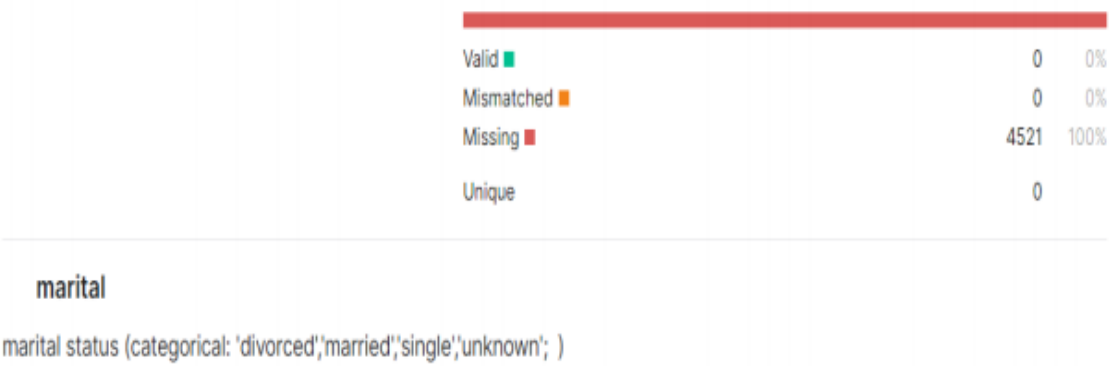
1. 1-Visualisation base de données :

1. Question :

1- Donner un tableau de bord des attributs par exemple l'attribut marital est représenté de la manière suivante :



2. Exemple les valeurs numériques :

Age

```
print('Missing: %d' %iris_df["age"].isnull().sum())
# seed the random number generator
seed(1)
# generate univariate observations
data = iris_df.iloc[:,0]
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers(Mismatched): %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations(valid): %d' % len(outliers_removed))
print('Unique: %d' %len(iris_df["age"].unique()))
```

```
Missing: 6
Percentiles: 25th=nan, 75th=nan, IQR=nan
Identified outliers(Mismatched): 0
Non-outlier observations(valid): 0
Unique: 68
```

3. Exemple les valeurs catégorielles :

education

```
]: iris_df.head()
   print('Missing: %d' %iris_df["education"].isnull().sum())
   print('Mismatched: 0')
   print('valid: %d'% len(iris_df["education"]))
   print('Unique: %d' %len(iris_df["education"].unique()))
```

```
Missing: 7
Mismatched: 0
valid: 4521
Unique: 5
```

On afficher les outliers , les Missing value …etc pour les valeurs numériques et catégorielles .

2. Remplacer les valeurs manquantes :

1. Question :

- L'attribut Age par la moyenne associée aux classes d'appartenances
- L'attribut marital par la valeur marital la plus fréquente associé aux classes d'appartenances

2. Remplacer les valeurs manquantes pour la variable age :

```
: iris_df.corr()
```

:

	age	day	duration	campaign	previous
age	1.000000	-0.017770	-0.002269	-0.005344	-0.003429
day	-0.017770	1.000000	-0.024629	0.160706	-0.059114
duration	-0.002269	-0.024629	1.000000	-0.068382	0.018080
campaign	-0.005344	0.160706	-0.068382	1.000000	-0.067833
previous	-0.003429	-0.059114	0.018080	-0.067833	1.000000

En utilisant la variable job pour classifier, et avec l'application de ces algorithmes qui permit de remplacer les valeur manquantes avec la moyenne de chaque classe :

```
# seed the random number generator
seed(1)
# generate univariate observations
data = iris_df.iloc[:,0]
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

```
Percentiles: 25th=nan, 75th=nan, IQR=nan
Identified outliers: 0
Non-outlier observations: 0
```

```
#iris_df['fixed acidity'][iris_df['fixed acidity'] in outliers]=np.nan
c=0
for i in iris_df.iloc[:,0]:
    c=c+1
    if i in outliers:
        iris_df.iloc[:,0][c-1]=np.nan
        """
        iris_df.iloc[:,1]=np.nan
        """
```

```
iris_df["job"].unique()
```

```
array(['unemployed', 'services', 'management', 'blue-collar',
       'self-employed', 'technician', 'entrepreneur', 'admin.', 'student',
       'housemaid', 'retired', 'unknown'], dtype=object)
```

```
percent_missing = iris_df.isnull().sum() * 100 / len(iris_df)

for i in range(len(percent_missing)):
    if percent_missing[i]!=0 and iris_df.iloc["job"]=="unemployed":
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='services':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='management':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='blue-collar':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='self-employed':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='technician':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='entrepreneur':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='admin.':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='student':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='housemaid':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='retired':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
    if percent_missing[i]!=0 and iris_df.iloc["job"]=='unknown':
            iris_df[iris_df.columns[i]].fillna(iris_df[iris_df.columns[i]].mean()[0] ,inplace=True)
```

3. Corriger le problème des valeurs Mismatched :

1. Question :

3- Corriger le problème des valeurs Mismatched on utilise les fonctions de gestion des chaines de caractères.

2. Corriger le problème des valeurs Mismatched on utilise les fonctions de gestion des chaines de caractères :

Q3

```
# convert to lower case
iris_df['job'] = iris_df['job'].str.lower()
# remove trailing white spaces
iris_df['job'] = iris_df['job'].str.strip()
```

```
# get all the unique values in the 'Country' column
job = iris_df['job'].unique()

# sort them alphabetically and then take a closer look
job.sort()
job
```

```
array(['admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management',
       'retired', 'self-employed', 'services', 'student', 'technician',
       'unemployed', 'unknown'], dtype=object)
```

```
# function to replace rows in the provided column of the provided dataframe
# that match the provided string above the provided ratio with the provided string
def replace_matches_in_column(df, column, string_to_match, min_ratio = 74):
    # get a list of unique strings
    strings = df[column].unique()

    # get the top 10 closest matches to our input string
    matches = fuzzywuzzy.process.extract(string_to_match, strings,
                                         limit=10, scorer=fuzzywuzzy.fuzz.token_sort_ratio)

    # only get matches with a ratio > 90
    close_matches = [matches[0] for matches in matches if matches[1] >= min_ratio]

    # get the rows of all the close matches in our dataframe
    rows_with_matches = df[column].isin(close_matches)

    # replace all rows with close matches with the input matches
    df.loc[rows_with_matches, column] = string_to_match

    # let us know the function's done
    print("All done!")
```

```
# use the function we just wrote to replace close matches to "south korea" with "south korea"
replace_matches_in_column(df=iris_df, column='job', string_to_match="unemployed")
```

```
All done!
```

aprés en continue pour les other variables

4. Donner un scénario pratique de l'ingénierie des attributs a fin de prédire le Y :

En utilise la fonction pd.get_dummies() pour coder les variables catégorial , et les deux fonction

scaler = StandardScaler().fit(X_train) qui normalise les donner ,

scaler = MinMaxScaler().fit(X_train) qui rende la dataset entre 0 et 1 .

Q4

```
]: iris_df['job']=pd.get_dummies(iris_df['job'])
   iris_df['marital']=pd.get_dummies(iris_df['marital'])
   iris_df['education']=pd.get_dummies(iris_df['education'])
   iris_df['default']=pd.get_dummies(iris_df['default'])
   iris_df['housing']=pd.get_dummies(iris_df['housing'])
   iris_df['contact']=pd.get_dummies(iris_df['contact'])
   iris_df['month']=pd.get_dummies(iris_df['month'])
   iris_df['loan']=pd.get_dummies(iris_df['loan'])
   iris_df['poutcome']=pd.get_dummies(iris_df['poutcome'])
   iris_df['y']=pd.get_dummies(iris_df['y'])
```

```
]: iris_df.head(100)
```

```
1.
```

```
iris_df.head(100)
```

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	previous	poutcome	y
0	30.0	0	1	0	0	0	1	0	19	1	79	1	0	1	0
1	33.0	0	1	1	0	1	0	0	11	1	220	1	4	0	0
2	35.0	0	1	1	0	1	1	0	16	0	185	1	1	0	0
3	30.0	0	1	1	0	1	0	1	3	1	199	4	0	1	0
4	59.0	0	1	1	0	1	1	1	5	1	226	1	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	41.0	0	1	1	0	0	1	0	1	1	291	2	0	1	0
96	41.0	1	1	1	0	1	1	1	14	1	149	2	0	1	0
97	30.0	1	1	1	0	0	1	0	19	1	233	1	20	0	0
98	36.0	0	1	1	0	0	1	0	12	1	473	1	1	1	0
99	31.0	0	1	0	0	0	1	0	4	1	736	1	0	1	1

Normlisation de dataset

In [48]:
```
X_data = iris_df
y_data = iris_df['y']
```

In [49]:
```
X_train,X_test,y_train,y_test = train_test_split(X_data,y_data,test_size=0.3, stratify=y_data)
```

In [50]:
```
X_train,X_val,y_train,y_val = train_test_split(X_train,y_train,test_size=0.3, stratify=y_train)
```

In [53]:
```
scaler = StandardScaler().fit(X_train)
#scaler = MinMaxScaler().fit(X_train)
```

In [54]:
```
a=scaler.transform(X_train)
b=scaler.transform(X_test)
c=scaler.transform(X_val)
```

In [ ]:
```
X_train=a
X_test =b
X_val  =c
```