Attribute Information:

1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - education
(categorical:'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknov
5 - default: has credit in default? (categorical: 'no','yes','unknown')
6 - housing: has housing loan? (categorical: 'no','yes','unknown')
7 - loan: has personal loan? (categorical: 'no','yes','unknown')
8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric)(e.g., if duration=0 then y='no')
12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - 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)

14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - 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#

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

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	pre
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	pre
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	

5 rows × 21 columns

In [3]:

```
#attribute information
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  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', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

1. Pre-proccessing of client data

- **To make things more clear, we are going to creat a new dataset 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

	age	job	marital	education	default	housing	loan
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

1.1 Age

- **Trying to find some insights about these variables**

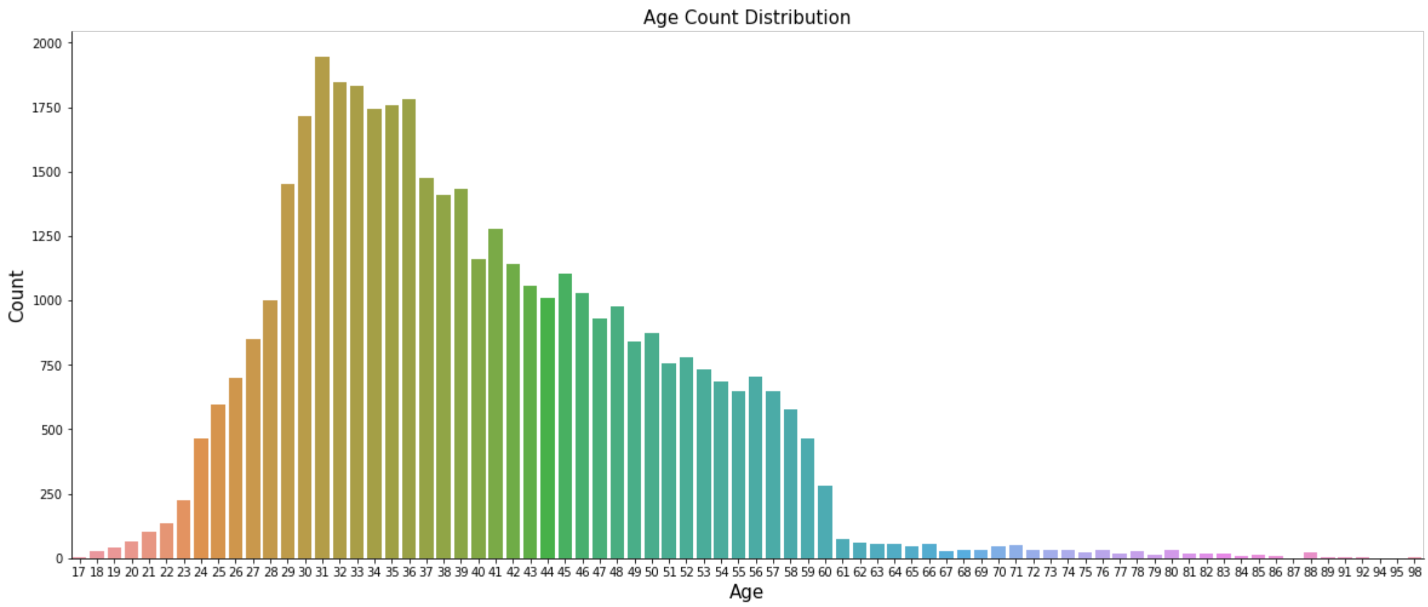
In [6]:

```
#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
```

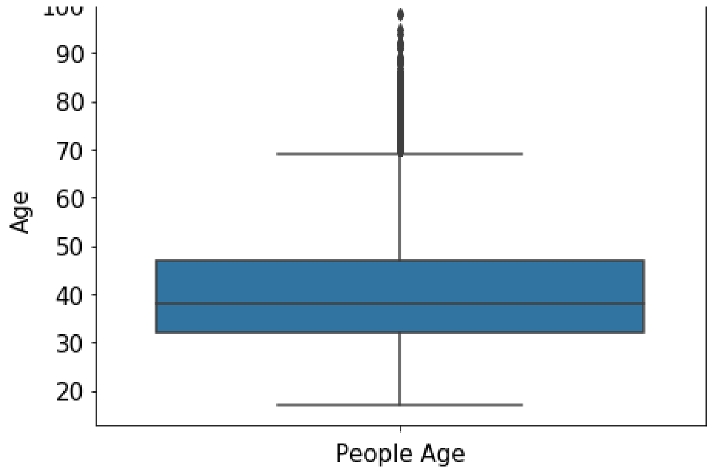
In [7]:

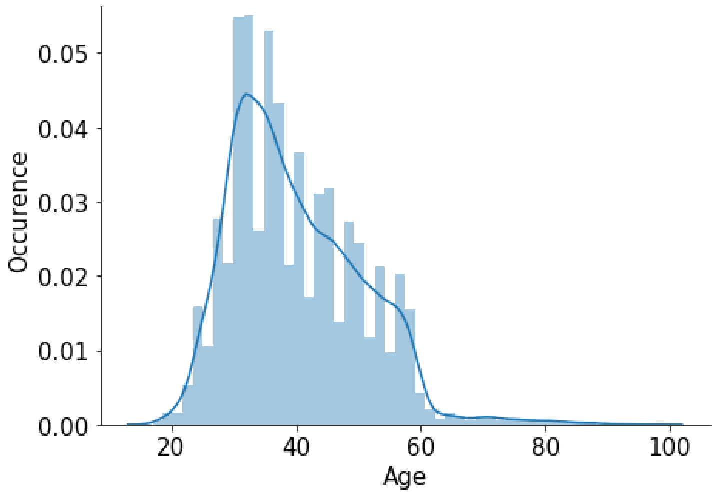
```
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'age', data = bank_client)
ax.set_xlabel('Age', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
sns.despine()
```



In [8]:

```
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_title('Age x Ocucurence', fontsize=15)
ax2.tick_params(labelsize=15)

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





In [9]:

```
# 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 close 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))
```

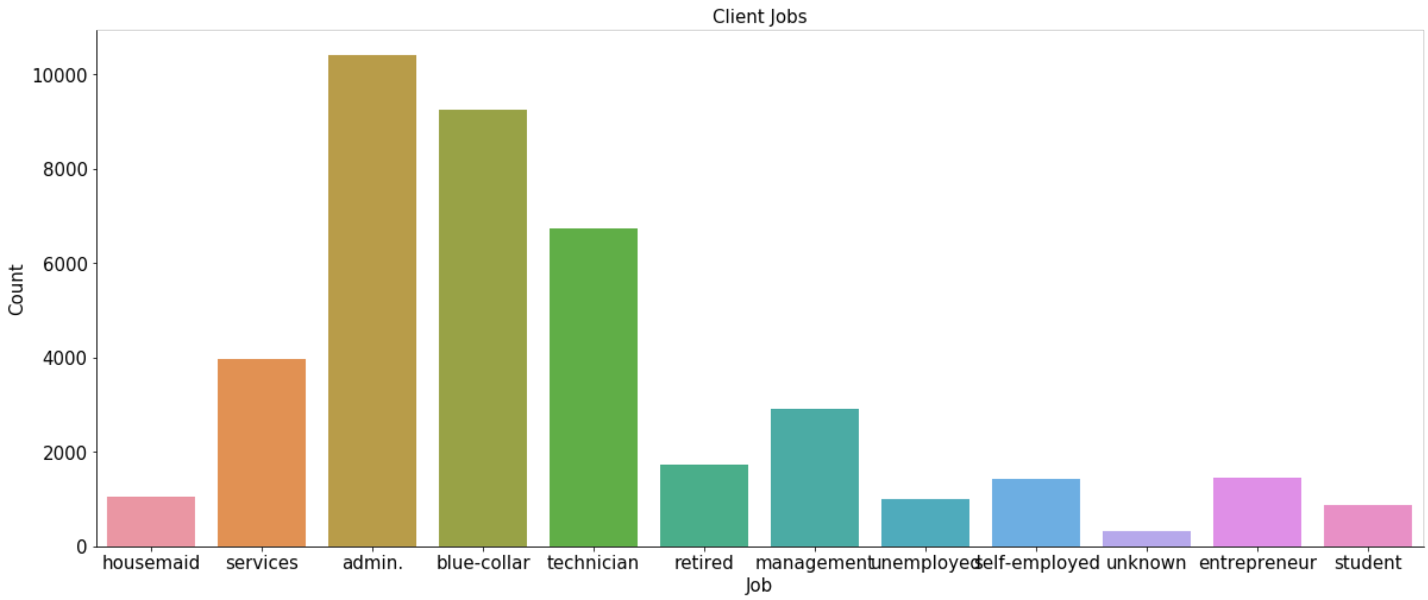
```
MEAN: 40.0
STD : 10.4
```

Conclusion about 'AGE' variable: according to the graph we find that age can be related to the output variable 'y'.

1.2 JOBS

In [10]:

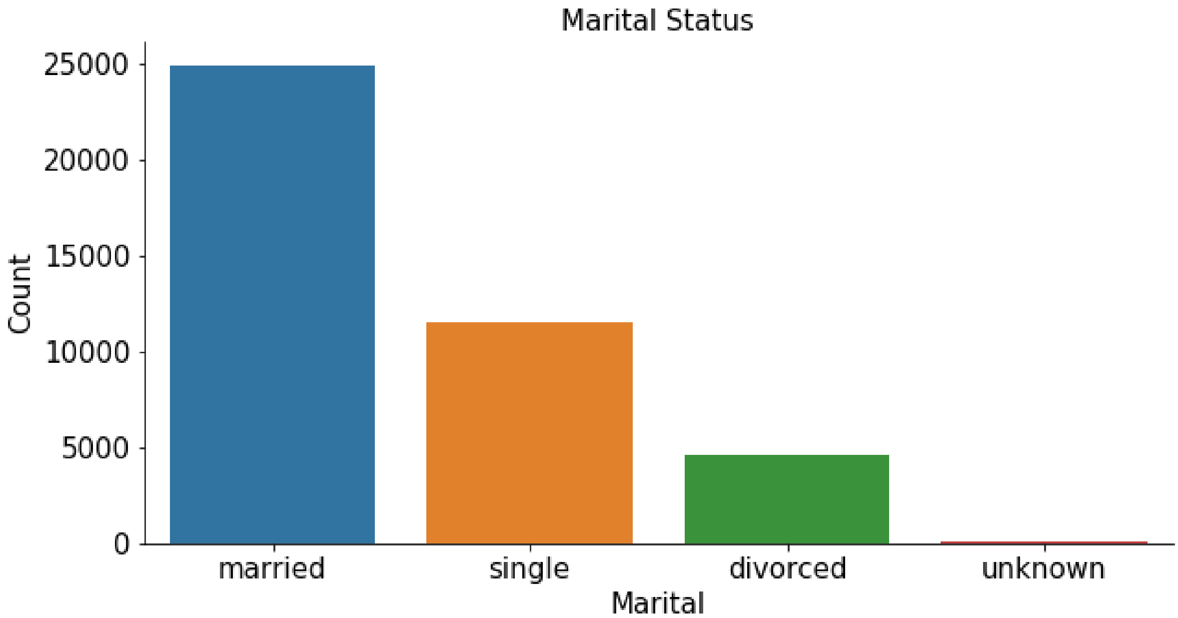
```
# What kind of jobs clients this bank have?
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'job', data = bank_client)
ax.set_xlabel('Job', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Client Jobs', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```



1.3 MARITAL

In [11]:

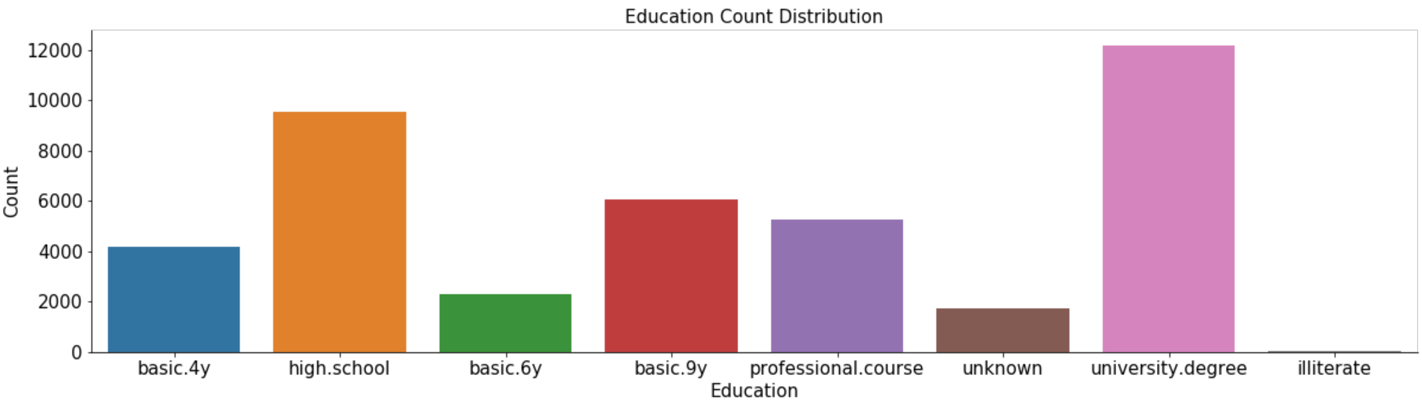
```
#Marital Status of clients
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'marital', data = bank_client)
ax.set_xlabel('Marital', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Marital Status', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```



1.4 EDUCATION

In [12]:

```
# Educational status of the clients
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()
```

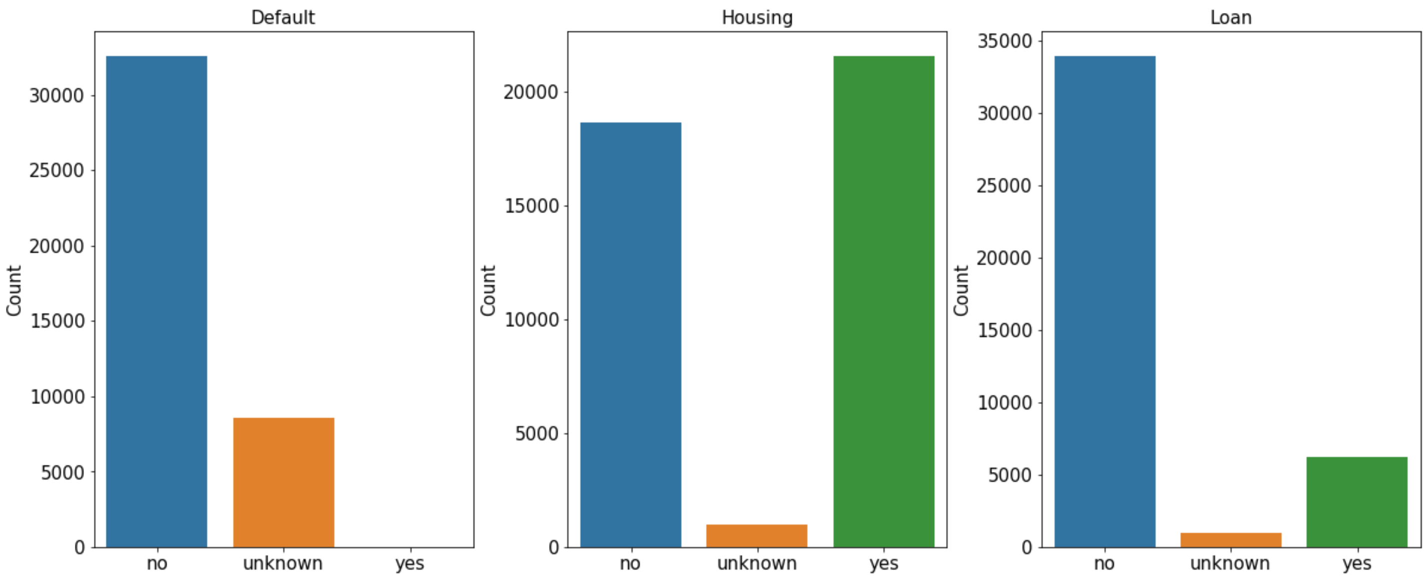


1.5 DEFAULT, HOUSING, LOAN

In [13]:

```
# Default, has credit in default ?
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (20,8))
```

```
sns.countplot(x = 'default', data = bank_client, ax = ax1, order = ['no', 'unknown', 'ye
s'])
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 = ['no', 'unknown', 'ye
s'])
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)
```



Finding exact number of default, housing and loan

In [14]:

```
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'] == 'un
known']['age'].count(),
                '\n Yes to credit in default:' , bank_client[bank_client['default'] == 'ye
s']     ['age'].count())
```

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

In [15]:

```
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'] == 'unkn
own']['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 [16]:

```
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'] == 'unknow
n']['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

We think the best representation for Jobs, Marital, Education, Default, load and housing is the count plot distribution.

1.6 Bank Client Categorical Treatment

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

In [17]:

```
# Label encoder order is alphabetical
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_client['job']      = labelencoder_X.fit_transform(bank_client['job'])
bank_client['marital']  = labelencoder_X.fit_transform(bank_client['marital'])
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 [18]:

```
#binning age and then label encoding the attribute
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 [19]:

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

```
(41188, 7)
```

Out[19]:

	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

	age	job	marital	education	default	housing	loan
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 [20]:

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

Out[20]:

	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 [21]:

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

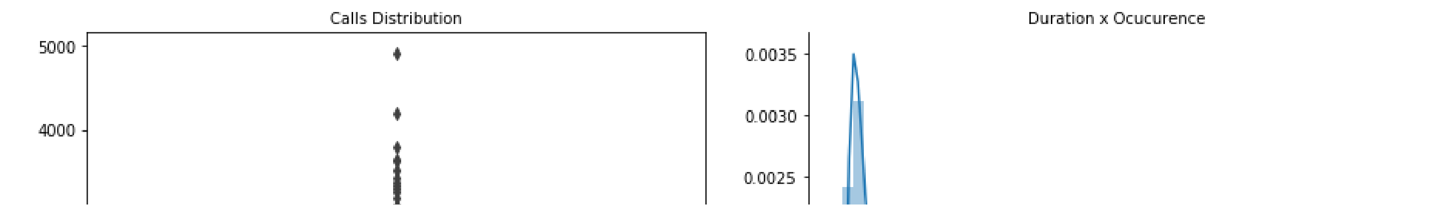
Out[21]:

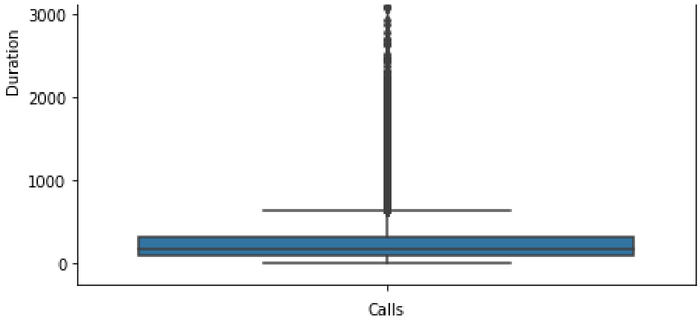
```
contact        False
month          False
day_of_week    False
duration       False
dtype: bool
```

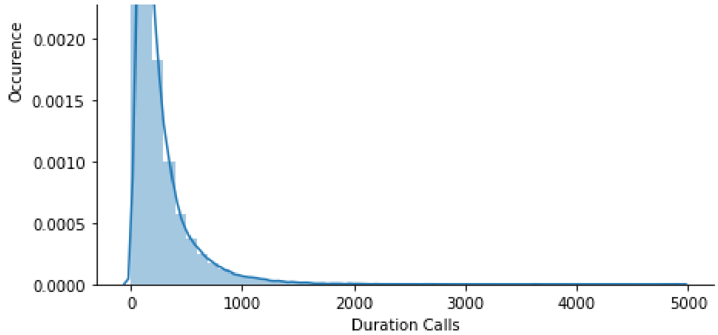
2.1 Duration

In [22]:

```
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5))
sns.boxplot(x = 'duration', data = bank_related, orient = 'v', ax = ax1)
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 [23]:

```
print("Max duration  call in minutes:  ", round((bank_related['duration'].max()/60),1))
print("Min duration  call in minutes:   ", round((bank_related['duration'].min()/60),1))
print("Mean duration call in minutes:   ", round((bank_related['duration'].mean()/60),1)
)
print("STD duration  call in minutes:   ", round((bank_related['duration'].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 [24]:

```
# Quartiles
print('1° Quartile: ', bank_related['duration'].quantile(q = 0.25))
print('2° Quartile: ', bank_related['duration'].quantile(q = 0.50))
print('3° Quartile: ', bank_related['duration'].quantile(q = 0.75))
print('4° Quartile: ', bank_related['duration'].quantile(q = 1.00))

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

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

In [25]:

```
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'] > 644.5]['duration'].
count()*100/len(bank_related),2), '%')
```

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

In [26]:

```
#If the call duration is equal to 0, then we observe that the person isn't subscribed.
bank[(bank['duration'] == 0)]
```

Out[26]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign
6251	39	admin.	married	high.school	no	yes	no	telephone	may	tue	...	4
23031	59	management	married	university.degree	no	yes	no	cellular	aug	tue	...	10

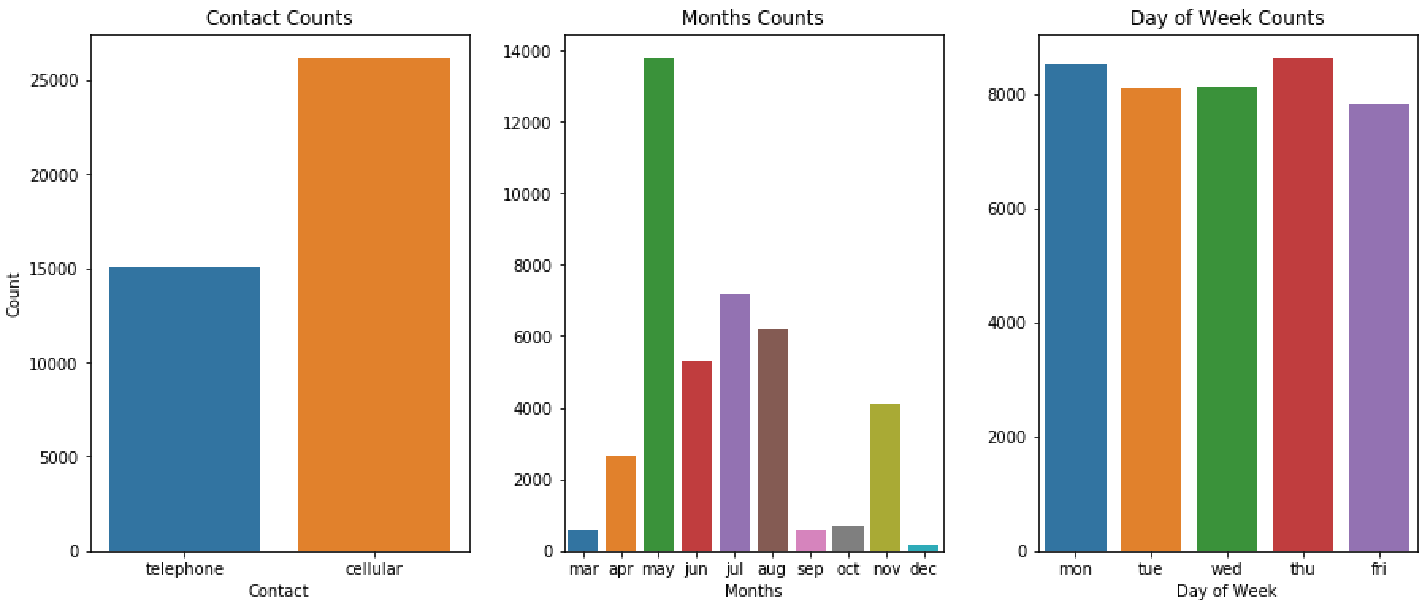
	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign
28063	33	blue-collar	divorced	high.school	no	yes	no	cellular	apr	fri	...	3
33015	31	blue-collar	married	basic.9y	no	no	no	cellular	may	mon	...	2

4 rows × 21 columns

2.2 Contact, Month, Day of Week

In [27]:

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (15,6))
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)
```



2.3 Contact, Month, Day of Week treatment

In [28]:

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

In [29]:

```
bank_related.head()
```

Out[29]:

	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 [30]:

```
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 [31]:

```
bank_related.head()
```

Out[31]:

	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 [32]:

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

Out[32]:

	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 [33]:

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

Out[33]:

	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 [34]:

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

Out[34]:

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

In [35]:

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

Model

In [36]:

```
bank_final= pd.concat([bank_client, bank_related, bank_se, bank_o], axis = 1)
bank_final = bank_final[['age', 'job', 'marital', 'education', 'default', 'housing', 'lo
an',
                     'contact', 'month', 'day_of_week', 'duration', 'emp.var.rate', 'con
s.price.idx',
                     'cons.conf.idx', 'euribor3m', 'nr.employed', 'campaign', 'pdays', '
previous', 'poutcome']]
bank_final.shape
```

Out[36]:

```
(41188, 20)
```

In [37]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(bank_final, y, test_size = 0.2, rand
om_state = 101) #check test_size

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 [38]:

```
X_train.head()
```

Out[38]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	emp.var.rate	cons.price.i
39577	4	5	1	2	0	0	0	0	6	3	2	-1.8	93.8
10104	3	9	1	6	0	2	0	1	4	2	4	1.4	94.4
17235	2	7	1	3	0	0	0	0	3	0	4	1.4	93.9
20926	1	0	2	6	0	2	0	0	1	2	4	1.4	93.4

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	emp.var.rate	cons.price.i
17626	3	2	1	2	0	2	0	0	3	1	2	1.4	93.9

In [39]:

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

In [40]:

```
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 = 'acc
uracy').mean())
```

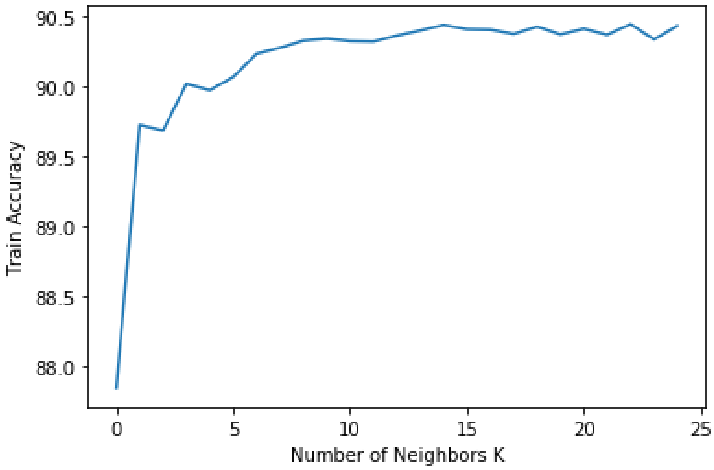
```
[[7106  173]
 [ 618  341]]
90.0
```

In [41]:

```
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='uniform', p=2, metric='eu
clidean')
    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, scores.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[optim
al_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 [42]:

```
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, scoring = 'accuracy
').mean())
```

```
[[7163  116]
 [ 706  253]]
90.0
```

In [43]:

```
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, scoring = 'accuracy
').mean())
```

```
[[6719  560]
 [ 605  354]]
86.0
```

In [44]:

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='gini') #criterion = entopy, gini
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 = 'accu
racy').mean())
```

```
[[6811  468]
 [ 491  468]]
88.0
```

In [45]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 200)#criterion = entopy,gini
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, scoring = 'accuracy
').mean())
```

```
[[6987  292]
 [ 515  444]]
90.0
```

In [46]:

```
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())
```

```
[[6451  828]
 [ 433  526]]
85.0
```

In [47]:

```
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())
```

```
[[6999  280]
 [ 482  477]]
91.0
```

In [48]:

```
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, scoring = 'accuracy
').mean())
```

```
[[7023  256]
 [ 487  472]]
91.0
```

In [49]:

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

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

Out[49]:

	Models	Score
7	Gradient Boosting	0.914203
6	XGBoost	0.912200
0	Random Forest Classifier	0.910046
4	Logistic Model	0.909772
3	K-Near Neighbors	0.904613
1	Decision Tree Classifier	0.884401
2	Support Vector Machine	0.856055
5	Gausian NB	0.845038