Name: Prateek Soni

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

In [2]:
```
bank=pd.read_csv('bank-additional-full.csv', sep=';')
```

In [3]:
```
bank
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	0
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	0
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	1

41188 rows × 21 columns

In [4]:
```
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[4]:

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

5 rows × 21 columns

In [5]:
```
# 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):
 #   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 [6]:
```
bank.columns
```

Out[6]:
```
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. Bank client data Analysis and Categorical Treatment

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

Out[7]:

	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 [8]:
```
# 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 [9]:
```
print('Marital:\n', bank_client['marital'].unique())
```

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

In [10]:
```
print('Education:\n', bank_client['education'].unique())
```

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

In [11]:
```
print('Default:\n', bank_client['default'].unique())
print('Housing:\n', bank_client['housing'].unique())
print('Loan:\n', bank_client['loan'].unique())
```

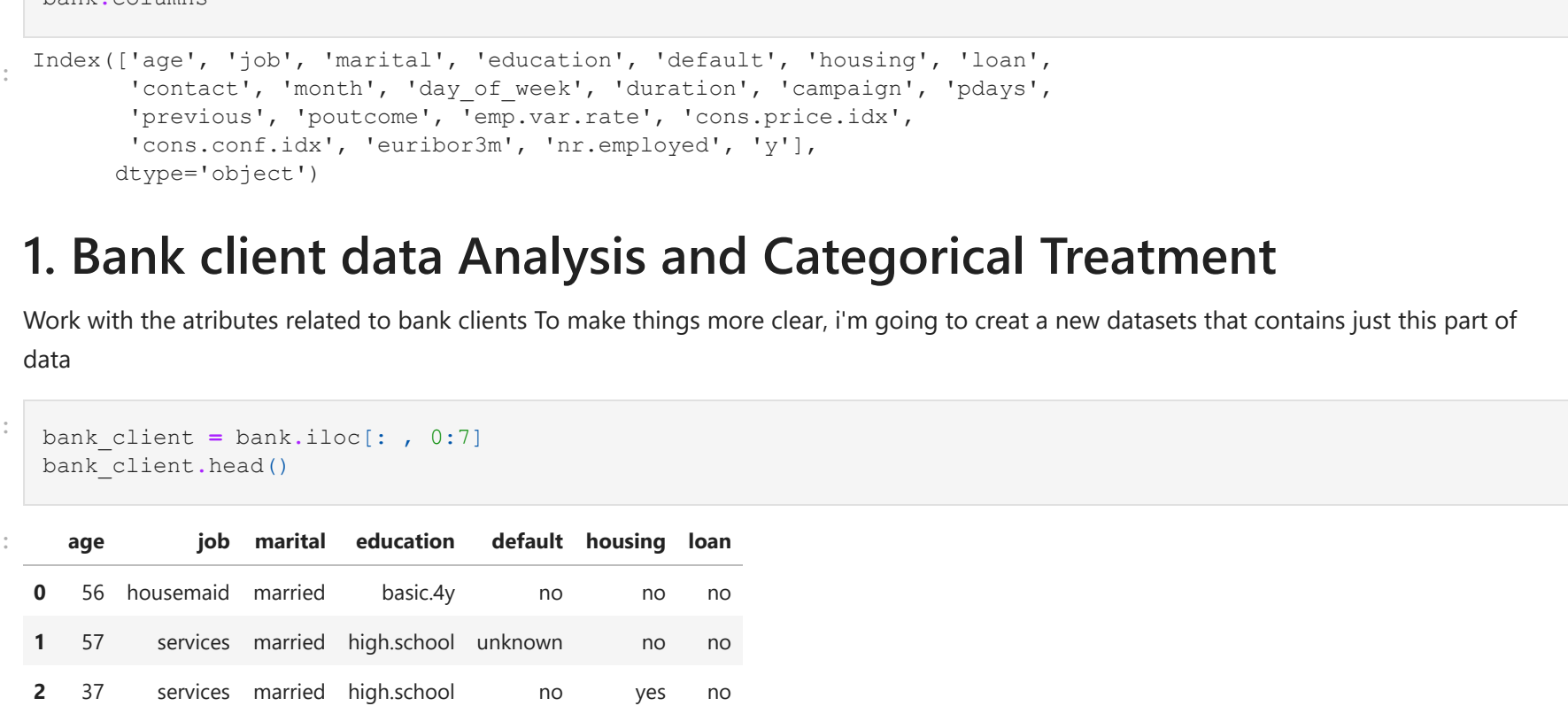
```
Default:
 ['no' 'unknown' 'yes']
Housing:
 ['no' 'yes' 'unknown']
Loan:
 ['no' 'yes' 'unknown']
```

1.2 Age

In [12]:
```
#Trying to find some strange values or null values
print('Max age: ', bank_client['age'].max())
print('Min age: ', bank_client['age'].min())
print('Null Values: ', bank_client['age'].isnull().any())
```

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

In [13]:
```
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 [14]:
```
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 [15]:
```
# 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) - bank_client['age'].quantile(q = 0.25)), 'are
```

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

In [16]:
```
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]['age'].count()*100/len(bank_client),2), '%'
```

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

In [17]:
```
# 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))
# I thing the best way to give a precisly insight abou dispersion is using the CV (coefficient variation) (STD/
#    cv < 15%, low dispersion
#    cv > 30%, high dispersion
print('CV  :',round(bank_client['age'].std()*100/bank_client['age'].mean(), 1), ', High middle dispersion')
```

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

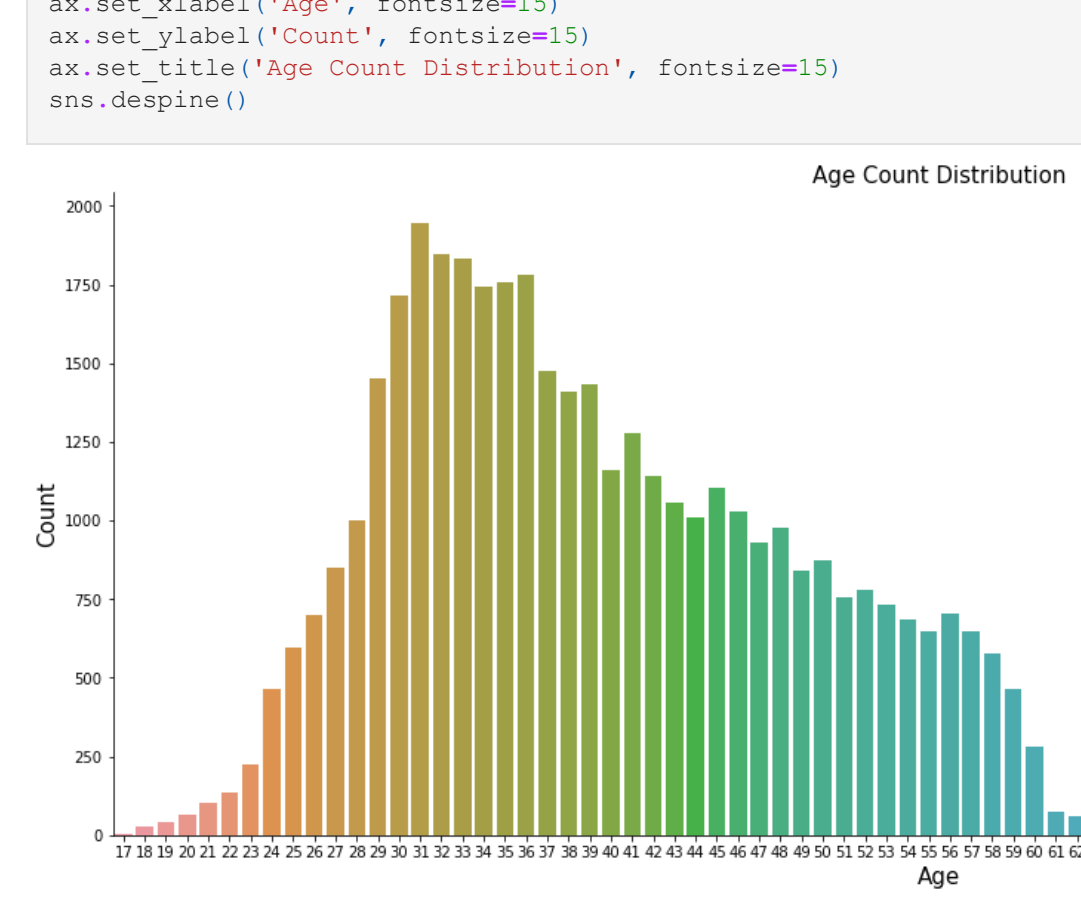
Conclusion about AGE, in my opinion due to almost high dispersion and just looking at this this graph we cannot conclude if age have a high effect to our variable y, need to keep searching for some pattern. high middle dispersion means we have people with all ages and maybe all of them can subscript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

1.3 Jobs

In [18]:
```
# What kind of jobs clients this bank have, if you cross jobs with default, loan or housing, there is no relati
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()
```

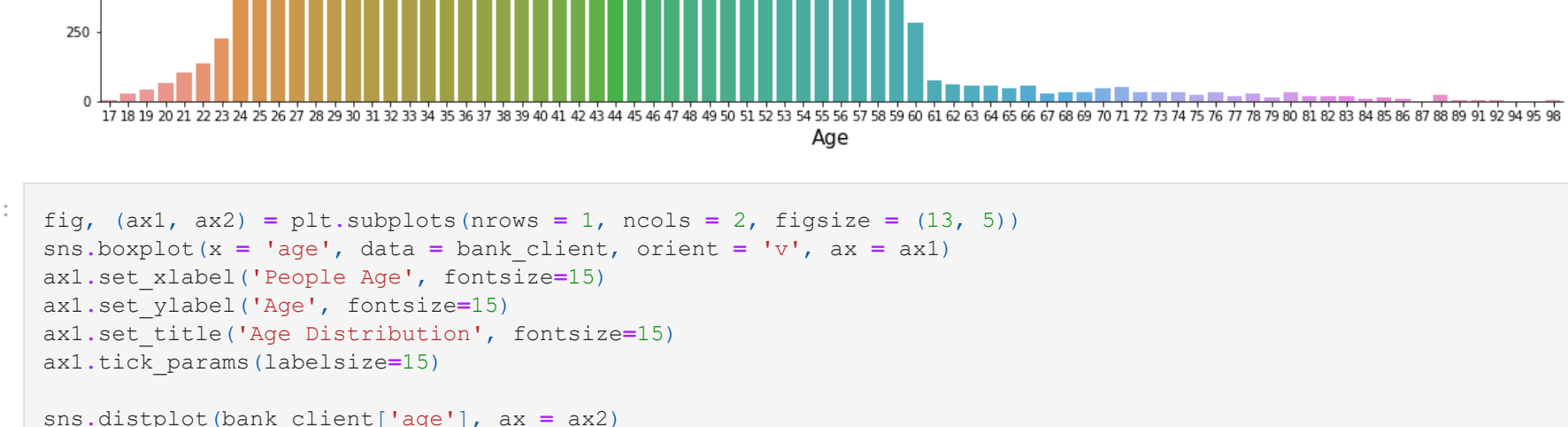
1.4 Marital

In [19]:
```
# What kind of 'marital clients' this bank have, if you cross marital with default, loan or housing, there is 
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()
```



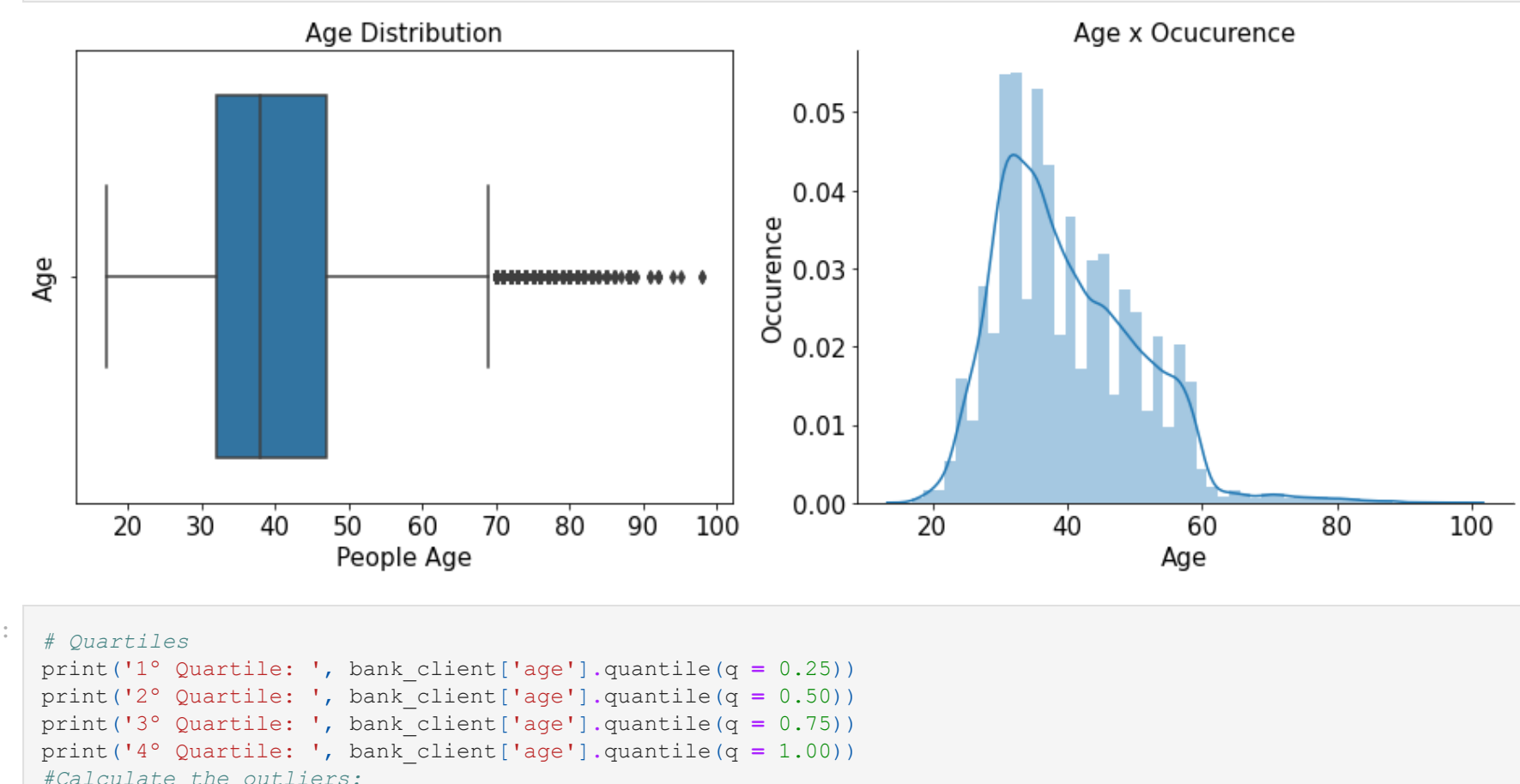
1.5 Education

In [20]:
```
# What kind of 'education clients this bank have, if you cross education with default, loan or housing, there i
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 [21]:
```
# 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', '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 = ['no', '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 [22]:
```
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 [23]:
```
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 [24]:
```
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 [25]:
```
# 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 [26]:
```
#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 [27]:
```
bank_client.head()
```

Out[27]:

	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

In [28]:
```
print(bank_client.shape)
bank_client.head()
```

```
(41188, 7)
```

Out[28]:

	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 [29]:
```
# Slicing DataFrame to treat separately, make things more easy
bank_related = bank.iloc[: , 7:11]
bank_related.head()
```

Out[29]:

	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 [30]:
```
bank_related.isnull().any()
```

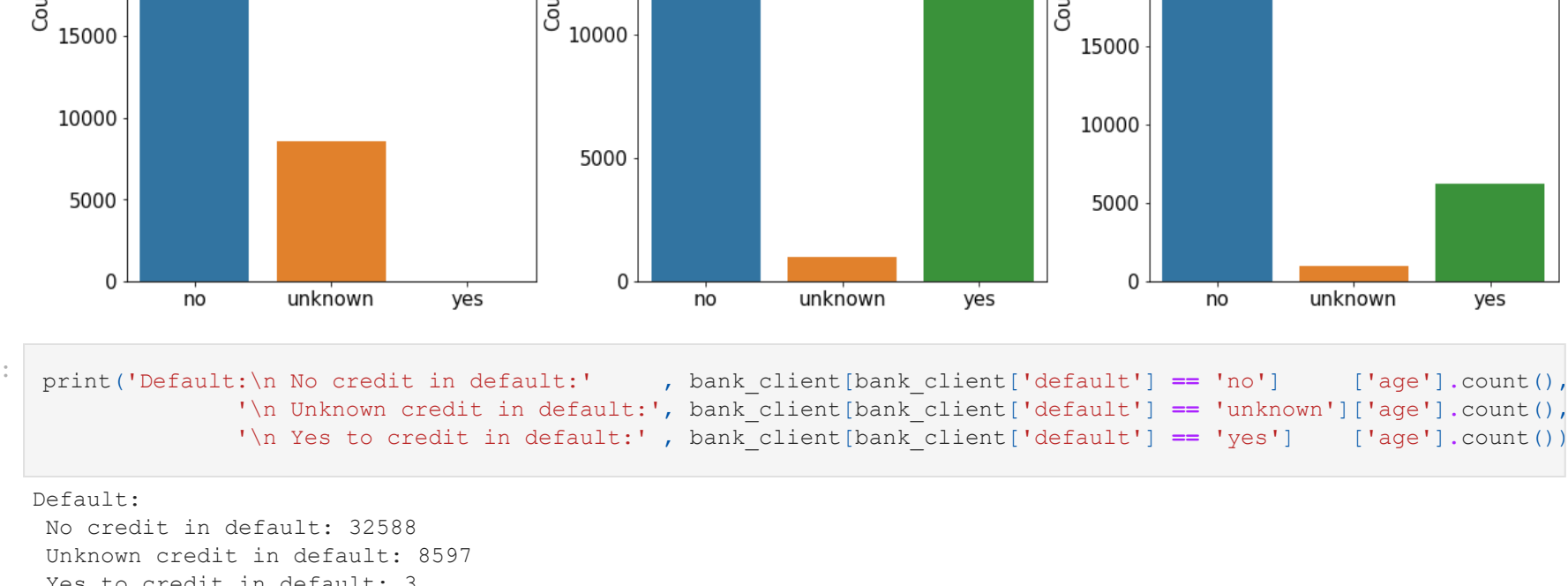
Out[30]:
```
contact        False
month          False
day_of_week    False
duration       False
dtype: bool
```

In [31]:
```
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 [32]:
```
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()
```



In [33]:
```
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 [34]:
```
# 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 =
```

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

In [35]:
```
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_re
```

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

In [36]:
```
# 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)]
```

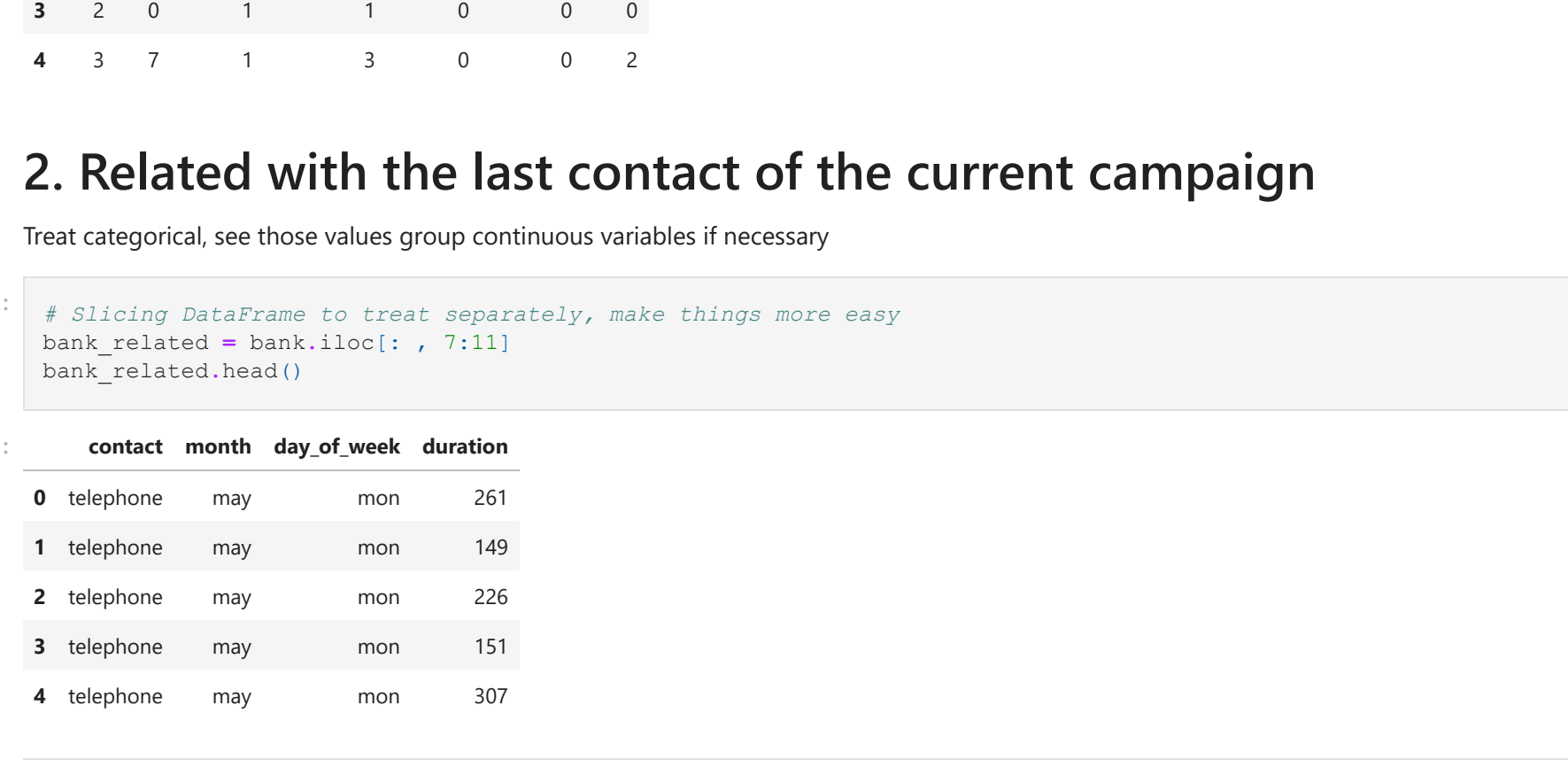
Out[36]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous
6251	39	admin.	married	high.school	no	yes	no	telephone	may	tue	...	4	999	0
23031	59	management	married	university.degree	no	yes	no	cellular	aug	tue	...	10	999	0
28063	53	blue-collar	divorced	high.school	no	yes	no	cellular	apr	fri	...	3	999	0
33015	31	blue-collar	married	basic.9y	no	no	no	cellular	may	mon	...	2	999	0

4 rows × 21 columns

2.2 Contact, Month, Day of the week

In [37]:
```
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',
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 [38]:
```
print('Ages above: ', bank_related['duration'].quantile(q = 0.75) +
                      1.5*(bank_related['duration'].quantile(q = 0.75) - bank_related['duration'].quantile(q =
```

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

In [39]:
```
bank_related[bank_related['duration'] > 640].count()
```

Out[39]:
```
contact        3008
month          3008
day_of_week    3008
duration       3008
dtype: int64
```

2.1 Contact, Month, Day of Week treatment

In [40]:
```
# 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 [41]:
```
bank_related.head()
```

Out[41]:

	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 [42]:
```
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 [43]:
```
bank_related.head()
```

Out[43]:

	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 [44]:
```
bank_se = bank.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']]
bank_se.head()
```

Out[44]:

	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 [45]:
```
bank_o = bank.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
bank_o.head()
```

Out[45]:

	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 [46]:
```
bank_o['poutcome'].unique()
```

Out[46]:
```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

In [47]:
```
bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2,3], inplace  = True)
```

Model

In [48]:
```
bank_final= pd.concat([bank_client, bank_related, bank_se, bank_o], axis = 1)
bank_final = bank_final[['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
                     'contact', 'month', 'day_of_week', 'duration', 'emp.var.rate', 'cons.price.idx',
                     'cons.conf.idx', 'euribor3m', 'nr.employed', 'campaign', 'pdays', 'previous', 'poutcome']]
bank_final.shape
```

Out[48]:
```
(41188, 20)
```

In [49]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(bank_final, y, test_size = 0.1942313295, 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 [50]:
```
X_train.head()
```

Out[50]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	emp.var.rate	cons.price.idx	cons.conf.id
38912	3	5	1	6	0	2	0	0	7	4	5	-3.4	92.649	-30
9455	2	7	1	5	1	0	0	1	4	0	2	1.4	94.465	-41
14153	1	4	1	6	0	2	0	0	3	1	5	1.4	93.918	-42
25021	3	6	1	6	0	2	0	0	7	3	1	-0.1	93.200	-42
30911	2	5	0	0	0	2	2	0	6	3	3	-1.8	92.893	-46

In [51]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [52]:

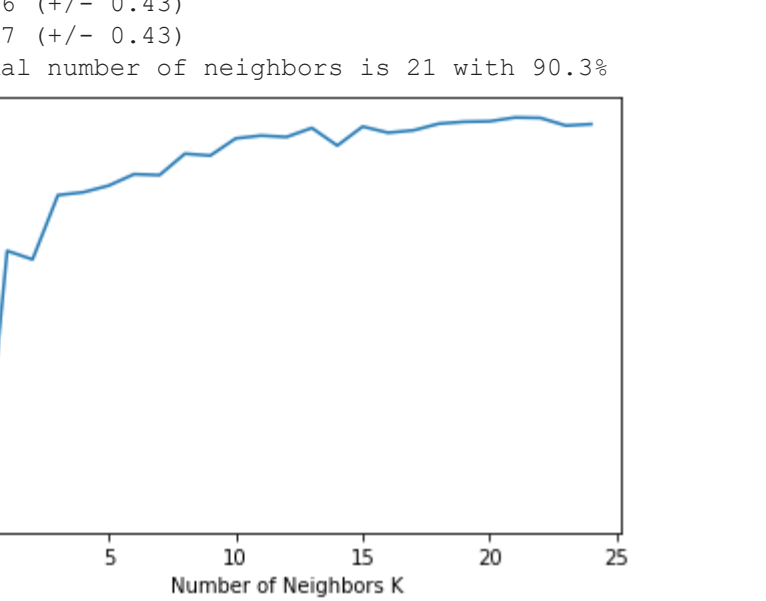
```python
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
logpred = logmodel.predict(X_test)


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 [55]:

```python
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)

#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='euclidean')
    kfold = model_selection.KFold(n_splits=10)
    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[optimal_k]))

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

```
k=1 87.80 (+/- 0.56)
k=2 89.47 (+/- 0.59)
k=3 89.41 (+/- 0.48)
k=4 89.82 (+/- 0.46)
k=5 89.84 (+/- 0.50)
k=6 89.88 (+/- 0.51)
k=7 89.95 (+/- 0.46)
k=8 89.95 (+/- 0.47)
k=9 90.08 (+/- 0.44)
k=10 90.07 (+/- 0.36)
k=11 90.18 (+/- 0.36)
k=12 90.20 (+/- 0.43)
k=13 90.19 (+/- 0.42)
k=14 90.25 (+/- 0.51)
k=15 90.14 (+/- 0.46)
k=16 90.26 (+/- 0.49)
k=17 90.22 (+/- 0.45)
k=18 90.23 (+/- 0.45)
k=19 90.28 (+/- 0.42)
k=20 90.29 (+/- 0.45)
k=21 90.29 (+/- 0.49)
k=22 90.32 (+/- 0.47)
k=23 90.31 (+/- 0.46)
k=24 90.26 (+/- 0.43)
k=25 90.27 (+/- 0.43)
The optimal number of neighbors is 21 with 90.3%
```



In [56]:

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=21)
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())
```

```
[[6951  122]
 [ 671  256]]
90.0
```

In [57]:

```python
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())
```

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

In [58]:

```python
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 = 'accuracy').mean())
```

```
[[6608  465]
 [ 483  444]]
88.0
```

In [59]:

```python
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())
```

```
[[6793  280]
 [ 491  436]]
90.0
```

In [60]:

```python
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 [62]:

```python
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())
```

```
[16:07:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[[6796  277]
 [ 462  465]]
91.0
[16:07:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[16:07:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [64]:

```python
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())
```

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

In [65]:

```python
models = pd.DataFrame({
                'Models': ['Random Forest Classifier', 'Decision Tree Classifier', 'Support 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[65]:

	Models	Score
7	Gradient Boosting	0.914306
6	XGBoost	0.910932
4	Logistic Model	0.909666
0	Random Forest Classifier	0.909636
3	K-Near Neighbors	0.904634
1	Decision Tree Classifier	0.885320
2	Support Vector Machine	0.855550
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)

