively than other calls.

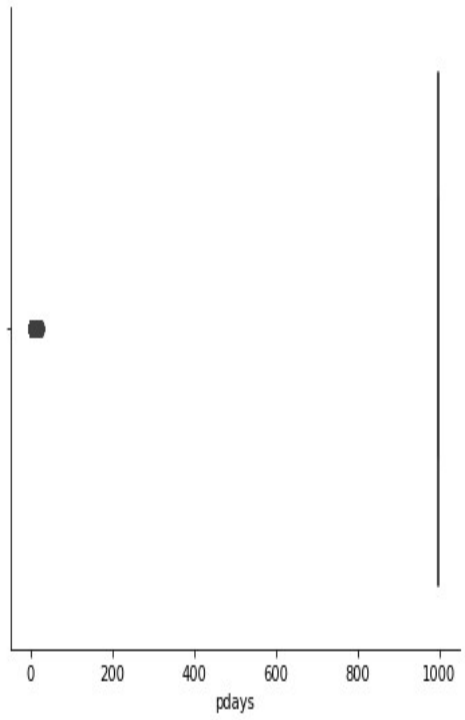


3. CAMPAIGN

Count = 41176.000000
Mean = 2.567879
Standard Deviation = 2.770318
First Quartile (Q1) = 1.000000
Median = 50% = 2.000000
Third Quartile (Q3) = 3.000000
Minimum value = 1.000000
Maximum value = 56.000000

Outlier's are points that lies outside of this range [-2.000000, 6.000000]

Inference → Most people were contacted at least 2 times regarding the term deposit subscription. Also this graph also shows a high number of outliers. These may be due to the fact that more lucrative indivisuals were contacted a higher number of times as compared to others.



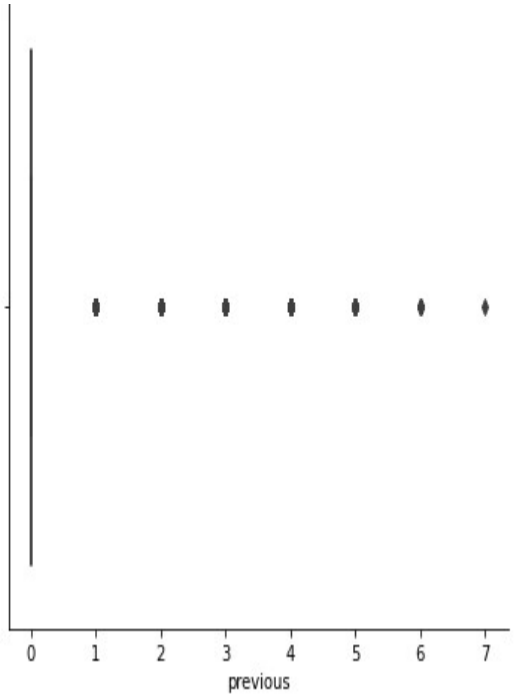
4. **PDAYS**

Count = 41176.000000
Mean = 962.464783
Standard Deviation = 186.937103
First Quartile (Q1) = 999.000000
Median = 50% = 999.000000
Third Quartile (Q3) = 999.000000
Minimum value = 0.000000
Maximum value = 999.000000

Outlier's are points that lies outside of this range [999.000000, 999.000000]

Inference → Here pdays = Number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
Almost all client have p value 999 , that means they were not previously contacted

5. PREVIOUS

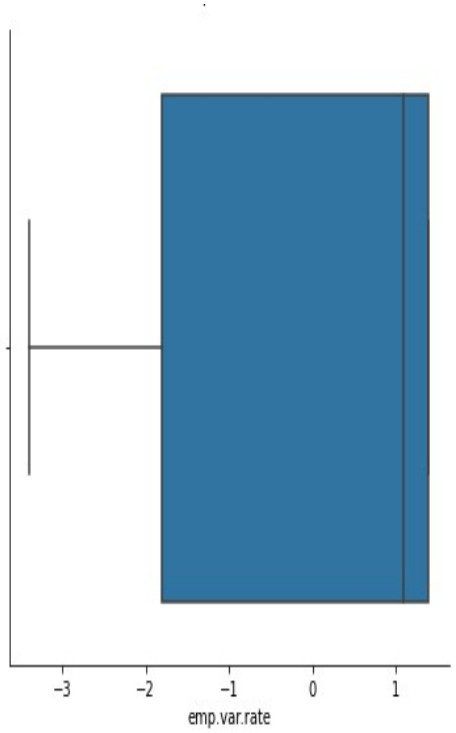


Count = 41176.000000
Mean = 0.173013
Standard Deviation = 0.494964
First Quartile (Q1) = 0.000000
Median = 50% = 0.000000
Third Quartile (Q3) = 0.000000
Minimum value = 0.000000
Maximum value = 7.000000

Outlier's are points that lies outside of this range [0.000000, 0.000000]

Inference → previous = number of contacts performed before this campaign. Almost all client have previous value 0 , that means they were not contacted before this campaign for any other campaigns.

6. **EMPLOYMENT VARIATION RATE**

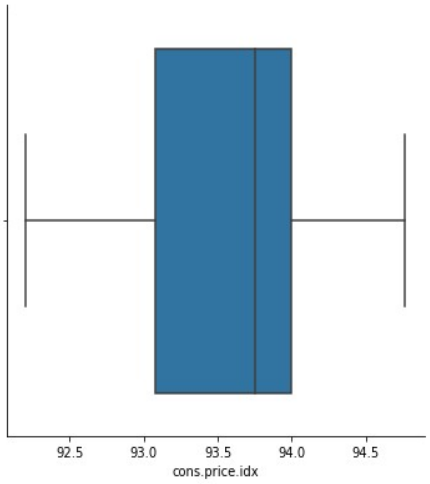


Count = 41176.000000
Mean = 0.081922
Standard Deviation = 1.570883
First Quartile (Q1) = -1.800000
Median = 50% = 1.100000
Third Quartile (Q3) = 1.400000
Minimum value = -3.400000
Maximum value = 1.400000

Outlier's are points that lies outside of this range [-6.599999, 6.200000]

Inference → People have employment variation rates (quarterly indicator) that are surprisingly negative. Mean is close to Zero. This may due to the fact that the data was collected during the period 2008-2010. This period was headlined by the great economic crisis of 2009

7. CONSUMER PRICE INDEX



Count = 41176.000000
Mean = 93.575722
Standard Deviation = 0.578839
First Quartile (Q1) = 93.074997
Median = 50% = 93.749001
Third Quartile (Q3) = 93.994003
Minimum value = 92.200996
Maximum value = 94.766998

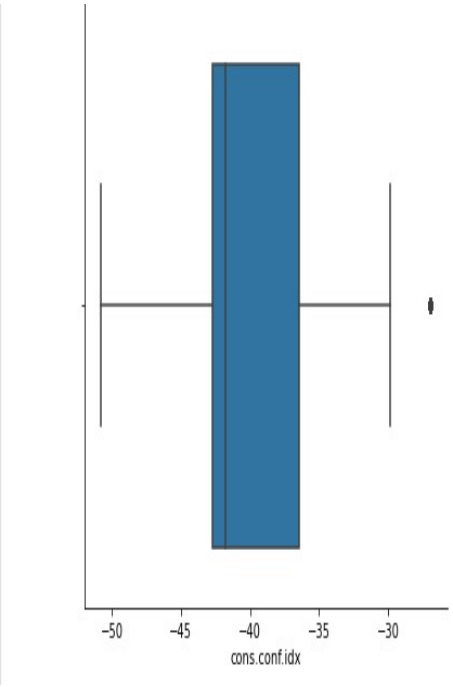
Outlier's are points that lies outside of this range 91.696487, 95.372513]

$$CPI_t = \frac{C_t}{C_0} * 100$$

CPI_t = consumer price index in current period
C_t = cost of market basket in current period
C_0 = cost of market basket in base period

Inference → Consumer price index data is very dense and below 100 which means price of goods are lower than the base year meaning demand is less. This implies that economy was in bad shape. We can infer that in such a case people are more likely to say no.

8. CONSUMER CONFIDENCE INDEX

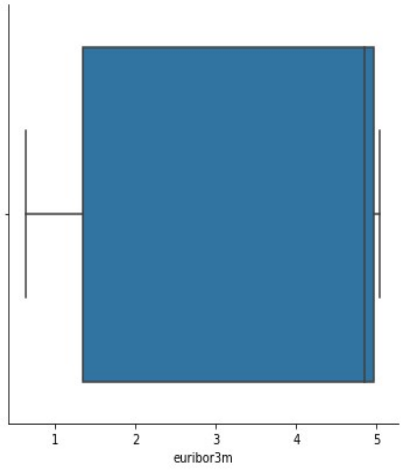


Count = 41176.000000
Mean = -40.502865
Standard Deviation = 4.627860
First Quartile (Q1) = -42.700001
Median = 50% = -41.799999
Third Quartile (Q3) = -36.400002
Minimum value = -50.799999
Maximum value = -26.900000

Outlier's are points that lies outside of this range [-52.150002, -26.950003]

Inference → CCI is overwhelmingly negative which implies a high degree of pessimism on the state of the economy. We can infer that in such a case people are more likely to say no for the term deposit policy.

9. EURIBOR 3 MONTH RATE



Count = 41176.000000
Mean = 3.621293
Standard Deviation = 1.734437
First Quartile (Q1) = 1.344000
Median = 50% = 4.857000
Third Quartile (Q3) = 4.961000
Minimum value = 0.634000
Maximum value = 5.045000

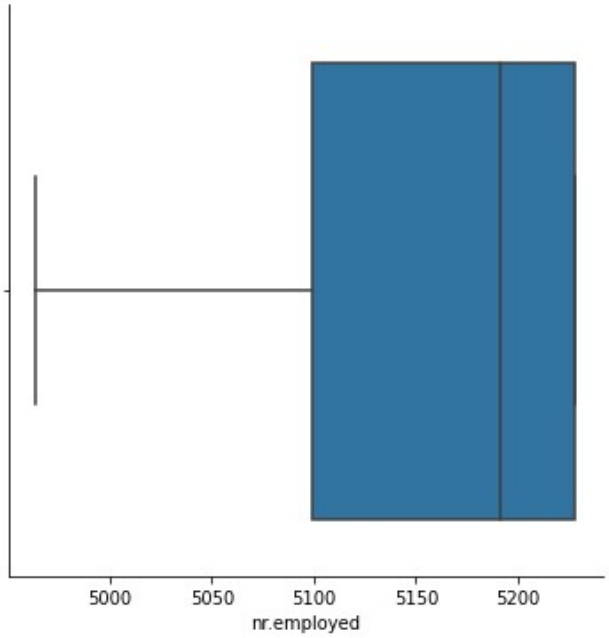
Outlier's are points that lies outside of this range [-4.081500, 10.386500]

Inference → Euribor 3 month rate is uncharacteristically high as compared to its Jan 2020 (Normal Market) rate being -0.463. This implies a strained lending market with high premium. We can further strengthen our belief that during the collection period of data, the economy was in bad shape. We can infer that according to the three economic indicators people are likely to say no for the term deposit.

Euribor

The Euro Interbank Offered Rate is a daily reference rate, published by the European Money Markets Institute, based on the averaged interest rates at which Eurozone banks offer to lend unsecured funds to other banks in the euro wholesale money market. Wikipedia

10 NUMBERS OF EMPLOYEES



Count = 41176.000000
Mean = 5167.034668
Standard Deviation = 72.251366
First Quartile (Q1) = 5099.100098
Median = 50% = 5191.000000
Third Quartile (Q3) = 5228.100098
Minimum value = 4963.600098
Maximum value = 5228.100098

Outlier's are points that lies outside of this range [4905.600098, 5421.600098]

Inference → The number of employees varies from 4963 to 5228 with a standered deviation of 72. This slight fluctuation seems normal enough for the number of employees at the bank.

Advanced Data Visualization

1. Now that we have summarized the composition of the various feature instances we dive deeper and plot feature against each other. We do this to draw out any correlation between features that may exist.

We have a total of 10 categorical features and we generated a total of 90 (2! * 10C2) graphs to study using the following code. While most of the graphs were not of any use some of them were quite insightful.

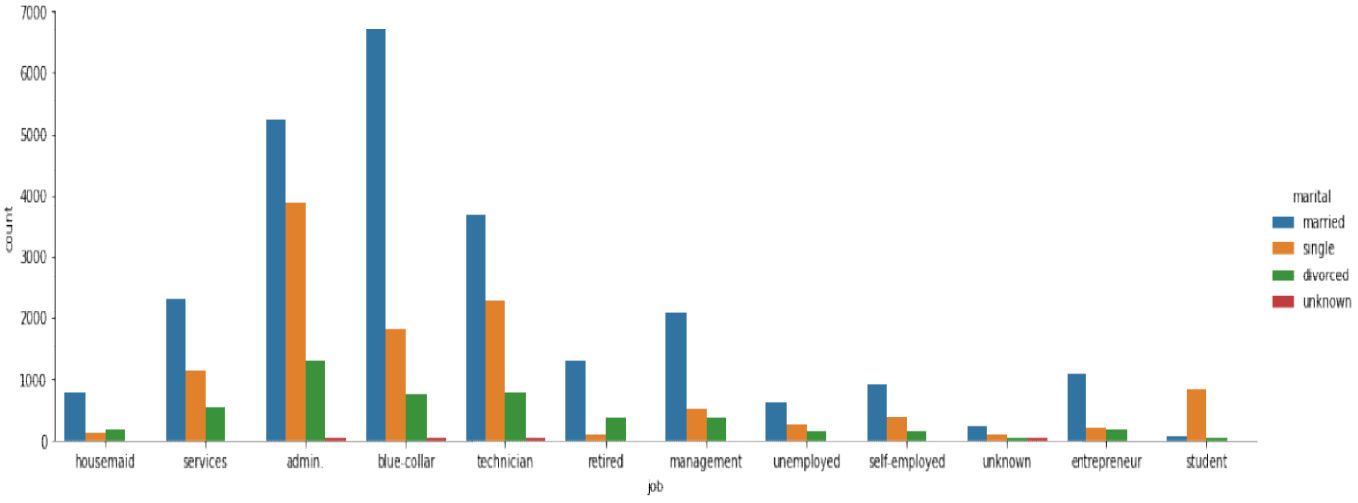
```
counter=[]

for column1 in columns_cat:
  for column2 in columns_cat:
    if column1!=column2 and [column1,column2] not in counter:
      sns.catplot(x=column1,hue=column2, data=main_data,height=4, aspect=4,kind='count')
      counter.append([column1,column2])
```

The most relevant insights drawn from these graphs is as follows:

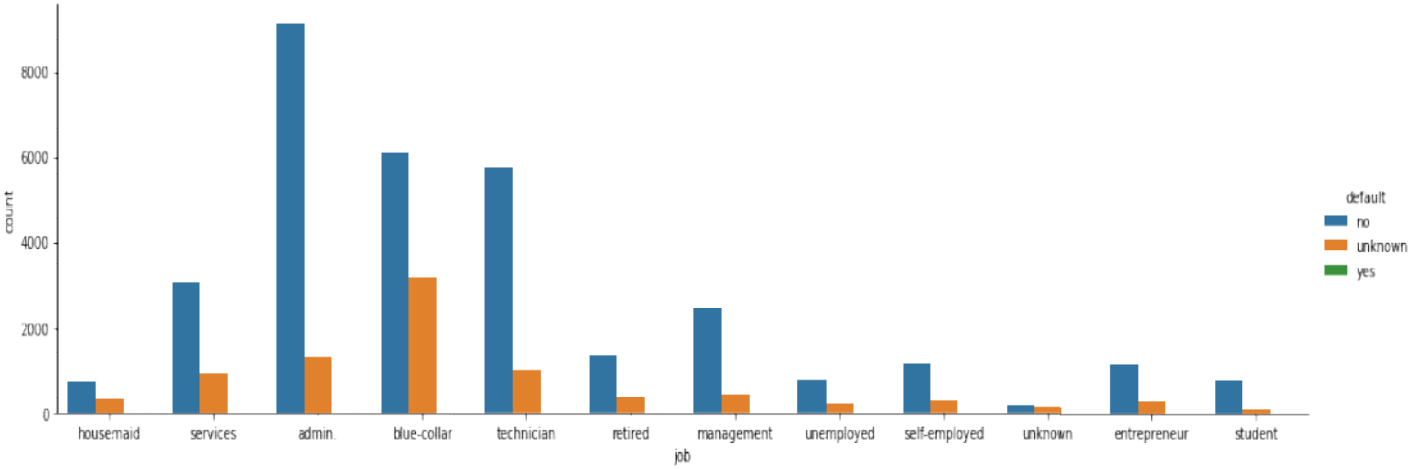
Q 1. What is the marital status of the instances with respect to their job status?

Ans. People contacted are predominantly married for all job levels except students which is along the expected lines.



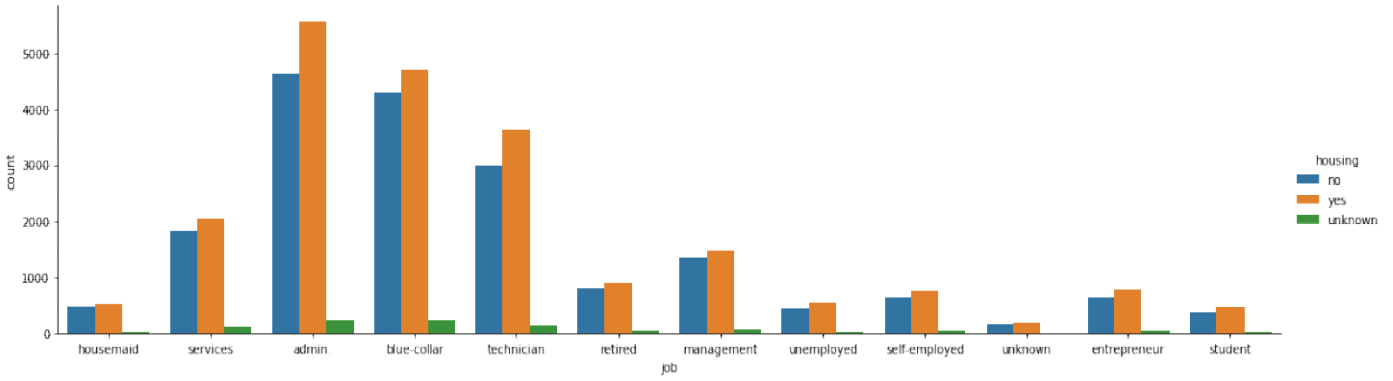
Q2. What is the credit default status of the instances with respect to their job status?

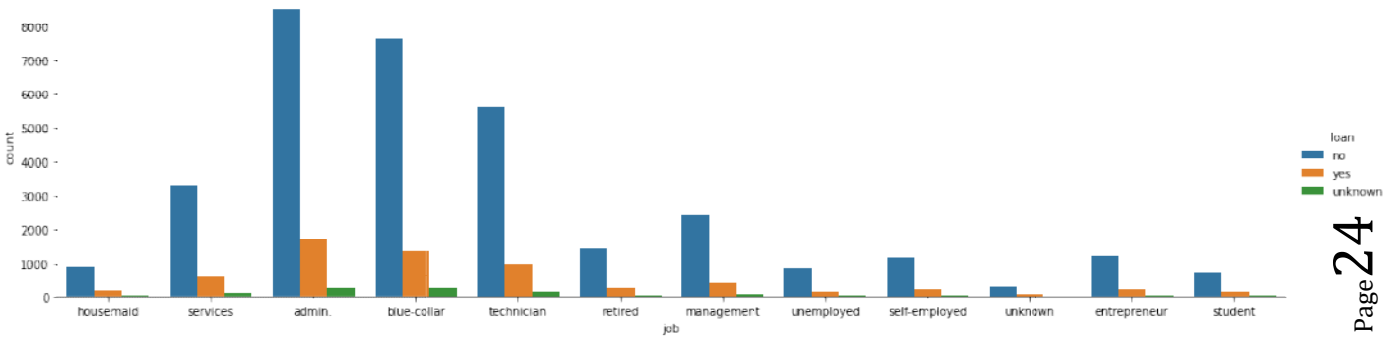
Ans. Almost all the people contacted have not defaulted on their credit.



Q2. What is the of denomination of the instances where people contacted have been sanctioned a type of loan from the bank with respect to their job status?

Ans. People contacted have been predominantly sanctioned home loan before and usually don't have a personal loan sanctioned. So its likely that the database of people with sanctioned home loans was used to campaign for term deposit maybe due to high credit credibility.

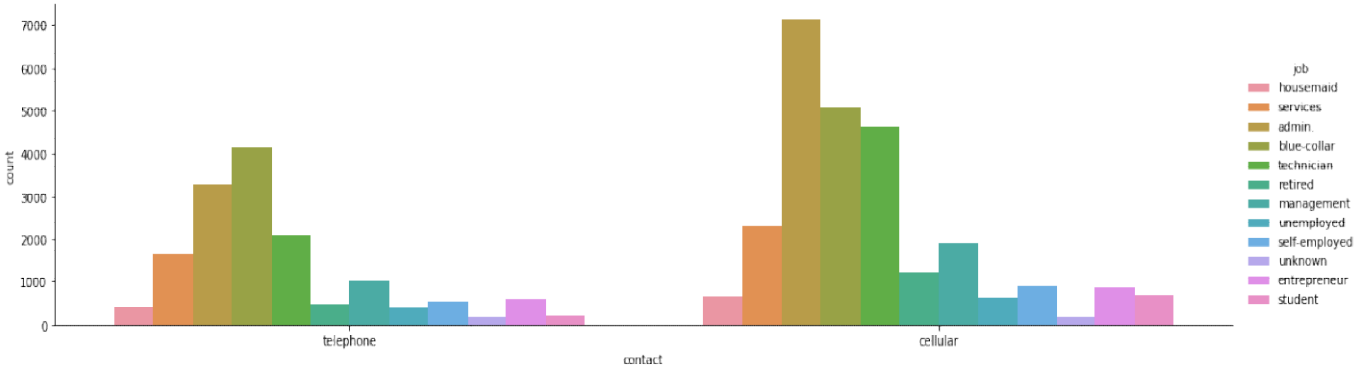


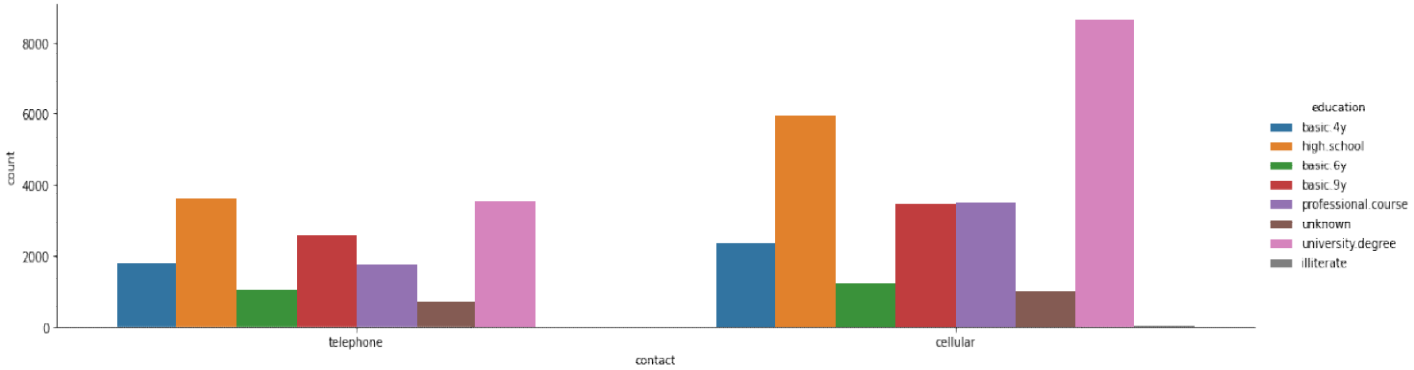


Q3. What is the composition of people who were contacted via telephone or cellular channel with respect to their job status and education?

Ans. Admin label leads in cellular contacts while blue collar leads in telephone based contacts. This may be due to different demographics of these two job labels.

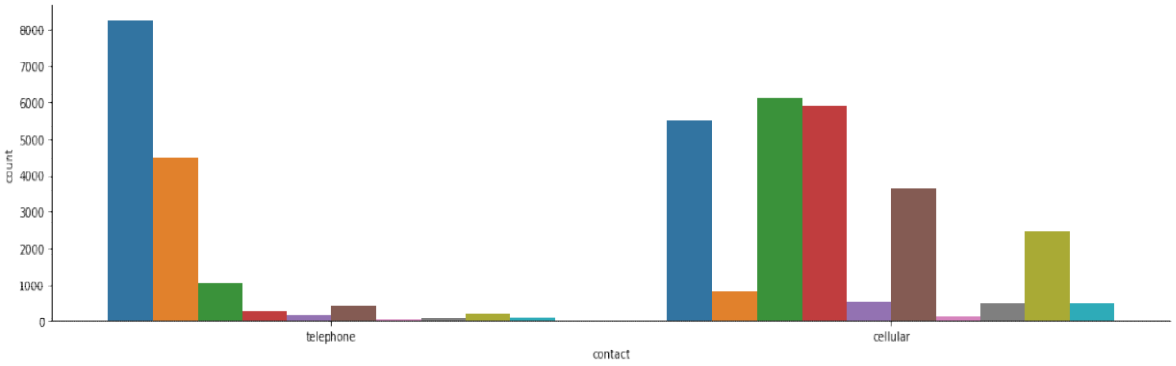
University education dominates for cellular contacts and high school education leads for telephone based contacts. This may be due to a slight conservative dominance in people having only high school education.





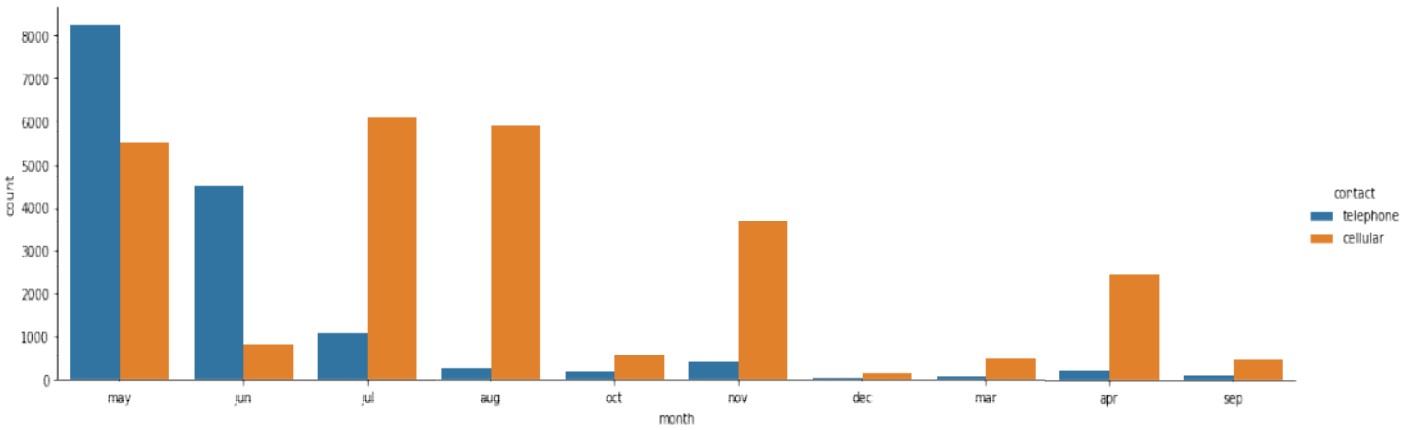
Q4. What is the month wise composition of people who were contacted via telephone or cellular channel?

Ans. Cellular contacts were made predominantly in July and August and Telephone based contacts were made mostly in month of may.



Q5. What composition of people who were contacted via telephone or cellular channel with respect to each month?

Ans. Q5 along with Q4 gives us a strong insight in campaign scheduling and strategy. The campaign for two years used months of May and June for predominantly calling out people with telephone connections and after that in subsequent months predominantly called out people with a cellular connection.

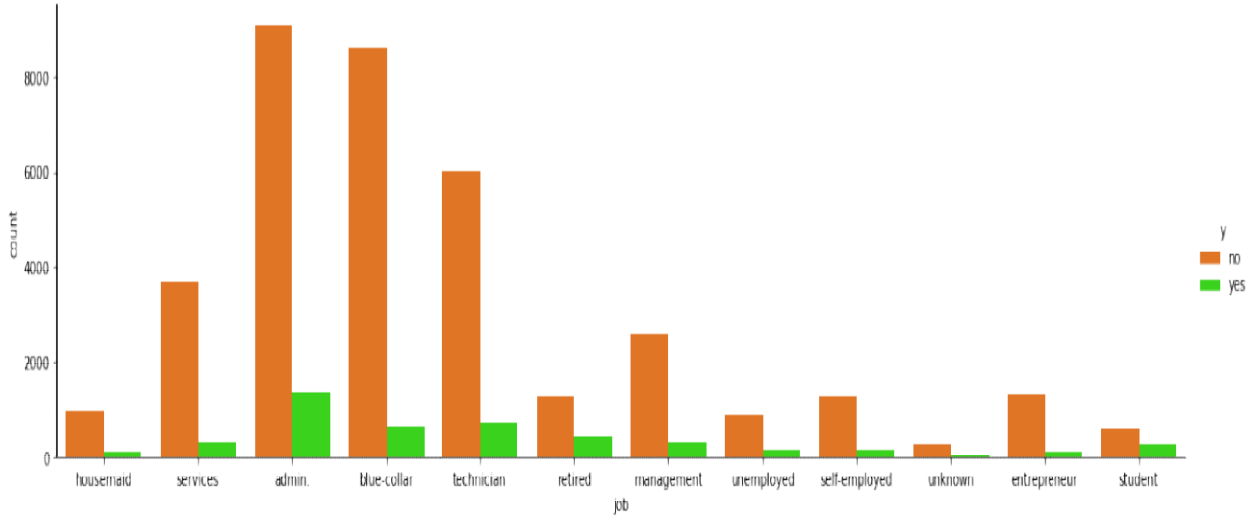


2. Next we plotted our categorical variables against the target variable y. There were a total of 10 graphs and the graphs that offered an insight follow. The code for generating the graphs is as shown below. .The main question that our inference from these graphs answer is that how does a unique categorical feature is affecting the outcome of our target variable?

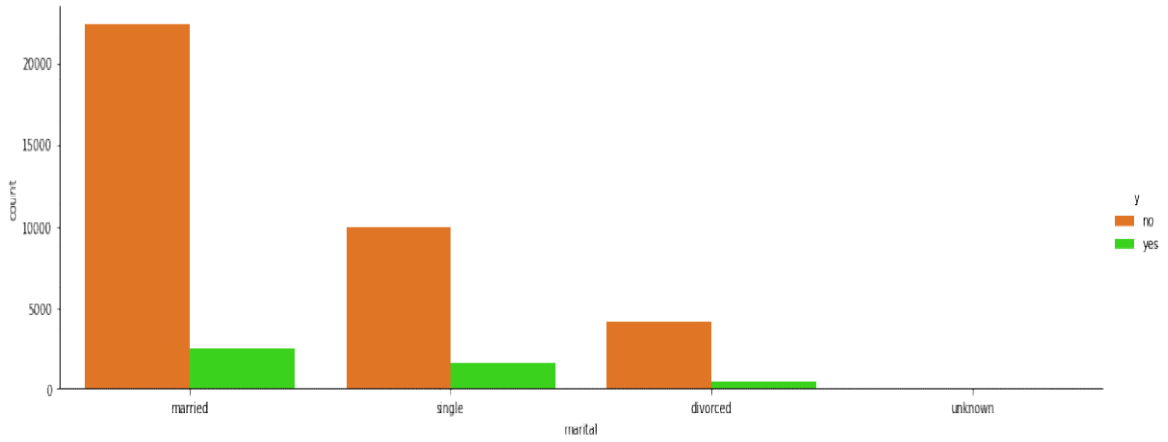
```
[ ]  1 for column in columns_cat:
     2   sns.catplot(x=column,hue='y',data=main_data,kind='count',palette="gist_ncar_r",height=4,aspect=4)
```

1. JOB

INFERENCE- Student label is more likely to say yes in comparison to all other labels.



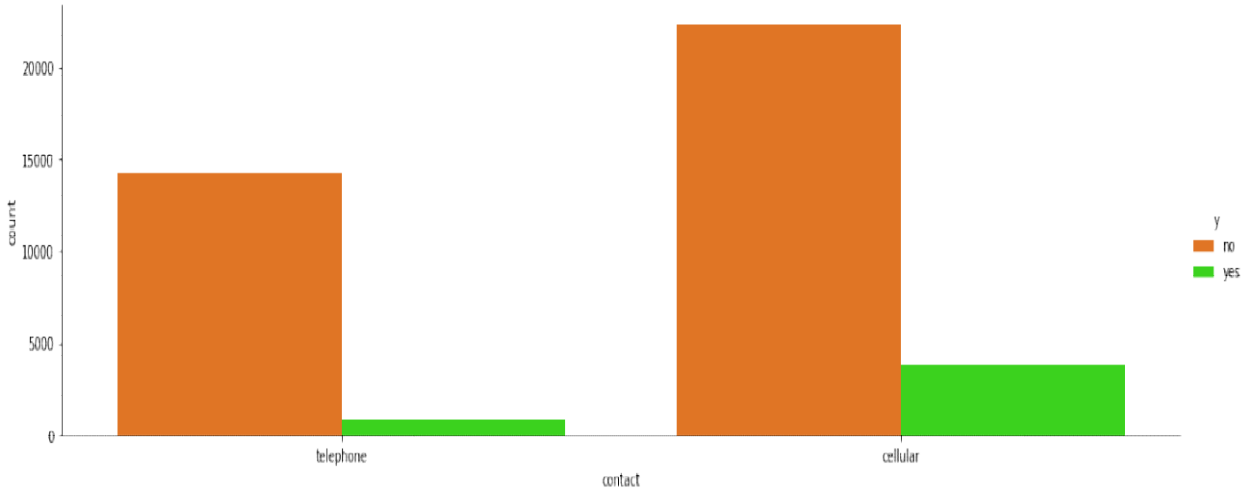
2. MARITAL STATUS



INFERENCE-

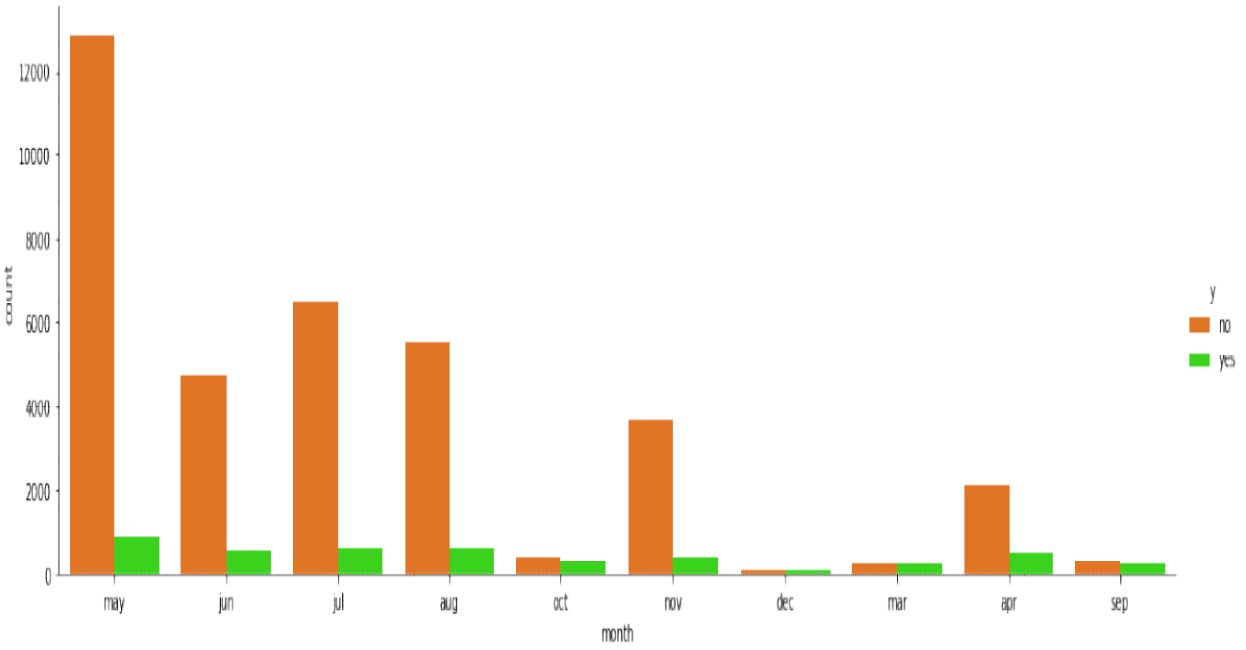
Single people are more likely to say yes in comparison to the married and divorced people.

3. CONTACT



INFERNCE- People having cellular as their contact communication type are more likely to say yes for the term deposit.

4. MONTH

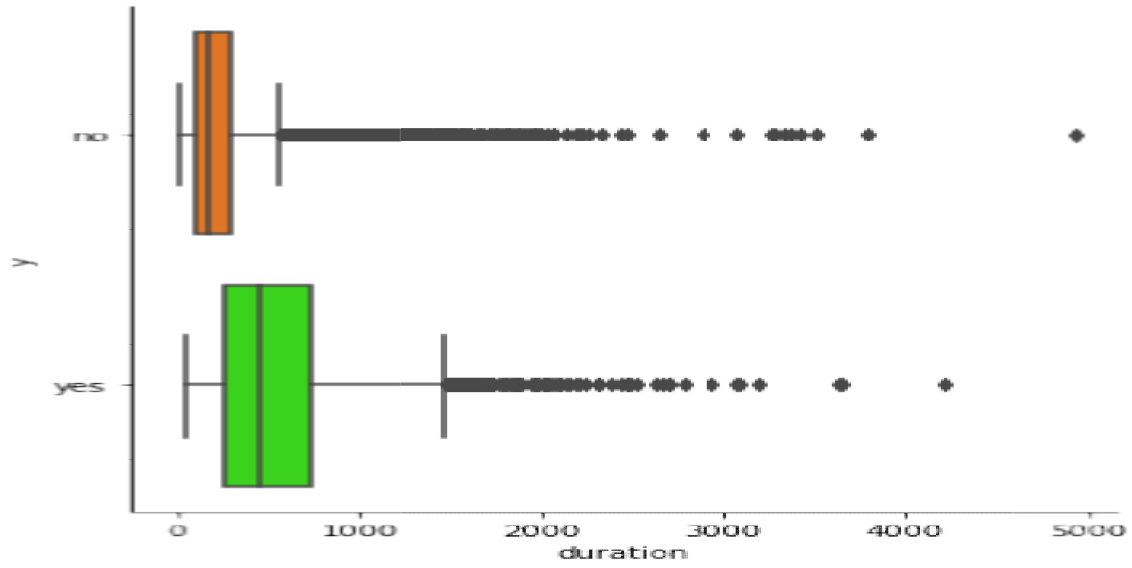


INFERNCE- People whose last contact month of the year is December/March/September/October are more likely to say yes.

3. Next we plotted our numerical variables against the target variable y as a boxplot. There were a total of 10 graphs and the graphs that offered an insight follow. The code for generating the graphs is as shown below. The main question that our inference from these graphs answer is that how does a unique numerical feature affect the outcome of our target variable?

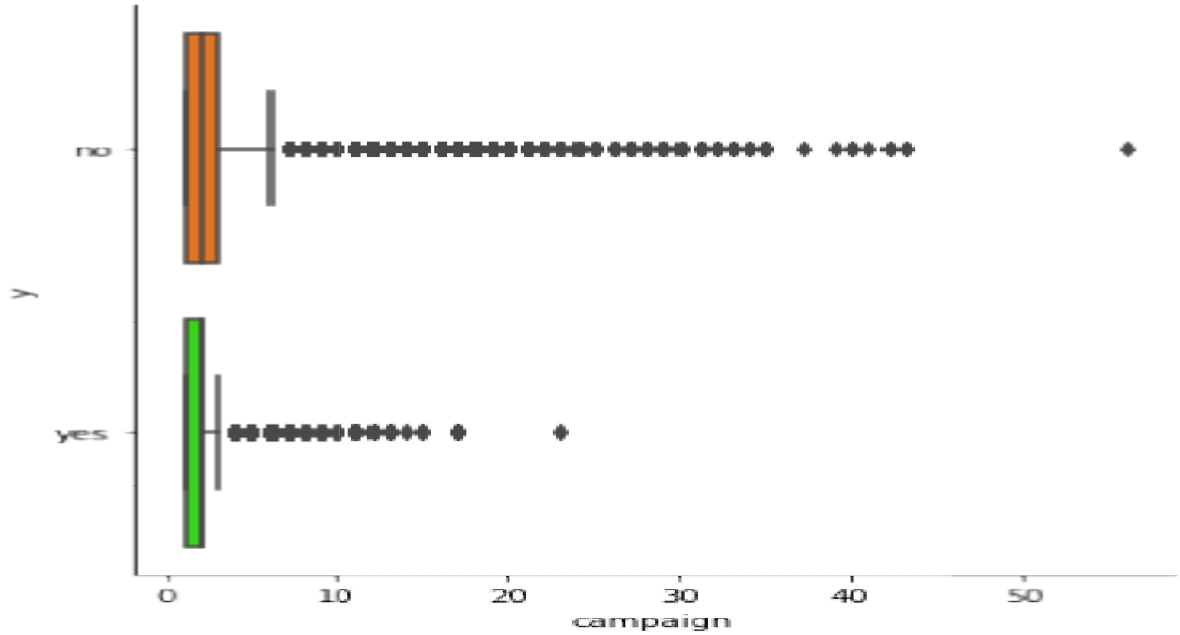
```
[ ]  1 for column in columns_num:
     2   sns.catplot(x=column,y='y',data=main_data,kind='box',palette="gist_ncar_r")
```

1. Duration



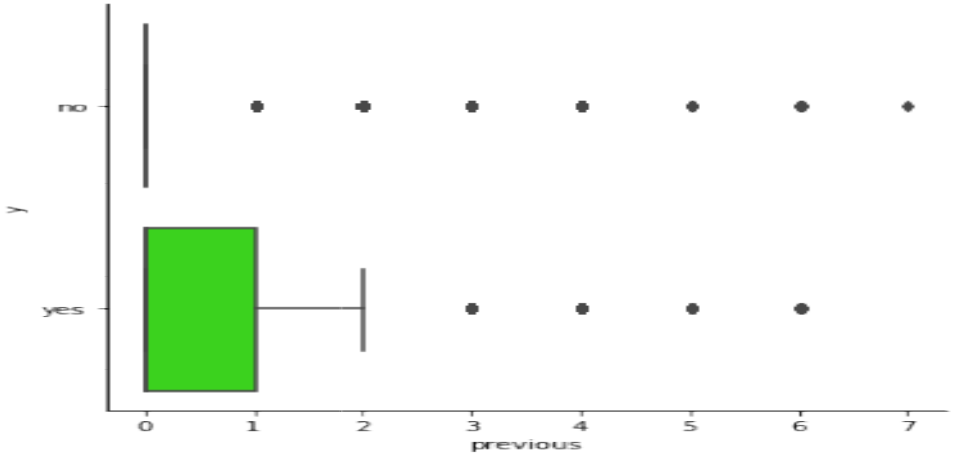
Inference: The Duration of call when the customer said no to the policy was disproportionately less as compared to when he said yes. Thus Duration feature should have high correlation with our target variable.

2. Campaign



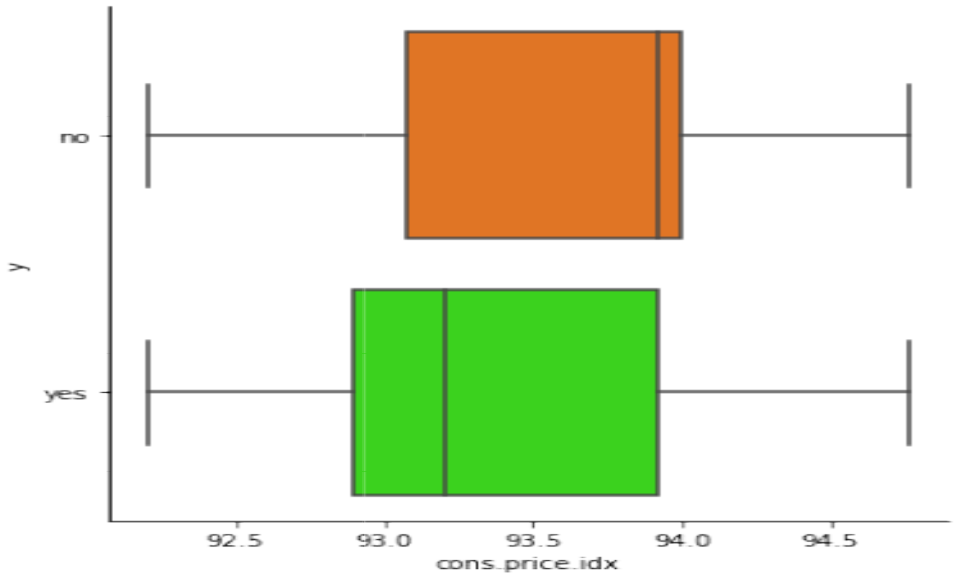
Inference: The campaign variable represents the number of contacts made during the campaign for an individual. We conclude from the graph that if a customer wants to open a term deposit he/she will more likely say yes in a few retries. More badgering of the customer is unlikely to result in a yes.

3. Previous



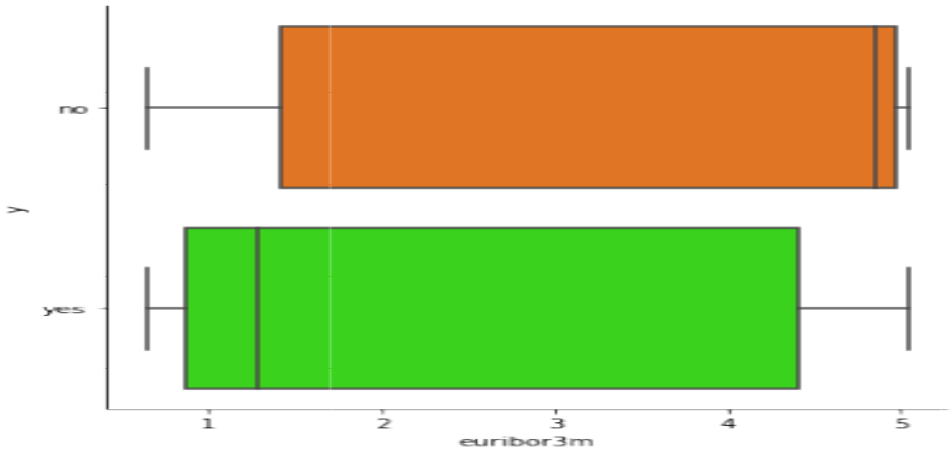
Inference: The previous variable represents the number of contacts made before the campaign for an individual. We conclude from the graph that if a customer was contacted by bank before for some other campaign he/she is more likely to say yes.

4. Consumer Price Index



Inference: The CPI seems to be inversely related to the response of the customer. The closer the CPI is to 93 higher are the chances of customer saying yes as median is near 93. This is not an absolute relation though as evident from the graph.

5. Euribor 3 month



Inference: The Euribor 3M seems to be inversely related to the response of the customer. Lower Euribor increases the chances of a positive response. This is not an absolute relation though as evident from the graph.

Feature Selection

We tried to study correlation between features and between the features and the target variable in order to equip ourselves with the knowledge necessary to choose the best model for our dataset.

```
[21] 1 train_columns=['age', 'job', 'marital', 'education', 'default', 'housing',
     2                 'loan', 'contact', 'month', 'day_of_week', 'duration', 'campaign',
     3                 'pdays', 'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
     4                 'cons.conf.idx', 'euribor3m', 'nr.employed']
```

```
[22] 1 X= main_data[train_columns].copy()
```

```
[23] 1 #Mapping
     2 for column in columns_cat:
     3   mapping = {}
     4   mapper=X[column].unique()
     5   for i,name in enumerate(mapper):
     6     mapping[name]=i
     7   X[column] = X[column].map(mapping)
```

```
1 y=main_data['y'].to_numpy()
2 y = y.reshape(y.shape[0], )
3 y=pd.Series(y)
```

a. We created a list called train column that consist of all of our 20 attributes
b. Then we created dummy dataset X and Label encoded our categorical variables to simplify their correlation with numerical variables.

We used three feature selection techniques to generate information about coorelations and relevance of features:

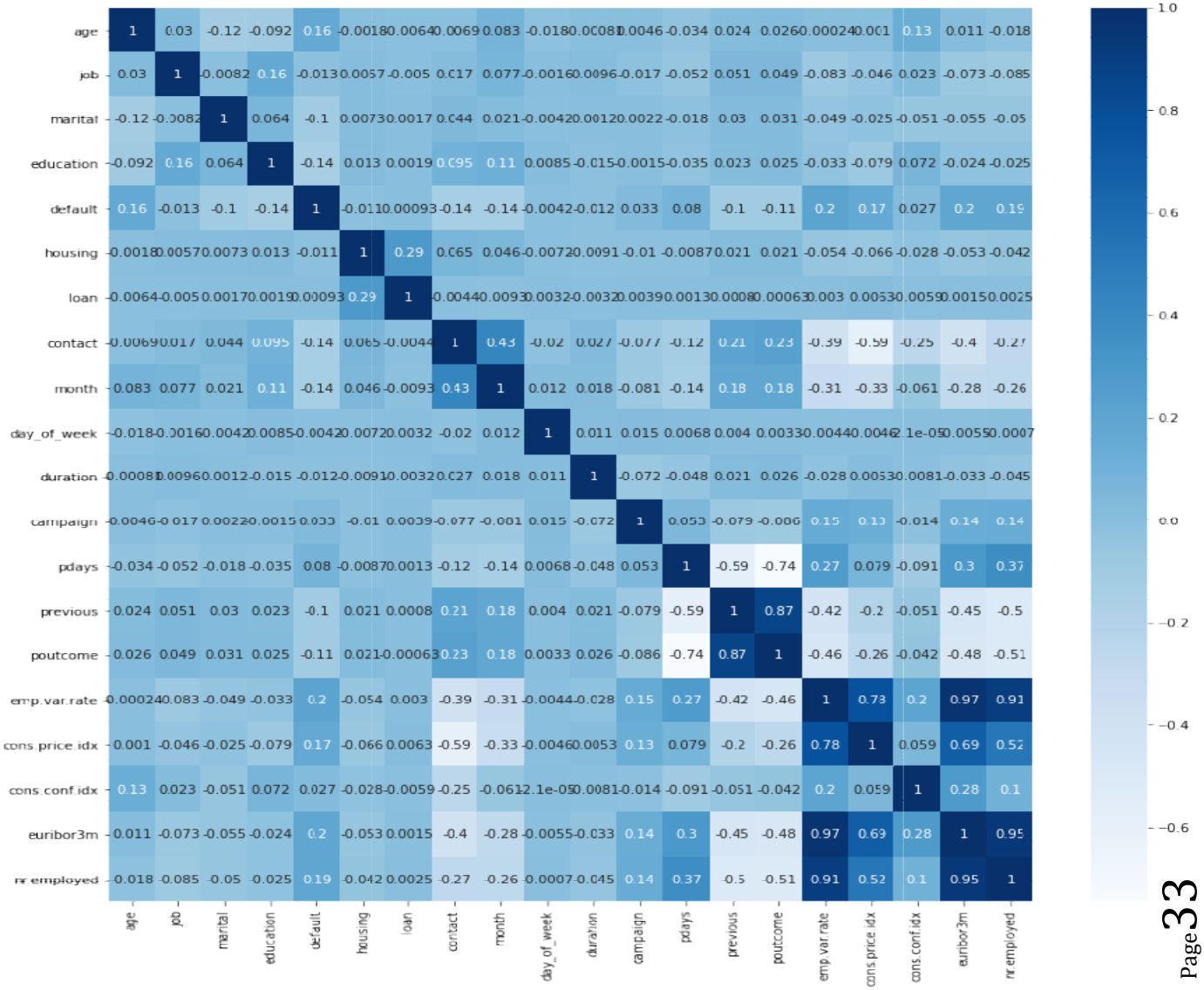
1. Heatmap
2. Generic Univariate Select using ANOVA F statistical test
3. Mutual Info Classification

1. Heatmap →

```
fig = plt.figure(figsize=(15,15))
sns.heatmap(X.corr(),cmap='Blues',annot=True)
```

The main inferences that we drew from this heatmap are:

1. All 5 Economic attributes (emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed) are strongly correlated with each other.(Correlation Coefficient ~90%)
2. All 4 Economic attributes are very weakly related to other attributes.(.(Correlation Coefficient is either negative or close to zero in most of the cases)
3.All attributes other than the economic indicators are very weakly correlated with each other. We can infer that due to small correlation we should not look towards elimination of these features as they are providing independent insights into the data .This is unless they show negligible correlation with the target variable.



2. Generic Univariate Select using ANOVA F statistical test

Image 1 →Code for applying Generic Univariate Select for sorting out the columns with respect to their correlation with target variable.

```
#Percentile selction using Generic Univariate Select

Selector = GenericUnivariateSelect(f_classif,mode="percentile")
Fit_data = Selector.fit_transform(X,y)

Sort_Score = Selector.scores_[::]

scores= zip(train_columns,Sort_Score)

final_scores= pd.DataFrame(scores, columns = ['Train_Features', 'Percentile'])
final_scores.sort_values(by=['Percentile'], inplace=True,ascending=False)

plt.figure(figsize=(40,8))
sns.barplot(x="Train_Features",y="Percentile",data=final_scores)
```

Code Explanation:

a. We here first initialize a Generic Univariate Select object with scoring function as ANOVA F statistical test using percentile of scores as our mode of selection

[ANOVA uses the **F-test** to determine whether the variability between group **means** is larger than the variability of the observations within the groups**.]**

b. Then we fit our data using fit_transform function and then sort our scores.

c. Then we create a final score dataframe and print it.

The inferences we draw from this table are as follows:

1. Duration feature is highly correlated to the target variable as speculated on page 29.

2. age,marital,day_of week,housing and loan features are very loosly related to target variable.

	Train_Features	Percentile
10	duration	8092.864331
19	nr.employed	5924.529145
12	pdays	4860.864315
18	euribor3m	4307.264549
15	emp.var.rate	4021.298297
14	poutcome	3247.936075
13	previous	2304.020429
8	month	1435.402405
7	contact	881.464614
16	cons.price.idx	777.463752
4	default	410.570123
11	campaign	182.121625
1	job	124.152866
17	cons.conf.idx	124.027852
3	education	86.032601
0	age	38.037762
2	marital	28.636788
9	day_of_week	4.111995
5	housing	3.799268
6	loan	1.050441

3. Mutual Info Classification

Mutual information (MI) between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.

The function relies on nonparametric methods based on entropy estimation from k-nearest neighbors distances

```
#Mutual Info Classification of Features

mi=mutual_info_classif(X,y)
lst=X.columns.tolist()
mi=pd.DataFrame(mi,columns=['MI_Classification'])
mi['Feature']=lst

columns_titles = ["Feature","MI_Classification"]
mi=mi.reindex(columns=columns_titles)
mi.sort_values(by=['MI_Classification'], inplace=True,ascending=False)
print(mi)
```

	Feature	MI_Classification
10	duration	0.078083
18	euribor3m	0.075453
16	cons.price.idx	0.068997
17	cons.conf.idx	0.067187
19	nr.employed	0.065230
15	emp.var.rate	0.055989
12	pdays	0.035782
14	poutcome	0.031362
8	month	0.027409
13	previous	0.019643
7	contact	0.013855
0	age	0.012764
1	job	0.011251
11	campaign	0.005082
4	default	0.005006
3	education	0.003186
5	housing	0.002660
9	day_of_week	0.002032
2	marital	0.001658
6	loan	0.000698

We can infer from this table that:

1. Duration feature is indeed highly correlated to the target variable and this has been verified twice now.

2. Out of age,marital,day_of week,housing and loan features that stood out due to their low ANOVA F score housing, marital,loan and day_of_week are again showing low correlation with target variable when calculated using mutual information.

We decide to use a Decision Tree classifier as our benchmark to deduce if removing them would have a positive effect or not.

Using All features:

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

X= main_data[train_columns].copy()

y=main_data['y'].to_numpy()
y = y.reshape(y.shape[0], )
y=pd.Series(y)
```

```
X=pd.get_dummies(X) # One Hot Encoding
X_train,X_valid,y_train,y_valid=train_test_split(X,y,test_size=0.3,random_sta
```

```

benchmark_model=DecisionTreeClassifier(random_state=21)

benchmark_model.fit(X_train, y_train)
pred=benchmark_model.predict(X_valid)

cf_rep=classification_report(y_valid,pred,digits=7)

print('---------------Benchmark Model with 20 features--------------\n')
print(cf_rep)
print('-------------------------------------------------------------')
```

Using 16 features(removing housing, marital,loan and day_of_week):

```
Trimmed_features = ['age', 'job', 'education', 'default', 'contact', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed']

X= main_data[Trimmed_features].copy()

y=main_data['y'].to_numpy()
y = y.reshape(y.shape[0], )
y=pd.Series(y)
```

```
X=pd.get_dummies(X) # One Hot Encoding
X_train,X_valid,y_train,y_valid=train_test_split(X,y,test_size=0.3,random_state=21)
```

```
benchmark_model=DecisionTreeClassifier(random_state=21)

benchmark_model.fit(X_train, y_train)
pred=benchmark_model.predict(X_valid)


cf_rep=classification_report(y_valid,pred,digits=7)
print('---------------Benchmark Model with 16 features--------------\n')
print(cf_rep)
print('-------------------------------------------------------------')
```

Code Description

We create a list containing the name of attributes and create test and train splits to test our benchmark model.

Results

----------Benchmark Model with 20 features----------

	precision	recall	f1-score	support
no	0.9378848	0.9338518	0.9358639	10930
yes	0.5081633	0.5249473	0.5164189	1423
accuracy			0.8867482	12353
macro avg	0.7230240	0.7293995	0.7261414	12353
weighted avg	0.8883831	0.8867482	0.8875461	12353

----------Benchmark Model with 16 features----------

	precision	recall	f1-score	support
no	0.9367853	0.9368710	0.9368281	10930
yes	0.5147679	0.5144062	0.5145870	1423
accuracy			0.8882053	12353
macro avg	0.7257766	0.7256386	0.7257076	12353
weighted avg	0.8881711	0.8882053	0.8881882	12353

The Benchmark model with 16 features shows a slight increase in recall and f1-score but precision declines slightly. As such no drastic effect was observed so we decided to not drop housing, marital,loan and day_of_week features.

Data Preprocessing

Before we get started with data preprocessing we should remove duplicate data as it may otherwise give us an inflated sense of efficacy of our model that we develop afterwards.

Here our code returns 12 duplicated entries and we drop them from our dataset to increase its robustness.

```
[ ]  1 # Cheking for duplicate instances
     2 if len(main_data[main_data.duplicated()]) > 0:
     3   print("\n***Number of duplicated entries: ", len(main_data[main_data.duplicated()]))
     4   display(main_data[main_data.duplicated(keep=False)].sort_values(by=list(main_data.columns)).head())
     5 else:
     6   print("\nNo duplicated entries found")
```

```
***Number of duplicated entries:  12
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx
28476	24	services	single	high.school	no	yes	no	cellular	apr	tue	114	1	999	0	nonexistent	-1.8	93.075
28477	24	services	single	high.school	no	yes	no	cellular	apr	tue	114	1	999	0	nonexistent	-1.8	93.075
14155	27	technician	single	professional.course	no	no	no	cellular	jul	mon	331	2	999	0	nonexistent	1.4	93.918
14234	27	technician	single	professional.course	no	no	no	cellular	jul	mon	331	2	999	0	nonexistent	1.4	93.918
18464	32	technician	single	professional.course	no	yes	no	cellular	jul	thu	128	1	999	0	nonexistent	1.4	93.918

```
[ ]  1 # Removing duplicate instances
     2
     3 main_data.drop_duplicates(inplace=True)
     4 main_data.reset_index(inplace=True)
```

```
[ ]  1 if len(main_data[main_data.duplicated()]) > 0:
     2   print("\n***Number of duplicated entries: ", len(main_data[main_data.duplicated()]))
     3   display(main_data[main_data.duplicated(keep=False)].sort_values(by=list(main_data.columns)).head())
     4 else:
     5   print("\nNo duplicated entries found")
```

```
No duplicated entries found
```

Categorical Encoding

When it comes to encoding categorical variables we have two main options:

1. Label Encoding
2. One Hot Encoding

We would be encoding our variables with One Hot Encoding because :

1. The categorical features are **not ordinal**

2. The number of categorical features is less so one-hot encoding can be effectively applied without high memory consumption.

We would be using pandas's get_dummies function to get one hot encoded data.

```
15 X=pd.get_dummies(X) # One Hot Encoding
```

Result

```
 1 X.columns
```

```
Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed',
       'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
       'marital_divorced', 'marital_married', 'marital_single',
       'marital_unknown', 'education_basic.4y', 'education_basic.6y',
       'education_basic.9y', 'education_high.school', 'education_illiterate',
       'education_professional.course', 'education_university.degree',
       'education_unknown', 'default_no', 'default_unknown', 'default_yes',
       'housing_no', 'housing_unknown', 'housing_yes', 'loan_no',
       'loan_unknown', 'loan_yes', 'contact_cellular', 'contact_telephone',
       'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun',
       'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
       'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu',
       'day_of_week_tue', 'day_of_week_wed', 'poutcome_failure',
       'poutcome_nonexistent', 'poutcome_success'],
      dtype='object')
```

Normalization

The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of value. It is required when features have different ranges. Here we have 10 numerical features with varied ranges, so normalization should be done in order to reduce the chance of overfitting.

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

X= main_data[train_columns].copy()

y=main_data['y'].to_numpy()
y = y.reshape(y.shape[0], )
y=pd.Series(y)

label=X[columns_num].columns
X[columns_num] = pd.DataFrame(data=Normalizer(norm='l2').fit_transform(X[columns_num]),columns=label)

X=pd.get_dummies(X) # One Hot Encoding
X_train,X_valid,y_train,y_valid=train_test_split(X,y,test_size=0.3,random_state=21)
```

Code Description

We used sklearn's preprocessing module to normalize our data. L2 norm for regularisation also called **ridge regression** is followed because:

1. The other option L1 norm optimizes the median thus is not sensitive to outliers. We have earlier observed substantial number of outliers while plotting numerical features on box plot, so L1 would not work well here.

2. L1 pushes to create sparse coefficient vectors and reduce dimentionality of data. As noted earlier in the feature selection section we don't intend to reduce dimentionality of the data. So L1 was not picked.

```
X_train
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
18510	0.005211	0.135302	0.000372	0.185924	0.0	0.000261	0.017479	-0.007947	0.000925	0.973000
464	0.006049	0.030625	0.000189	0.188854	0.0	0.000208	0.017769	-0.006881	0.000918	0.981323
6850	0.006615	0.038743	0.000567	0.188800	0.0	0.000208	0.017764	-0.006879	0.000918	0.981043
11745	0.010518	0.001878	0.002817	0.187640	0.0	0.000263	0.017743	-0.007851	0.000931	0.981984
33294	0.007886	0.025198	0.000192	0.192156	0.0	-0.000346	0.017868	-0.008887	0.000248	0.980806
...	...	...	...	...	...	...	...	...	...	...
16432	0.006761	0.014462	0.002066	0.187627	0.0	0.000263	0.017639	-0.008020	0.000932	0.981917
8964	0.006385	0.024036	0.000188	0.187594	0.0	0.000263	0.017739	-0.007849	0.000914	0.981740
5944	0.006958	0.106254	0.000376	0.187872	0.0	0.000207	0.017677	-0.006845	0.000913	0.976220
5327	0.007369	0.043836	0.000189	0.188760	0.0	0.000208	0.017760	-0.006878	0.000918	0.980831
15305	0.009952	0.022722	0.000376	0.187594	0.0	0.000263	0.017636	-0.008018	0.000931	0.981742

Results:

```
[41]  1 benchmark_model=DecisionTreeClassifier(random_state=21)
      2
      3 benchmark_model.fit(X_train, y_train)
      4 pred=benchmark_model.predict(X_valid)
      5
      6 cf_rep=classification_report(y_valid,pred,digits=7)
      7 print('------------Benchmark Model with 20 features using Normalized Data-----------------\n')
      8 print(cf_rep)
      9 print('-----------------------------------------------------------------------------------')
```

```
------------Benchmark Model with 20 features using Normalized Data-----------------

              precision    recall  f1-score   support

          no  0.9375572 0.9368710 0.9372140     10930
         yes  0.5178197 0.5207309 0.5192712      1423

    accuracy                      0.8889339     12353
   macro avg  0.7276885 0.7288009 0.7282426     12353
weighted avg  0.8892057 0.8889339 0.8890692     12353

-----------------------------------------------------------------------------
----------------Benchmark Model with 20 features--------------

                precision    recall  f1-score   support

           no  0.9378848 0.9338518 0.9358639     10930
          yes  0.5081633 0.5249473 0.5164189      1423

    accuracy                      0.8867482     12353
   macro avg  0.7230240 0.7293995 0.7261414     12353
weighted avg  0.8883831 0.8867482 0.8875461     12353

-------------------------------------------------------------
```

Apart from inherently preventing over fitting of dataset L2 has also helped us marginally improve our classification report.

Note : Here and hereafter we choose f1 score to be our primary metric because it integrates the precision and recall metric and gives us a good idea of our model's performance.

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

TP = number of true positives

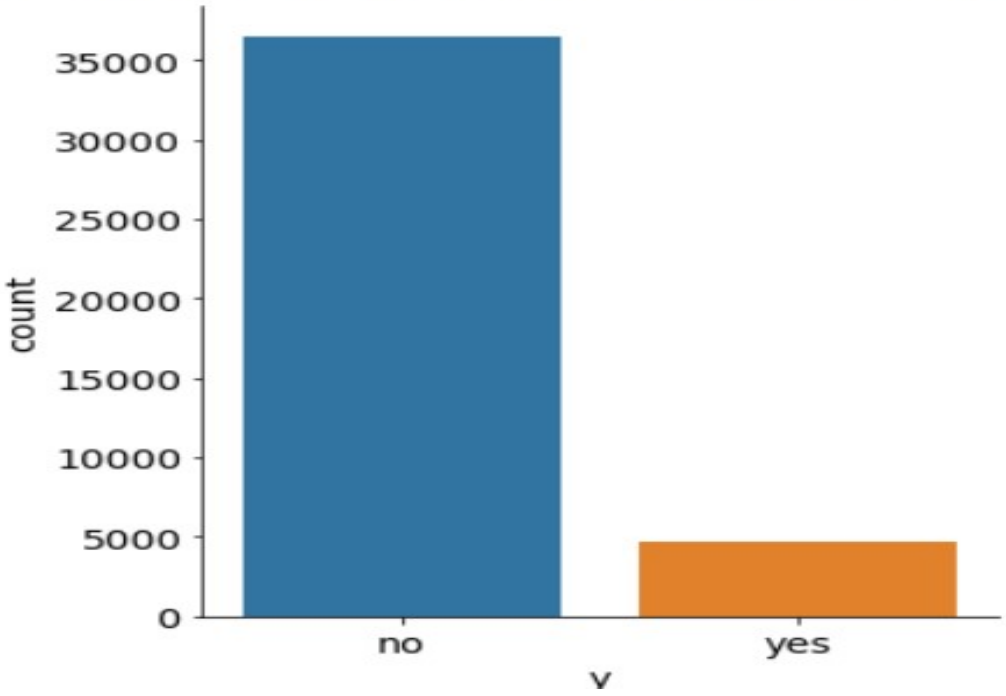
FP = number of false positives

FN = number of false negatives

The Problem of Imbalance:

```
1 sns.catplot(x='y',data=main_data,kind='count')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4da6418c88>
```



Imbalanced classes are a common problem in machine learning classification where there is a disproportionate ratio of observations in each class.

In the binary classification task that we are undertaking our dataset is disproportionately skewed towards no as the probable response.

The model might over fit to the class that's represented more in our dataset and become oblivious to the existence of the minority class. It might even give us a good accuracy but fail miserably in real life.

To overcome this problem we use SMOTE.

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them.

SMOTE allows us to generate samples. However, this method of over-sampling does not have any knowledge regarding the underlying distribution. Therefore, some noisy samples can be generated, e.g. when the different classes cannot be well separated. Hence, it can be beneficial to apply an under-sampling algorithm to clean the noisy samples. Two methods are usually used

(i) Tomek's link
(ii) Edited nearest neighbours

Imbalanced-learn module provides two ready-to-use samplers SMOTETomek and SMOTEENN.
In general, SMOTEENN cleans more noisy data than SMOTETomek so we will be using that to create balance in our dataset.

Code Description:

We used imblearn's module to oversample the dataset.

```
print("Before OverSampling, counts of label 'yes': {}".format(sum(y_train == 'yes')))
print("Before OverSampling, counts of label 'no': {} \n".format(sum(y_train == 'no')))

```

```
Before OverSampling, counts of label 'yes': 3216
Before OverSampling, counts of label 'no': 25607
```

```
sm = SMOTEENN(random_state = 21,sampling_strategy='all')
X_train, y_train = sm.fit_sample(X_train, y_train.ravel())
```

Then we used our benchmark model to investigate the effect of SMOTE.

```
print("After OverSampling, counts of label 'yes': {}".format(sum(y_train == 'yes')))
print("After OverSampling, counts of label 'no': {} \n".format(sum(y_train == 'no')))

```

```
After OverSampling, counts of label 'yes': 21221
After OverSampling, counts of label 'no': 16616
```

```
benchmark_model=DecisionTreeClassifier(random_state=21)

benchmark_model.fit(X_train, y_train)
pred=benchmark_model.predict(X_valid)

cf_rep=classification_report(y_valid,pred,digits=7)
print('------------Benchmark Model with 20 features using SMOTE with Normalized Data----------------\n')
print(cf_rep)
print('-------------------------------------------------------------------------')

```

Results

```
------------Benchmark Model with 20 features using Normalized Data-----------------

              precision    recall  f1-score   support

           no  0.9375572 0.9368710 0.9372140     10930
          yes  0.5178197 0.5207309 0.5192712      1423

     accuracy                      0.8889339     12353
    macro avg  0.7276885 0.7288009 0.7282426     12353
 weighted avg  0.8892057 0.8889339 0.8890692     12353

-------------------------------------------------------------------------------


------------Benchmark Model with 20 features using SMOTE with Normalized Data-----------------

              precision    recall  f1-score   support

           no  0.9603931 0.8940531 0.9260365     10930
          yes  0.4683196 0.7167955 0.5665093      1423

     accuracy                      0.8736339     12353
    macro avg  0.7143563 0.8054243 0.7462729     12353
 weighted avg  0.9037089 0.8736339 0.8846209     12353

-------------------------------------------------------------------------------------------------
```

We find that overall weighted recall and f1-score have decreased and overall weighted precision has increased but our main concern has been resolved.

Before oversampling for the minority class ‘yes’ we had

Recall = 0.52
F1-Score = 0.51

After oversampling for the minority class ‘yes’ we have

Recall = 0.71
F1-Score = 0.56

Oversampling has led to improvements in recall and f1-score and has helped create some semblance of balance in our previously highly skewed dataset and should help reduce the risk of overfitting.

Model Selection

Our data is multivariate and we are undergoing binary classification. We would be using decision trees to build our model and would then further optimize it by using boosting.

We choose decision trees for our project because a significant advantage of a decision tree is that it forces the consideration of all possible outcomes of a decision and traces each path to a conclusion. It creates a comprehensive analysis of the consequences along each branch and identifies decision nodes that need further analysis.

The only disadvantage of decision tree is that a single tree is a weak learner by default.

We resolve this issue by optimizing our model using Boosting. Boosting is an **ensemble technique**, where a set of weak learners are combined to create a strong learner that obtains much better performance than a single weak learner.

1. Decision tree model

We have previously deployed an un-optimized Decision Tree Model as our benchmark. Now we tune it using Randomized Grid Cross Validation. The two parameters we are looking to optimize are the max leaf nodes and max depth of our tree.

```
[197]
# Create the parameter grid: Decsion Tree Parameter Grid

param_grid = {
    'criterion': ['entropy'],
    'splitter' : ['best'],
    'max_leaf_nodes' : range(512,4096,256),
    'max_depth' : range(8,16,2)
}


# Instantiate the regressor: gbm
dec_classifier = DecisionTreeClassifier(random_state=21)

#Perform random search: grid_mse
dec_classifier_ = RandomizedSearchCV(param_distributions=param_grid, estimator = dec_classifier,verbose=21, scoring = ['f1_weighted','precision_weighted','recall_weighted',
                                                                                                                        'roc_auc'],refit='f1_weighted',n_jobs=4,n_iter=10,cv=10)


#Fit randomized_mse to the data
dec_classifier_.fit(X_train, y_train)

#Print the best parameters and lowest RMSE
print("Best parameters found: ", dec_classifier_.best_params_)
print("Best f1_weighted found: ", dec_classifier_.best_score_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 tasks      | elapsed:    5.0s
[Parallel(n_jobs=4)]: Done   2 tasks      | elapsed:    5.3s
[Parallel(n_jobs=4)]: Done   3 tasks      | elapsed:    5.3s
[Parallel(n_jobs=4)]: Done  91 tasks      | elapsed:   1.0min
[Parallel(n_jobs=4)]: Done  92 tasks      | elapsed:   1.0min
[Parallel(n_jobs=4)]: Done  93 tasks      | elapsed:   1.1min
[Parallel(n_jobs=4)]: Done  98 out of 100 | elapsed:   1.1min remaining:    1.4s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:   1.1min finished
Best parameters found:  {'splitter': 'best', 'max_leaf_nodes': 3840, 'max_depth': 14, 'criterion': 'entropy'}
Best f1_weighted found:  0.9577292006286602
```

We run 100 iterations with tenfold cross validation with four scoring metrics and refitting the data based on f1_weighted metric.

Result

```
[198] 1 benchmark_model=DecisionTreeClassifier(random_state=21,splitter ='best', max_depth=14 , criterion= 'entropy',max_leaf_nodes=3840)
      2
      3 benchmark_model.fit(X_train, y_train)
      4 pred=benchmark_model.predict(X_valid)
      5
      6 cf_rep=classification_report(y_valid,pred,digits=7)
      7 print('-----------Decison Tree Model with Hyperparameter tuning----------------\n')
      8 print(cf_rep)
      9 print('-------------------------------------------------------------------------')
     10
     11
```

```
-----------Decison Tree Model with Hyperparameter tuning----------------

              precision    recall  f1-score   support

          no  0.9649349 0.8887466 0.9252750     10930
         yes  0.4680665 0.7519325 0.5769749      1423

    accuracy                      0.8729863     12353
   macro avg  0.7165007 0.8203396 0.7511250     12353
weighted avg  0.9076983 0.8729863 0.8851527     12353

-------------------------------------------------------------------------
```

```
------------Benchmark Model with 20 features using SMOTE with Normalized Data-----------------

               precision    recall  f1-score   support

           no  0.9603931 0.8940531 0.9260365     10930
          yes  0.4683196 0.7167955 0.5665093      1423

     accuracy                      0.8736339     12353
    macro avg  0.7143563 0.8054243 0.7462729     12353
 weighted avg  0.9037089 0.8736339 0.8846209     12353

-------------------------------------------------------------------------------------------------------
```

The result of randomized grid cross validation gives max_depth=14 and max_leaf_node=3840.

When we apply these observations on our model we see overall improvement in all the weighted metrics.

Earlier we had :

Precision→0.903
Recall →0.873
F1-score →0.884

After optimization we have :

Precision→0.907
Recall →0.873
F1-score →0.885

2. Boosted Decision tree model

```
[191]  1 # Create the parameter grid: gbm_param_grid
       2 gbm_param_grid = {
       3     'boosting_type':['gbdt'],
       4     'n_estimators': range(512,1024,128),
       5     'max_depth': range(3,6),
       6     'class_weight' : ['balanced'],
       7     'min_data_in_leaf': [20,30,40,50,60,70],
       8     'num_leaves': range(256,512,4),
       9     'learning_rate': np.linspace(0.05,0.50,10),
      10     'subsample':[0.8,0.9,1],
      11     'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1],
      12     'early_stopping_rounds': [100],
      13     'n_jobs':[4]
      14 }
      15
      16 eval_set = [(X_train, y_train)]
      17
      18 # Instantiate the regressor: gbm
      19 lgbm_random = lgb(random_state=21,objective='binary')
      20
      21 #Perform random search: grid_mse
      22 lgb_random = RandomizedSearchCV(param_distributions=gbm_param_grid,estimator = lgbm_random,verbose=20, scoring = ['f1_weighted','precision_weighted','recall_weighted',
      23                                                                                                                  'roc_auc'],refit='f1_weighted', n_iter = 100,cv=3,n_jobs=4)
      24
      25 #Fit randomized_mse to the data
      26 lgb_random.fit(X_train, y_train,eval_metric="error", eval_set=eval_set)
      27
      28 #Print the best parameters and lowest RMSE
      29 print("Best parameters found: ", lgb_random.best_params_)
      30 print("Best f1_weighted found: ", lgb_random.best_score_)
      31
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 tasks      | elapsed:   13.2s

[473]   training's binary_logloss: 0.00527171   training's binary_error: 0
Best parameters found:  {'subsample': 0.9, 'num_leaves': 376, 'n_jobs': 4, 'n_estimators': 640, 'min_data_in_leaf': 20, 'max_depth': 5, 'learning_rate': 0.2, 'early_stopping
Best f1 weighted found:  0.9734197476416663
```

For boosting our decision tree we used Light GBM classifier which is based on GBDT (Gradient Boosted Decision Tree).

Now we tune our Light GBM boosted using Randomized Grid Cross Validation. The parameters we are looking to optimize are:

1. n_estimators = Number of trees
2. max_depth = maximum allowed depth of the tree
3. mi_data_in_leaf = minimum instances of data allowed per leaf
4.num_leaves = maxium allowed leaves per tree
5. learning_rate = The learning rate shrinks the contribution of each new base model and prevents overfitting.
6. subsample = The fraction of samples to be used for fitting the individual base learners
7. colsample_bytree = It is the subsample ratio of feature to consider when constructing each tree.

We run 100 iterations with threefold cross validation with four scoring metrics and refitting the data based on f1_weighted metric.

Result

```
[196] 1 #LGBM
      2 model=lgb(objective='binary',max_bin=128,subsample = 1, num_leaves = 376, n_jobs = 4, n_estimators = 640, min_data_in_leaf = 20,
      3                 max_depth = 5, learning_rate = 0.2, colsample_bytree = 0.6, boosting_type = 'gbdt',class_weight= 'balanced',random_state=21)
      4 eval_set = [(X_train, y_train)]
      5
      6 model.fit(X_train, y_train,eval_metric="error", eval_set=eval_set, verbose=False)
      7 pred=model.predict(X_valid)
      8
      9 cf_rep=classification_report(y_valid,pred,digits=7)
     10 print('------------Boosted Decison Tree Model with Hyperparameter tuning----------------\n')
     11 print(cf_rep)
     12 print('-------------------------------------------------------------------------------')
     13
```

```
------------Boosted Decison Tree Model with Hyperparameter tuning----------------

              precision    recall  f1-score   support

          no  0.9569587 0.9235133 0.9399385     10930
         yes  0.5368421 0.6809557 0.6003717      1423

    accuracy                      0.8955719     12353
   macro avg  0.7469004 0.8022345 0.7701551     12353
weighted avg  0.9085635 0.8955719 0.9008223     12353

-------------------------------------------------------------------------------
```

```
------------Decison Tree Model with Hyperparameter tuning------------------

              precision    recall  f1-score   support

          no  0.9649349 0.8887466 0.9252750     10930
         yes  0.4680665 0.7519325 0.5769749      1423

    accuracy                      0.8729863     12353
   macro avg  0.7165007 0.8203396 0.7511250     12353
weighted avg  0.9076983 0.8729863 0.8851527     12353

-------------------------------------------------------------------------------
```

The result of randomized grid cross validation gives
1. max_depth=5
2. max_leaf_node=376
3.subsample=1
4.n_estimators = 640
5.min_data_in_leaf=20
6.learning rate=0.2
7.colsample_bytree=0.6

When we apply these observations on our model we see drastic overall improvement in all the weighted metrics.

Earlier we had :

Precision→0.9076
Recall →0.873
F1-score →0.885

After optimization we have:

Precision→0.9085
Recall →0.8955
F1-score →0.9008

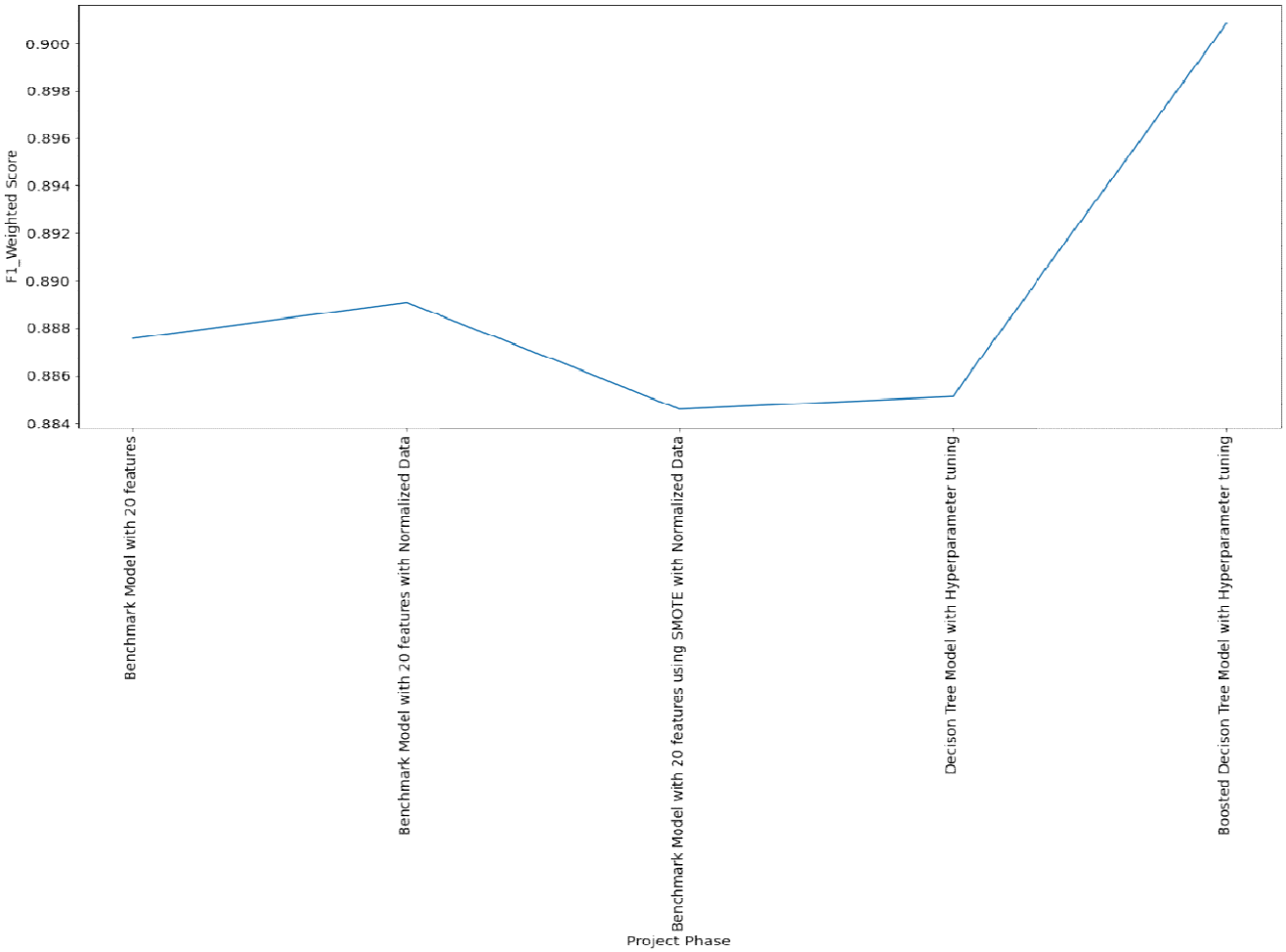
Summary

The graph below summarizes the project.
Github Repository [https://github.com/insomniac-klutz/Bank-Marketing-Classification]

```
feed=[['Benchmark Model with 20 features',0.8875461],['Benchmark Model with 20 features with Normalized Data',0.8890692],
      ['Benchmark Model with 20 features using SMOTE with Normalized Data',0.8846209],['Decison Tree Model with Hyperparameter tuning',0.8851527]
      ,['Boosted Decison Tree Model with Hyperparameter tuning',0.9008223]]
pd.set_option('max_colwidth', 80)
df=pd.DataFrame(feed,columns=['Project Phase','F1_Weighted Score'])
df
```

	Project Phase	F1_Weighted Score
0	Benchmark Model with 20 features	0.887546
1	Benchmark Model with 20 features with Normalized Data	0.889069
2	Benchmark Model with 20 features using SMOTE with Normalized Data	0.884621
3	Decison Tree Model with Hyperparameter tuning	0.885153
4	Boosted Decison Tree Model with Hyperparameter tuning	0.900822

```
plt.figure(figsize=(20,8))
plt.xticks(rotation=90)
sns.lineplot(x='Project Phase',y='F1_Weighted Score',data=df)
```



Bibliography

1. https://scikit-learn.org/stable/modules/classes.html
2. https://imbalanced-learn.readthedocs.io/en/stable/api.html
3. https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/
4. https://medium.com/datadriveninvestor/decision-tree-adventures-2-explanation-of-decision-tree-classifier-parameters-84776f39a28
5. https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html
6. https://seaborn.pydata.org/generated/seaborn.catplot.html