In [308]:
```
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
```

Inital data processing

In [309]:
```
data=pd.read_csv("C://Users//sthakal//OneDrive - George Weston Limited-6469347-MTCAD//Psnal/
```

In [310]:
```
data.head(15)
```

Out[310]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	ca
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	
5	45	services	married	basic.9y	unknown	no	no	telephone	may	mon	...	
6	59	admin.	married	professional.course	no	no	no	telephone	may	mon	...	
7	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon	...	
8	24	technician	single	professional.course	no	yes	no	telephone	may	mon	...	
9	25	services	single	high.school	no	yes	no	telephone	may	mon	...	
10	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon	...	
11	25	services	single	high.school	no	yes	no	telephone	may	mon	...	
12	29	blue-collar	single	high.school	no	no	yes	telephone	may	mon	...	
13	57	housemaid	divorced	basic.4y	no	yes	no	telephone	may	mon	...	
14	35	blue-collar	married	basic.6y	no	yes	no	telephone	may	mon	...	

15 rows × 21 columns

In [311]:
```
data.dtypes
```

Out[311]:
```
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 [312]:
```
data.isna().count()
# CHECKING missing values
```

Out[312]:
```
age               41188
job               41188
marital           41188
education         41188
default           41188
housing           41188
loan              41188
contact           41188
month             41188
day_of_week       41188
duration          41188
campaign          41188
pdays             41188
previous          41188
poutcome          41188
emp.var.rate      41188
cons.price.idx    41188
cons.conf.idx     41188
euribor3m         41188
nr.employed       41188
y                 41188
dtype: int64
```

In [313]:
```
data.shape
```

Out[313]:
```
(41188, 21)
```

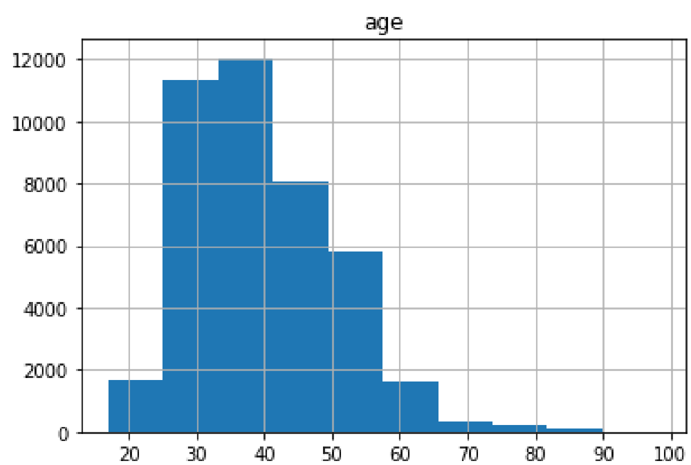
In [314]:
```
data.describe()
#Checking the stats of numerical variable
```

Out[314]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.c
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.

In [315]:
```
data.hist('age')
```

Out[315]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000267320009E8>]],
      dtype=object)
```



Checking the counts of Categorical variables

In [316]:
```
#we are now seeing the counts of the categorical variable one by one to see the values
#Checking the counts of categorical variables to see what value they are concentrated in

job_cnt=data['job'].value_counts()
print(job_cnt)
```

```
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64
```

In [317]:
```
data['marital'].value_counts()
```

Out[317]:
```
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64
```

In [318]:
```
data['education'].value_counts()
```

Out[318]:
```
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
```

In [319]:
```
data['marital'].value_counts()
```

Out[319]:
```
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64
```

In [320]:
```
data['education'].value_counts()
```

Out[320]:
```
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
```

In [321]:
```
data['housing'].value_counts()
```

Out[321]:
```
yes        21576
no         18622
unknown      990
Name: housing, dtype: int64
```

In [322]:
```
data['loan'].value_counts()
```

Out[322]:
```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```

In [323]:
```
data['contact'].value_counts()
```

Out[323]:
```
cellular     26144
telephone    15044
Name: contact, dtype: int64
```

In [324]:
```
data['month'].value_counts()
```

Out[324]:
```
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182
Name: month, dtype: int64
```

In [325]:
```
data['day_of_week'].value_counts()
```

Out[325]:
```
thu    8623
mon    8514
wed    8134
tue    8090
fri    7827
Name: day_of_week, dtype: int64
```

In [326]:
```
data['poutcome'].value_counts()
```

Out[326]:
```
nonexistent    35563
failure         4252
success         1373
Name: poutcome, dtype: int64
```

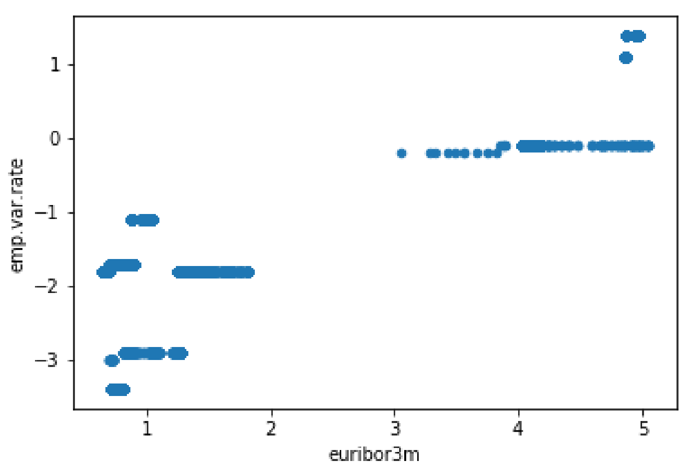
In [327]:
```
data['month'].value_counts()
```
Out[327]:
```
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182
Name: month, dtype: int64
```

In [328]:
```
data['y'].value_counts()
#understanding the count of the Predictor variable
```
Out[328]:
```
no     36548
yes     4640
Name: y, dtype: int64
```

In [329]:
```
data['default'].value_counts()
```
Out[329]:
```
no         32588
unknown     8597
yes            3
Name: default, dtype: int64
```

Bivarate Analysis

In [330]:
```
data.plot.scatter('euribor3m', 'emp.var.rate',)
```
Out[330]: <matplotlib.axes._subplots.AxesSubplot at 0x26733590048>



In [331]:
```
data.corr()
#Checking numerical correlation
```

Out[331]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx
age	1.000000	-0.000866	0.004594	-0.034369	0.024365	-0.000371	0.000857	0.129372
duration	-0.000866	1.000000	-0.071699	-0.047577	0.020640	-0.027968	0.005312	-0.008173
campaign	0.004594	-0.071699	1.000000	0.052584	-0.079141	0.150754	0.127836	-0.013733
pdays	-0.034369	-0.047577	0.052584	1.000000	-0.587514	0.271004	0.078889	-0.091342
previous	0.024365	0.020640	-0.079141	-0.587514	1.000000	-0.420489	-0.203130	-0.050936
emp.var.rate	-0.000371	-0.027968	0.150754	0.271004	-0.420489	1.000000	0.775334	0.196041
cons.price.idx	0.000857	0.005312	0.127836	0.078889	-0.203130	0.775334	1.000000	0.058986
cons.conf.idx	0.129372	-0.008173	-0.013733	-0.091342	-0.050936	0.196041	0.058986	1.000000
euribor3m	0.010767	-0.032897	0.135133	0.296899	-0.454494	0.972245	0.688230	0.277686
nr.employed	-0.017725	-0.044703	0.144095	0.372605	-0.501333	0.906970	0.522034	0.100513

Some preprocessing for Bivarate

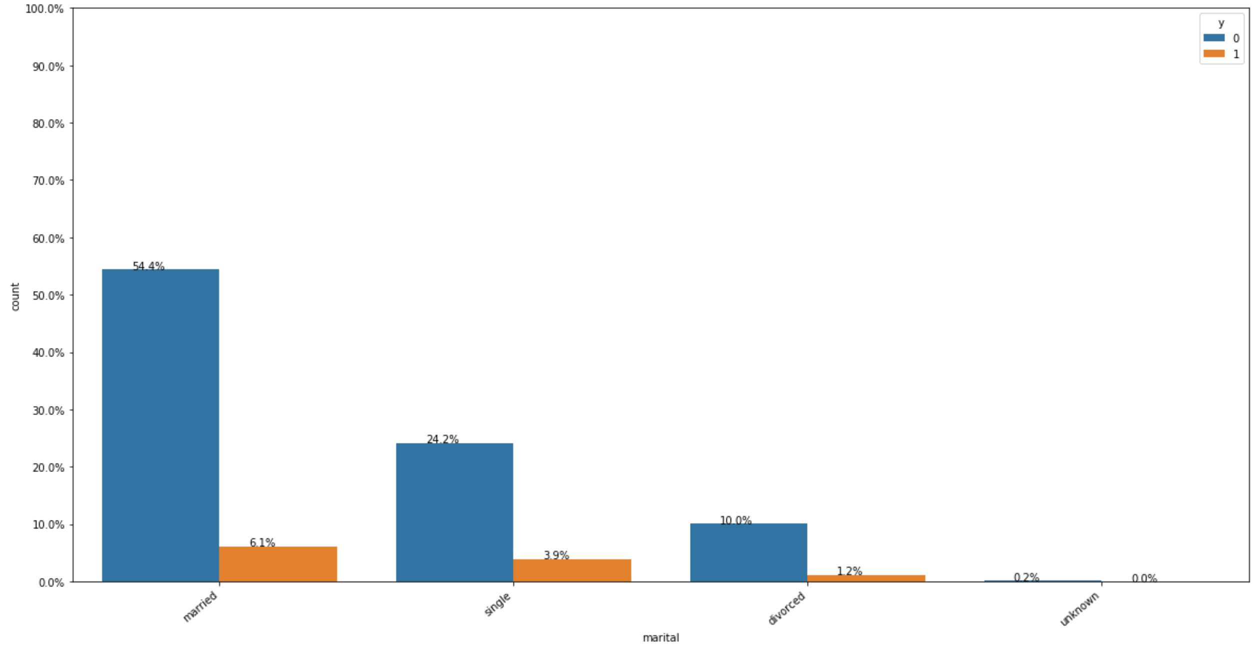
In [332]:
```
#Clubbing all basic eduction in one
data['education']=np.where(data['education'] =='basic.9y', 'Basic', data['education'])
data['education']=np.where(data['education'] =='basic.6y', 'Basic', data['education'])
data['education']=np.where(data['education'] =='basic.4y', 'Basic', data['education'])
```

In [333]:
```
data['y']=np.where(data['y'] =='yes',1, data['y'])# changing into binary/
data['y']=np.where(data['y'] =='no',0, data['y'])
```

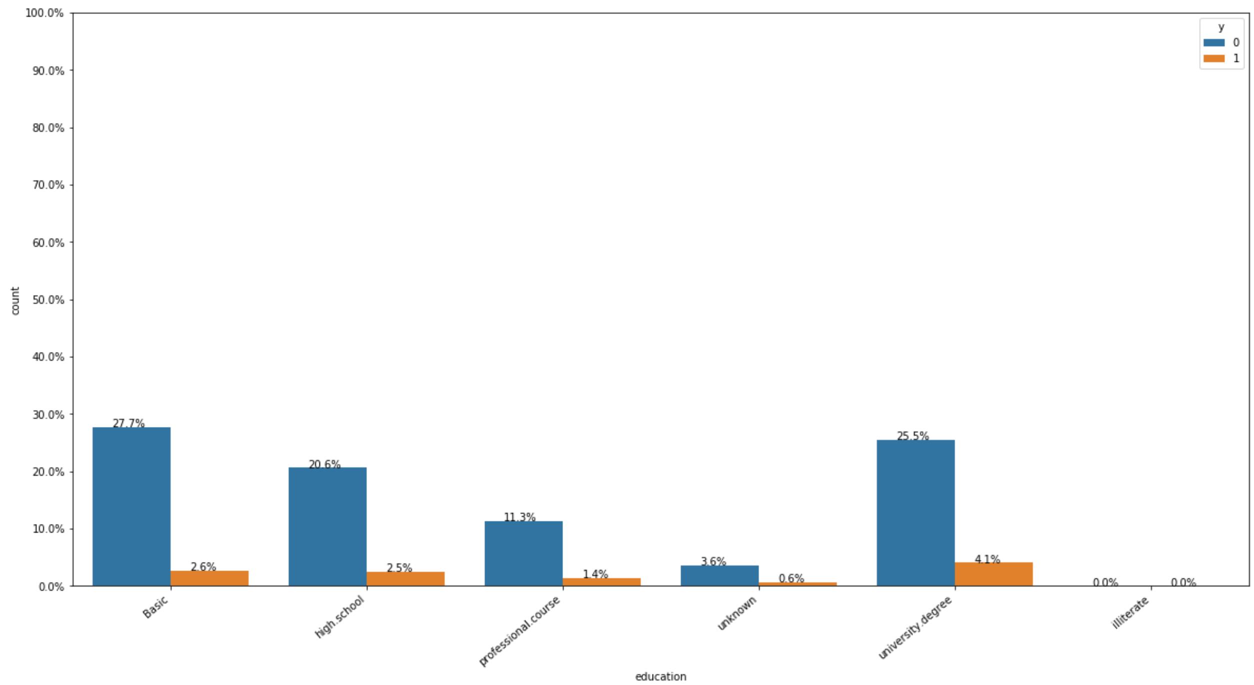
In [334]:
```
#Checking how the categorical variable is related to the Response variable
%matplotlib inline

def countplot_withY(label, dataset):
  plt.figure(figsize=(20,10))
  Y = data[label]
  total = len(Y)*1.
  ax=sns.countplot(x=label, data=dataset, hue="y")
  for p in ax.patches:
    ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5

  #put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
  ax.yaxis.set_ticks(np.linspace(0, total, 11))
  #adjust the ticklabel to the desired format, without changing the position of the ticks.
  ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
  ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
  # ax.legend(labels=["no","yes"])
  plt.show()
```

Seeing the relationship of categorical variables with Response Variable

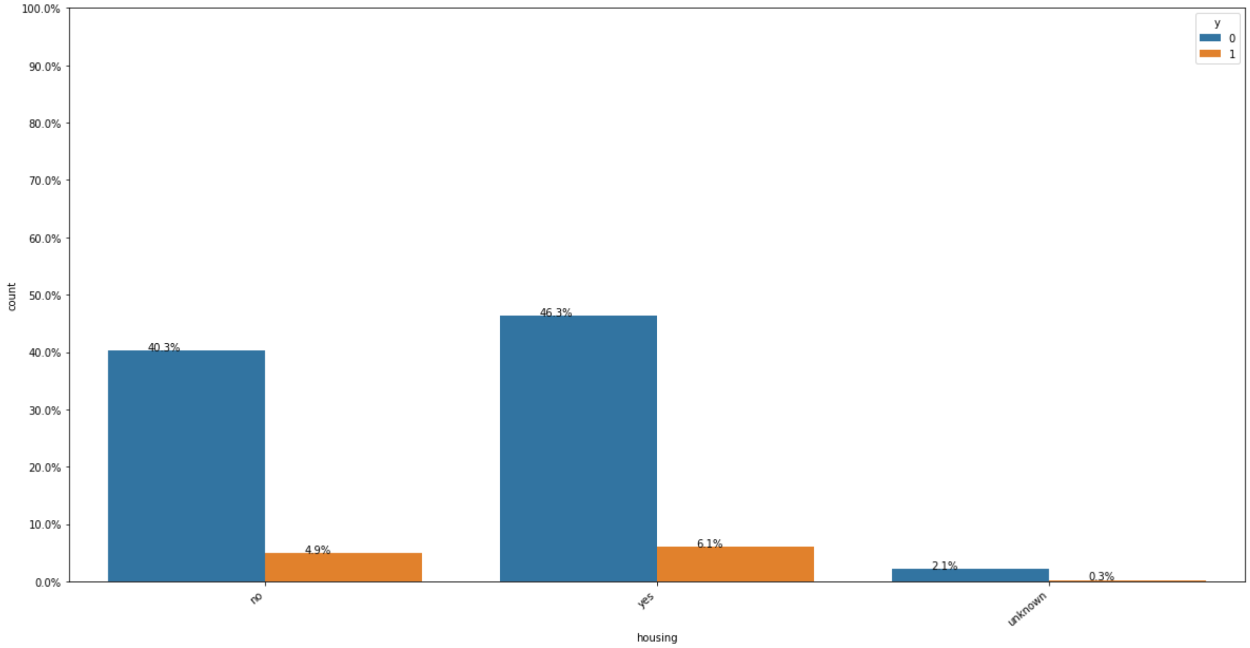
In [335]:
```
countplot_withY("marital",data)
```



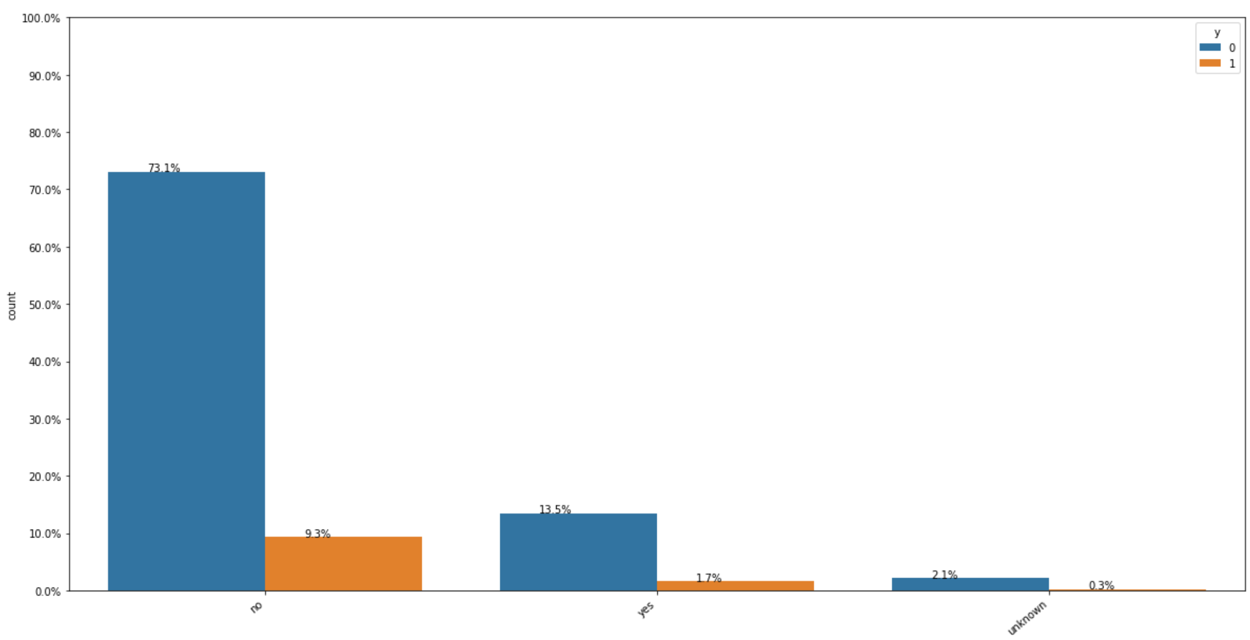
In [336]:
```
countplot_withY("education",data)
```



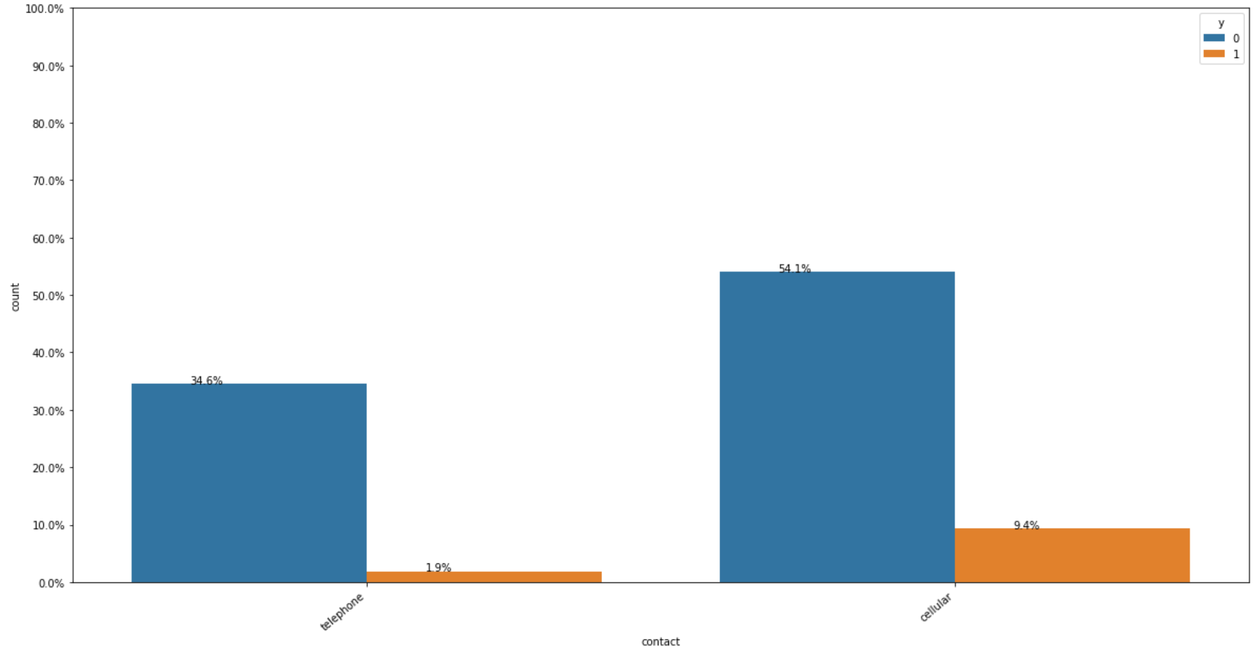
In [337]:
```
countplot_withY("housing",data)
```



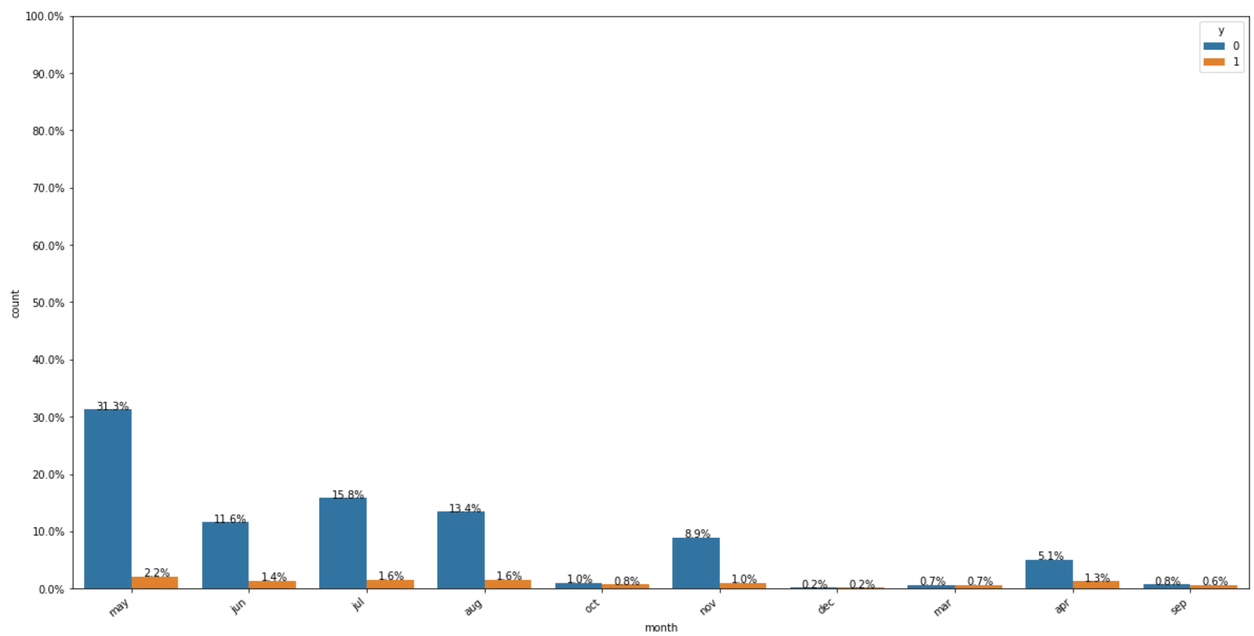
In [338]:
```
countplot_withY("loan",data)
```



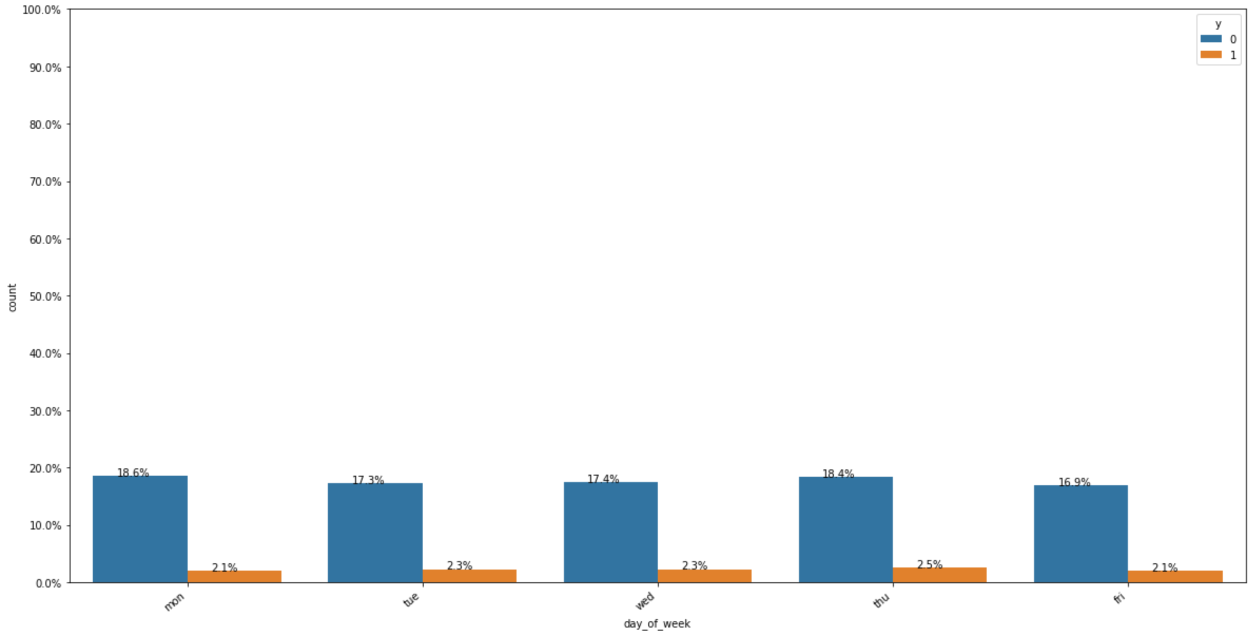
In [339]:
```
countplot_withY("contact",data)
```



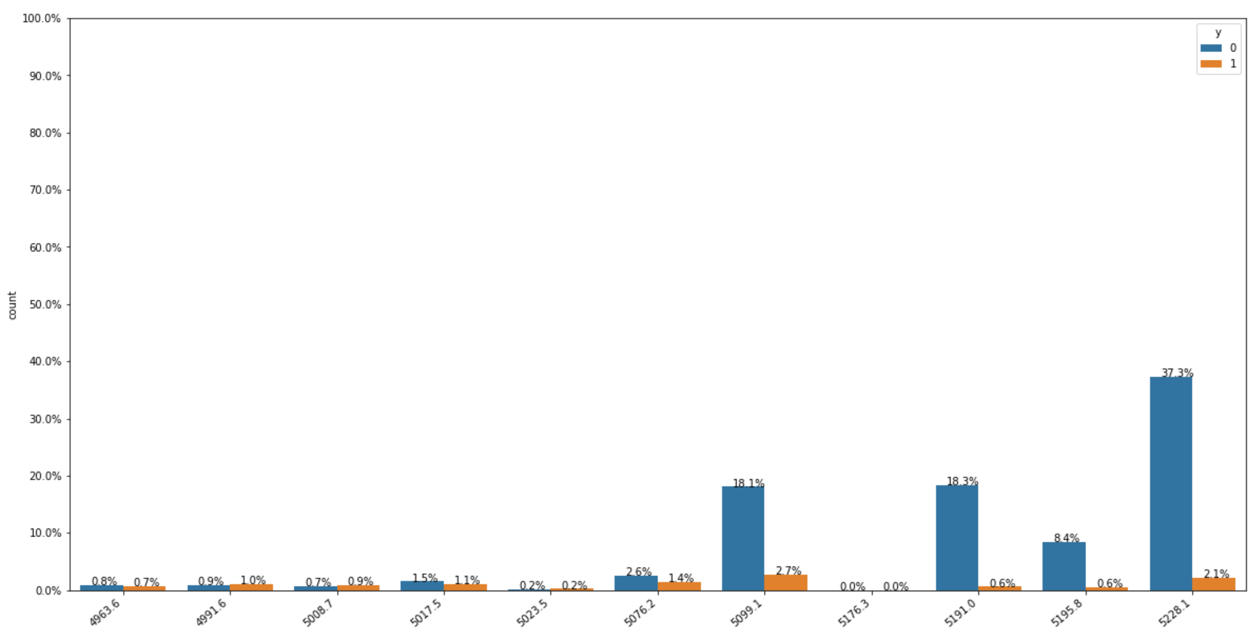
In [340]:
```
countplot_withY("month",data)
```



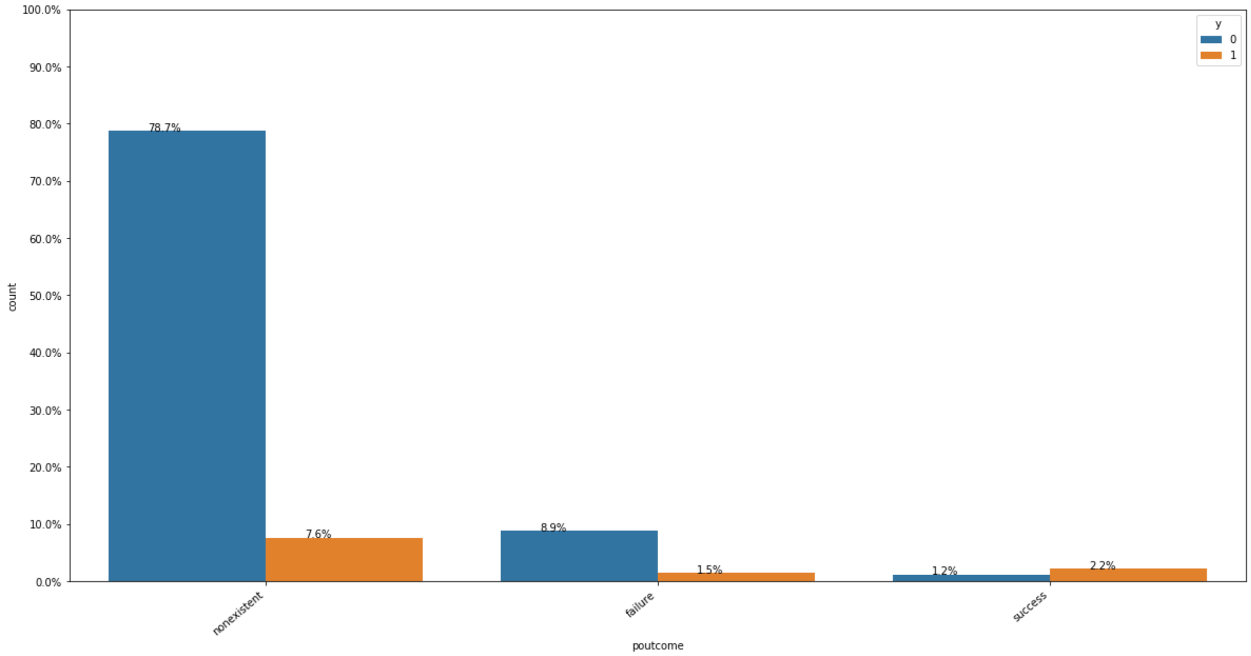
In [341]:
```
countplot_withY("day_of_week",data)
```



In [342]:
```
countplot_withY("nr.employed",data)
```



In [343]:
```
countplot_withY("poutcome",data)
```



In [344]:
```
data_response_yes=data[data['y']==1]
#only selecting those dataset where response is yes
```

In [345]:
```
data_response_yes.describe()
#Just checking how the numerical variable of success looks like
```

Out[345]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.
count	4640.000000	4640.000000	4640.000000	4640.000000	4640.000000	4640.000000	4640.000000	4640.000(
mean	40.913147	553.191164	2.051724	792.035560	0.492672	-1.233448	93.354386	-39.7897
std	13.837476	401.171871	1.666245	403.407181	0.860344	1.623626	0.676644	6.139(
min	17.000000	37.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800(
25%	31.000000	253.000000	1.000000	999.000000	0.000000	-1.800000	92.893000	-46.200(
50%	37.000000	449.000000	2.000000	999.000000	0.000000	-1.800000	93.200000	-40.400(
75%	50.000000	741.250000	2.000000	999.000000	1.000000	-0.100000	93.918000	-36.100(
max	98.000000	4199.000000	23.000000	999.000000	6.000000	1.400000	94.767000	-26.900(

Some preprocessing for model building

In [346]:
```
#Create dummy variables
data_x = data.iloc[:, :-1]
print("Shape of X:", data_x.shape)
data_y = data["y"]
print("Shape of Y:", data_y.shape)
```

```
Shape of X: (41188, 20)
Shape of Y: (41188,)
```

In [347]:
```
from sklearn.model_selection import train_test_split

X_rest, X_test, y_rest, y_test = train_test_split(data_x, data_y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_rest, y_rest, test_size=0.2)

print("X Train:", X_train.shape)
print("X CV:", X_cv.shape)
print("X Test:", X_test.shape)
print("Y Train:", y_train.shape)
print("Y CV:", y_cv.shape)
print("Y Test:", y_test.shape)
```

```
X Train: (26360, 20)
X CV: (6590, 20)
X Test: (8238, 20)
Y Train: (26360,)
Y CV: (6590,)
Y Test: (8238,)
```

In [348]:
```
y_train.replace({"no":0, "yes":1}, inplace=True)
y_cv.replace({"no":0, "yes":1}, inplace=True)
y_test.replace({"no":0, "yes":1}, inplace=True)
```

In [349]:
```
# Categorical boolean mask
categorical_feature_mask = data_x.dtypes==object

# filter categorical columns using mask and turn it into a list
categorical_cols = data_x.columns[categorical_feature_mask].tolist()
```

In [350]:
```
data_x.dtypes==object
#checking object type
```

Out[350]:
```
age               False
job                True
marital            True
education          True
default            True
housing            True
loan               True
contact            True
month              True
day_of_week        True
duration          False
campaign          False
pdays             False
previous          False
poutcome           True
emp.var.rate      False
cons.price.idx    False
cons.conf.idx     False
euribor3m         False
nr.employed       False
dtype: bool
```

In [351]:
```
from sklearn.feature_extraction.text import CountVectorizer

def add_onehot_to_dataframe(sparse, df, vectorizer, name):
  '''
      This function will add the one hot encoded to the dataframe.

  '''
  for i, col in enumerate(vectorizer.get_feature_names()):
    colname = name+"_"+col
    # df[colname] = pd.SparseSeries(sparse[:, i].toarray().flatten(), fill_value=0)
    df[colname] = sparse[:, i].toarray().ravel().tolist()

  return df

def OneHotEncoder(categorical_cols, X_train, X_test, X_cv=None, include_cv=False):
  '''
    This function takes categorical column names as inputs. The objective
    of this function is to take the column names iteratively and encode the
    features using One hot Encoding mechanism and also adding the encoded feature
    to the respective dataframe.

    The include_cv parameter indicates whether we should include CV dataset or not.
    This is added specifically because when using GridSearchCV or RandomizedSearchCV,
    we only split the dataset into train and test to give more data to training purposes.
    This is done because GridSearchCV splits the data internally anyway.
  '''

  for i in categorical_cols:
    Vectorizer = CountVectorizer(token_pattern="[A-Za-z0-9-.]+")
    print("Encoding for feature: ", i)
    # Encoding training dataset
    temp_cols = Vectorizer.fit_transform(X_train[i])
    X_train = add_onehot_to_dataframe(temp_cols, X_train, Vectorizer, i)

    # Encoding Cross validation dataset
    if include_cv:
      temp_cols = Vectorizer.transform(X_cv[i])
      X_cv = add_onehot_to_dataframe(temp_cols, X_cv, Vectorizer, i)

    # Encoding Test dataset
    temp_cols = Vectorizer.transform(X_test[i])
    X_test = add_onehot_to_dataframe(temp_cols, X_test, Vectorizer, i)
```

In [352]:
```
OneHotEncoder(categorical_cols, X_train, X_test, X_cv, True)

# Drop the categorical features as the one hot encoded representation is present
X_train = X_train.drop(categorical_cols, axis=1)
X_cv = X_cv.drop(categorical_cols, axis=1)
X_test = X_test.drop(categorical_cols, axis=1)

print("Shape of train: ", X_train.shape)
print("Shape of CV: ", X_cv.shape)
print("Shape of test: ", X_test.shape)
```

```
Encoding for feature:  job

C:\Users\sthakal\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_launcher.
py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexi
ng.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.
html#indexing-view-versus-copy)
  # This is added back by InteractiveShellApp.init_path()

Encoding for feature:  marital
Encoding for feature:  education
Encoding for feature:  default
Encoding for feature:  housing
Encoding for feature:  loan
Encoding for feature:  contact
Encoding for feature:  month
Encoding for feature:  day_of_week
```

In [356]:
```
# with "duration" column
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(class_weight='balanced',max_iter=500)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)

print("AUC score with duration column: ", roc_auc_score(y_test, y_pred[:,1]))
```

```
AUC score with duration column:  0.7902335490207407
```

In [354]:
```
# Removing duration feature

# From Train
X_train = X_train.drop("duration", axis=1)
print("The shape of the train dataset: ", X_train.shape)

# From CV
X_cv = X_cv.drop("duration", axis=1)
print("The shape of the cv dataset: ", X_cv.shape)

# From Test
X_test = X_test.drop("duration", axis=1)
print("The shape of the test dataset: ", X_test.shape)
```

```
The shape of the train dataset:  (26360, 60)
The shape of the cv dataset:  (6590, 60)
The shape of the test dataset:  (8238, 60)
```

In [357]:
```
# without "duration" column

model = LogisticRegression(class_weight='balanced',max_iter=500)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)

print("AUC score without duration column: ", roc_auc_score(y_test, y_pred[:,1]))
```

```
AUC score without duration column:  0.7902335490207407
```

In [358]:
```
# without "duration" column
# X_train = X_train.drop("duration", axis=1)
# X_test = X_test.drop("duration", axis=1)

# print(X_train.shape)
# print(X_test.shape)

model = LogisticRegression(class_weight='balanced',max_iter=500)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)

print("AUC score without duration column: ", roc_auc_score(y_test, y_pred[:,1]))
```

```
AUC score without duration column:  0.7889597897047779
```