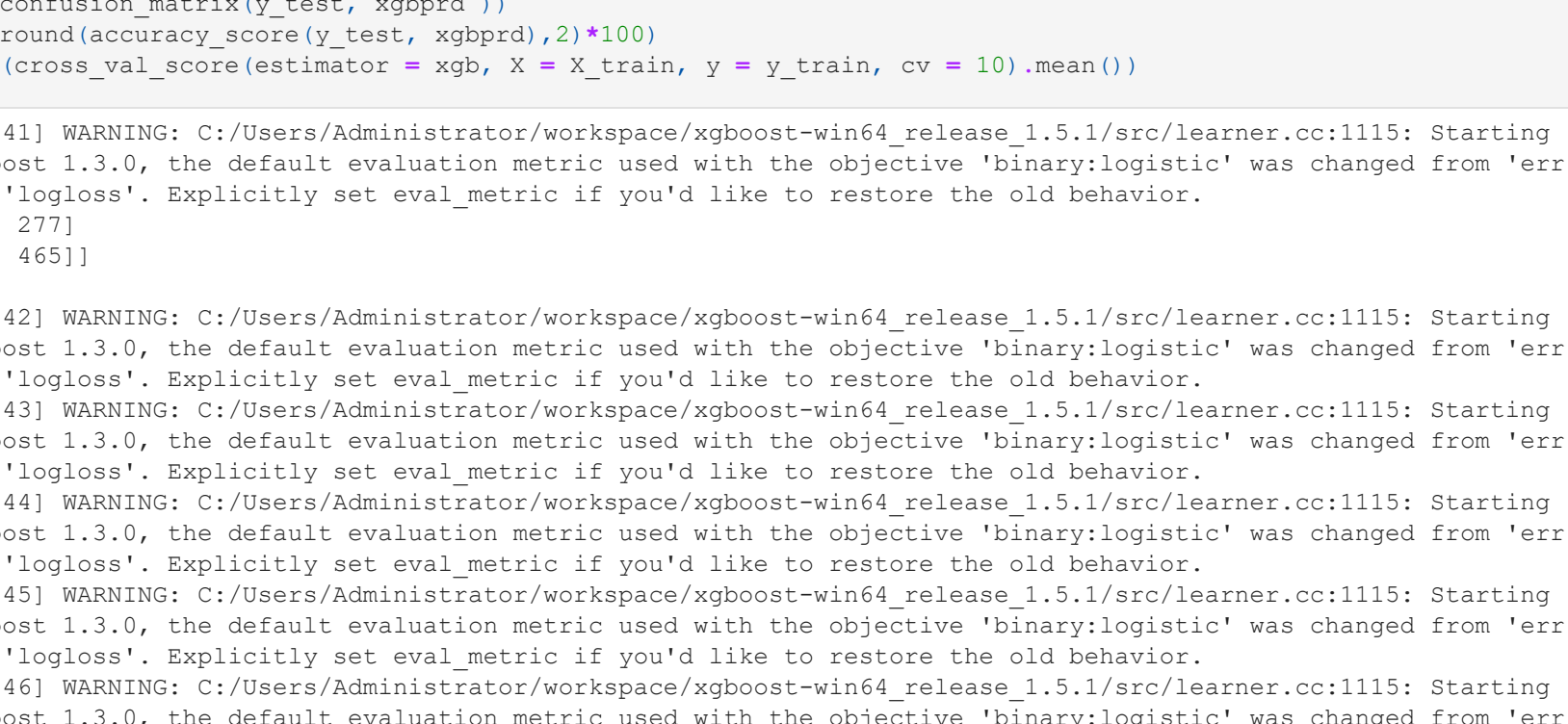
.70-.80 = fair (C)

.60-.70 = poor (D)

.50-.60 = fail (F)

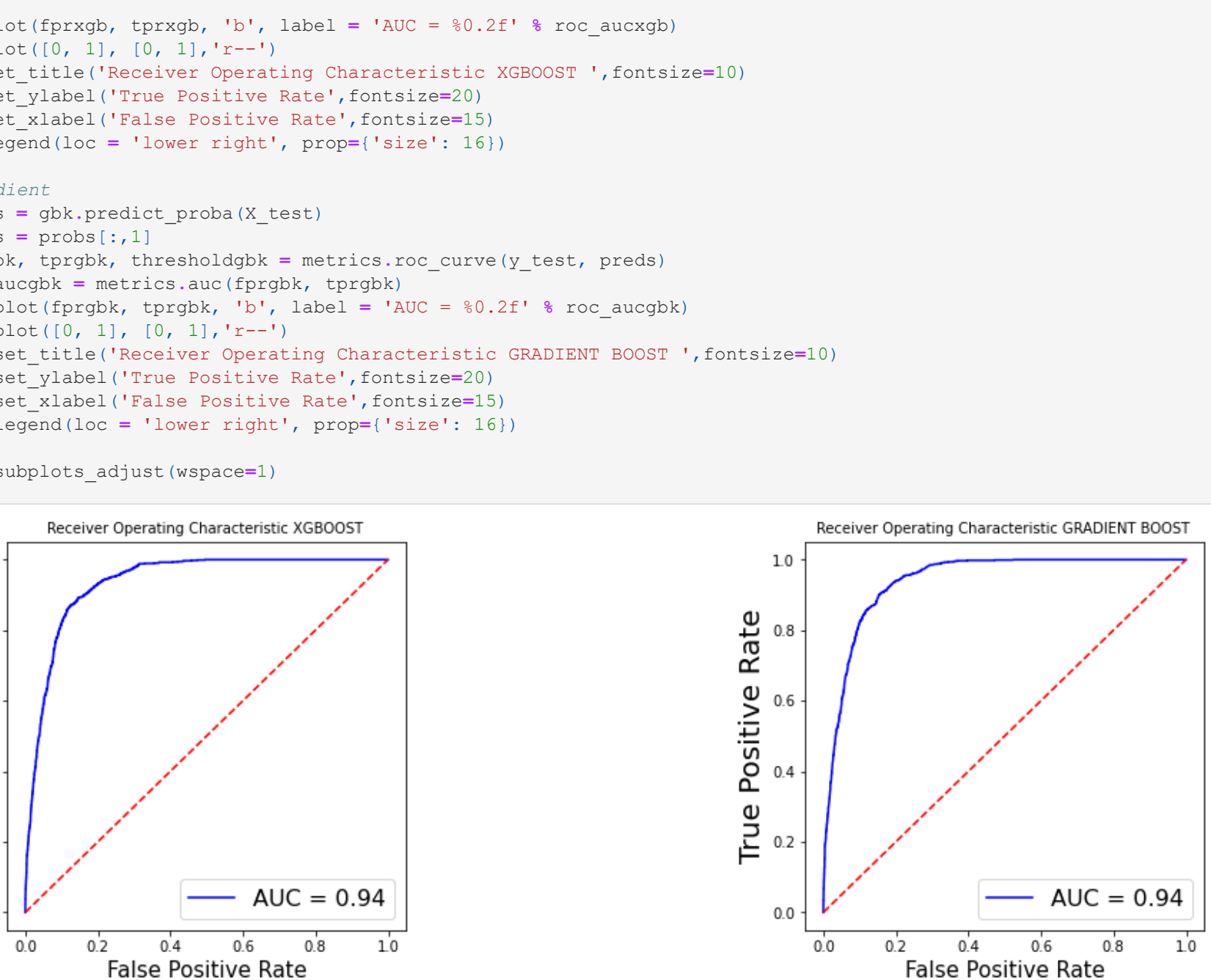
In [68]:

```python
# 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 ',fontsize=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 [69]:

```python
#fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows = 2, ncols = 3, figsize = (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_auclog)
ax_arr[0,0].plot([0, 1], [0, 1],'r--')
ax_arr[0,0].set_title('Receiver Operating Characteristic Logistic ',fontsize=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_aucrfc)
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_aucknn)
ax_arr[0,2].plot([0, 1], [0, 1],'r--')
ax_arr[0,2].set_title('Receiver Operating Characteristic KNN ',fontsize=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_aucgau)
ax_arr[1,1].plot([0, 1], [0, 1],'r--')
ax_arr[1,1].set_title('Receiver Operating Characteristic Gaussian ',fontsize=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='black')
ax_arr[1,2].plot(fprdtree, tprdtree, 'b', label = 'Decision Tree', color='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 Result

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 [70]:

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

In [71]:

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

```
KNN Confusion Matrix
 [[6951  122]
 [ 671  256]]
```

In [72]:

```python
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       0.68      0.28      0.39       927

    accuracy                           0.90      8000
   macro avg       0.79      0.63      0.67      8000
weighted avg       0.88      0.90      0.88      8000
```

Recall

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 [73]:

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

```
0.98
```

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 [74]:

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

```
0.26
0.28
```

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 [75]:

```python
print(round(7163 / (7163 + 706),2))
```

```
0.91
```

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 [76]:

```python
print(round(253 / (253 + 116),2))
print(round(metrics.precision_score(y_test, knnpred),2))
```

```
0.69
0.68
```

F1 Score

F1-SCORE F1-Score is a "median" of Recall and Precision, consider this when you want a balance between this metrics F1 = 2(Precision(0) Recall(0)) / (Precision(0) + Recall(0))

In [77]:

```python
F1_0 = 2*0.91*0.98/(0.91+0.98)
round(F1_0,2)
```

Out[77]:

```
0.94
```

In [78]:

```python
F1_1 = 2*0.69*0.26/(0.69+0.26)
round(F1_1,2)
```

Out[78]:

```
0.38
```

Average Precision

In [79]:

```python
AVG_precision =  (0.91*(7279/8238))+ (0.69*(959/8238))
round(AVG_precision,2)
```

Out[79]:

```
0.88
```

In [80]:

```python
AVG_Recall =  (0.98*(7279/8238))+ (0.26*(959/8238))
round(AVG_Recall,2)
```

Out[80]:

```
0.9
```

In [81]:

```python
AVG_f1 =  (0.95*(7279/8238))+ (0.38*(959/8238))
round(AVG_f1,2)
```

Out[81]:

```
0.88
```

In [ ]: