importing neccessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Load the dataset
data=pd.read_csv("C:/Users/raksh/Desktop/new_train.csv")
data
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	da
0	49	blue-collar	married	basic.9y	unknown	no	no	cellular	nov	
1	37	entrepreneur	married	university.degree	no	no	no	telephone	nov	
2	78	retired	married	basic.4y	no	no	no	cellular	jul	
3	36	admin.	married	university.degree	no	yes	no	telephone	may	
4	59	retired	divorced	university.degree	no	no	no	cellular	jun	
..	...	...	...	...	...	...	...	...	...	
5	28	services	single	high.school	no	yes	no	cellular	jul	
6	52	technician	married	professional.course	no	yes	no	cellular	nov	
7	54	admin.	married	basic.9y	no	no	yes	cellular	jul	
8	29	admin.	married	university.degree	no	no	no	telephone	may	
9	35	admin.	married	university.degree	no	no	yes	telephone	jun	

0 rows × 16 columns

In [3]:
```
# check shape of dataset
print("shape of the data:", data.shape)
data.head()
```

```
shape of the data: (32950, 16)
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	da
0	49	blue-collar	married	basic.9y	unknown	no	no	cellular	nov	
1	37	entrepreneur	married	university.degree	no	no	no	telephone	nov	
2	78	retired	married	basic.4y	no	no	no	cellular	jul	
3	36	admin.	married	university.degree	no	yes	no	telephone	may	
4	59	retired	divorced	university.degree	no	no	no	cellular	jun	

In [4]:
```
# check data types of all columns
data.dtypes
```

Out[4]:
```
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
y                object
dtype: object
```

checking missing data

Handling missing data is the one of the important step in data preprocessing. Missing data means absence of observations in columns due to lack of information or incomplete results . Feeding missing data to your machine learning model could lead to wrong prediction or classification. so we find missing values here.

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
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
y              0
dtype: int64
```

check for class imbalance

By using the target variable y and class count we will find the how many number of no/yes[catogerical]in dataset

In [6]:
```
data["y"].value_counts()

#so here the data completely imbalanced were the No type is of 29238 and ye
```

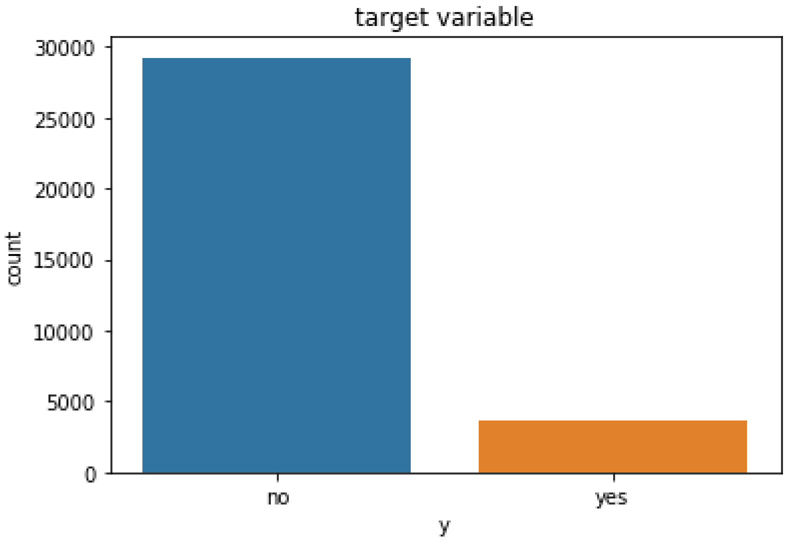
Out[6]:
```
no     29238
yes     3712
Name: y, dtype: int64
```

countplot

It is used to represent the counts of the observation present in the categoriacal data which is target variable 'y' containing yes/no type

In [7]:
```
sns.countplot(data["y"])
plt.title("target variable")
```

Out[7]:
```
Text(0.5, 1.0, 'target variable')
```



In [8]:
```
# percentage of class present in target variable(y)

print("percentage of NO and YES\n",data["y"].value_counts()/len(data)*100)
```

```
percentage of NO and YES
 no     88.734446
yes    11.265554
Name: y, dtype: float64
```

In [9]:
```
#As we see here the data is completely imbalenced here
#The class distribution in the target variable is ~89:11 indicating an imba
```

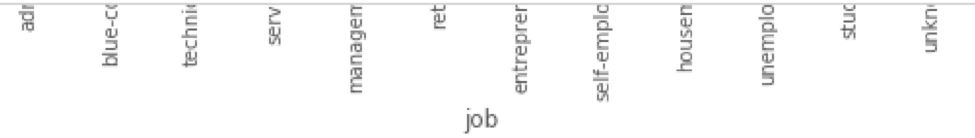
Exploratory data analysis

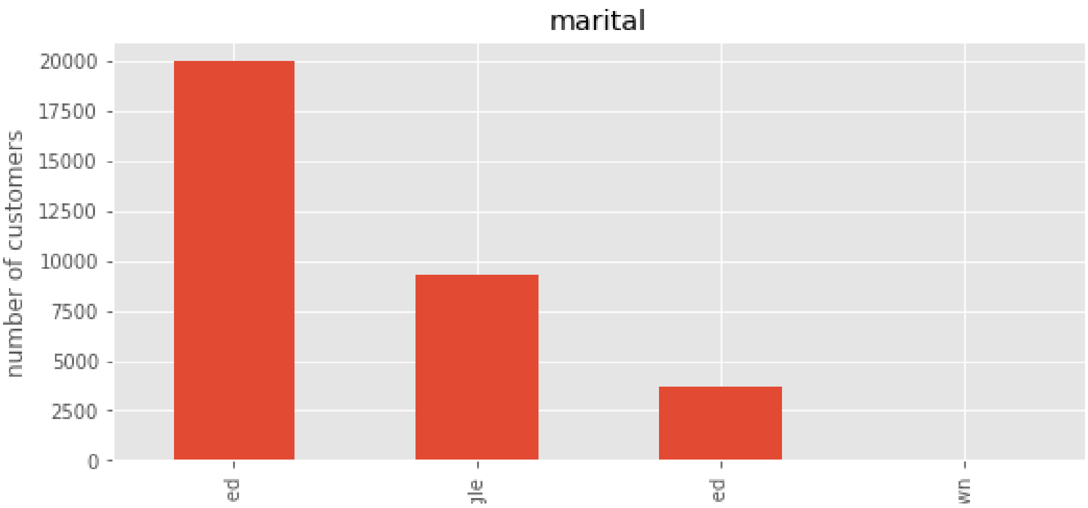
Univariate analyisis of catogerical variables

In [10]:
```
#Identify the catogerical variables
#Displaying the column attributes with dtype as object


cato_var=data.select_dtypes(include=["object"]).columns
print(cato_var)
```

```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'conta
ct',
       'month', 'day_of_week', 'poutcome', 'y'],
      dtype='object')
```

```
In [11]: # plotting bar chart for each categorical variable
plt.style.use("ggplot")

for column in cato_var:
    plt.figure(figsize=(20,4))
    plt.subplot(121)
    data[column].value_counts().plot(kind="bar")
    plt.xlabel(column)
    plt.ylabel("number of customers")
    plt.title(column)
```





Inferences

1.. As comparing the number of customers and job ,we have top three professions belong to are administration, blue-collar jobs and technicians.

2.In the matital status graph the more number of customers were married here.

3.Majority of the customers do not have a credit in default.

4.Many of our past customers have applied for a housing loan but very few have applied for personal loans.

5.As compare to Telephones ,Cell-phones seem to be the most favoured method of reaching out to customers.

6.In May month many of the customers are contacted here.

The plot for the target variable shows heavy imbalance in the target variable.

Here in some columns missing values are represented as unknown which represents the missing data

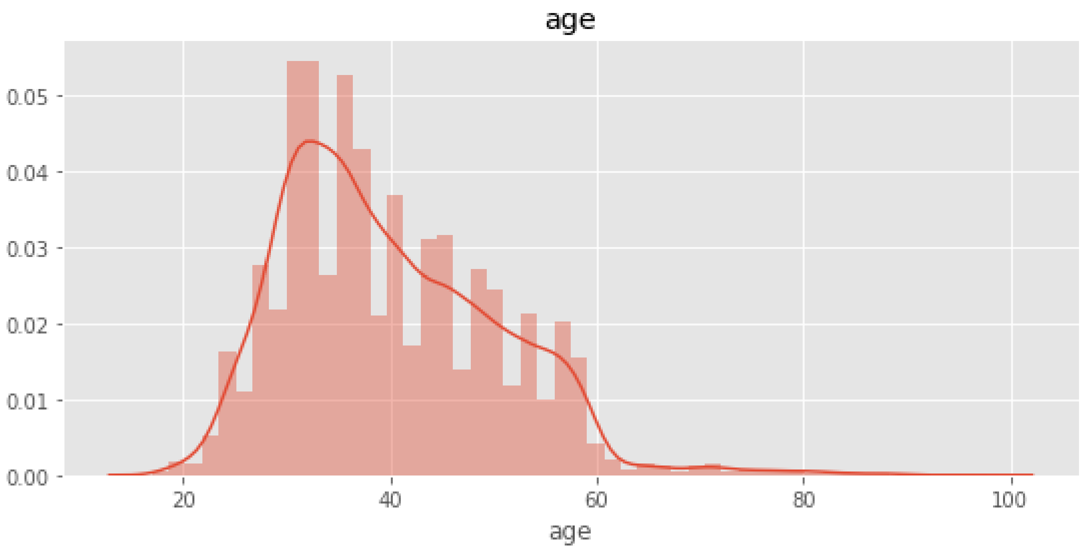
In [12]:
```
# replacing "unknown" with the mode
for column in cato_var:
    mode= data[column].mode()[0]
    data[column]= data[column].replace("unknown", mode)
```

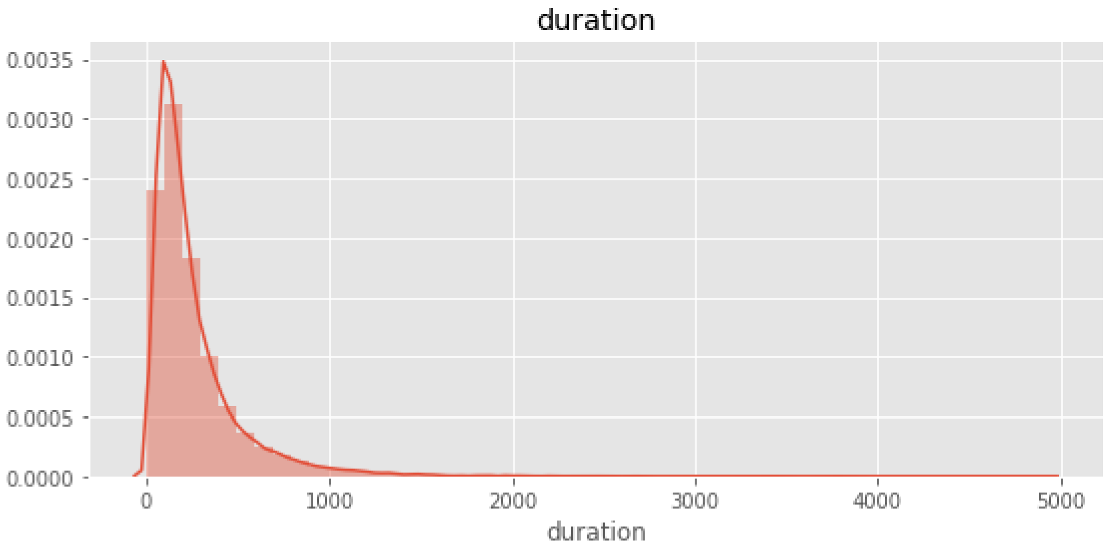
In [13]:
```
#Univariate analysis of Numeriacal columns
num_var= data.select_dtypes(include=np.number)
num_var.head()
```

