kaggle Search Competitions Datasets Notebooks Discussion Courses •••

In [1]:
```
#import linear algebra and data manipulation libraries
import numpy as np
import pandas as pd

#import standard visualization
import matplotlib.pyplot as plt
import seaborn as sns

#import machine learning
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

import xgboost

from sklearn.model_selection import train_test_split #split
from sklearn.metrics import accuracy_score #metrics

#tools for hyperparameters search
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['bank.csv']
```

Introduction

Today organizations, which hire data scientists are especially interested in job candidate's portfolio. Analysis of organization's marketing data is one of the most typical applications of data science and machine learning. Such analysis will definetely be a nice contribution to the protfolio.

In general, datasets which contain marketing data can be used for 2 different business goals:

1. Prediction of the results of the marketing campaign for each customer and clarification of factors which affect the campaign results. This helps to find out the ways how to make marketing campaigns more efficient.
2. Finding out customer segments, using data for customers, who subscribed to term deposit. This helps to identify the profile of a customer, who is more likely to acquire the product and develop more targeted marketing campaigns.

This dataset contains banking marketing campaign data and we can use it to optimize marketing campaigns to attract more customers to term deposit subscription. Detailed description of the dataset's content is describe in this kaggle kernel (https://www.kaggle.com/janiobachmann/marketing-in-banking-opening-term-deposits).

Approach

In order to optimize marketing campaigns with the help of the dataset, we will have to take the following steps:

1. Import data from dataset and perform initial high-level analysis: look at the number of rows, look at the missing values, look at dataset columns and their values respective to the campaign outcome.
2. Clean the data: remove irrelevant columns, deal with missing and incorrect values, turn categorical columns into dummy variables.
3. Use machine learning techniques to predict the marketing campaign outcome and to find out factors, which affect the success of the campaign.

Import Data

First of all to perform the analysis, we have to import the data:

In [2]:
```
#import dataset

df = pd.read_csv('../input/bank.csv')
```

In [3]:
```
df.head()
```

Out[3]:

al	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
ed	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
ed	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
ed	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
ed	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
ed	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

Data Exploration

After we imported the dataset, we have to look at the total number of rows in the dataset and analyze the number of missing values.

In [4]:
```
# number of rows in dataset

print("Bank marketing dataset consists of {rows} rows.".format(rows = len(df)))
```

```
Bank marketing dataset consists of 11162 rows.
```

In [5]:
```
#find percentage of missing values for each column
missing_values = df.isnull().mean()*100
```

```
missing_values.sum()
```

Out[5]:

```
0.0
```

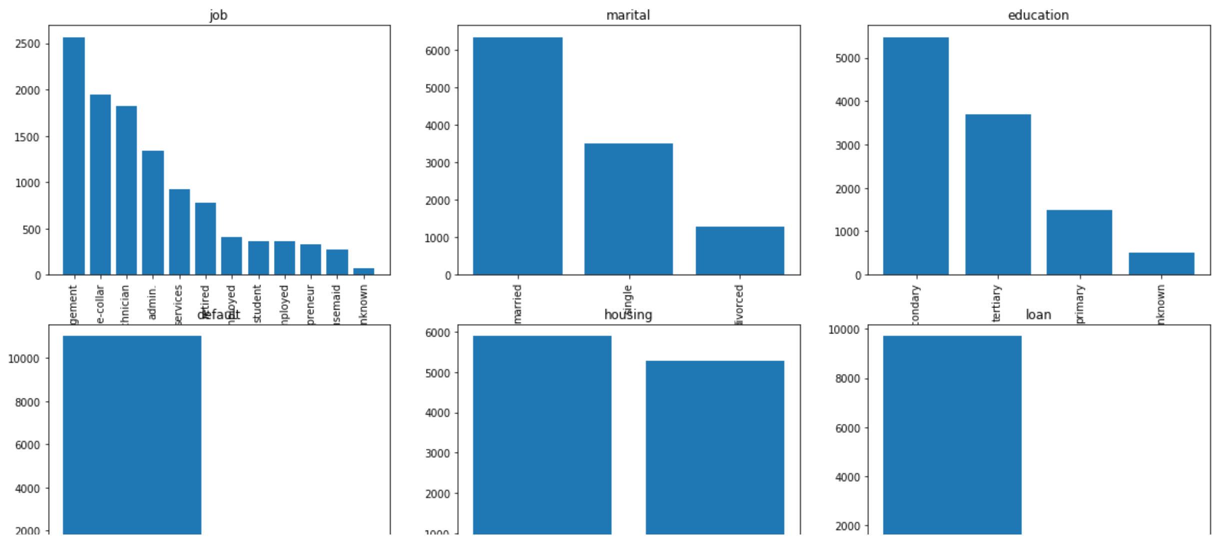
So we see that there are no missing values.

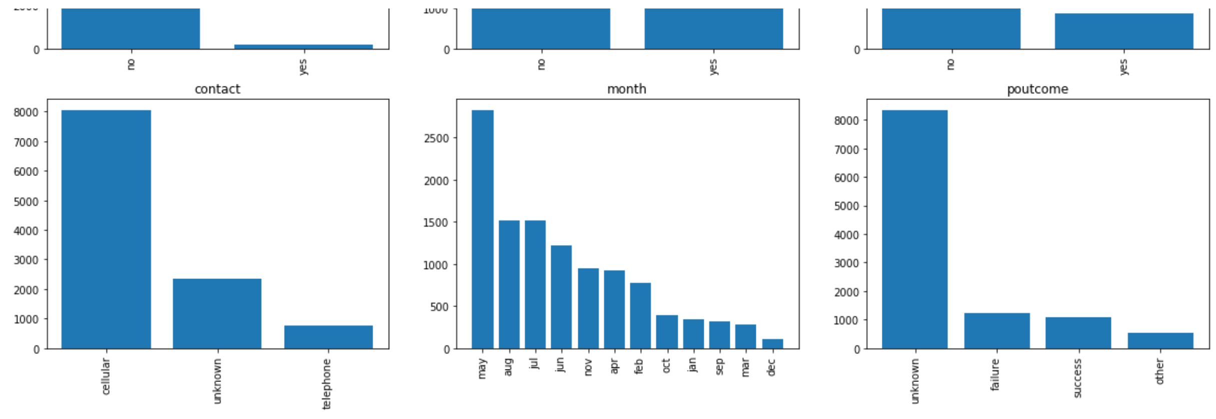
Categorical columns exploration

In the dataset we have both categorical and numerical columns. Let's look at the values of categorical columns first.

In [6]:

```
cat_columns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month',
'poutcome']

fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(20, 15))

counter = 0
for cat_column in cat_columns:
    value_counts = df[cat_column].value_counts()

    trace_x = counter // 3
    trace_y = counter % 3
    x_pos = np.arange(0, len(value_counts))

    axs[trace_x, trace_y].bar(x_pos, value_counts.values, tick_label = value_counts.index)

    axs[trace_x, trace_y].set_title(cat_column)

    for tick in axs[trace_x, trace_y].get_xticklabels():
        tick.set_rotation(90)

    counter += 1

plt.show()
```



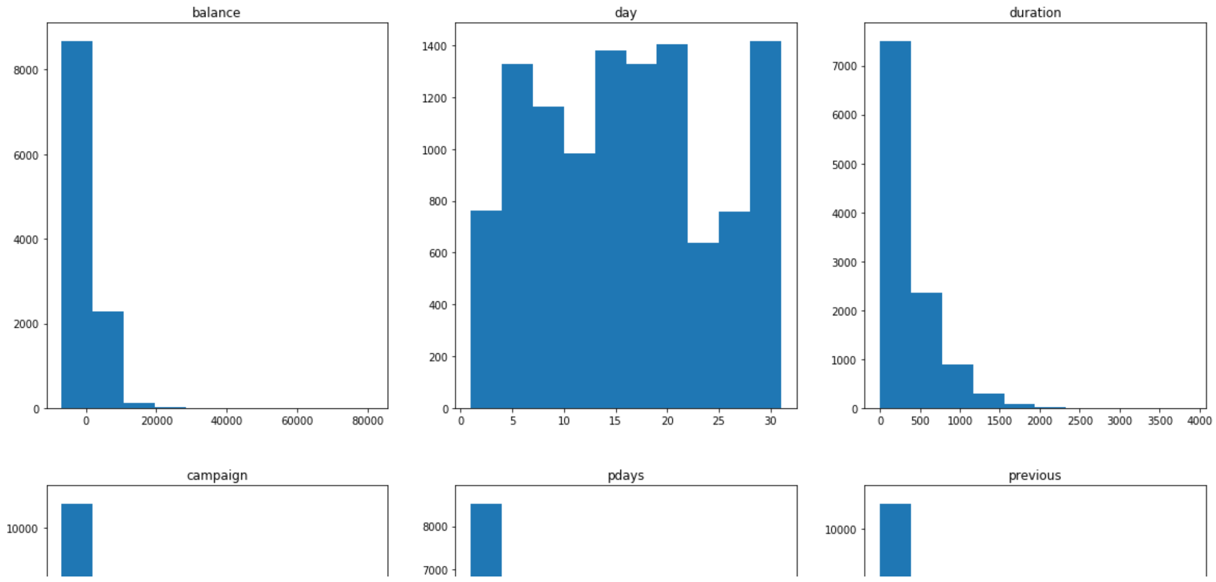


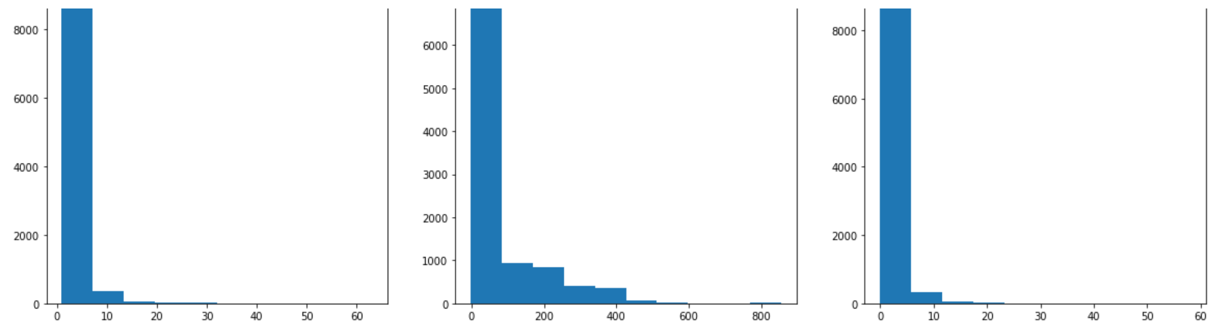
Numerical columns exploration

Now let's look at the numerical columns' values. The most convenient way to look at the numerical values is plotting histograms.

In [7]:

```
num_columns = ['balance', 'day','duration', 'campaign', 'pdays', 'previous']

fig, axs = plt.subplots(2, 3, sharex=False, sharey=False, figsize=(20, 15))

counter = 0
for num_column in num_columns:

    trace_x = counter // 3
    trace_y = counter % 3

    axs[trace_x, trace_y].hist(df[num_column])

    axs[trace_x, trace_y].set_title(num_column)

    counter += 1

plt.show()
```





We can see that numerical columns have outliers (especially 'pdays', 'campaign' and 'previous' columns). Possibly there are incorrect values (noisy data), so we should look closer at the data and decide how do we manage the noise.
Let's look closer at the values of 'campaign', 'pdays' and 'previous' columns:

In [8]:
```
df[['pdays', 'campaign', 'previous']].describe()
```

Out[8]:

	pdays	campaign	previous
count	11162.000000	11162.000000	11162.000000
mean	51.330407	2.508421	0.832557
std	108.758282	2.722077	2.292007
min	-1.000000	1.000000	0.000000
25%	-1.000000	1.000000	0.000000
50%	-1.000000	2.000000	0.000000
75%	20.750000	3.000000	1.000000
max	854.000000	63.000000	58.000000

Percentage of 'pdays' values above 400:

In [9]:
```
len (df[df['pdays'] > 400] ) / len(df) * 100
```

Out[9]:
```
1.2005017022039062
```

'pdays' holds the number of days that passed by after the client was last contacted from a previous campaign Looking closer into 'pdays' data we can see that:

- only 1.2% of values above 400. They are possibly outliers, so we should consider imputing something (possibly mean value) instead of these values.
- -1 possibly means that the client wasn't contacted before or stands for missing data.

Since we are not sure exactly what -1 means I suggest to drop this column, because -1 makes more than 50% of the values of the column.

Percentage of 'campaign' values above 20:

In [10]:
```
len (df[df['campaign'] > 34] ) / len(df) * 100
```

Out[10]:
```
0.035835871707579285
```

'campaign' holds the number of contacts performed during this campaign and for this client (numeric, includes last contact) Numbers for 'campaign' above 34 are clearly noise, so I suggest to impute them with average campaign values while data cleaning.

Percentage of 'previous' values above 20:

In [11]:
```
len (df[df['previous'] > 34] ) / len(df) * 100
```

Out[11]:
```
0.04479483963447411
```

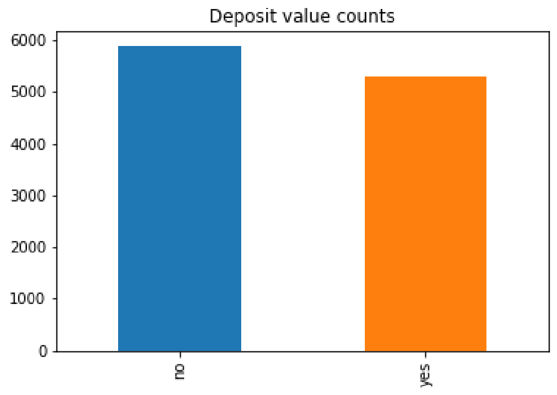
'previous' holds the number of contacts performed before this campaign and for this client (numeric) Numbers for 'previous' above 34 are also really strange, so I suggest to impute them with average campaign values while data cleaning.

Analysis of the response column

It is very important to look at the response column, which holds the information, which we are going to predict. In our case we should look at 'deposit' column and compare its values to other columns.
First of all we should look at the number of 'yes' and 'no' values in the response column 'deposit'.

In [12]:
```
value_counts = df['deposit'].value_counts()

value_counts.plot.bar(title = 'Deposit value counts')
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d70a1b38>
```

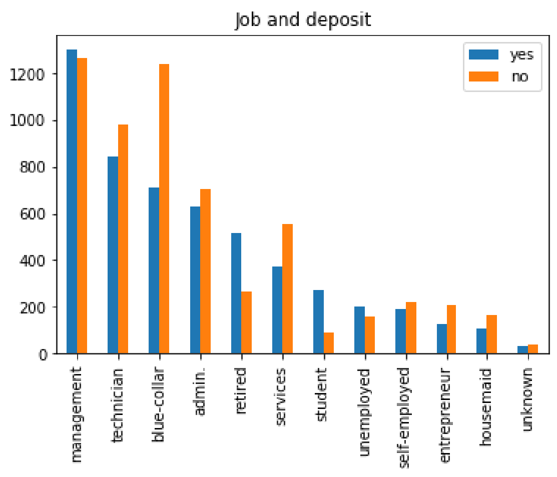


On the diagram we see that counts for 'yes' and 'no' values for 'deposit' are close, so we can use accuracy as a metric for a model, which predicts the campaign outcome.

Let's see how 'deposit' column value varies depending on other categorical columns' values:

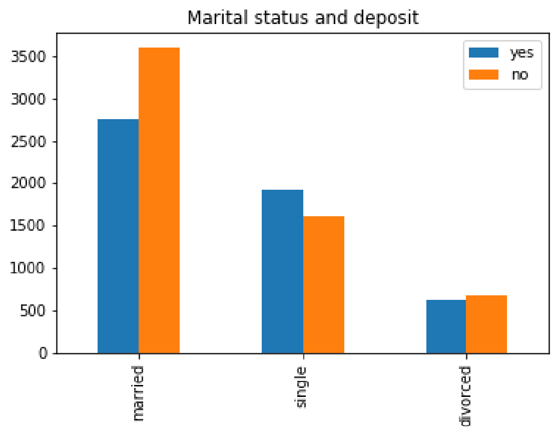
In [13]:
```
#job and deposit
j_df = pd.DataFrame()

j_df['yes'] = df[df['deposit'] == 'yes']['job'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['job'].value_counts()

j_df.plot.bar(title = 'Job and deposit')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6f16eb8>
```



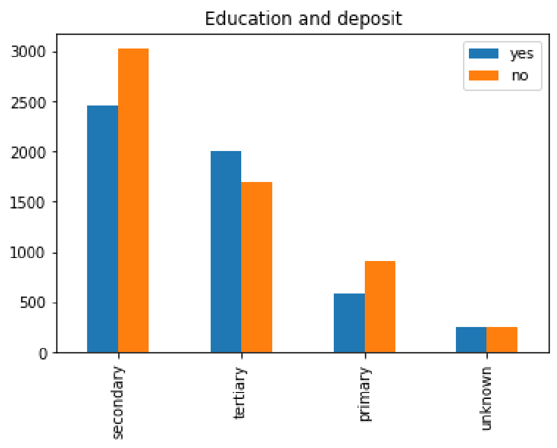
```
j_df['yes'] = df[df['deposit'] == 'yes']['marital'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['marital'].value_counts()

j_df.plot.bar(title = 'Marital status and deposit')
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6fb42e8>
```



In [15]:
```
#education and deposit
j_df = pd.DataFrame()

j_df['yes'] = df[df['deposit'] == 'yes']['education'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['education'].value_counts()

j_df.plot.bar(title = 'Education and deposit')
```

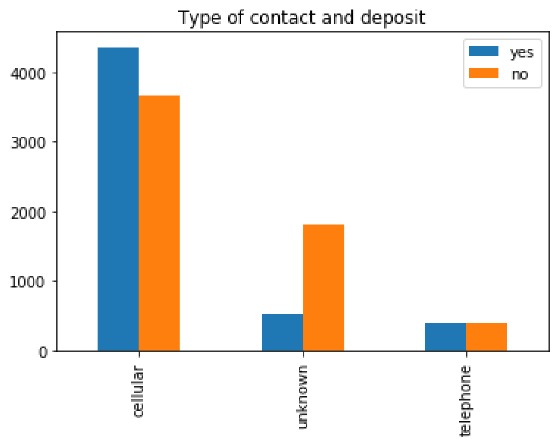
Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6db39e8>
```



In [16]:
```
#type of contact and deposit
j_df = pd.DataFrame()

j_df['yes'] = df[df['deposit'] == 'yes']['contact'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['contact'].value_counts()
```

Notebook Data Comments

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6d2e198>
```



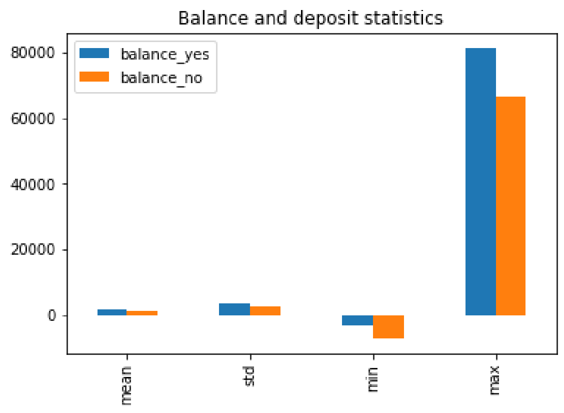
Regarding the diagrams we can tell that according to our dataset:

1. Customers with 'blue-collar' and 'services' jobs are less likely to subscribe for term deposit.
2. Married customers are less likely to subscribe for term deposit.
3. Customers with 'cellular' type of contact are less likely to subscribe for term deposit.

Now let's look how numerical columns affect term deposit subscription.

In [17]:
```
#balance and deposit

b_df = pd.DataFrame()
b_df['balance_yes'] = (df[df['deposit'] == 'yes'][['deposit','balance']].describe())['balance']
b_df['balance_no'] = (df[df['deposit'] == 'no'][['deposit','balance']].describe())['balance']

b_df
```
Out[17]:

	balance_yes	balance_no
count	5289.000000	5873.000000
mean	1804.267915	1280.227141
std	3501.104777	2933.411934
min	-3058.000000	-6847.000000
25%	210.000000	64.000000
50%	733.000000	414.000000
75%	2159.000000	1324.000000
max	81204.000000	66653.000000

In [18]:
```
b_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Balance and deposit statistics')
```
Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6cfba58>
```


In [19]:
```
#age and deposit

a_df = pd.DataFrame()
```

```
a_df['age_yes'] = (df[df['deposit'] == 'yes'][['deposit','age']].describe())['age']
a_df['age_no'] = (df[df['deposit'] == 'no'][['deposit','age']].describe())['age']

a_df
```

Out[19]:

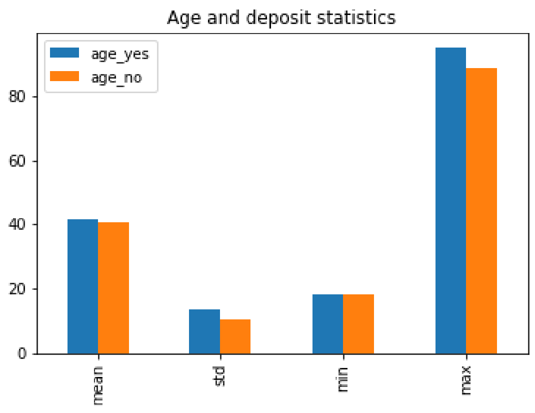
	age_yes	age_no
count	5289.000000	5873.000000
mean	41.670070	40.837391
std	13.497781	10.264815
min	18.000000	18.000000
25%	31.000000	33.000000
50%	38.000000	39.000000
75%	50.000000	48.000000
max	95.000000	89.000000

In [20]:

```
a_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Age and deposit statistics')
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6c6c4a8>
```



In [21]:

```
#number of contacts performed during this campaign ('campaign') and deposit
c_df = pd.DataFrame()
c_df['campaign_yes'] = (df[df['deposit'] == 'yes'][['deposit','campaign']].describe())['campaig
n']
c_df['campaign_no'] = (df[df['deposit'] == 'no'][['deposit','campaign']].describe())['campaign'
]

c_df
```

Out[21]:

	campaign_yes	campaign_no
count	5289.000000	5873.000000
mean	2.141047	2.839264
std	1.921826	3.244474
min	1.000000	1.000000

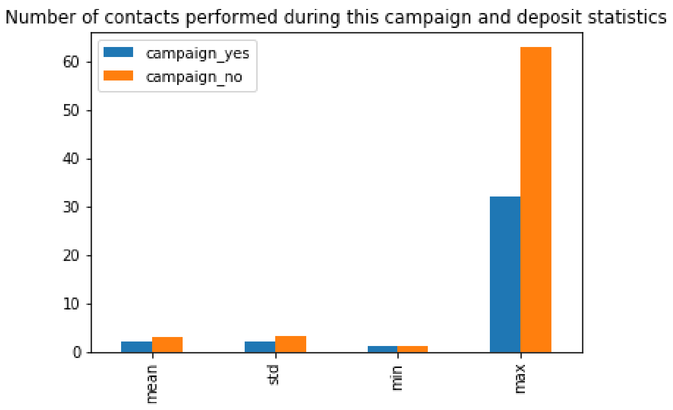
25%	1.000000	1.000000
50%	2.000000	2.000000
75%	3.000000	3.000000
max	32.000000	63.000000

In [22]:

```
c_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Number of contacts performed during
this campaign and deposit statistics')
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30d6c5dd68>
```



In [23]:

```
#number of contacts performed during previous campaign ('previous') and deposit
p_df = pd.DataFrame()
p_df['previous_yes'] = (df[df['deposit'] == 'yes'][['deposit','previous']].describe())['previou
s']
p_df['previous_no'] = (df[df['deposit'] == 'no'][['deposit','previous']].describe())['previous'
]

p_df
```

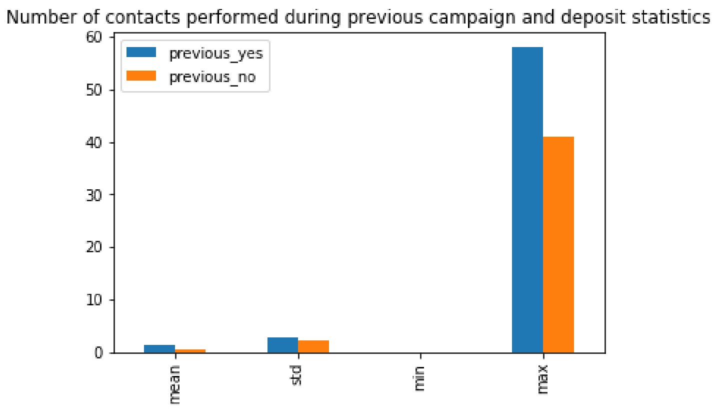
Out[23]:

	previous_yes	previous_no
count	5289.000000	5873.00000
mean	1.170354	0.52835
std	2.553272	1.97961
min	0.000000	0.00000
25%	0.000000	0.00000
50%	0.000000	0.00000
75%	1.000000	0.00000
max	58.000000	41.00000

In [24]:

```
p_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Number of contacts performed during
previous campaign and deposit statistics')
```

```
Out[24]:
        <matplotlib.axes._subplots.AxesSubplot at 0x7f30d2e59978>
```



Looking at the diagrams above we can conclude that:

1. People who subscribed for term deposit tend to have greater balance and age values.
2. People who subscribed for term deposit tend to have fewer number of contacts during this campaign.

Data Cleaning

Before we will be able to apply machine learning techniques, we should prepare the dataset for processing:

1. Convert columns with 'yes' and 'no' values to boolean columns;
2. Convert categorical columns into dummy variables.

```
In [25]:
def get_dummy_from_bool(row, column_name):
    ''' Returns 0 if value in column_name is no, returns 1 if value in column_name is yes'''
    return 1 if row[column_name] == 'yes' else 0

def get_correct_values(row, column_name, threshold, df):
    ''' Returns mean value if value in column_name is above threshold'''
    if row[column_name] <= threshold:
        return row[column_name]
    else:
        mean = df[df[column_name] <= threshold][column_name].mean()
        return mean

def clean_data(df):
    '''
    INPUT
    df - pandas dataframe containing bank marketing campaign dataset

    OUTPUT
    df - cleaned dataset:
    1. columns with 'yes' and 'no' values are converted into boolean variables;
    2. categorical columns are converted into dummy variables;
    3. drop irrelevant columns.
    4. impute incorrect values
    '''
    '''

    cleaned_df = df.copy()

    #convert columns containing 'yes' and 'no' values to boolean variables and drop original colu
mns
    bool_columns = ['default', 'housing', 'loan', 'deposit']
    for bool_col in bool_columns:
        cleaned_df[bool_col + '_bool'] = df.apply(lambda row: get_dummy_from_bool(row, bool_col
),axis=1)

    cleaned_df = cleaned_df.drop(columns = bool_columns)

    #convert categorical columns to dummies
    cat_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

    for col in  cat_columns:
        cleaned_df = pd.concat([cleaned_df.drop(col, axis=1),
                                pd.get_dummies(cleaned_df[col], prefix=col, prefix_sep='_',
                                               drop_first=True, dummy_na=False)], axis=1)

    #drop irrelevant columns
    cleaned_df = cleaned_df.drop(columns = ['pdays'])

    #impute incorrect values and drop original columns
    cleaned_df['campaign_cleaned'] = df.apply(lambda row: get_correct_values(row, 'campaign', 3
4, cleaned_df),axis=1)
    cleaned_df['previous_cleaned'] = df.apply(lambda row: get_correct_values(row, 'previous', 3
4, cleaned_df),axis=1)

    cleaned_df = cleaned_df.drop(columns = ['campaign', 'previous'])

    return cleaned_df
```

In [26]:

```
#clean the dataset
cleaned_df = clean_data(df)
cleaned_df.head()
```

Out[26]:

	age	balance	day	duration	default_bool	housing_bool	loan_bool	deposit_bool	job_blue-collar	job_entrepreneur	job_housemaid
0	59	2343	5	1042	0	1	0	1	0	0	0
1	56	45	5	1467	0	0	0	1	0	0	0
2	41	1270	5	1389	0	1	0	1	0	0	0
3	55	2476	5	579	0	1	0	1	0	0	0
4	54	184	5	673	0	0	0	1	0	0	0

Machine Learning for prediction of campaign outcome

Classification model for the campaign outcome prediction

Now let's use cleaned datasets for prediction of campaign outcome with help of machine learning classification models. I will use **XGBoost (https://xgboost.readthedocs.io/en/latest/)**, which is one of the most common machine learning libraries for modelling.
Resulting model will also help me to understand, which features have the greatest importance for the prediction of the results of the campaing.

Create X and y datasets for training the model and split into train and test datasets.

In [27]:
```
X = cleaned_df.drop(columns = 'deposit_bool')
y = cleaned_df[['deposit_bool']]
```

In [28]:
```
TEST_SIZE = 0.3
RAND_STATE = 42
```

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = TEST_SIZE, random_state=R
AND_STATE)
```

Train XGBoost classifier model:

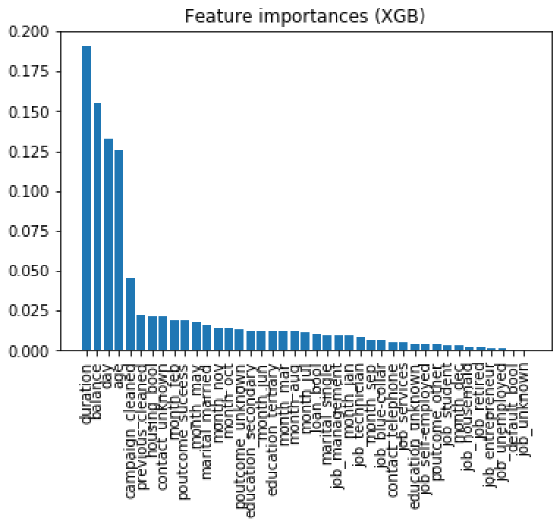
In [30]:
```
#train XGBoost model
xgb = xgboost.XGBClassifier(n_estimators=100, learning_rate=0.08, gamma=0, subsample=0.75,
                           colsample_bytree=1, max_depth=7)
xgb.fit(X_train,y_train.squeeze().values)

#calculate and print scores for the model for top 15 features
y_train_preds = xgb.predict(X_train)
y_test_preds = xgb.predict(X_test)

print('XGB accuracy score for train: %.3f: test: %.3f' % (
        accuracy_score(y_train, y_train_preds),
        accuracy_score(y_test, y_test_preds)))
```

```
XGB accuracy score for train: 0.911: test: 0.848
```

Get the feature importances from the trained model:

In [31]:
```
#get feature importances from the model
headers = ["name", "score"]
values = sorted(zip(X_train.columns, xgb.feature_importances_), key=lambda x: x[1] * -1)
xgb_feature_importances = pd.DataFrame(values, columns = headers)

#plot feature importances
```

```
x_pos = np.arange(0, len(xgb_feature_importances))
plt.bar(x_pos, xgb_feature_importances['score'])
plt.xticks(x_pos, xgb_feature_importances['name'])
plt.xticks(rotation=90)
plt.title('Feature importances (XGB)')

plt.show()
```



As we can see from the diagram showing feature importances, the most important features are:

- Customer's account balance,
- Customer's age,
- Number of contacts performed during this campaign and contact duration,
- Number of contacts performed before this campaign.

So the main outcomes of the modelling are:

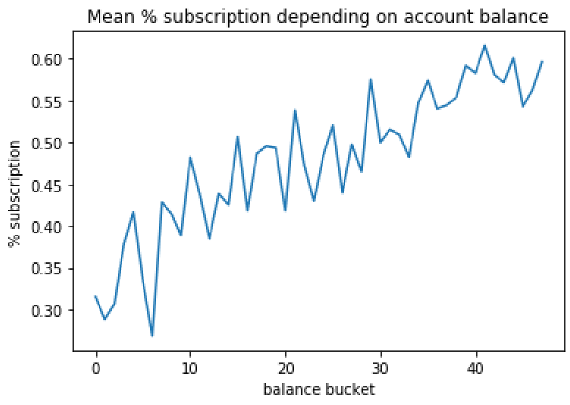
- Customers of greater age are more likely to subscribe for the term deposit.
- Customers with greater account balance are more likely to subscribe for the term deposit.
- Number of contacts with the customers really matters. Too many contacts with the customer could make him decline the offer.

Let's try to make more specific recommendations:

1. Find out account balance, which marketing campaign should focus on:

In [32]:
```
df_new = cleaned_df.copy()

#introduce new column 'balance_buckets' to  ''
df_new['balance_buckets'] = pd.qcut(df_new['balance'], 50, labels=False, duplicates = 'drop')

#group by 'balance_buckets' and find average campaign outcome per balance bucket
mean_deposit = df_new.groupby(['balance_buckets'])['deposit_bool'].mean()

#plot
```

```
#plot
plt.plot(mean_deposit.index, mean_deposit.values)
plt.title('Mean % subscription depending on account balance')
plt.xlabel('balance bucket')
plt.ylabel('% subscription')
plt.show()
```



In [33]:

```
df_new[df_new['balance_buckets'] == 34]['balance'].min()
```

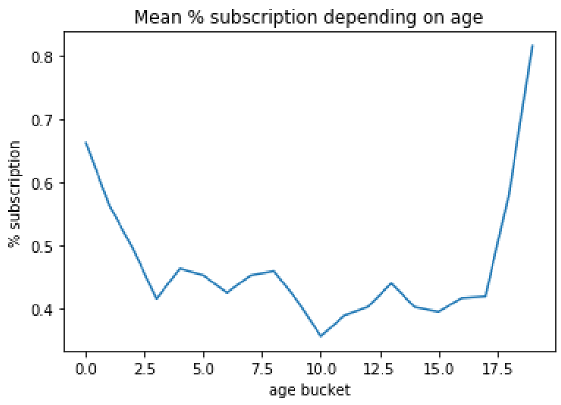
Out[33]:

```
1490
```

From the diagram above we can conclude, that marketing campaigns should concentrate on customers with account balance greater than 1490$.

In [34]:

```
#introduce new column 'age_buckets' to  ''
df_new['age_buckets'] = pd.qcut(df_new['age'], 20, labels=False, duplicates = 'drop')

#group by 'balance_buckets' and find average campaign outcome per balance bucket
mean_age = df_new.groupby(['age_buckets'])['deposit_bool'].mean()

#plot
plt.plot(mean_age.index, mean_age.values)
plt.title('Mean % subscription depending on age')
plt.xlabel('age bucket')
plt.ylabel('% subscription')
plt.show()
```



In [35]:
```
df_new[df_new['age_buckets'] == 3]['age'].max()
```

Out[35]:
```
31
```

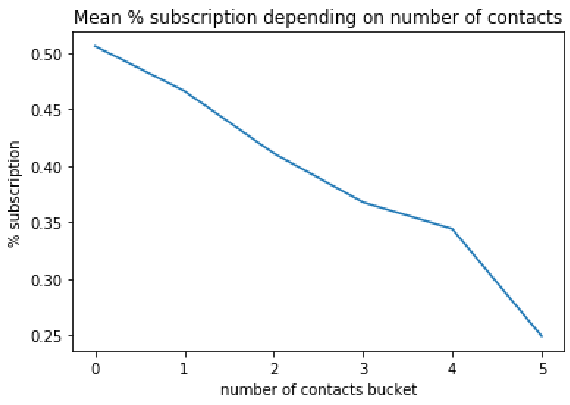
In [36]:
```
df_new[df_new['age_buckets'] == 17]['age'].min()
```

Out[36]:
```
56
```

So we see that average subscrition rate tends to be higher for customers below 31 years old or above 56 years old.

1. Find out appropriate number of contacts with the customer during campaign:

In [37]:
```
#introduce new column 'age_buckets' to  ''
df_new['campaign_buckets'] = pd.qcut(df_new['campaign_cleaned'], 20, labels=False, duplicates =
'drop')

#group by 'balance_buckets' and find average campaign outcome per balance bucket
mean_campaign = df_new.groupby(['campaign_buckets'])['deposit_bool'].mean()

#plot average campaign outcome per bucket
plt.plot(mean_campaign.index, mean_campaign.values)
plt.title('Mean % subscription depending on number of contacts')
plt.xlabel('number of contacts bucket')
plt.ylabel('% subscription')
plt.show()
```



In [38]:
```
df_new[df_new['campaign_buckets'] == 2]['campaign_cleaned'].min()
```

Out[38]:
```
4.0
```

From the plot above we see that average subscription rate is below 50% if the number of contacts during the campaign exceeds 4.

Conclusion

Key outcomes of the analysis are the recommendations for future marketing campaigns:

- The customer's account balance has a huge influence on the campaign's outcome. People with account balance above 1490$ are more likely to subscribe for term deposit, so future address those customers.
- The customer's age affects campaign outcome as well. Future campains should concentrate on customers from age categories below 30 years old and above 50 years old.
- Number of contacts with the customer during the campaign is also very important. The number of contacts with the customer shouldn't exceed 4.

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

14

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 (10)

Sort by

All Comments | Hotness

Click here to comment...

Firat Gonen • Posted on Latest Version • 4 months ago • Options • Reply 1

very clean and organized, simple yet effective

@aleksandradeis thank you very much

Aleksandra Deis Kernel Author • Posted on Latest Version • 4 months ago • Options • Reply 1

Thanks for appreciation @frtgnn !

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

Thats good! Not so huge, but all about the target!

Aleksandra Deis Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Thanks!

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

Thanks for sharing your work! I have learned a lot. I have a question. Would it be better to standardize feature data (such as 'age', 'balance' and 'duration') before training with xgboost ?

Aleksandra Deis Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Hi! Thanks!
I don't think that feature standardization will affect xgboost results much. Algorithms which compute distance or gradients are more sensitive to it.

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

Really good!

Aleksandra Deis Kernel Author • Posted on Latest Version • 5 months ago • Options • Reply 0

Thanks!

Janio Martinez • Posted on Latest Version • 6 months ago • Options • Reply 1

Great work! Interesting analysis on this dataset.

Aleksandra Deis Kernel Author • Posted on Latest Version • 6 months ago • Options • Reply 0

Thanks!

Similar Kernels

Our Team Terms Privacy Contact/Support