

Problem Statement

Provided with the client data such as : age of the client, their job type, their marital status, etc. Along with the client data, the information of the call such as the duration of the call, day and month of the call, etc is also given. Given this information, task is to predict if the client will subscribe to term deposit.

In [1]:

```
# importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

For mathematical calculations
For data visualization
For plotting graphs
To ignore any warnings

In [2]:

```
#Loading the data
train=pd.read_csv("C:\\Users\\Mr Sinha\\Desktop\\internshala\\final\\train.csv")
test=pd.read_csv("C:\\Users\\Mr Sinha\\Desktop\\internshala\\final\\test.csv")
```

In [3]:

```
train.columns
```

Out[3]:

```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

In [4]:

```
test.columns
```

Out[4]:

```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome'],
      dtype='object')
```

It can be inferred that subscribed is the target variable as it is not present in the test dataset. Let's look at the shape of the dataset.

In [5]:

```
train.shape, test.shape
```

Out[5]:

```
((31647, 18), (13564, 17))
```

We have 17 independent variables and 1 target variable, i.e. subscribed in the train dataset. We have similar features in the test dataset as the train dataset except the subscribed. We will predict the subscribed with the help of model built using the train data.

Next, let's look at how many categorical and numerical variables are there in our dataset. We will look at their data types.

In [6]:

```
# Print data types for each variable
train.dtypes
```

Out[6]:

```
ID                int64
age               int64
job              object
marital          object
education        object
default          object
balance          int64
housing          object
loan            object
contact          object
day             int64
month           object
duration         int64
campaign         int64
pdays          int64
previous         int64
poutcome        object
subscribed      object
dtype: object
```

We can see there are two format of data types:


1. **object:** Object format means variables are categorical. Categorical variables in our dataset are: job, marital, education, default, housing, loan, contact, month, poutcome, subscribed
2. **int64:** It represents the integer variables. Integer variables in our dataset are: ID, age, balance, day, duration, campaign, pdays, previous

In [7]:

```
#printing first five rows of the dataset
train.head()
```

Out[7]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign
0	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	
1	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	
2	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	
3	43962	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	
4	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	



Univariate Analysis

Now Let's look at the distribution of our target variable, i.e. subscribed. As it is a categorical variable, let us look at its frequency table, percentage distribution and bar plot.

In [8]:

```
train['subscribed'].value_counts()
```

Out[8]:

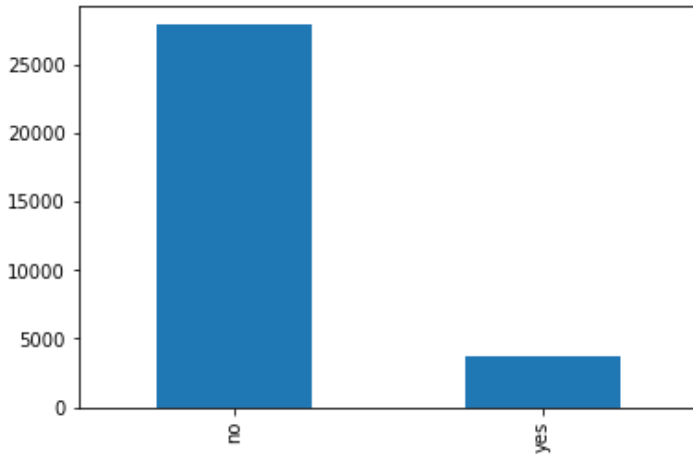
```
no      27932
yes      3715
Name: subscribed, dtype: int64
```

In [9]:

```
# plotting the bar plot of frequencies
train['subscribed'].value_counts().plot.bar()
```

Out[9]:

<matplotlib.axes._subplots.AxesSubplot at 0x1923eb3fdc0>



So, 3715 users out of total 31647 have subscribed which is around 12%. Let's now explore the variables to have a better understanding of the dataset. We will first explore the variables individually using univariate analysis, then we will look at the relation between various independent variables and the target variable. We will also look at the correlation plot to see which variables affects the target variable most.

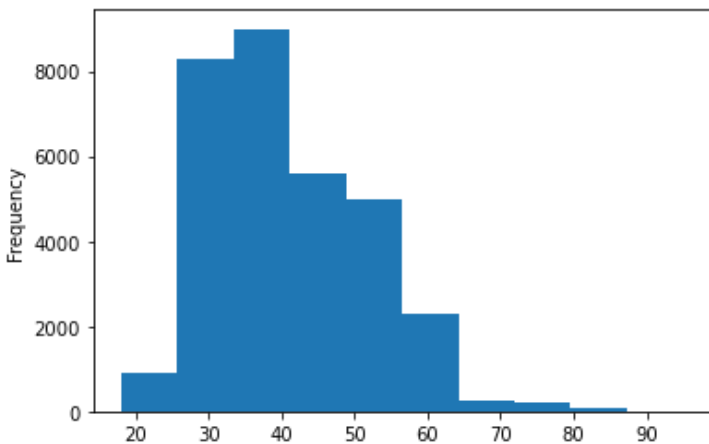
Let's first look at the distribution of age variable to see how many people belongs to a particular age group.

In [10]:

```
train['age'].plot.hist()
```

Out[10]:

<matplotlib.axes._subplots.AxesSubplot at 0x1923ee35790>



We can infer that most of the clients fall in the age group between 20-60. Now let's look at what are the different types of jobs of the clients. As job is a categorical variable, we will look at its frequency table

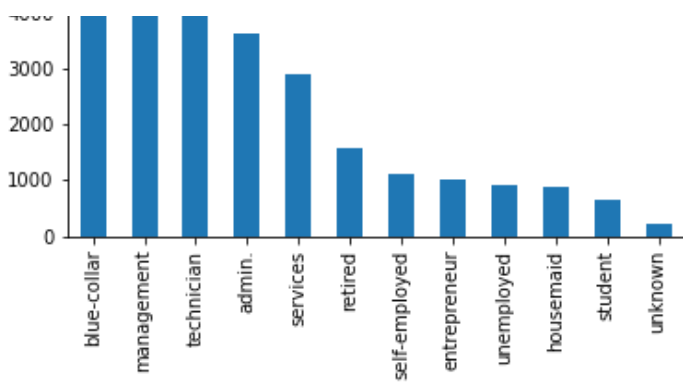
In [11]:

```
train['job'].value_counts().plot.bar()
```

Out[11]:

<matplotlib.axes._subplots.AxesSubplot at 0x1923f7df8b0>





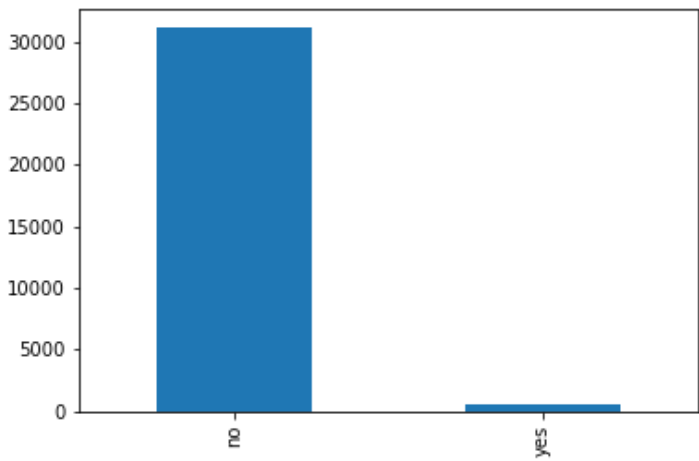
We see that most of the clients belongs to blue-collar job and the students are least in number as students generally do not take a term deposit. Let's also look at how many clients have default history.

In [12]:

```
train['default'].value_counts().plot.bar()
```

Out[12]:

<matplotlib.axes._subplots.AxesSubplot at 0x1923ee35df0>



More than 90% of the clients have no default history. Now we will explore these variables against the target variable using bivariate analysis. We will make use of scatter plots for continuous or numeric variables and crosstabs for the categorical variables. Let's start with job and subscribed variable.

Bivariate Analysis

In [13]:

```
print(pd.crosstab(train['job'],train['subscribed']))

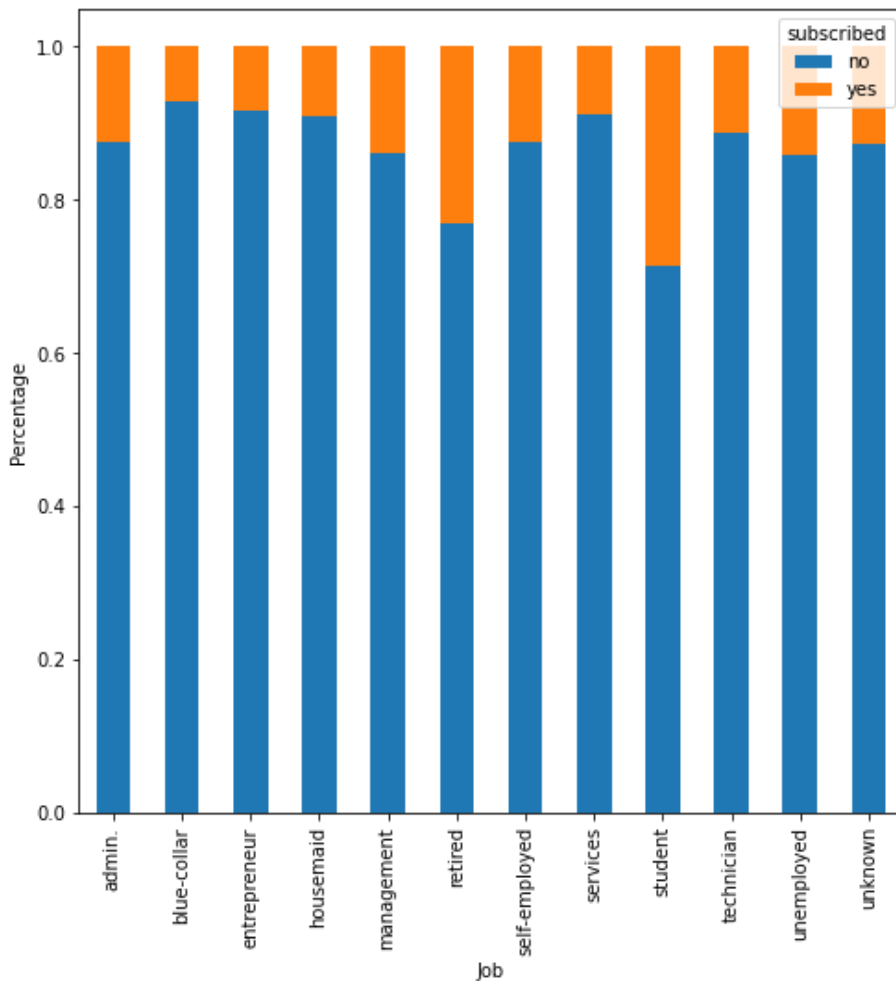
job=pd.crosstab(train['job'],train['subscribed'])
job.div(job.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(8,8))
plt.xlabel('Job')
plt.ylabel('Percentage')
```

subscribed	no	yes
job		
admin.	3179	452
blue-collar	6353	489
entrepreneur	923	85
housemaid	795	79
management	5716	923
retired	1212	362
self-employed	983	140
services	2649	254
student	453	182
technician	4713	594
unemployed	776	129
unknown	100	20

unknown 100 20

Out[13]:

Text(0, 0.5, 'Percentage')



From the above graph we can infer that students and retired people have higher chances of subscribing to a term deposit, which is surprising as students generally do not subscribe to a term deposit. The possible reason is that the number of students in the dataset is less and comparatively to other job types, more students have subscribed to a term deposit.

Next, let's explore the default variable against the subscribed variable.

In [14]:

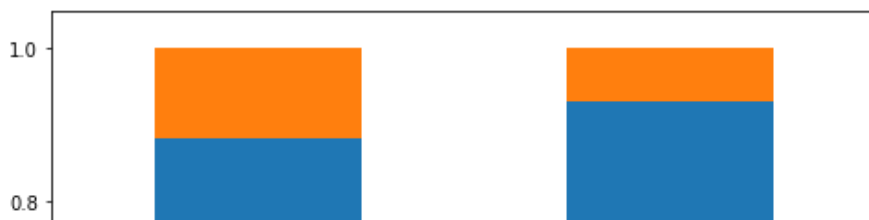
```
print(pd.crosstab(train['default'], train['subscribed']))

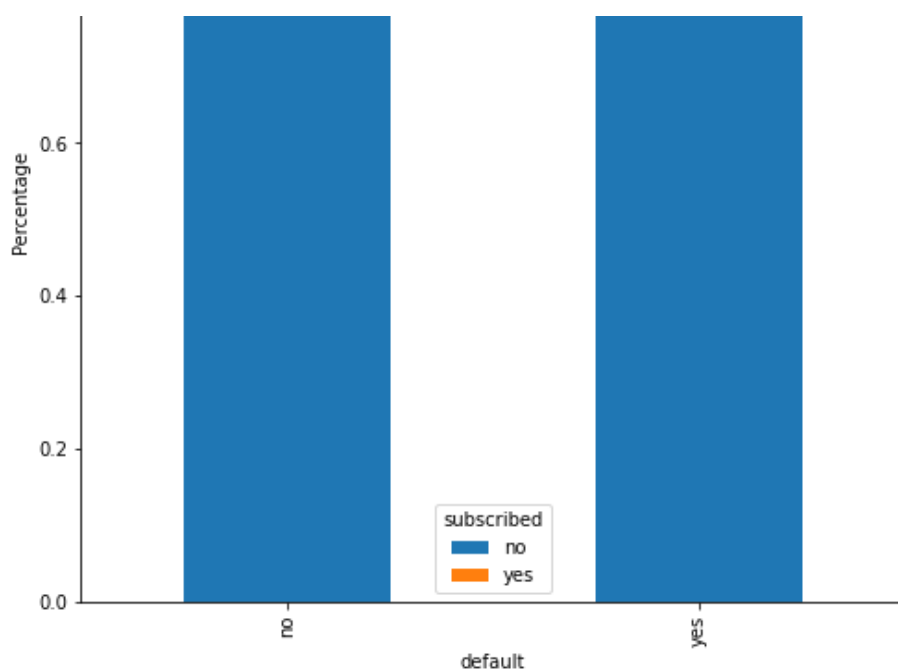
default=pd.crosstab(train['default'], train['subscribed'])
default.div(default.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize
=(8,8))
plt.xlabel('default')
plt.ylabel('Percentage')
```

subscribed	no	yes
default		
no	27388	3674
yes	544	41

Out[14]:

Text(0, 0.5, 'Percentage')





We can infer that clients having no previous default have slightly higher chances of subscribing to a term loan as compared to the clients who have previous default history.

Let's now look at how correlated our numerical variables are. We will see the correlation between each of these variables and the variable which have high negative or positive values are correlated. By this we can get an overview of the variables which might affect our target variable. We will convert our target variable into numeric values first.

In [15]:

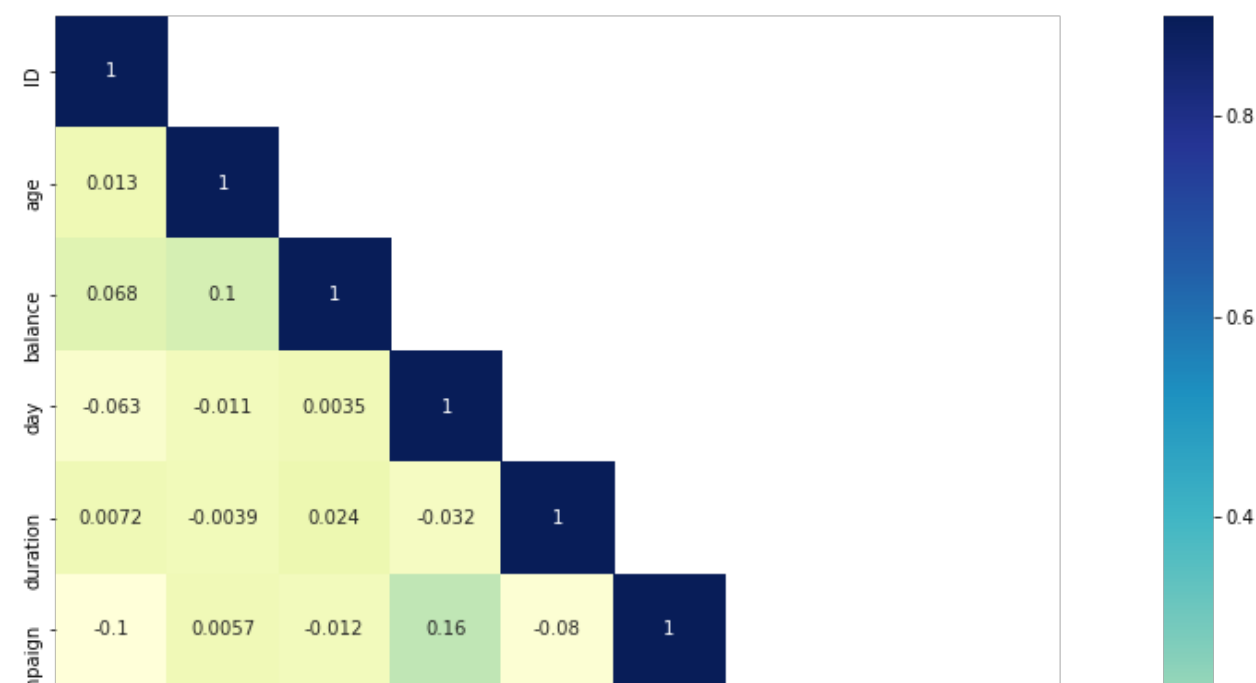
```
train['subscribed'].replace('no', 0,inplace=True)
train['subscribed'].replace('yes', 1,inplace=True)
```

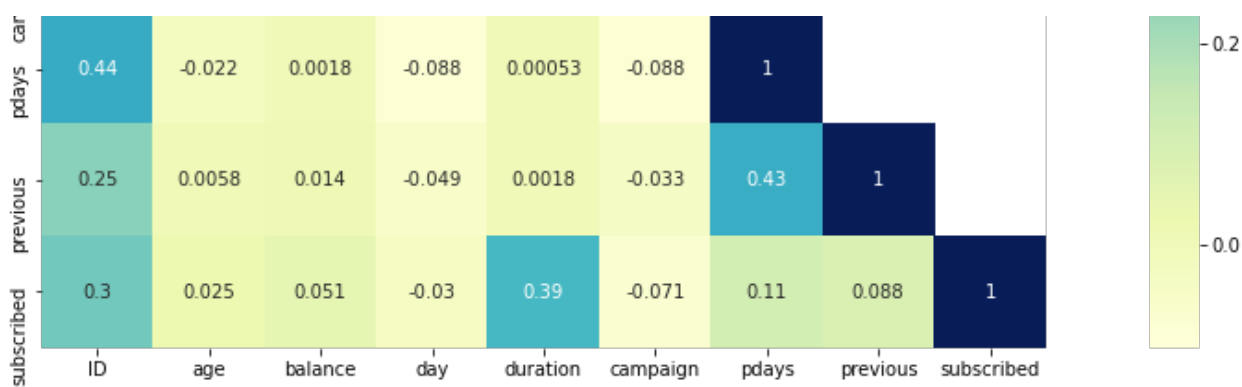
In [16]:

```
corr = train.corr()
mask = np.array(corr)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corr, mask=mask,vmax=.9, square=True,annot=True, cmap="YlGnBu")
```

Out[16]:

<matplotlib.axes._subplots.AxesSubplot at 0x1923f9e9dc0>





We can infer that duration of the call is highly correlated with the target variable. This can be verified as well. As the duration of the call is more, there are higher chances that the client is showing interest in the term deposit and hence there are higher chances that the client will subscribe to term deposit.

Next we will look for any missing values in the dataset.

In [17]:

```
train.isnull().sum()
```

Out[17]:

```
ID          0
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays       0
previous     0
poutcome     0
subscribed   0
dtype: int64
```

There are no missing values in the train dataset.

Next, we will start to build our predictive model to predict whether a client will subscribe to a term deposit or not.

As the sklearn models takes only numerical input, we will convert the categorical variables into numerical values using dummies. We will remove the ID variables as they are unique values and then apply dummies. We will also remove the target variable and keep it in a separate variable.



Model Building

In [18]:

```
target = train['subscribed']
train = train.drop('subscribed', 1)
```

In [19]:

```
# applying dummies on the train dataset
train = pd.get_dummies(train)
```

In [21]:

```
from sklearn.model_selection import train_test_split
```

In [22]:

```
# splitting into train and validation with 20% data in validation set and 80% data in tra
in set.
X_train, X_val, y_train, y_val = train_test_split(train, target, test_size = 0.2, random
_state=12)
```

Now our data is ready. Its time to build our model and check its performance. Logistic regression is used for classification problems and as it is a classification problem let's first build a Logistic Regression model.

Logistic Regression

In [23]:

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

In [24]:

```
# defining the logistic regression model
lreg = LogisticRegression()
```

In [25]:

```
# fitting the model on X_train and y_train
lreg.fit(X_train,y_train)
```

Out[25]:

```
LogisticRegression()
```

In [26]:

```
# making prediction on the validation set
prediction = lreg.predict(X_val)
```

Now we will evaluate how accurate our predictions are. As the evaluation metric for this problem is accuracy, let's calculate the accuracy on validation set.

In [27]:

```
from sklearn.metrics import accuracy_score
```

In [28]:

```
# calculating the accuracy score
accuracy_score(y_val, prediction)
```

Out[28]:

```
0.8928909952606635
```

We got an accuracy score of around 90% on the validation dataset. Logistic regression has a linear decision boundary. What if our data have non linearity? We need a model that can capture this non linearity.

Let's try decision tree algorithm now to check if we get better accuracy with that.

Decision Tree

In [29]:

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


In [30]:

```
# defining the decision tree model with depth of 4, you can tune it further to improve the accuracy score
clf = DecisionTreeClassifier(max_depth=4, random_state=0)
```

In [31]:

```
# fitting the decision tree model
clf.fit(X_train,y_train)
```

Out[31]:

```
DecisionTreeClassifier(max_depth=4, random_state=0)
```

In [32]:

```
# making prediction on the validation set
predict = clf.predict(X_val)
```

In [33]:

```
# calculating the accuracy score
accuracy_score(y_val, predict)
```

Out[33]:

```
0.9042654028436019
```

We got an accuracy of more than 90% on the validation set. You can try to improve the score by tuning hyperparameters of the model. Let's now make the prediction on test dataset. We will make the similar changes in the test set as we have done in the training set before making the predictions.

In [34]:

```
test = pd.get_dummies(test)
```

In [35]:

```
test_prediction = clf.predict(test)
```

Finally, we will save these predictions into a csv file. You can then open this csv file and copy paste the predictions on the provided excel file to generate score.

In [36]:

```
submission = pd.DataFrame()
```

In [37]:

```
# creating a Business_Sourced column and saving the predictions in it
submission['ID'] = test['ID']
submission['subscribed'] = test_prediction
```

Since the target variable is yes or no, we will convert 1 and 0 in the predictions to yes and no respectively.

In [38]:

```
submission['subscribed'].replace(0, 'no', inplace=True)
submission['subscribed'].replace(1, 'yes', inplace=True)
```

In [43]:

```
submission.to_csv('abc.csv', header=True, index=False)
```

In []:

