Back to the Roots!

This was the second kernel I published in my Kaggle journey, and I would like to give it some changes in order to make it more attractive to the Kaggle community. Please upvote if you enjoy this kernel. I will be making updates to this kernel whenever I get some free time from school, hope you like it. Let's Begin!

Bank Marketing DataSet - Intelligent Targeting:

Marketing Introduction:

The process by which companies create value for customers and build strong customer relationships in order to capture value from customers in return.

Kotler and Armstrong (2010).

Marketing campaigns are characterized by focusing on the customer needs and their overall satisfaction. Nevertheless, there are different variables that determine whether a marketing campaign will be successful or not. There are certain variables that we need to take into consideration when making a marketing campaign.

The 4 Ps:

1) Segment of the **Population:** To which segment of the population is the marketing campaign going to address and why? This aspect of the marketing campaign is extremely important since it will tell to which part of the population should most likely receive the message of the marketing campaign.

2) Distribution channel to reach the customer's **place**: Implementing the most effective strategy in order to get the most out of this marketing campaign. What segment of the population should we address? Which instrument should we use to get our message out? (Ex: Telephones, Radio, TV, Social Media Etc.)

3) **Price:** What is the best price to offer to potential clients? (In the case of the bank's marketing campaign this is not necessary since the main interest for the bank is for potential clients to open depost accounts in order to make the operative activities of the bank to keep on running.)

4) **Promotional** Strategy: This is the way the strategy is going to be implemented and how are potential clients going to be address. This should be the last part of the marketing campaign analysis since there has to be an indepth analysis of previous campaigns (If possible) in order to learn from previous mistakes and to determine how to make the marketing campaign much more effective.

Regarding this Kernel:

I know this is a well known dataset since it comes from **UCI Machine Learning Repository**. However, I believe there are some interesting insights you could see that you could integrate to your own data analysis. All in all, Kaggle is meant to learn from others and I hope this example suits you well.

Please feel free to use this kernel to your projects it will be my pleasure!

Also, I'm open to new ideas and things that I could improve to make this kernel even better! Open to constructie criticisms! Lastly, I will like to give a special thanks to **Randy Lao** and his well-known **Predicting Employee Kernelover**. His kernel gave me different ideas as to how should I approach an analysis of a dataset.

Also, I want to give credit to this stackoverflow post, which helped me change the name of legends from Facetgrids.
https://stackoverflow.com/questions/45201514/edit-seaborn-plot-figure-legend (https://stackoverflow.com/questions/45201514/edit-seaborn-plot-figure-legend)
Check it out if you are struggling with the same problem.

New Updates:

- Determine clusters among the sample population that will most likely open term deposit accounts.

What is a Term Deposit?

A **Term deposit** is a deposit that a bank or a financial institurion offers with a fixed rate (often better than just opening deposit account) in which your money will be returned back at a specific maturity time. For more information with regards to Term Deposits please click on this link from Investopedia: https://www.investopedia.com/terms/t/termdeposit.asp (https://www.investopedia.com/terms/t/termdeposit.asp)

Outline:

A. **Attribute Descriptions**
I. *Bank client data (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#bank_client_data)*
II. Related with the last contact of the current campaign (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#last_contact)
III. Other attributes (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#other_attributes)

B. **Structuring the data:**
I. *Overall Analysis of the Data (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#overall_analysis)*
II. Data Structuring and Conversions (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#data_structuring)

C. **Exploratory Data Analysis (EDA)**
I. *Accepted vs Rejected Term Deposits (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#accepted_rejected)*
II. Distribution Plots (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#distribution_plots)

D. **Different Aspects of the Analysis:**
I. *Months of Marketing Activty (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#months_activity)*
II. Seasonalities (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#seasonality)
III. *Number of Calls to the potential client (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#number_calls)*
IV. Age of the Potential Clients (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#age_clients)
V. Types of Occupations that leads to more term deposits suscriptions (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#occupations)

E. **Correlations that impacted the decision of Potential Clients.** I. *Analysis of our Correlation Matrix (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#analysis_correlation)*
II. Balance Categories vs Housing Loans

II. Balance Categories vs Housing Loans (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#balance_housing)
III. Negative Relationship between H.Loans and Term Deposits (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#negative_relationship)

F. **Classification Model**
I. Introduction (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#classification_model)
II. Stratified Sampling (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#stratified)
III. Classification Models (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#models)
IV. Confusion Matrix (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#confusion)
V. Precision and Recall Curve (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#precision_recall)
VI. Feature Importances Decision Tree C. (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#decision)

G. **Next Campaign Strategy**
I. Actions the Bank should Consider (https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/comments#bank_actions)

A. Attributes Description:

Input variables:

Ai. bank client data:

1 - **age:** (numeric)
2 - **job:** type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - **marital:** marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - **education:** (categorical: primary, secondary, tertiary and unknown)
5 - **default:** has credit in default? (categorical: 'no','yes','unknown')
6 - **housing:** has housing loan? (categorical: 'no','yes','unknown')
7 - **loan:** has personal loan? (categorical: 'no','yes','unknown')
8 - **balance:** Balance of the individual.

Aii. Related with the last contact of the current campaign:

8 - **contact:** contact communication type (categorical: 'cellular','telephone')
9 - **month:** last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

dec')
10 - **day:** last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - **duration:** last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Aiii. other attributes:

12 - **campaign:** number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - **pdays:** number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - **previous:** number of contacts performed before this campaign and for this client (numeric)
15 - **poutcome:** outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

Output variable (desired target):
21 - **y** - has the client subscribed a term deposit? (binary: 'yes','no')

In [1]:

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

from plotly import tools
import plotly.plotly as py
import plotly.figure_factory as ff
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_not
ebook_mode, plot, iplot
init_notebook_mode(connected=True)

MAIN_PATH = '../input/'
df = pd.read_csv(MAIN_PATH +'bank.csv')
term_deposits = df.copy()
# Have a grasp of how our data looks.
df.head()
```

Out[1]:

	age	job	marital	education	default	balance
0	59	admin.	married	secondary	no	2343
1	56	admin.	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	admin.	married	tertiary	no	184

Exploring the Basics

Summary:

- **Mean Age** is aproximately 41 years old. (Minimum: 18 years old and Maximum: 95 years old.)
- The **mean balance** is 1,528. However, the Standard Deviation (std) is a high number so we can understand through this that the balance is heavily distributed across the dataset.
- As the data information said it will be better to drop the duration column since duration is highly correlated in whether a potential client will buy a term deposit. Also, **duration is obtained after the call is made to the potential client** so if the target client has never received calls this feature is not that useful. The reason why duration is highly correlated with opening a term deposit is because the more the bank talks to a target client the higher the probability the target client will open a term deposit since a higher duration means a higher interest (commitment) from the potential client.

Note: There are not that much insights we can gain from the descriptive dataset since most of our descriptive data is located not in the "numeric" columns but in the "categorical columns".

Code

```
Out[2]:
```

	age	balance	day	d
count	11162.000000	11162.000000	11162.000000	1
mean	41.231948	1528.538524	15.658036	3
std	11.913369	3225.413326	8.420740	3
min	18.000000	-6847.000000	1.000000	2
25%	32.000000	122.000000	8.000000	1
50%	39.000000	550.000000	15.000000	2
75%	49.000000	1708.000000	22.000000	4
max	95.000000	81204.000000	31.000000	3

Fortunately, there are no **missing values**. If there were missing values we will have to fill them with the median, mean or mode. I tend to use the median but in this scenario there is no need to fill any missing values. This will definitely make our job easier!

Code

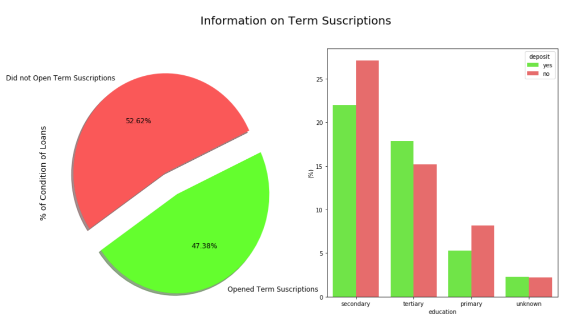
```
<class 'pandas.core.frame.DataFram
e'>
RangeIndex: 11162 entries, 0 to 1116
1
Data columns (total 17 columns):
age          11162 non-null int64
job          11162 non-null object
marital      11162 non-null object
education    11162 non-null object
default      11162 non-null object
balance      11162 non-null int64
housing      11162 non-null object
loan         11162 non-null object
contact      11162 non-null object
day          11162 non-null int64
month        11162 non-null object
duration     11162 non-null int64
campaign     11162 non-null int64
pdays        11162 non-null int64
previous     11162 non-null int64
poutcome     11162 non-null object
deposit      11162 non-null object
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
```

In [4]:

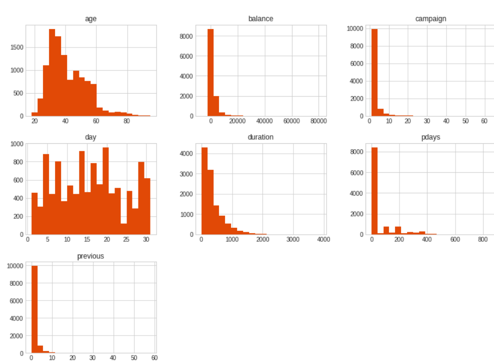
```
f, ax = plt.subplots(1,2, figsize=(16,8))

colors = ["#FA5858", "#64FE2E"]
labels ="Did not Open Term Suscriptions", "Opened Term
Suscriptions"

plt.suptitle('Information on Term Suscriptions', fonts
ize=20)

df["deposit"].value_counts().plot.pie(explode=[0,0.25
], autopct='%1.2f%%', ax=ax[0], shadow=True, colors=co
lors,
                                             labels=la
bels, fontsize=12, startangle=25)


# ax[0].set_title('State of Loan', fontsize=16)
ax[0].set_ylabel('% of Condition of Loans', fontsize=1
4)

# sns.countplot('loan_condition', data=df, ax=ax[1], pa
lette=colors)
# ax[1].set_title('Condition of Loans', fontsize=20)
# ax[1].set_xticklabels(['Good', 'Bad'], rotation='hori
zontal')
palette = ["#64FE2E", "#FA5858"]
```

```
sns.barplot(x="education", y="balance", hue="deposit", data=df, palette=palette, estimator=lambda x: len(x) / len(df) * 100)
ax[1].set(ylabel="(%)")
ax[1].set_xticklabels(df["education"].unique(), rotation=0, rotation_mode="anchor")
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
```



Code



In [6]:

```
df['deposit'].value_counts()
```

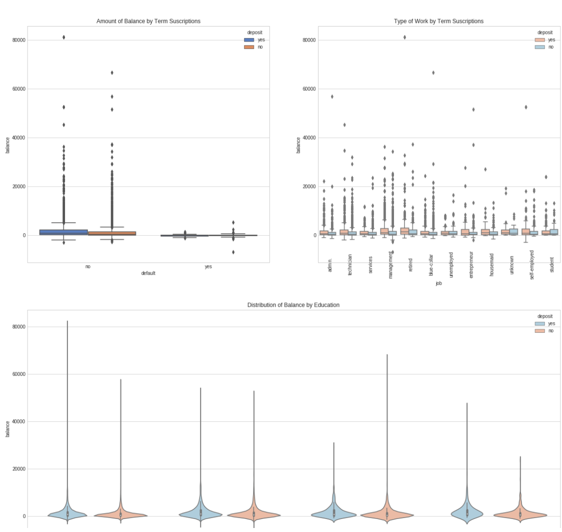
Out[6]:

```
no     5873
yes    5289
Name: deposit, dtype: int64
```

In [7]:

```
# plt.style.use('dark_background')
fig = plt.figure(figsize=(20,20))
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(212)

g = sns.boxplot(x="default", y="balance", hue="deposi
t",
                    data=df, palette="muted", ax=ax1)

g.set_title("Amount of Balance by Term Suscriptions")

# ax.set_xticklabels(df["default"].unique(), rotation=4
5, rotation_mode="anchor")

g1 = sns.boxplot(x="job", y="balance", hue="deposit",
                 data=df, palette="RdBu", ax=ax2)

g1.set_xticklabels(df["job"].unique(), rotation=90, ro
tation_mode="anchor")
g1.set_title("Type of Work by Term Suscriptions")

g2 = sns.violinplot(data=df, x="education", y="balanc
e", hue="deposit", palette="RdBu_r")

g2.set_title("Distribution of Balance by Education")


plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/sc
ipy/stats/stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimen
sional indexing is deprecated; use `arr[t
uple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which wil
l result either in an error or a differen
t result.
```



In [8]:

```
df.head()
```

Out[8]:

	age	job	marital	education	default	balance
0	59	admin.	married	secondary	no	2343
1	56	admin.	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	admin.	married	tertiary	no	184

Analysis by Occupation:

- **Number of Occupations:** Management is the occupation that is more prevalent in this dataset.
- **Age by Occupation:** As expected, the retired are the ones who have the highest median age while student are the lowest.
- **Balance by Occupation:** Management and Retirees are the ones who have the highest balance in their accounts.

Code

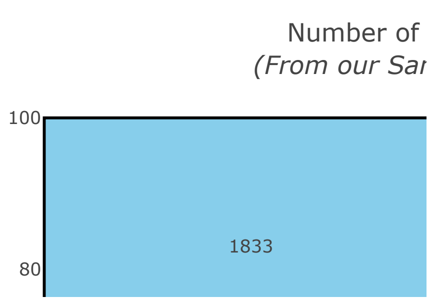
In [10]:

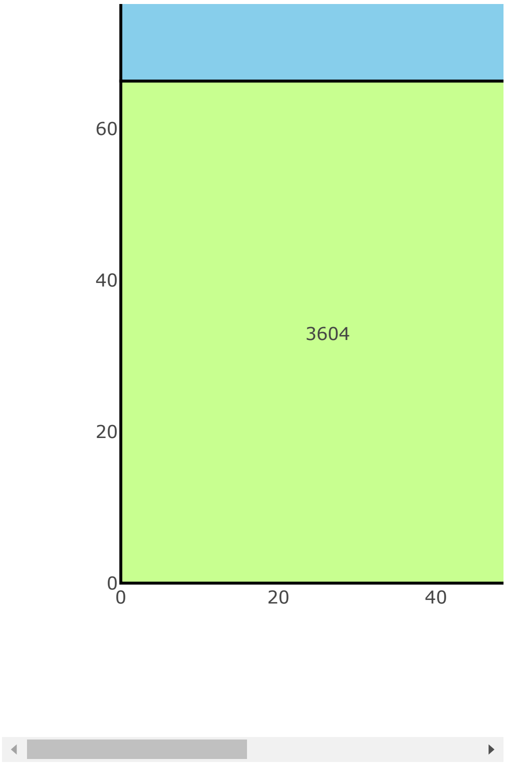
```
df.columns
```

Out[10]:

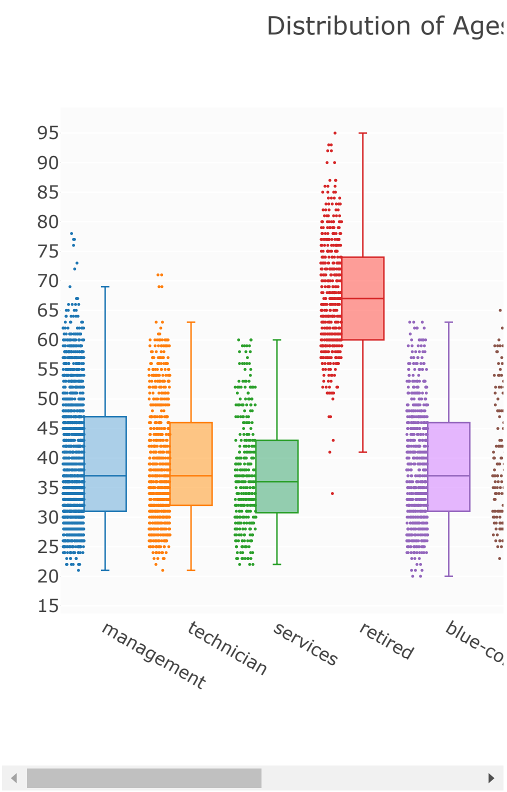
```
Index(['age', 'job', 'marital', 'educatio
n', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month',
'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'deposi
t'],
      dtype='object')
```

Code

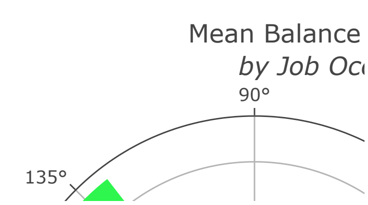


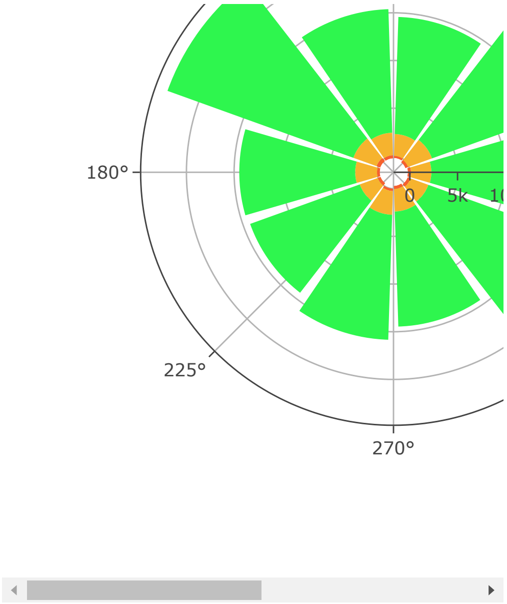

60
40
3604
20
0
0
20
40

Code


Distribution of Ages
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
management
technician
services
retired

Code


Mean Balance
90°
135°



Marital Status

Well in this analysis we didn't find any significant insights other than most **divorced individuals** are broke. No wonder since they have to split financial assets! Nevertheless, since no further insights have been found we will proceed to clustering marital status with education status. Let's see if we can find other groups of people in the sample population.

Code

Out[14]:

```
married     5815
single      3336
divorced    1174
Name: marital, dtype: int64
```

Code

Out[15]:

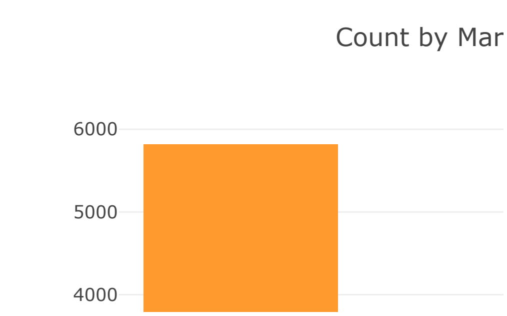
```
array(['married', 'divorced', 'singl
e'], dtype=object)
```

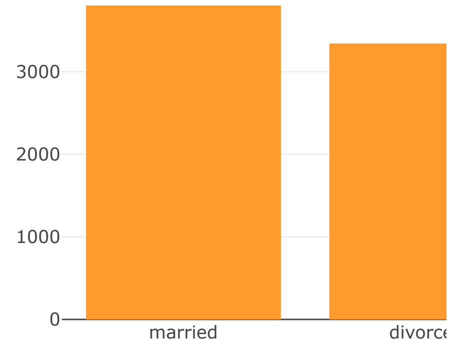
Code

Out[16]:

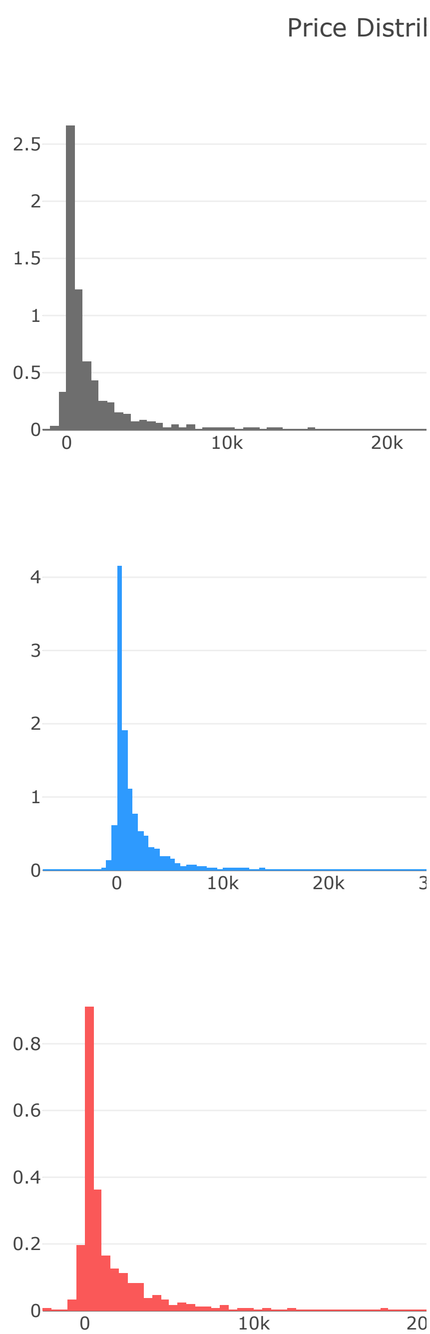
```
[5815, 3336, 1174]
```

Code




3000
2000
1000
0
married
divorce

Code


Price Distril
2.5
2
1.5
1
0.5
0
0
10k
20k
4
3
2
1
0
0
10k
20k
0.8
0.6
0.4
0.2
0
0
10k
20

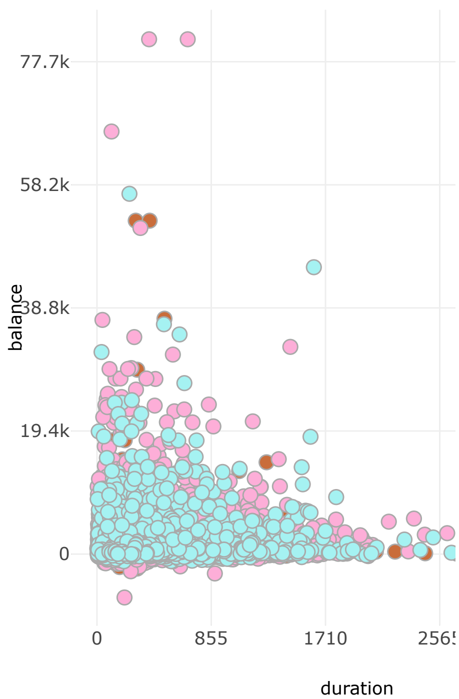
In [19]:

```
df.head()
```

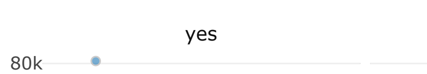
Out[19]:

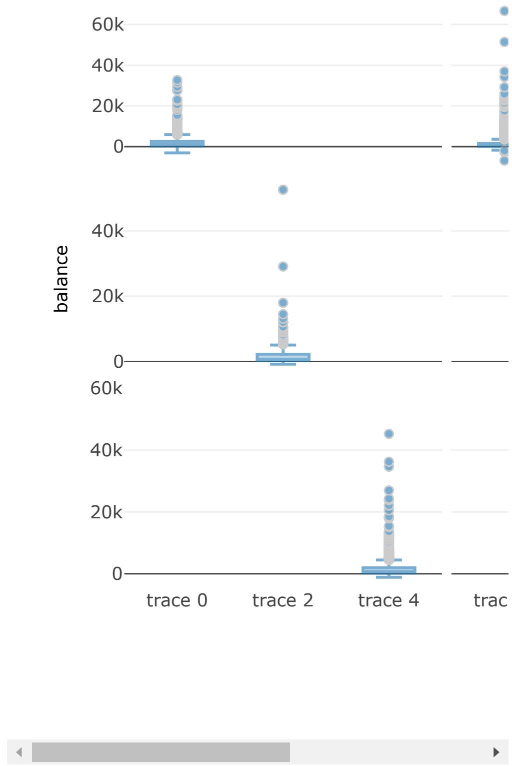
	age	job	marital	education	default	balance
0	59	management	married	secondary	no	2343
1	56	management	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	management	married	tertiary	no	184

Code



Code





In [22]:

```
df.head()
```

Out[22]:

	age	job	marital	education	default	balance
0	59	management	married	secondary	no	2343
1	56	management	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	management	married	tertiary	no	184

Clustering Marital Status and Education:

- **Marital Status:** As discussed previously, the impact of a divorce has a significant impact on the balance of the individual.
- **Education:** The level of education also has a significant impact on the amount of balance a prospect has.
- **Loans:** Whether the prospect has a previous loan has a significant impact on the amount of balance he or she has.

Code

Out[23]:

```
array(['secondary', 'tertiary', 'pri
mary'], dtype=object)
```

Code

Out[24]:

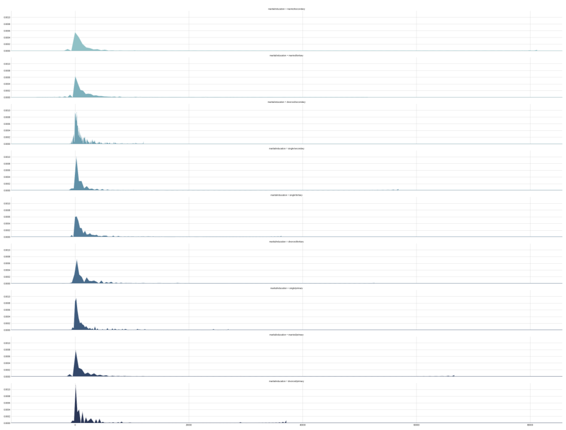
Out[24]:

	age	job	marital	education	default
0	59	management	married	secondary	no
1	56	management	married	secondary	no
2	41	technician	married	secondary	no
3	55	services	married	secondary	no
4	54	management	married	tertiary	no

Code

Out[25]:

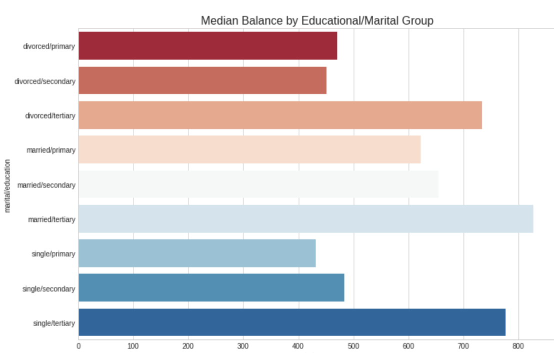
```
<seaborn.axisgrid.FacetGrid at 0x7f0
a4ac0e160>
```



Code

Out[26]:

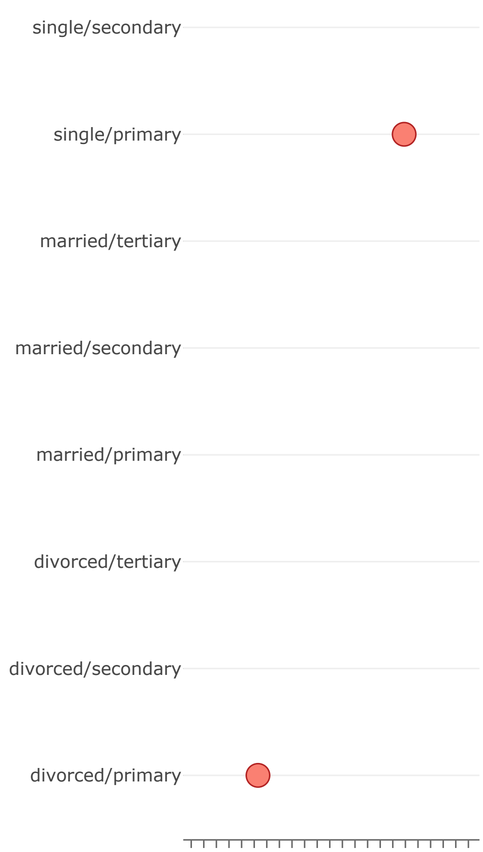
```
Text(0.5,1,'Median Balance by Educat
ional/Marital Group')
```



Code

The Imp

single/tertiary



In [28]:

```
df.head()
```

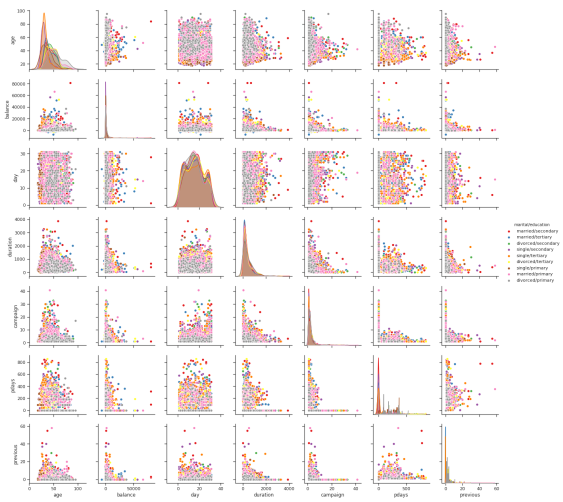
Out[28]:

	age	job	marital	education	default	balance
0	59	management	married	secondary	no	2343
1	56	management	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	management	married	tertiary	no	184

In [29]:

```
import seaborn as sns
sns.set(style="ticks")

sns.pairplot(df, hue="marital/education", palette="Set
1")
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/sc
ipy/stats/stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimen
sional indexing is deprecated; use `arr[t
uple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which wil
l result either in an error or a differen
t result.
```



In [30]:

```
df.head()
```

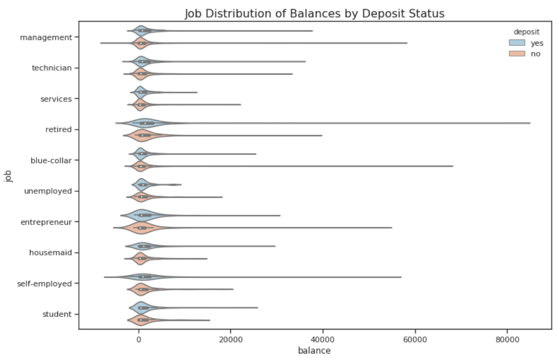
Out[30]:

	age	job	marital	education	default	balance
0	59	management	married	secondary	no	2343
1	56	management	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	management	married	tertiary	no	184

In [31]:

```
fig = plt.figure(figsize=(12,8))

sns.violinplot(x="balance", y="job", hue="deposit", pa
lette="RdBu_r",
            data=df);

plt.title("Job Distribution of Balances by Deposit Sta
tus", fontsize=16)

plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/sc
ipy/stats/stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimen
sional indexing is deprecated; use `arr[t
uple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which wil
l result either in an error or a differen
t result.
```



Campaign Duration:

- **Campaign Duration:** Hmm, we see that duration has a high correlation with term deposits meaning the higher the duration, the more likely it is for a client to open a term deposit.
- **Average Campaign Duration:** The average campaign duration is 374.76, let's see if clients that were above this average were more likely to open a term deposit.
- **Duration Status:** People who were above the duration status, were more likely to open a term deposit. 78% of the group that is above average in duration opened term deposits while those that were below average 32% opened term deposit accounts. This tells us that it will be a good idea to target individuals who are in the above average category.

In [32]:

```
df.drop(['marital/education', 'balance_status'], axis=
1, inplace=True)
```

In [33]:

```
df.head()
```

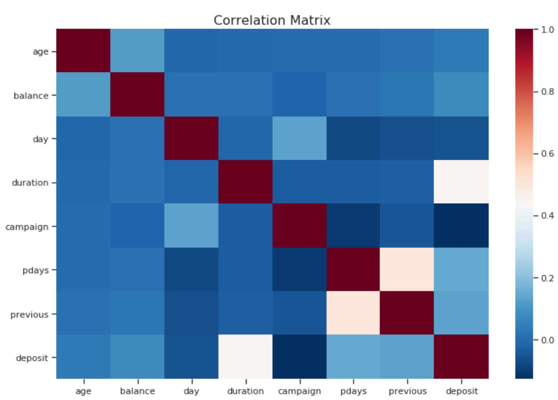
Out[33]:

	age	job	marital	education	default	balance
0	59	management	married	secondary	no	2343

1	56	management	married	secondary	no	45
2	41	technician	married	secondary	no	1270
3	55	services	married	secondary	no	2476
4	54	management	married	tertiary	no	184

In [34]:

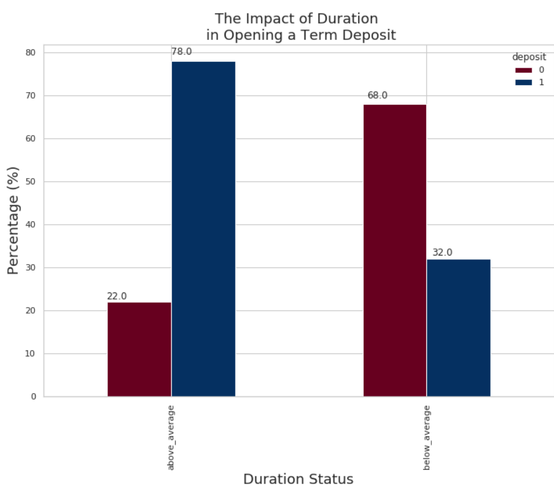
```
# Let's drop marital/education and balance status
# Let's scale both numeric and categorical vaues
# Then let's use a correlation matrix
# With that we can determine if duration has influence
 on term deposits

from sklearn.preprocessing import StandardScaler, OneH
otEncoder, LabelEncoder
fig = plt.figure(figsize=(12,8))
df['deposit'] = LabelEncoder().fit_transform(df['depos
it'])



# Separate both dataframes into
numeric_df = df.select_dtypes(exclude="object")
# categorical_df = df.select_dtypes(include="object")

corr_numeric = numeric_df.corr()


sns.heatmap(corr_numeric, cbar=True, cmap="RdBu_r")
plt.title("Correlation Matrix", fontsize=16)
plt.show()
```



In [35]:

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.set_style('whitegrid')
avg_duration = df['duration'].mean()

lst = [df]
df["duration_status"] = np.nan
```

```
for col in lst:
    col.loc[col["duration"] < avg_duration, "duration_status"] = "below_average"
    col.loc[col["duration"] > avg_duration, "duration_status"] = "above_average"

pct_term = pd.crosstab(df['duration_status'], df['deposit']).apply(lambda r: round(r/r.sum(), 2) * 100, axis=1)


ax = pct_term.plot(kind='bar', stacked=False, cmap='RdBu')
plt.title("The Impact of Duration \n in Opening a Term Deposit", fontsize=18)
plt.xlabel("Duration Status", fontsize=18);
plt.ylabel("Percentage (%)", fontsize=18)

for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.02, p.get_height() * 1.02))


plt.show()
```



Classification Model:

In [36]:

```
dep = term_deposits['deposit']
term_deposits.drop(labels=['deposit'], axis=1,inplace=True)
term_deposits.insert(0, 'deposit', dep)
term_deposits.head()
# housing has a -20% correlation with deposit let's see how it is distributed.
# 52 %
term_deposits["housing"].value_counts()/len(term_deposits)
```

Out[36]:

```
no     0.526877
yes    0.473123
Name: housing, dtype: float64
```

In [37]:

```
term_deposits["loan"].value_counts()/len(term_deposits
)
```

Out[37]:

```
no     0.869199
yes    0.130801
Name: loan, dtype: float64
```

Stratified Sampling:

Stratified Sampling: Is an important concept that is often missed when developing a model either for regression or classification. Remember, that in order to avoid overfitting of our data we must implement a cross validation however, we must make sure that at least the features that have the greatest influence on our label (whether a potential client will open a term deposit or not) is equally distributed. What do I mean by this?

Personal Loans:
For instance, having a personal loan is an important feature that determines whether a potential client will open a term deposit or not. To confirm it has a heavy weight on the final output you can check the correlation matrix above and you can see it has a -11% correlation with opening a deposit. What steps we should take before implementing stratified sampling in our train and test data?
1) We need to see how our data is distributed.
2) After noticiing that the column of loan contains 87% of "no" (Does not have personal loans) and 13% of "yes" (Have personal loans.)
3) We want to make sure that our training and test set contains the same ratio of 87% "no" and 13% "yes"." Stratified Sampling: Is an important concept that is often missed when developing a model either for regression or classification. Remember, that in order to avoid overfitting of our data we must implement a cross validation however, we must make sure that at least the features that have the greatest influence on our label (whether a potential client will open a term deposit or not) is equally distributed. What do I mean by this?

Personal Loans:
For instance, having a personal loan is an important feature that determines whether a potential client will open a term deposit or not. To confirm it has a heavy weight on the final output you can check the correlation matrix above and you can see it has a -11% correlation with opening a deposit. What steps we should take before implementing stratified sampling in our train and test data?

1) We need to see how our data is distributed.
2) After noticiing that the column of loan contains 87% of "no" (Does not have personal loans) and 13% of "yes" (Have personal loans.)
3) We want to make sure that our training and test set contains the same ratio of 87% "no" and 13% "yes".

In [38]:

```
from sklearn.model_selection import StratifiedShuffleS
plit
# Here we split the data into training and test sets an
d implement a stratified shuffle split.
stratified = StratifiedShuffleSplit(n_splits=1, test_s
ize=0.2, random_state=42)

for train_set, test_set in stratified.split(term_depos
its, term_deposits["loan"]):
    stratified_train = term_deposits.loc[train_set]
    stratified_test = term_deposits.loc[test_set]

stratified_train["loan"].value_counts()/len(df)
stratified_test["loan"].value_counts()/len(df)
```

Out[38]:

```
no     0.196219
yes    0.029519
Name: loan, dtype: float64
```

In [39]:

```
# Separate the labels and the features.
train_data = stratified_train # Make a copy of the stra
tified training set.
test_data = stratified_test
train_data.shape
test_data.shape
train_data['deposit'].value_counts()
```

Out[39]:

```
no     4697
yes    4232
Name: deposit, dtype: int64
```

In [40]:

```
# Definition of the CategoricalEncoder class, copied fr
om PR #9151.
# Just run this cell, or copy it to your code, no need
 to try to
# understand every line.
# Code reference Hands on Machine Learning with Scikit
 Learn and Tensorflow by Aurelien Geron.

from sklearn.base import BaseEstimator, TransformerMix
in
from sklearn.utils import check_array
```

```
from sklearn.utils import check_array
from sklearn.preprocessing import LabelEncoder
from scipy import sparse

class CategoricalEncoder(BaseEstimator, TransformerMix
in):
    """Encode categorical features as a numeric array.
    The input to this transformer should be a matrix of
integers or strings,
    denoting the values taken on by categorical (discre
te) features.
    The features can be encoded using a one-hot aka one
-of-K scheme
    (``encoding='onehot'``, the default) or converted t
o ordinal integers
    (``encoding='ordinal'``).
    This encoding is needed for feeding categorical dat
a to many scikit-learn
    estimators, notably linear models and SVMs with the
standard kernels.
    Read more in the :ref:`User Guide <preprocessing_ca
tegorical_features>`.
    Parameters
    ----------
    encoding : str, 'onehot', 'onehot-dense' or 'ordina
l'
        The type of encoding to use (default is 'oneho
t'):
        - 'onehot': encode the features using a one-hot
aka one-of-K scheme
          (or also called 'dummy' encoding). This creat
es a binary column for
          each category and returns a sparse matrix.
        - 'onehot-dense': the same as 'onehot' but retu
rns a dense array
          instead of a sparse matrix.
        - 'ordinal': encode the features as ordinal int
egers. This results in
          a single column of integers (0 to n_categorie
s - 1) per feature.
    categories : 'auto' or a list of lists/arrays of va
lues.
        Categories (unique values) per feature:
        - 'auto' : Determine categories automatically f
rom the training data.
        - list : ``categories[i]`` holds the categories
expected in the ith
          column. The passed categories are sorted befo
re encoding the data
          (used categories can be found in the ``catego
ries_`` attribute).
    dtype : number type, default np.float64
        Desired dtype of output.
    handle_unknown : 'error' (default) or 'ignore'
        Whether to raise an error or ignore if a unknow
n categorical feature is
        present during transform (default is to raise).
When this is parameter
        is set to 'ignore' and an unknown category is e
```

```
ncountered during
        transform, the resulting one-hot encoded column
s for this feature
        will be all zeros.
        Ignoring unknown categories is not supported fo
r
        ``encoding='ordinal'``.
    Attributes
    ----------
    categories_ : list of arrays
        The categories of each feature determined durin
g fitting. When
        categories were specified manually, this holds
 the sorted categories
        (in order corresponding with output of `transfo
rm`).
    Examples
    --------
    Given a dataset with three features and two sample
s, we let the encoder
    find the maximum value per feature and transform th
e data to a binary
    one-hot encoding.
    >>> from sklearn.preprocessing import CategoricalEn
coder
    >>> enc = CategoricalEncoder(handle_unknown='ignor
e')
    >>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1,
 0, 2]])
    ... # doctest: +ELLIPSIS
    CategoricalEncoder(categories='auto', dtype=<... 'n
umpy.float64'>,
              encoding='onehot', handle_unknown='ignor
e')
    >>> enc.transform([[0, 1, 1], [1, 0, 4]]).toarray()
    array([[ 1.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,
 0.],
           [ 0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,
 0.]])
    See also
    --------
    sklearn.preprocessing.OneHotEncoder : performs a on
e-hot encoding of
      integer ordinal features. The ``OneHotEncoder ass
umes`` that input
      features take on values in the range ``[0, max(fe
ature)]`` instead of
      using the unique values.
    sklearn.feature_extraction.DictVectorizer : perform
s a one-hot encoding of
      dictionary items (also handles string-valued feat
ures).
    sklearn.feature_extraction.FeatureHasher : performs
an approximate one-hot
      encoding of dictionary items or strings.
    """

    def __init__(self, encoding='onehot', categories=
```

```
'auto', dtype=np.float64,
                 handle_unknown='error'):
        self.encoding = encoding
        self.categories = categories
        self.dtype = dtype
        self.handle_unknown = handle_unknown

    def fit(self, X, y=None):
        """Fit the CategoricalEncoder to X.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_feature]
            The data to determine the categories of eac
h feature.
        Returns
        -------
        self
        """

        if self.encoding not in ['onehot', 'onehot-den
se', 'ordinal']:
            template = ("encoding should be either 'on
ehot', 'onehot-dense' "
                        "or 'ordinal', got %s")
            raise ValueError(template % self.handle_un
known)

        if self.handle_unknown not in ['error', 'ignor
e']:
            template = ("handle_unknown should be eith
er 'error' or "
                        "'ignore', got %s")
            raise ValueError(template % self.handle_un
known)

        if self.encoding == 'ordinal' and self.handle_
unknown == 'ignore':
            raise ValueError("handle_unknown='ignore'
 is not supported for"
                             " encoding='ordinal'")

        X = check_array(X, dtype=np.object, accept_spa
rse='csc', copy=True)
        n_samples, n_features = X.shape

        self._label_encoders_ = [LabelEncoder() for _
in range(n_features)]

        for i in range(n_features):
            le = self._label_encoders_[i]
            Xi = X[:, i]
            if self.categories == 'auto':
                le.fit(Xi)
            else:
                valid_mask = np.in1d(Xi, self.categori
es[i])
                if not np.all(valid_mask):
                    if self.handle_unknown == 'error':
```

```
                        diff = np.unique(Xi[~valid_mas
k])
                        msg = ("Found unknown categori
es {0} in column {1}"
                               " during fit".format(di
ff, i))
                        raise ValueError(msg)
                le.classes_ = np.array(np.sort(self.ca
tegories[i]))

        self.categories_ = [le.classes_ for le in self
._label_encoders_]

        return self

    def transform(self, X):
        """Transform X using one-hot encoding.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_features]
            The data to encode.
        Returns
        -------
        X_out : sparse matrix or a 2-d array
            Transformed input.
        """
        X = check_array(X, accept_sparse='csc', dtype=
np.object, copy=True)
        n_samples, n_features = X.shape
        X_int = np.zeros_like(X, dtype=np.int)
        X_mask = np.ones_like(X, dtype=np.bool)

        for i in range(n_features):
            valid_mask = np.in1d(X[:, i], self.categor
ies_[i])

            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(X[~valid_mask, i
])
                    msg = ("Found unknown categories
{0} in column {1}"
                           " during transform".format(
diff, i))
                    raise ValueError(msg)
                else:
                    # Set the problematic rows to an ac
ceptable value and
                    # continue `The rows are marked `X_
mask` and will be
                    # removed later.
                    X_mask[:, i] = valid_mask
                    X[:, i][~valid_mask] = self.catego
ries_[i][0]
            X_int[:, i] = self._label_encoders_[i].tra
nsform(X[:, i])

        if self.encoding == 'ordinal':
```

```
            return X_int.astype(self.dtype, copy=False
)

        mask = X_mask.ravel()
        n_values = [cats.shape[0] for cats in self.cat
egories_]
        n_values = np.array([0] + n_values)
        indices = np.cumsum(n_values)

        column_indices = (X_int + indices[:-1]).ravel
()[mask]
        row_indices = np.repeat(np.arange(n_samples, d
type=np.int32),
                                n_features)[mask]
        data = np.ones(n_samples * n_features)[mask]

        out = sparse.csc_matrix((data, (row_indices, c
olumn_indices)),
                                shape=(n_samples, indi
ces[-1]),
                                dtype=self.dtype).tocs
r()
        if self.encoding == 'onehot-dense':
            return out.toarray()
        else:
            return out
```

In [41]:

```
from sklearn.base import BaseEstimator, TransformerMix
in

# A class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixi
n):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names]
```

In [42]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8929 entries, 9867 to 9672
Data columns (total 17 columns):
deposit      8929 non-null object
age          8929 non-null int64
job          8929 non-null object
marital      8929 non-null object
education    8929 non-null object
default      8929 non-null object
balance      8929 non-null int64
```

```
housing      8929 non-null object
loan         8929 non-null object
contact      8929 non-null object
day          8929 non-null int64
month        8929 non-null object
duration     8929 non-null int64
campaign     8929 non-null int64
pdays        8929 non-null int64
previous     8929 non-null int64
poutcome     8929 non-null object
dtypes: int64(7), object(10)
memory usage: 1.2+ MB
```

In [43]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Making pipelines
numerical_pipeline = Pipeline([
    ("select_numeric", DataFrameSelector(["age", "bala
nce", "day", "campaign", "pdays", "previous","duratio
n"])),
    ("std_scaler", StandardScaler()),
])

categorical_pipeline = Pipeline([
    ("select_cat", DataFrameSelector(["job", "educatio
n", "marital", "default", "housing", "loan", "contact"
, "month",
                                     "poutcome"])),
    ("cat_encoder", CategoricalEncoder(encoding='oneho
t-dense'))
])

from sklearn.pipeline import FeatureUnion
preprocess_pipeline = FeatureUnion(transformer_list=[
        ("numerical_pipeline", numerical_pipeline),
        ("categorical_pipeline", categorical_pipeline
),
    ])
```

In [44]:

```
X_train = preprocess_pipeline.fit_transform(train_data
)
X_train
```

```
/opt/conda/lib/python3.6/site-packages/sk
learn/preprocessing/data.py:645: DataConv
ersionWarning:

Data with input dtype int64 were all conv
erted to float64 by StandardScaler.

/opt/conda/lib/python3.6/site-packages/sk
learn/base.py:464: DataConversionWarning:
```

```
Data with input dtype int64 were all conv
erted to float64 by StandardScaler.
```

Out[44]:

```
array([[ 1.14643868,  1.68761105,  1.6944
2818, ...,  0.        ,
         0.        ,  1.        ],
       [-0.86102339, -0.35066205, -0.5560
058 , ...,  0.        ,
         0.        ,  1.        ],
       [-0.94466765, -0.20504785,  0.3915
4535, ...,  0.        ,
         0.        ,  1.        ],
       ...,
       [-0.86102339, -0.26889658, -1.0297
8138, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.2263519 , -0.32166951,  0.5099
8924, ...,  0.        ,
         0.        ,  1.        ],
       [-0.61009063, -0.34740446,  1.6944
2818, ...,  1.        ,
         0.        ,  0.        ]])
```

In [45]:

```
y_train = train_data['deposit']
y_test = test_data['deposit']
y_train.shape
```

Out[45]:

```
(8929,)
```

In [46]:

```
from sklearn.preprocessing import LabelEncoder

encode = LabelEncoder()
y_train = encode.fit_transform(y_train)
y_test = encode.fit_transform(y_test)
y_train_yes = (y_train == 1)
y_train
y_train_yes
```

Out[46]:

```
array([False, False,  True, ...,  True,
True, False])
```

In [47]:

```
some_instance = X_train[1250]
```

In [48]:

```
# Time for Classification Models
import time


from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, Labe
lEncoder

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifie
r
from sklearn.gaussian_process.kernels import RBF
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB


dict_classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Nearest Neighbors": KNeighborsClassifier(),
    "Linear SVM": SVC(),
    "Gradient Boosting Classifier": GradientBoostingCl
assifier(),
    "Decision Tree": tree.DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimato
rs=18),
    "Neural Net": MLPClassifier(alpha=1),
    "Naive Bayes": GaussianNB()
}
```

In [49]:

```
#  Thanks to Ahspinar for the function.
no_classifiers = len(dict_classifiers.keys())

def batch_classify(X_train, Y_train, verbose = True):
    df_results = pd.DataFrame(data=np.zeros(shape=(no_
classifiers,3)), columns = ['classifier', 'train_scor
e', 'training_time'])
    count = 0
    for key, classifier in dict_classifiers.items():
        t_start = time.clock()
        classifier.fit(X_train, Y_train)
        t_end = time.clock()
        t_diff = t_end - t_start
        train_score = classifier.score(X_train, Y_trai
n)
        df_results.loc[count,'classifier'] = key
        df_results.loc[count,'train_score'] = train_sc
ore
        df_results.loc[count,'training_time'] = t_diff
        if verbose:
            print("trained {c} in {f:.2f} s".format(c=
```

```
key, f=t_diff))
        count+=1
    return df_results
```

In [50]:

```
df_results = batch_classify(X_train, y_train)
print(df_results.sort_values(by='train_score', ascendi
ng=False))
```

```
/opt/conda/lib/python3.6/site-packages/sk
learn/linear_model/logistic.py:433: Futur
eWarning:

Default solver will be changed to 'lbfgs'
in 0.22. Specify a solver to silence this
warning.



trained Logistic Regression in 0.05 s
trained Nearest Neighbors in 0.13 s

/opt/conda/lib/python3.6/site-packages/sk
learn/svm/base.py:196: FutureWarning:

The default value of gamma will change fr
om 'auto' to 'scale' in version 0.22 to a
ccount better for unscaled features. Set
gamma explicitly to 'auto' or 'scale' to
avoid this warning.



trained Linear SVM in 6.21 s
trained Gradient Boosting Classifier in
1.50 s
trained Decision Tree in 0.08 s
trained Random Forest in 0.17 s
trained Neural Net in 13.22 s
trained Naive Bayes in 0.03 s
                     classifier  train_sc
ore  training_time
4                 Decision Tree     1.000
000       0.080934
5                 Random Forest     0.995
968       0.171750
1             Nearest Neighbors     0.863
255       0.126915
3  Gradient Boosting Classifier     0.861
463       1.501796
6                    Neural Net     0.854
071      13.216751
2                    Linear SVM     0.852
391       6.206555
0           Logistic Regression     0.830
776       0.053495
7                   Naive Bayes     0.721
```

```
693       0.033293
```

Avoiding Overfitting:

Brief Description of Overfitting?
This is an error in the modeling algorithm that takes into consideration random noise in the fitting process rather than the pattern itself. You can see that this occurs when the model gets an awsome score in the training set but when we use the test set (Unknown data for the model) we get an awful score. This is likely to happen because of overfitting of the data (taking into consideration random noise in our pattern). What we want our model to do is to take the overall pattern of the data in order to correctly classify whether a potential client will suscribe to a term deposit or not. In the examples above, it is most likely that the Decision Tree Classifier and Random Forest classifiers are overfitting since they both give us nearly perfect scores (100% and 99%) accuracy scores.

How can we avoid Overfitting?
The best alternative to avoid overfitting is to use cross validation. Taking the training test and splitting it. For instance, if we split it by 3, 2/3 of the data or 66% will be used for training and 1/3 33% will be used or testing and we will do the testing process three times. This algorithm will iterate through all the training and test sets and the main purpose of this is to grab the overall pattern of the data.

In [51]:

```
# Use Cross-validation.
from sklearn.model_selection import cross_val_score

# Logistic Regression
log_reg = LogisticRegression()
log_scores = cross_val_score(log_reg, X_train, y_train
, cv=3)
log_reg_mean = log_scores.mean()

# SVC
svc_clf = SVC()
svc_scores = cross_val_score(svc_clf, X_train, y_train
, cv=3)
svc_mean = svc_scores.mean()

# KNearestNeighbors
knn_clf = KNeighborsClassifier()
knn_scores = cross_val_score(knn_clf, X_train, y_train
, cv=3)
knn_mean = knn_scores.mean()

# Decision Tree
tree_clf = tree.DecisionTreeClassifier()
tree_scores = cross_val_score(tree_clf, X_train, y_tra
in, cv=3)
tree_mean = tree_scores.mean()
```

```
# Gradient Boosting Classifier
grad_clf = GradientBoostingClassifier()
grad_scores = cross_val_score(grad_clf, X_train, y_tra
in, cv=3)
grad_mean = grad_scores.mean()

# Random Forest Classifier
rand_clf = RandomForestClassifier(n_estimators=18)
rand_scores = cross_val_score(rand_clf, X_train, y_tra
in, cv=3)
rand_mean = rand_scores.mean()

# NeuralNet Classifier
neural_clf = MLPClassifier(alpha=1)
neural_scores = cross_val_score(neural_clf, X_train, y
_train, cv=3)
neural_mean = neural_scores.mean()

# Naives Bayes
nav_clf = GaussianNB()
nav_scores = cross_val_score(nav_clf, X_train, y_train
, cv=3)
nav_mean = neural_scores.mean()

# Create a Dataframe with the results.
d = {'Classifiers': ['Logistic Reg.', 'SVC', 'KNN', 'D
ec Tree', 'Grad B CLF', 'Rand FC', 'Neural Classifier'
, 'Naives Bayes'],
    'Crossval Mean Scores': [log_reg_mean, svc_mean, k
nn_mean, tree_mean, grad_mean, rand_mean, neural_mean,
nav_mean]}

result_df = pd.DataFrame(data=d)
```

```
/opt/conda/lib/python3.6/site-packages/sk
learn/linear_model/logistic.py:433: Futur
eWarning:

Default solver will be changed to 'lbfgs'
in 0.22. Specify a solver to silence this
warning.

/opt/conda/lib/python3.6/site-packages/sk
learn/linear_model/logistic.py:433: Futur
eWarning:

Default solver will be changed to 'lbfgs'
in 0.22. Specify a solver to silence this
warning.

/opt/conda/lib/python3.6/site-packages/sk
learn/linear_model/logistic.py:433: Futur
eWarning:

Default solver will be changed to 'lbfgs'
in 0.22. Specify a solver to silence this
warning.
```

```
/opt/conda/lib/python3.6/site-packages/sk
learn/svm/base.py:196: FutureWarning:

The default value of gamma will change fr
om 'auto' to 'scale' in version 0.22 to a
ccount better for unscaled features. Set
gamma explicitly to 'auto' or 'scale' to
avoid this warning.

/opt/conda/lib/python3.6/site-packages/sk
learn/svm/base.py:196: FutureWarning:

The default value of gamma will change fr
om 'auto' to 'scale' in version 0.22 to a
ccount better for unscaled features. Set
gamma explicitly to 'auto' or 'scale' to
avoid this warning.

/opt/conda/lib/python3.6/site-packages/sk
learn/svm/base.py:196: FutureWarning:

The default value of gamma will change fr
om 'auto' to 'scale' in version 0.22 to a
ccount better for unscaled features. Set
gamma explicitly to 'auto' or 'scale' to
avoid this warning.
```

In [52]:

```
# All our models perform well but I will go with Gradie
ntBoosting.
result_df = result_df.sort_values(by=['Crossval Mean S
cores'], ascending=False)
result_df
```

Out[52]:

	Classifiers	Crossval Mean Scores
6	Neural Classifier	0.847689
7	Naives Bayes	0.847689
4	Grad B CLF	0.845224
5	Rand FC	0.843655
1	SVC	0.840186
0	Logistic Reg.	0.828425
2	KNN	0.804458
3	Dec Tree	0.786313

Confusion Matrix:

		Prediction	
		0	1
Actual	0	TN	FP
	1	FN	TP

Insights of a Confusion Matrix:

The main purpose of a confusion matrix is to see how our model is performing when it comes to classifying potential clients that are likely to suscribe to a term deposit. We will see in the confusion matrix four terms the True Positives, False Positives, True Negatives and False Negatives.

Positive/Negative: Type of Class (label) ["No", "Yes"] **True/False:** Correctly or Incorrectly classified by the model.

True Negatives (Top-Left Square): This is the number of **correctly** classifications of the "No" class or potenial clients that are **not willing** to suscribe a term deposit.

False Negatives (Top-Right Square): This is the number of **incorrectly** classifications of the "No" class or potential clients that are **not willing** to suscribe a term depositt.

False Positives (Bottom-Left Square): This is the number of **incorrectly** classifications of the "Yes" class or potential clients that are **willing** to suscribe a term deposit.

True Positives (Bottom-Right Square): This is the number of **correctly** classifications of the "Yes" class or potenial clients that are **willing** to suscribe a term deposit.

In [53]:

```
# Cross validate our Gradient Boosting Classifier
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(grad_clf, X_train, y_
train, cv=3)
```

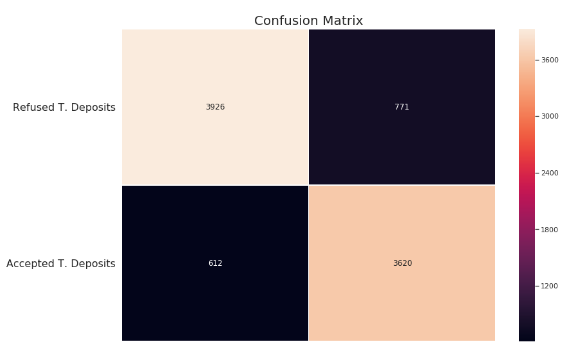
In [54]:

```
from sklearn.metrics import accuracy_score
grad_clf.fit(X_train, y_train)
print ("Gradient Boost Classifier accuracy is %2.2f" %
accuracy_score(y_train, y_train_pred))
```

```
Gradient Boost Classifier accuracy is 0.8
5
```

In [55]:

```
from sklearn.metrics import confusion_matrix
# 4697: no's, 4232: yes
conf_matrix = confusion_matrix(y_train, y_train_pred)
f, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", linewidt
hs=.5, ax=ax)
plt.title("Confusion Matrix", fontsize=20)
plt.subplots_adjust(left=0.15, right=0.99, bottom=0.15
, top=0.99)
ax.set_yticks(np.arange(conf_matrix.shape[0]) + 0.5, m
inor=False)
ax.set_xticklabels("")
ax.set_yticklabels(['Refused T. Deposits', 'Accepted
 T. Deposits'], fontsize=16, rotation=360)
plt.show()
```



Precision and Recall:

Recall: Is the total number of "Yes" in the label column of the dataset. So how many "Yes" labels does our model detect.

Precision: Means how sure is the prediction of our model that the actual label is a "Yes".

Recall Precision Tradeoff:

As the precision gets higher the recall gets lower and vice versa. For instance, if we increase the precision from 30% to 60% the model is picking the predictions that the model believes is 60% sure. If there is an instance where the model believes that is 58% likely to be a potential client that will suscribe to a term deposit then the model will classify it as a **"No."** However, that instance was actually a **"Yes"** (potential client did suscribe to a term deposit.) That is why the higher the precision the more likely the model is to miss instances that are actually a **"Yes"!**

In [56]:

```
# Let's find the scores  for precision and recall.
from sklearn.metrics import precision_score, recall_score
# The model is 77% sure that the potential client will suscribe to a term deposit.
# The model is only retaining 60% of clients that agree to suscribe a term deposit.
print('Precision Score: ', precision_score(y_train, y_train_pred))
# The classifier only detects 60% of potential clients that will suscribe to a term deposit.
print('Recall Score: ', recall_score(y_train, y_train_pred))
```

```
Precision Score:  0.8244135732179458
Recall Score:  0.8553875236294896
```

In [57]:

```
from sklearn.metrics import f1_score

f1_score(y_train, y_train_pred)
```

Out[57]:

```
0.8396149831845067
```

In [58]:

```
y_scores = grad_clf.decision_function([some_instance])
y_scores
```

Out[58]:

```
array([-3.65645629])
```

In [59]:

```
# Increasing the threshold decreases the recall.
threshold = 0
y_some_digit_pred = (y_scores > threshold)
```

In [60]:

```
y_scores = cross_val_predict(grad_clf, X_train, y_trai
n, cv=3, method="decision_function")
neural_y_scores = cross_val_predict(neural_clf, X_trai
n, y_train, cv=3, method="predict_proba")
naives_y_scores = cross_val_predict(nav_clf, X_train,
y_train, cv=3, method="predict_proba")
```

In [61]:

```
# hack to work around issue #9589 introduced in Scikit-
Learn 0.19.0
if y_scores.ndim == 2:
    y_scores = y_scores[:, 1]

if neural_y_scores.ndim == 2:
    neural_y_scores = neural_y_scores[:, 1]

if naives_y_scores.ndim == 2:
    naives_y_scores = naives_y_scores[:, 1]
```

In [62]:

```
y_scores.shape
```

Out[62]:

```
(8929,)
```

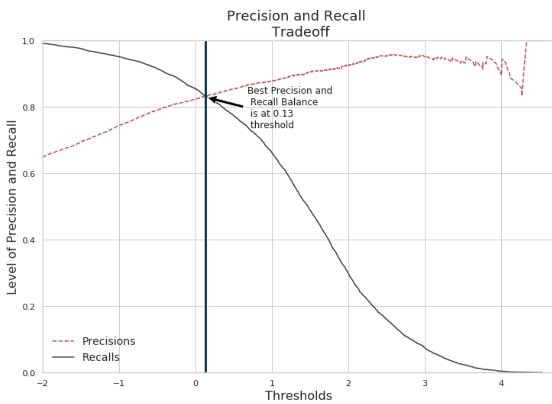
In [63]:

```
# How can we decide which threshold to use? We want to
 return the scores instead of predictions with this cod
e.
from sklearn.metrics import precision_recall_curve

precisions, recalls, threshold = precision_recall_curv
e(y_train, y_scores)
```

In [64]:

```
def precision_recall_curve(precisions, recalls, thresh
olds):
    fig, ax = plt.subplots(figsize=(12,8))
    plt.plot(thresholds, precisions[:-1], "r--", label
="Precisions")
    plt.plot(thresholds, recalls[:-1], "#424242", labe
l="Recalls")
    plt.title("Precision and Recall \n Tradeoff", font
size=18)
    plt.ylabel("Level of Precision and Recall", fontsi
ze=16)
    plt.xlabel("Thresholds", fontsize=16)
    plt.legend(loc="best", fontsize=14)
    plt.xlim([-2, 4.7])
    plt.ylim([0, 1])
```

```
    plt.axvline(x=0.13, linewidth=3, color="#0B3861")
    plt.annotate('Best Precision and \n Recall Balance
\n is at 0.13 \n threshold ', xy=(0.13, 0.83), xytext=
(55, -40),
             textcoords="offset points",
            arrowprops=dict(facecolor='black', shrink=
0.05),
                fontsize=12,
                color='k')

precision_recall_curve(precisions, recalls, threshold)
plt.show()
```



ROC Curve (Receiver Operating Characteristic):

The **ROC curve** tells us how well our classifier is classifying between term deposit suscriptions (True Positives) and non-term deposit suscriptions. The **X-axis** is represented by False positive rates (Specificity) and the **Y-axis** is represented by the True Positive Rate (Sensitivity.) As the line moves the threshold of the classification changes giving us different values. The closer is the line to our top left corner the better is our model separating both classes.

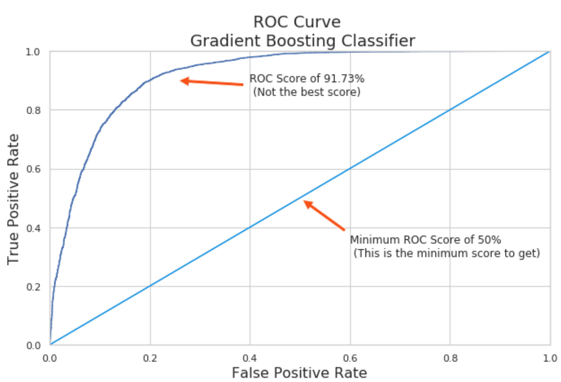
In [65]:

```
from sklearn.metrics import roc_curve
# Gradient Boosting Classifier
# Neural Classifier
# Naives Bayes Classifier
grd_fpr, grd_tpr, thresold = roc_curve(y_train, y_scor
es)
neu_fpr, neu_tpr, neu_threshold = roc_curve(y_train, n
eural_y_scores)
nav_fpr, nav_tpr, nav_threshold = roc_curve(y_train, n
aives_y_scores)
```

```
aives_y_scores)
```

In [66]:

```
def graph_roc_curve(false_positive_rate, true_positive
_rate, label=None):
    plt.figure(figsize=(10,6))
    plt.title('ROC Curve \n Gradient Boosting Classifi
er', fontsize=18)
    plt.plot(false_positive_rate, true_positive_rate,
label=label)
    plt.plot([0, 1], [0, 1], '#0C8EE0')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('ROC Score of 91.73% \n (Not the best
score)', xy=(0.25, 0.9), xytext=(0.4, 0.85),
            arrowprops=dict(facecolor='#F75118', shrin
k=0.05),
            )
    plt.annotate('Minimum ROC Score of 50% \n (This is
the minimum score to get)', xy=(0.5, 0.5), xytext=(0.6
, 0.3),
                arrowprops=dict(facecolor='#F75118', s
hrink=0.05),
                )


graph_roc_curve(grd_fpr, grd_tpr, threshold)
plt.show()
```



In [67]:

```
from sklearn.metrics import roc_auc_score

print('Gradient Boost Classifier Score: ', roc_auc_sco
re(y_train, y_scores))
print('Neural Classifier Score: ', roc_auc_score(y_tra
in, neural_y_scores))
print('Naives Bayes Classifier: ', roc_auc_score(y_tra
in, naives_y_scores))
```

```
Gradient Boost Classifier Score:  0.91731
28596743366
```

```
Neural Classifier Score:  0.9167698643666
292
Naives Bayes Classifier:  0.8033639599422
55
```

In [68]:

```
def graph_roc_curve_multiple(grd_fpr, grd_tpr, neu_fpr
, neu_tpr, nav_fpr, nav_tpr):
    plt.figure(figsize=(8,6))
    plt.title('ROC Curve \n Top 3 Classifiers', fontsi
ze=18)
    plt.plot(grd_fpr, grd_tpr, label='Gradient Boostin
```


Did you find this Kernel useful?
Show your appreciation with an upvote

134

Data

Data Sources

Bank Marketing Dataset

bank.csv 17 columns

Bank Marketing Dataset

Predicting Term Deposit Suscriptions

Last Updated: 2 years ago (Version 1)

About this Dataset

Context

Find the best strategies to improve for the next marketing campaign. How can the financial institution have a greater effectiveness for future marketing campaigns? In order to answer this, we have to analyze the last marketing campaign the bank performed and identify the patterns that will help us find conclusions in order to develop future strategies.

Source

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

Comments (55)

Sort by

All Comments | Hotness

Click here to comment...

Utku_Kubilay • Posted on Latest Version • 5 months ago • Options • Reply 8

I am looking forward to see your next kernel :) Nice work

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Many thanks! I'm glad you liked it.

Karan Jakhar • Posted on Latest Version • 5 months ago • Options • Reply 1

Great work as always. Thanks for sharing @janiobachmann . There is a lot to learn from this kernel.

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 1

Thanks Karan! Hopefully in the future I'll provide some more interesting topics.

Akhilesh • Posted on Latest Version • 5 months ago • Options • Reply 1

Great Work! Upvoted!!!

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Thanks! Greatly appreciate it!

john okemu • Posted on Latest Version • 5 months ago • Options • Reply 1

You are awesome, very informative I am a newbie,but I have learned a lot, keep the spirit going @janiobachmann

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Thanks John! Welcome to the community you will learn a lot!

GSD • Posted on Latest Version • 6 months ago • Options • Reply 1

WoW ..Again coming back to this kernel after your update popped up in my newsfeed and I cant return without leaving a comment .Amazing work @janiobachmann ..I have learnt how to present our findings in a logical and neat report format from this kernel ..Thank you for sharing your work to the community ..

Janio Martinez Kernel Author • Posted on Latest Version • 6 months ago • Options • Reply 0

Thanks for the kind words GSD! Looking forward to come up with more interesting projects. Have a great day!

franck • Posted on Latest Version • 6 months ago • Options • Reply 1

It's very impressive ! Brilliant. I ' am looking forward to see your next kernel

Janio Martinez Kernel Author • Posted on Latest Version • 6 months ago • Options • Reply 0

Thanks John! Really appreciate it.

Sachin Kaushik • Posted on Version 78 of 81 • 8 months ago • Options • Reply 1

Hi Martinez, the code for building Classifiers is not present. Can You please update the notebook to include that. Thanks a lot.

Janio Martinez Kernel Author • Posted on Version 78 of 81 • 8 months ago • Options • Reply 0

Yeah sorry Sachin, I am still in the process of updating this kernel into a newer version. Thanks for pointing it out.

Janio Martinez Kernel Author • Posted on Version 79 of 81 • 8 months ago • Options • Reply 0

I added the classification phase, although I am aiming to bring further updates soon. Have a great day!

Sachin Kaushik • Posted on Version 79 of 81 • 8 months ago • Options • Reply 0

Yeah sure. Thanks a Lot.

Steven Rolka • Posted on Version 78 of 81 • 10 months ago • Options • Reply 1

I've looked through a bunch of your kernals and I've learned so much from each one. Thanks!

Janio Martinez Kernel Author • Posted on Version 78 of 81 • 10 months ago • Options • Reply 1

Thanks Steven! Currently, working in an R kernel. New to the programming language but hopefully I can provide to the Kaggle community a well elaborated project. Thanks again for your comment! Helps me keep up motivated!

Danny • Posted on Version 74 of 81 • a year ago • Options • Reply 1

Great Visualizations! Definitely, lots to learn from this kernel!

Janio Martinez Kernel Author • Posted on Version 74 of 81 • a year ago • Options • Reply 0

Thank you Danny! Appreciate it!

Serkan Peldek · Posted on Version 74 of 81 · a year ago · Options · Reply 1

Impressive and clear write up kernel. Thanks for sharing.

Janio Martinez Kernel Author · Posted on Version 74 of 81 · a year ago · Options · Reply 0

Thanks Serkan! Appreciate it!

Erik Bruin · Posted on Version 72 of 81 · a year ago · Options · Reply 1

You are very close to becoming kernel master! Hope that you make it with this kernel!

Janio Martinez Kernel Author · Posted on Version 72 of 81 · a year ago · Options · Reply 1

Haha thanks Erik Bruin! A lot of my kernels are based on your good kernels hope I can keep learning more about your work!

Erik Bruin · Posted on Version 74 of 81 · a year ago · Options · Reply 1

Haha! Yours are very good too! And you will definitely beat me when it comes down who becomes a Master first ;-)

Janio Martinez Kernel Author · Posted on Version 74 of 81 · a year ago · Options · Reply 1

Well my goal next time will be to make R kernels as impressive as you do, your kernels were an inspiration for me to start learning R, Thanks for the good quality kernels you share!

Erik Bruin · Posted on Version 78 of 81 · a year ago · Options · Reply 1

Four red dots! You made it, congrats!

Bank Marketing Campaign || Opening a Term Deposit

Python notebook using data from Bank Marketing Dataset · 24,349 views · 7mo ago · beginner, data visualization, classification, +1 more

134 Copy and Edit 317

Version 81
81 commits

forked from [F

Thanks Erik! You will defninitely get the four dots in a matter of no time. Your kernels are always impressive.

el amraoui Sohayb · Posted on Version 71 of 81 · a year ago · Options · Reply 1

Wow, Great Visuals and very detailed. Nice work @janiobachmann

Janio Martinez Kernel Author · Posted on Version 71 of 81 · a year ago · Options · Reply 0

Thanks!

MJ Bahmani • Posted on Version 68 of 81 • a year ago • Options • Reply 1

Excellent work.

Janio Martinez Kernel Author • Posted on Version 68 of 81 • a year ago • Options • Reply 0

Thanks Mj! Still looking forward to share deeper insights with the community.

Pavan Sanagapati • Posted on Version 66 of 81 • a year ago • Options • Reply 1

Excellent kernel! You have got tremendous financial knowledge. Thanks for sharing. Up voted

Janio Martinez Kernel Author • Posted on Version 66 of 81 • a year ago • Options • Reply 0

Thanks Pavan! Going to look into your new kernel later throughout the day. Still have things to update for this kernel.

dust • Posted on Version 65 of 81 • a year ago • Options • Reply 1

Thanks for sharing this amazing work!

Janio Martinez Kernel Author • Posted on Version 65 of 81 • a year ago • Options • Reply 0

Thanks Dust! This was my second kernel so I will come up with more interesting updates. I think there is a lot more into this dataset to explore.

Notebook Data Comments

Dilip Kumar • Posted on Version 69 of 81 • a year ago • Options • Reply 2

excellent work thank you for sharing

Janio Martinez Kernel Author • Posted on Version 69 of 81 • a year ago • Options • Reply 0

Thanks Dilip! Looking forward in sharing more interesting kernels with the community.

Leonardo Ferreira • Posted on Version 63 of 81 • a year ago • Options • Reply 1

Hey bro, amazing work!! Very clear and advanced plots! I also learn a lot with your kernels, thank you for share with us!

Janio Martinez Kernel Author • Posted on Version 63 of 81 • a year ago • Options • Reply 0

Thanks Leonardo! Actually this was my second kernel! I'm pretty sure it needs further updates!

GSD • Posted on Version 60 of 81 • 2 years ago • Options • Reply 1

Excellent visualizations and summary ...Thanks ...

Janio Martinez Kernel Author • Posted on Version 60 of 81 • 2 years ago • Options • Reply 0

Thank you Kumar for your comment! I really appreciated!

xachi • Posted on Version 60 of 81 • 2 years ago • Options • Reply 1

Thank you for sharing your work.

Janio Martinez Kernel Author • Posted on Version 60 of 81 • 2 years ago • Options • Reply 0

You are welcome! It's a bit long since I am always eager in implementing an extensive data analysis. and then getting into the modeling phase thats why this post seems a bit long. Thanks for the review!

Elwin Vasquez • Posted on Version 39 of 81 • 2 years ago • Options • Reply 1

Thank you for sharing this Data analyst Janio, it has given me a clear idea what are the real life situation in which this could be use..!

Janio Martinez Kernel Author • Posted on Version 39 of 81 • 2 years ago • Options • Reply 0

I am glad you liked it! I am still working on a classification model which I should upload pretty soon. I will also be posting in the future further insights that we could apply to this example.

Vishvajit Phalke • Posted on Latest Version • a month ago • Options • Reply 0

Hello Janio., I have a lot of doubts regarding this code. So can you help me to know this code.

Binota Naidu Roy • Posted on Latest Version • 3 months ago • Options • Reply 0

Awesome work .. Janio! Upvoted

Kostas Voul • Posted on Latest Version • 5 months ago • Options • Reply 0

Wonderful kernel (as always), Janio. Well done. Keep up the good work!

Just one thing I couldn't figure out why you do it:
Why use this:

lst = [df]
for col in df:
col.loc[] = ...

Instead of the more straightforward:

df.loc[] = ...

Thanks in advance.

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

`df["balance_status"] = np.nan
lst = [df]

for col in lst:
col.loc[col["balance"] < 0, "balance*status"] = "negative" col.loc[(col["balance"] >= 0) & (col["balance"] status"] = "low" col.loc[(col["balance"] > 30000) & (col["balance"] <= 40000), "balance*status"] = "middle" col.loc[col["balance"] > 40000, "balancestatus"] = "high"`

Hey Kostas, at least in this case I had to do an iteration to go through all the cells and meet the conditions from the other cells. I normally do this through iteration however, let me check if it is possible filling Nan values through the way you have provided. Lately, I have been using more R for my projects so I have to run this code and see if it is possible doing the same task in the same form that you have provided.

Kostas Voul • Posted on Latest Version • 5 months ago • Options • Reply 1

Don't get into that trouble, I have forked your kernel and tried to run in order to check if everything runs smoothly. It appears it does.
Here is my (your actually) kernel, where I tried to produce the same outputs in a different (wherever possible) way. Take a look whenever you have the time and feel free to adopt in case you like anything. Every feedback is more than welcome.

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 1

I'll give it a look throughout the day and see if I could adopt it to this kernel. Have a great day!

Johnny • Posted on Latest Version • 5 months ago • Options • Reply 0

hello @janiobachmann first I would like to thank you for this. It really helped me have a better understanding! Great job:)
However, I have a small clarification/question, when you built the decision tree classifier (to determine the importance of the feature) you mentioned that contact is the second most important feature. You mentioned a well that contact refers to the number of times a person was contacted. However, according to the description, contact refers to the type of communication (if cellular, telephone or other). Knowing this the conclusions might change. Perhaps it could be better to contact customers on their cellular.

Janio Martinez Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Thanks for the clarification, I would look into it later throughout the day.

a year ago

This Comment was deleted.

Showing 50 of 55 comments. Show More