THANKS FOR CLICKING !!!!

What are you going to learn with this Kernel?

- Atribute information Analysis.
- Categorical to Continuous/Dummies Easy way
- Machine Learning (Logistic Regression, KNN, SVM, Decision Tree, Random Forest, GradientBoostingClassifier, XGBClassifier, GaussianNB)
- ROC curve
- How to understand the problem and see which is the best model for your Dependent Variable
- Precision, Recall, F1, Avg_total Analysis

Bank Marketing

Abstract: The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

Data Set Information: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Attribute Information:

Bank client data:

- Age (numeric)
- Job : type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
- Marital : marital status (categorical: 'divorced', 'married', 'single', 'unknown' ; note: 'divorced' means divorced or widowed)
- Education (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')
- Default: has credit in default? (categorical: 'no', 'yes', 'unknown')
- Housing: has housing loan? (categorical: 'no', 'yes', 'unknown')
- Loan: has personal loan? (categorical: 'no', 'yes', 'unknown')

Related with the last contact of the current campaign:

- Contact: contact communication type (categorical: 'cellular','telephone')
- Month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- Day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
- Duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed.

Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes:

- Campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
- Pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- Previous: number of contacts performed before this campaign and for this client (numeric)
- Poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

Social and economic context attributes

- Emp.var.rate: employment variation rate - quarterly indicator (numeric)
- Cons.price.idx: consumer price index - monthly indicator (numeric)
- Cons.conf.idx: consumer confidence index - monthly indicator (numeric)
- Euribor3m: euribor 3 month rate - daily indicator (numeric)
- Nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

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

Source:

- Dataset from : http://archive.ics.uci.edu/ml/datasets/Bank+Marketing# (http://archive.ics.uci.edu/ml/datasets/Bank+Marketing#)

In [1]:

```
# Importing Data Analysis Librarys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
bank = pd.read_csv('../input/bank-additional-full.csv', sep = ';')
#Converting dependent variable categorical to dummy
y = pd.get_dummies(bank['y'], columns = ['y'], prefix = ['y'], drop_fi
rst = True)
bank.head()
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_w
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

In [3]:

```
# take a look at the type, number of columns, entries, null values etc..
bank.info()
# bank.isnull().any() # one way to search for null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [4]:

```
bank.columns
```

Out[4]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'lo
an',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pda
ys',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

1. Bank client data Analysis and Categorical Treatment

- Work with the atributes related to bank clients

- Work with the atributes related to bank clients
- To make things more clear, i'm going to creat a new datasets that contains just this part of data

In [5]:
```
bank_client = bank.iloc[: , 0:7]
bank_client.head()
```

Out[5]:

	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

1.1. Knowing the categorical variables

In [6]:
```
# knowing the categorical variables
print('Jobs:\n', bank_client['job'].unique())
```

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

In [7]:
```
print('Marital:\n', bank_client['marital'].unique())
```

```
Marital:
 ['married' 'single' 'divorced' 'unknown']
```

In [8]:
```
print('Education:\n', bank_client['education'].unique())
```

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

In [9]:
```
print('Default:\n', bank_client['default'].unique())
print('Housing:\n', bank_client['housing'].unique())
print('Loan:\n', bank_client['loan'].unique())
```

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

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

1.2. Age

- Trying to find some insights crossing those variables

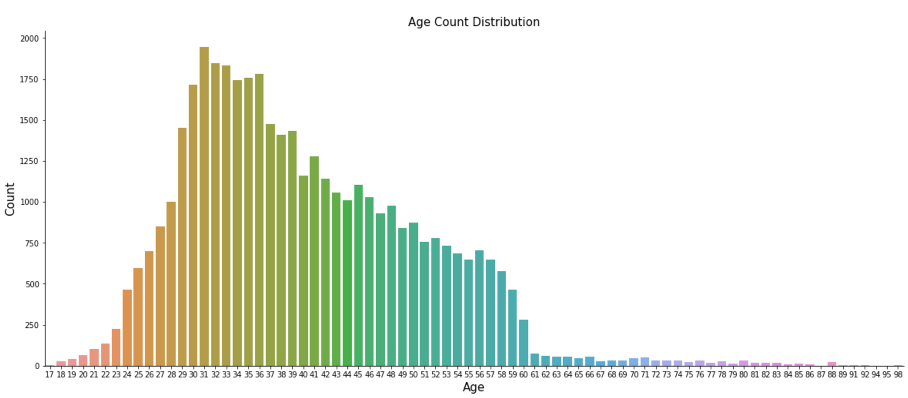
In [10]:

```python
#Trying to find some strange values or null values
print('Min age: ', bank_client['age'].max())
print('Max age: ', bank_client['age'].min())
print('Null Values: ', bank_client['age'].isnull().any())
```

```
Min age:  98
Max age:  17
Null Values:  False
```

In [11]:

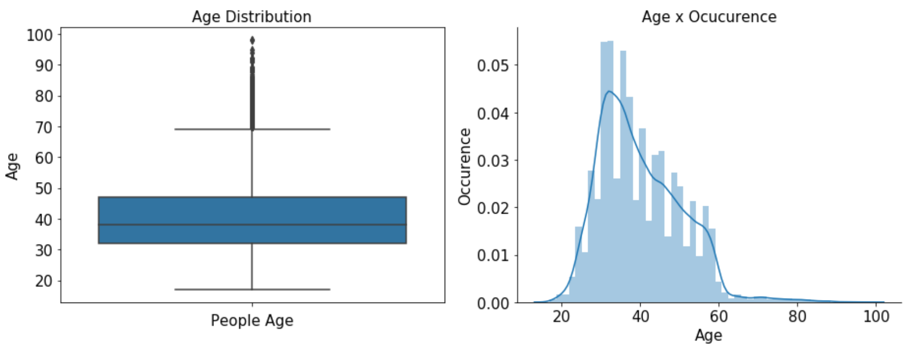
```python
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'age', data = bank_client)
ax.set_xlabel('Age', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
sns.despine()
```



In [12]:

```python
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5
))
sns.boxplot(x = 'age', data = bank_client, orient = 'v', ax = ax1)
ax1.set_xlabel('People Age', fontsize=15)
ax1.set_ylabel('Age', fontsize=15)
ax1.set_title('Age Distribution', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(bank_client['age'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('Age', fontsize=15)
ax2.set_ylabel('Occurence', fontsize=15)
```

```
ax2.set_ylabel('Occurence', fontsize=15)
ax2.set_title('Age x Ocucurence', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



In [13]:

```
# Quartiles
print('1º Quartile: ', bank_client['age'].quantile(q = 0.25))
print('2º Quartile: ', bank_client['age'].quantile(q = 0.50))
print('3º Quartile: ', bank_client['age'].quantile(q = 0.75))
print('4º Quartile: ', bank_client['age'].quantile(q = 1.00))
#Calculate the outliers:
  # Interquartile range, IQR = Q3 - Q1
  # lower 1.5*IQR whisker = Q1 - 1.5 * IQR
  # Upper 1.5*IQR whisker = Q3 + 1.5 * IQR

print('Ages above: ', bank_client['age'].quantile(q = 0.75) +
                      1.5*(bank_client['age'].quantile(q = 0.75) - ban
k_client['age'].quantile(q = 0.25)), 'are outliers')
```

```
1º Quartile:  32.0
2º Quartile:  38.0
3º Quartile:  47.0
4º Quartile:  98.0
Ages above:  69.5 are outliers
```

In [14]:

```
print('Numerber of outliers: ', bank_client[bank_client['age'] > 69.6]
['age'].count())
print('Number of clients: ', len(bank_client))
#Outliers in %
print('Outliers are:', round(bank_client[bank_client['age'] > 69.6]['a
ge'].count()*100/len(bank_client),2), '%')
```

```
Numerber of outliers:  469
Number of clients:  41188
Outliers are: 1.14 %
```

In [15]:

```
# Calculating some values to evaluete this independent variable
print('MEAN:', round(bank_client['age'].mean(), 1))
# A low standard deviation indicates that the data points tend to be clo
```

```
se to the mean or expected value
# A high standard deviation indicates that the data points are scattered
print('STD :', round(bank_client['age'].std(), 1))
# I thing the best way to give a precisly insight abou dispersion is usi
ng the CV (coefficient variation) (STD/MEAN)*100
#    cv < 15%, low dispersion
#    cv > 30%, high dispersion
print('CV  :',round(bank_client['age'].std()*100/bank_client['age'].me
an(), 1), ', High middle dispersion')
```

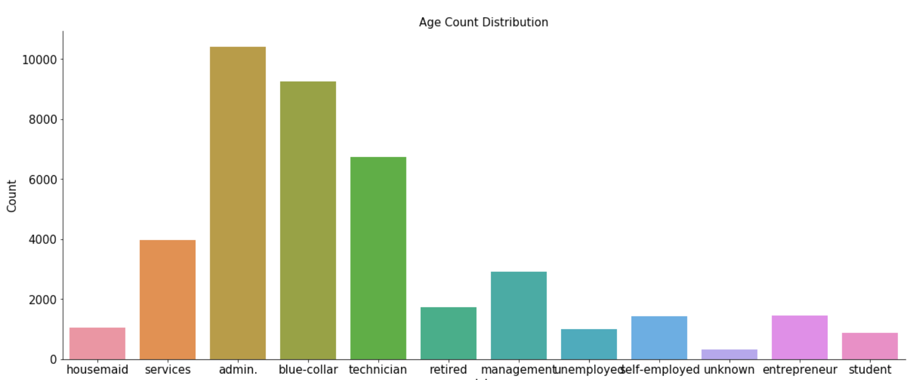
```
MEAN: 40.0
STD : 10.4
CV  : 26.0 , High middle dispersion
```

Conclusion about AGE, in my opinion due to almost high dispersion and just looking at this this graph we cannot conclude if age have a high effect to our variable y, need to keep searching for some pattern. high middle dispersion means we have people with all ages and maybe all of them can subscript a term deposit, or not. The outliers was calculated, so my thinking is fit the model with and without them

1.3. JOBS

In [16]:

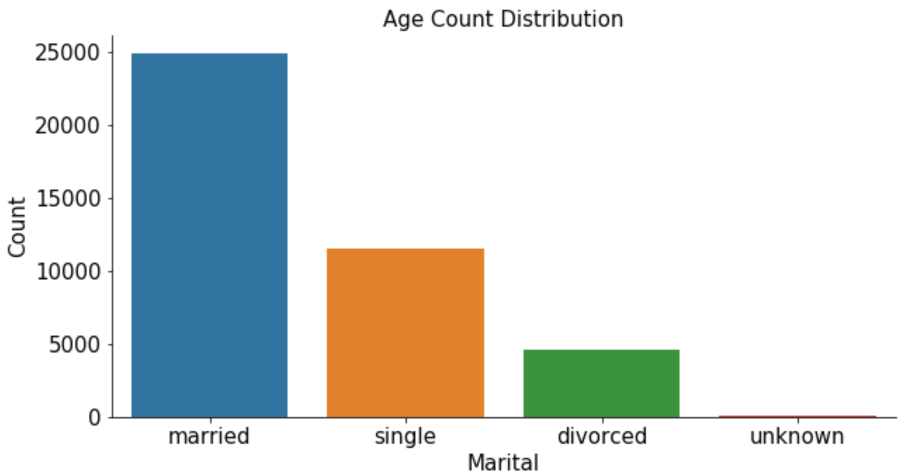
```
# What kind of jobs clients this bank have, if you cross jobs with defau
lt, loan or housing, there is no relation
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'job', data = bank_client)
ax.set_xlabel('Job', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```



1.4. MARITAL

In [17]:

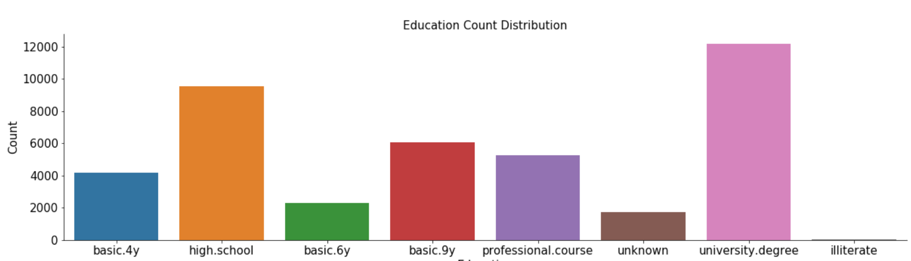
```
# What kind of 'marital clients' this bank have, if you cross marital wi
th default, loan or housing, there is no relation
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'marital', data = bank_client)
ax.set_xlabel('Marital', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```



1.5. EDUCATION

In [18]:

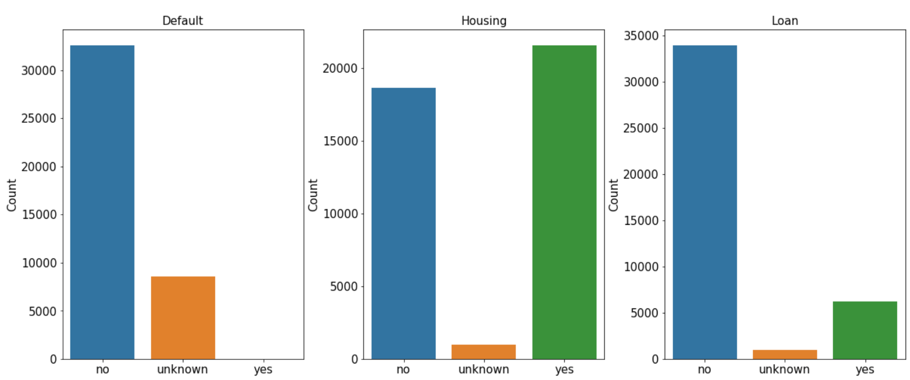
```
# What kind of 'education clients this bank have, if you cross education
with default, loan or housing, there is no relation
fig, ax = plt.subplots()
fig.set_size_inches(20, 5)
sns.countplot(x = 'education', data = bank_client)
ax.set_xlabel('Education', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Education Count Distribution', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```



1.6. DEFAULT, HOUSING, LOAN

In [19]:

```
# Default, has credit in default ?
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (2
0,8))
sns.countplot(x = 'default', data = bank_client, ax = ax1, order = ['n
o', 'unknown', 'yes'])
ax1.set_title('Default', fontsize=15)
ax1.set_xlabel('')
ax1.set_ylabel('Count', fontsize=15)
ax1.tick_params(labelsize=15)

# Housing, has housing loan ?
sns.countplot(x = 'housing', data = bank_client, ax = ax2, order = ['n
o', 'unknown', 'yes'])
ax2.set_title('Housing', fontsize=15)
ax2.set_xlabel('')
ax2.set_ylabel('Count', fontsize=15)
ax2.tick_params(labelsize=15)

# Loan, has personal loan ?
sns.countplot(x = 'loan', data = bank_client, ax = ax3, order = ['no',
'unknown', 'yes'])
ax3.set_title('Loan', fontsize=15)
ax3.set_xlabel('')
ax3.set_ylabel('Count', fontsize=15)
ax3.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.25)
```



In [20]:

```
print('Default:\n No credit in default:'     , bank_client[bank_client
['default'] == 'no']     ['age'].count(),
              '\n Unknown credit in default:', bank_client[bank_client
['default'] == 'unknown']['age'].count(),
              '\n Yes to credit in default:' , bank_client[bank_client
['default'] == 'yes']    ['age'].count())
```

```
Default:
 No credit in default: 32588
 Unknown credit in default: 8597
 Yes to credit in default: 3
```

In [21]:
```
print('Housing:\n No housing in loan:'     , bank_client[bank_client[
'housing'] == 'no']     ['age'].count(),
              '\n Unknown housing in loan:', bank_client[bank_client[
'housing'] == 'unknown']['age'].count(),
              '\n Yes to housing in loan:' , bank_client[bank_client[
'housing'] == 'yes']    ['age'].count())
```

```
Housing:
 No housing in loan: 18622
 Unknown housing in loan: 990
 Yes to housing in loan: 21576
```

In [22]:
```
print('Housing:\n No to personal loan:'     , bank_client[bank_client[
'loan'] == 'no']     ['age'].count(),
              '\n Unknown to personal loan:', bank_client[bank_client[
'loan'] == 'unknown']['age'].count(),
              '\n Yes to personal loan:'    , bank_client[bank_client[
'loan'] == 'yes']    ['age'].count())
```

```
Housing:
 No to personal loan: 33950
 Unknown to personal loan: 990
 Yes to personal loan: 6248
```

BANK CLIENTS CONCLUSION

The ages dont mean to much, has a medium dispersion and dont make sense relate with other variables will not tell any insight

Jobs, Marital and Education i think the best analisys is just the count of each variable, if we related with the other ones its is not conclusive, all this kind of variables has yes, unknown and no for loan, default and housing.

Default, loan and housing, its just to see the distribution of people.

1.7. Bank Client Categorical Treatment

- Jobs, Marital, Education, Default, Housing, Loan. Converting to continuous due the feature scaling will be apllyed later

In [23]:
```
# Label encoder order is alphabetical
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_client['job']      = labelencoder_X.fit_transform(bank_client['jo
b'])
bank_client['marital']  = labelencoder_X.fit_transform(bank_client['ma
rital'])
bank_client['education']= labelencoder_X.fit_transform(bank_client['ed
```

```
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 [24]:

```
#function to creat group of ages, this helps because we have 78 differente values here
def age(dataframe):
    dataframe.loc[dataframe['age'] <= 32, 'age'] = 1
    dataframe.loc[(dataframe['age'] > 32) & (dataframe['age'] <= 47), 'age'] = 2
    dataframe.loc[(dataframe['age'] > 47) & (dataframe['age'] <= 70), 'age'] = 3
    dataframe.loc[(dataframe['age'] > 70) & (dataframe['age'] <= 98), 'age'] = 4

    return dataframe

age(bank_client);
```

In [25]:

```
bank_client.head()
```

Out[25]:

	age	job	marital	education	default	housing	loan
0	3	3	1	0	0	0	0
1	3	7	1	3	1	0	0
2	2	7	1	3	0	2	0
3	2	0	1	1	0	0	0
4	3	7	1	3	0	0	2

Manualy way to convert Categorical in Continuous

bank_client['job'].replace(['housemaid' , 'services' , 'admin.' , 'blue-collar' , 'technician', 'retired' , 'management', 'unemployed', 'self-employed', 'unknown' , 'entrepreneur', 'student'] , [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], inplace=True)

bank_client['education'].replace(['basic.4y' , 'high.school', 'basic.6y', 'basic.9y', 'professional.course', 'unknown' , 'university.degree' , 'illiterate'], [1, 2, 3, 4, 5, 6, 7, 8], inplace=True)

bank_client['marital'].replace(['married', 'single', 'divorced', 'unknown'], [1, 2, 3, 4], inplace=True)

bank_client['default'].replace(['yes', 'no', 'unknown'],[1, 2, 3], inplace=True)

bank_client['housing'].replace(['yes', 'no', 'unknown'],[1, 2, 3], inplace=True)

bank_client['loan'].replace(['yes', 'no', 'unknown'],[1, 2, 3], inplace=True)

A way to Converting Categorical variables using dummies if you judge necessary

bank_client = pd.get_dummies(data = bank_client, columns = ['job'] , prefix = ['job'] , drop_first = True)

bank_client = pd.get_dummies(data = bank_client, columns = ['job'] , prefix = ['job'] , drop_first = True)

bank_client = pd.get_dummies(data = bank_client, columns = ['marital'] , prefix = ['marital'] , drop_first = True)

bank_client = pd.get_dummies(data = bank_client, columns = ['education'], prefix = ['education'], drop_first = True)

bank_client = pd.get_dummies(data = bank_client, columns = ['default'] , prefix = ['default'] , drop_first = True)

bank_client = pd.get_dummies(data = bank_client, columns = ['housing'] , prefix = ['housing'] , drop_first = True)

bank_client = pd.get_dummies(data = bank_client, columns = ['loan'] , prefix = ['loan'] , drop_first = True)

In [26]:

```
print(bank_client.shape)
bank_client.head()
```

```
(41188, 7)
```

Out[26]:

	age	job	marital	education	default	housing	loan
0	3	3	1	0	0	0	0
1	3	7	1	3	1	0	0
2	2	7	1	3	0	2	0
3	2	0	1	1	0	0	0
4	3	7	1	3	0	0	2

2. Related with the last contact of the current campaign

- Treat categorical, see those values
- group continuous variables if necessary

In [27]:

```
# Slicing DataFrame to treat separately, make things more easy
bank_related = bank.iloc[: , 7:11]
bank_related.head()
```

Out[27]:

	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 [28]:

```
bank_related.isnull().any()
```

Out[28]:

```
contact        False
month          False
day_of_week    False
duration       False
dtype: bool
```

In [29]:

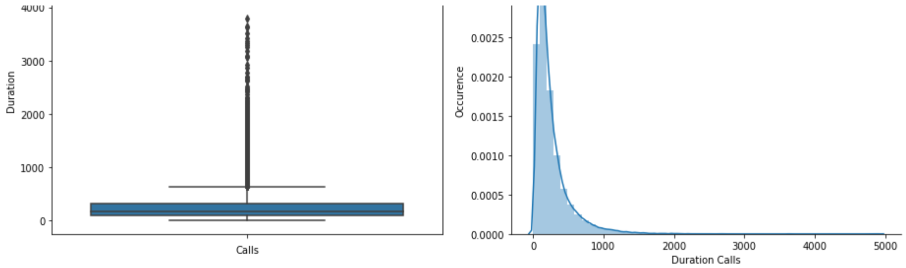
```
print("Kind of Contact: \n", bank_related['contact'].unique())
print("\nWhich monthis this campaing work: \n", bank_related['month'].
unique())
print("\nWhich days of week this campaing work: \n", bank_related['day
_of_week'].unique())
```

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

Which monthis this campaing work:
 ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']

Which days of week this campaing work:
 ['mon' 'tue' 'wed' 'thu' 'fri']
```

2.1 Duration

In [30]:

```
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5
))
sns.boxplot(x = 'duration', data = bank_related, orient = 'v', ax = ax
1)
ax1.set_xlabel('Calls', fontsize=10)
ax1.set_ylabel('Duration', fontsize=10)
ax1.set_title('Calls Distribution', fontsize=10)
ax1.tick_params(labelsize=10)

sns.distplot(bank_related['duration'], ax = ax2)
sns.despine(ax = ax2)
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)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



PLease note: duration is different from age, Age has 78 values and Duration has 1544 different values

In [31]:

```
print("Max duration  call in minutes:  ", round((bank_related['duratio
n'].max()/60),1))
print("Min duration  call in minutes:   ", round((bank_related['durati
on'].min()/60),1))
print("Mean duration call in minutes:   ", round((bank_related['durati
on'].mean()/60),1))
print("STD duration  call in minutes:   ", round((bank_related['durati
on'].std()/60),1))
# Std close to the mean means that the data values are close to the mean
```

```
Max duration  call in minutes:   82.0
Min duration  call in minutes:    0.0
Mean duration call in minutes:    4.3
STD duration  call in minutes:    4.3
```

In [32]:

```
# Quartiles
print('1º Quartile: ', bank_related['duration'].quantile(q = 0.25))
print('2º Quartile: ', bank_related['duration'].quantile(q = 0.50))
print('3º Quartile: ', bank_related['duration'].quantile(q = 0.75))
print('4º Quartile: ', bank_related['duration'].quantile(q = 1.00))
#Calculate the outliers:
  # Interquartile range, IQR = Q3 - Q1
  # lower 1.5*IQR whisker = Q1 - 1.5 * IQR
  # Upper 1.5*IQR whisker = Q3 + 1.5 * IQR

print('Duration calls above: ', bank_related['duration'].quantile(q =
0.75) +
                      1.5*(bank_related['duration'].quantile(q = 0.75)
- bank_related['duration'].quantile(q = 0.25)), 'are outliers')
```

```
1º Quartile:  102.0
2º Quartile:  180.0
3º Quartile:  319.0
4º Quartile:  4918.0
Duration calls above:  644.5 are outliers
```

In [33]:

```
print('Numerber of outliers: ', bank_related[bank_related['duration']
> 644.5]['duration'].count())
```

```
print('Number of clients: ', len(bank_related))
#Outliers in %
print('Outliers are:', round(bank_related[bank_related['duration'] > 6
44.5]['duration'].count()*100/len(bank_related),2), '%')
```

```
Numerber of outliers:  2963
Number of clients:  41188
Outliers are: 7.19 %
```

In [34]:

```
# Look, if the call duration is iqual to 0, then is obviously that this
 person didn't subscribed,
# THIS LINES NEED TO BE DELETED LATER
bank[(bank['duration'] == 0)]
```

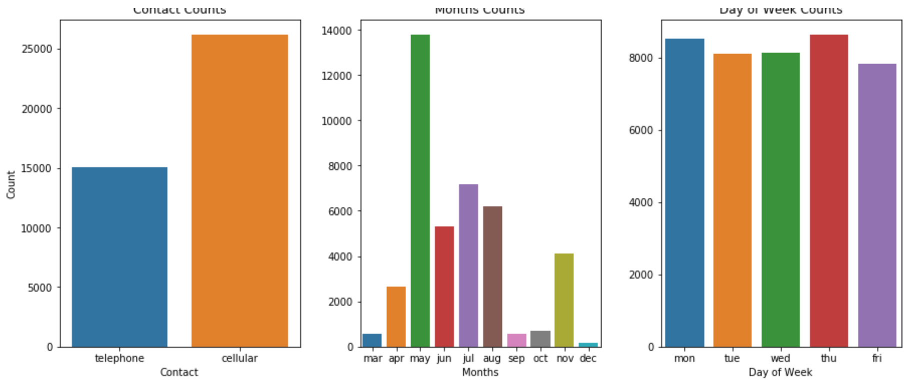
Out[34]:

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

2.2 Contact, Month, Day of Week

In [35]:

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (1
5,6))
sns.countplot(bank_related['contact'], ax = ax1)
ax1.set_xlabel('Contact', fontsize = 10)
ax1.set_ylabel('Count', fontsize = 10)
ax1.set_title('Contact Counts')
ax1.tick_params(labelsize=10)

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

sns.countplot(bank_related['day_of_week'], ax = ax3)
ax3.set_xlabel('Day of Week', fontsize = 10)
ax3.set_ylabel('')
ax3.set_title('Day of Week Counts')
ax3.tick_params(labelsize=10)

plt.subplots_adjust(wspace=0.25)
```



In [36]:

```
print('Ages above: ', bank_related['duration'].quantile(q = 0.75) +
                      1.5*(bank_related['duration'].quantile(q = 0.75)
- bank_related['duration'].quantile(q = 0.25)), 'are outliers')
```

```
Ages above:  644.5 are outliers
```

In [37]:

```
bank_related[bank_related['duration'] > 640].count()
```

Out[37]:

```
contact        3008
month          3008
day_of_week    3008
duration       3008
dtype: int64
```

2.1 Contact, Month, Day of Week treatment

In [38]:

```
# Label encoder order is alphabetical
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_related['contact']     = labelencoder_X.fit_transform(bank_relate
d['contact'])
bank_related['month']       = labelencoder_X.fit_transform(bank_relate
d['month'])
bank_related['day_of_week'] = labelencoder_X.fit_transform(bank_relate
d['day_of_week'])
```

A way to Converting Categorical variables using dummies if you judge necessary

```
bank_related = pd.get_dummies(data = bank_related, prefix = ['contact'] , columns = ['contact'] ,
drop_first = True)

bank_related = pd.get_dummies(data = bank_related, prefix = ['month'] , columns = ['month'] ,
drop_first = True)

bank_related = pd.get_dummies(data = bank_related, prefix = ['day_of_week'], columns =
['day_of_week'], drop_first = True)
```

```
['day_of_week'], drop_first = True)
```

In [39]:

```
bank_related.head()
```

Out[39]:

	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

In [40]:

```
def duration(data):

    data.loc[data['duration'] <= 102, 'duration'] = 1
    data.loc[(data['duration'] > 102) & (data['duration'] <= 180)  ,
'duration']    = 2
    data.loc[(data['duration'] > 180) & (data['duration'] <= 319)  ,
'duration']   = 3
    data.loc[(data['duration'] > 319) & (data['duration'] <= 644.5),
'duration'] = 4
    data.loc[data['duration']  > 644.5, 'duration'] = 5

    return data
duration(bank_related);
```

In [41]:

```
bank_related.head()
```

Out[41]:

	contact	month	day_of_week	duration
0	1	6	1	3
1	1	6	1	2
2	1	6	1	3
3	1	6	1	2
4	1	6	1	3

Social and economic context attributes

In [42]:

```
bank_se = bank.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.i
dx', 'euribor3m', 'nr.employed']]
```

```
bank_se.head()
```

Out[42]:

	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

Other attributes

In [43]:

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

Out[43]:

	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 [44]:

```
bank_o['poutcome'].unique()
```

Out[44]:

```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

In [45]:

```
bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2
,3], inplace  = True)
```

Model

In [46]:

```
bank_final= pd.concat([bank_client, bank_related, bank_se, bank_o], ax
is = 1)
bank_final = bank_final[['age', 'job', 'marital', 'education', 'defaul
t', 'housing', 'loan',
                     'contact', 'month', 'day_of_week', 'duration', 'e
mp.var.rate', 'cons.price.idx',
```

```
mp.var.rate', 'cons.price.idx',
                     'cons.conf.idx', 'euribor3m', 'nr.employed', 'cam
paign', 'pdays', 'previous', 'poutcome']]
bank_final.shape
```

Out[46]:

```
(41188, 20)
```

In [47]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(bank_final, y, tes
t_size = 0.1942313295, random_state = 101)

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
```

In [48]:

```
X_train.head()
```

Out[48]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	dur
38912	3	5	1	6	0	2	0	0	7	4	5
9455	2	7	1	5	1	0	0	1	4	0	2
14153	1	4	1	6	0	2	0	0	3	1	5
25021	3	6	1	6	0	2	0	0	7	3	1
30911	2	5	0	0	0	2	2	0	6	3	3

In [49]:

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [50]:

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
logpred = logmodel.predict(X_test)


print(confusion_matrix(y_test, logpred))
print(round(accuracy_score(y_test, logpred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs
=1, scoring = 'accuracy').mean())
```

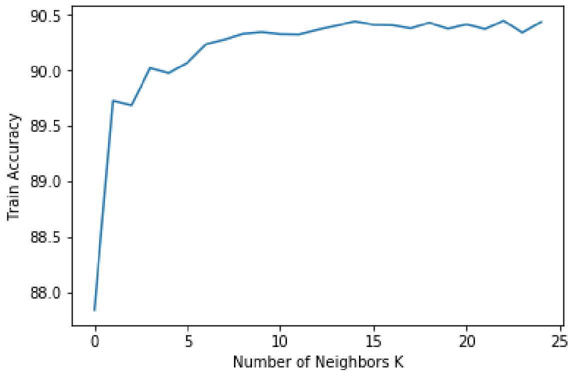
```
[[6909  164]
 [ 598  329]]
90.0
```

In [51]:

```
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier

X_trainK, X_testK, y_trainK, y_testK = train_test_split(bank_final, y,
test_size = 0.2, random_state = 101)

#Neighbors
neighbors = np.arange(0,25)

#Create empty list that will hold cv scores
cv_scores = []

#Perform 10-fold cross validation on training set for odd values of k:
for k in neighbors:
    k_value = k+1
    knn = KNeighborsClassifier(n_neighbors = k_value, weights='unifor
m', p=2, metric='euclidean')
    kfold = model_selection.KFold(n_splits=10, random_state=123)
    scores = model_selection.cross_val_score(knn, X_trainK, y_trainK,
cv=kfold, scoring='accuracy')
    cv_scores.append(scores.mean()*100)
    print("k=%d %0.2f (+/- %0.2f)" % (k_value, scores.mean()*100, scor
es.std()*100))

optimal_k = neighbors[cv_scores.index(max(cv_scores))]
print ("The optimal number of neighbors is %d with %0.1f%%" % (optimal
_k, cv_scores[optimal_k]))

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

```
k=1 87.84 (+/- 0.59)
k=2 89.73 (+/- 0.50)
k=3 89.69 (+/- 0.49)
k=4 90.02 (+/- 0.51)
k=5 89.98 (+/- 0.41)
k=6 90.07 (+/- 0.47)
k=7 90.24 (+/- 0.41)
k=8 90.28 (+/- 0.48)
k=9 90.33 (+/- 0.46)
k=10 90.35 (+/- 0.49)
k=11 90.33 (+/- 0.51)
k=12 90.32 (+/- 0.59)
k=13 90.37 (+/- 0.51)
k=14 90.40 (+/- 0.48)
k=15 90.44 (+/- 0.47)
k=16 90.41 (+/- 0.50)
k=17 90.41 (+/- 0.50)
k=18 90.38 (+/- 0.52)
k=19 90.43 (+/- 0.45)
k=20 90.38 (+/- 0.48)
k=21 90.42 (+/- 0.46)
k=22 90.37 (+/- 0.48)
k=23 90.45 (+/- 0.44)
k=24 90.34 (+/- 0.49)
k=25 90.44 (+/- 0.47)
```

```
The optimal number of neighbors is 22 with 90.4%
```



In [52]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=22)
knn.fit(X_train, y_train)
knnpred = knn.predict(X_test)

print(confusion_matrix(y_test, knnpred))
print(round(accuracy_score(y_test, knnpred),2)*100)
KNNCV = (cross_val_score(knn, X_train, y_train, cv=k_fold, n_jobs=1, s
coring = 'accuracy').mean())
```

```
[[6962  111]
 [ 684  243]]
90.0
```

In [53]:

```
from sklearn.svm import SVC
svc= SVC(kernel = 'sigmoid')
svc.fit(X_train, y_train)
svcpred = svc.predict(X_test)
print(confusion_matrix(y_test, svcpred))
print(round(accuracy_score(y_test, svcpred),2)*100)
SVCCV = (cross_val_score(svc, X_train, y_train, cv=k_fold, n_jobs=1, s
coring = 'accuracy').mean())
```

```
[[6531  542]
 [ 584  343]]
86.0
```

In [54]:

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='gini') #criterion = entopy, g
ini
dtree.fit(X_train, y_train)
dtreepred = dtree.predict(X_test)

print(confusion_matrix(y_test, dtreepred))
print(round(accuracy_score(y_test, dtreepred),2)*100)
DTREECV = (cross_val_score(dtree, X_train, y_train, cv=k_fold, n_jobs=
1, scoring = 'accuracy').mean())
```

```
[[6609  464]
 [ 474  453]]
88.0
```

In [55]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 200)#criterion = entopy,gin
i
rfc.fit(X_train, y_train)
rfcpred = rfc.predict(X_test)

print(confusion_matrix(y_test, rfcpred ))
print(round(accuracy_score(y_test, rfcpred),2)*100)
RFCCV = (cross_val_score(rfc, X_train, y_train, cv=k_fold, n_jobs=1, s
coring = 'accuracy').mean())
```

```
[[6797  276]
 [ 491  436]]
90.0
```

In [56]:

```
from sklearn.naive_bayes import GaussianNB
gaussiannb= GaussianNB()
gaussiannb.fit(X_train, y_train)
gaussiannbpred = gaussiannb.predict(X_test)
probs = gaussiannb.predict(X_test)

print(confusion_matrix(y_test, gaussiannbpred ))
print(round(accuracy_score(y_test, gaussiannbpred),2)*100)
GAUSIAN = (cross_val_score(gaussiannb, X_train, y_train, cv=k_fold, n_
jobs=1, scoring = 'accuracy').mean())
```

```
[[6272  801]
 [ 417  510]]
85.0
```

In [57]:

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
xgbprd = xgb.predict(X_test)

print(confusion_matrix(y_test, xgbprd ))
print(round(accuracy_score(y_test, xgbprd),2)*100)
XGB = (cross_val_score(estimator = xgb, X = X_train, y = y_train, cv =
10).mean())
```

```
[[6858  215]
 [ 512  415]]
91.0
```

In [58]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbk = GradientBoostingClassifier()
gbk.fit(X_train, y_train)
gbkpred = gbk.predict(X_test)
print(confusion_matrix(y_test, gbkpred ))
print(round(accuracy_score(y_test, gbkpred),2)*100)
GBKCV = (cross_val_score(gbk, X_train, y_train, cv=k_fold, n_jobs=1, s
coring = 'accuracy').mean())
```

```
[[6826  247]
 [ 460  467]]
91.0
```

In [59]:

```
models = pd.DataFrame({
                'Models': ['Random Forest Classifier', 'Decision Tree
 Classifier', 'Support Vector Machine',
                           'K-Near Neighbors', 'Logistic Model', 'Gaus
ian NB', 'XGBoost', 'Gradient Boosting'],
                'Score':  [RFCCV, DTREECV, SVCCV, KNNCV, LOGCV, GAUSIA
N, XGB, GBKCV]})

models.sort_values(by='Score', ascending=False)
```

Out[59]:

	Models	Score
7	Gradient Boosting	0.914306
6	XGBoost	0.913584
4	Logistic Model	0.909726
0	Random Forest Classifier	0.909365
3	K-Near Neighbors	0.904815
1	Decision Tree Classifier	0.884054
2	Support Vector Machine	0.855640
5	Gausian NB	0.844432

Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test.

A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system:

.90-1 = excellent (A)

.80-.90 = good (B)

.70-.80 = fair (C)

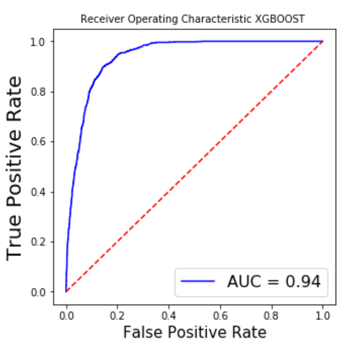
.60-.70 = poor (D)

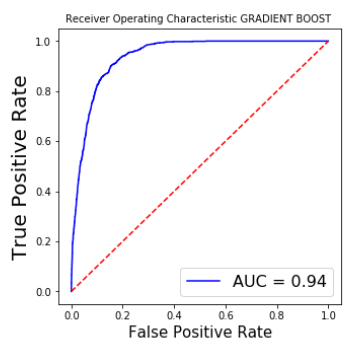
.50-.60 = fail (F)

In [60]:

```
# XGBOOST ROC/ AUC , BEST MODEL
from sklearn import metrics
```

```
fig, (ax, ax1) = plt.subplots(nrows = 1, ncols = 2, figsize = (15,5))
probs = xgb.predict_proba(X_test)
preds = probs[:,1]
fprxgb, tprxgb, thresholdxgb = metrics.roc_curve(y_test, preds)
roc_aucxgb = metrics.auc(fprxgb, tprxgb)

ax.plot(fprxgb, tprxgb, 'b', label = 'AUC = %0.2f' % roc_aucxgb)
ax.plot([0, 1], [0, 1],'r--')
ax.set_title('Receiver Operating Characteristic XGBOOST ',fontsize=10)
ax.set_ylabel('True Positive Rate',fontsize=20)
ax.set_xlabel('False Positive Rate',fontsize=15)
ax.legend(loc = 'lower right', prop={'size': 16})

#Gradient
probs = gbk.predict_proba(X_test)
preds = probs[:,1]
fprgbk, tprgbk, thresholdgbk = metrics.roc_curve(y_test, preds)
roc_aucgbk = metrics.auc(fprgbk, tprgbk)

ax1.plot(fprgbk, tprgbk, 'b', label = 'AUC = %0.2f' % roc_aucgbk)
ax1.plot([0, 1], [0, 1],'r--')
ax1.set_title('Receiver Operating Characteristic GRADIENT BOOST ',font
size=10)
ax1.set_ylabel('True Positive Rate',fontsize=20)
ax1.set_xlabel('False Positive Rate',fontsize=15)
ax1.legend(loc = 'lower right', prop={'size': 16})

plt.subplots_adjust(wspace=1)
```



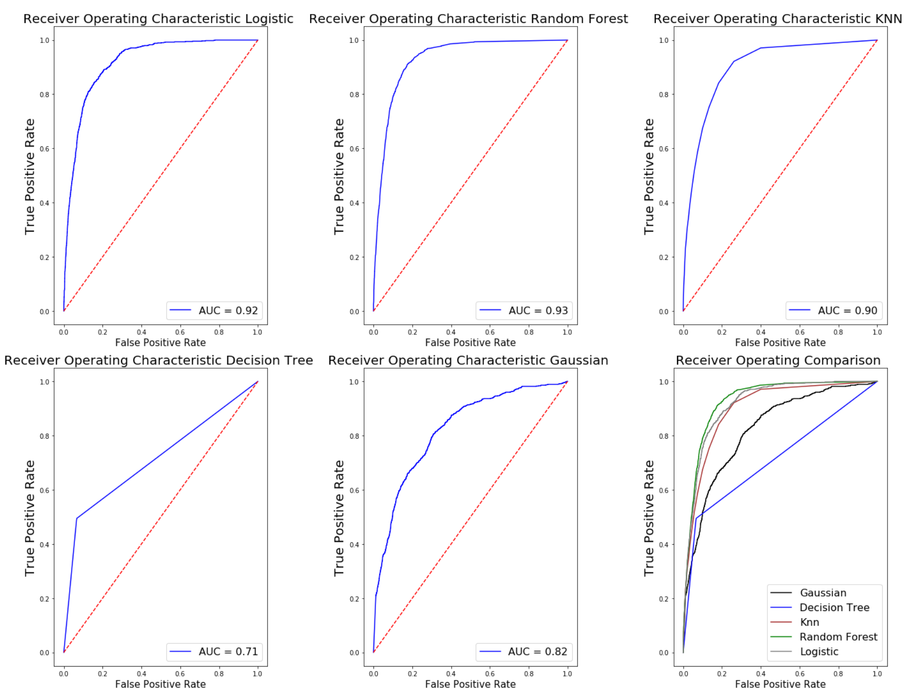


In [61]:

```
#fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows = 2, ncols = 3, fig
size = (15, 4))
fig, ax_arr = plt.subplots(nrows = 2, ncols = 3, figsize = (20,15))

#LOGMODEL
probs = logmodel.predict_proba(X_test)
preds = probs[:,1]
fprlog, tprlog, thresholdlog = metrics.roc_curve(y_test, preds)
roc_auclog = metrics.auc(fprlog, tprlog)

ax_arr[0,0].plot(fprlog, tprlog, 'b', label = 'AUC = %0.2f' % roc_aucl
og)
ax_arr[0,0].plot([0, 1], [0, 1],'r--')
ax_arr[0,0].set_title('Receiver Operating Characteristic Logistic ',fo
ntsize=20)
```

```
ax_arr[0,0].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,0].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,0].legend(loc = 'lower right', prop={'size': 16})

#RANDOM FOREST --------------------
probs = rfc.predict_proba(X_test)
preds = probs[:,1]
fprrfc, tprrfc, thresholdrfc = metrics.roc_curve(y_test, preds)
roc_aucrfc = metrics.auc(fprrfc, tprrfc)

ax_arr[0,1].plot(fprrfc, tprrfc, 'b', label = 'AUC = %0.2f' % roc_aucr
fc)
ax_arr[0,1].plot([0, 1], [0, 1],'r--')
ax_arr[0,1].set_title('Receiver Operating Characteristic Random Forest
',fontsize=20)
ax_arr[0,1].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,1].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,1].legend(loc = 'lower right', prop={'size': 16})

#KNN----------------------
probs = knn.predict_proba(X_test)
preds = probs[:,1]
fprknn, tprknn, thresholdknn = metrics.roc_curve(y_test, preds)
roc_aucknn = metrics.auc(fprknn, tprknn)

ax_arr[0,2].plot(fprknn, tprknn, 'b', label = 'AUC = %0.2f' % roc_auck
nn)
ax_arr[0,2].plot([0, 1], [0, 1],'r--')
ax_arr[0,2].set_title('Receiver Operating Characteristic KNN ',fontsiz
e=20)
ax_arr[0,2].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,2].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,2].legend(loc = 'lower right', prop={'size': 16})

#DECISION TREE ---------------------
probs = dtree.predict_proba(X_test)
preds = probs[:,1]
fprdtree, tprdtree, thresholddtree = metrics.roc_curve(y_test, preds)
roc_aucdtree = metrics.auc(fprdtree, tprdtree)

ax_arr[1,0].plot(fprdtree, tprdtree, 'b', label = 'AUC = %0.2f' % roc_
aucdtree)
ax_arr[1,0].plot([0, 1], [0, 1],'r--')
ax_arr[1,0].set_title('Receiver Operating Characteristic Decision Tree
',fontsize=20)
ax_arr[1,0].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,0].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,0].legend(loc = 'lower right', prop={'size': 16})

#GAUSSIAN ---------------------
probs = gaussiannb.predict_proba(X_test)
preds = probs[:,1]
fprgau, tprgau, thresholdgau = metrics.roc_curve(y_test, preds)
roc_aucgau = metrics.auc(fprgau, tprgau)

ax_arr[1,1].plot(fprgau, tprgau, 'b', label = 'AUC = %0.2f' % roc_aucg
au)
ax_arr[1,1].plot([0, 1], [0, 1],'r--')
ax_arr[1,1].set_title('Receiver Operating Characteristic Gaussian ',fo
```

```
ax_arr[1,1].set_title('Receiver Operating Characteristic Gaussian ',fo
ntsize=20)
ax_arr[1,1].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,1].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,1].legend(loc = 'lower right', prop={'size': 16})

#ALL PLOTS ----------------------------------
ax_arr[1,2].plot(fprgau, tprgau, 'b', label = 'Gaussian', color='blac
k')
ax_arr[1,2].plot(fprdtree, tprdtree, 'b', label = 'Decision Tree', col
or='blue')
ax_arr[1,2].plot(fprknn, tprknn, 'b', label = 'Knn', color='brown')
ax_arr[1,2].plot(fprrfc, tprrfc, 'b', label = 'Random Forest', color=
'green')
ax_arr[1,2].plot(fprlog, tprlog, 'b', label = 'Logistic', color='grey'
)
ax_arr[1,2].set_title('Receiver Operating Comparison ',fontsize=20)
ax_arr[1,2].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,2].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,2].legend(loc = 'lower right', prop={'size': 16})

plt.subplots_adjust(wspace=0.2)
plt.tight_layout()
```



ANALYZING THE RESULTS

So now we have to decide which one is the best model, and we have two types of wrong values:

- False Positive, means the client do NOT SUBSCRIBED to term deposit, but the model thinks he did.
- False Negative, means the client SUBSCRIBED to term deposit, but the model said he dont.

In my opinion:

- The first one its most harmful, because we think that we already have that client but we dont and maybe we lost him in other future campaings
- The second its not good but its ok, we have that client and in the future we'll discovery that in truth he's already our client

So, our objective here, is to find the best model by confusion matrix with the lowest False Positive as possible.

Obs1 - lets go back and look the best confusion matrix that attend this criteria Obs2 - i'll do the math manualy to be more visible and understanding

In [62]:

```
from sklearn.metrics import classification_report
```

In [63]:

```
print('KNN Confusion Matrix\n', confusion_matrix(y_test, knnpred))
```

```
KNN Confusion Matrix
 [[6962  111]
 [ 684  243]]
```

In [64]:

```
print('KNN Reports\n',classification_report(y_test, knnpred))
```

```
KNN Reports
              precision    recall  f1-score   support

          0       0.91      0.98      0.95      7073
          1       0.69      0.26      0.38       927

avg / total       0.88      0.90      0.88      8000
```

Ok, now lets go deep into this values

CHOOSED MODEL ANALYSIS

RECALL

In [65]:

```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "http://i68.tinypic.com/iyj4fc.jpg")
```

Out[65]:

Recall - Specificity

TN / (TN + FP) [MATRIX LINE 1]

- For all NEGATIVE(0) **REAL** VALUES how much we predict correct ?
- other way to understand, our real test set has 7163+116 = 7279 clients that didin't subscribe(0), and our model predict 98% correct or 7163 correct and 116 incorrect

In [66]:

```
print(round(7163 /(7163 + 116),2))
```

```
0.98
```

Recall - Sensitivity

TP / (TP + FN) [MATRIX LINE 2]

- For all POSITIVE(1) **REAL** VALUES how much we predict correct ?
- other way to understand, our real test set has 706 + 253 = 959 clients that subscribe(1), and our model predict 26% correct or 253 correct and 706 incorrect, **BUT REMEMBER, its best we miss by False negative instead of False Positive**

In [67]:

```
print(round(253 / (253 + 706  ),2))
print(round(metrics.recall_score(y_test, knnpred),2))
```

```
0.26
0.26
```

PRECISION

Precision

TN / (TN + FN) [MATRIX COLUMN 1]

- For all NEGATIVE(0) **PREDICTIONS** by our model, how much we predict correct ?
- other way to understand, our model pointed 7163 + 706 = 7869 clients that didin't subscribe(0), and our model predict 91% correct or 7163 correct and 706 incorrect

In [68]:
```
print(round(7163 / (7163 + 706),2))
```

```
0.91
```

Precision

TN / (TN + FN) [MATRIX COLUMN 1]

- For all POSITIVE(1) **PREDICTIONS** by our model, how much we predict correct ?
- other way to understand, our model pointed 116 + 253 = 369 clients that subscribe(1), and our model predict 69% correct or 253 correct and 116 incorrect

In [69]:
```
print(round(253 / (253 + 116),2))
print(round(metrics.precision_score(y_test, knnpred),2))
```

```
0.69
0.69
```

F1-SCORE

- F1-Score is a "median" of Recall and Precision, consider this when you want a balance between this metrics

Data

Data Sources

Bank Marketing

Bank Marketing
Bank Marketing, UCI Dataset
Last Updated: a year ago (Version 1)

bank-additional-full.csv 21 columns

bank-additional-names.txt

About this Dataset

Bank Marketing

Abstract: The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

Data Set Information: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Attribute Information:

Bank client data:

- Age (numeric)
- Job : type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
- Marital : marital status (categorical: 'divorced', 'married', 'single', 'unknown' ; note: 'divorced' means divorced or widowed)
- Education (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')
- Default: has credit in default? (categorical: 'no', 'yes', 'unknown')

Comments (9)

Sort by

All Comments | Hotness

Click here to comment...

lulupan2018 • Posted on Latest Version • 10 months ago • Options • Reply 4

think Duration shall be removed from the features

Suyash Khare • Posted on Latest Version • a year ago • Options • Reply 1

brilliant!

Rajan Sharma • Posted on Latest Version • 2 months ago • Options • Reply 0

Hi @henriqueyamahata, awesome work but i have one doubt above you tranform Jobs, Marital, Education, Default, Housing, Loan categorical column into numerical via label encoder. but after converting these column via label encoder model will misunderstand the data to be in some kind of order,0<1<2. But this isn't the case at all.
I think one hot encoding is useful in this case. please correct me if i am wrong.

Ashish • Posted on Latest Version • 2 months ago • Options • Reply 0

AWESOME!!!!

Suhas Shastry • Posted on Latest Version • 4 months ago • Options • Reply 0

Excellent work.

Humma04 • Posted on Latest Version • 7 months ago • Options • Reply 0

Can anyone please help me to do any machine learning technique like regression, classification or clustering in this dataset of bank marketing?

Kiruthika94 • Posted on Latest Version • a year ago • Options • Reply 0

Hi, Its really great work . I am actually trying to use DBSCAN to remove outliers. can u post that also.?

Dushyant Dhankar • Posted on Latest Version • a year ago • Options • Reply 0

can you please tell me how did you check that there is no relation between variables?

Henrique Yamah... Kernel Author • Posted on Latest Version • a year ago • Options • Reply 2

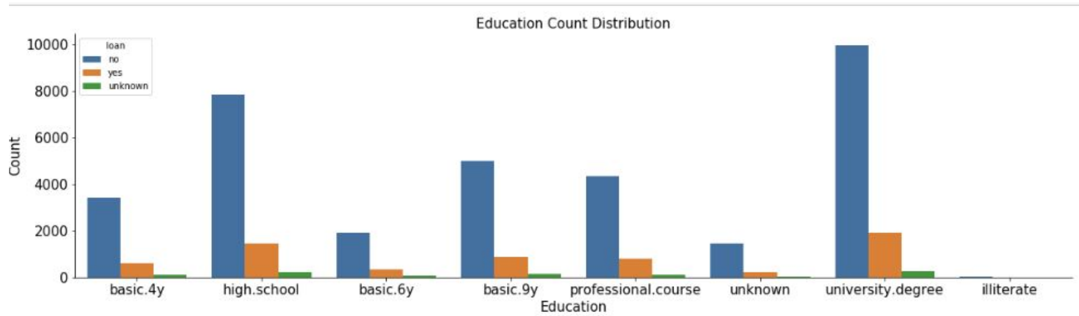
Version 6
6 commits

Thanks Dushyant Dhankar,

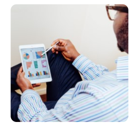
For example, if you plot this graph

```
fig, ax = plt.subplots()
fig.set_size_inches(20, 5)
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=15)
sns.despine()
```

you can see all peoples, independent of education level, has loans or dont have loans.



Similar Kernels



Notebook Data Comments

Our Team Terms Privacy Contact/Support