In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import random
```

In [2]:

```
df = pd.read_csv("bank-full.csv")
df
```

Out[2]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	
1	44	technician	single	secondary	no	29	yes	no	unknown	5	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	
4	33	unknown	single	unknown	no	1	no	no	unknown	5	
...	...	...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	

45211 rows × 17 columns

In [3]:

```
df.head()
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may

In [4]:

```
df.shape
```

Out[4]: (45211, 17)

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
```

```
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64 
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64 
 12  campaign   45211 non-null  int64 
 13  pdays      45211 non-null  int64 
 14  previous   45211 non-null  int64 
 15  poutcome   45211 non-null  object
 16  Target     45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
Target       0
dtype: int64
```

In [7]:
```
df.describe(include = 'all')
```

Out[7]:

	age	job	marital	education	default	balance	housing	loan	contact
count	45211.000000	45211	45211	45211	45211	45211.000000	45211	45211	45211
unique	NaN	12	3	4	2	NaN	2	2	3
top	NaN	blue-collar	married	secondary	no	NaN	yes	no	cellular
freq	NaN	9732	27214	23202	44396	NaN	25130	37967	29285
mean	40.936210	NaN	NaN	NaN	NaN	1362.272058	NaN	NaN	NaN
std	10.618762	NaN	NaN	NaN	NaN	3044.765829	NaN	NaN	NaN
min	18.000000	NaN	NaN	NaN	NaN	-8019.000000	NaN	NaN	NaN
25%	33.000000	NaN	NaN	NaN	NaN	72.000000	NaN	NaN	NaN
50%	39.000000	NaN	NaN	NaN	NaN	448.000000	NaN	NaN	NaN

	age	job	marital	education	default	balance	housing	loan	contact
75%	48.000000	NaN	NaN	NaN	NaN	1428.000000	NaN	NaN	NaN
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	NaN	NaN

In [8]:

```
df.dtypes
```

Out[8]:

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

Manipulating the data

In [9]:

```
#Creating User Columns
df_user = pd.DataFrame(np.arange(0,len(df)), columns=['user'])
df = pd.concat([df_user, df], axis=1)
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 18 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   user       45211 non-null  int32 
 1   age        45211 non-null  int64 
 2   job        45211 non-null  object
 3   marital    45211 non-null  object
 4   education  45211 non-null  object
 5   default    45211 non-null  object
 6   balance    45211 non-null  int64 
 7   housing    45211 non-null  object
 8   loan       45211 non-null  object
 9   contact    45211 non-null  object
 10  day        45211 non-null  int64 
 11  month      45211 non-null  object
 12  duration   45211 non-null  int64 
 13  campaign   45211 non-null  int64 
 14  pdays      45211 non-null  int64 
 15  previous   45211 non-null  int64 
 16  poutcome   45211 non-null  object
 17  Target     45211 non-null  object
dtypes: int32(1), int64(7), object(10)
memory usage: 6.0+ MB
```

In [11]:
```
df.columns.values
```

Out[11]:
```
array(['user', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration',
       'campaign', 'pdays', 'previous', 'poutcome', 'Target'],
      dtype=object)
```

In [12]:
```
df.groupby('Target').mean()
```

Out[12]:

	user	age	balance	day	duration	campaign	pdays	previous
Target								
no	21197.503081	40.838986	1303.714969	15.892290	221.182806	2.846350	36.421372	0.502154
yes	33228.953867	41.670070	1804.267915	15.158253	537.294574	2.141047	68.702968	1.170354

In [13]:
```
df['Target'].value_counts()
```

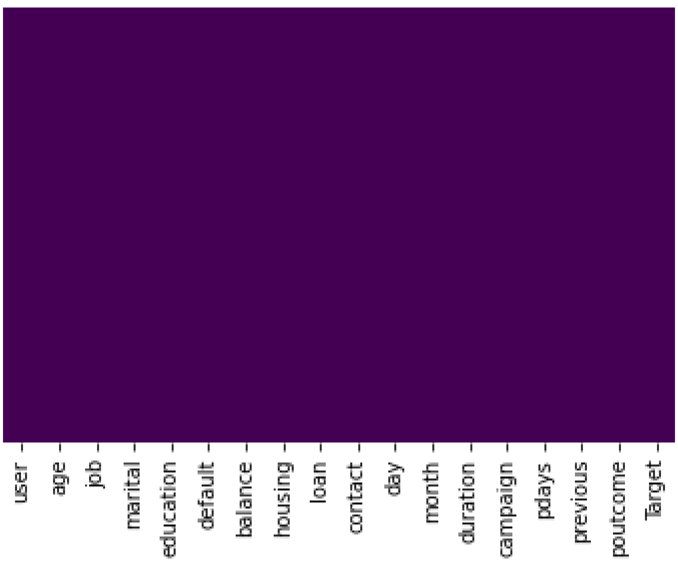
Out[13]:
```
no     39922
yes     5289
Name: Target, dtype: int64
```

In [14]:
```
countNo = len(df[df.Target == 'no'])
countYes = len(df[df.Target == 'yes'])
print('Percentage of "No": {:.3f}%'. format((countNo/(len(df.Target))*100)))
print('Percentage of "Yes": {:.3f}%'. format((countYes/(len(df.Target))*100)))
```

```
Percentage of "No": 88.302%
Percentage of "Yes": 11.698%
```

In [15]:
```
#Verifying null values
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

Out[15]: <AxesSubplot:>



In [16]:
```
df.isna().any()
```

Out[16]:
```
user          False
age           False
```

```
job          False
marital      False
education    False
default      False
balance      False
housing      False
loan         False
contact      False
day          False
month        False
duration     False
campaign     False
pdays        False
previous     False
poutcome     False
Target       False
dtype: bool
```

In [17]:

```
df.isna().sum()
```

Out[17]:

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

In [18]:

```
#Define X and y
X = df.drop(['Target','user','job','marital', 'education', 'contact',
             'housing', 'loan', 'day', 'month', 'poutcome' ], axis=1)
y = df['Target']
```

In [19]:

```
X = pd.get_dummies(X)
y = pd.get_dummies(y)
```

In [20]:

```
X.columns
X = X.drop(['default_no'], axis= 1)
X = X.rename(columns = {'default_yes': 'default'})
y.columns
y = y.drop(['yes'], axis=1)
y = y.rename(columns= {'no': 'y'})
```

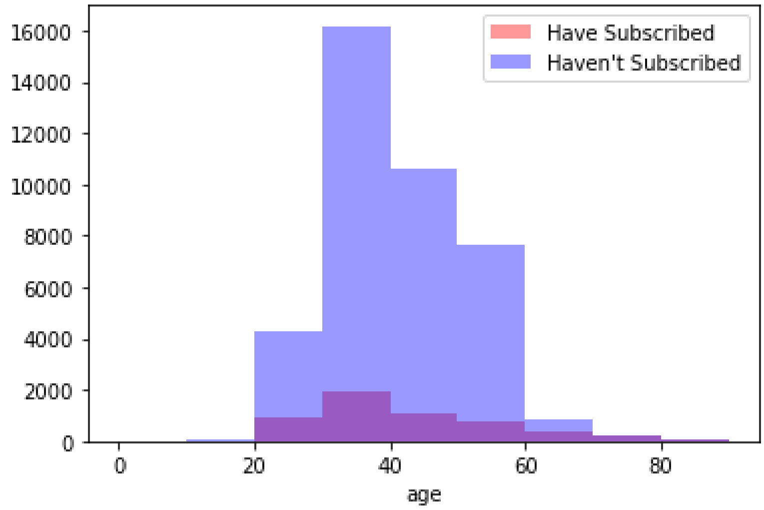
visualizing data

In [21]:

```
#Age group
bins = range(0, 100, 10)
ax = sns.distplot(df.age[df.Target=='yes'],
```

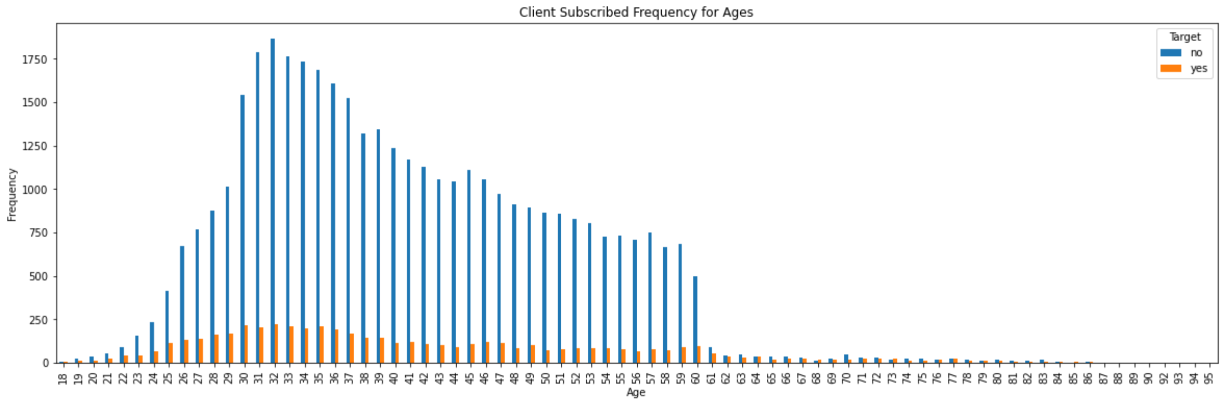
```
                 color='red', kde=False, bins=bins, label='Have Subscribed')
sns.distplot(df.age[df.Target=='no'],
          ax=ax,  # Overplots on first plot
          color='blue', kde=False, bins=bins, label="Haven't Subscribed")
plt.legend()
plt.show()
```

```
C:\Users\Admin\sachin\lib\site-packages\seaborn\distributions.py:2557: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



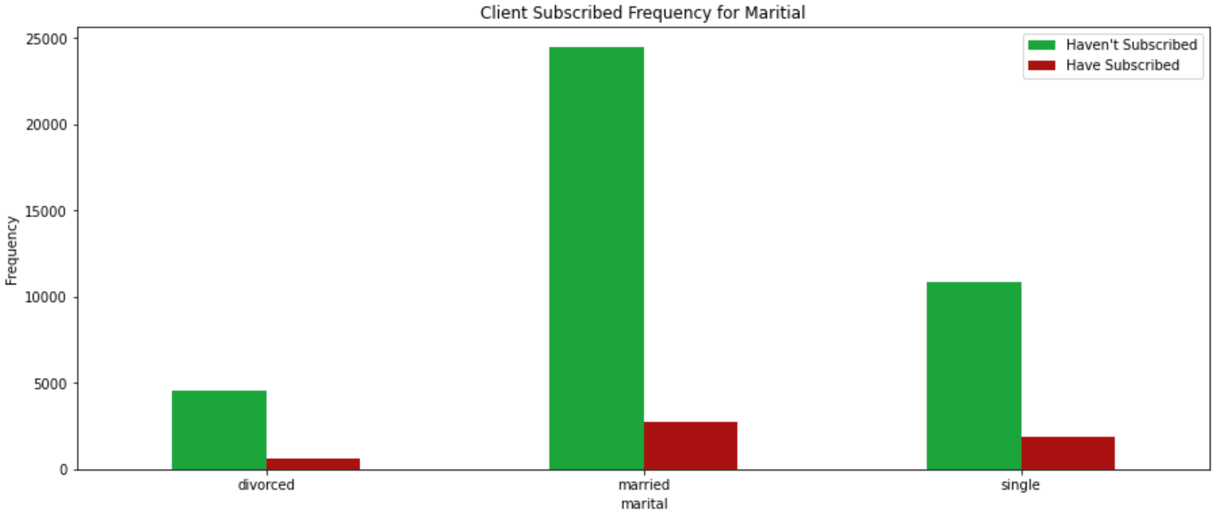
In [22]:

```
#Age
pd.crosstab(df.age,df.Target).plot(kind="bar",figsize=(20,6))
plt.title('Client Subscribed Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



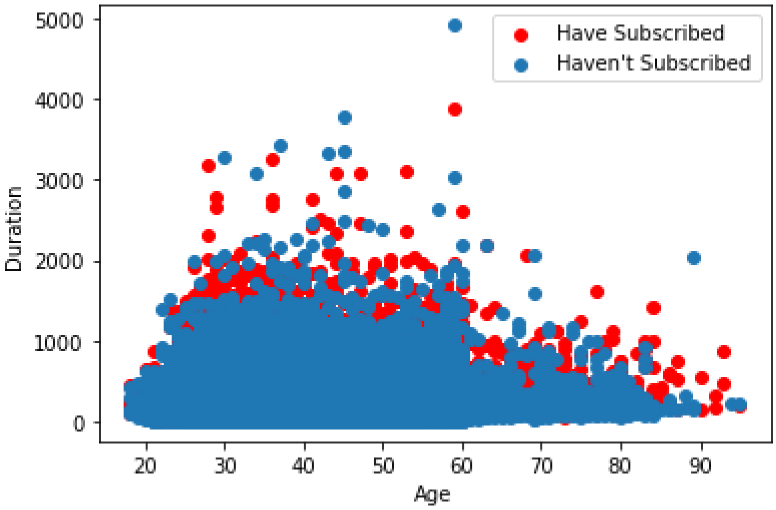
In [23]:

```
pd.crosstab(df.marital,df.Target).plot(kind="bar",figsize=(15,6),color=['#1CA53B','#
plt.title('Client Subscribed Frequency for Maritial')
plt.xlabel('marital')
plt.xticks(rotation=0)
plt.legend(["Haven't Subscribed", "Have Subscribed"])
plt.ylabel('Frequency')
plt.show()
```



In [24]:

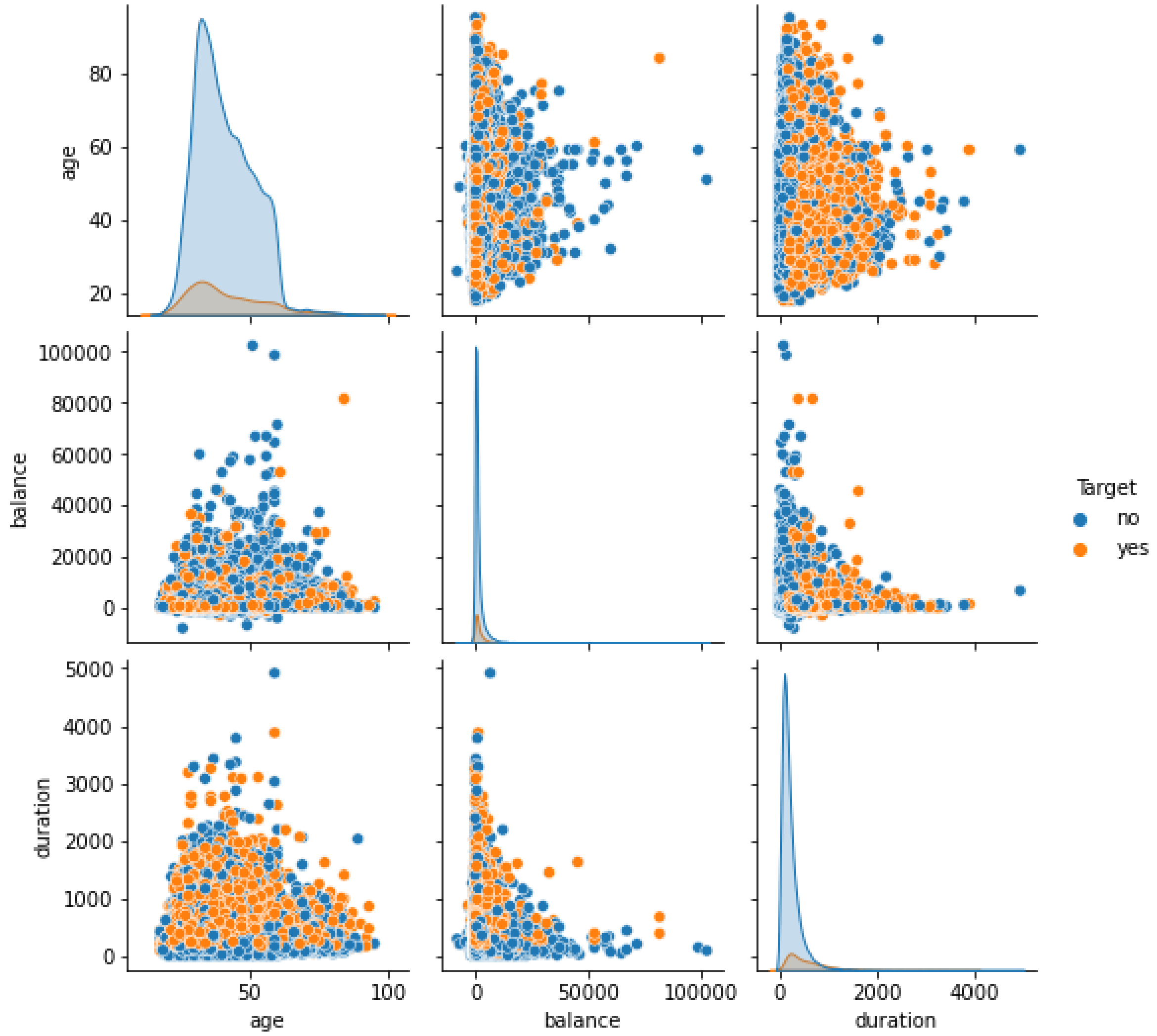
```
plt.scatter(x=df.age[df.Target=='yes'], y=df.duration[(df.Target=='yes')], c="red")
plt.scatter(x=df.age[df.Target=='no'], y=df.duration[(df.Target=='no')])
plt.legend(["Have Subscribed", "Haven't Subscribed"])
plt.xlabel("Age")
plt.ylabel("Duration")
plt.show()
```



In [25]:

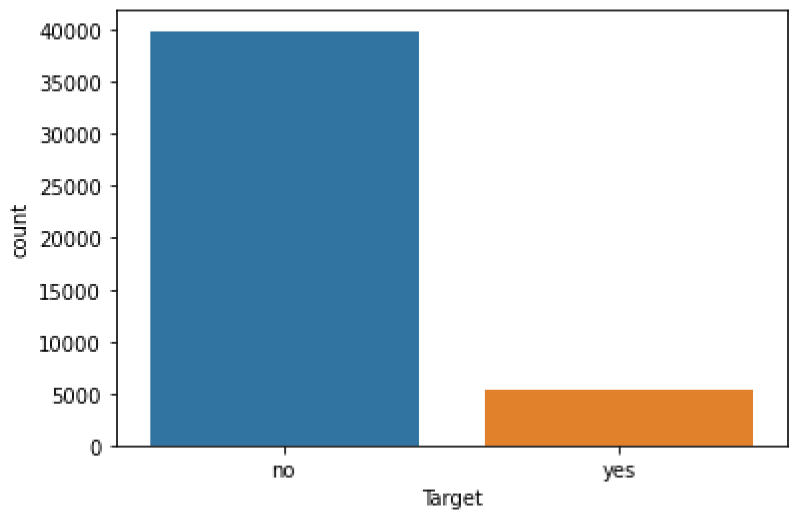
```
sns.pairplot(data=df, hue='Target', vars= ['age', 'balance', 'duration'])
```

Out[25]: <seaborn.axisgrid.PairGrid at 0x1c9abf612e0>



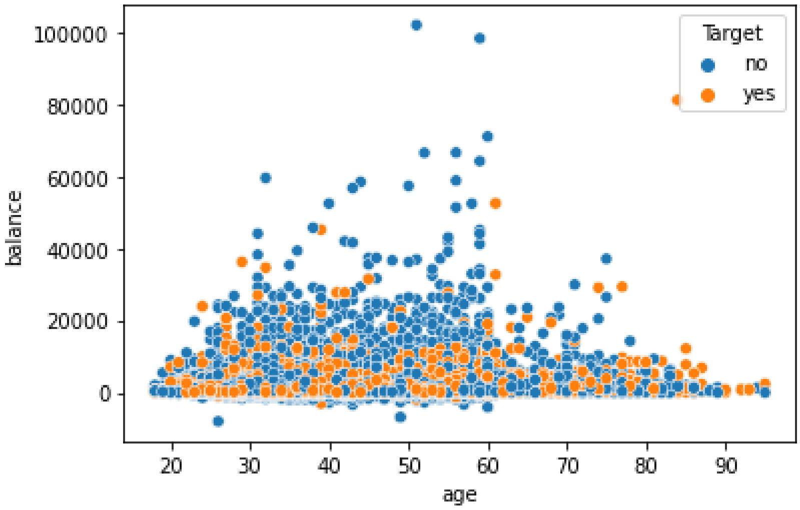
In [26]:
```
sns.countplot(x='Target', data=df, label='Count')
```

Out[26]: <AxesSubplot:xlabel='Target', ylabel='count'>



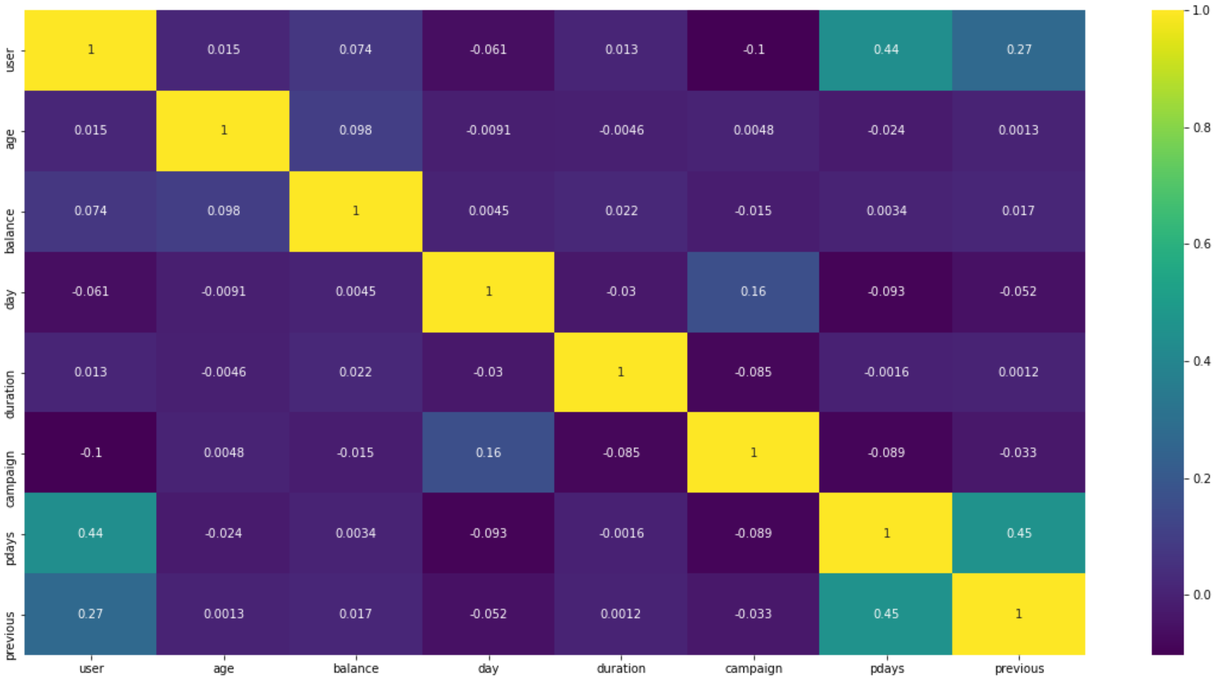
In [27]:
```
sns.scatterplot(x='age', y='balance',hue='Target', data=df)
```

Out[27]: <AxesSubplot:xlabel='age', ylabel='balance'>



In [28]:
```python
plt.figure(figsize=(20,10))
sns.heatmap(data=df.corr(), annot=True, cmap='viridis')
```

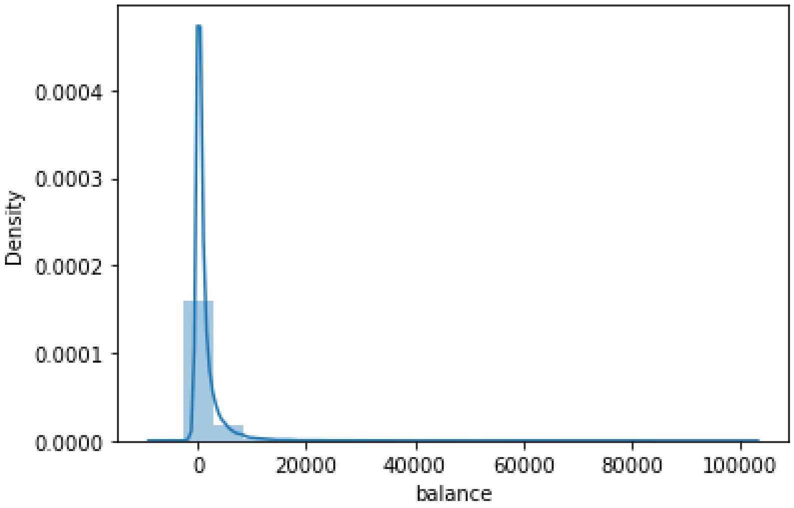
Out[28]: <AxesSubplot:>



In [29]:
```python
import warnings
warnings.filterwarnings("ignore")
```

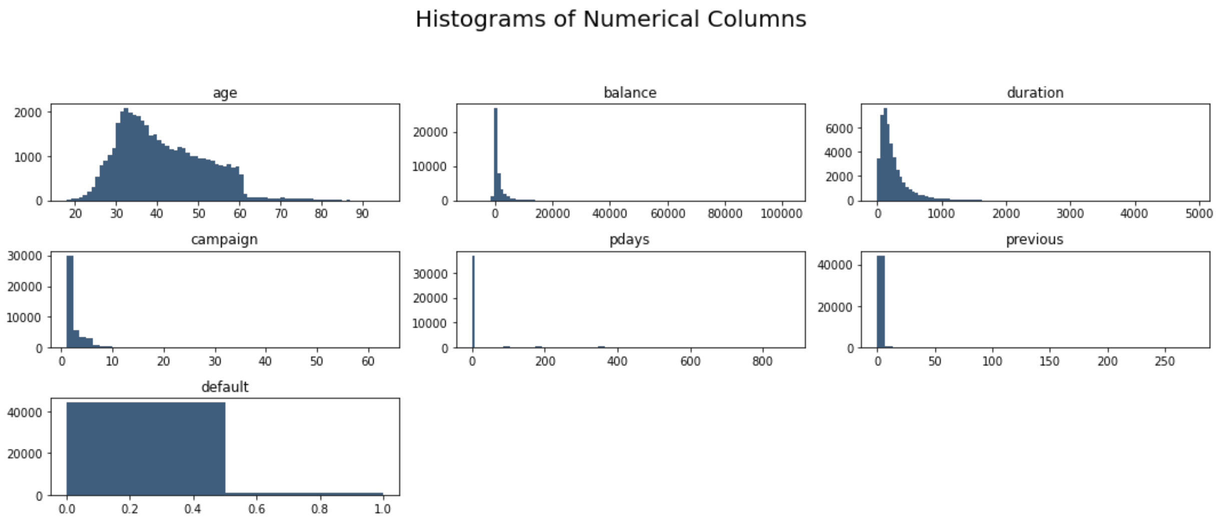
In [30]:
```python
sns.distplot(df.balance, bins = 20)
```

Out[30]: <AxesSubplot:xlabel='balance', ylabel='Density'>



In [31]:

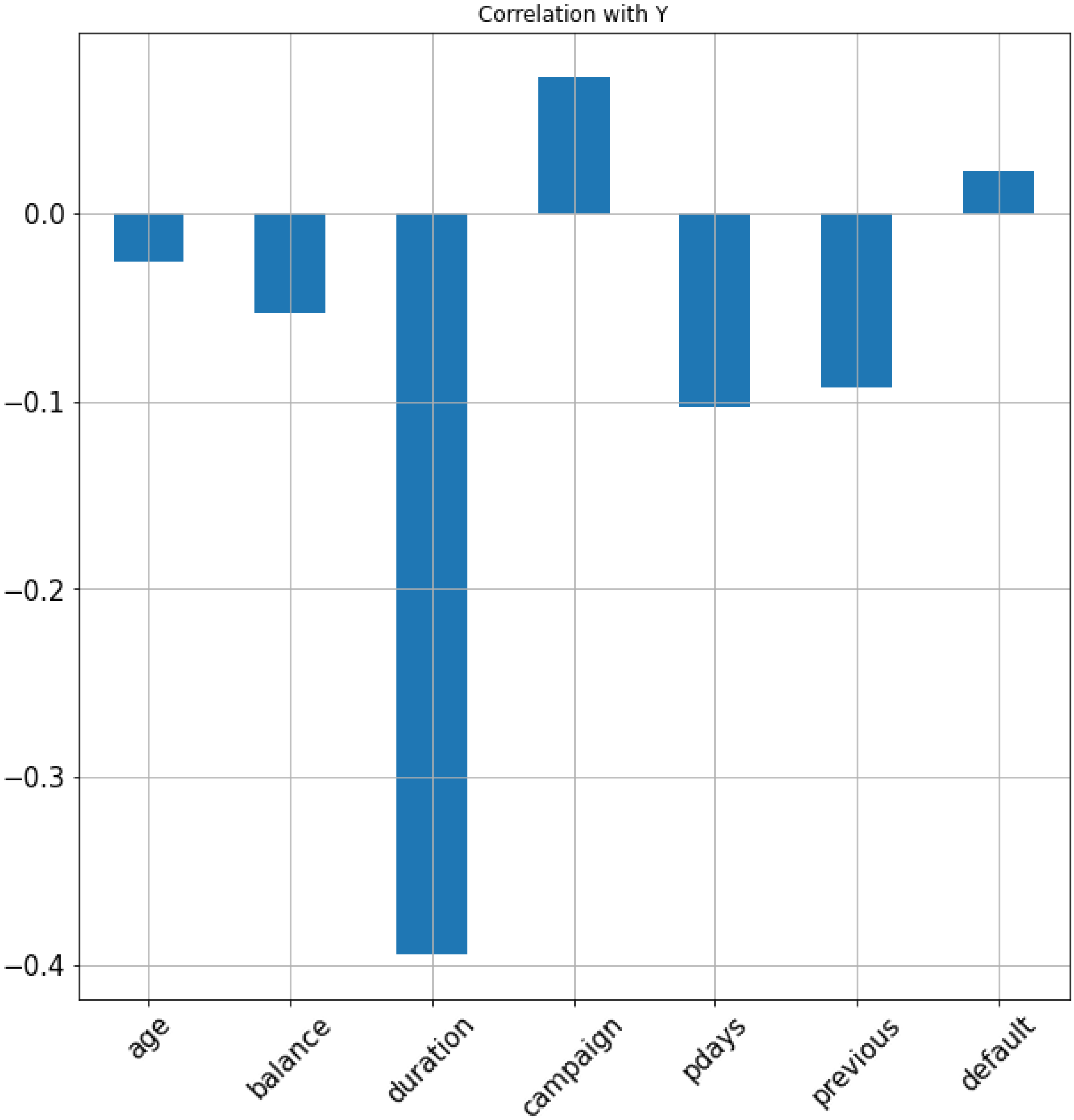
```
df2 = X
fig = plt.figure(figsize=(15, 12))
plt.suptitle('Histograms of Numerical Columns', fontsize=20)
for i in range(df2.shape[1]):
    plt.subplot(6, 3, i + 1)
    f = plt.gca()
    f.set_title(df2.columns.values[i])

    vals = np.size(df2.iloc[:, i].unique())
    if vals >= 100:
        vals = 100

    plt.hist(df2.iloc[:, i], bins=vals, color='#3F5D7D')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [32]:

```
## Correlation with independent Variable
df2.corrwith(y.y).plot.bar(
        figsize = (10, 10), title = "Correlation with Y", fontsize = 15,
        rot = 45, grid = True)
```

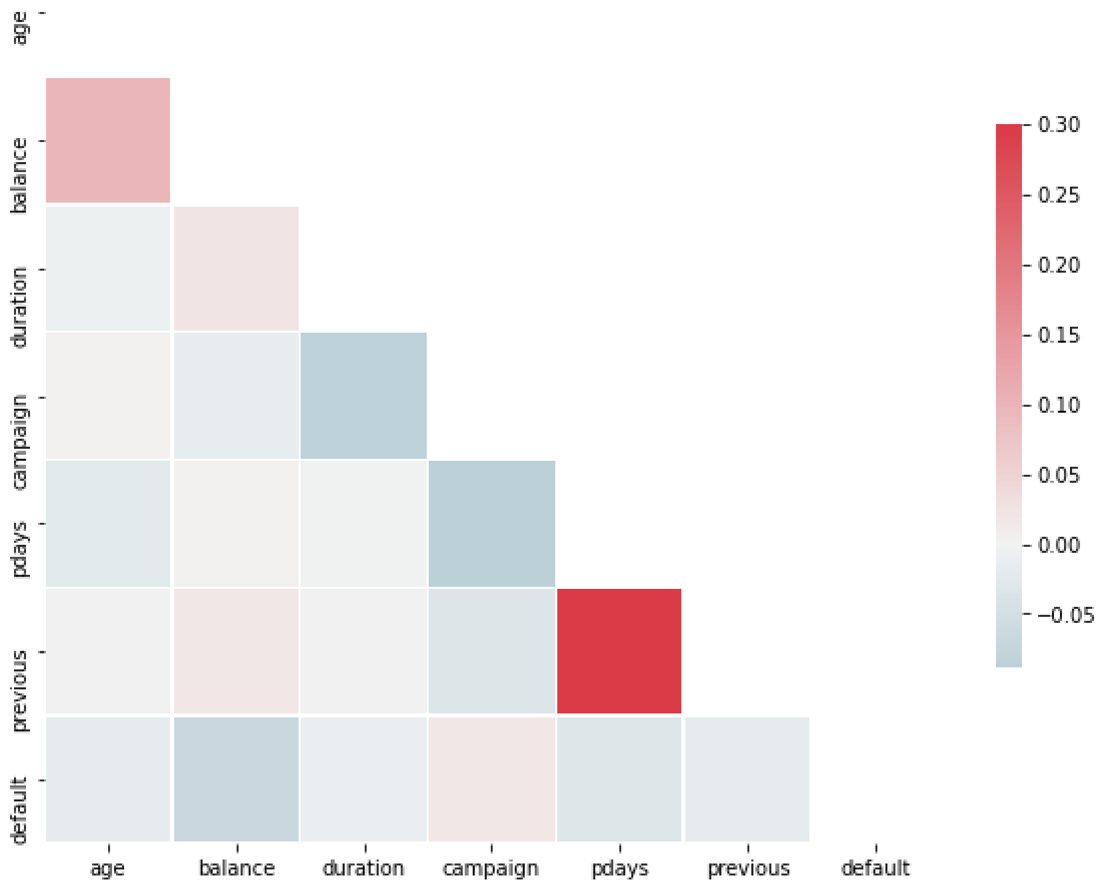
Out[32]: <AxesSubplot:title={'center':'Correlation with Y'}>



In [33]:

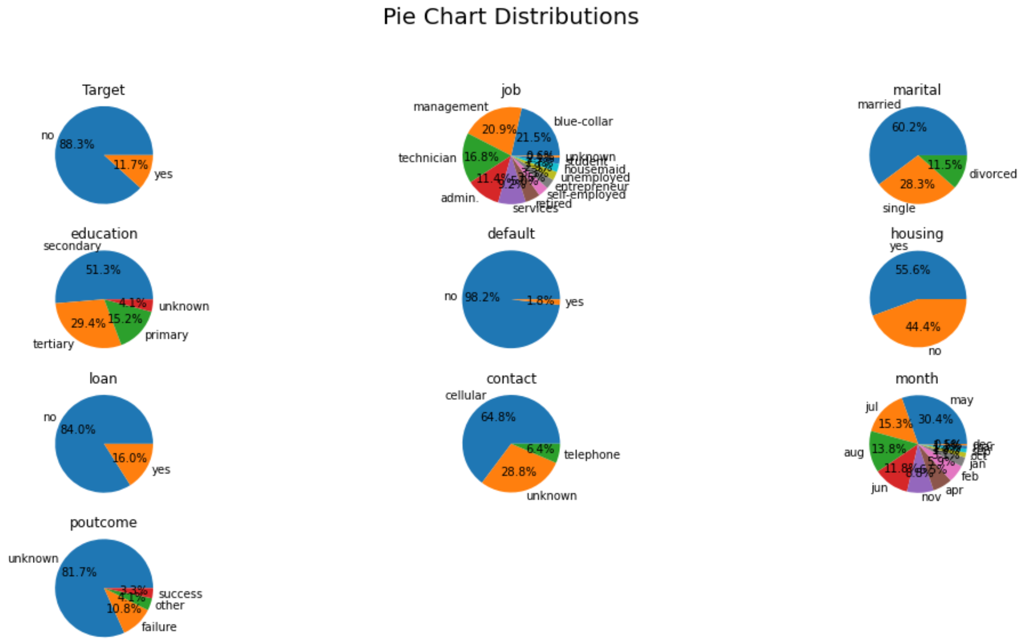
```
# Compute the correlation matrix
corr = df2.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(10, 10))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[33]: <AxesSubplot:>



In [34]:

```
## Pie Plots
df.columns
df2 = df[['Target','job','marital', 'education', 'default', 'housing','loan', 'conta
             'month', 'poutcome'
                    ]]
fig = plt.figure(figsize=(15, 12))
plt.suptitle('Pie Chart Distributions', fontsize=20)
for i in range(1, df2.shape[1] + 1):
    plt.subplot(6, 3, i)
    f = plt.gca()
    f.axes.get_yaxis().set_visible(False)
    f.set_title(df2.columns.values[i - 1])

    values = df2.iloc[:, i - 1].value_counts(normalize = True).values
    index = df2.iloc[:, i - 1].value_counts(normalize = True).index
    plt.pie(values, labels = index, autopct='%1.1f%%')
    plt.axis('equal')
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



Splitting data into traning and testing data

In [35]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
```

In [36]:

```
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(36168, 7) (9043, 7) (36168, 1) (9043, 1)
```

In [37]:

```
## Balance the data
y_train['y'].value_counts()
```

Out[37]:

```
1    31937
0     4231
Name: y, dtype: int64
```

In [38]:

```
pos_index = y_train[y_train.values == 1].index
neg_index = y_train[y_train.values == 0].index

if len(pos_index) > len(neg_index):
    higher = pos_index
    lower = neg_index
else:
    higher = neg_index
    lower = pos_index

random.seed(0)
higher = np.random.choice(higher, size=len(lower))
lower = np.asarray(lower)
new_indexes = np.concatenate((lower, higher))

X_train = X_train.loc[new_indexes]
y_train = y_train.loc[new_indexes]
```

In [39]:

```
y_train['y'].value_counts()
```

```
0    4231
```

Out[39]:
```
1    4231
Name: y, dtype: int64
```

Feature Scaling

In [40]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train2 = pd.DataFrame(sc.fit_transform(X_train))
X_test2 = pd.DataFrame(sc.transform(X_test))
X_train2.columns = X_train.columns.values
X_test2.columns = X_test.columns.values
X_train2.index = X_train.index.values
X_test2.index = X_test.index.values
X_train = X_train2
X_test = X_test2
```

Model Buliding

Comparing models

In [42]:
```
## Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, penalty = 'l2')
classifier.fit(X_train, y_train)

# Predicting Test Set
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_sc
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

results = pd.DataFrame([['Logistic Regression (Lasso)', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

In [48]:
```
results
```

Out[48]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335

In [50]:
```
## SVM (Linear)
from sklearn.svm import SVC
classifier = SVC(random_state = 0, kernel = 'linear', probability= True)
classifier.fit(X_train, y_train)
```

```
# Predicting Test Set
y_pred = classifier.predict(X_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['SVM (Linear)', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

results = results.append(model_results, ignore_index = True)
```

In [51]:

```
results
```

Out[51]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335
6	SVM (Linear)	0.797965	0.952661	0.811522	0.876446

In [44]:

```
## SVM (rbf)
from sklearn.svm import SVC
classifier = SVC(random_state = 0, kernel = 'rbf', probability= True)
classifier.fit(X_train, y_train)

# Predicting Test Set
y_pred = classifier.predict(X_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['SVM (RBF)', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

results = results.append(model_results, ignore_index = True)
```

In [52]:

```
results
```

Out[52]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335

	Model	Accuracy	Precision	Recall	F1 Score
6	SVM (Linear)	0.797965	0.952661	0.811522	0.876446

In [45]:
```
## Naive Bayes
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting Test Set
y_pred = classifier.predict(X_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Naive Bayes (Gaussian)', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

results = results.append(model_results, ignore_index = True)
```

In [53]:
```
results
```

Out[53]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335
6	SVM (Linear)	0.797965	0.952661	0.811522	0.876446

In [46]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(X_train, y_train)

#Predicting the best set result
y_pred = classifier.predict(X_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Decision Tree', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

results = results.append(model_results, ignore_index = True)
```

In [54]:
```
results
```

Out[54]:

	Model	Accuracy	Precision	Recall	F1 Score

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335
6	SVM (Linear)	0.797965	0.952661	0.811522	0.876446

In [47]:

```
## Random Forest Gini (n=100)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state = 0, n_estimators = 100,
                                    criterion = 'gini')
classifier.fit(X_train, y_train)

# Predicting Test Set
y_pred = classifier.predict(X_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Random Forest Gini (n=100)', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

results = results.append(model_results, ignore_index = True)
```

In [55]:

```
results
```

Out[55]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335
6	SVM (Linear)	0.797965	0.952661	0.811522	0.876446

Applying k-flod validation

In [56]:

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train,cv=10)
accuracies.mean()
accuracies.std()
```

Out[56]: 0.009004867531703015

In [57]:

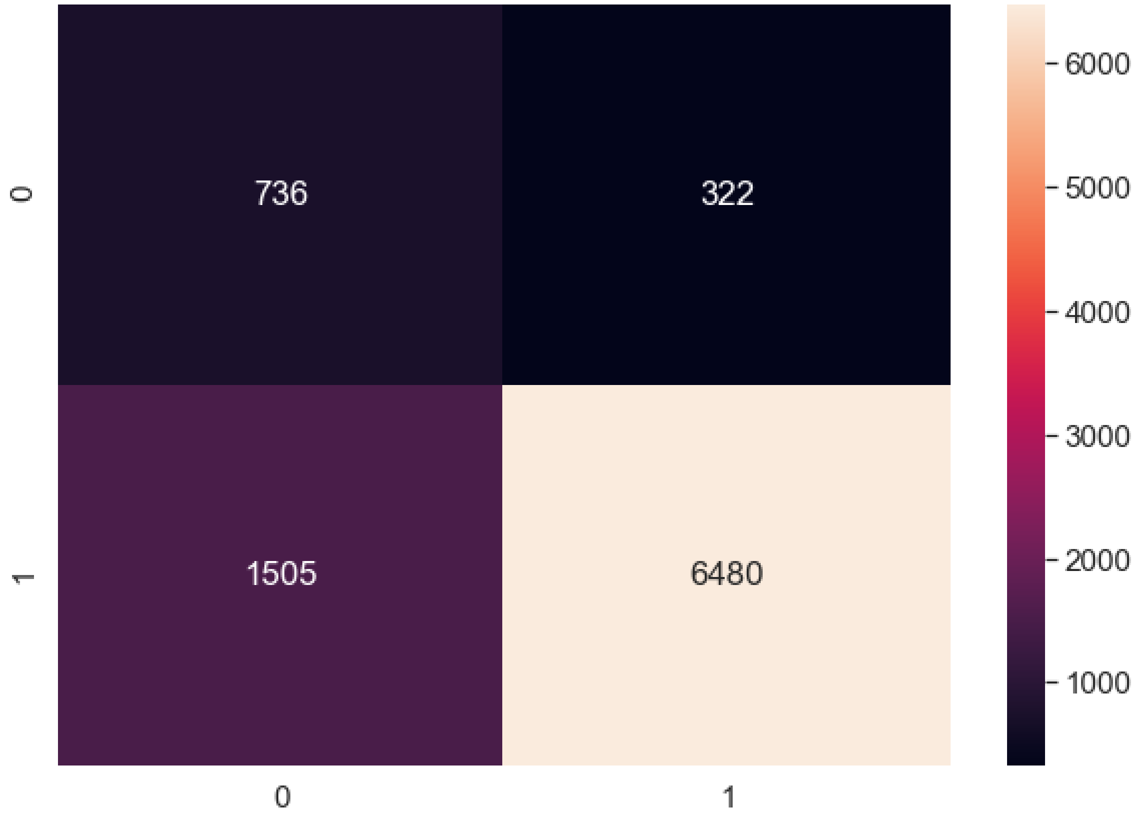
```
print("SVM (Linear) Accuracy: %0.3f (+/- %0.3f)" % (accuracies.mean(), accuracies.st
```

```
SVM (Linear) Accuracy: 0.751 (+/- 0.018)
```

In [58]:

```
#####  Confusion Matrix
cm = confusion_matrix(y_test, y_pred) # rows = truth, cols = prediction
df_cm = pd.DataFrame(cm, index = (0, 1), columns = (0, 1))
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, fmt='g')
print("Test Data Accuracy: %0.4f" % accuracy_score(y_test, y_pred))
```

```
Test Data Accuracy: 0.7980
```



In [59]:

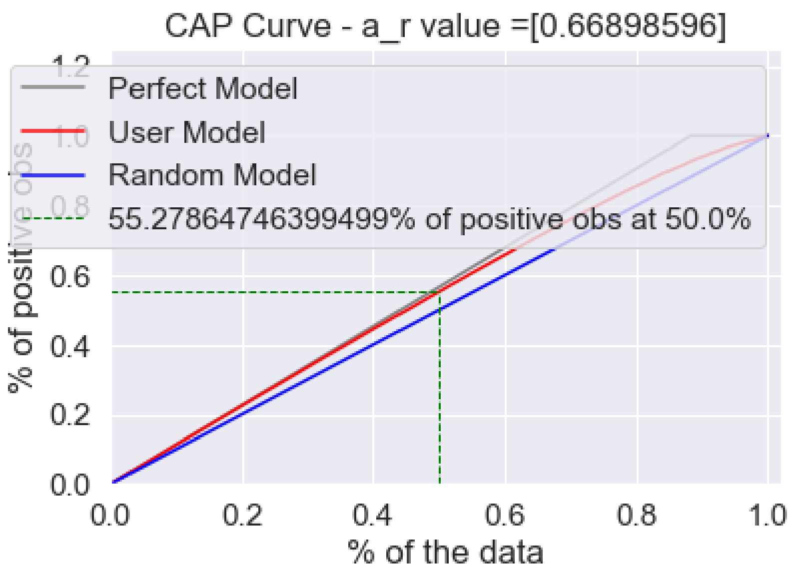
```
#Plotting Cumulative Accuracy Profile (CAP)
y_pred_proba = classifier.predict_proba(X=X_test)
import matplotlib.pyplot as plt
from scipy import integrate
def capcurve(y_values, y_preds_proba):
    num_pos_obs = np.sum(y_values)
    num_count = len(y_values)
    rate_pos_obs = float(num_pos_obs) / float(num_count)
    ideal = pd.DataFrame({'x':[0,rate_pos_obs,1],'y':[0,1,1]})
    xx = np.arange(num_count) / float(num_count - 1)

    y_cap = np.c_[y_values,y_preds_proba]
    y_cap_df_s = pd.DataFrame(data=y_cap)
    y_cap_df_s = y_cap_df_s.sort_values([1], ascending=False).reset_index(level = y_

    print(y_cap_df_s.head(20))

    yy = np.cumsum(y_cap_df_s[0]) / float(num_pos_obs)
    yy = np.append([0], yy[0:num_count-1]) #add the first curve point (0,0) : for xx

    percent = 0.5
    row_index = int(np.trunc(num_count * percent))
```

```
    val_y1 = yy[row_index]
    val_y2 = yy[row_index+1]
    if val_y1 == val_y2:
        val = val_y1*1.0
    else:
        val_x1 = xx[row_index]
        val_x2 = xx[row_index+1]
        val = val_y1 + ((val_x2 - percent)/(val_x2 - val_x1))*(val_y2 - val_y1)

    sigma_ideal = 1 * xx[num_pos_obs - 1 ] / 2 + (xx[num_count - 1] - xx[num_pos_obs
    sigma_model = integrate.simps(yy,xx)
    sigma_random = integrate.simps(xx,xx)

    ar_value = (sigma_model - sigma_random) / (sigma_ideal - sigma_random)

    fig, ax = plt.subplots(nrows = 1, ncols = 1)
    ax.plot(ideal['x'],ideal['y'], color='grey', label='Perfect Model')
    ax.plot(xx,yy, color='red', label='User Model')
    ax.plot(xx,xx, color='blue', label='Random Model')
    ax.plot([percent, percent], [0.0, val], color='green', linestyle='--', linewidth
    ax.plot([0, percent], [val, val], color='green', linestyle='--', linewidth=1, la

    plt.xlim(0, 1.02)
    plt.ylim(0, 1.25)
    plt.title("CAP Curve - a_r value ="+str(ar_value))
    plt.xlabel('% of the data')
    plt.ylabel('% of positive obs')
    plt.legend()
```

In [60]:

```
capcurve(y_test,y_pred_proba[:,1])
```

	0	1
0	1.0	0.999993
1	1.0	0.995807
2	1.0	0.991640
3	1.0	0.990919
4	1.0	0.990914
5	1.0	0.989523
6	1.0	0.987459
7	1.0	0.986814
8	1.0	0.986797
9	1.0	0.986063
10	1.0	0.986025
11	1.0	0.985244
12	1.0	0.985150
13	1.0	0.983236
14	1.0	0.982235
15	1.0	0.981915
16	1.0	0.980047
17	1.0	0.978314
18	1.0	0.978290
19	1.0	0.977016



In [61]:

```
# Analyzing Coefficients
pd.concat([pd.DataFrame(X_train.columns, columns = ["features"]),
           pd.DataFrame(np.transpose(classifier.coef_), columns = ["coef"])
           ],axis = 1)
```

Out[61]:

	features	coef
0	age	-0.066575
1	balance	-0.193843
2	duration	-1.528572
3	campaign	0.204395
4	pdays	-0.157947
5	previous	-0.425870
6	default	0.026428

In [62]:

```
##############    Recursive Feature Elimination
from sklearn.feature_selection import RFE
from sklearn.svm import SVC

# Model to Test
classifier = SVC(random_state = 0, kernel = 'linear', probability= True)

# Select Best X Features
rfe = RFE(classifier, n_features_to_select=None)
rfe = rfe.fit(X_train, y_train)
```

In [63]:

```
# summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
```

```
[False False  True  True False  True False]
[4 2 1 1 3 1 5]
```

In [64]:

```
X_train.columns[rfe.support_]
```

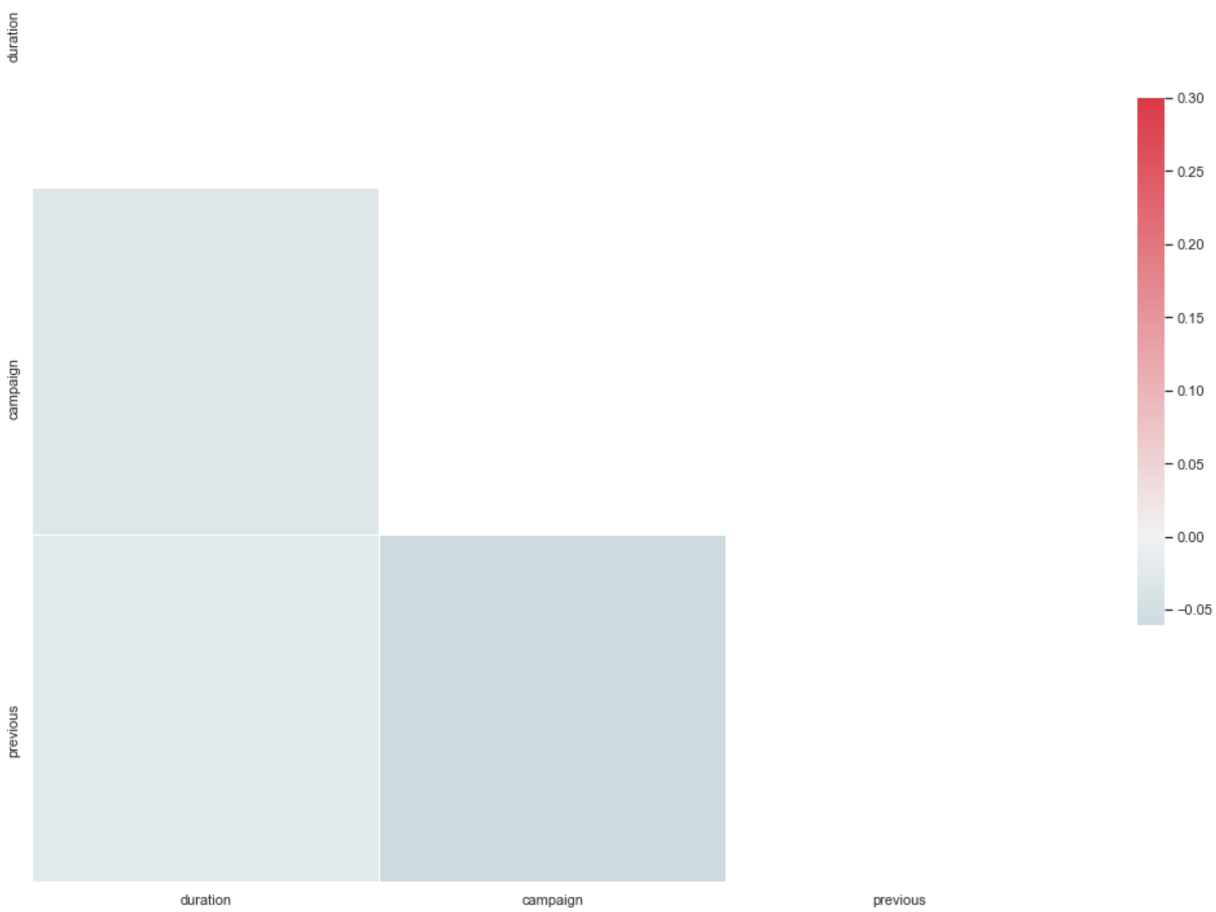
Out[64]:

```
Index(['duration', 'campaign', 'previous'], dtype='object')
```

In [65]:

```
# New Correlation Matrix
sns.set(style="white")

# Compute the correlation matrix
corr = X_train[X_train.columns[rfe.support_]].corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(18, 15))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[65]: <AxesSubplot:>



In [66]:

```
# Fitting Model to the Training Set
classifier = SVC(random_state = 0, kernel = 'linear', probability= True)
classifier.fit(X_train[X_train.columns[rfe.support_]], y_train)

# Predicting Test Set
y_pred = classifier.predict(X_test[X_train.columns[rfe.support_]])
```

```
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['SVM RFE (Linear)', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

results = results.append(model_results, ignore_index = True)
```

In [67]:

```
results
```

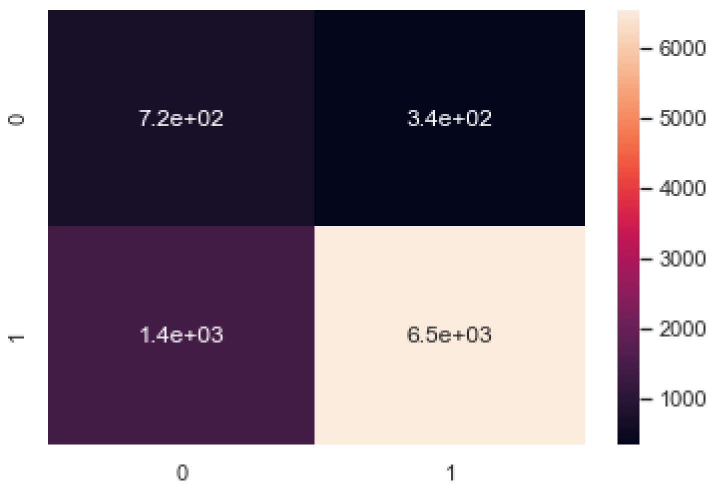
Out[67]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression (Lasso)	0.794095	0.952952	0.806637	0.873711
1	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
2	SVM (RBF)	0.782705	0.966956	0.780589	0.863835
3	Naive Bayes (Gaussian)	0.773195	0.948594	0.785723	0.859511
4	Decision Tree	0.719894	0.954940	0.716594	0.818774
5	Random Forest Gini (n=100)	0.768771	0.969866	0.761803	0.853335
6	SVM (Linear)	0.797965	0.952661	0.811522	0.876446
7	SVM RFE (Linear)	0.803937	0.951060	0.820163	0.880775

In [68]:

```
# Evaluating Results
#Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
sns.heatmap(data=cm, annot=True)
```

Out[68]: <AxesSubplot:>



In [69]:

```
#Making the classification report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.33      0.68      0.45      1058
           1       0.95      0.82      0.88      7985
```

```
    accuracy                           0.80      9043
   macro avg       0.64      0.75      0.66      9043
weighted avg       0.88      0.80      0.83      9043
```

In [70]:

```
# Applying k-Fold Cross Validation (RFE)
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier,
                             X = X_train[X_train.columns[rfe.support_]],
                             y = y_train, cv = 10)
```

In [72]:

```
print("SVM RFE (Linear) Accuracy: %0.3f (+/- %0.3f)" % (accuracies.mean(), accuracie
```

```
SVM RFE (Linear) Accuracy: 0.747 (+/- 0.020)
```

In [73]:

```
# Analyzing Coefficients
pd.concat([pd.DataFrame(X_train[X_train.columns[rfe.support_]].columns, columns = ["
           pd.DataFrame(np.transpose(classifier.coef_), columns = ["coef"])
           ],axis = 1)
```

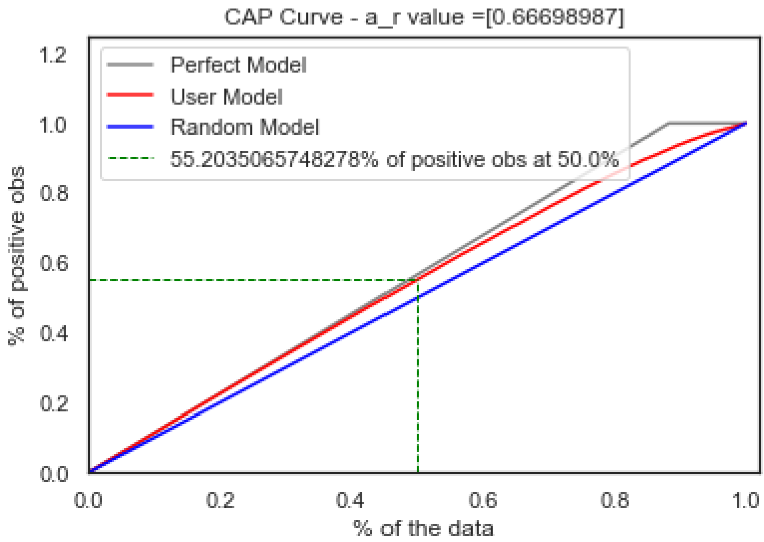
Out[73]:

	features	coef
0	duration	-1.539859
1	campaign	0.214335
2	previous	-0.574785

In [74]:

```
#CAP Curve
y_pred_proba = classifier.predict_proba(X=X_test[X_train.columns[rfe.support_]])
capcurve(y_test,y_pred_proba[:,1])
```

```
      0         1
0   1.0  0.999990
1   1.0  0.995396
2   1.0  0.990650
3   1.0  0.990570
4   1.0  0.990336
5   1.0  0.987674
6   1.0  0.986908
7   1.0  0.986697
8   1.0  0.986470
9   1.0  0.984798
10  1.0  0.984451
11  1.0  0.983978
12  1.0  0.983303
13  1.0  0.983208
14  1.0  0.981000
15  1.0  0.979525
16  1.0  0.977628
17  1.0  0.976670
18  1.0  0.974681
19  1.0  0.972450
```



In [75]:
```
### End of the Model

# Formatting Final Results
user_identifier = df['user']
final_results = pd.concat([y_test, user_identifier], axis = 1).dropna()
final_results['predicted'] = y_pred
final_results = final_results[['user', 'y', 'predicted']].reset_index(drop=True)
```

In [76]:
```
final_results.head()
```

Out[76]:

	user	y	predicted
0	5	1.0	1
1	11	1.0	1
2	20	1.0	1
3	21	1.0	1
4	26	1.0	1

In [ ]:
```
```