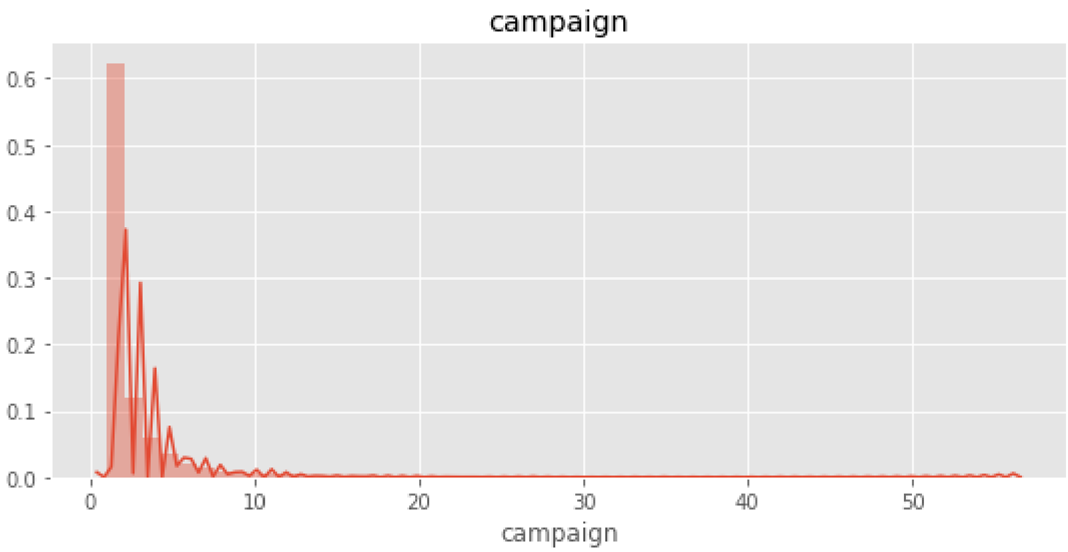
Out[13]:

	age	duration	campaign	pdays	previous
0	49	227	4	999	0
1	37	202	2	999	1
2	78	1148	1	999	0
3	36	120	2	999	0
4	59	368	2	999	0

In [14]:
```python
# plotting histogram for each numerical variable
plt.style.use("ggplot")
for column in ["age", "duration", "campaign"]:
    plt.figure(figsize=(20,4))
    plt.subplot(121)
    sns.distplot(data[column], kde=True)
    plt.title(column)
```







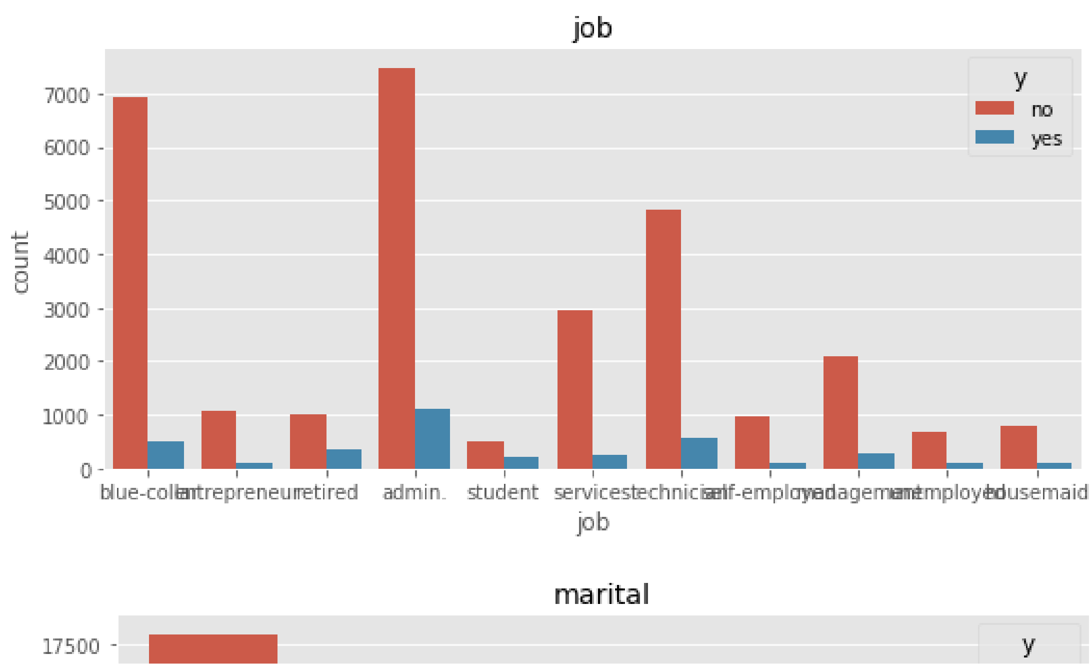
inference

1..From the histogram we see the features as age,duration and campaign are highly skewed here.may be due to the presence of outliers.

2..pdays have the value 999 indicates that the customer had not been contacted previously.

In [15]:
```
#drop pdays because consist  of a single value,and their
#variance is quite less and hence we can drop them

data.drop(columns=["pdays", "previous"], axis=1, inplace=True)
```

In [16]:
```
#BIVATRIATE ANALYSIS OF CATOGORICAL COLUMN
plt.style.use("ggplot")
for column in cato_var:
    plt.figure(figsize=(20,4))
    plt.subplot(121)
    sns.countplot(data[column], hue=data["y"])
    plt.title(column)
    plt.xticks(rotation=360)
```



Observations:

1.Customers with admin jobs have the majority amongst those who have subscirbed to the term deposit.

2..They are married.

3..customers hold a university degree

4.They do not hold a credit in default

5..There is more availability of phone calls to contacting customers.

Handling outliers

In [17]:
```
data.describe()
```

Out[17]:

	age	duration	campaign
count	32950.000000	32950.000000	32950.000000
mean	40.014112	258.127466	2.560607
std	10.403636	258.975917	2.752326
min	17.000000	0.000000	1.000000
25%	32.000000	103.000000	1.000000
50%	38.000000	180.000000	2.000000
75%	47.000000	319.000000	3.000000
max	98.000000	4918.000000	56.000000

age duration and campaign are skewed towards right, we will compute the IQR and replace the outliers with the lower and upper boundaries

In [18]:
```
# compute interquantile range to calculate the boundaries
lower_boundries= []
upper_boundries= []
for i in ["age", "duration", "campaign"]:
    IQR= data[i].quantile(0.75) - data[i].quantile(0.25)
    lower_bound= data[i].quantile(0.25) - (1.5*IQR)
    upper_bound= data[i].quantile(0.75) + (1.5*IQR)

    print(i, ":", lower_bound, ",",  upper_bound)

    lower_boundries.append(lower_bound)
    upper_boundries.append(upper_bound)
```

```
age : 9.5 , 69.5
duration : -221.0 , 643.0
campaign : -2.0 , 6.0
```

In [19]:
```
lower_boundries
```

Out[19]:
```
[9.5, -221.0, -2.0]
```

In [20]:
```
upper_boundries
```

Out[20]:
```
[69.5, 643.0, 6.0]
```

In [21]:
```
print(IQR)
```

```
2.0
```

In [22]:
```
# replace the all the outliers which is greater than upper boundary by upper
j = 0
for i in ["age", "duration", "campaign"]:
    data.loc[data[i] > upper_boundries[j], i] = int(upper_boundries[j])
    j = j + 1
```

Since,

for age the lower boundary (9.5) < minimum value (17) for duration and campaign the lower boundaries are negative (-221.0), (-2.0) resp. replacing outliers with the lower boundary is not required

In [23]:
```
data.describe()
```

Out[23]:

	age	duration	campaign
count	32950.000000	32950.000000	32950.000000
mean	39.929894	234.923915	2.271077
std	10.118566	176.854558	1.546302
min	17.000000	0.000000	1.000000
25%	32.000000	103.000000	1.000000
50%	38.000000	180.000000	2.000000
75%	47.000000	319.000000	3.000000
max	69.000000	643.000000	6.000000

In [24]:
```
#After replacing the outliers with the upper boundary,
#the maximum values has been changed without impacting any other parameters
```

#Encoding catogerical Features

In [25]:
```
cato_var
```

Out[25]:
```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'conta
ct',
       'month', 'day_of_week', 'poutcome', 'y'],
      dtype='object')
```

In [26]:
```
# check categorical class
for i in cato_var:
    print(i, ":", data[i].unique())
```

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

In [27]:
```
# initializing label encoder
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()

# iterating through each categorical feature and label encoding them
for feature in cato_var:
    data[feature]= le.fit_transform(data[feature])
```

In [28]:
```
data.head()
```

Out[28]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	durati
0	49	1	1	2	0	0	0	0	7	4	2
1	37	2	1	6	0	0	0	1	7	4	2
2	69	5	1	0	0	0	0	0	3	1	6
3	36	0	1	6	0	1	0	1	6	1	1
4	59	5	0	6	0	0	0	0	4	3	3

Seperating dependent and independent variables

In [29]:
```
# feature variables
x= data.iloc[:, :-1]

# target variable
y= data.iloc[:, -1]
```

In [30]:
```
print(x)
```

```
        age  job  marital  education  default  housing  loan  contact  mont
h  \
0        49    1        1          2        0        0     0        0
7
1        37    2        1          6        0        0     0        1
7
2        69    5        1          0        0        0     0        0
3
3        36    0        1          6        0        1     0        1
6
4        59    5        0          6        0        0     0        0
4
...     ...  ...      ...        ...      ...      ...   ...      ...
...
32945    28    7        2          3        0        1     0        0
3
32946    52    9        1          5        0        1     0        0
7
32947    54    0        1          2        0        0     1        0
3
32948    29    0        1          6        0        0     0        1
6
32949    35    0        1          6        0        0     1        1
4

       day_of_week  duration  campaign  poutcome
0                4       227         4         1
1                4       202         2         0
2                1       643         1         1
3                1       120         2         1
4                3       368         2         1
...            ...       ...       ...       ...
32945            3       192         1         1
32946            0        64         1         0
32947            1       131         4         1
32948            0       165         1         1
32949            3       544         3         1

[32950 rows x 13 columns]
```

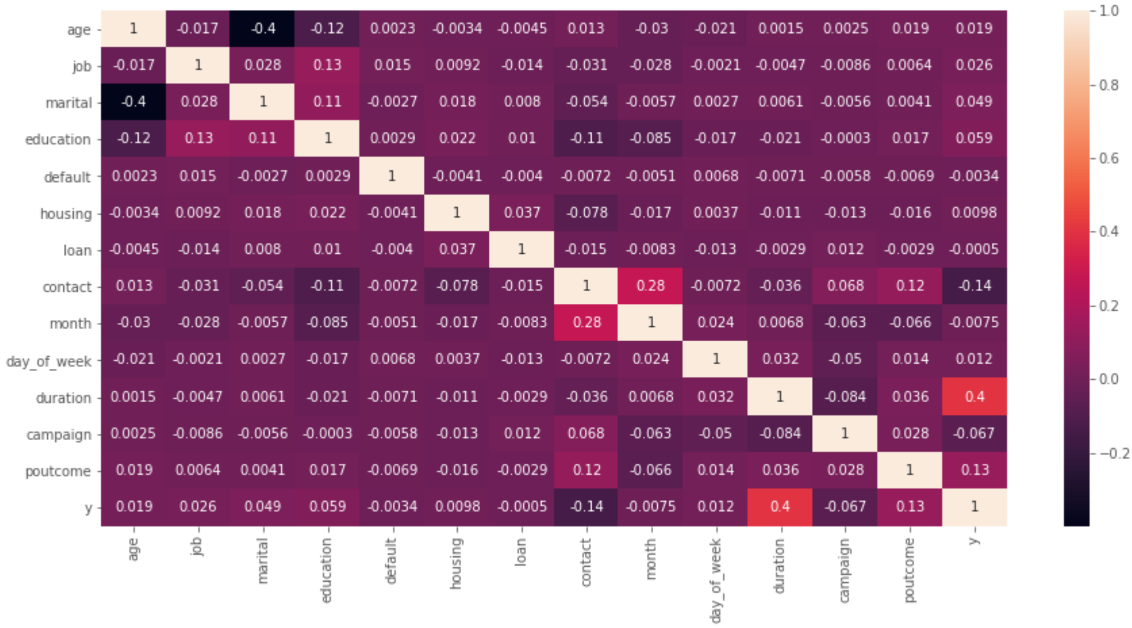
In [31]:
```
print(y)
```

```
0        0
1        0
2        1
3        0
4        0
        ..
32945    0
32946    0
32947    0
32948    0
32949    0
Name: y, Length: 32950, dtype: int32
```

checking correlation of feature variables

In [32]:
```python
plt.figure(figsize=(15,7))
sns.heatmap(data.corr(), annot=True)
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16f87630>
```



There are no features that are highly correlated and inversely correlated. If we had, we could have written the condition that if the correlation is higher than 0.8 (or can be any threshold value depending on the domain knowledge) and less than -0.8, we could have drop those features. Because those correlated features would have been doing the same job.

In [33]:
```python
plt.figure(figsize=(15,7))
```

Out[33]:
```
<Figure size 1080x504 with 0 Axes>
```

```
<Figure size 1080x504 with 0 Axes>
```

In [34]:
```python
# There are no features that are highly correlated and inversely correlated
#If we had, we could have written the condition that if the correlation is h
#(or can be any threshold value depending on the domain knowledge) and less
#we could have drop those features. Because those correlated features would
```

Handling imbalanced dataset.

Since the class distribution in the target variable is ~89:11 indicating an imbalance dataset, we need to resample it.

In [ ]:

```

```

In [35]:

```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\raksh\anaconda
3\lib\site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\raksh\anaconda3\l
ib\site-packages (from imbalanced-learn) (1.18.1)
Requirement already satisfied: scipy>=1.5.0 in c:\users\raksh\anaconda3\li
b\site-packages (from imbalanced-learn) (1.7.3)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\raksh\anaco
nda3\lib\site-packages (from imbalanced-learn) (1.0.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\raksh\anaconda3\l
ib\site-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\raksh\anac
onda3\lib\site-packages (from imbalanced-learn) (3.1.0)
Note: you may need to restart the kernel to use updated packages.

DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number. p
ip 23.3 will enforce this behaviour change. A possible replacement is to u
pgrade to a newer version of pyodbc or contact the author to suggest that
they release a version with a conforming version number. Discussion can be
found at https://github.com/pypa/pip/issues/12063 (https://github.com/pyp
a/pip/issues/12063)
```

In [36]:

```
from imblearn.combine import SMOTETomek
```

In [37]:

```
#initialising oversampling
smote = SMOTETomek(sampling_strategy=0.75)

# Implement oversampling to training data
x_sm, y_sm = smote.fit_resample(x, y)

# x_sm and y_sm are the resampled data

# target class count of resampled dataset
y_sm.value_counts()
```

Out[37]:

```
0    28934
1    21624
Name: y, dtype: int64
```

splitting resampled data in train and test data

In [38]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x_sm, y_sm, test_size=0.
```

Gridsearch and Hyperparameter tuning

LOGISTIC REGRESSION

In [39]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
```

In [40]:
```
# selecting the classifier
log_reg= LogisticRegression()

# selecting hyperparameter tuning
log_param= {"C": 10.0**np.arange(-2,3), "penalty": ["l1", "l2"]}

# defining stratified Kfold cross validation
cv_log= StratifiedKFold(n_splits=5)

# using gridsearch for respective parameters
gridsearch_log = GridSearchCV(log_reg, log_param, cv=cv_log, scoring="f1_ma

# fitting the model on resampled data
gridsearch_log.fit(x_train, y_train)

# printing best score and best parameters
print("best score is:" ,gridsearch_log.best_score_)
print("best parameters are:" ,gridsearch_log.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END .................................C=0.01, penalty=l1; total time=
0.0s
[CV] END .................................C=0.01, penalty=l1; total time=
0.0s
[CV] END .................................C=0.01, penalty=l1; total time=
0.0s
[CV] END .................................C=0.01, penalty=l1; total time=
0.0s
[CV] END .................................C=0.01, penalty=l1; total time=
0.0s
[CV] END .................................C=0.01, penalty=l2; total time=
0.2s
[CV] END .................................C=0.01, penalty=l2; total time=
0.2s
[CV] END .................................C=0.01, penalty=l2; total time=
0.2s
[CV] END .................................C=0.01, penalty=l2; total time=
0.2s
[CV] END .................................C=0.01, penalty=l2; total time=
0.2s
[CV] END ..................................C=0.1, penalty=l1; total time=
0.0s
[CV] END ..................................C=0.1, penalty=l1; total time=
0.0s
[CV] END ..................................C=0.1, penalty=l1; total time=
0.0s
[CV] END ..................................C=0.1, penalty=l1; total time=
0.0s
[CV] END ..................................C=0.1, penalty=l1; total time=
0.0s
[CV] END ..................................C=0.1, penalty=l2; total time=
0.2s
[CV] END ..................................C=0.1, penalty=l2; total time=
0.2s
[CV] END ..................................C=0.1, penalty=l2; total time=
0.2s
[CV] END ..................................C=0.1, penalty=l2; total time=
0.2s
[CV] END ..................................C=0.1, penalty=l2; total time=
0.2s
[CV] END ..................................C=1.0, penalty=l1; total time=
0.0s
[CV] END ..................................C=1.0, penalty=l1; total time=
0.0s
[CV] END ..................................C=1.0, penalty=l1; total time=
0.0s
[CV] END ..................................C=1.0, penalty=l1; total time=
0.0s
[CV] END ..................................C=1.0, penalty=l1; total time=
0.0s
[CV] END ..................................C=1.0, penalty=l2; total time=
0.2s
[CV] END ..................................C=1.0, penalty=l2; total time=
0.2s
[CV] END ..................................C=1.0, penalty=l2; total time=
0.2s
[CV] END ..................................C=1.0, penalty=l2; total time=
0.2s
[CV] END ..................................C=1.0, penalty=l2; total time=
0.2s
```

```
[CV] END ................................C=10.0, penalty=l1; total time=
0.0s
[CV] END ................................C=10.0, penalty=l1; total time=
0.0s
[CV] END ................................C=10.0, penalty=l1; total time=
0.0s
[CV] END ................................C=10.0, penalty=l1; total time=
0.0s
[CV] END ................................C=10.0, penalty=l1; total time=
0.0s
[CV] END ................................C=10.0, penalty=l2; total time=
0.2s
[CV] END ................................C=10.0, penalty=l2; total time=
0.2s
[CV] END ................................C=10.0, penalty=l2; total time=
0.2s
[CV] END ................................C=10.0, penalty=l2; total time=
0.2s
[CV] END ................................C=10.0, penalty=l2; total time=
0.2s
[CV] END ...............................C=100.0, penalty=l1; total time=
0.0s
[CV] END ...............................C=100.0, penalty=l1; total time=
0.0s
[CV] END ...............................C=100.0, penalty=l1; total time=
0.0s
[CV] END ...............................C=100.0, penalty=l1; total time=
0.0s
[CV] END ...............................C=100.0, penalty=l1; total time=
0.0s
[CV] END ...............................C=100.0, penalty=l2; total time=
0.2s
[CV] END ...............................C=100.0, penalty=l2; total time=
0.2s
[CV] END ...............................C=100.0, penalty=l2; total time=
0.2s
[CV] END ...............................C=100.0, penalty=l2; total time=
0.2s
[CV] END ...............................C=100.0, penalty=l2; total time=
0.2s
best score is: 0.7921968066413011
best parameters are: {'C': 10.0, 'penalty': 'l2'}
```

In [41]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classificatio
```

In [42]:

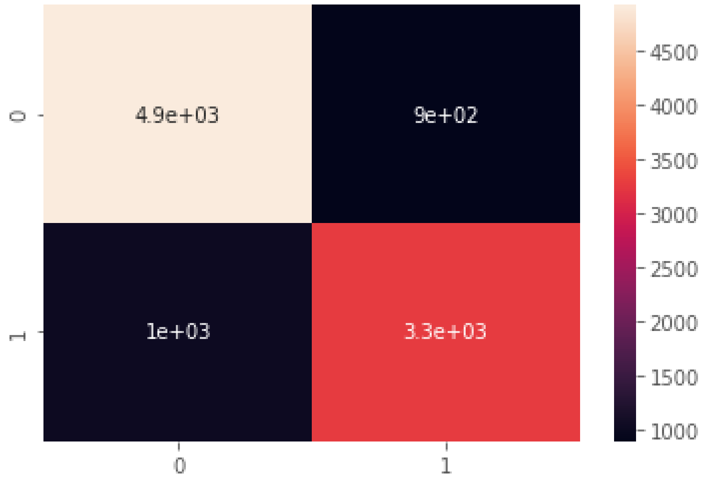
```
# checking model performance
y_predicted= gridsearch_log.predict(x_test)

cm= confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[4912  898]
 [1027 3275]]
```

In [43]:
```
print(accuracy_score(y_test, y_predicted))
print(classification_report(y_test, y_predicted))
sns.heatmap(cm, annot=True)
```

```
0.8096321202531646
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      5810
           1       0.78      0.76      0.77      4302

    accuracy                           0.81     10112
   macro avg       0.81      0.80      0.80     10112
weighted avg       0.81      0.81      0.81     10112
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x15c82c10>



Random forest

In [44]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
```

In [45]:

```
rf=RandomForestClassifier()

rf_param= {
            "n_estimators": [int(x) for x in np.linspace(start=100, stop=100
            "max_features": ["auto", "sqrt", "log2"],
#             "max_depth": [4,5,6,7,8],
            "max_depth": [int(x) for x in np.linspace(start=5, stop=30, num=
            "min_samples_split": [5,10,15,100],
            "min_samples_leaf": [1,2,5,10],
            "criterion":['gini', 'entropy']
          }

cv_rf= StratifiedKFold(n_splits=5)

randomsearch_rf = RandomizedSearchCV(rf, rf_param, cv=cv_rf, scoring="f1_ma


randomsearch_rf.fit(x_train, y_train)
print("best score is:", randomsearch_rf.best_score_)
print("best parameters are:", randomsearch_rf.best_params_)
```

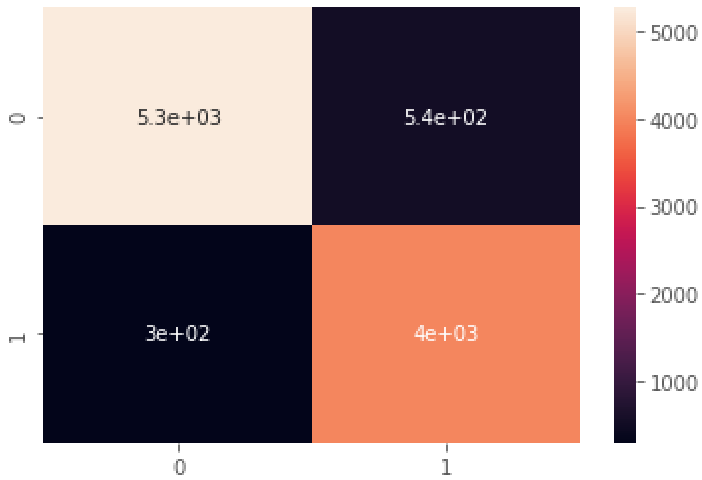
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END criterion=entropy, max_depth=25, max_features=log2, min_samples_l
eaf=1, min_samples_split=5, n_estimators=200; total time=   6.5s
[CV] END criterion=entropy, max_depth=25, max_features=log2, min_samples_l
eaf=1, min_samples_split=5, n_estimators=200; total time=   6.6s
[CV] END criterion=entropy, max_depth=25, max_features=log2, min_samples_l
eaf=1, min_samples_split=5, n_estimators=200; total time=   6.5s
[CV] END criterion=entropy, max_depth=25, max_features=log2, min_samples_l
eaf=1, min_samples_split=5, n_estimators=200; total time=   6.7s
[CV] END criterion=entropy, max_depth=25, max_features=log2, min_samples_l
eaf=1, min_samples_split=5, n_estimators=200; total time=   7.6s
[CV] END criterion=gini, max_depth=25, max_features=auto, min_samples_leaf
=1, min_samples_split=100, n_estimators=700; total time=  15.3s
[CV] END criterion=gini, max_depth=25, max_features=auto, min_samples_leaf
=1, min_samples_split=100, n_estimators=700; total time=  14.9s
[CV] END criterion=gini, max_depth=25, max_features=auto, min_samples_leaf
=1, min_samples_split=100, n_estimators=700; total time=  15.0s
[CV] END criterion=gini, max_depth=25, max_features=auto, min_samples_leaf
=1, min_samples_split=100, n_estimators=700; total time=  14.9s
[CV] END criterion=gini, max_depth=25, max_features=auto, min_samples_leaf
=1, min_samples_split=100, n_estimators=700; total time=  14.7s
[CV] END criterion=gini, max_depth=10, max_features=auto, min_samples_leaf
=1, min_samples_split=15, n_estimators=700; total time=  12.8s
[CV] END criterion=gini, max_depth=10, max_features=auto, min_samples_leaf
=1, min_samples_split=15, n_estimators=700; total time=  12.8s
[CV] END criterion=gini, max_depth=10, max_features=auto, min_samples_leaf
=1, min_samples_split=15, n_estimators=700; total time=  12.8s
[CV] END criterion=gini, max_depth=10, max_features=auto, min_samples_leaf
=1, min_samples_split=15, n_estimators=700; total time=  12.9s
[CV] END criterion=gini, max_depth=10, max_features=auto, min_samples_leaf
=1, min_samples_split=15, n_estimators=700; total time=  12.9s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
5, min_samples_split=5, n_estimators=400; total time=   4.5s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
5, min_samples_split=5, n_estimators=400; total time=   4.6s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
5, min_samples_split=5, n_estimators=400; total time=   4.5s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
5, min_samples_split=5, n_estimators=400; total time=   4.5s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
5, min_samples_split=5, n_estimators=400; total time=   4.4s
[CV] END criterion=gini, max_depth=25, max_features=sqrt, min_samples_leaf
=2, min_samples_split=5, n_estimators=500; total time=  13.5s
[CV] END criterion=gini, max_depth=25, max_features=sqrt, min_samples_leaf
=2, min_samples_split=5, n_estimators=500; total time=  13.7s
[CV] END criterion=gini, max_depth=25, max_features=sqrt, min_samples_leaf
=2, min_samples_split=5, n_estimators=500; total time=  13.3s
[CV] END criterion=gini, max_depth=25, max_features=sqrt, min_samples_leaf
=2, min_samples_split=5, n_estimators=500; total time=  13.4s
[CV] END criterion=gini, max_depth=25, max_features=sqrt, min_samples_leaf
=2, min_samples_split=5, n_estimators=500; total time=  13.6s
[CV] END criterion=entropy, max_depth=5, max_features=sqrt, min_samples_le
af=2, min_samples_split=5, n_estimators=300; total time=   3.6s
[CV] END criterion=entropy, max_depth=5, max_features=sqrt, min_samples_le
af=2, min_samples_split=5, n_estimators=300; total time=   3.5s
[CV] END criterion=entropy, max_depth=5, max_features=sqrt, min_samples_le
af=2, min_samples_split=5, n_estimators=300; total time=   3.4s
[CV] END criterion=entropy, max_depth=5, max_features=sqrt, min_samples_le
af=2, min_samples_split=5, n_estimators=300; total time=   3.4s
[CV] END criterion=entropy, max_depth=5, max_features=sqrt, min_samples_le
af=2, min_samples_split=5, n_estimators=300; total time=   3.5s
```

```
[CV] END criterion=entropy, max_depth=15, max_features=auto, min_samples_l
eaf=2, min_samples_split=15, n_estimators=1000; total time=  25.1s
[CV] END criterion=entropy, max_depth=15, max_features=auto, min_samples_l
eaf=2, min_samples_split=15, n_estimators=1000; total time=  25.5s
[CV] END criterion=entropy, max_depth=15, max_features=auto, min_samples_l
eaf=2, min_samples_split=15, n_estimators=1000; total time=  25.6s
[CV] END criterion=entropy, max_depth=15, max_features=auto, min_samples_l
eaf=2, min_samples_split=15, n_estimators=1000; total time=  25.2s
[CV] END criterion=entropy, max_depth=15, max_features=auto, min_samples_l
eaf=2, min_samples_split=15, n_estimators=1000; total time=  25.5s
[CV] END criterion=entropy, max_depth=25, max_features=auto, min_samples_l
eaf=5, min_samples_split=5, n_estimators=400; total time=  10.8s
[CV] END criterion=entropy, max_depth=25, max_features=auto, min_samples_l
eaf=5, min_samples_split=5, n_estimators=400; total time=  11.1s
[CV] END criterion=entropy, max_depth=25, max_features=auto, min_samples_l
eaf=5, min_samples_split=5, n_estimators=400; total time=  11.0s
[CV] END criterion=entropy, max_depth=25, max_features=auto, min_samples_l
eaf=5, min_samples_split=5, n_estimators=400; total time=  10.9s
[CV] END criterion=entropy, max_depth=25, max_features=auto, min_samples_l
eaf=5, min_samples_split=5, n_estimators=400; total time=  10.8s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
10, min_samples_split=10, n_estimators=200; total time=   2.4s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
10, min_samples_split=10, n_estimators=200; total time=   2.4s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
10, min_samples_split=10, n_estimators=200; total time=   2.2s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
10, min_samples_split=10, n_estimators=200; total time=   2.2s
[CV] END criterion=gini, max_depth=5, max_features=sqrt, min_samples_leaf=
10, min_samples_split=10, n_estimators=200; total time=   2.2s
[CV] END criterion=entropy, max_depth=30, max_features=log2, min_samples_l
eaf=2, min_samples_split=100, n_estimators=300; total time=   6.7s
[CV] END criterion=entropy, max_depth=30, max_features=log2, min_samples_l
eaf=2, min_samples_split=100, n_estimators=300; total time=   6.8s
[CV] END criterion=entropy, max_depth=30, max_features=log2, min_samples_l
eaf=2, min_samples_split=100, n_estimators=300; total time=   6.7s
[CV] END criterion=entropy, max_depth=30, max_features=log2, min_samples_l
eaf=2, min_samples_split=100, n_estimators=300; total time=   6.7s
[CV] END criterion=entropy, max_depth=30, max_features=log2, min_samples_l
eaf=2, min_samples_split=100, n_estimators=300; total time=   6.7s
best score is: 0.9068039289876884
best parameters are: {'n_estimators': 200, 'min_samples_split': 5, 'min_sa
mples_leaf': 1, 'max_features': 'log2', 'max_depth': 25, 'criterion': 'ent
ropy'}
```

In [ ]:

In [46]:
```python
# checking model performance
y_predicted_rf= randomsearch_rf.predict(x_test)

print(confusion_matrix(y_test, y_predicted_rf))
sns.heatmap(confusion_matrix(y_test, y_predicted_rf), annot=True)
print(accuracy_score(y_test, y_predicted_rf))
```

```
[[5265  545]
 [ 298 4004]]
0.9166337025316456
```



Prediction on the Test dataset

We have to perform the same preprocessing operations on the test data that we have performed on the train data. But here we already have preprocessed data which is present in the csv file new_test.csv

In [47]:
```python
test_data= pd.read_csv("C:/Users/raksh/Downloads/new_test (1).csv")
test_data.head()
```

Out[47]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	durati
0	32	4	0	6	0	0	0	0	3	3	1
1	37	10	3	6	0	0	0	0	4	3	1
2	55	5	0	5	1	2	0	0	3	2	1
3	44	2	1	0	1	0	0	1	4	3	
4	28	0	2	3	0	0	0	0	5	0	1

In [ ]:

Inference

As comparing logistic regression and Randpm forest .Random Forest classifier has given the best metric score on the validation data.

In [48]:
```python
# predicting the test data
y_predicted= randomsearch_rf.predict(test_data)
y_predicted
```

Out[48]:
```
array([0, 0, 0, ..., 1, 0, 0])
```

In [49]:
```python
 #dataset of predicted values for target variable y
prediction= pd.DataFrame(y_predicted, columns=["y_predicted"])
prediction_dataset= pd.concat([test_data, prediction], axis=1)
prediction_dataset
```

Out[49]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	du
0	32	4	0	6	0	0	0	0	3	3	
1	37	10	3	6	0	0	0	0	4	3	
2	55	5	0	5	1	2	0	0	3	2	
3	44	2	1	0	1	0	0	1	4	3	
4	28	0	2	3	0	0	0	0	5	0	
...	...	...	...	...	...	...	...	...	...	...	
8233	48	4	1	2	0	2	0	0	6	3	
8234	30	7	2	3	0	2	0	0	6	0	
8235	33	7	1	3	0	0	0	0	4	1	
8236	44	1	1	1	0	2	2	1	6	1	
8237	42	1	1	2	1	2	0	0	6	3	

8238 rows × 14 columns

In [ ]: