Trabalho Final para aprovação do curso Sistemas Inteligentes

Instruções:

- Comentem bastante o código, deixe o mais claro possível
- Usem e abusem do Markdown
- Isso ae!

Pre-Processamento

Importação de bibliotecas

In [1]:

```
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import *
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import matplotlib.colors
import pandas as pd
from sklearn.model_selection import train_test_split as tts
```

In [2]:

```
%config InlineBackend.figure_format = 'svg'

params = {'figure.figsize': [5, 5],
          'axes.labelsize': 16,
          'axes.titlesize':18,
          'font.size': 16,
          'legend.fontsize': 10,
          'xtick.labelsize': 12,
          'ytick.labelsize': 12
    }

plt.rcParams.update(params)
```

Carregando dataset

In [3]:

```
#Dataset para treinamento
db = pd.read_csv("bank-additional-dataset/bank-additional-full.csv", sep= ";");
```

Informações sobre o dataset

In [4]:

```
print("Dimensões do dataset", db.shape)
```

```
Dimensões do dataset (41188, 21)
```

In [5]:

```
print("Features do dataset: ", db.columns)
```

```
Features do dataset:  Index(['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', 'y'],
      dtype='object')
```

bank client data:

```
1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneu
r','housemaid','management','retired','self-employed','services','studen
t','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','singl
e','unknown'; note: 'divorced' means divorced or widowed)
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.schoo
l','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','telepho
ne')
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','tu
e','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric). Important not
e: this attribute highly affects the output target (e.g., if duration=0 t
hen y='no'). Yet, the duration is not known before a call is performed. A
lso, after the end of the call y is obviously known. Thus, this input sho
uld only be included for benchmark purposes and should be discarded if th
e intention is to have a realistic predictive model.
```

other attributes:

```
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 conta
cted from a previous campaign (numeric; 999 means client was not previous
ly 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')
```

social and economic context attributes

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

Output variable (desired target):

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

Como podemos observar há 11 features categóricas, é um número significante e teremos que tratá-los num futuro próximo

In [6]:

```
#tipos de dados para cada feature
db.dtypes
```

Out[6]:

```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
```

In [7]:

```
# print the first 20 rows of data
print(db.head(20))
```

```
    age          job   marital            education  default housin
g loan  \
0    56    housemaid   married             basic.4y       no      n
o   no
1    57     services   married          high.school  unknown      n
o   no
2    37     services   married          high.school       no     ye
s   no
3    40       admin.   married             basic.6y       no      n
o   no
4    56     services   married          high.school       no      n
o  yes
5    45     services   married             basic.9y  unknown      n
o   no
6    59       admin.   married  professional.course       no      n
o   no
7    41  blue-collar   married              unknown  unknown      n
o   no
8    24   technician    single  professional.course       no     ye
s   no
9    25     services    single          high.school       no     ye
s   no
10   41  blue-collar   married              unknown  unknown      n
o   no
11   25     services    single          high.school       no     ye
s   no
12   29  blue-collar    single          high.school       no      n
o  yes
13   57    housemaid  divorced             basic.4y       no     ye
s   no
14   35  blue-collar   married             basic.6y       no     ye
s   no
15   54      retired   married             basic.9y  unknown     ye
s  yes
16   35  blue-collar   married             basic.6y       no     ye
s   no
17   46  blue-collar   married             basic.6y  unknown     ye
s  yes
18   50  blue-collar   married             basic.9y       no     ye
s  yes
19   39   management    single             basic.9y  unknown      n
o   no

      contact month day_of_week ...  campaign  pdays  previous    
poutcome  \
0   telephone   may         mon ...         1    999         0  non
existent
1   telephone   may         mon ...         1    999         0  non
existent
2   telephone   may         mon ...         1    999         0  non
existent
3   telephone   may         mon ...         1    999         0  non
existent
4   telephone   may         mon ...         1    999         0  non
existent
5   telephone   may         mon ...         1    999         0  non
existent
6   telephone   may         mon ...         1    999         0  non
existent
7   telephone   may         mon ...         1    999         0  non
existent
```

```
8   telephone   may         mon ...         1    999         0  non
existent
9   telephone   may         mon ...         1    999         0  non
existent
10  telephone   may         mon ...         1    999         0  non
existent
11  telephone   may         mon ...         1    999         0  non
existent
12  telephone   may         mon ...         1    999         0  non
existent
13  telephone   may         mon ...         1    999         0  non
existent
14  telephone   may         mon ...         1    999         0  non
existent
15  telephone   may         mon ...         1    999         0  non
existent
16  telephone   may         mon ...         1    999         0  non
existent
17  telephone   may         mon ...         1    999         0  non
existent
18  telephone   may         mon ...         1    999         0  non
existent
19  telephone   may         mon ...         1    999         0  non
existent

   emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.emplo
yed   y
0           1.1          93.994          -36.4      4.857       519
1.0  no
1           1.1          93.994          -36.4      4.857       519
1.0  no
2           1.1          93.994          -36.4      4.857       519
1.0  no
3           1.1          93.994          -36.4      4.857       519
1.0  no
4           1.1          93.994          -36.4      4.857       519
1.0  no
5           1.1          93.994          -36.4      4.857       519
1.0  no
6           1.1          93.994          -36.4      4.857       519
1.0  no
7           1.1          93.994          -36.4      4.857       519
1.0  no
8           1.1          93.994          -36.4      4.857       519
1.0  no
9           1.1          93.994          -36.4      4.857       519
1.0  no
10          1.1          93.994          -36.4      4.857       519
1.0  no
11          1.1          93.994          -36.4      4.857       519
1.0  no
12          1.1          93.994          -36.4      4.857       519
1.0  no
13          1.1          93.994          -36.4      4.857       519
1.0  no
14          1.1          93.994          -36.4      4.857       519
1.0  no
15          1.1          93.994          -36.4      4.857       519
1.0  no
16          1.1          93.994          -36.4      4.857       519
1.0  no
```

```
17          1.1          93.994          -36.4       4.857       519
1.0  no
18          1.1          93.994          -36.4       4.857       519
1.0  no
19          1.1          93.994          -36.4       4.857       519
1.0  no

[20 rows x 21 columns]
```

In [8]:

```
db.describe()
```

Out[8]:

	age	duration	campaign	pdays	previous	en
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	411
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.0
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.5
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.4
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.8
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.1
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.4
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.4

In [9]:

```
# Contando número de valores nulos no dataset
db.isnull().sum(axis = 0)
```

Out[9]:

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

In [10]:

```
#Visualizando valores unicos do dataset
print("Age: ",db.age.unique())
print()
print("Job: ",db.job.unique())
print()
print("Maritial: ",db.marital.unique())
print()
print("Education: ",db.education.unique())
print()
print("Default: ",db.default.unique())
print()
print("Housing: ",db.housing.unique())
print()
print("loan: ",db.loan.unique())
print()
print("Contact: ",db.contact.unique())
print()
print("Month : ",db.month.unique())
print()
print("Duration: ",db.duration.unique())
print()
print("Campaign: ",db.campaign.unique())
print()
print("Pdays: ",db.pdays.unique())
print()
print("previous: ",db.previous.unique())
print()
print("Poutcome: ",db.poutcome.unique())
```

```
Age:  [56 57 37 40 45 59 41 24 25 29 35 54 46 50 39 30 55 49 34 52
58 32 38 44
 42 60 53 47 51 48 33 31 43 36 28 27 26 22 23 20 21 61 19 18 70 66
76 67
 73 88 95 77 68 75 63 80 62 65 72 82 64 71 69 78 85 79 83 81 74 17
87 91
 86 98 94 84 92 89]

Job:  ['housemaid' 'services' 'admin.' 'blue-collar' 'technician'
'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']

Maritial:  ['married' 'single' 'divorced' 'unknown']

Education:  ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'profes
sional.course'
 'unknown' 'university.degree' 'illiterate']

Default:  ['no' 'unknown' 'yes']

Housing:  ['no' 'yes' 'unknown']

loan:  ['no' 'yes' 'unknown']

Contact:  ['telephone' 'cellular']

Month :  ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'se
p']

Duration:  [ 261  149  226 ... 1246 1556 1868]

Campaign:  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 19 18 23 14 22 2
5 16 17 15 20 56
 39 35 42 28 26 27 32 21 24 29 31 30 41 37 40 33 34 43]

Pdays:  [999   6   4   3   5   1   0  10   7   8   9  11   2  12  1
3  14  15  16
  21  17  18  22  25  26  19  27  20]

previous:  [0 1 2 3 4 5 6 7]

Poutcome:  ['nonexistent' 'failure' 'success']
```

Tratamento do dataset por blocos

```
        - Bank client data
        - Last contact of the current campaign
        - Social and economic context attributes
        - Other attributes:
```

1. Bank Client Data

In [11]:

```
#Particionando o Dataset para trabalhar apenas com os Client Data
bank_client = db.iloc[: , 0:7]
bank_client.head()
```

Out[11]:

	age	job	marital	education	default	housing	loan
0	56	housemaid	married	basic.4y	no	no	no
1	57	services	married	high.school	unknown	no	no
2	37	services	married	high.school	no	yes	no
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

In [12]:

```
#Visualizando as amostras
print("Age: ",db.age.unique())
print()
print("Job: ",db.job.unique())
print()
print("Maritial: ",db.marital.unique())
print()
print("Education: ",db.education.unique())
print()
print("Default: ",db.default.unique())
print()
print("Housing: ",db.housing.unique())
print()
print("loan: ",db.loan.unique())
```

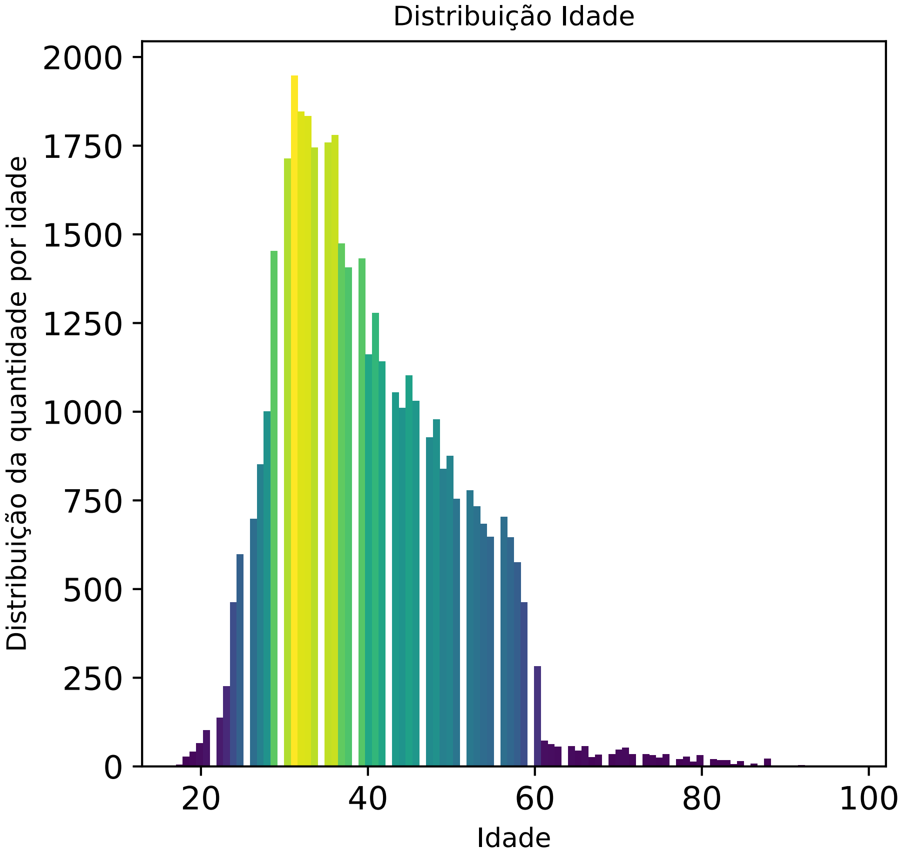
```
Age:  [56 57 37 40 45 59 41 24 25 29 35 54 46 50 39 30 55 49 34 52
58 32 38 44
 42 60 53 47 51 48 33 31 43 36 28 27 26 22 23 20 21 61 19 18 70 66
76 67
 73 88 95 77 68 75 63 80 62 65 72 82 64 71 69 78 85 79 83 81 74 17
87 91
 86 98 94 84 92 89]

Job:  ['housemaid' 'services' 'admin.' 'blue-collar' 'technician'
'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']

Maritial:  ['married' 'single' 'divorced' 'unknown']

Education:  ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'profes
sional.course'
 'unknown' 'university.degree' 'illiterate']

Default:  ['no' 'unknown' 'yes']

Housing:  ['no' 'yes' 'unknown']

loan:  ['no' 'yes' 'unknown']
```

Visualização dos Dados

Distribuição de idades

In [13]:

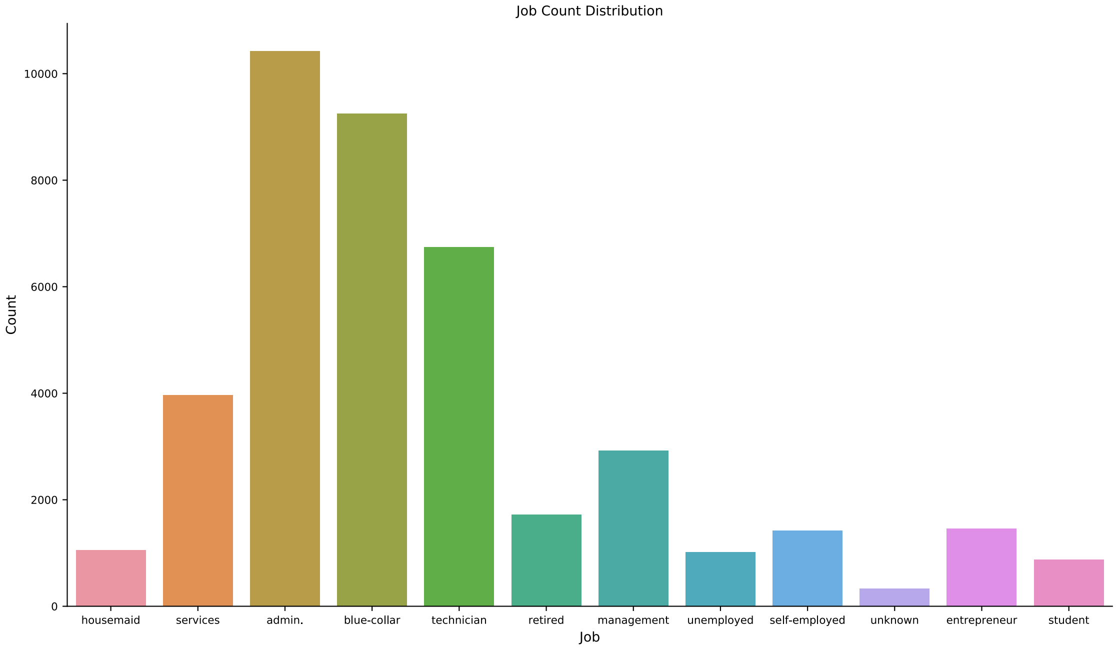
```
#Deixar Histograma colorido
N, bins, patches = plt.hist(bank_client['age'], bins = 100, orientation = 'verti
cal')
fracs = N/N.max()
norm = matplotlib.colors.Normalize(fracs.min(), fracs.max())
for thisfrac, thispatch in zip(fracs, patches):
    color = plt.cm.viridis(norm(thisfrac))
    thispatch.set_facecolor(color)
#fim comando para deixar colorido
plt.xlabel('Idade', fontsize =10)
plt.ylabel('Distribuição da quantidade por idade', fontsize =10)
plt.title('Distribuição Idade', fontsize =10)

plt.show()
```



Distribuição de trabalhos dos clientes

In [14]:

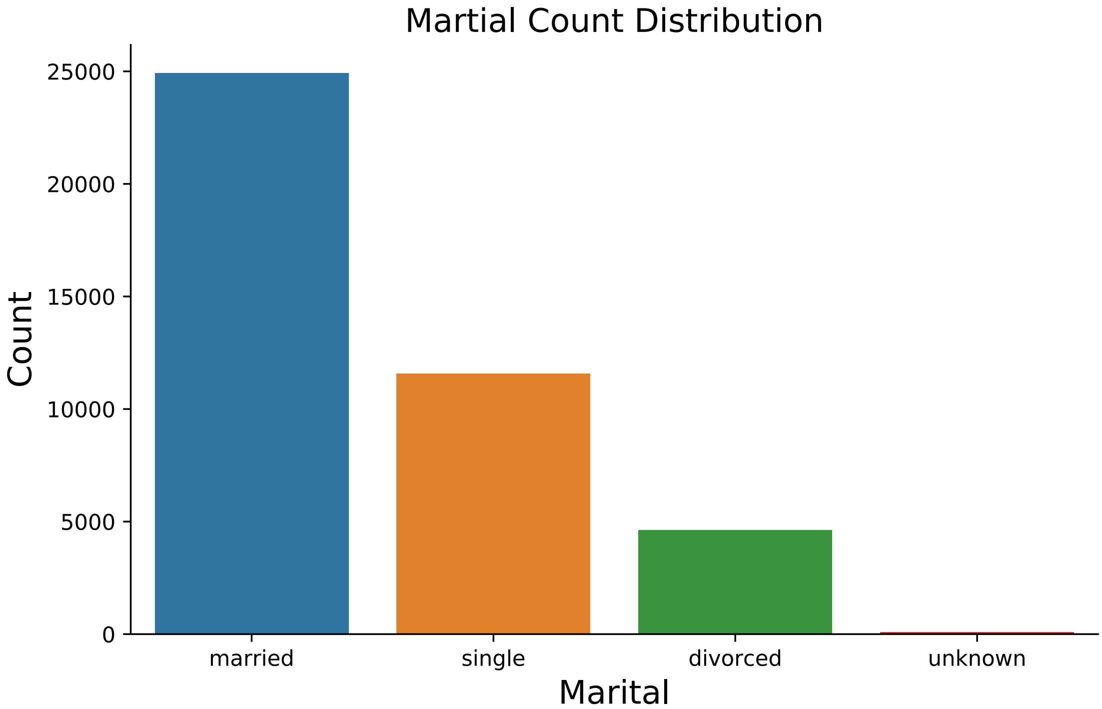
```
fig, ax = plt.subplots()
fig.set_size_inches(14, 8)
sns.countplot(x = 'job', data = bank_client)
ax.set_xlabel('Job', fontsize=10)
ax.set_ylabel('Count', fontsize=10)
ax.set_title('Job Count Distribution', fontsize=10)
ax.tick_params(labelsize=8)
sns.despine()
```



Distribuição estado civil

In [15]:

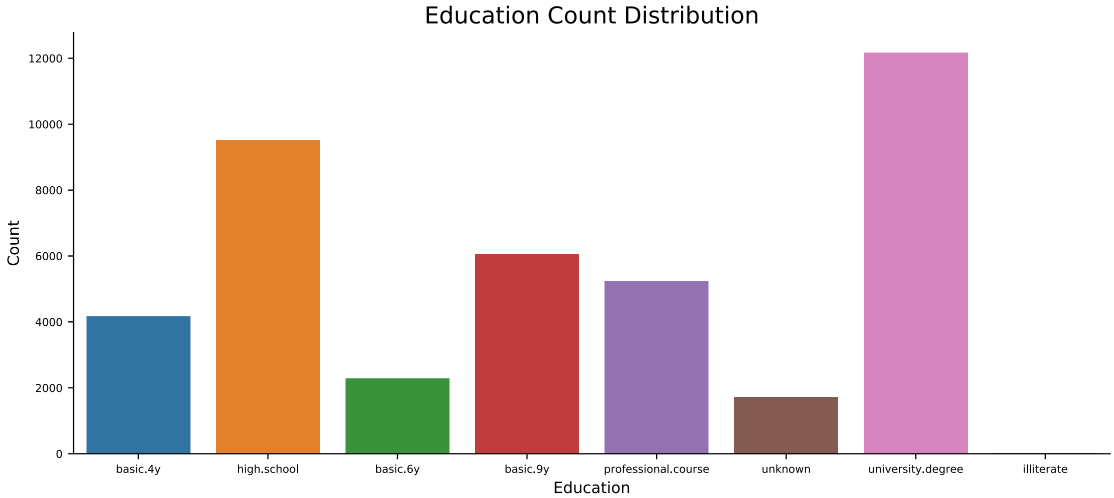
```
fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
sns.countplot(x = 'marital', data = bank_client)
ax.set_xlabel('Marital', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Martial Count Distribution', fontsize=15)
ax.tick_params(labelsize=10)
sns.despine()
```



Distribuição Escolaridade

In [16]:

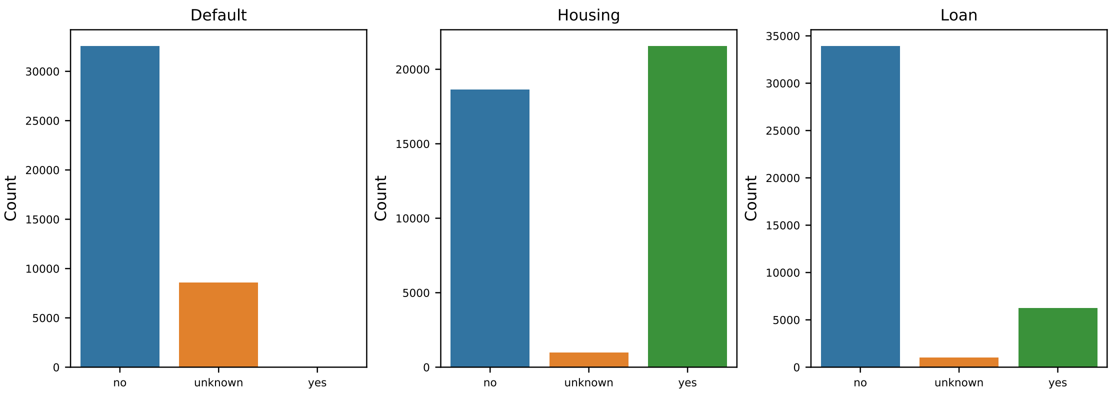
```
fig, ax = plt.subplots()
fig.set_size_inches(12, 5)
sns.countplot(x = 'education', data = bank_client)
ax.set_xlabel('Education', fontsize=10)
ax.set_ylabel('Count', fontsize=10)
ax.set_title('Education Count Distribution', fontsize=15)
ax.tick_params(labelsize=7)
sns.despine()
```



Distribuição Housing, Loan e Default

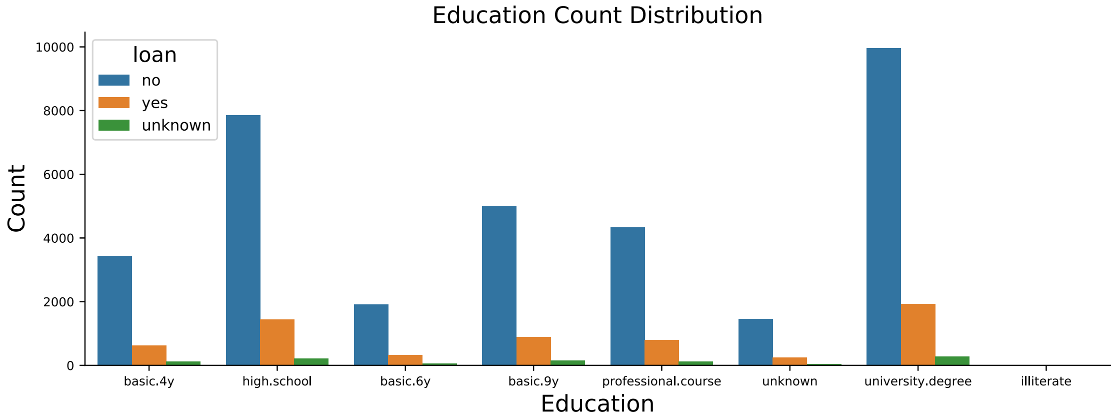
In [17]:

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (12,4))

sns.countplot(x = 'default', data = bank_client, ax = ax1, order = ['no', 'unkno
wn', 'yes'])
ax1.set_title('Default', fontsize=10)
ax1.set_xlabel('')
ax1.set_ylabel('Count', fontsize=10)
ax1.tick_params(labelsize=7)


sns.countplot(x = 'housing', data = bank_client, ax = ax2, order = ['no', 'unkno
wn', 'yes'])
ax2.set_title('Housing', fontsize=10)
ax2.set_xlabel('')
ax2.set_ylabel('Count', fontsize=10)
ax2.tick_params(labelsize=7)


sns.countplot(x = 'loan', data = bank_client, ax = ax3, order = ['no', 'unknown'
, 'yes'])
ax3.set_title('Loan', fontsize=10)
ax3.set_xlabel('')
ax3.set_ylabel('Count', fontsize=10)
ax3.tick_params(labelsize=7)

plt.subplots_adjust(wspace=0.25)
```



In [18]:

```
fig, ax = plt.subplots()
fig.set_size_inches(12, 4)
sns.countplot(x = 'education', hue = 'loan', data = bank_client)
ax.set_xlabel('Education', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Education Count Distribution', fontsize=15)
ax.tick_params(labelsize=8)
sns.despine()
```



Tratamento com os dados Categoricos

In [19]:

```
#Transformações de dados categoricos para valores numericos
labelencoder_X = LabelEncoder()
bank_client['job'] = labelencoder_X.fit_transform(bank_client['job'])
bank_client['marital'] = labelencoder_X.fit_transform(bank_client['marital'])
bank_client['education'] = labelencoder_X.fit_transform(bank_client['education'
])
bank_client['default'] = labelencoder_X.fit_transform(bank_client['default'])
bank_client['housing'] = labelencoder_X.fit_transform(bank_client['housing'])
bank_client['loan'] = labelencoder_X.fit_transform(bank_client['loan'])
```

In [20]:

```
bank_client.head()
```

Out[20]:

	age	job	marital	education	default	housing	loan
0	56	3	1	0	0	0	0
1	57	7	1	3	1	0	0
2	37	7	1	3	0	2	0
3	40	0	1	1	0	0	0
4	56	7	1	3	0	0	2

2. Related with the last contact of the current campaign

In [21]:

```
#Particionando o Dataset para trabalhar apenas com 'Related with the last contac
t of the current campaign'
bank_related = db.iloc[: , 7:11]
bank_related.head()
```

Out[21]:

	contact	month	day_of_week	duration
0	telephone	may	mon	261
1	telephone	may	mon	149
2	telephone	may	mon	226
3	telephone	may	mon	151
4	telephone	may	mon	307

In [22]:

```
#Visualizando as amostras do dataset
print("Contact: ", bank_related.contact.unique())
print()
print("Month : ", bank_related.month.unique())
print()
print("Day of week: ", bank_related.day_of_week.unique())
print()
print("Duration: ", bank_related.duration.unique())
print()
```

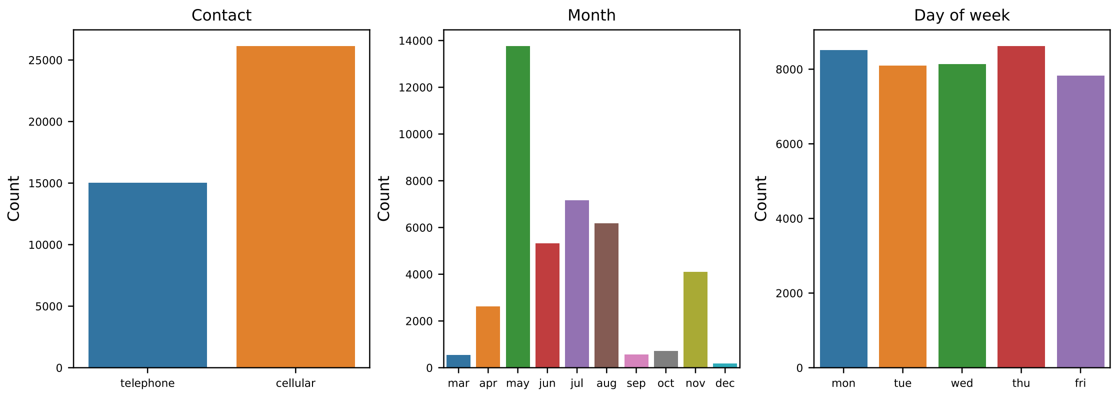
```
Contact:  ['telephone' 'cellular']

Month :  ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'se
p']

Day of week:  ['mon' 'tue' 'wed' 'thu' 'fri']

Duration:  [ 261  149  226 ... 1246 1556 1868]
```

In [23]:

```
# Distribuição Contatos, mes e dias da semana
```

In [24]:

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (12,4))

sns.countplot(x = 'contact', data = bank_related, ax = ax1, order = ['telephone'
, 'cellular'])
ax1.set_title('Contact', fontsize=10)
ax1.set_xlabel('')
ax1.set_ylabel('Count', fontsize=10)
ax1.tick_params(labelsize=7)


sns.countplot(x = 'month', data = bank_related, ax = ax2, order = ['mar', 'apr',
 'may', 'jun', 'jul', 'aug', 'sep', 'oct','nov', 'dec']
)
ax2.set_title('Month', fontsize=10)
ax2.set_xlabel('')
ax2.set_ylabel('Count', fontsize=10)
ax2.tick_params(labelsize=7)


sns.countplot(x = 'day_of_week', data = bank_related, ax = ax3, order = ['mon',
'tue', 'wed', 'thu', 'fri'])
ax3.set_title('Day of week', fontsize=10)
ax3.set_xlabel('')
ax3.set_ylabel('Count', fontsize=10)
ax3.tick_params(labelsize=7)

plt.subplots_adjust(wspace=0.25)

#['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'mar', 'apr', 'sep']
```

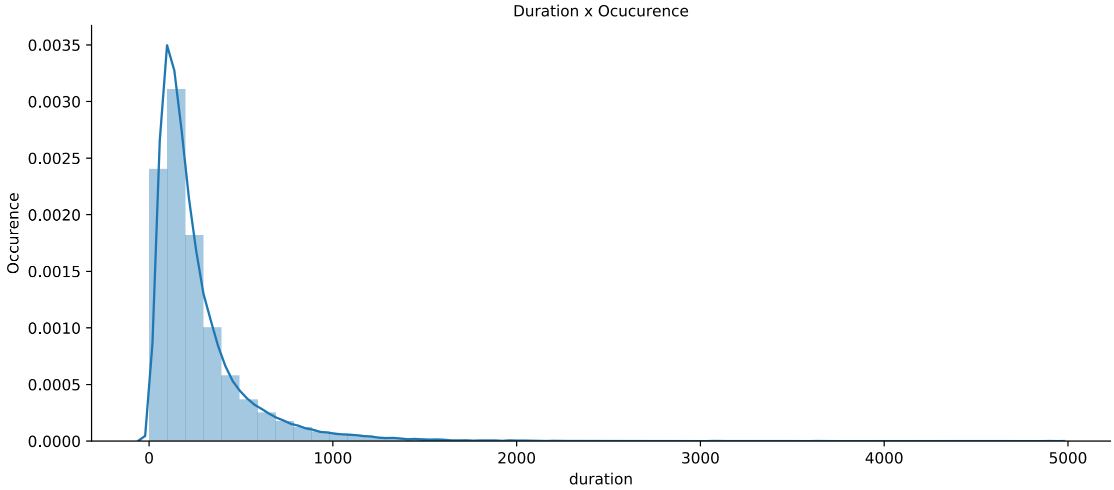


Distribuição duração vs ocorrência

In [25]:

```
fig, ax2 = plt.subplots()
fig.set_size_inches(12, 5)
ax2.set_xlabel('Duration Calls', fontsize=10)
ax2.set_ylabel('Occurence', fontsize=10)
ax2.set_title('Duration x Ocucurence', fontsize=10)
ax2.tick_params(labelsize=10)
sns.distplot(bank_related['duration'], ax = ax2)
sns.despine(ax = ax2)
```

```
/home/william/anaconda3/lib/python3.6/site-packages/matplotlib/axe
s/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and
has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [26]:

```
# As ligações que tiverem o tempo de duração da ligação zero automaticamente o t
arget será 'no', assim devemos excluir essas linhas
db[(db['duration'] == 0)]
```

Out[26]:

	age	job	marital	education	default	housing	loan	conta
6251	39	admin.	married	high.school	no	yes	no	telepho
23031	59	management	married	university.degree	no	yes	no	cellular
28063	53	blue-collar	divorced	high.school	no	yes	no	cellular
33015	31	blue-collar	married	basic.9y	no	no	no	cellular

4 rows × 21 columns

Tratamento com os dados Categoricos

In [27]:

```
#Transformações de dados categoricos para valores numericos
bank_related['contact']     = labelencoder_X.fit_transform(bank_related['contac
t'])
bank_related['month']       = labelencoder_X.fit_transform(bank_related['month'
])
bank_related['day_of_week'] = labelencoder_X.fit_transform(bank_related['day_of_
week'])


bank_related.head()
```

Out[27]:

	contact	month	day_of_week	duration
0	1	6	1	261
1	1	6	1	149
2	1	6	1	226
3	1	6	1	151
4	1	6	1	307

3. Social and economic context attributes

In [28]:

```
#Particionando o Dataset para trabalhar apenas com 'Social and Economic context
 attibutes'
bank_se = db.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribo
r3m', 'nr.employed']]
bank_se.head()
```

Out[28]:

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	1.1	93.994	-36.4	4.857	5191.0
1	1.1	93.994	-36.4	4.857	5191.0
2	1.1	93.994	-36.4	4.857	5191.0
3	1.1	93.994	-36.4	4.857	5191.0
4	1.1	93.994	-36.4	4.857	5191.0

4. Other attributes

In [29]:

```
bank_o = db.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
bank_o.head()
```

Out[29]:

	campaign	pdays	previous	poutcome
0	1	999	0	nonexistent
1	1	999	0	nonexistent
2	1	999	0	nonexistent
3	1	999	0	nonexistent
4	1	999	0	nonexistent

In [30]:

```
#Visualizando as amostras do dataset
print("Contact: ", bank_o.poutcome.unique())
print()
```

```
Contact:  ['nonexistent' 'failure' 'success']
```

Tratamento com dados categoricos

In [31]:

```
bank_o['poutcome'] = labelencoder_X.fit_transform(bank_o['poutcome'])
#bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2,3], inpl
ace  = True)

bank_o.head()
```

Out[31]:

	campaign	pdays	previous	poutcome
0	1	999	0	1
1	1	999	0	1
2	1	999	0	1
3	1	999	0	1
4	1	999	0	1

Validação dos dados

In [91]:

```
#Montado o dataset pre-processado
db_pronto= pd.concat([bank_client, bank_related, bank_se, bank_o], axis = 1)
db_pronto = db_pronto[['age', 'job', 'marital', 'education', 'default', 'housin
g', 'loan',
                     'contact', 'month', 'day_of_week', 'duration', 'emp.var.rat
e', 'cons.price.idx',
                     'cons.conf.idx', 'euribor3m', 'nr.employed', 'campaign', 'p
days', 'previous', 'poutcome']]
#Criando Target
y = pd.get_dummies(db['y'], columns = ['y'], prefix = ['y'], drop_first = True)

#Excluindo linhas no qual o tempo de ligação é 0
db_pronto = db_pronto.drop([6251,23031,28063, 33015], axis=0)
y = y.drop([6251,23031,28063, 33015], axis = 0)

#Confirmando exclusão
print(db_pronto.loc[db_pronto['duration'] == 0])

print(db_pronto.shape)
print(y.shape)
```

```
Empty DataFrame
Columns: [age, job, marital, education, default, housing, loan, con
tact, month, day_of_week, duration, emp.var.rate, cons.price.idx, c
ons.conf.idx, euribor3m, nr.employed, campaign, pdays, previous, po
utcome]
Index: []
(41184, 20)
(41184, 1)
```

Treinamento do modelo

In [92]:

```
y_array = np.array(y)

#transformando em um vetor 1D
y_array = y_array.reshape(-1)
#y_array = y_array.flatten()

y_array.shape
```

Out[92]:

```
(41184,)
```

In [93]:

```
#Bibliotecas para validação dos modelos
from sklearn.model_selection import cross_validate, cross_val_score
from sklearn.model_selection import KFold
```

1. KNN

In [86]:

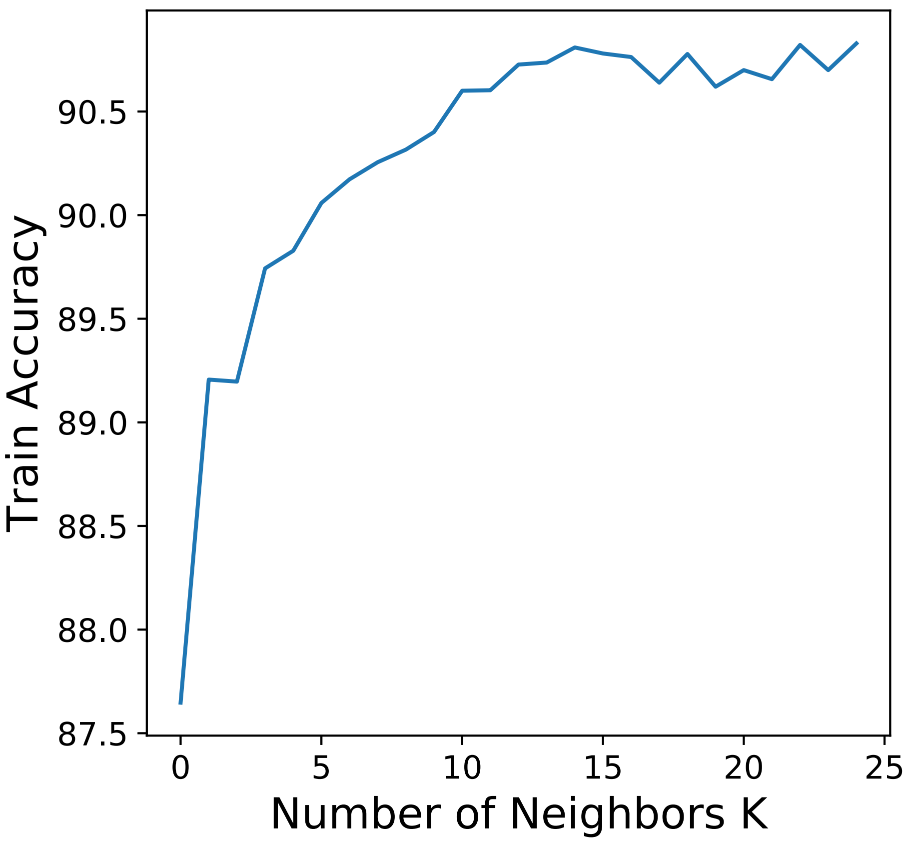
```
from sklearn.neighbors import KNeighborsClassifier as KNN
```

Escolhendo K vizinhos para o modelo

In [96]:

```
neighbors = np.arange(0,25)

#Lista vazia para guardar os resultados
cv_scores = []

#Interação para poder decidir quando vizinhos usar no KNN com 10-fold
for k in neighbors:
    k_value = k+1
    knn = KNN(n_neighbors = k_value, weights='uniform', p=2, metric='euclidean')
    kfold = KFold(n_splits=10)
    scores = cross_val_score(knn, db_pronto, y_array, cv=kfold, scoring='accurac
y')
    cv_scores.append(scores.mean()*100)
    print("k=%d %0.2f " % (k_value, scores.mean()*100))

optimal_k = neighbors[cv_scores.index(max(cv_scores))]
print ("Número ótimo de vizinhos é  %d com %0.1f%%" % (optimal_k, cv_scores[opti
mal_k]))

plt.plot(neighbors, cv_scores)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Train Accuracy')
plt.show()
```

```
k=1 87.65
k=2 89.21
k=3 89.20
k=4 89.74
k=5 89.83
k=6 90.06
k=7 90.17
k=8 90.26
k=9 90.32
k=10 90.40
k=11 90.60
k=12 90.60
k=13 90.73
k=14 90.74
k=15 90.81
k=16 90.78
k=17 90.76
k=18 90.64
k=19 90.78
k=20 90.62
k=21 90.70
k=22 90.66
k=23 90.82
k=24 90.70
k=25 90.83
Número ótimo de vizinhos é  24 com 90.8%
```



In [103]:

```
#Uma vez escolhido k vizinhos vamos validar o modelo com os k vizinhos
knn = KNN(n_neighbors = 24, weights='uniform', p=2, metric='euclidean')
scores = cross_validate(estimator=knn, X=db_pronto, y = y_array, scoring=['accur
acy', 'precision','recall','f1'], cv=kfold)
```

In [112]:

```
#Visualizando as métricas para cada K-fold do treinamento.
scores = pd.DataFrame(scores)
scores.head(10)
#Temos as métricas Acurácias, precisão, recall e f1
```

Out[112]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_prec
0	0.750064	1.804733	0.968439	0.911480	0.413043	0.687556
1	0.749651	1.565197	0.964069	0.912424	0.469388	0.690920
2	0.679790	1.570873	0.960185	0.912856	0.531646	0.692764
3	1.097730	2.037847	0.943433	0.914556	0.604167	0.689994
4	0.936274	2.003972	0.946090	0.913991	0.611465	0.690105
5	0.935774	2.251072	0.951190	0.914207	0.611111	0.690058
6	0.918801	1.924168	0.899223	0.919009	0.518182	0.686221
7	1.113227	2.084138	0.889995	0.919981	0.564516	0.684704
8	1.108644	2.431741	0.856241	0.922085	0.609375	0.690083
9	1.240317	1.600899	0.691112	0.935601	0.688794	0.625434

In [117]:

```
print("Média Acurácia", round(scores['test_accuracy'].mean(), 3))
print("Média Precisão", round(scores['test_precision'].mean(), 3))
print("Média Recall", round(scores['test_recall'].mean(), 3))
print("Média Recall", round(scores['test_f1'].mean(), 3))
```

```
Média Acurácia 0.907
Média Precisão 0.562
Média Recall 0.34
Média Recall 0.411
```

2. Decision Tree

In [119]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [120]:

```
dtree = DecisionTreeClassifier(criterion='gini') #ou Gini
scores_1 = cross_validate(estimator=dtree, X=db_pronto, y = y_array, scoring=['a
ccuracy', 'precision','recall','f1'], cv=kfold)
```

In [123]:

```
#Visualizando as métricas para cada K-fold do treinamento.
scores_1 = pd.DataFrame(scores_1)
scores_1.head(10)
#Temos as métricas Acurácias, precisão, recall e f1
```

Out[123]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_prec
0	0.217129	0.006352	0.952901	1.0	0.264706	1.0
1	0.198115	0.007267	0.949988	1.0	0.303226	1.0
2	0.239363	0.006505	0.941248	1.0	0.345455	1.0
3	0.197003	0.006552	0.924253	1.0	0.431095	1.0
4	0.194602	0.006493	0.936134	1.0	0.489362	1.0
5	0.198114	0.006384	0.927149	1.0	0.325581	1.0
6	0.226866	0.006806	0.857698	1.0	0.326403	1.0
7	0.180223	0.006556	0.857212	1.0	0.379747	1.0
8	0.182637	0.007201	0.832686	1.0	0.467305	1.0
9	0.183003	0.007493	0.604177	1.0	0.581875	1.0

In [129]:

```
print("Média Acurácia", round(scores_1['test_accuracy'].mean(), 3))
print("Média Precisão", round(scores_1['test_precision'].mean(), 3))
print("Média Recall", round(scores_1['test_recall'].mean(), 3))
print("Média Recall", round(scores_1['test_f1'].mean(), 3))
```

```
Média Acurácia 0.878
Média Precisão 0.391
Média Recall 0.407
Média Recall 0.396
```

3. Artificial Neural Networks

In [132]:

```
#Decidir quando hidden layer usar, talvez seja um problema de otimização
#Fazer iterativamente para a escolha de neuronios
from sklearn.neural_network import MLPClassifier
ANN = MLPClassifier(hidden_layer_sizes=(15))
```

In [133]:

```
scores_2 = cross_validate(estimator=ANN, X=db_pronto, y = y_array, scoring=['acc
uracy', 'precision','recall','f1'], cv=kfold)
```

In [128]:

```
#Visualizando as métricas para cada K-fold do treinamento.
scores_2 = pd.DataFrame(scores_2)
scores_2.head(10)
#Temos as métricas Acurácias, precisão, recall e f1
```

Out[128]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_prec
0	3.724794	0.006598	0.967225	0.900850	0.428571	0.630481
1	1.902635	0.008315	0.964797	0.878673	0.000000	0.250000
2	4.414217	0.006742	0.962855	0.900526	0.569536	0.595583
3	0.820253	0.006606	0.937121	0.891704	0.625000	0.705471
4	1.372052	0.006479	0.945847	0.900475	0.680851	0.671273
5	1.475896	0.006512	0.945847	0.889737	0.533333	0.706030
6	0.993322	0.006666	0.893152	0.901149	0.453333	0.622958
7	1.019836	0.006821	0.879553	0.903820	0.497110	0.628534
8	2.509186	0.007027	0.864983	0.909378	0.602222	0.621033
9	1.336846	0.008188	0.607334	0.927508	0.549550	0.510572

In [134]:

```
print("Média Acurácia", round(scores_2['test_accuracy'].mean(), 3))
print("Média Precisão", round(scores_2['test_precision'].mean(), 3))
print("Média Recall", round(scores_2['test_recall'].mean(), 3))
print("Média Recall", round(scores_2['test_f1'].mean(), 3))
```

```
Média Acurácia 0.892
Média Precisão 0.569
Média Recall 0.264
Média Recall 0.299
```

4. Linear Discriminant Analysis (LDA)

In [136]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [137]:

```
LDA = LinearDiscriminantAnalysis()
scores_3 = cross_validate(estimator=LDA, X=db_pronto, y = y_array, scoring=['acc
uracy', 'precision','recall','f1'], cv=kfold)
```

In [140]:

```
#Visualizando as métricas para cada K-fold do treinamento.
scores_3 = pd.DataFrame(scores_3)
scores_3.head(10)
#Temos as métricas Acurácias, precisão, recall e f1
```

Out[140]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precisi
0	0.101903	0.005602	0.969410	0.903062	0.450000	0.635257
1	0.050533	0.005157	0.966011	0.902846	0.530864	0.629866
2	0.048474	0.005148	0.961884	0.903224	0.610390	0.628999
3	0.048830	0.005221	0.940762	0.905194	0.605839	0.626671
4	0.056611	0.007342	0.946819	0.904630	0.663793	0.625296
5	0.049301	0.005157	0.948033	0.905115	0.564706	0.629919
6	0.051483	0.005745	0.894366	0.910106	0.454545	0.629933
7	0.049343	0.005299	0.886595	0.910646	0.535714	0.624071
8	0.049451	0.005845	0.855270	0.914315	0.569892	0.628625
9	0.052393	0.006267	0.661000	0.930448	0.695276	0.539388

In [141]:

```
print("Média Acurácia", round(scores_3['test_accuracy'].mean(), 3))
print("Média Precisão", round(scores_3['test_precision'].mean(), 3))
print("Média Recall", round(scores_3['test_recall'].mean(), 3))
print("Média Recall", round(scores_3['test_f1'].mean(), 3))
```

```
Média Acurácia 0.903
Média Precisão 0.568
Média Recall 0.324
Média Recall 0.404
```

5. Logistic Regression

In [143]:

```
from sklearn.linear_model import LogisticRegression
```

In [145]:

```
logmodel = LogisticRegression()
scores_4 = cross_validate(estimator=logmodel, X=db_pronto, y = y_array, scoring=
['accuracy', 'precision','recall','f1'], cv=kfold)
```

In [148]:

```
#Visualizando as métricas para cada K-fold do treinamento.
scores_4 = pd.DataFrame(scores_4)
scores_4.head(10)
#Temos as métricas Acurácias, precisão, recall e f1
```

Out[148]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precisi
0	0.354411	0.005435	0.971352	0.903278	0.477778	0.669917
1	0.379584	0.005793	0.965768	0.903575	0.533333	0.668620
2	0.385641	0.005458	0.960913	0.904519	0.606557	0.671223
3	0.469140	0.005535	0.936635	0.907028	0.573171	0.674896
4	0.479800	0.005569	0.945119	0.905736	0.659794	0.670896
5	0.349824	0.005504	0.948276	0.904845	0.593750	0.660637
6	0.399228	0.006099	0.897280	0.911671	0.486842	0.676219
7	0.384103	0.005931	0.892424	0.912022	0.573099	0.672642
8	0.361499	0.005982	0.860855	0.914504	0.625418	0.663488
9	0.420548	0.006276	0.629189	0.931366	0.748299	0.585859

In [149]:

```
print("Média Acurácia", round(scores_4['test_accuracy'].mean(), 3))
print("Média Precisão", round(scores_4['test_precision'].mean(), 3))
print("Média Recall", round(scores_4['test_recall'].mean(), 3))
print("Média Recall", round(scores_4['test_f1'].mean(), 3))
```

```
Média Acurácia 0.901
Média Precisão 0.588
Média Recall 0.255
Média Recall 0.348
```

6. Support Vector Machine (SVM)

In [152]:

```
from sklearn.svm import LinearSVC
```

In [153]:

```
SVM = LinearSVC()
scores_5 = cross_validate(estimator=SVM, X=db_pronto, y = y_array, scoring=['acc
uracy', 'precision','recall','f1'], cv=kfold)
```

In [156]:

```
#Visualizando as métricas para cada K-fold do treinamento.
scores_5 = pd.DataFrame(scores_5)
scores_5.head(10)
#Temos as métricas Acurácias, precisão, recall e f1
```

Out[156]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precis
0	9.460046	0.009249	0.973294	0.886497	0.555556	0.714094
1	12.390656	0.007202	0.965526	0.902361	0.511111	0.618442
2	11.613157	0.005831	0.926681	0.801565	0.357143	0.369759
3	11.460008	0.008619	0.936150	0.895859	0.678571	0.742120
4	11.745255	0.006043	0.941476	0.900610	0.673913	0.682316
5	10.252776	0.005210	0.945847	0.894162	0.538462	0.763383
6	9.933803	0.006342	0.900437	0.891437	0.714286	0.744417
7	10.591744	0.006104	0.883681	0.893676	0.769231	0.772487
8	11.224879	0.008153	0.847256	0.910835	0.531148	0.652870
9	10.142964	0.007201	0.540554	0.926644	0.500000	0.623932

In [157]:

```
print("Média Acurácia", round(scores_5['test_accuracy'].mean(), 3))
print("Média Precisão", round(scores_5['test_precision'].mean(), 3))
print("Média Recall", round(scores_5['test_recall'].mean(), 3))
print("Média Recall", round(scores_5['test_f1'].mean(), 3))
```

```
Média Acurácia 0.886
Média Precisão 0.583
Média Recall 0.216
Média Recall 0.222
```

Resultados e Metricas

Discussões