Predicting Engagement - What drives ad performance?

In [0]:

```
import matplotlib.pyplot as plt
import pandas as pd
# Uncomment this line if using this notebook locally
#bank = pd.read_csv('./data/bank/bank-full.csv', sep=';')

file_name = "https://raw.githubusercontent.com/rajeevratan84/datascienceforbusiness/master/bank-full.csv"
bank = pd.read_csv(file_name, sep=';')

bank.head()
```

Out[0]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

In [0]:

```
# Let's see a summary of our dataframe
print ("Rows     : " , bank.shape[0])
print ("Columns  : " , bank.shape[1])
print ("\nFeatures : \n" ,bank.columns.tolist())
print ("\nMissing values :  ", bank.isnull().sum().values.sum())
print ("\nUnique values :  \n", bank.nunique())
```

```
Rows     :  45211
Columns  :  17

Features :
 ['age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
'previous', 'poutcome', 'y']

Missing values :   0

Unique values :
 age          77
job          12
marital       3
education     4
```

```
default         2
balance      7168
housing         2
loan            2
contact         3
day            31
month          12
duration     1573
campaign       48
pdays         559
previous       41
poutcome        4
y               2
dtype: int64
```

In [0]:

```
bank.describe()
```

Out[0]:

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

In [0]:

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
age          45211 non-null int64
job          45211 non-null object
marital      45211 non-null object
education    45211 non-null object
default      45211 non-null object
balance      45211 non-null int64
housing      45211 non-null object
loan         45211 non-null object
contact      45211 non-null object
day          45211 non-null int64
```

```
month        45211 non-null object
duration     45211 non-null int64
campaign     45211 non-null int64
pdays        45211 non-null int64
previous     45211 non-null int64
poutcome     45211 non-null object
y            45211 non-null object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [0]:

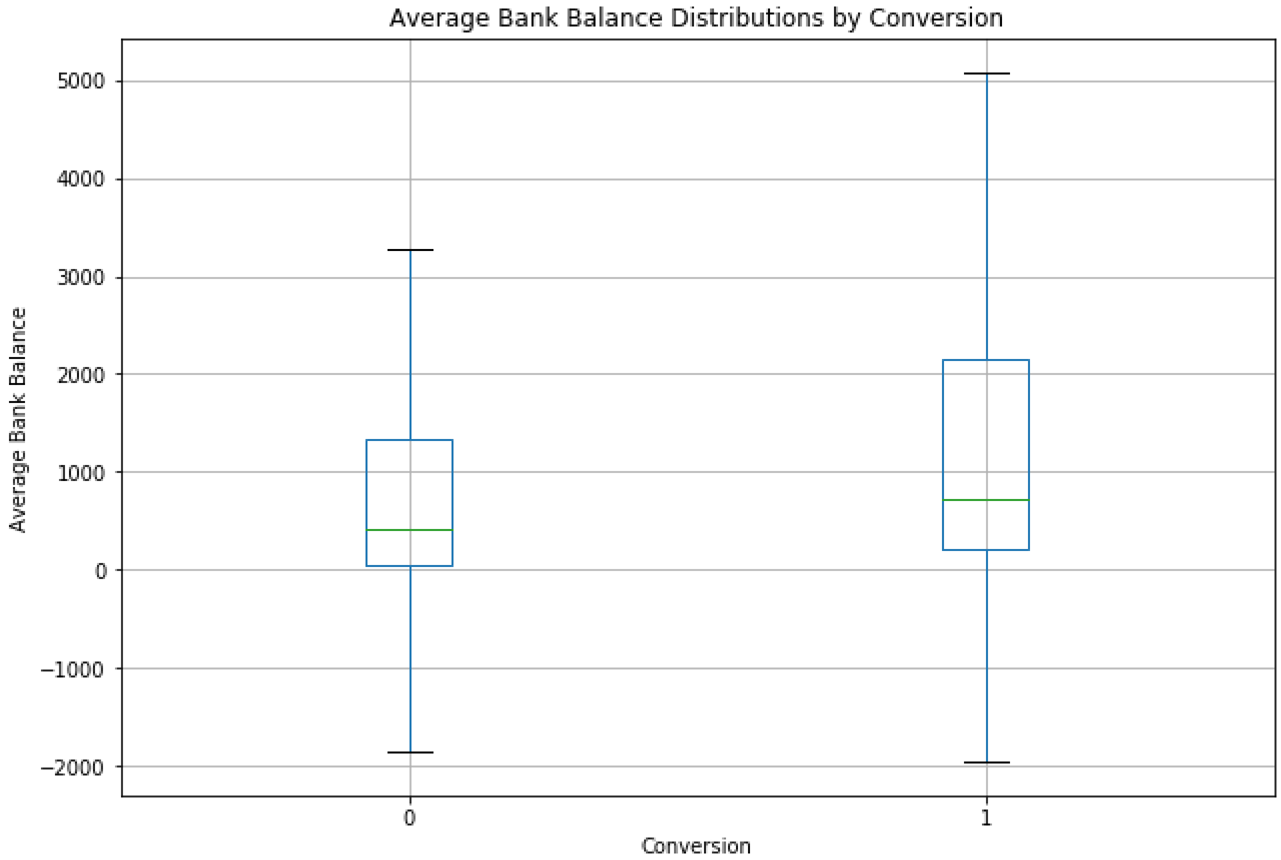
```python
# Here we use the apply funtion to transform 'y' from yes or no to 0s and 1s
bank['converted'] = bank['y'].apply(lambda x: 0 if x == 'no' else 1)
del bank['y']
bank.head()
```

Out[0]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	0
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	0
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	0
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	0
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	0

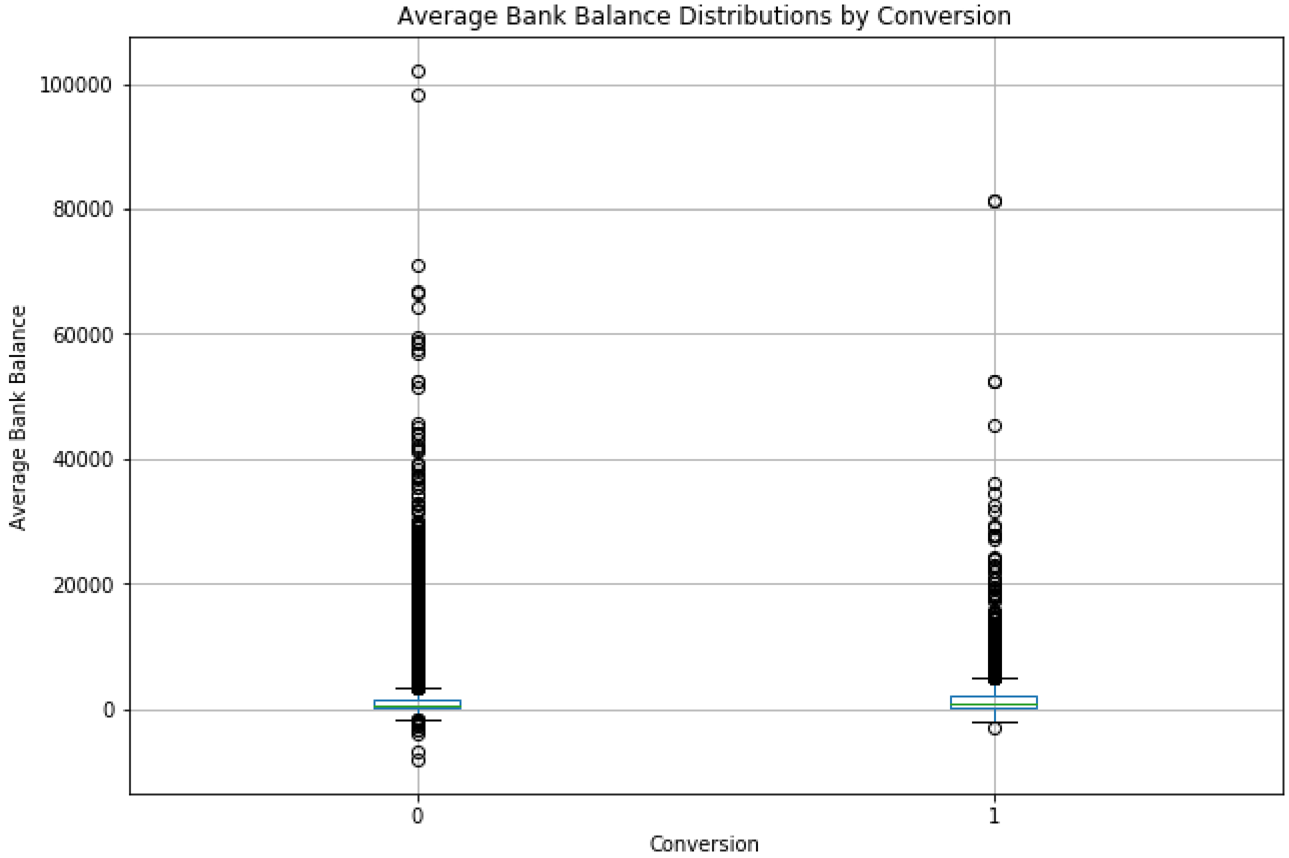
In [0]:

```python
# Let's Visualize how our output variable (converted) changes with different incomes
ax = bank[['converted', 'balance']].boxplot(by='converted', showfliers=False, figsize=(10, 7))

ax.set_xlabel('Conversion')
ax.set_ylabel('Average Bank Balance')
ax.set_title('Average Bank Balance Distributions by Conversion')

plt.suptitle("")
plt.show()
```



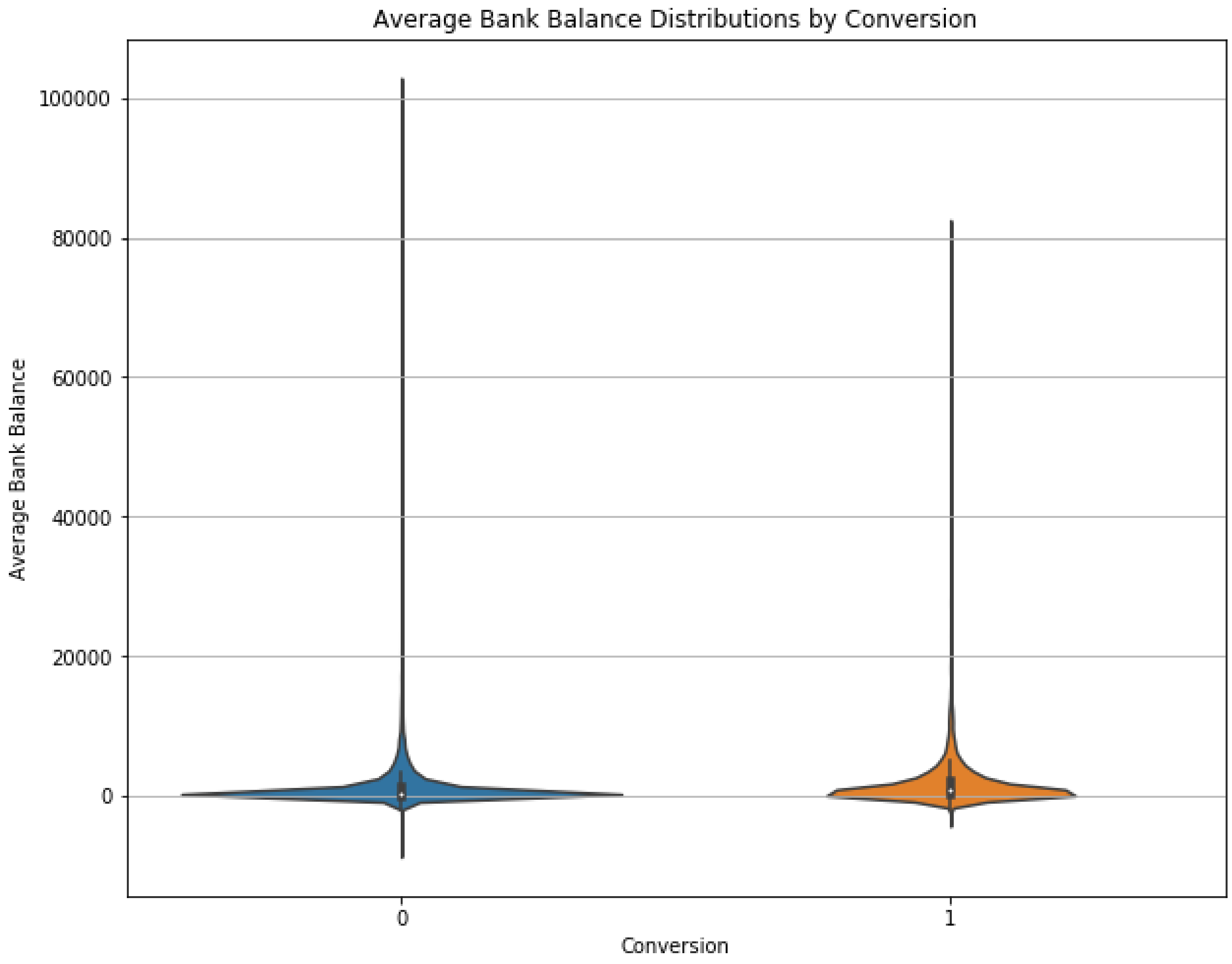
In [0]:

```
# Let's Visualize how our output variable (converted) changes with different incomes
ax = bank[['converted', 'balance']].boxplot(by='converted', showfliers=True, figsize=(10, 7))

ax.set_xlabel('Conversion')
ax.set_ylabel('Average Bank Balance')
ax.set_title('Average Bank Balance Distributions by Conversion')

plt.suptitle("")
plt.show()
```



In [0]:

```
# Let's do the same withing using Violin plots
import seaborn as sns

fontsize = 10

fig, axes = plt.subplots()
# plot violin. 'Scenario' is according to x axis,
# 'LMP' is y axis, data is your dataframe. ax - is axes instance


fig.set_size_inches(10, 8)

sns.violinplot('converted','balance', data=bank, ax = axes)
axes.set_title('Average Bank Balance Distributions by Conversion')

axes.yaxis.grid(True)
axes.set_xlabel('Conversion')
axes.set_ylabel('Average Bank Balance')

plt.show()
```



In [0]:

```
bank['campaign'].unique()
```

Out[0]:

```
array([ 1,  2,  3,  5,  4,  6,  7,  8,  9, 10, 11, 12, 13, 19, 14, 24, 16,
       32, 18, 22, 15, 17, 25, 21, 43, 51, 63, 41, 26, 28, 55, 50, 38, 23,
       20, 29, 31, 37, 30, 46, 27, 58, 33, 35, 34, 36, 39, 44])
```

In [0]:

```
# Conversion rate by campaign
conversions_by_contacts = bank.groupby('campaign')['converted'].sum() / bank.groupby('campaign')['converted'].count() * 100.0
# Let's see the top ten campaigns in terms of % converted
conversions_by_contacts.head(10)
```

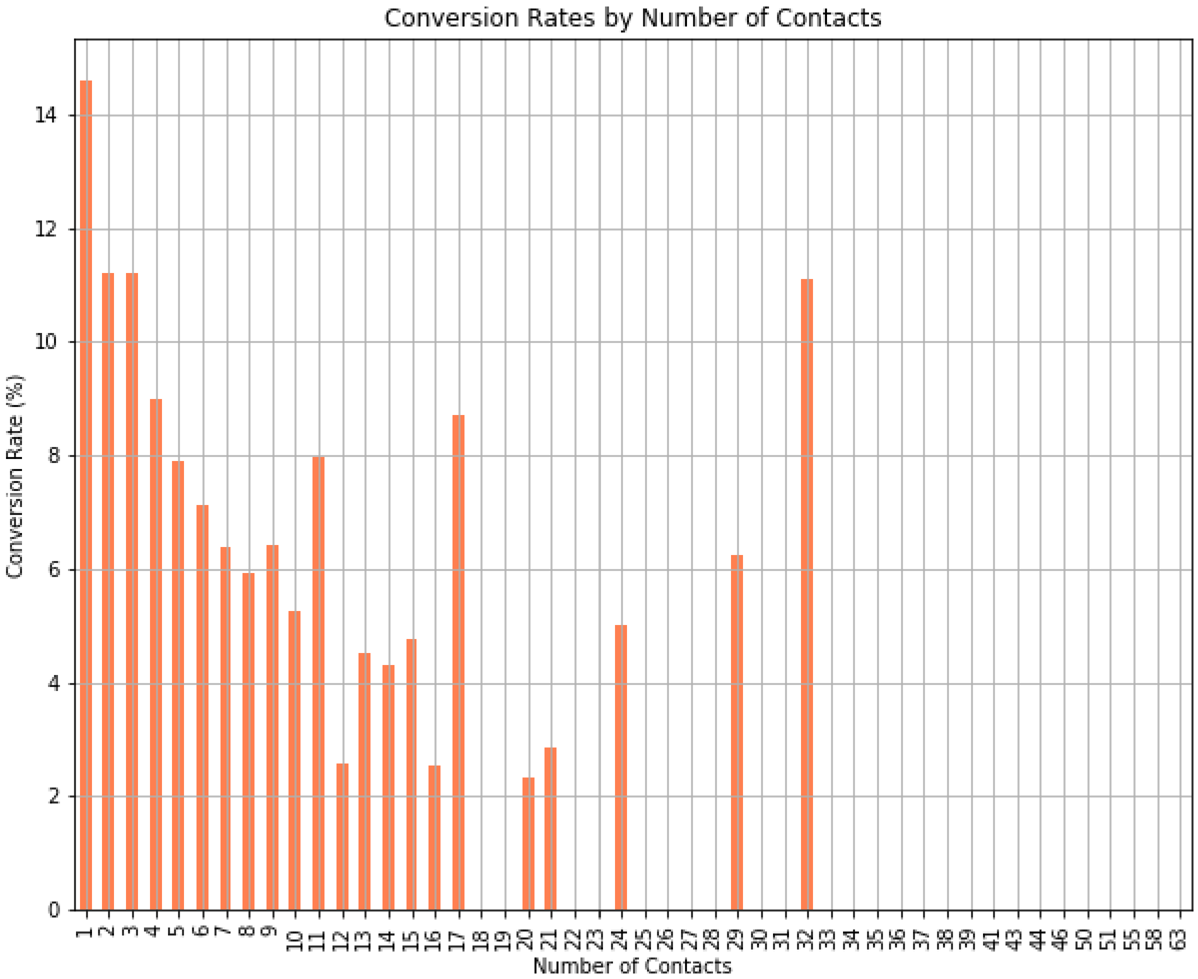
Out[0]:

```
campaign
1      14.597583
2      11.203519
```

```
3      11.193624
4       9.000568
5       7.879819
6       7.126259
7       6.394558
8       5.925926
9       6.422018
10      5.263158
Name: converted, dtype: float64
```

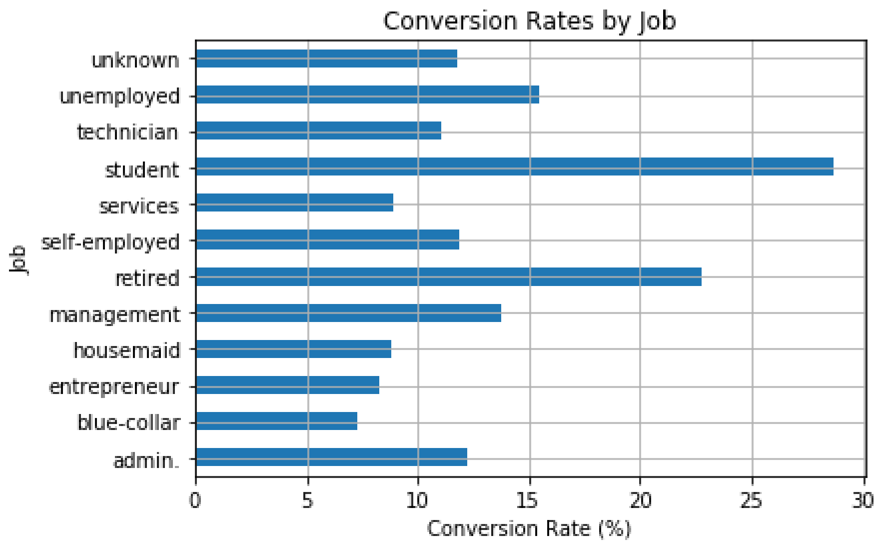
In [0]:

```python
ax = conversions_by_contacts.plot(
    kind='bar',
    figsize=(10, 8),
    title='Conversion Rates by Number of Contacts',
    grid=True,
    color='coral'
)

ax.set_xlabel('Number of Contacts')
ax.set_ylabel('Conversion Rate (%)')

plt.show()
```



In [0]:

```python
# How about conversion rate by job?
conversion_rate_by_job = bank.groupby(by='job')['converted'].sum() / bank.groupby(by='job')['converted'].count() * 100.0
ax = conversion_rate_by_job.plot(kind='barh', grid=True, title='Conversion Rates by Job')

ax.set_xlabel('Conversion Rate (%)')
ax.set_ylabel('Job')

plt.show()
```



In [0]:

```python
# View the number of unique elements in each feature
bank.nunique()
```

Out[0]:

```
age            77
job            12
marital         3
education       4
default         2
balance      7168
housing         2
loan            2
contact         3
day            31
month          12
duration     1573
campaign       48
pdays         559
previous       41
poutcome        4
converted       2
dtype: int64
```

In [0]:

```python
# Get our category type columns
cols = bank.columns
num_cols = bank._get_numeric_data().columns
cat_cols = list(set(cols) - set(num_cols))
cat_cols
```

Out[0]:

```
['y',
 'marital',
```

```
 'loan',
 'month',
 'contact',
 'housing',
 'education',
 'poutcome',
 'default',
 'job']
```

We need to encode our cateogorical varaibles

marital', 'loan', 'month', 'contact', 'housing', 'education', 'poutcome', 'default', 'job'

In [0]:

```
# Starting with month first
bank['month'].unique()
```

Out[0]:

```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb',
       'mar', 'apr', 'sep'], dtype=object)
```

In [0]:

```
bank.groupby('month').count()['converted']
```

Out[0]:

```
month
apr     2932
aug     6247
dec      214
feb     2649
jan     1403
jul     6895
jun     5341
mar      477
may    13766
nov     3970
oct      738
sep      579
Name: converted, dtype: int64
```

In [0]:

```
months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']

bank['month'] = bank['month'].apply(lambda x: months.index(x)+1)
bank.head()
```

Out[0]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	5	261	1	-1	0	unknown	0

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted
1	44	technician	single	secondary	no	29	yes	no	unknown	5	5	151	1	-1	0	unknown	0
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	5	76	1	-1	0	unknown	0
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	5	92	1	-1	0	unknown	0
4	33	unknown	single	unknown	no	1	no	no	unknown	5	5	198	1	-1	0	unknown	0

Let's encode jobs & marital

In [0]:

```
bank['job'].unique()
```

Out[0]:

```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

In [0]:

```
bank = pd.get_dummies(data=bank, columns=['job'])
bank.head()
```

Out[0]:

	age	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blu col
0	58	married	tertiary	no	2143	yes	no	unknown	5	5	261	1	-1	0	unknown	0	0	
1	44	single	secondary	no	29	yes	no	unknown	5	5	151	1	-1	0	unknown	0	0	
2	33	married	secondary	no	2	yes	yes	unknown	5	5	76	1	-1	0	unknown	0	0	
3	47	married	unknown	no	1506	yes	no	unknown	5	5	92	1	-1	0	unknown	0	0	
4	33	single	unknown	no	1	no	no	unknown	5	5	198	1	-1	0	unknown	0	0	

In [0]:

```
bank = pd.get_dummies(data=bank, columns=['marital'])
bank.head()
```

Out[0]:

	age	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_e
0	58	tertiary	no	2143	yes	no	unknown	5	5	261	1	-1	0	unknown	0	0	0	
1	44	secondary	no	29	yes	no	unknown	5	5	151	1	-1	0	unknown	0	0	0	

	age	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_e
2	33	secondary	no	2	yes	yes	unknown	5	5	76	1	-1	0	unknown	0	0	0	
3	47	unknown	no	1506	yes	no	unknown	5	5	92	1	-1	0	unknown	0	0	1	
4	33	unknown	no	1	no	no	unknown	5	5	198	1	-1	0	unknown	0	0	0	

Encoding Housing

In [0]:

```
bank['housing'].unique()
```

Out[0]:

```
array(['yes', 'no'], dtype=object)
```

In [0]:

```
bank['housing'] = bank['housing'].map(lambda s :1  if s == 'yes' else 0)
bank.head()
```

Out[0]:

	age	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_e
0	58	tertiary	no	2143	1	no	unknown	5	5	261	1	-1	0	unknown	0	0	0	
1	44	secondary	no	29	1	no	unknown	5	5	151	1	-1	0	unknown	0	0	0	
2	33	secondary	no	2	1	yes	unknown	5	5	76	1	-1	0	unknown	0	0	0	
3	47	unknown	no	1506	1	no	unknown	5	5	92	1	-1	0	unknown	0	0	1	
4	33	unknown	no	1	0	no	unknown	5	5	198	1	-1	0	unknown	0	0	0	

Encoding loans

In [0]:

```
bank['loan'].unique()
```

Out[0]:

```
array(['no', 'yes'], dtype=object)
```

In [0]:

```
bank['loan'] = bank['loan'].map(lambda s :1  if s == 'yes' else 0)
bank.head()
```

Out[0]:

	age	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_e
0	58	tertiary	no	2143	1	0	unknown	5	5	261	1	-1	0	unknown	0	0	0	
1	44	secondary	no	29	1	0	unknown	5	5	151	1	-1	0	unknown	0	0	0	
2	33	secondary	no	2	1	1	unknown	5	5	76	1	-1	0	unknown	0	0	0	
3	47	unknown	no	1506	1	0	unknown	5	5	92	1	-1	0	unknown	0	0	1	
4	33	unknown	no	1	0	0	unknown	5	5	198	1	-1	0	unknown	0	0	0	

In [0]:

```
bank.columns
```

Out[0]:

```
Index(['age', 'education', 'default', 'balance', 'housing', 'loan', 'contact',
       'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome',
       'converted', 'job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired', 'job_self-employed',
       'job_services', 'job_student', 'job_technician', 'job_unemployed',
       'job_unknown', 'marital_divorced', 'marital_married', 'marital_single'],
      dtype='object')
```

In [0]:

```
bank.head()
```

Out[0]:

	age	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_e
0	58	tertiary	no	2143	1	0	unknown	5	5	261	1	-1	0	unknown	0	0	0	
1	44	secondary	no	29	1	0	unknown	5	5	151	1	-1	0	unknown	0	0	0	
2	33	secondary	no	2	1	1	unknown	5	5	76	1	-1	0	unknown	0	0	0	
3	47	unknown	no	1506	1	0	unknown	5	5	92	1	-1	0	unknown	0	0	1	
4	33	unknown	no	1	0	0	unknown	5	5	198	1	-1	0	unknown	0	0	0	

In [0]:

```
bank['education'].unique()
```

Out[0]:

```
array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)
```

In [0]:

```python
bank = pd.get_dummies(data=bank, columns=['education'])
bank.head()
```

Out[0]:

	age	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_entrepreneu
0	58	no	2143	1	0	unknown	5	5	261	1	-1	0	unknown	0	0	0	0
1	44	no	29	1	0	unknown	5	5	151	1	-1	0	unknown	0	0	0	0
2	33	no	2	1	1	unknown	5	5	76	1	-1	0	unknown	0	0	0	1
3	47	no	1506	1	0	unknown	5	5	92	1	-1	0	unknown	0	0	1	0
4	33	no	1	0	0	unknown	5	5	198	1	-1	0	unknown	0	0	0	0

In [0]:

```python
#default, , contact
bank['contact'].unique()
```

Out[0]: array(['unknown', 'cellular', 'telephone'], dtype=object)

In [0]:

```python
bank = pd.get_dummies(data=bank, columns=['contact'])
bank.head()
```

Out[0]:

	age	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_entrepreneur	job_hou
0	58	no	2143	1	0	5	5	261	1	-1	0	unknown	0	0	0	0	
1	44	no	29	1	0	5	5	151	1	-1	0	unknown	0	0	0	0	
2	33	no	2	1	1	5	5	76	1	-1	0	unknown	0	0	0	1	
3	47	no	1506	1	0	5	5	92	1	-1	0	unknown	0	0	1	0	
4	33	no	1	0	0	5	5	198	1	-1	0	unknown	0	0	0	0	

In [0]:

```python
bank['default'].unique()
```

Out[0]: array(['no', 'yes'], dtype=object)

In [0]:

```python
bank = pd.get_dummies(data=bank, columns=['default'])
bank.head()
```

Out[0]:

	age	balance	housing	loan	day	month	duration	campaign	pdays	previous	poutcome	converted	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	j
0	58	2143	1	0	5	5	261	1	-1	0	unknown	0	0	0	0	0	
1	44	29	1	0	5	5	151	1	-1	0	unknown	0	0	0	0	0	
2	33	2	1	1	5	5	76	1	-1	0	unknown	0	0	0	1	0	
3	47	1506	1	0	5	5	92	1	-1	0	unknown	0	0	1	0	0	
4	33	1	0	0	5	5	198	1	-1	0	unknown	0	0	0	0	0	

In [0]:

```python
bank['poutcome'].unique()
```

Out[0]:

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

In [0]:

```python
bank = pd.get_dummies(data=bank, columns=['poutcome'])
bank.head()
```

Out[0]:

	age	balance	housing	loan	day	month	duration	campaign	pdays	previous	converted	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_manager
0	58	2143	1	0	5	5	261	1	-1	0	0	0	0	0	0	
1	44	29	1	0	5	5	151	1	-1	0	0	0	0	0	0	
2	33	2	1	1	5	5	76	1	-1	0	0	0	0	1	0	
3	47	1506	1	0	5	5	92	1	-1	0	0	0	1	0	0	
4	33	1	0	0	5	5	198	1	-1	0	0	0	0	0	0	

In [0]:

```python
# What categoric columns are left?
cols = bank.columns
num_cols = bank._get_numeric_data().columns
cat_cols = list(set(cols) - set(num_cols))
cat_cols
```

Out[0]: []

In [0]:

```
Y_train = bank['converted']
X_train = bank.drop(labels = ["converted"], axis = 1)
X_train
```

Out[0]:

	age	balance	housing	loan	day	month	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	j
0	58	2143	1	0	5	5	261	1	-1	0	0	0	0	0	1	
1	44	29	1	0	5	5	151	1	-1	0	0	0	0	0	0	
2	33	2	1	1	5	5	76	1	-1	0	0	0	1	0	0	
3	47	1506	1	0	5	5	92	1	-1	0	0	1	0	0	0	
4	33	1	0	0	5	5	198	1	-1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
45206	51	825	0	0	17	11	977	3	-1	0	0	0	0	0	0	
45207	71	1729	0	0	17	11	456	2	-1	0	0	0	0	0	0	
45208	72	5715	0	0	17	11	1127	5	184	3	0	0	0	0	0	
45209	57	668	0	0	17	11	508	4	-1	0	0	1	0	0	0	
45210	37	2971	0	0	17	11	361	2	188	11	0	0	1	0	0	

45211 rows × 38 columns

Now let's Fit Our Decision Tree Model

In [0]:

```
from sklearn import tree

dec_tree_model = tree.DecisionTreeClassifier(max_depth=5)
```

In [0]:

```
dec_tree_model.fit(X_train, Y_train)
```

Out[0]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
```

```
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, presort=False,
                             random_state=None, splitter='best')
```

In [0]:

```
# Install graphviz if you need to install the module
#!pip install graphviz
```

In [0]:

```
features = list(X_train.columns)
response_var = 'converted'
```

In [0]:

```
features
```

Out[0]:

```
['age',
 'balance',
 'housing',
 'loan',
 'day',
 'month',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'job_admin.',
 'job_blue-collar',
 'job_entrepreneur',
 'job_housemaid',
 'job_management',
 'job_retired',
 'job_self-employed',
 'job_services',
 'job_student',
 'job_technician',
 'job_unemployed',
 'job_unknown',
 'marital_divorced',
 'marital_married',
 'marital_single',
 'education_primary',
 'education_secondary',
 'education_tertiary',
 'education_unknown',
 'contact_cellular',
 'contact_telephone',
 'contact_unknown',
 'default_no',
 'default_yes',
 'poutcome_failure',
 'poutcome_other',
```

```
 'poutcome_success',
 'poutcome_unknown']
```

Generate and Visualize Our Decision Tree

In [0]:

```python
import graphviz

# We export our tree to a DOT format is a graphic description language
dot_data = tree.export_graphviz(dec_tree_model, feature_names=features, class_names=['0', '1'],
                                filled=True, rounded=True, special_characters=True)

# Create a visual graph of our tree
graph = graphviz.Source(dot_data)
```

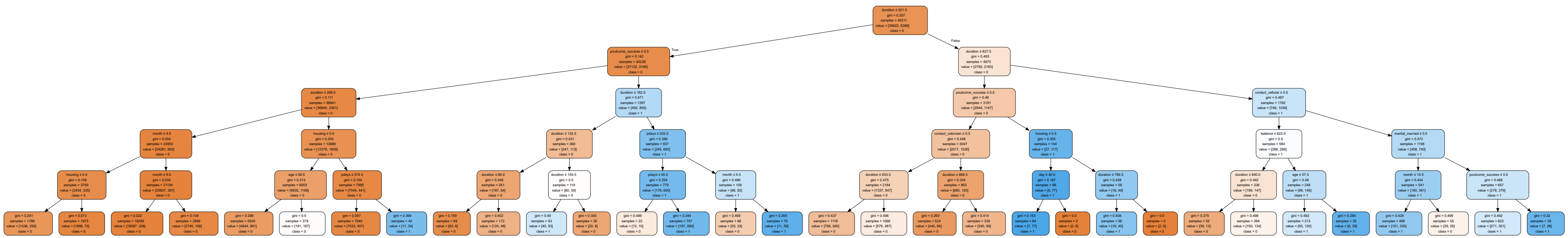
Understanding our Tree

- The first line contains split threshold
- The second line is the Gini impurity which is the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution in the dataset
- The third line gives us the total number of records that belong to that node
- The fourth line in each node gives us the composition of the records in two different classes.
- The fifth line is the class prediction (only use as a predictor when looking at the bottom nodes or root nodes)

In [0]:

```python
# Display our tree below
from IPython.core.display import display, HTML
display(HTML("<style>text {font-size: 10px;}</style>"))

graph
```

Out[0]:



In [0]:

```python
from sklearn.base import clone

def imp_df(column_names, importances):
  df = pd.DataFrame({'feature': column_names,
                     'feature_importance': importances}).sort_values('feature_importance', ascending = False).reset_index(drop = True)
  return df

def drop_col_feat_imp(model, X_train, y_train, random_state = 42):

    # clone the model to have the exact same specification as the one initially trained
    model_clone = clone(model)
    # set random_state for comparability
    model_clone.random_state = random_state
    # training and scoring the benchmark model
    model_clone.fit(X_train, y_train)
    benchmark_score = model_clone.score(X_train, y_train)
    # list for storing feature importances
    importances = []

    # iterating over all columns and storing feature importance (difference between benchmark and new model)
    for col in X_train.columns:
        model_clone = clone(model)
        model_clone.random_state = random_state
        model_clone.fit(X_train.drop(col, axis = 1), y_train)
        drop_col_score = model_clone.score(X_train.drop(col, axis = 1), y_train)
        importances.append(benchmark_score - drop_col_score)

    importances_df = imp_df(X_train.columns, importances)
    return importances_df
```

In [0]:

```python
drop_col_feat_imp(dec_tree_model, X_train, Y_train)
```

Out[0]:

	feature	feature_importance
0	duration	0.009002
1	poutcome_success	0.004711
2	pdays	0.000929
3	month	0.000177
4	marital_married	0.000066
5	day	0.000022
6	job_technician	0.000000
7	education_tertiary	0.000000

	feature	feature_importance
8	marital_divorced	0.000000
9	marital_single	0.000000
10	education_primary	0.000000
11	education_secondary	0.000000
12	default_no	0.000000
13	education_unknown	0.000000
14	contact_telephone	0.000000
15	job_unemployed	0.000000
16	default_yes	0.000000
17	poutcome_failure	0.000000
18	poutcome_other	0.000000
19	job_unknown	0.000000
20	poutcome_unknown	0.000000
21	loan	0.000000
22	job_services	0.000000
23	job_self-employed	0.000000
24	job_retired	0.000000
25	job_management	0.000000
26	job_housemaid	0.000000
27	job_entrepreneur	0.000000
28	job_student	0.000000
29	job_blue-collar	0.000000
30	job_admin.	0.000000
31	previous	0.000000
32	campaign	0.000000
33	balance	-0.000088
34	housing	-0.000088

	feature	feature_importance
35	age	-0.000155
36	contact_cellular	-0.000177
37	contact_unknown	-0.000221

In [0]:

```
```