Bank

February 19, 2020

0.1 Marketing:

The process by which companies create value for customers and build strong customer relationshi

0.2 Term Deposit:

A Term deposit is a deposit that a bank or a financial institurion offers with a fixed rate (o

0.3 Outline:

1. Attribute Description
2. Structuring the Data
3. Exploratory Data Analysis
4. Data Visualizations
5. Correlation that impacted the decision of Potential Clients
6. Classification Models
7. Next Campaign Strategy

0.4 1. Attribute Description

0.4.1 1.1 Input Variables (Bank Client Data):

1 - age: (numeric)
2 - job: type of job (categorical: 'admin.','blue-   collar','entrepreneur','housemaid','ma
3 - marital: marital status (categorical: 'divorced','married','single','unknown'; note: 'divo
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.

0.4.2 1.2 Input Variables (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')
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 hig

0.4.3 1.3 Input Variables (Other):

12 - campaign: number of contacts performed during this campaign and for this client (numeric,

13 - pdays: number of days that passed by after the client was last contacted from a previous c
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

0.4.4 1.4 Output Variable:

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

[7]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

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

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

[7]:
```
   age         job  marital  education default  balance housing loan  contact  \
0   59      admin.  married  secondary      no     2343     yes   no  unknown
1   56      admin.  married  secondary      no       45      no   no  unknown
2   41  technician  married  secondary      no     1270     yes   no  unknown
3   55    services  married  secondary      no     2476     yes   no  unknown
4   54      admin.  married   tertiary      no      184      no   no  unknown

   day month  duration  campaign  pdays  previous poutcome deposit
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes
```

0.5 2. Structuring the Data

[8]:
```
df.describe()
```

[8]:
```
                age       balance           day      duration      campaign  \
count  11162.000000  11162.000000  11162.000000  11162.000000  11162.000000
mean      41.231948   1528.538524     15.658036    371.993818      2.508421
std       11.913369   3225.413326      8.420740    347.128386      2.722077
min       18.000000  -6847.000000      1.000000      2.000000      1.000000
25%       32.000000    122.000000      8.000000    138.000000      1.000000
50%       39.000000    550.000000     15.000000    255.000000      2.000000
75%       49.000000   1708.000000     22.000000    496.000000      3.000000
max       95.000000  81204.000000     31.000000   3881.000000     63.000000

              pdays      previous
count  11162.000000  11162.000000
mean      51.330407      0.832557
std      108.758282      2.292007
min       -1.000000      0.000000
25%       -1.000000      0.000000
50%       -1.000000      0.000000
75%       20.750000      1.000000
max      854.000000     58.000000
```

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.

0.6 3. Exploratory Data Analysis

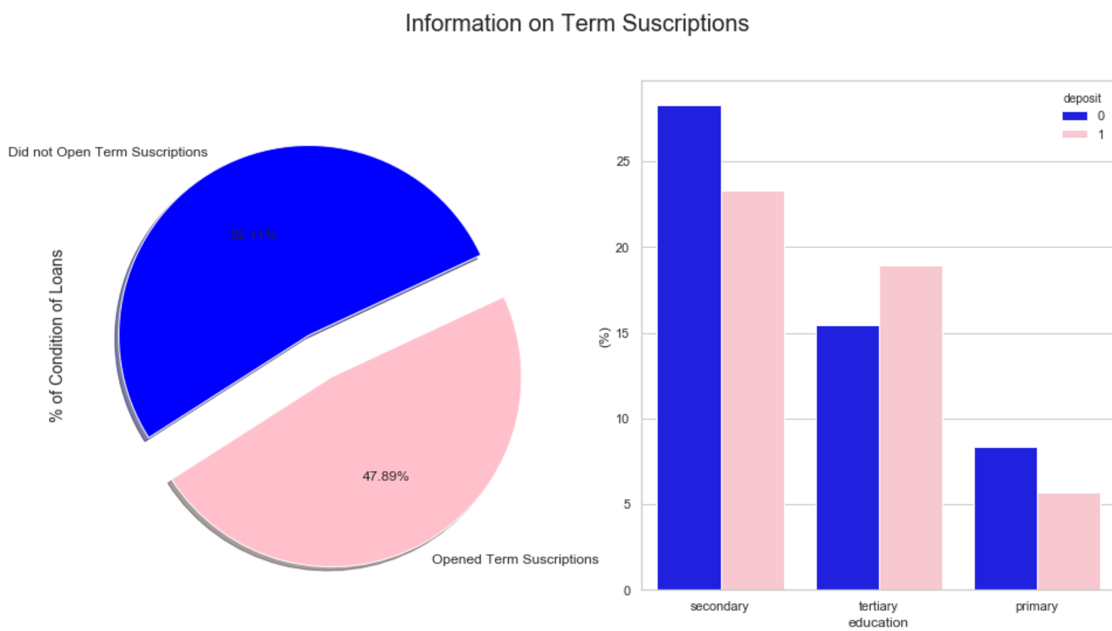
[66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9892 entries, 0 to 11159
Data columns (total 18 columns):
age                  9892 non-null int64
job                  9892 non-null object
marital              9892 non-null object
education            9892 non-null object
default              9892 non-null object
```

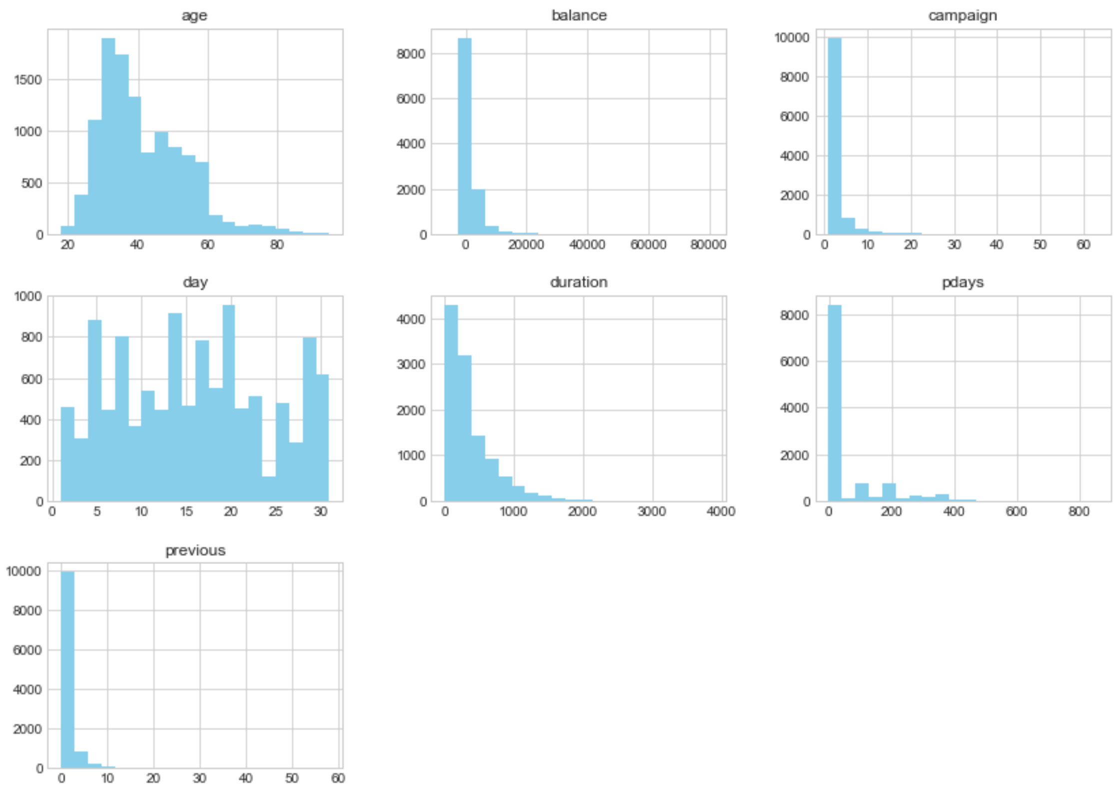
```
balance            9892 non-null int64
housing            9892 non-null object
loan               9892 non-null object
contact            9892 non-null object
day                9892 non-null int64
month              9892 non-null object
duration           9892 non-null int64
campaign           9892 non-null int64
pdays              9892 non-null int64
previous           9892 non-null int64
poutcome           9892 non-null object
deposit            9892 non-null int64
duration_status    9892 non-null object
dtypes: int64(8), object(10)
memory usage: 1.7+ MB
```

No missing values

[67]:
```
f, ax = plt.subplots(1,2, figsize=(16,8))

colors = ["Blue", "Pink"]
labels ="Did not Open Term Suscriptions", "Opened Term Suscriptions"

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

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


# ax[0].set_title('State of Loan', fontsize=16)
ax[0].set_ylabel('% of Condition of Loans', fontsize=14)

# sns.countplot('loan_condition', data=df, ax=ax[1], palette=colors)
# ax[1].set_title('Condition of Loans', fontsize=20)
# ax[1].set_xticklabels(['Good', 'Bad'], rotation='horizontal')
palette = ["Blue", "Pink"]

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()
```




```
[18]: import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

df.hist(bins=20, figsize=(14,10), color='skyblue')
plt.show()
```



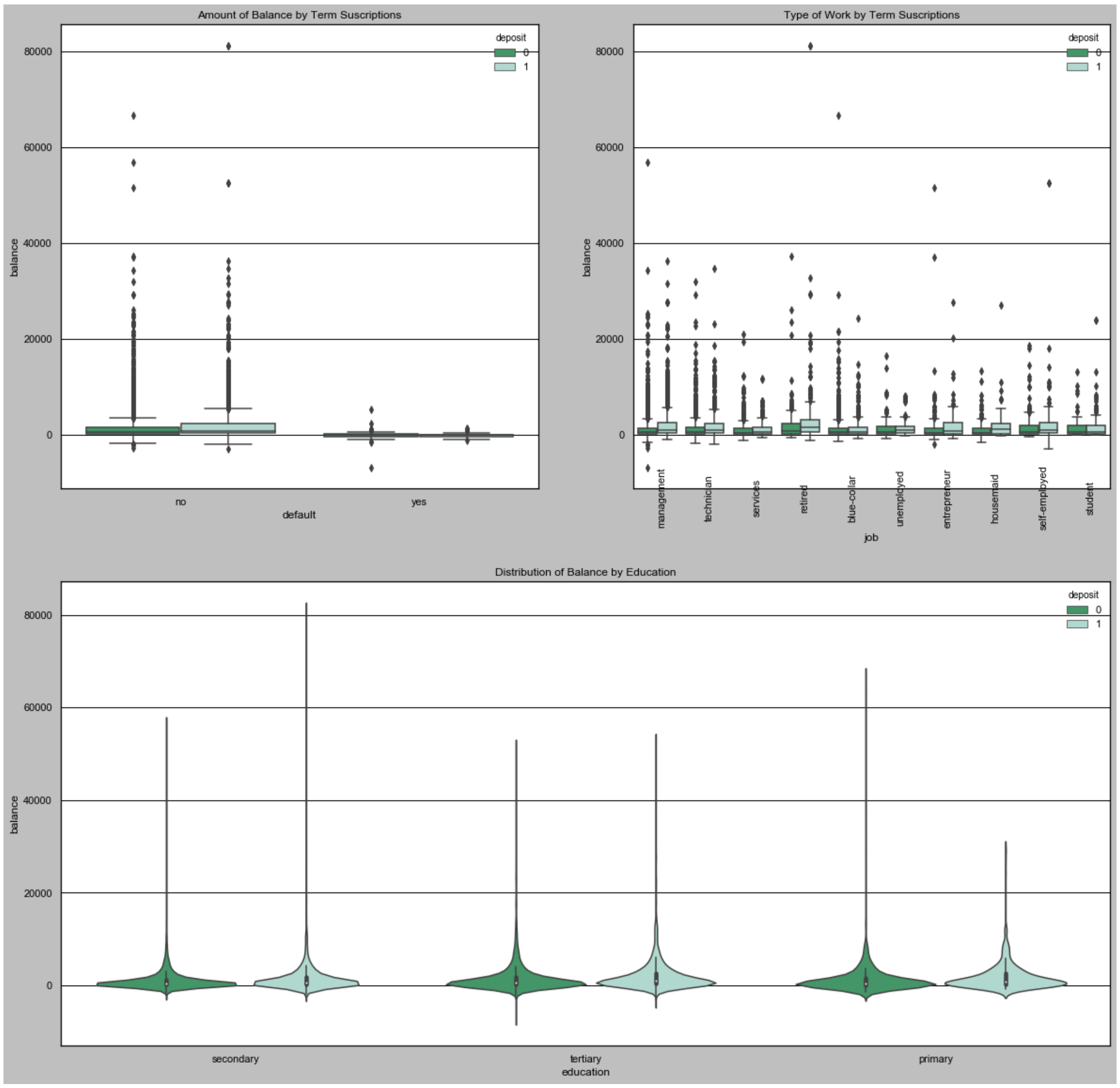
4. Data Visualizations by Analysis

[68]:
```
df['deposit'].value_counts()
```

[68]:
```
0    5155
1    4737
Name: deposit, dtype: int64
```

[69]:
```
plt.style.use('grayscale')
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="deposit",
                    data=df, palette="BuGn_r", ax=ax1)

g.set_title("Amount of Balance by Term Suscriptions")
g1 = sns.boxplot(x="job", y="balance", hue="deposit",
                 data=df, palette="BuGn_r", ax=ax2)

g1.set_xticklabels(df["job"].unique(), rotation=90, rotation_mode="anchor")
```

```
g1.set_title("Type of Work by Term Suscriptions")

g2 = sns.violinplot(data=df, x="education", y="balance", hue="deposit",␣
 ↪palette="BuGn_r")

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


plt.show()
```



[70]:
```
df.head()
```

```
[70]:    age          job  marital  education default  balance housing loan  contact  \
      0   59   management  married  secondary      no     2343     yes   no  unknown
      1   56   management  married  secondary      no       45      no   no  unknown
      2   41   technician  married  secondary      no     1270     yes   no  unknown
      3   55     services  married  secondary      no     2476     yes   no  unknown
      4   54   management  married   tertiary      no      184      no   no  unknown

         day month  duration  campaign  pdays  previous poutcome  deposit  \
      0    5   may      1042         1     -1         0  unknown        1
      1    5   may      1467         1     -1         0  unknown        1
      2    5   may      1389         1     -1         0  unknown        1
      3    5   may       579         1     -1         0  unknown        1
      4    5   may       673         2     -1         0  unknown        1

        duration_status
      0   above_average
      1   above_average
      2   above_average
      3   above_average
      4   above_average
```

0.6.1 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.

[71]:
```
# Drop the Job Occupations that are "Unknown"
df = df.drop(df.loc[df["job"] == "unknown"].index)

# Admin and management are basically the same let's put it under the same␣
↪categorical value
lst = [df]

for col in lst:
    col.loc[col["job"] == "admin.", "job"] = "management"
```

[72]:
```
df.columns
```

```
[72]: Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
             'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
             'previous', 'poutcome', 'deposit', 'duration_status'],
            dtype='object')
```

[73]:

```python
import squarify
df = df.drop(df.loc[df["balance"] == 0].index)


x = 0
y = 0
width = 100
height = 100

job_names = df['job'].value_counts().index
values = df['job'].value_counts().tolist()

normed = squarify.normalize_sizes(values, width, height)
rects = squarify.squarify(normed, x, y, width, height)

colors = ['rgb(200, 255, 144)','rgb(135, 206, 235)',
          'rgb(235, 164, 135)','rgb(220, 208, 255)',
          'rgb(253, 253, 150)','rgb(255, 127, 80)',
         'rgb(218, 156, 133)', 'rgb(245, 92, 76)',
         'rgb(252,64,68)', 'rgb(154,123,91)']

shapes = []
annotations = []
counter = 0

for r in rects:
    shapes.append(
        dict(
            type = 'rect',
            x0 = r['x'],
            y0 = r['y'],
            x1 = r['x'] + r['dx'],
            y1 = r['y'] + r['dy'],
            line = dict(width=2),
            fillcolor = colors[counter]
        )
    )
    annotations.append(
        dict(
            x = r['x']+(r['dx']/2),
            y = r['y']+(r['dy']/2),
            text = values[counter],
            showarrow = False
        )
    )
    counter = counter + 1
    if counter >= len(colors):
```

```
        counter = 0

# For hover text
trace0 = go.Scatter(
    x = [ r['x']+(r['dx']/2) for r in rects],
    y = [ r['y']+(r['dy']/2) for r in rects],
    text = [ str(v) for v in job_names],
    mode='text',
)

layout = dict(
    title='Number of Occupations <br> <i>(From our Sample Population)</i>',
    height=700,
    width=700,
    xaxis=dict(showgrid=False,zeroline=False),
    yaxis=dict(showgrid=False,zeroline=False),
    shapes=shapes,
    annotations=annotations,
    hovermode='closest'
)

# With hovertext
figure = dict(data=[trace0], layout=layout)

iplot(figure, filename='squarify-treemap')
```

[74]:

```
# Now let's see which occupation tended to have more balance in their accounts

suscribed_df = df.loc[df["deposit"] == "yes"]

occupations = df["job"].unique().tolist()

# Get the balances by jobs
management = suscribed_df["age"].loc[suscribed_df["job"] == "management"].values
technician = suscribed_df["age"].loc[suscribed_df["job"] == "technician"].values
services = suscribed_df["age"].loc[suscribed_df["job"] == "services"].values
retired = suscribed_df["age"].loc[suscribed_df["job"] == "retired"].values
blue_collar = suscribed_df["age"].loc[suscribed_df["job"] == "blue-collar"].
 ↪values
unemployed = suscribed_df["age"].loc[suscribed_df["job"] == "unemployed"].values
entrepreneur = suscribed_df["age"].loc[suscribed_df["job"] == "entrepreneur"].
 ↪values
housemaid = suscribed_df["age"].loc[suscribed_df["job"] == "housemaid"].values
self_employed = suscribed_df["age"].loc[suscribed_df["job"] == "self-employed"].
 ↪values
student = suscribed_df["age"].loc[suscribed_df["job"] == "student"].values
```

```python
ages = [management, technician, services, retired, blue_collar, unemployed,
         entrepreneur, housemaid, self_employed, student]

colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)',
          'rgba(44, 160, 101, 0.5)', 'rgba(255, 65, 54, 0.5)',
          'rgba(207, 114, 255, 0.5)', 'rgba(127, 96, 0, 0.5)',
         'rgba(229, 126, 56, 0.5)', 'rgba(229, 56, 56, 0.5)',
         'rgba(174, 229, 56, 0.5)', 'rgba(229, 56, 56, 0.5)']

traces = []

for xd, yd, cls in zip(occupations, ages, colors):
        traces.append(go.Box(
            y=yd,
            name=xd,
            boxpoints='all',
            jitter=0.5,
            whiskerwidth=0.2,
            fillcolor=cls,
            marker=dict(
                size=2,
            ),
            line=dict(width=1),
        ))

layout = go.Layout(
    title='Distribution of Ages by Occupation',
    yaxis=dict(
        autorange=True,
        showgrid=True,
        zeroline=True,
        dtick=5,
        gridcolor='rgb(255, 255, 255)',
        gridwidth=1,
        zerolinecolor='rgb(255, 255, 255)',
        zerolinewidth=2,
    ),
    margin=dict(
        l=40,
        r=30,
        b=80,
        t=100,
    ),
    paper_bgcolor='rgb(224,255,246)',
    plot_bgcolor='rgb(251,251,251)',
    showlegend=False
```

```
)

fig = go.Figure(data=traces, layout=layout)
iplot(fig)
```

```
/opt/anaconda3/lib/python3.7/site-packages/pandas/core/ops/__init__.py:1115:
FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison
```

[75]:
```
# Balance Distribution

# Create a Balance Category
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"] <= 30000),␣
 ↪"balance_status"] = "low"
    col.loc[(col["balance"] > 30000) & (col["balance"] <= 40000),␣
 ↪"balance_status"] = "middle"
    col.loc[col["balance"] > 40000, "balance_status"] = "high"

# balance by balance_status
negative = df["balance"].loc[df["balance_status"] == "negative"].values.tolist()
low = df["balance"].loc[df["balance_status"] == "low"].values.tolist()
middle = df["balance"].loc[df["balance_status"] == "middle"].values.tolist()
high = df["balance"].loc[df["balance_status"] == "high"].values.tolist()


# Get the average by occupation in each balance category
job_balance = df.groupby(['job', 'balance_status'])['balance'].mean()


trace1 = go.Barpolar(
    r=[-199.0, -392.0, -209.0, -247.0, -233.0, -270.0, -271.0, 0, -276.0, -134.
 ↪5],
    text=["blue-collar", "entrepreneur", "housemaid", "management", "retired",␣
 ↪"self-employed",
          "services", "student", "technician", "unemployed"],
    name='Negative Balance',
    marker=dict(
        color='rgb(246, 46, 46)'
    )
```

```
)
trace2 = go.Barpolar(
    r=[319.5, 283.0, 212.0, 313.0, 409.0, 274.5, 308.5, 253.0, 316.0, 330.0],
    text=["blue-collar", "entrepreneur", "housemaid", "management", "retired",␣
 ↪"self-employed",
         "services", "student", "technician", "unemployed"],
    name='Low Balance',
    marker=dict(
        color='rgb(246, 97, 46)'
    )
)
trace3 = go.Barpolar(
    r=[2128.5, 2686.0, 2290.0, 2366.0, 2579.0, 2293.5, 2005.5, 2488.0, 2362.0,␣
 ↪1976.0],
    text=["blue-collar", "entrepreneur", "housemaid", "management", "retired",␣
 ↪"self-employed",
         "services", "student", "technician", "unemployed"],
    name='Middle Balance',
    marker=dict(
        color='rgb(246, 179, 46)'
    )
)
trace4 = go.Barpolar(
    r=[14247.5, 20138.5, 12278.5, 12956.0, 20723.0, 12159.0, 12223.0, 13107.0,␣
 ↪12063.0, 15107.5],
    text=["blue-collar", "entrepreneur", "housemaid", "management", "retired",␣
 ↪"self-employed",
         "services", "student", "technician", "unemployed"],
    name='High Balance',
    marker=dict(
        color='rgb(46, 246, 78)'
    )
)
data = [trace1, trace2, trace3, trace4]
layout = go.Layout(
    title='Mean Balance in Account<br> <i> by Job Occupation</i>',
    font=dict(
        size=12
    ),
    legend=dict(
        font=dict(
            size=16
        )
    ),
    radialaxis=dict(
        ticksuffix='%'
    ),
```

```
    orientation=270
)
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='polar-area-chart')
```

0.6.2 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.

[76]:
```
df['marital'].value_counts()
```

[76]:
```
married     5571
single      3192
divorced    1129
Name: marital, dtype: int64
```

[77]:
```
df['marital'].unique()
```

[77]:
```
array(['married', 'divorced', 'single'], dtype=object)
```

[78]:
```
df['marital'].value_counts().tolist()
```

[78]:
```
[5571, 3192, 1129]
```

[37]:
```
vals = df['marital'].value_counts().tolist()
labels = ['married', 'divorced', 'single']

data = [go.Bar(
            x=labels,
            y=vals,
    marker=dict(
    color="skyblue")
    )]

layout = go.Layout(
    title="Count by Marital Status",
)

fig = go.Figure(data=data, layout=layout)



iplot(fig, filename='basic-bar')
```

[39]:

```python
# Distribution of Balances by Marital status
single = df['balance'].loc[df['marital'] == 'single'].values
married = df['balance'].loc[df['marital'] == 'married'].values
divorced = df['balance'].loc[df['marital'] == 'divorced'].values


single_dist = go.Histogram(
    x=single,
    histnorm='density',
    name='single',
    marker=dict(
        color='skyblue'
    )
)


married_dist = go.Histogram(
    x=married,
    histnorm='density',
    name='married',
    marker=dict(
        color='pink'
    )
)

divorced_dist = go.Histogram(
    x=divorced,
    histnorm='density',
    name='divorced',
    marker=dict(
        color='yellow'
    )
)


fig = tools.make_subplots(rows=3, print_grid=False)

fig.append_trace(single_dist, 1, 1)
fig.append_trace(married_dist, 2, 1)
fig.append_trace(divorced_dist, 3, 1)


fig['layout'].update(showlegend=False, title="Price Distributions by Marital␣
↪Status",
                    height=1000, width=800)

iplot(fig, filename='custom-sized-subplot-with-subplot-titles')
```

```
[40]: df.head()
```

```
[40]:    age          job  marital  education default  balance housing loan  contact  \
      0   59   management  married  secondary      no     2343     yes   no  unknown
      1   56   management  married  secondary      no       45      no   no  unknown
      2   41   technician  married  secondary      no     1270     yes   no  unknown
      3   55     services  married  secondary      no     2476     yes   no  unknown
      4   54   management  married   tertiary      no      184      no   no  unknown

         day month  duration  campaign  pdays  previous poutcome deposit  \
      0    5   may      1042         1     -1         0  unknown     yes
      1    5   may      1467         1     -1         0  unknown     yes
      2    5   may      1389         1     -1         0  unknown     yes
      3    5   may       579         1     -1         0  unknown     yes
      4    5   may       673         2     -1         0  unknown     yes

        balance_status
      0            low
      1            low
      2            low
      3            low
      4            low
```

```
[41]: # Notice how divorced have a considerably low amount of balance.
fig = ff.create_facet_grid(
    df,
    x='duration',
    y='balance',
    color_name='marital',
    show_boxes=False,
    marker={'size': 10, 'opacity': 1.0},
    colormap={'single': 'rgb(165, 242, 242)', 'married': 'rgb(253, 174, 216)',␣
↪'divorced': 'rgba(201, 109, 59, 0.82)'}
)

iplot(fig, filename='facet - custom colormap')
```

```
[45]: fig = ff.create_facet_grid(
    df,
    y='balance',
    facet_row='marital',
    facet_col='deposit',
    trace_type='box',
)

iplot(fig, filename='facet - box traces')
```

[46]:
```
df.head()
```

[46]:
```
   age         job  marital  education default  balance housing loan  contact  \
0   59  management  married  secondary      no     2343     yes   no  unknown
1   56  management  married  secondary      no       45      no   no  unknown
2   41  technician  married  secondary      no     1270     yes   no  unknown
3   55    services  married  secondary      no     2476     yes   no  unknown
4   54  management  married   tertiary      no      184      no   no  unknown

   day month  duration  campaign  pdays  previous poutcome deposit  \
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes

  balance_status
0            low
1            low
2            low
3            low
4            low
```

0.6.3 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.

[48]:
```
df = df.drop(df.loc[df["education"] == "unknown"].index)
df['education'].unique()
```

[48]:
```
array(['secondary', 'tertiary', 'primary'], dtype=object)
```

[49]:
```
df['marital/education'] = np.nan
lst = [df]

for col in lst:
    col.loc[(col['marital'] == 'single') & (df['education'] == 'primary'),␣
 ↪'marital/education'] = 'single/primary'
    col.loc[(col['marital'] == 'married') & (df['education'] == 'primary'),␣
 ↪'marital/education'] = 'married/primary'
```

```
    col.loc[(col['marital'] == 'divorced') & (df['education'] == 'primary'),␣
 ↪'marital/education'] = 'divorced/primary'
    col.loc[(col['marital'] == 'single') & (df['education'] == 'secondary'),␣
 ↪'marital/education'] = 'single/secondary'
    col.loc[(col['marital'] == 'married') & (df['education'] == 'secondary'),␣
 ↪'marital/education'] = 'married/secondary'
    col.loc[(col['marital'] == 'divorced') & (df['education'] == 'secondary'),␣
 ↪'marital/education'] = 'divorced/secondary'
    col.loc[(col['marital'] == 'single') & (df['education'] == 'tertiary'),␣
 ↪'marital/education'] = 'single/tertiary'
    col.loc[(col['marital'] == 'married') & (df['education'] == 'tertiary'),␣
 ↪'marital/education'] = 'married/tertiary'
    col.loc[(col['marital'] == 'divorced') & (df['education'] == 'tertiary'),␣
 ↪'marital/education'] = 'divorced/tertiary'



df.head()
```

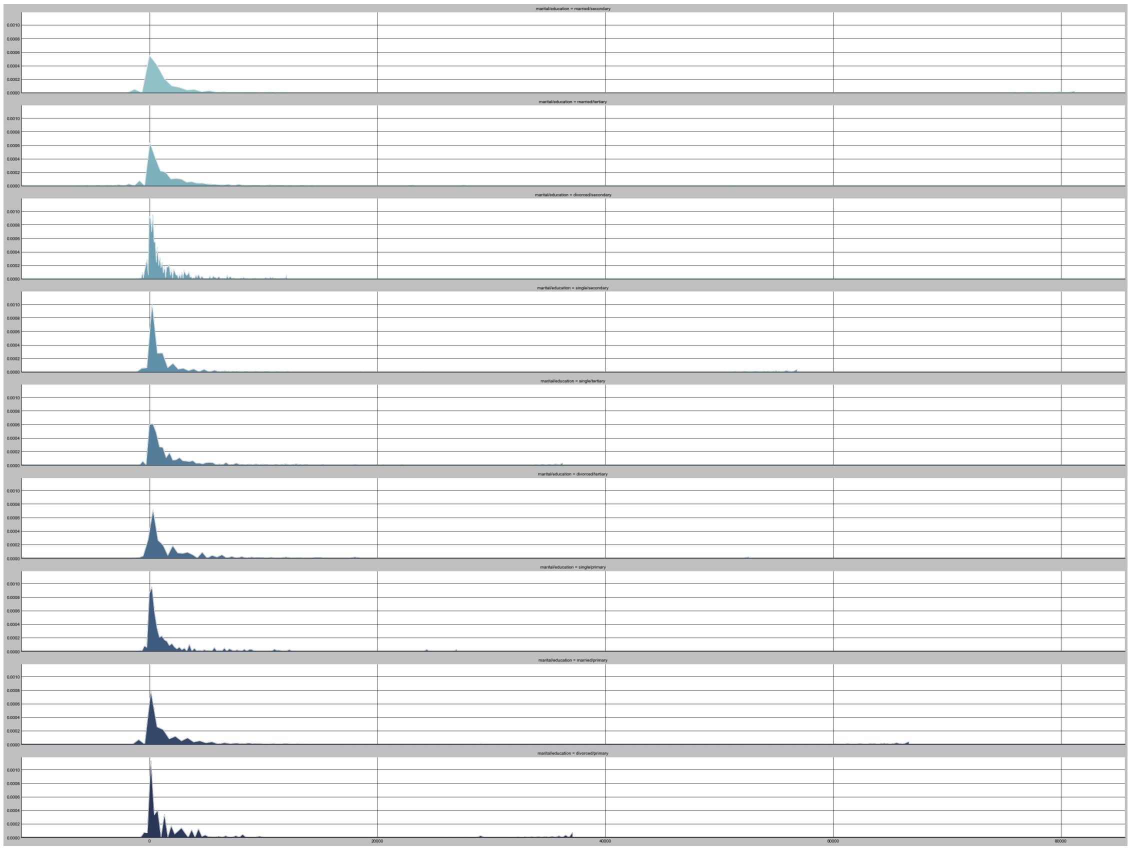
[49]:

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

   day month  duration  campaign  pdays  previous poutcome deposit  \
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes

  balance_status  marital/education
0            low  married/secondary
1            low  married/secondary
2            low  married/secondary
3            low  married/secondary
4            low   married/tertiary
```

[50]:

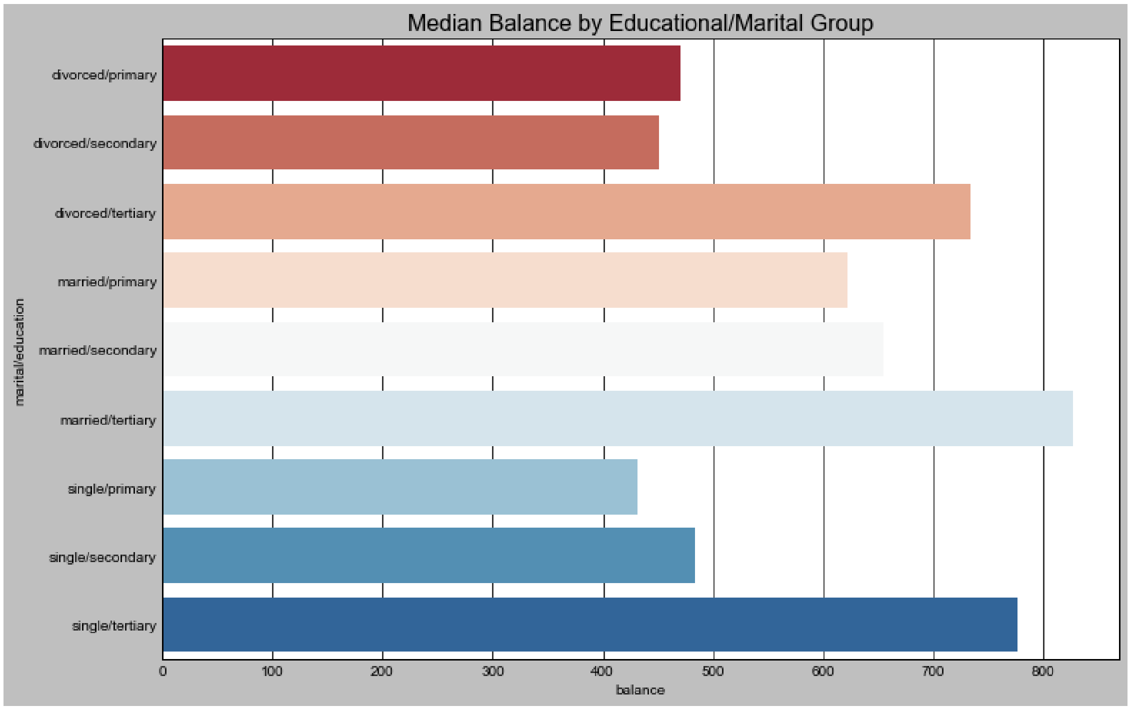
```
pal = sns.cubehelix_palette(10, rot=-.25, light=.7)
g = sns.FacetGrid(df, row="marital/education", hue="marital/education",␣
 ↪aspect=12, palette=pal)

g.map(sns.kdeplot, "balance", clip_on=False, shade=True, alpha=1, lw=1.5, bw=.2)
g.map(sns.kdeplot, "balance", clip_on=False, color="w", lw=1, bw=0)
g.map(plt.axhline, y=0, lw=2, clip_on=False)
```

[50]: <seaborn.axisgrid.FacetGrid at 0x1c224f8f90>



[51]:
```
education_groups = df.groupby(['marital/education'], as_index=False)['balance'].
↪median()

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




sns.barplot(x="balance", y="marital/education", data=education_groups,
            label="Total", palette="RdBu")

plt.title('Median Balance by Educational/Marital Group', fontsize=16)
```

[51]: Text(0.5, 1.0, 'Median Balance by Educational/Marital Group')



[52]:
```python
# Let's see the group who had loans from the marital/education group

loan_balance = df.groupby(['marital/education', 'loan'],␣
↪as_index=False)['balance'].median()


no_loan = loan_balance['balance'].loc[loan_balance['loan'] == 'no'].values
has_loan = loan_balance['balance'].loc[loan_balance['loan'] == 'yes'].values


labels = loan_balance['marital/education'].unique().tolist()


trace0 = go.Scatter(
    x=no_loan,
    y=labels,
    mode='markers',
    name='No Loan',
    marker=dict(
        color='rgb(175,238,238)',
        line=dict(
            color='rgb(0,139,139)',
            width=1,
        ),
```

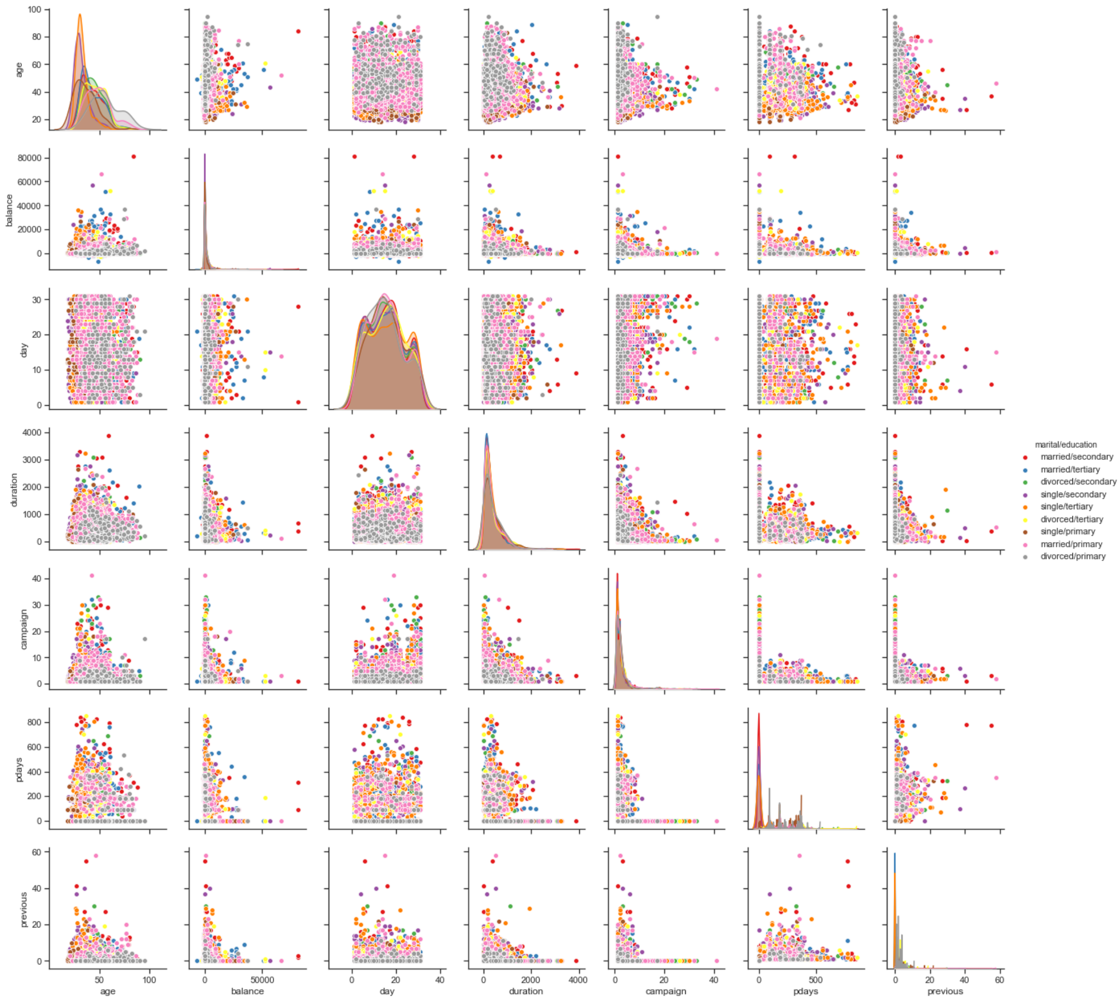
```
        symbol='circle',
        size=16,
    )
)
trace1 = go.Scatter(
    x=has_loan,
    y=labels,
    mode='markers',
    name='Has a Previous Loan',
    marker=dict(
        color='rgb(250,128,114)',
        line=dict(
            color='rgb(178,34,34)',
            width=1,
        ),
        symbol='circle',
        size=16,
    )
)

data = [trace0, trace1]
layout = go.Layout(
    title="The Impact of Loans to Married/Educational Clusters",
    xaxis=dict(
        showgrid=False,
        showline=True,
        linecolor='rgb(102, 102, 102)',
        titlefont=dict(
            color='rgb(204, 204, 204)'
        ),
        tickfont=dict(
            color='rgb(102, 102, 102)',
        ),
        showticklabels=False,
        dtick=10,
        ticks='outside',
        tickcolor='rgb(102, 102, 102)',
    ),
    margin=dict(
        l=140,
        r=40,
        b=50,
        t=80
    ),
    legend=dict(
        font=dict(
            size=10,
```

```
        ),
        yanchor='middle',
        xanchor='right',
    ),
    width=1000,
    height=800,
    paper_bgcolor='rgb(255,250,250)',
    plot_bgcolor='rgb(255,255,255)',
    hovermode='closest',
)
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='lowest-oecd-votes-cast')
```

[53]:
```
df.head()
```

[53]:
```
   age         job  marital  education default  balance housing loan  contact  \
0   59  management  married  secondary      no     2343     yes   no  unknown
1   56  management  married  secondary      no       45      no   no  unknown
2   41  technician  married  secondary      no     1270     yes   no  unknown
3   55    services  married  secondary      no     2476     yes   no  unknown
4   54  management  married   tertiary      no      184      no   no  unknown

   day month  duration  campaign  pdays  previous poutcome deposit  \
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes

  balance_status  marital/education
0            low  married/secondary
1            low  married/secondary
2            low  married/secondary
3            low  married/secondary
4            low   married/tertiary
```

[54]:
```
import seaborn as sns
sns.set(style="ticks")

sns.pairplot(df, hue="marital/education", palette="Set1")
plt.show()
```

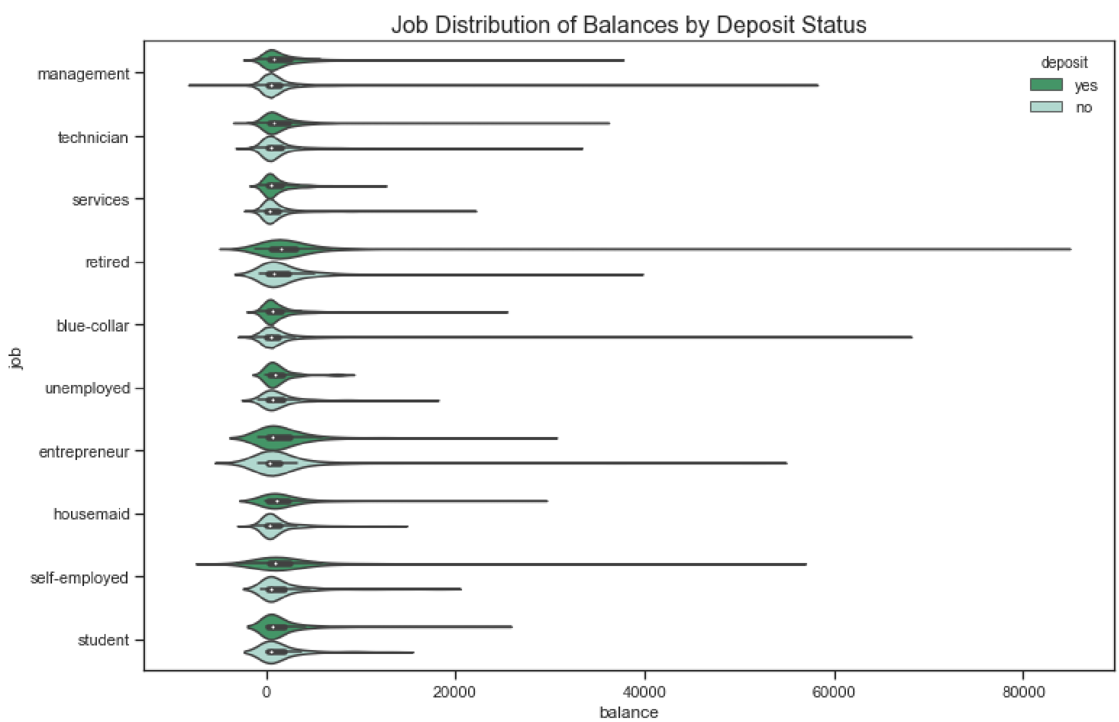


[55]:
```python
df.head()
```

[55]:
```
   age          job  marital  education default  balance housing loan  contact  \
0   59   management  married  secondary      no     2343     yes   no  unknown
1   56   management  married  secondary      no       45      no   no  unknown
2   41   technician  married  secondary      no     1270     yes   no  unknown
3   55     services  married  secondary      no     2476     yes   no  unknown
4   54   management  married   tertiary      no      184      no   no  unknown

   day month  duration  campaign  pdays  previous poutcome deposit  \
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes
```

```
  balance_status  marital/education
0            low  married/secondary
1            low  married/secondary
2            low  married/secondary
3            low  married/secondary
4            low   married/tertiary
```

[57]:
```python
fig = plt.figure(figsize=(12,8))

sns.violinplot(x="balance", y="job", hue="deposit", palette="BuGn_r",
            data=df);

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

plt.show()
```



0.6.4 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.

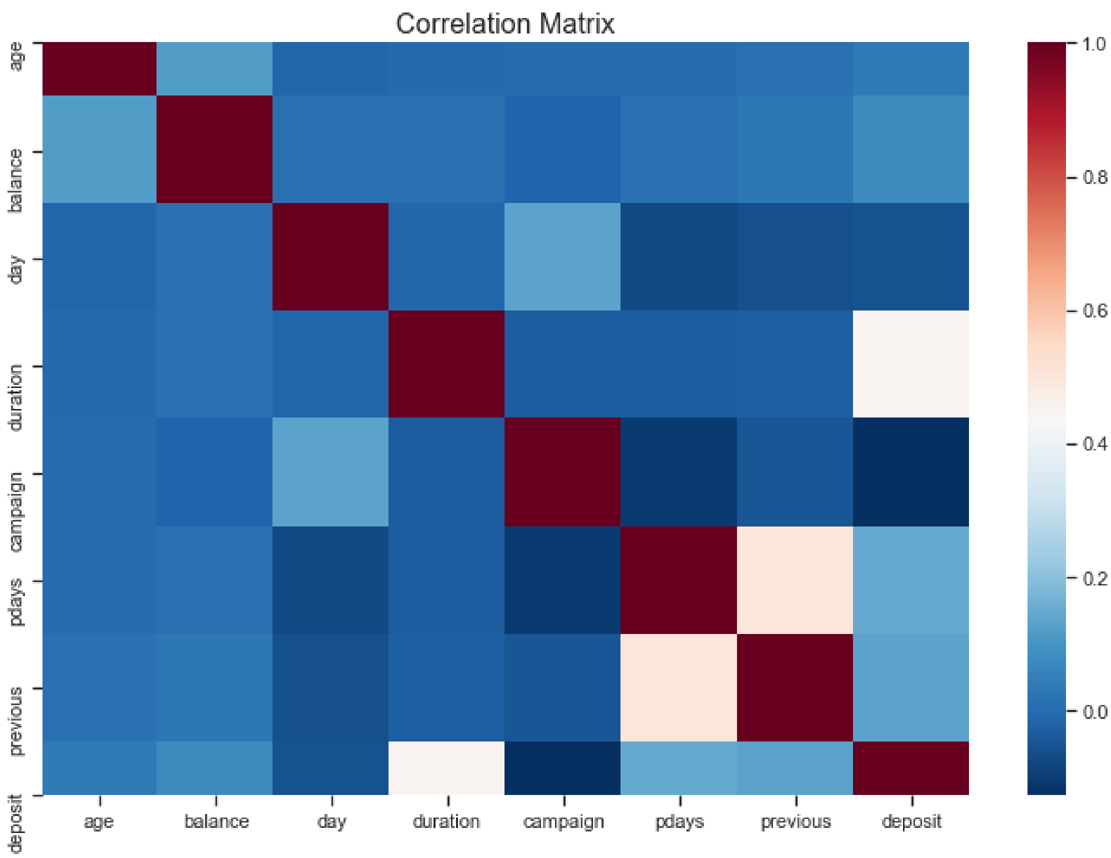
[59]:
```
df.drop(['marital/education', 'balance_status'], axis=1, inplace=True)
```

[60]:
```
df.head()
```

[60]:
```
   age         job  marital  education default  balance housing loan  contact  \
0   59  management  married  secondary      no     2343     yes   no  unknown
1   56  management  married  secondary      no       45      no   no  unknown
2   41  technician  married  secondary      no     1270     yes   no  unknown
3   55    services  married  secondary      no     2476     yes   no  unknown
4   54  management  married   tertiary      no      184      no   no  unknown

   day month  duration  campaign  pdays  previous poutcome deposit
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes
```

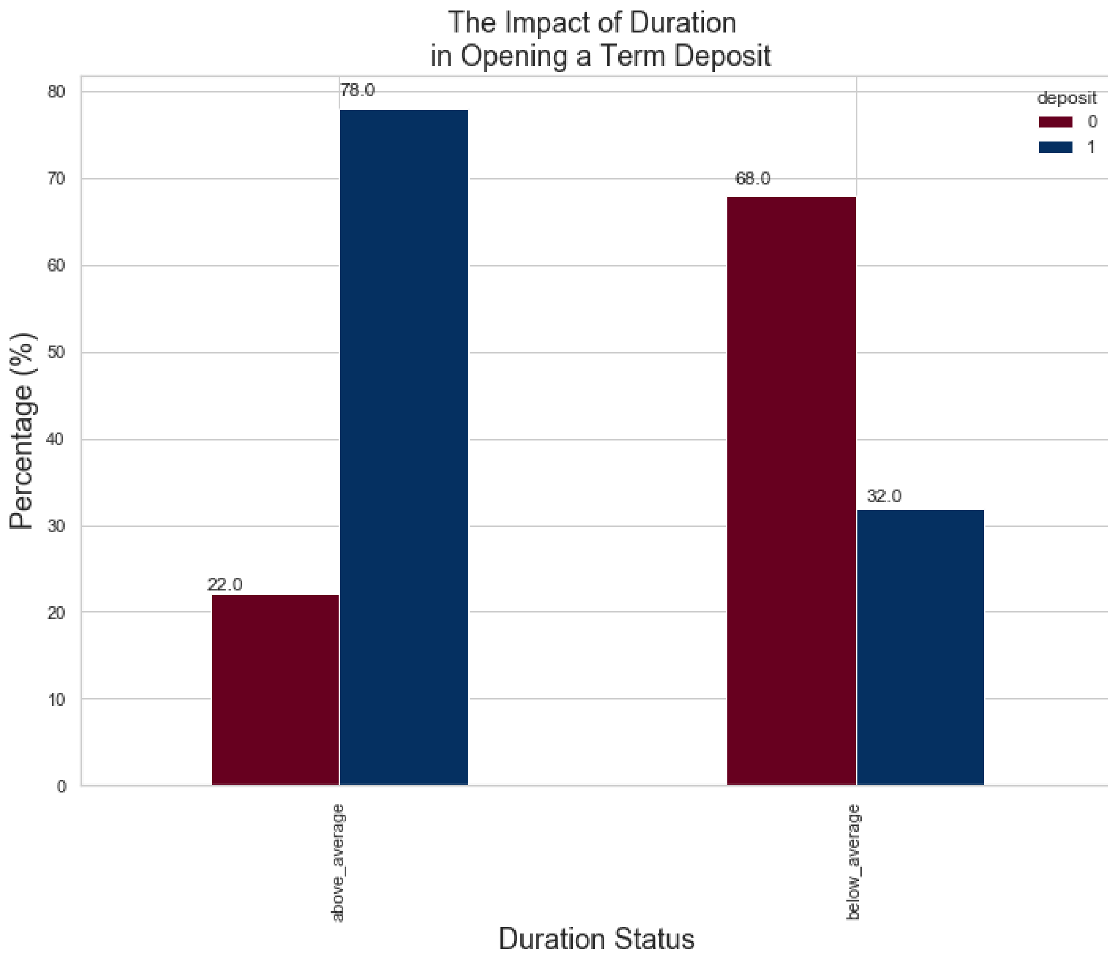
0.7 5.Correlation that impacted the decision of Potential Clients

[63]:
```
# Drop marital/education and balance status
# Scale both numeric and categorical vaues
# Use a correlation matrix to determine if duration has influence on term␣
 ↪deposits

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



# 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()
```



[64]:
```
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()
```



0.8 6. Classification Model

[79]:
```
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)
```

```
[79]: no     0.526877
      yes    0.473123
      Name: housing, dtype: float64
```

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

```
[80]: no     0.869199
      yes    0.130801
      Name: loan, dtype: float64
```

0.8.1 6.1 Stratified Sampling

```
[82]: from sklearn.model_selection import StratifiedShuffleSplit
      # Here we split the data into training and test sets and implement a stratified␣
       ↪shuffle split.
      stratified = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

      for train_set, test_set in stratified.split(term_deposits,␣
       ↪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)
```

```
[82]: no     0.196219
      yes    0.029519
      Name: loan, dtype: float64
```

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

```
[83]: no     4697
      yes    4232
      Name: deposit, dtype: int64
```

```
[84]: from sklearn.base import BaseEstimator, TransformerMixin
      from sklearn.utils import check_array
      from sklearn.preprocessing import LabelEncoder
      from scipy import sparse

      class CategoricalEncoder(BaseEstimator, TransformerMixin):
          """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 (discrete) features.
The features can be encoded using a one-hot aka one-of-K scheme
(``encoding='onehot'``, the default) or converted to ordinal integers
(``encoding='ordinal'``).
This encoding is needed for feeding categorical data to many scikit-learn
estimators, notably linear models and SVMs with the standard kernels.
Read more in the :ref:`User Guide <preprocessing_categorical_features>`.
Parameters
----------
encoding : str, 'onehot', 'onehot-dense' or 'ordinal'
    The type of encoding to use (default is 'onehot'):
    - 'onehot': encode the features using a one-hot aka one-of-K scheme
      (or also called 'dummy' encoding). This creates a binary column for
      each category and returns a sparse matrix.
    - 'onehot-dense': the same as 'onehot' but returns a dense array
      instead of a sparse matrix.
    - 'ordinal': encode the features as ordinal integers. This results in
      a single column of integers (0 to n_categories - 1) per feature.
categories : 'auto' or a list of lists/arrays of values.
    Categories (unique values) per feature:
    - 'auto' : Determine categories automatically from the training data.
    - list : ``categories[i]`` holds the categories expected in the ith
      column. The passed categories are sorted before encoding the data
      (used categories can be found in the ``categories_`` 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 unknown categorical feature is
    present during transform (default is to raise). When this is parameter
    is set to 'ignore' and an unknown category is encountered during
    transform, the resulting one-hot encoded columns for this feature
    will be all zeros.
    Ignoring unknown categories is not supported for
    ``encoding='ordinal'``.
Attributes
----------
categories_ : list of arrays
    The categories of each feature determined during fitting. When
    categories were specified manually, this holds the sorted categories
    (in order corresponding with output of `transform`).
Examples
--------
Given a dataset with three features and two samples, we let the encoder
find the maximum value per feature and transform the data to a binary
one-hot encoding.
>>> from sklearn.preprocessing import CategoricalEncoder
```

```python
    >>> enc = CategoricalEncoder(handle_unknown='ignore')
    >>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
    ... # doctest: +ELLIPSIS
    CategoricalEncoder(categories='auto', dtype=<... 'numpy.float64'>,
              encoding='onehot', handle_unknown='ignore')
    >>> 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 one-hot encoding of
      integer ordinal features. The ``OneHotEncoder assumes`` that input
      features take on values in the range ``[0, max(feature)]`` instead of
      using the unique values.
    sklearn.feature_extraction.DictVectorizer : performs a one-hot encoding of
      dictionary items (also handles string-valued features).
    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 each feature.
        Returns
        -------
        self
        """

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

        if self.handle_unknown not in ['error', 'ignore']:
            template = ("handle_unknown should be either 'error' or "
                        "'ignore', got %s")
            raise ValueError(template % self.handle_unknown)
```

```
    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_sparse='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.categories[i])
            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(Xi[~valid_mask])
                    msg = ("Found unknown categories {0} in column {1}"
                           " during fit".format(diff, i))
                    raise ValueError(msg)
            le.classes_ = np.array(np.sort(self.categories[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.categories_[i])
```

```python
            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 acceptable value and
                    # continue `The rows are marked `X_mask` and will be
                    # removed later.
                    X_mask[:, i] = valid_mask
                    X[:, i][~valid_mask] = self.categories_[i][0]
            X_int[:, i] = self._label_encoders_[i].transform(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.categories_]
        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, dtype=np.int32),
                                n_features)[mask]
        data = np.ones(n_samples * n_features)[mask]

        out = sparse.csc_matrix((data, (row_indices, column_indices)),
                                shape=(n_samples, indices[-1]),
                                dtype=self.dtype).tocsr()
        if self.encoding == 'onehot-dense':
            return out.toarray()
        else:
            return out
```

[85]:
```python
from sklearn.base import BaseEstimator, TransformerMixin

# A class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    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]
```

[86]:
```
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
```

[87]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Making pipelines
numerical_pipeline = Pipeline([
    ("select_numeric", DataFrameSelector(["age", "balance", "day", "campaign",␣
 ↪"pdays", "previous","duration"])),
    ("std_scaler", StandardScaler()),
])

categorical_pipeline = Pipeline([
    ("select_cat", DataFrameSelector(["job", "education", "marital", "default",␣
 ↪"housing", "loan", "contact", "month",
                                     "poutcome"])),
    ("cat_encoder", CategoricalEncoder(encoding='onehot-dense'))
])

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

[88]:
```
X_train = preprocess_pipeline.fit_transform(train_data)
X_train
```

[88]:
```
array([[ 1.14643868,  1.68761105,  1.69442818, …,  0.        ,
         0.        ,  1.        ],
       [-0.86102339, -0.35066205, -0.5560058 , …,  0.        ,
         0.        ,  1.        ],
       [-0.94466765, -0.20504785,  0.39154535, …,  0.        ,
         0.        ,  1.        ],
       …,
       [-0.86102339, -0.26889658, -1.02978138, …,  0.        ,
         0.        ,  1.        ],
       [ 0.2263519 , -0.32166951,  0.50998924, …,  0.        ,
         0.        ,  1.        ],
       [-0.61009063, -0.34740446,  1.69442818, …,  1.        ,
         0.        ,  0.        ]])
```

[89]:
```
y_train = train_data['deposit']
y_test = test_data['deposit']
y_train.shape
```

[89]: (8929,)

[90]:
```
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
```

[90]:
```
array([False, False,  True, …,  True,  True, False])
```

[91]:
```
some_instance = X_train[1250]
```

[92]:
```
import time


from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder

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 GradientBoostingClassifier
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": GradientBoostingClassifier(),
    "Decision Tree": tree.DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimators=18),
    "Neural Net": MLPClassifier(alpha=1),
    "Naive Bayes": GaussianNB()
}
```

[93]:
```
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_score', '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_train)
        df_results.loc[count,'classifier'] = key
        df_results.loc[count,'train_score'] = train_score
        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
```

[94]:
```
df_results = batch_classify(X_train, y_train)
print(df_results.sort_values(by='train_score', ascending=False))
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning:

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

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead
```

```
trained Logistic Regression in 0.04 s
trained Nearest Neighbors in 0.25 s
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning:

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

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead
```

```
trained Linear SVM in 2.66 s
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead


trained Gradient Boosting Classifier in 1.06 s
trained Decision Tree in 0.06 s

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead


trained Random Forest in 0.13 s
trained Neural Net in 6.82 s
```

```
trained Naive Bayes in 0.03 s
                    classifier  train_score  training_time
4                Decision Tree     1.000000       0.055442
5                Random Forest     0.997536       0.133377
1            Nearest Neighbors     0.863255       0.245500
3  Gradient Boosting Classifier     0.861463       1.058515
6                   Neural Net     0.852839       6.818850
2                   Linear SVM     0.852391       2.661102
0          Logistic Regression     0.830776       0.040675
7                  Naive Bayes     0.721693       0.026358
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9:
DeprecationWarning:

time.clock has been deprecated in Python 3.3 and will be removed from Python
3.8: use time.perf_counter or time.process_time instead
```

0.8.2 6.2 Overfitting

[95]:
```python
# 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_train, cv=3)
tree_mean = tree_scores.mean()

# Gradient Boosting Classifier
grad_clf = GradientBoostingClassifier()
grad_scores = cross_val_score(grad_clf, X_train, y_train, 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_train, 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', 'Dec Tree', 'Grad B CLF',␣
 ↪'Rand FC', 'Neural Classifier', 'Naives Bayes'],
    'Crossval Mean Scores': [log_reg_mean, svc_mean, knn_mean, tree_mean,␣
 ↪grad_mean, rand_mean, neural_mean, nav_mean]}

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

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning:

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

/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning:

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

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning:

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

/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning:

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

/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning:

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

/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning:

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

[97]:
```python
# All the models perform well but I choose GradientBoosting.
result_df = result_df.sort_values(by=['Crossval Mean Scores'], ascending=False)
result_df
```

[97]:

	Classifiers	Crossval Mean Scores
4	Grad B CLF	0.845224
6	Neural Classifier	0.844666
7	Naives Bayes	0.844666
5	Rand FC	0.841753
1	SVC	0.840186
0	Logistic Reg.	0.828425
2	KNN	0.804458
3	Dec Tree	0.783401

0.8.3 6.3 Cross Validation & Confusion Matrix

[102]:
```
# 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)
```

[103]:
```
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 0.85
```

[104]:
```
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", linewidths=.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, minor=False)
ax.set_xticklabels("")
ax.set_yticklabels(['Refused T. Deposits', 'Accepted T. Deposits'],␣
↪fontsize=16, rotation=360)
plt.show()
```

0.8.4 6.4 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"!**

[105]:
```
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:  0.8246013667425968
Recall Score:  0.8553875236294896
```

[106]:
```
from sklearn.metrics import f1_score

f1_score(y_train, y_train_pred)
```

[106]: 0.8397123637207144

[107]:
```
y_scores = grad_clf.decision_function([some_instance])
y_scores
```

[107]: array([-3.65645629])

[108]:
```
# Increasing the threshold decreases the recall.
threshold = 0
y_some_digit_pred = (y_scores > threshold)
```

[109]:
```
y_scores = cross_val_predict(grad_clf, X_train, y_train, cv=3,␣
↪method="decision_function")
neural_y_scores = cross_val_predict(neural_clf, X_train, y_train, cv=3,␣
↪method="predict_proba")
```

```
naives_y_scores = cross_val_predict(nav_clf, X_train, y_train, cv=3,␣
↪method="predict_proba")
```

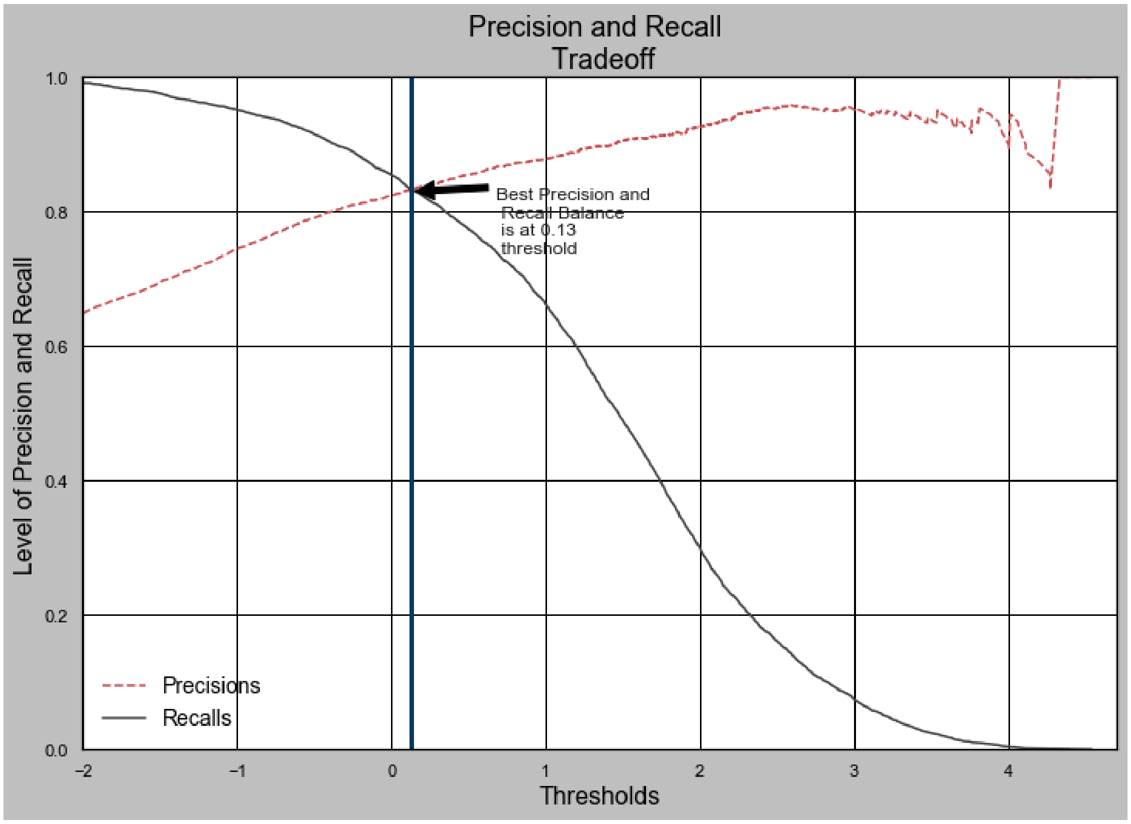
[110]:
```
if y_scores.ndim == 2:
    y_scores = y_scores[:, 1]

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]
```

[111]:
```
y_scores.shape
```

[111]: (8929,)

[113]:
```
# Determining threshold value
from sklearn.metrics import precision_recall_curve
precisions, recalls, threshold = precision_recall_curve(y_train, y_scores)
```

[114]:
```
def precision_recall_curve(precisions, recalls, thresholds):
    fig, ax = plt.subplots(figsize=(12,8))
    plt.plot(thresholds, precisions[:-1], "r--", label="Precisions")
    plt.plot(thresholds, recalls[:-1], "#424242", label="Recalls")
    plt.title("Precision and Recall \n Tradeoff", fontsize=18)
    plt.ylabel("Level of Precision and Recall", fontsize=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()
```

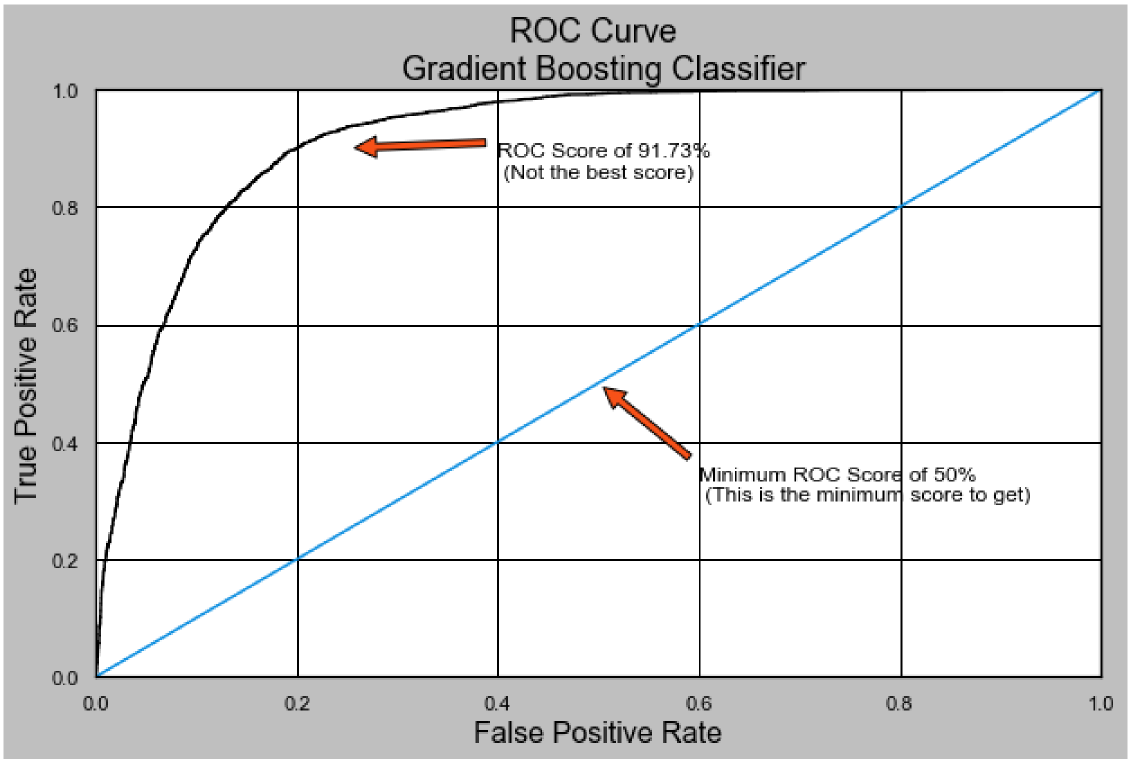


0.8.5 6.5 ROC

[115]:
```python
from sklearn.metrics import roc_curve
# Gradient Boosting Classifier
# Neural Classifier
# Naives Bayes Classifier
grd_fpr, grd_tpr, thresold = roc_curve(y_train, y_scores)
neu_fpr, neu_tpr, neu_threshold = roc_curve(y_train, neural_y_scores)
nav_fpr, nav_tpr, nav_threshold = roc_curve(y_train, naives_y_scores)
```

[116]:
```python
def graph_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.figure(figsize=(10,6))
    plt.title('ROC Curve \n Gradient Boosting Classifier', 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', shrink=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', shrink=0.05),
                )


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

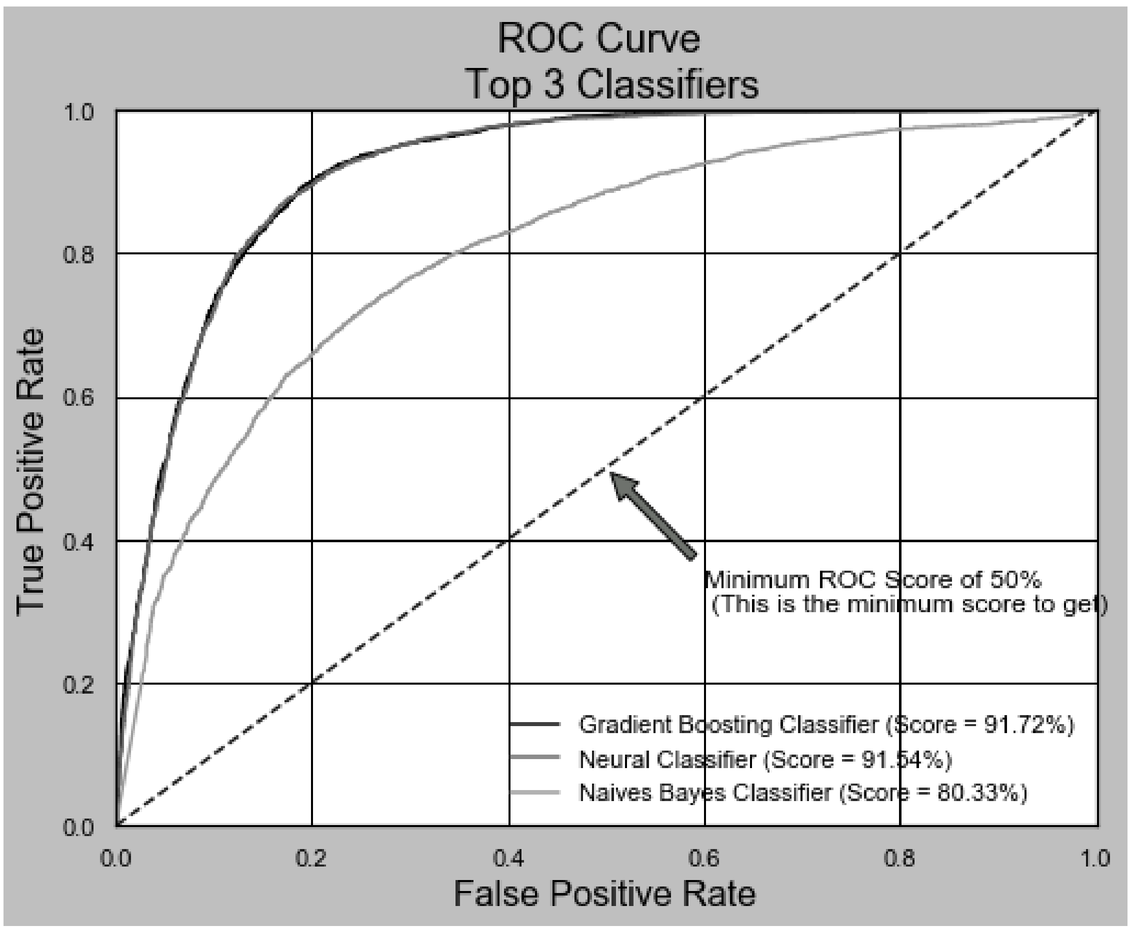


[117]:
```python
from sklearn.metrics import roc_auc_score

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

```
Gradient Boost Classifier Score:  0.917315676901115
Neural Classifier Score:  0.9161829756595631
Naives Bayes Classifier:  0.803363959942255
```

[118]:
```python
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', fontsize=18)
    plt.plot(grd_fpr, grd_tpr, label='Gradient Boosting Classifier (Score = 91.
↪72%)')
    plt.plot(neu_fpr, neu_tpr, label='Neural Classifier (Score = 91.54%)')
    plt.plot(nav_fpr, nav_tpr, label='Naives Bayes Classifier (Score = 80.33%)')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    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='#6E726D', shrink=0.05),
                )
    plt.legend()

graph_roc_curve_multiple(grd_fpr, grd_tpr, neu_fpr, neu_tpr, nav_fpr, nav_tpr)
plt.show()
```



[119]:
```
grad_clf.predict_proba([some_instance])
```

[119]: array([[0.97482622, 0.02517378]])

[120]:
```
# Classifier Prediction
grad_clf.predict([some_instance])
```

[120]: array([0])

[121]:
```
y_train[1250]
```

[121]: 0

1 Which Features Influence the Result of a Term Deposit Suscription?

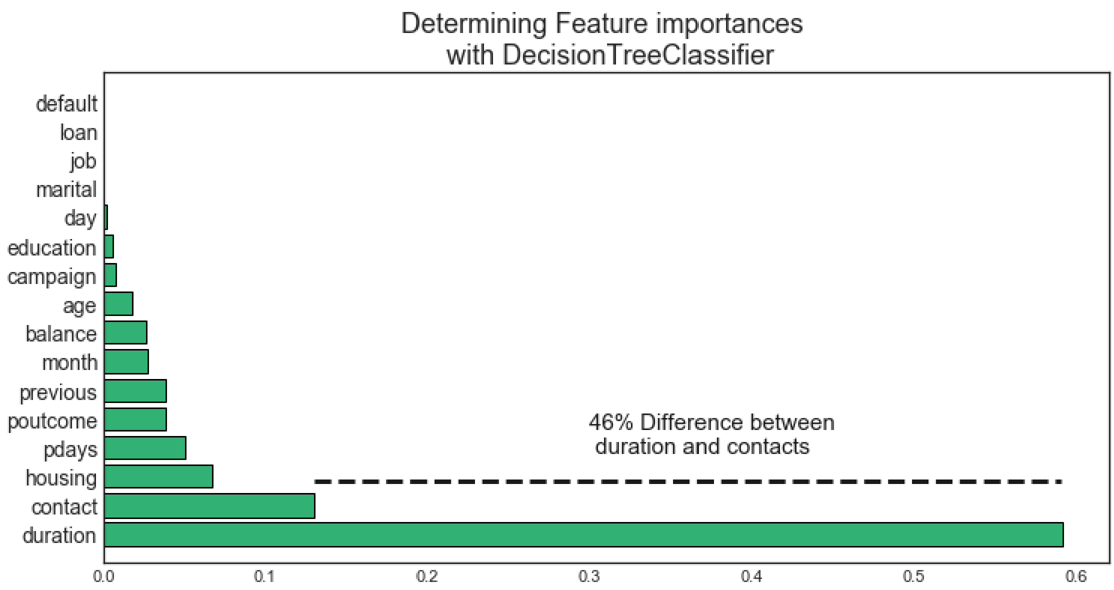
1.1 6.6 DecisionTreeClassifier:

The top three most important features for our classifier are **Duration (how long it took the conversation between the sales representative and the potential client), contact (number of contacts to the potential client within the same marketing campaign), month (the month of the year).

[122]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
plt.style.use('seaborn-white')

# Convert the columns into categorical variables
term_deposits['job'] = term_deposits['job'].astype('category').cat.codes
term_deposits['marital'] = term_deposits['marital'].astype('category').cat.codes
term_deposits['education'] = term_deposits['education'].astype('category').cat.
 ↪codes
term_deposits['contact'] = term_deposits['contact'].astype('category').cat.codes
term_deposits['poutcome'] = term_deposits['poutcome'].astype('category').cat.
 ↪codes
term_deposits['month'] = term_deposits['month'].astype('category').cat.codes
term_deposits['default'] = term_deposits['default'].astype('category').cat.codes
term_deposits['loan'] = term_deposits['loan'].astype('category').cat.codes
term_deposits['housing'] = term_deposits['housing'].astype('category').cat.codes

# Let's create new splittings like before but now we modified the data so we␣
 ↪need to do it one more time.
# Create train and test splits
target_name = 'deposit'
```

```
X = term_deposits.drop('deposit', axis=1)


label=term_deposits[target_name]

X_train, X_test, y_train, y_test = train_test_split(X,label,test_size=0.2,␣
↪random_state=42, stratify=label)

# Build a classification task using 3 informative features
tree = tree.DecisionTreeClassifier(
    class_weight='balanced',
    min_weight_fraction_leaf = 0.01

)



tree = tree.fit(X_train, y_train)
importances = tree.feature_importances_
feature_names = term_deposits.drop('deposit', axis=1).columns
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest
def feature_importance_graph(indices, importances, feature_names):
    plt.figure(figsize=(12,6))
    plt.title("Determining Feature importances \n with DecisionTreeClassifier",␣
↪fontsize=18)
    plt.barh(range(len(indices)), importances[indices], color='#31B173', ␣
↪align="center")
    plt.yticks(range(len(indices)), feature_names[indices],␣
↪rotation='horizontal',fontsize=14)
    plt.ylim([-1, len(indices)])
    plt.axhline(y=1.85, xmin=0.21, xmax=0.952, color='k', linewidth=3,␣
↪linestyle='--')
    plt.text(0.30, 2.8, '46% Difference between \n duration and contacts',␣
↪color='k', fontsize=15)

feature_importance_graph(indices, importances, feature_names)
plt.show()
```

```
Feature ranking:
1. feature 11 (0.591310)
2. feature 8 (0.129966)
3. feature 6 (0.067020)
4. feature 13 (0.049923)
5. feature 15 (0.038138)
6. feature 14 (0.037830)
7. feature 10 (0.026646)
8. feature 5 (0.025842)
9. feature 0 (0.017757)
10. feature 12 (0.007889)
11. feature 3 (0.005280)
12. feature 9 (0.002200)
13. feature 2 (0.000147)
14. feature 1 (0.000050)
15. feature 7 (0.000000)
16. feature 4 (0.000000)
```



1.2 GradientBoosting Classifier!

Gradient Boosting classifier is the best model to predict whether or not a **potential client** will suscribe to a term deposit or not. 84% accuracy!

[124]:
```
# Our three classifiers are grad_clf, nav_clf and neural_clf
from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(
    estimators=[('gbc', grad_clf), ('nav', nav_clf), ('neural', neural_clf)],
```

```
    voting='soft'
)

voting_clf.fit(X_train, y_train)
```

[124]:
```
VotingClassifier(estimators=[('gbc',
GradientBoostingClassifier(criterion='friedman_mse',
                                                        init=None,
                                                        learning_rate=0.1,
                                                        loss='deviance',
                                                        max_depth=3,
                                                        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,
                                                        n_estimators=100,
                                                        n_iter_no_change=None,
                                                        presort='au…
                                           beta_2=0.999, early_stopping=False,
                                           epsilon=1e-08,
                                           hidden_layer_sizes=(100,),
                                           learning_rate='constant',
                                           learning_rate_init=0.001,
                                           max_iter=200, momentum=0.9,
                                           n_iter_no_change=10,
                                           nesterovs_momentum=True,
                                           power_t=0.5, random_state=None,
                                           shuffle=True, solver='adam',
                                           tol=0.0001, validation_fraction=0.1,
                                           verbose=False, warm_start=False))],
                 flatten_transform=True, n_jobs=None, voting='soft',
                 weights=None)
```

[125]:
```
from sklearn.metrics import accuracy_score

for clf in (grad_clf, nav_clf, neural_clf, voting_clf):
    clf.fit(X_train, y_train)
    predict = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, predict))
```

```
GradientBoostingClassifier 0.8463949843260188
GaussianNB 0.7514554411106136
MLPClassifier 0.7689207344379758
VotingClassifier 0.8029556650246306
```

1.3 7. What Actions should the Bank Consider?

1.3.1 Solutions for the Next Marketing Campaign (Conclusion):

1) **Months of Marketing Activity:** We saw that the the month of highest level of marketing activity was the month of **May**. However, this was the month that potential clients tended to reject term deposits offers (Lowest effective rate: -34.49%). For the next marketing campaign, it will be wise for the bank to focus the marketing campaign during the months of **March, September, October and December.** (December should be under consideration because it was the month with the lowest marketing activity, there might be a reason why december is the lowest.)

2) **Seasonality:** Potential clients opted to suscribe term deposits during the seasons of **fall** and **winter**. The next marketing campaign should focus its activity throghout these seasons.

3) **Campaign Calls:** A policy should be implemented that states that no more than 3 calls should be applied to the same potential client in order to save time and effort in getting new potential clients. Remember, the more we call the same potential client, the likely he or she will decline to open a term deposit.

4) **Age Category:** The next marketing campaign of the bank should target potential clients in their 20s or younger and 60s or older. The youngest category had a 60% chance of suscribing to a term deposit while the eldest category had a 76% chance of suscribing to a term deposit. It will be great if for the next campaign the bank addressed these two categories and therefore, increase the likelihood of more term deposits suscriptions.

5) **Occupation:** Not surprisingly, potential clients that were students or retired were the most likely to suscribe to a term deposit. Retired individuals, tend to have more term deposits in order to gain some cash through interest payments. Remember, term deposits are short-term loans in which the individual (in this case the retired person) agrees not to withdraw the cash from the bank until a certain date agreed between the individual and the financial institution. After that time the individual gets its capital back and its interest made on the loan. Retired individuals tend to not spend bigly its cash so they are morelikely to put their cash to work by lending it to the financial institution. Students were the other group that used to suscribe term deposits.

6) **House Loans and Balances:** Potential clients in the low balance and no balance category were more likely to have a house loan than people in the average and high balance category. What does it mean to have a house loan? This means that the potential client has financial compromises to pay back its house loan and thus, there is no cash for he or she to suscribe to a term deposit account. However, we see that potential clients in the average and hih balances are less likely to have a house loan and therefore, more likely to open a term deposit. Lastly, the next marketing campaign should focus on individuals of average and high balances in order to increase the likelihood of suscribing to a term deposit.

7) **Develop a Questionaire during the Calls:** Since duration of the call is the feature that most positively correlates with whether a potential client will open a term deposit or not, by providing an interesting questionaire for potential clients during the calls the conversation length might increase. Of course, this does not assure us that the potential client will suscribe to a term deposit! Nevertheless, we don't loose anything by implementing a strategy that will increase the level of engagement of the potential client leading to an increase probability of

suscribing to a term deposit, and therefore an increase in effectiveness for the next marketing campaign the bank will excecute.

8) Target individuals with a higher duration (above 375): Target the target group that is above average in duration, there is a highly likelihood that this target group would open a term deposit account. The likelihood that this group would open a term deposit account is at 78% which is pretty high. This would allow that the success rate of the next marketing campaign would be highly successful.

By combining all these strategies and simplifying the market audience the next campaign should address, it is likely that the next marketing campaign of the bank will be more effective than the current one.