Description

You are asked to make predictions for each unique id in the test dataset about the likelihood of the person having a bank account or not, i.e. Yes = 1, No = 0. You will train your model on 70% of the data and test your model on the final 30% of the data.

In [0]:

```
# Importing necessary Libraries to use
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
```

In [0]:

```
train =  pd.read_csv("Train_v2.csv")
test = pd.read_csv ('Test_v2.csv')
```

In [0]:

```
#DATA CLEANING ON TRAIN DATASET
```

In [264]:

```
train.head()
```

Out[264]:

	country	year	uniqueid	bank_account	location_type	cellphone_access	household_size	age_of_respondent	gender_of_respondent	relationship_with_head	marital_status	education_level	job_type
0	Kenya	2018	uniqueid_1	Yes	Rural	Yes	3	24	Female	Spouse	Married/Living together	Secondary education	Self employed
1	Kenya	2018	uniqueid_2	No	Rural	No	5	70	Female	Head of Household	Widowed	No formal education	Government Dependent
2	Kenya	2018	uniqueid_3	Yes	Urban	Yes	5	26	Male	Other relative	Single/Never Married	Vocational/Specialised training	Self employed
3	Kenya	2018	uniqueid_4	No	Rural	Yes	5	34	Female	Head of Household	Married/Living together	Primary education	Formally employe

	country	year	uniqueid	bank_account	location_type	cellphone_access	household_size	age_of_respondent	gender_of_respondent	relationship_with_head	marital_status	education_level	job_type
													d Private
4	Kenya	2018	uniqueid_5	No	Urban	No	8	26	Male	Child	Single/Never Married	Primary education	Informally employed

In [265]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23524 entries, 0 to 23523
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   country                 23524 non-null  object
 1   year                    23524 non-null  int64 
 2   uniqueid                23524 non-null  object
 3   bank_account            23524 non-null  object
 4   location_type           23524 non-null  object
 5   cellphone_access        23524 non-null  object
 6   household_size          23524 non-null  int64 
 7   age_of_respondent       23524 non-null  int64 
 8   gender_of_respondent    23524 non-null  object
 9   relationship_with_head  23524 non-null  object
 10  marital_status          23524 non-null  object
 11  education_level         23524 non-null  object
 12  job_type                23524 non-null  object
dtypes: int64(3), object(10)
memory usage: 2.3+ MB
```

In [266]:

```
# Checking the null values from the train dataset
train.isna().sum()
```

Out[266]:

```
country                   0
year                      0
uniqueid                  0
bank_account              0
location_type             0
cellphone_access          0
household_size            0
```

```
age_of_respondent         0
gender_of_respondent      0
relationship_with_head    0
marital_status            0
education_level           0
job_type                  0
dtype: int64
```

In [267]:

```python
# now it time to remove outliers in our train dataset - we are using z-scor
e to detect and remove the outliers
from scipy import stats
z = np.abs(stats.zscore(train._get_numeric_data()))
z
```

Out[267]:

```
array([[1.20854126, 0.35800673, 0.89618796],
       [1.20854126, 0.53983446, 1.88827897],
       [1.20854126, 0.53983446, 0.77512418],
       ...,
       [1.20854126, 0.53983446, 0.71459229],
       [1.20854126, 1.43767565, 0.53299662],
       [1.20854126, 2.78443744, 1.13831551]])
```

In [0]:

```python
train=train[(z<3).all(axis=1)]
```

In [269]:

```python
train.shape
```

Out[269]:

```
(23232, 13)
```

In [270]:

```python
train.describe() # 3 numerical columns only. Most of the columns are catego
rical.
```

Out[270]:

	year	household_size	age_of_respondent
count	23232.000000	23232.000000	23232.000000
mean	2016.969697	3.733815	38.610064
std	0.846098	2.095128	16.173751
min	2016.000000	1.000000	16.000000
25%	2016.000000	2.000000	26.000000

	year	household_size	age_of_respondent
50%	2017.000000	3.000000	35.000000
75%	2018.000000	5.000000	49.000000
max	2018.000000	10.000000	88.000000

In [271]:

```
train.dtypes
```

Out[271]:

```
country                   object
year                       int64
uniqueid                  object
bank_account              object
location_type             object
cellphone_access          object
household_size             int64
age_of_respondent          int64
gender_of_respondent      object
relationship_with_head    object
marital_status            object
education_level           object
job_type                  object
dtype: object
```

In [272]:

```
# Lets see if our dataset is balanced or not by checking our target distrib
ution
train.bank_account.value_counts()
```

Out[272]:

```
No     19946
Yes     3286
Name: bank_account, dtype: int64
```

In [273]:

```
a = len(train[train.bank_account=='Yes'])
b = len(train[train.bank_account=='No'])
c = len(train)
print('We have an imbalanced dataset with a %i/%i ratio'%((b/c*100),(a/c*10
0)+1))
```

```
We have an imbalanced dataset with a 85/15 ratio
```

The id+country thing

In [274]:

```
train.uniqueid.value_counts().head(5) #
```

Out[274]:

```
uniqueid_1502    4
```

```
uniqueid_985     4
uniqueid_18      4
uniqueid_1746    4
uniqueid_12      4
Name: uniqueid, dtype: int64
```

In [0]:

```
# some uniqueid's have the same value 4 times. this can be explained by the number of countries in the dataset. Hence the need to
  # identify people by 'country'+'uniqueid' to avoid having duplicate uniqueid values.
```

In [276]:

```
test.uniqueid.value_counts().head(5)
```

Out[276]:

```
uniqueid_8612    3
uniqueid_8627    3
uniqueid_8650    3
uniqueid_8633    3
uniqueid_8605    3
Name: uniqueid, dtype: int64
```

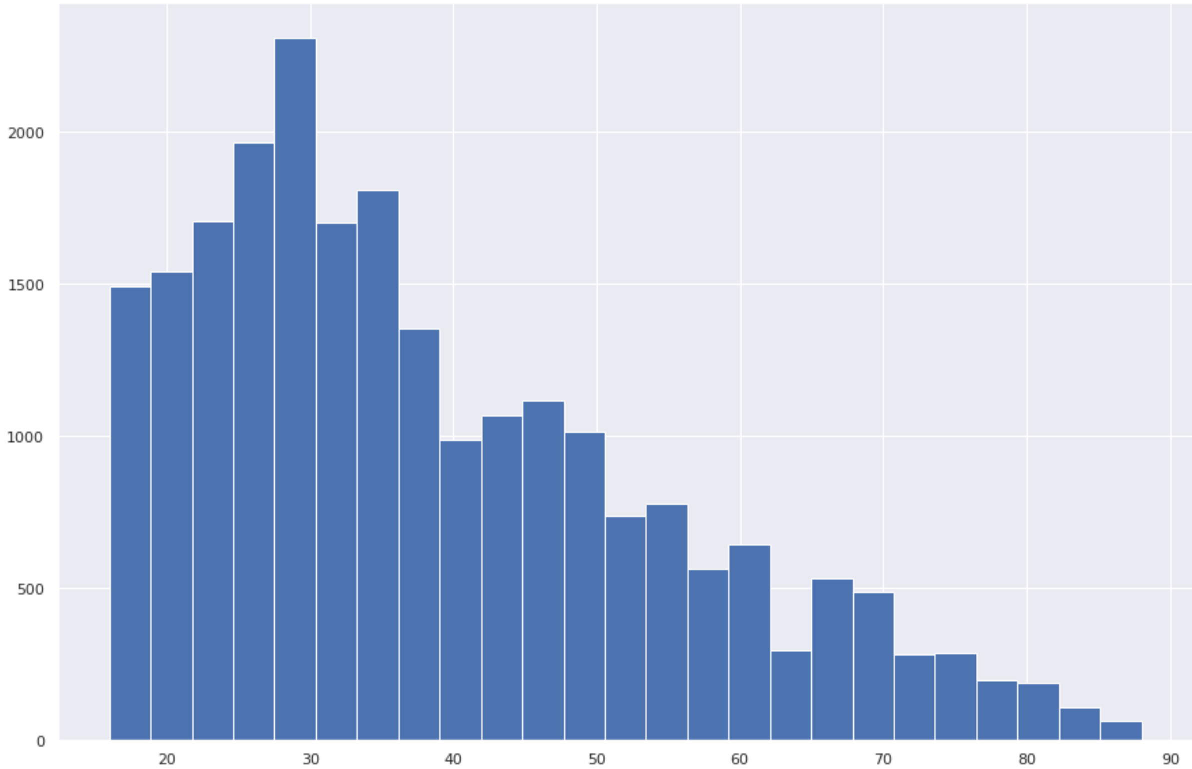
In [0]:

```
# Same thing needs to be done for the test set.
```

Exploratory Data Analysis

In [278]:

```
# age_of_respondant
hist_age = train.age_of_respondent.hist(bins=25,figsize=[15,10])
```



In [0]:

```
# We have a skewed to the right distribution for this variable.
```

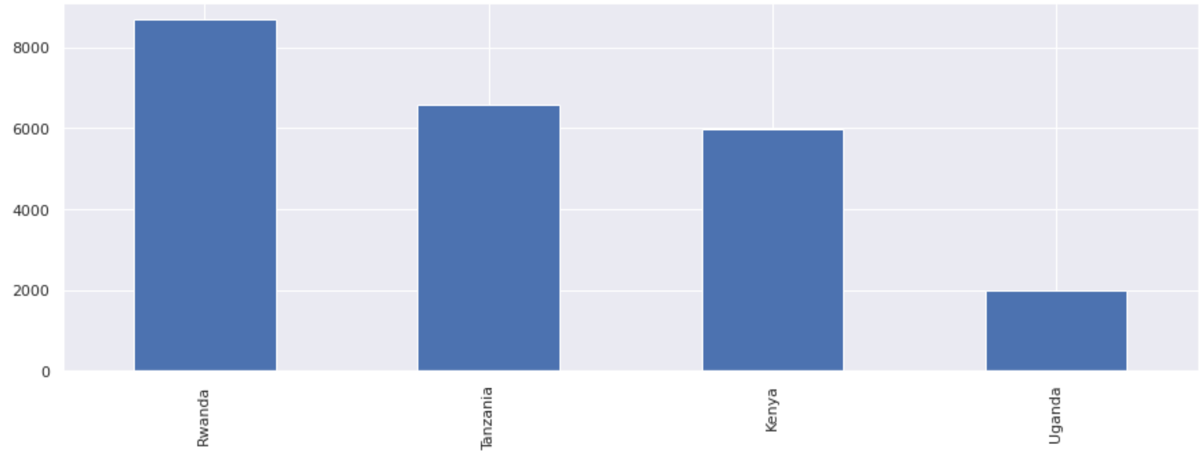
Country

In [280]:

```
train['country'].value_counts().plot(kind='bar',figsize=[15,5])
```

Out[280]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7e20668>
```



In [0]:

```
# Rwanda is the most occuring value for country, while Uganda is the least
occuring one.
```

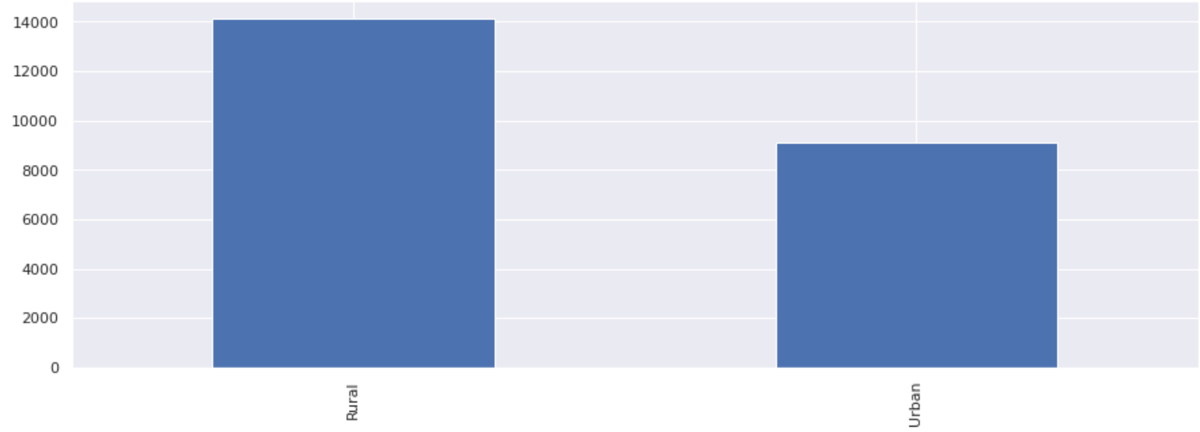
location type

In [282]:

```
train['location_type'].value_counts().plot(kind='bar',figsize=[15,5])
#plot()
```

Out[282]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7dfd7b8>
```



In [0]:

```
# More people from rural places have been interviewed than people in urban
places.
```

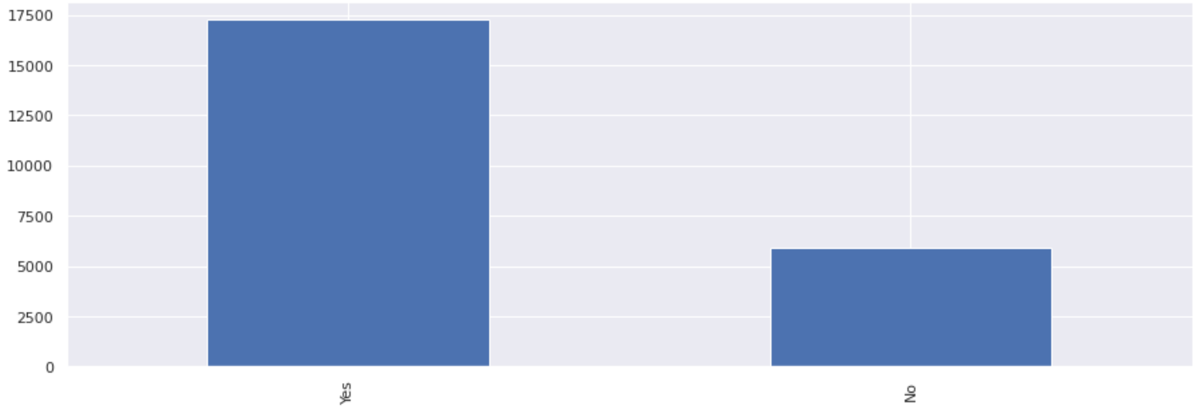
cellphone access

In [284]:

```
train['cellphone_access'].value_counts().plot(kind='bar',figsize=[15,5])
```

Out[284]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7d4c8d0>
```



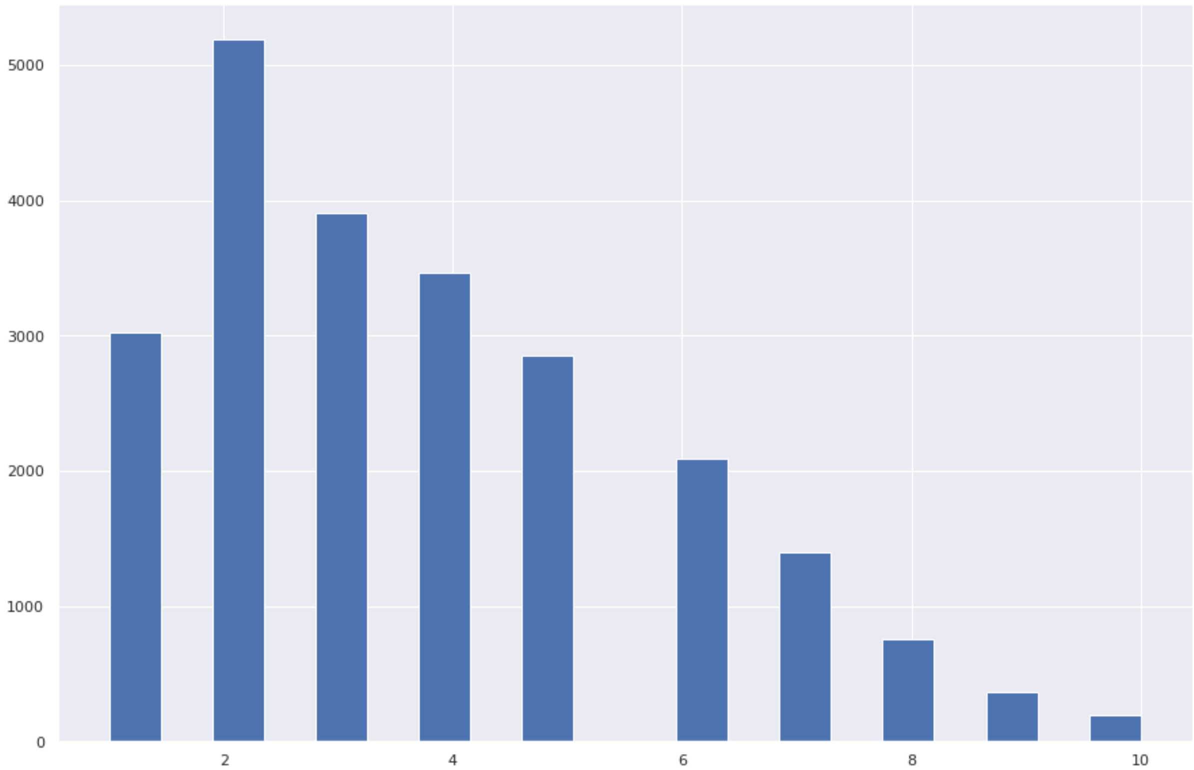
In [0]:

```
# Yes' indicates the non-possession of a bank account which is unlikely.
```

household_size

In [286]:

```
hist_hs = train.household_size.hist(bins=20,figsize=[15,10])
```



In [0]:

```
# Another numerical distribution that's skewed to the right
```

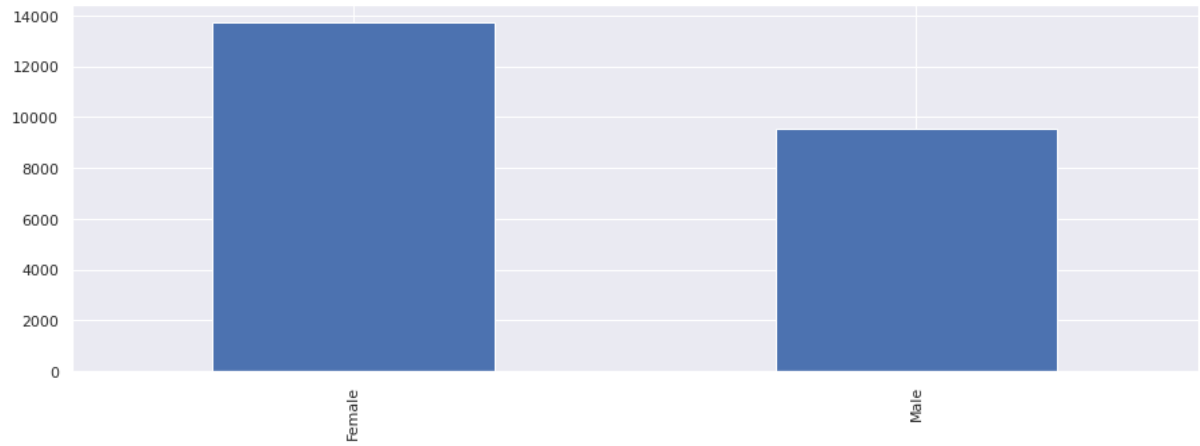
gender of respondant

In [288]:

```
train['gender_of_respondent'].value_counts().plot(kind='bar',figsize=[15,5]
)
```

Out[288]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d848ba90>
```



In [0]:

```
# Trainset has more females than males.
```

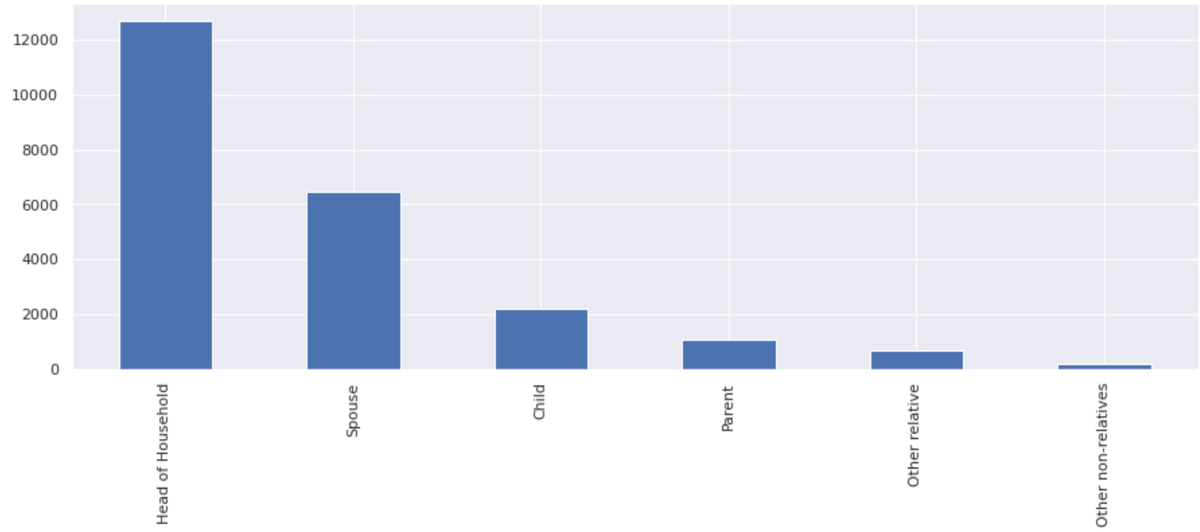
relationship with head

In [290]:

```
train['relationship_with_head'].value_counts().plot(kind='bar',figsize=[15,
5])
```

Out[290]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d848b4e0>
```



In [0]:

```
# 6 category and the most occuring is 'Head of Household' followed by Spous
e.
```

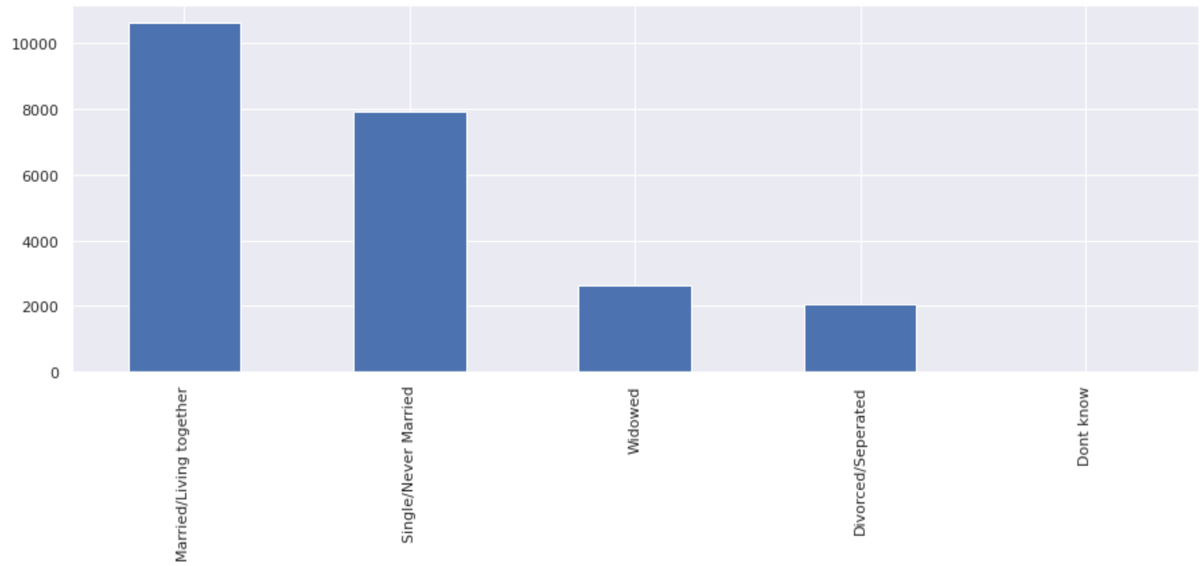
marital status

In [292]:

```
train['marital_status'].value_counts().plot(kind='bar',figsize=[15,5])
```

Out[292]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95dad8aa20>
```



In [0]:

```
# 5 categories with one category 'Don't know' being significantly undersamp
led
```

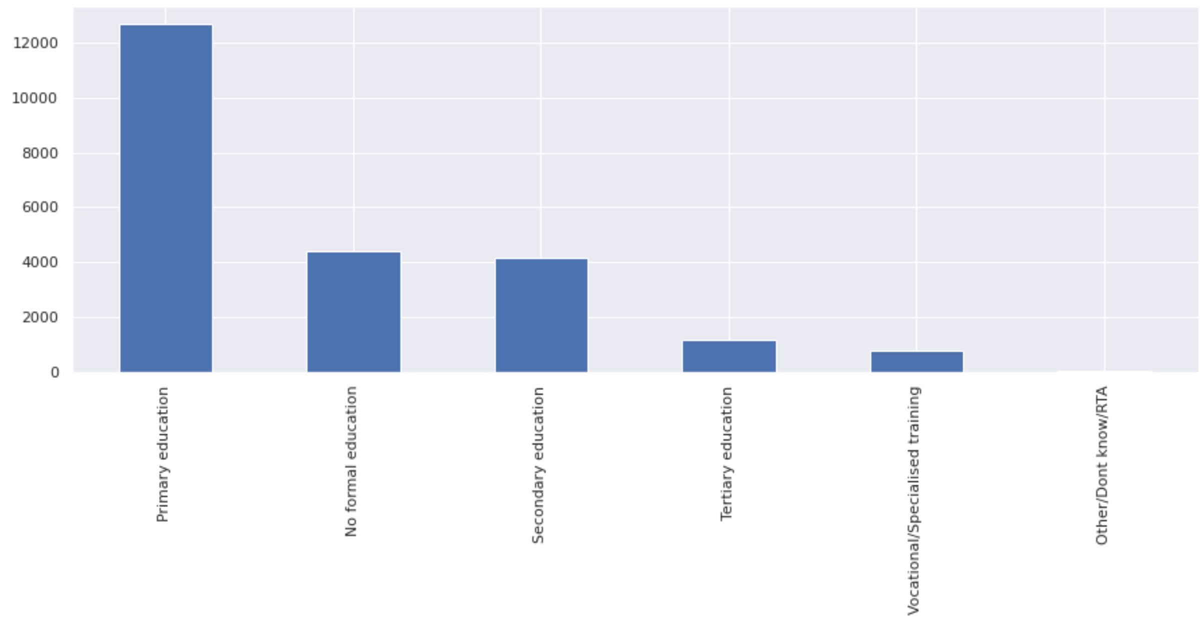
education_level

In [294]:

```
train['education_level'].value_counts().plot(kind='bar',figsize=[15,5])
```

Out[294]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95dac035c0>
```



In [0]:

```
# 7 categories, one category is called '6' and is undersampled; another cat
egory is 'Other/Dont know/RTA' is also undersampled
```

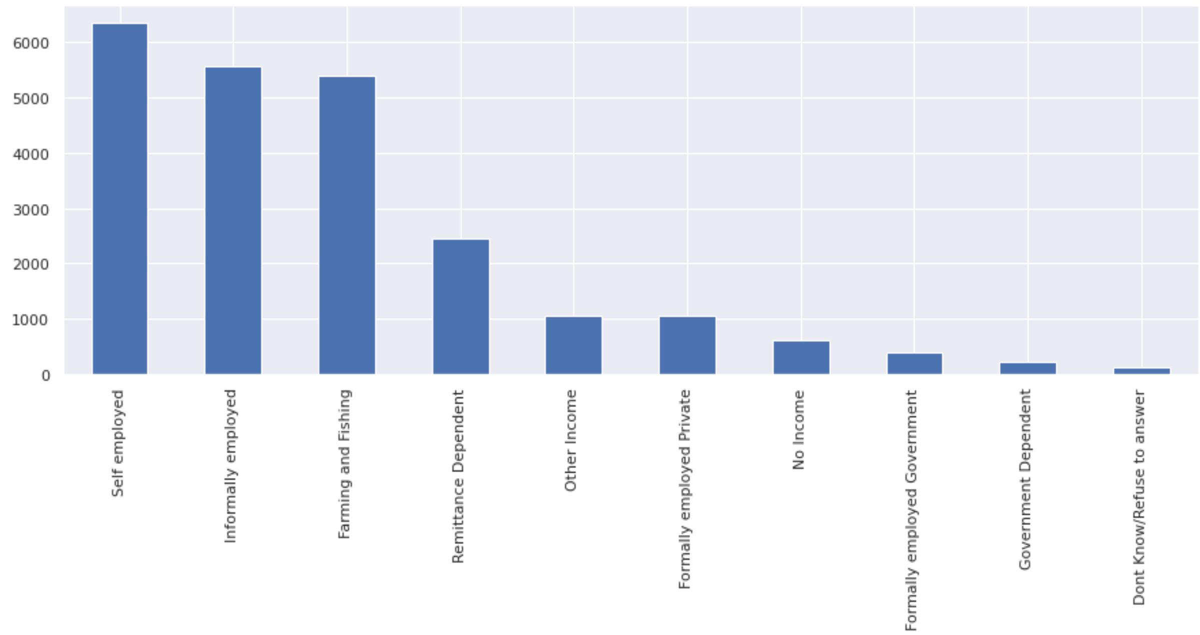
Job_type

In [296]:

```
train['job_type'].value_counts().plot(kind='bar',figsize=[15,5])
```

Out[296]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7c2eda0>
```



Let's check the year variable

In [297]:

```
test.year.value_counts()
```

Out[297]:

```
2016    3745
2018    3502
2017    2839
Name: year, dtype: int64
```

In [298]:

```
train.year.value_counts()
```

Out[298]:

```
2016    8678
2018    7974
2017    6580
Name: year, dtype: int64
```

In [0]:

```
# Same thing, the year seems to have been a condition to be respected when
splitting the train/test
```

In [300]:

```
train[train.year==2016].country.value_counts()
```

Out[300]:

```
Rwanda    8678
Name: country, dtype: int64
```

In [301]:

```
train[train.year==2017].country.value_counts()
```

Out[301]:

```
Tanzania    6580
Name: country, dtype: int64
```

In [302]:

```
train[train.year==2018].country.value_counts()
```

Out[302]:

```
Kenya     5978
Uganda    1996
Name: country, dtype: int64
```

In [0]:

```
# The year variable is indicative of which country is mentionned. That's ho
w the train set is made.
```

In [0]:

```
train['bank_account'].replace({'No': 0, 'Yes': 1}, inplace = True)
```

In [0]:

```
#labels = ['0-19','20-29','30-39','40-49','50-59','60-69','70-79','80-100']
#train['age_group']= pd.cut(train.age_of_respondent, range(0,81,10), right=
False, labels=labels)
```

age_group

In [0]:

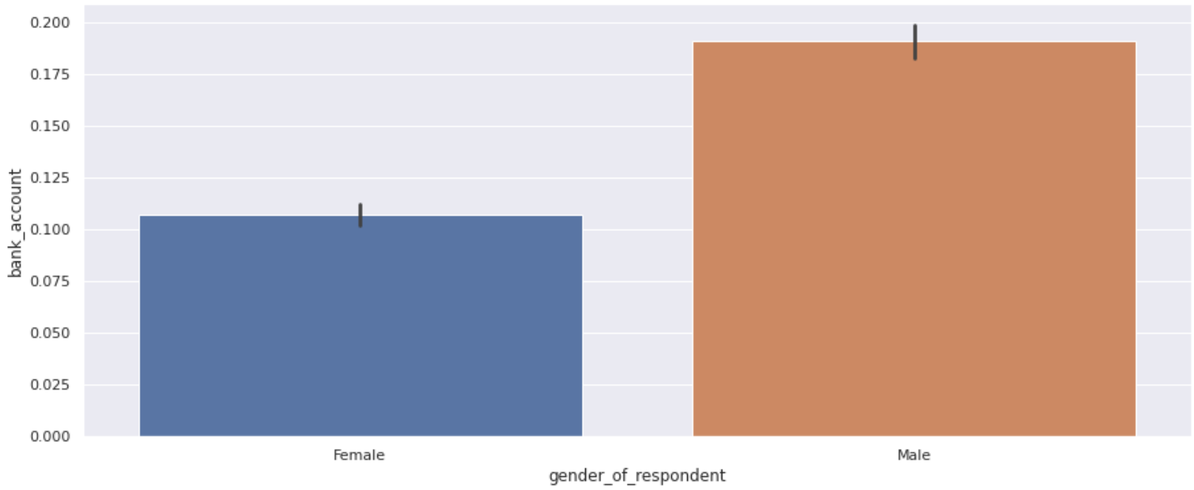
```
#plt.figure(figsize=[18,12])
#sns.barplot('age_group', 'bank_account', data=train)
```

In [307]:

```
plt.figure(figsize=[15,6])
sns.barplot('gender_of_respondent', 'bank_account', data=train)
```

Out[307]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f95d7c2ec50&gt;
```



In [0]:

```
# Males are more likely to have a bank account according to this plot
```

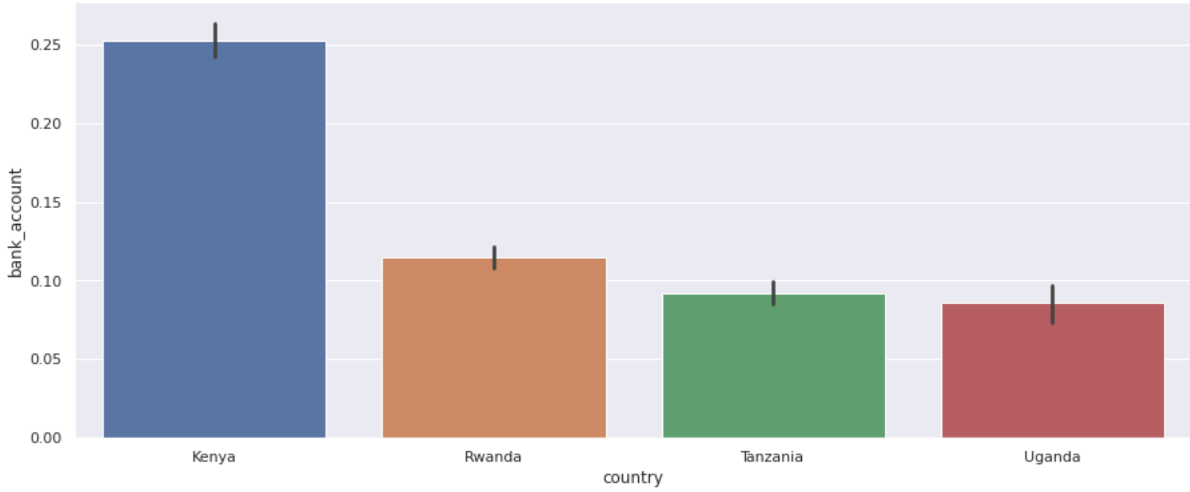
Country

In [309]:

```
plt.figure(figsize=[15,6])
sns.barplot('country', 'bank_account', data=train)
```

Out[309]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f95d7c256a0&gt;
```



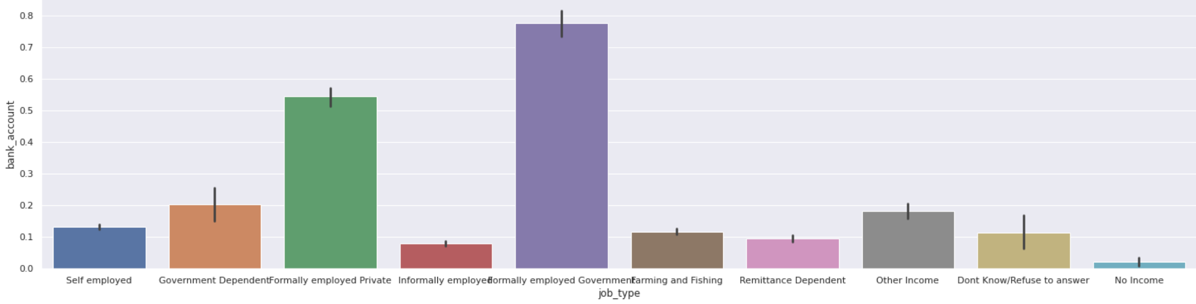
Job Type

In [310]:

```
plt.figure(figsize=[25,6])
sns.barplot('job_type', 'bank_account', data=train)
```

Out[310]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7af7d68>
```



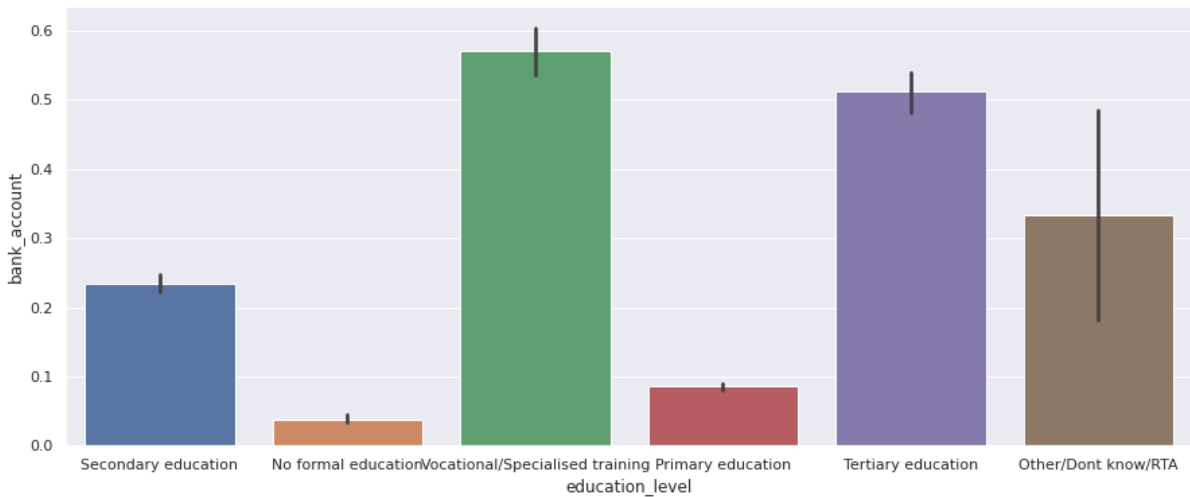
Education_Level

In [311]:

```
plt.figure(figsize=[15,6])
sns.barplot('education_level', 'bank_account', data=train)
```

Out[311]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7a190b8>
```



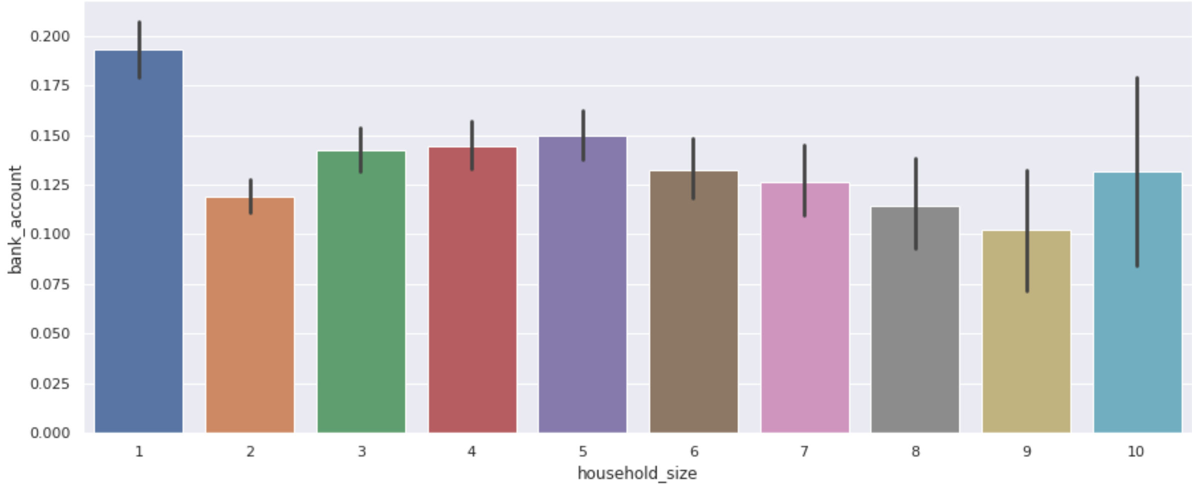
Household_Size

In [312]:

```
plt.figure(figsize=[15,6])
sns.barplot('household_size', 'bank_account', data=train)
```

Out[312]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7af7b38>
```



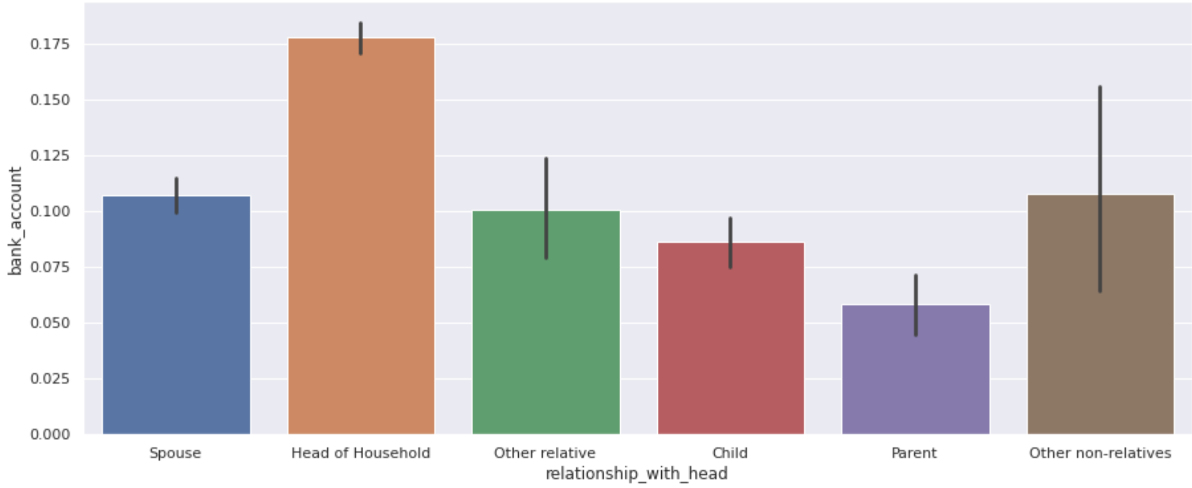
Relationship_with_head

In [313]:

```
plt.figure(figsize=[15,6])
sns.barplot('relationship_with_head', 'bank_account', data=train)
```

Out[313]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d791d390>
```



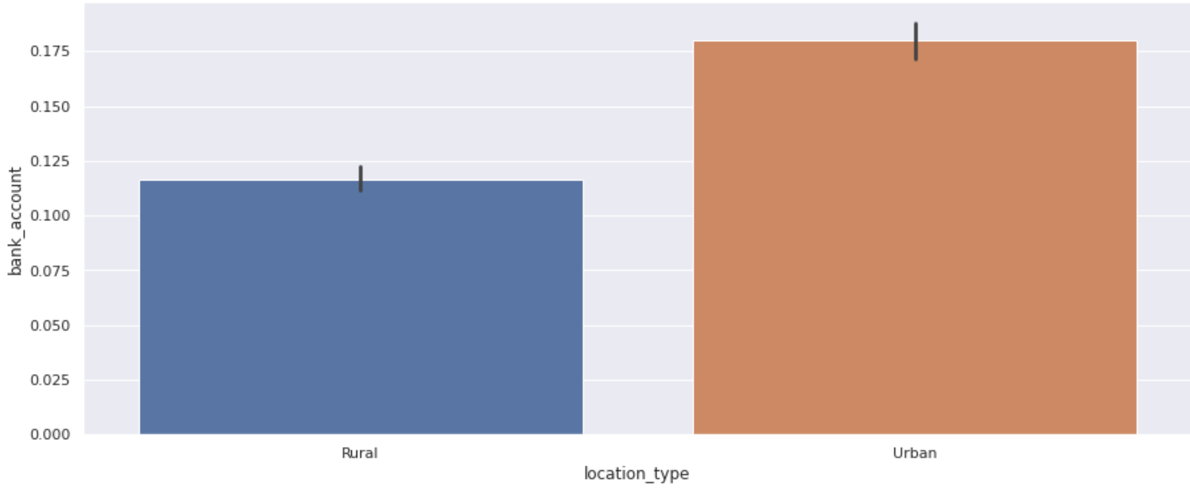
Location_Type

In [314]:

```
plt.figure(figsize=[15,6])
sns.barplot('location_type', 'bank_account', data=train)
```

Out[314]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d7917860>
```



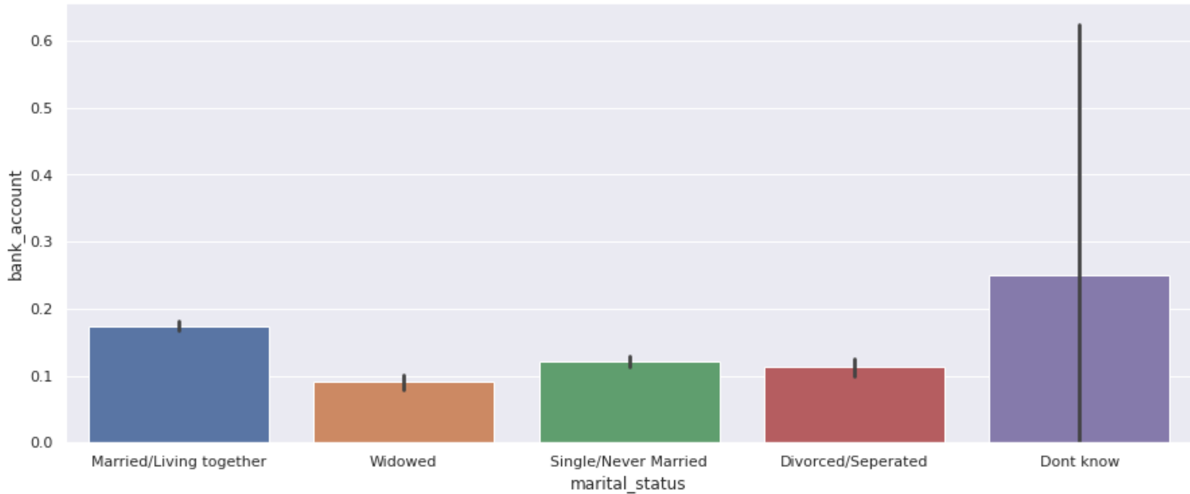
Marital Status

In [315]:

```
plt.figure(figsize=[15,6])
sns.barplot('marital_status', 'bank_account', data=train)
```

Out[315]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f95d798a5c0>
```



In [0]:

DATA MODELING

In [0]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import model_selection, preprocessing

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn import metrics
```

In [0]:

```
#Data Conversion for the Train Dataset
for e in train.columns:
    if train[e].dtype == 'object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train[e].values))
        train[e] = lbl.transform(list(train[e].values))
```

In [318]:

```
train.head()
```

Out[318]:

	country	year	uniqueid	bank_account	location_type	cellphone_access	household_size	age_of_respondent	gender_of_respondent	relationship_with_head	marital_status	education_level	job_type
0	0	2018	0	1	0	1	3	24	0	5	2	3	9
1	0	2018	1111	0	0	0	5	70	0	1	4	0	4
2	0	2018	2222	1	1	1	5	26	1	3	3	5	9
3	0	2018	3333	0	0	1	5	34	0	1	2	2	3
4	0	2018	4444	0	1	0	8	26	1	0	3	2	5

In [319]:

```
train['country'].unique()
```

Out[319]:

```
array([0, 1, 2, 3])
```

In [0]:

```
train = train.drop(columns=['year','uniqueid','household_size','education_level','marital_status','relationship_with_head'],axis=1)
```

In [321]:

```
train.head()
```

Out[321]:

	country	bank_account	location_type	cellphone_access	age_of_respondent	gender_of_respondent	job_type
0	0	1	0	1	24	0	9
1	0	0	0	0	70	0	4
2	0	1	1	1	26	1	9
3	0	0	0	1	34	0	3
4	0	0	1	0	26	1	5

Train and test split

In [0]:

```
#X_train, X_test, y_train, y_test = train_test_split(X,Y , test_size = 0.3)
# Splitt+ing the data from the train dataset
```

In [0]:

```
X = train.loc[:,train.columns!='bank_account']
```

In [0]:

```
Y = train['bank_account']
```

In [0]:

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

In [0]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y , test_size = 0.3)
```

In [327]:

```
X_train.shape, X_test.shape
```

Out[327]:

```
((16262, 6), (6970, 6))
```

Build the model on training data

In [0]:

```
# Hypothesis statement

# Ho is not significant
# HR is significant
# P-value < 5%
```

In [0]:

```
# import statsmodels.api as sm
```

In [0]:

```
# model_1 = sm.OLS(Y_train,X_train).fit()
```

In [0]:

```
# model_1.summary2()
```

In [0]:

```
# P-value < 5%
# HR: True
# H0: False
```

Logistic Regression Model

In [333]:

```
#Training
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression()
model1.fit(X_train,Y_train)
```

Out[333]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True
,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0
,
                   warm_start=False)
```

In [334]:

```
#Testing
predicted = model1.predict(X_test)
predicted
```

Out[334]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [335]:

```
#Evaluation
#Confusion Matrix
print(metrics.confusion_matrix(Y_test, predicted))
```

```
[[5956   16]
 [ 978   20]]
```

In [336]:

```
#Classification Report
print("\nAcuracy Score of LogisticRegression Model:")
print(metrics.accuracy_score(Y_test, predicted))
print("\nClassification Report:")
print(metrics.classification_report(Y_test, predicted))
```

```
Acuracy Score of LogisticRegression Model:
0.8573888091822095

Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      5972
           1       0.56      0.02      0.04       998

    accuracy                           0.86      6970
```

```
   macro avg       0.71      0.51      0.48      6970
weighted avg       0.82      0.86      0.80      6970
```

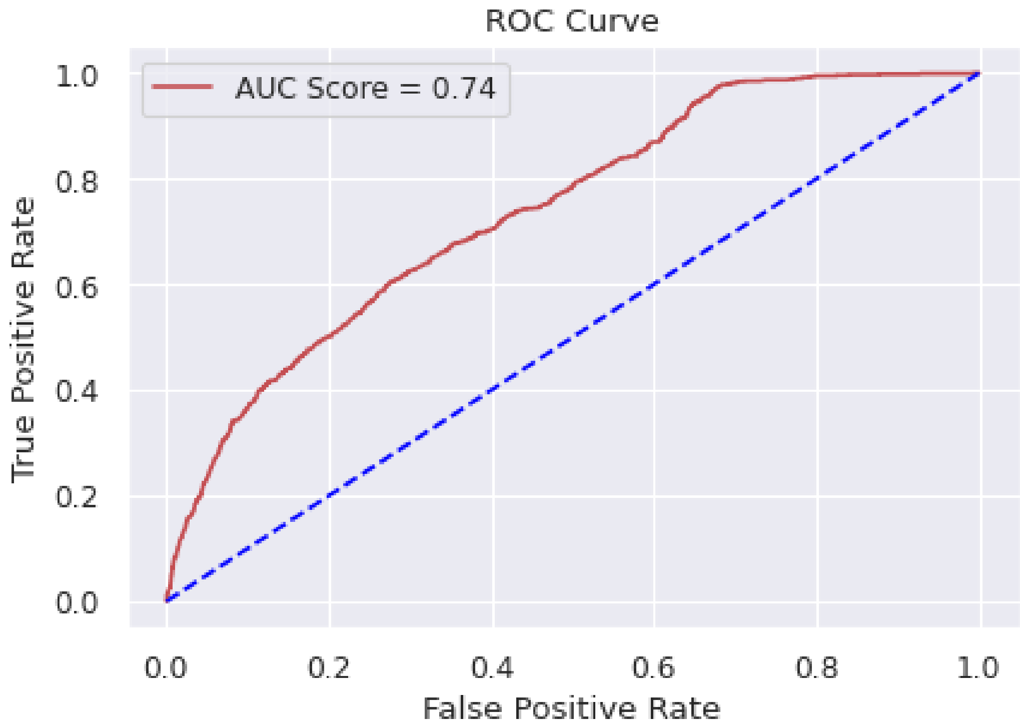
In [337]:

```
# Using the ROC CUrve to See the accuracy of our Model
print("ROC Curve")
model1_prob = model1.predict_proba(X_test)
model1_prob1 = model1_prob[:,1]
fpr,tpr,thresh = metrics.roc_curve(Y_test,model1_prob1)
roc_auc_lr = metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.plot(fpr,tpr,'r',label = 'AUC Score = %0.2f'%roc_auc_lr)
plt.plot(fpr,fpr,'b--',color='blue')
plt.legend()
```

```
ROC Curve
```

Out[337]:

```
<matplotlib.legend.Legend at 0x7f95d77182b0>
```



Random Forest Model

In [338]:

```
#Training
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier()
model2.fit(X_train,Y_train)
```

Out[338]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto
',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [339]:

```
#Testing
predicted = model2.predict(X_test)
predicted
```

Out[339]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [340]:

```
#Evaluation
#Confusion Matrix
print(metrics.confusion_matrix(Y_test, predicted))
```

```
[[5616  356]
 [ 662  336]]
```

In [341]:

```
#Classification Report
print("\nAcuracy Score of RF Model:")
print(metrics.accuracy_score(Y_test, predicted))
print("\nClassification Report:")
print(metrics.classification_report(Y_test, predicted))
```

```
Acuracy Score of RF Model:
0.853945480631277

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      5972
           1       0.49      0.34      0.40       998

    accuracy                           0.85      6970
   macro avg       0.69      0.64      0.66      6970
weighted avg       0.84      0.85      0.84      6970
```

In [342]:

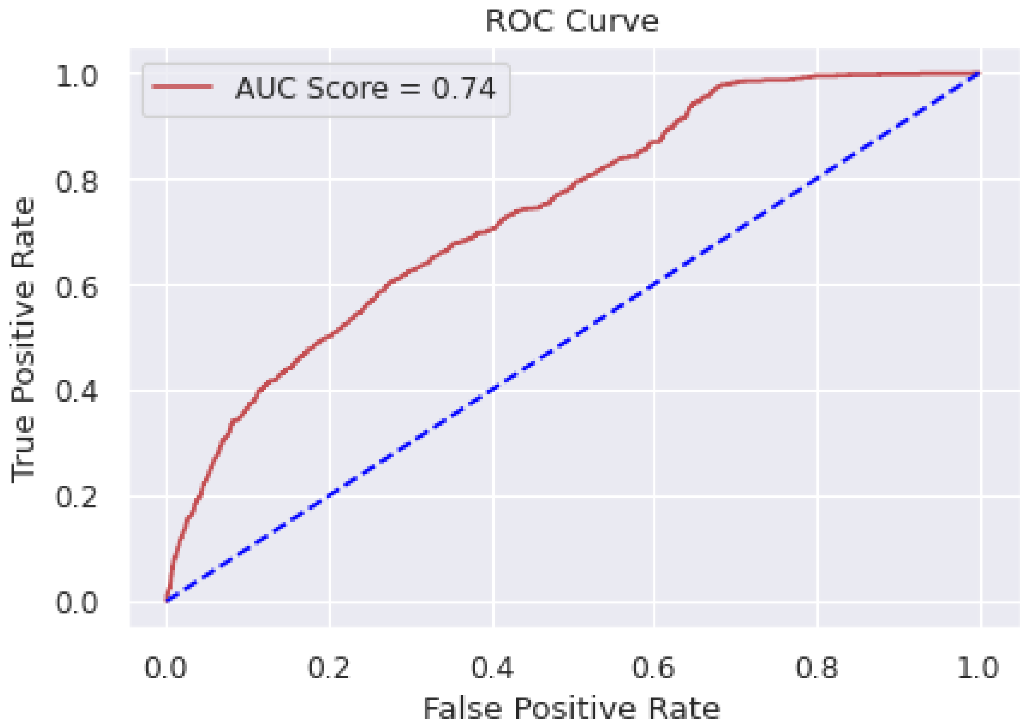
```
print("ROC Curve")
model2_prob = model2.predict_proba(X_test)
model2_prob1 = model1_prob[:,1]
fpr,tpr,thresh = metrics.roc_curve(Y_test,model2_prob1)
roc_auc_rf = metrics.auc(fpr,tpr)
```

```
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.plot(fpr,tpr,'r',label = 'AUC Score = %0.2f'%roc_auc_rf)
plt.plot(fpr,fpr,'b--',color='blue')
plt.legend()
```

ROC Curve

Out[342]:

<matplotlib.legend.Legend at 0x7f95d768c908>



KNN Model

In [343]:

```
#Training
from sklearn import neighbors
model3 = neighbors.KNeighborsClassifier()

model3.fit(X_train,Y_train)
```

Out[343]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [344]:

```
#Testing
predicted = model3.predict(X_test)
predicted
```

Out[344]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [345]:

```
#Evaluation
#Confusion Matrix
print(metrics.confusion_matrix(Y_test, predicted))
```

```
[[5741  231]
 [ 704  294]]
```

In [346]:

```
#Classification Report
print("\nAcuracy Score of KNN Model:")
print(metrics.accuracy_score(Y_test, predicted))
print("\nClassification Report:")
print(metrics.classification_report(Y_test, predicted))
```

```
Acuracy Score of KNN Model:
0.8658536585365854

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.96      0.92      5972
           1       0.56      0.29      0.39       998

    accuracy                           0.87      6970
   macro avg       0.73      0.63      0.66      6970
weighted avg       0.84      0.87      0.85      6970
```

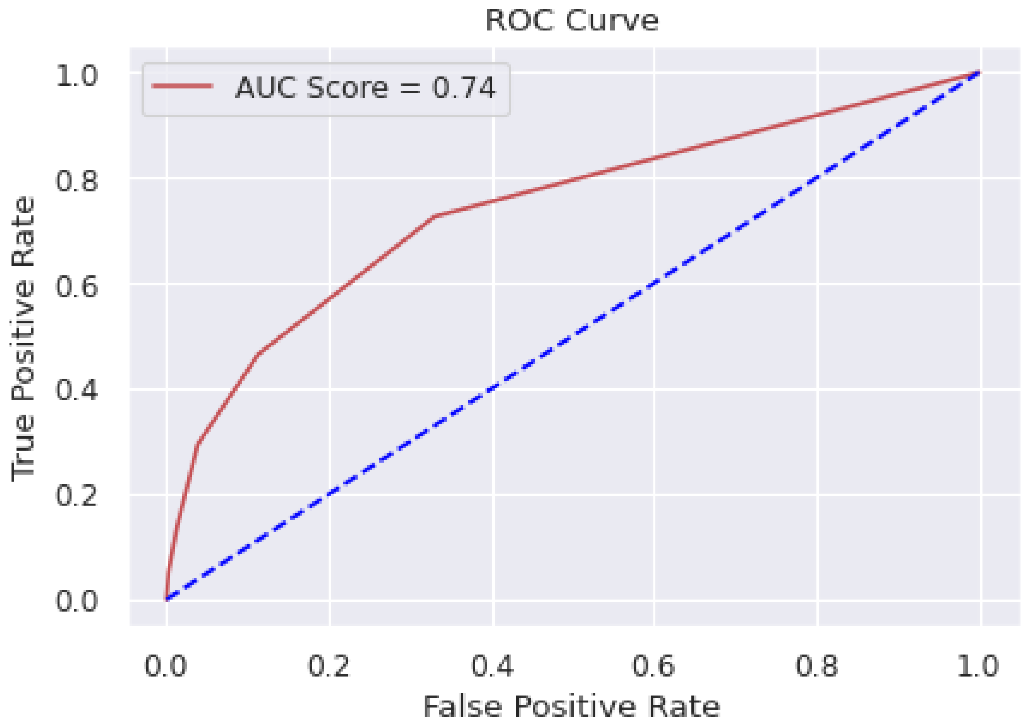
In [347]:

```
print("ROC Curve")
model3_prob = model3.predict_proba(X_test)
model3_prob1 = model3_prob[:,1]
fpr,tpr,thresh = metrics.roc_curve(Y_test,model3_prob1)
roc_auc_knn = metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.plot(fpr,tpr,'r',label = 'AUC Score = %0.2f'%roc_auc_knn)
plt.plot(fpr,fpr,'b--',color='blue')
plt.legend()
```

```
ROC Curve
```

Out[347]:

```
<matplotlib.legend.Legend at 0x7f95d76746a0>
```


ROC Curve
AUC Score = 0.74
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

In [0]:

In [0]: