BANK MARKETING ANALYSIS

This dataset contains banking marketing campaign data and we can use it to optimize marketing campaigns to attract more customers to term deposit subscription.

In [10]:
```
import numpy as np
import pandas as pd

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

The read.csv() function is used to read data from CSV (Comma Separated Values) files into a DataFrame.

In [11]:
```
df= pd.read_csv("bank.csv")
df.head()
```

Out[11]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previou
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	

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

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.

Clean the data: remove irrelevant columns, deal with missing and incorrect values, turn categorical columns into dummy variables.

In [12]:
```
print("Bank marketing dataset consists of {rows} rows.".format(rows = len(df)))
```

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

In [13]:
```
#find percentage of missing values for each column
missing_values = df.isnull().mean()*100

missing_values.sum()
```

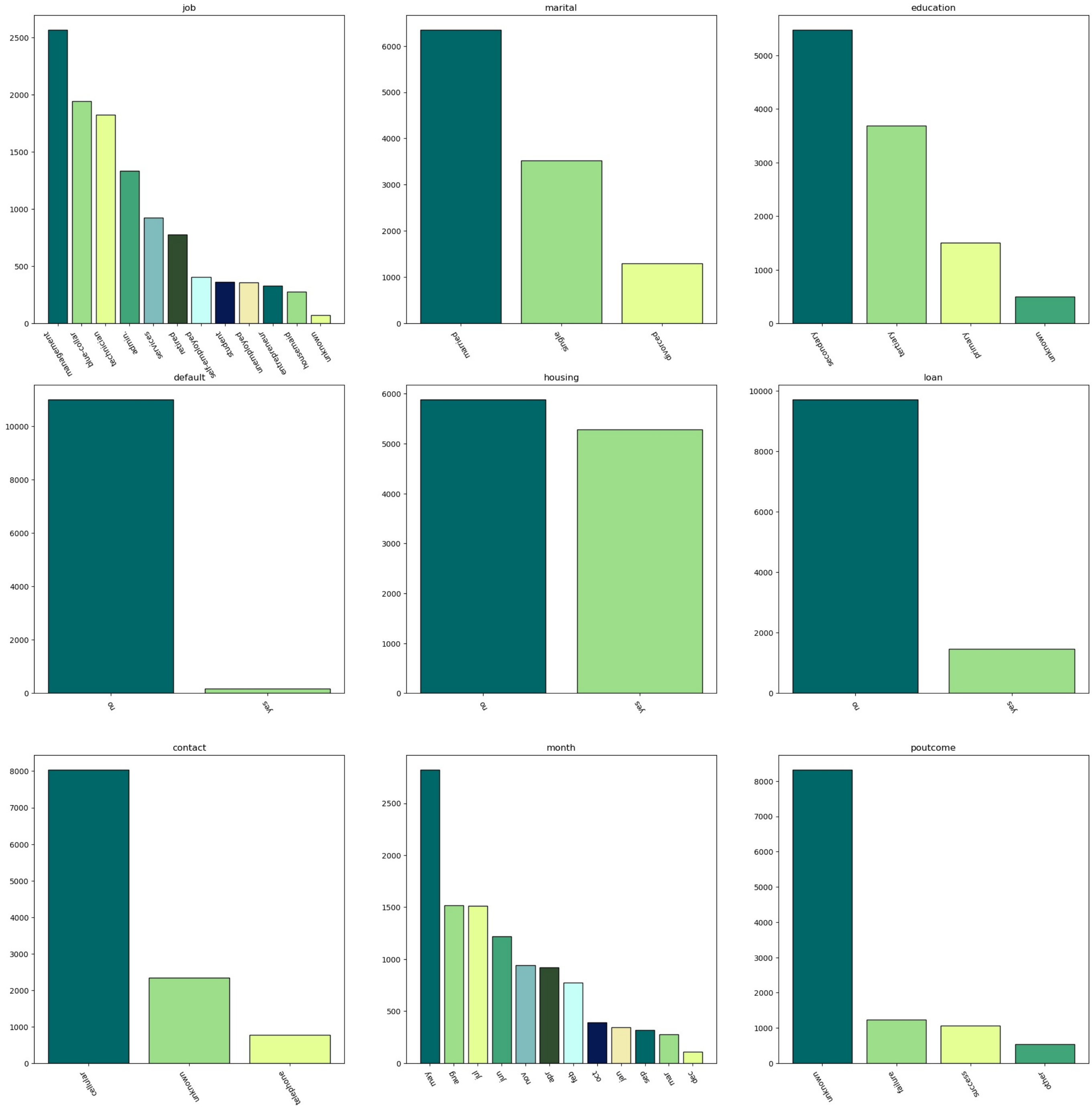
Out[13]:
```
0.0
```

Categorical columns exploration

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

In [17]:
```
# Define colors for each category
colors = ['#006769', '#9DDE8B', '#E6FF94', '#40A578', '#80BCBD', '#304D30', '#C5FFF8', '#071952', '#F3ECB0']

# Define the categorical columns
cat_columns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']

# Create subplots
fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(25, 25))

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))
```

```
    # Use custom colors for each category
    axs[trace_x, trace_y].bar(x_pos, value_counts.values, tick_label = value_counts.index, color=colors[:len(va

    axs[trace_x, trace_y].set_title(cat_column)

    for tick in axs[trace_x, trace_y].get_xticklabels():
        tick.set_rotation(120)

    counter += 1

plt.show()
```



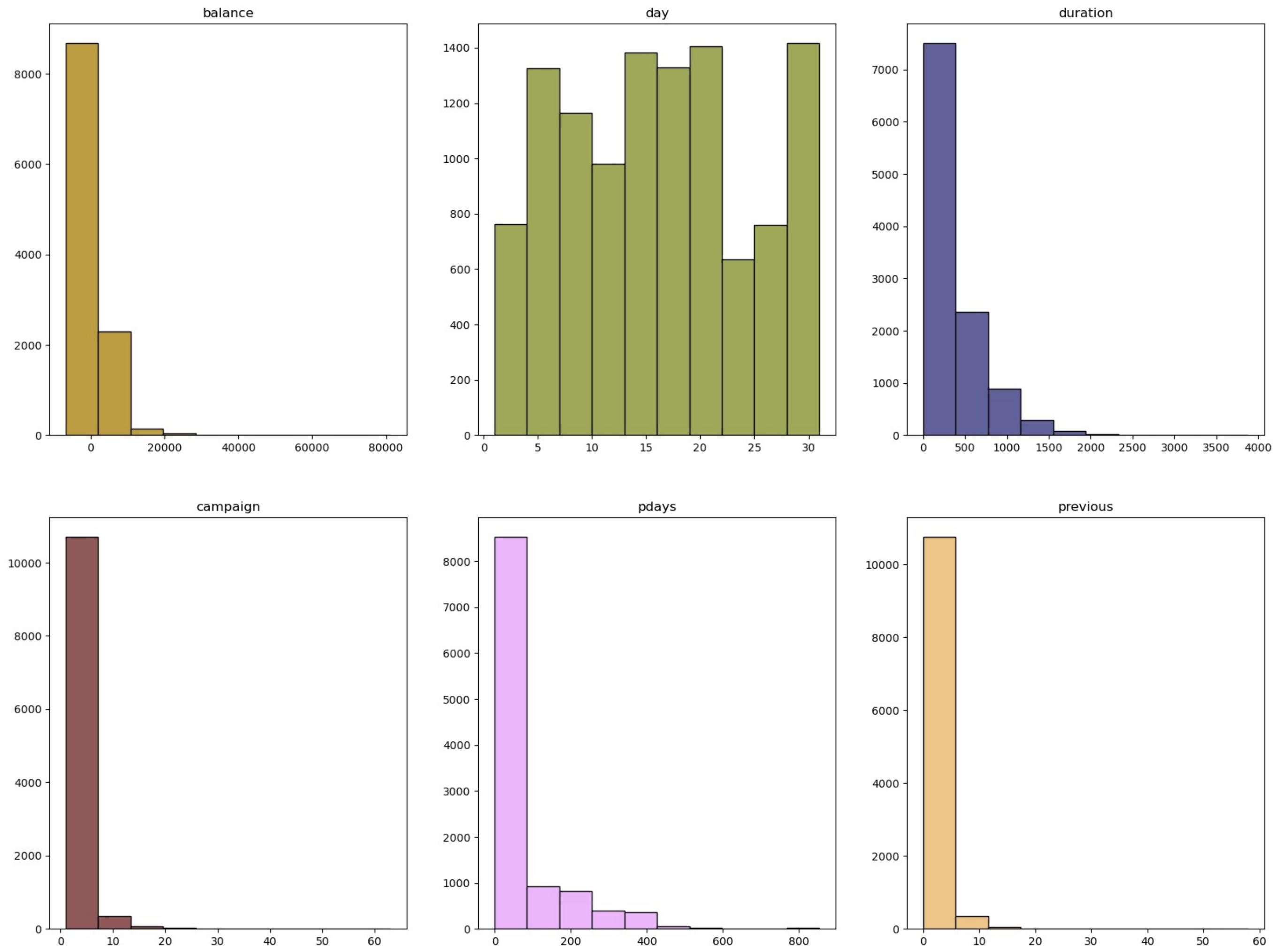
Numerical columns exploration

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

In [31]:

```
num_columns = ['balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
colors = ['#BD9D41', '#9DA958', '#626098', '#8F5859', '#EBB7FA', '#EDC588']  # Define colors

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

counter = 0
for num_column, color in zip(num_columns, colors):  # Iterate over columns and colors

    trace_x = counter // 3
    trace_y = counter % 3

    axs[trace_x, trace_y].hist(df[num_column], color=color,edgecolor = 'black')  # Specify color
```

```
    axs[trace_x, trace_y].set_title(num_column)

    counter += 1

plt.show()
```



It is evident that certain numerical columns (particularly the "pdays," "campaign," and "previous" columns) include outliers. We should examine the data more closely and determine how to manage the noise as there may be inaccurate values there (noisy data).

Let's examine the values of the "pdays," "campaign," and "previous" columns in more detail:

In [7]:
```
df[['pdays', 'campaign', 'previous']].describe()
```

Out[7]:

	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 [8]:
```
len (df[df['pdays'] > 400] ) / len(df) * 100
```

Out[8]:
```
1.2005017022039062
```

Pdays is a variable that stores the number of days that have happened since the client was last contacted during a previous campaign. A closer examination of the 'pdays' data reveals that:

Merely 1.2% of values surpassing 400. Since these values might be outliers, we might think about imputing a different value—possibly the mean value—instead of these. "-One might indicate that no previous contact was made with the client or indicate missing data.

I propose eliminating this column since it contains more than 50% of the entries in the column and we are unsure of its actual meaning.

Percentage of 'previous' values above 20:

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

Out[10]: 0.04479483963447411

The number of contacts made for this client prior to this campaign is stored in the variable "previous" (numeric). I recommend replacing the 'prior' numbers above 34 with average campaign values throughout the data cleaning process because they are also really unusual.

Percentage of 'campaign' values above 20:

In [9]:
```
len (df[df['campaign'] > 34] ) / len(df) * 100
```

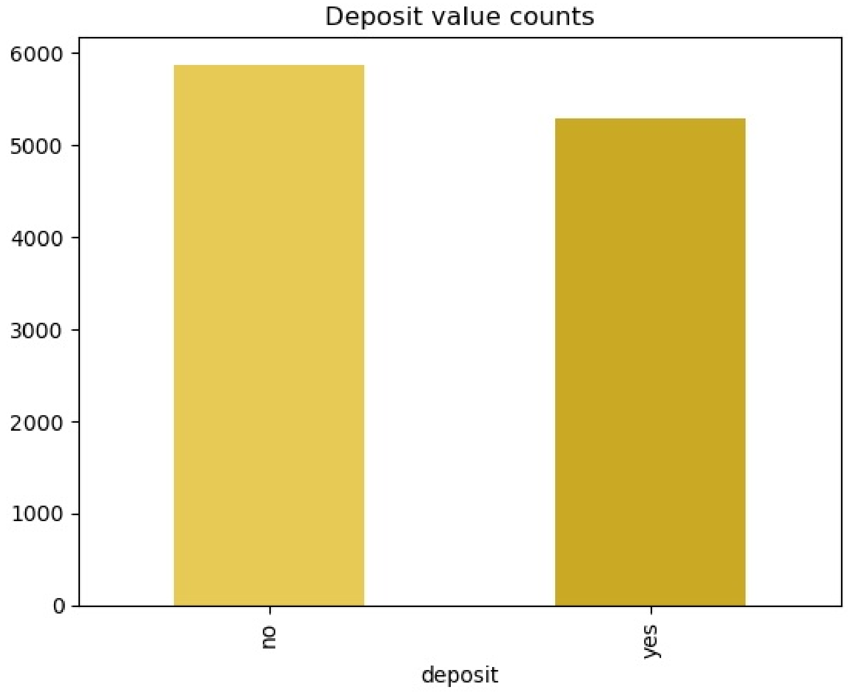
Out[9]: 0.035835871707579285

campaign contains the total number of contacts made for this customer and during this campaign (number, includes final contact) The 'campaign' numbers above 34 are obviously noise, so while cleaning the data, I recommend impute them with average campaign values.

Analysis of the response column

Examining the response column, which contains the data that we will be predicting, is crucial. We ought to examine the 'deposit' column in this instance and contrast its values with those in the other columns. The quantity of "yes" and "no" entries in the answer column "deposit" should be our first point of concern.

In [11]:
```
# Calculate value counts
value_counts = df['deposit'].value_counts()

# Define colors for each category
colors = ['#E7CA55', '#CAA924']

# Plot the bar chart with colors
value_counts.plot.bar(title='Deposit value counts', color=colors)

# Display the plot
plt.show()
```



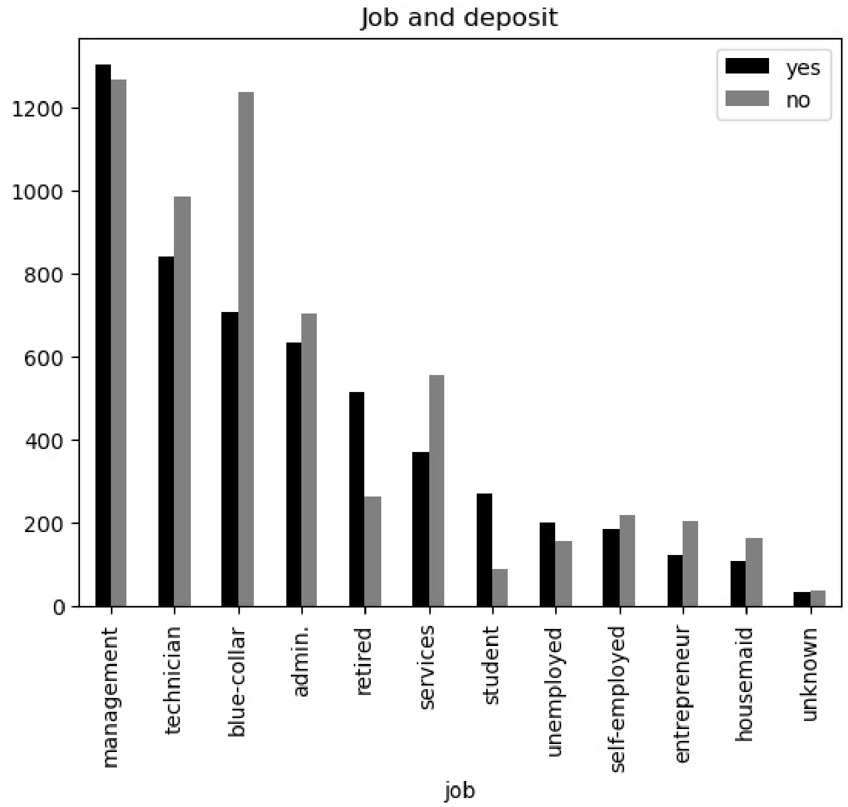
On the diagram we see that counts for 'yes' and 'no' values for 'deposit' are close, so we can use accuracy as a metric for a model, which predicts the campaign outcome.

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

In [12]:
```
j_df = pd.DataFrame()
colors=['black','grey']
```

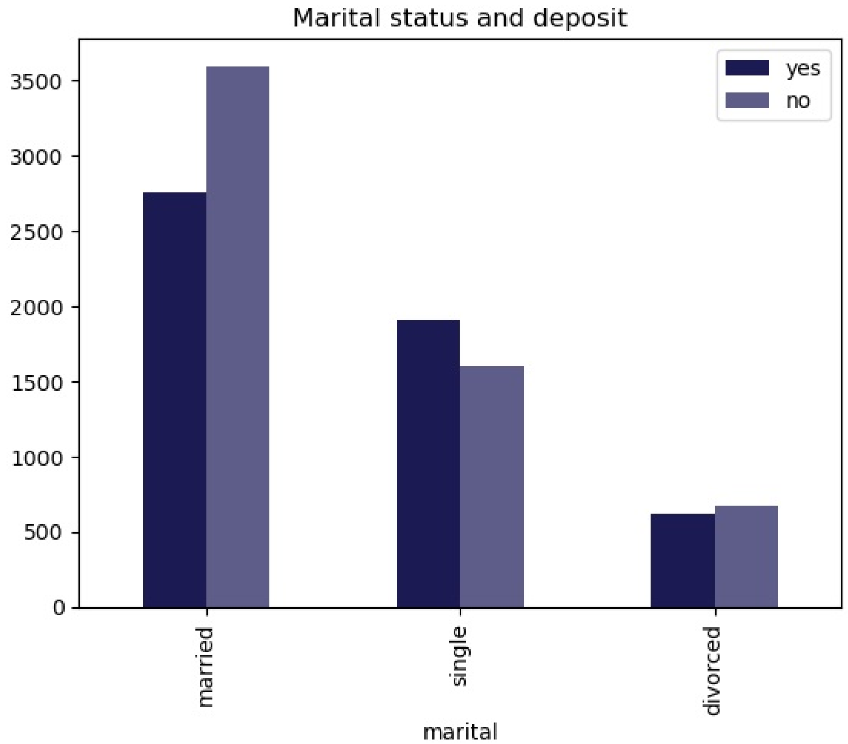
```
j_df['yes'] = df[df['deposit'] == 'yes']['job'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['job'].value_counts()

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

Out[12]: <Axes: title={'center': 'Job and deposit'}, xlabel='job'>



In [13]:
```
j_df = pd.DataFrame()
colors=['#1C1A53','#5E5C88']
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',color=colors)
```

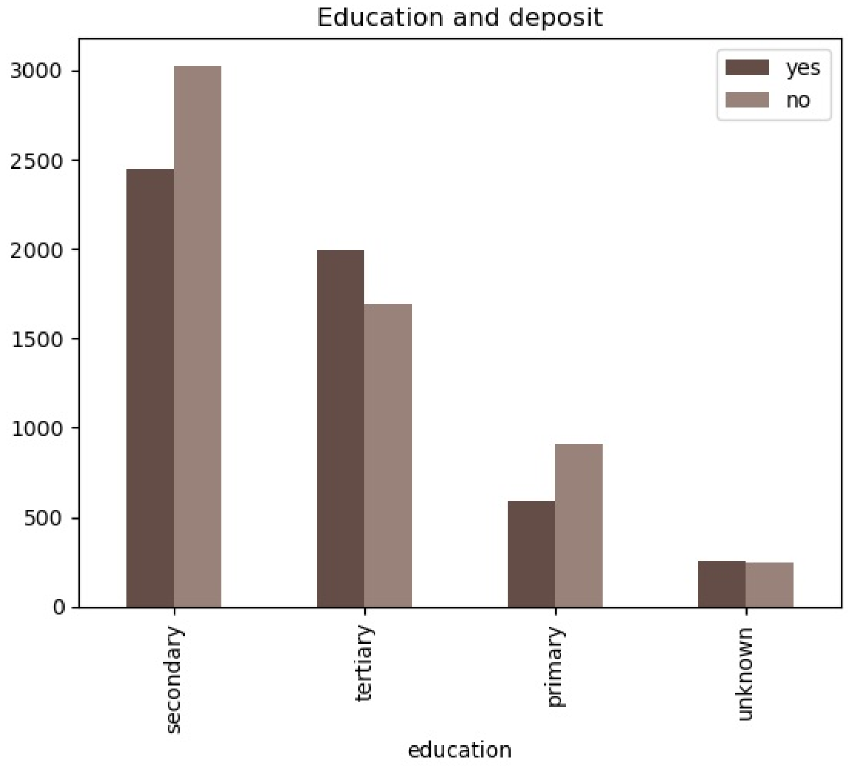
Out[13]: <Axes: title={'center': 'Marital status and deposit'}, xlabel='marital'>



In [14]:
```
j_df = pd.DataFrame()
colors=['#644D46','#99827A']
j_df['yes'] = df[df['deposit'] == 'yes']['education'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['education'].value_counts()
```

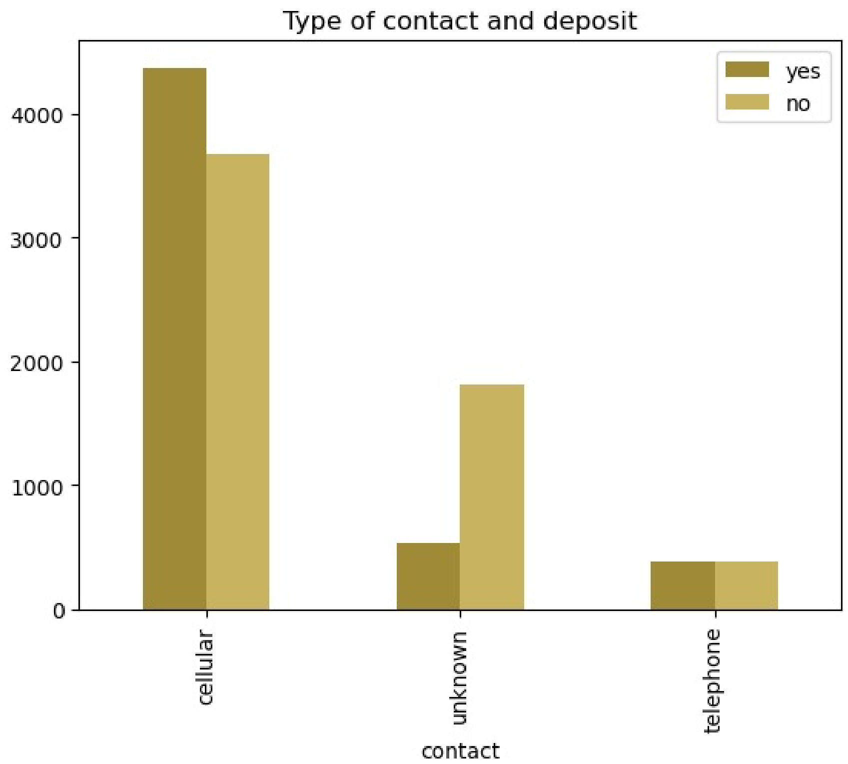
```
j_df.plot.bar(title ='Education and deposit',color=colors)
```

Out[14]: <Axes: title={'center': 'Education and deposit'}, xlabel='education'>



In [15]:
```
j_df = pd.DataFrame()
colors=['#9F8A36','#C8B35E']
j_df['yes'] = df[df['deposit'] == 'yes']['contact'].value_counts()
j_df['no'] = df[df['deposit'] == 'no']['contact'].value_counts()

j_df.plot.bar(title = 'Type of contact and deposit',color=colors)
```

Out[15]: <Axes: title={'center': 'Type of contact and deposit'}, xlabel='contact'>



Based on our dataset, we may infer from the diagrams that:

Term deposit subscriptions are less common among customers with "blue-collar" and "services" jobs. Term deposit subscriptions from married consumers are less common.

Subscribers to term deposits are less likely to have 'cellular' contact information.

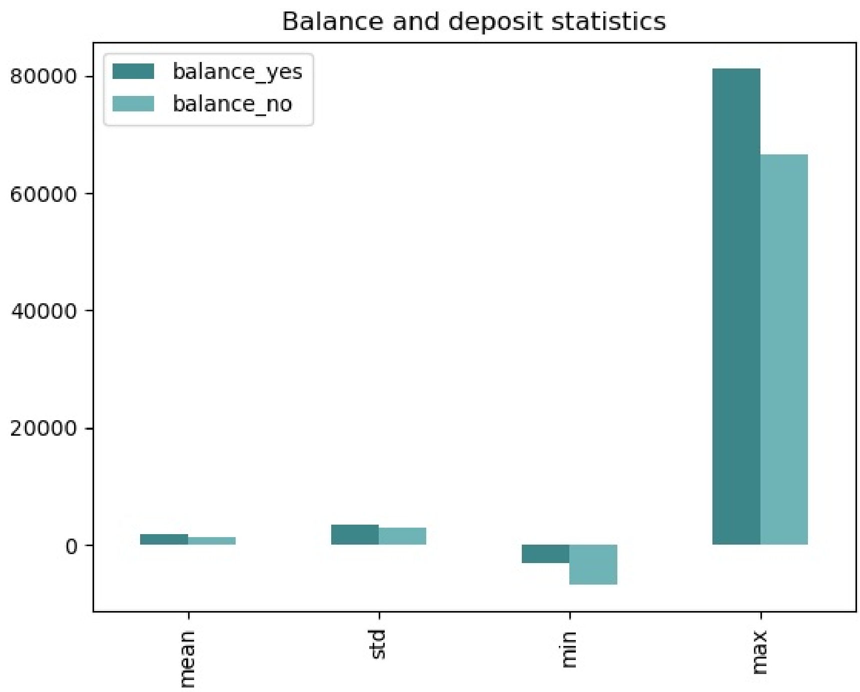
Now let's look how numerical columns affect term deposit subscription

In [19]:
```
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[19]:

	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 [20]:
```
colors = ['#3C8588', '#6EB4B6']
b_df.drop(['count', '25%', '50%', '75%']).plot.bar(title='Balance and deposit statistics', color=colors)
plt.show()
```



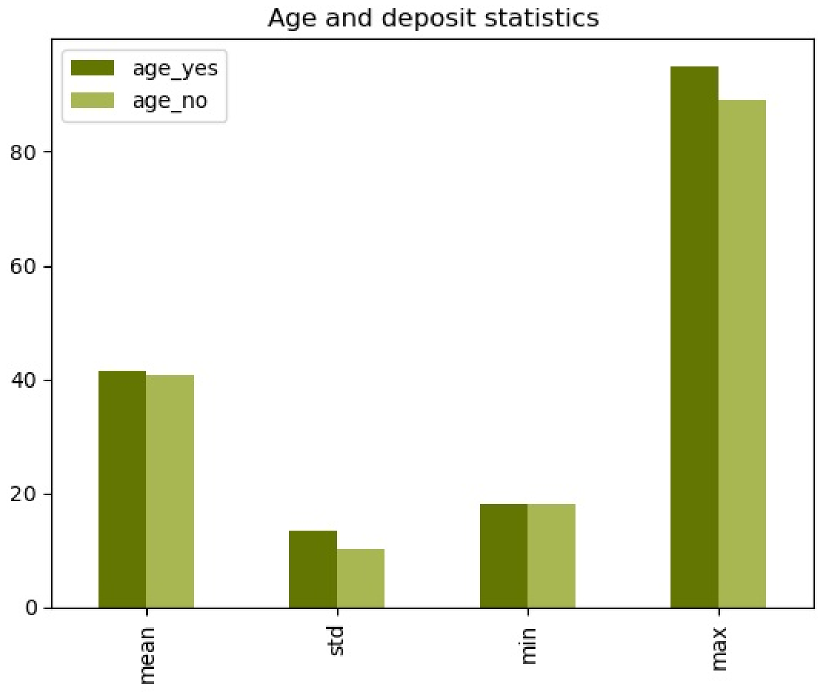
In [21]:
```
a_df = pd.DataFrame()
a_df['age_yes'] = (df[df['deposit'] == 'yes'][['deposit','age']].describe())['age']
a_df['age_no'] = (df[df['deposit'] == 'no'][['deposit','age']].describe())['age']

a_df
```

Out[21]:

	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 [22]:
```
colors=['#647602','#A8B751']
a_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Age and deposit statistics',color=colors)
```

Out[22]: <Axes: title={'center': 'Age and deposit statistics'}>



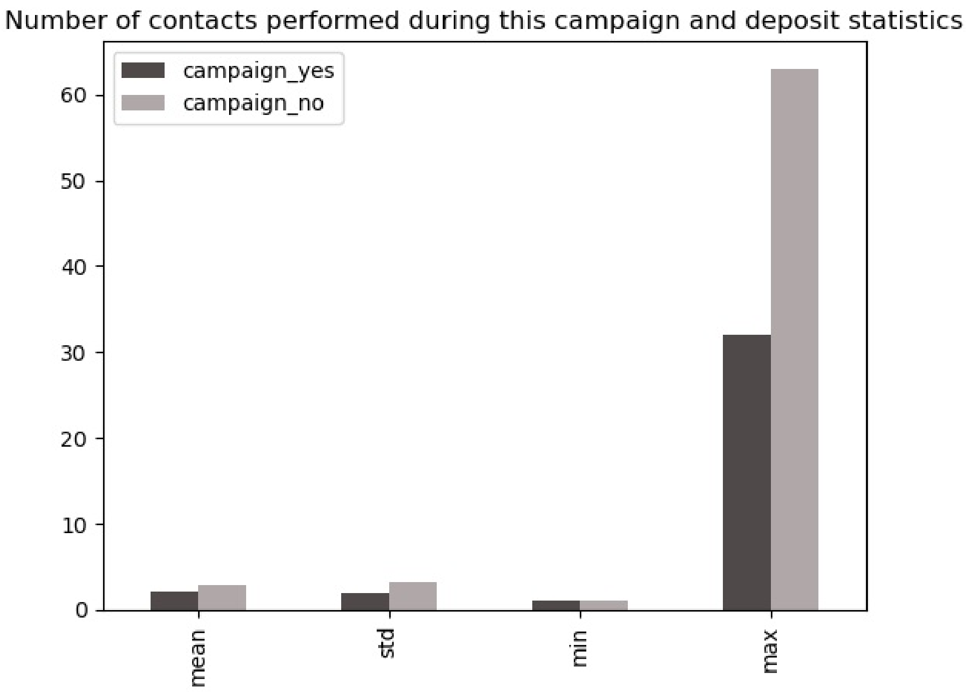
In [23]:
```
c_df = pd.DataFrame()
c_df['campaign_yes'] = (df[df['deposit'] == 'yes'][['deposit','campaign']].describe())['campaign']
c_df['campaign_no'] = (df[df['deposit'] == 'no'][['deposit','campaign']].describe())['campaign']

c_df
```

Out[23]:

	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 [24]:
```
colors=['#4F4949','#B0A8A8']
c_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Number of contacts performed during this campaign ai
```

Out[24]:
```
<Axes: title={'center': 'Number of contacts performed during this campaign and deposit statistics'}>
```



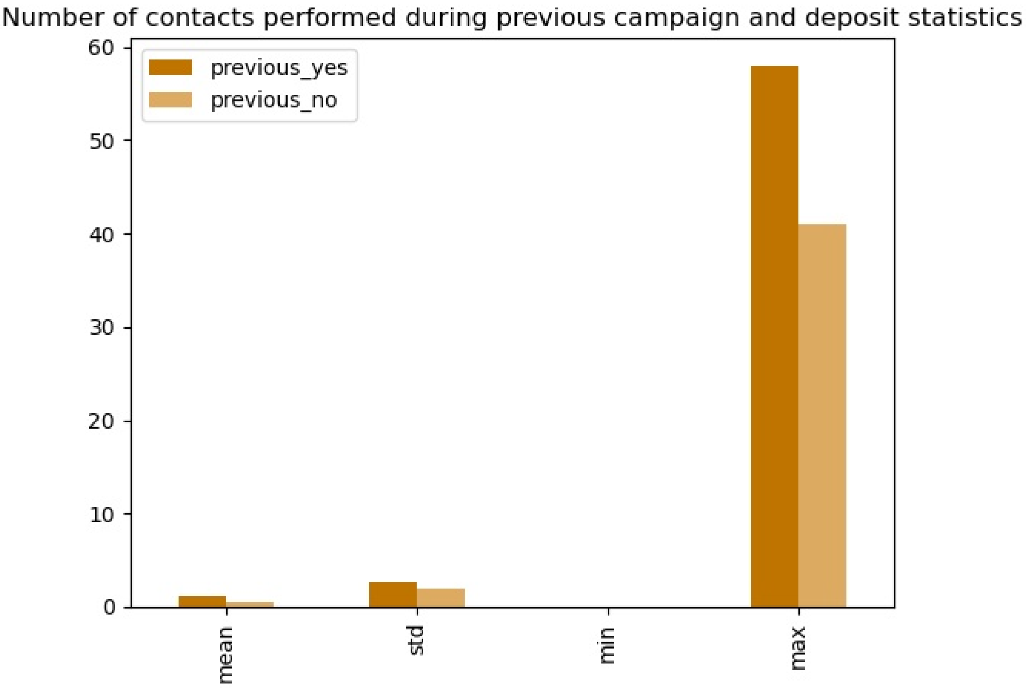
In [18]:
```python
p_df = pd.DataFrame()
p_df['previous_yes'] = (df[df['deposit'] == 'yes'][['deposit','previous']].describe())['previous']
p_df['previous_no'] = (df[df['deposit'] == 'no'][['deposit','previous']].describe())['previous']

p_df
```

Out[18]:

	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 [19]:
```python
colors=['#C07400','#DCAB61']
p_df.drop(['count', '25%', '50%', '75%']).plot.bar(title = 'Number of contacts performed during previous campai
```

Out[19]:
```
<Axes: title={'center': 'Number of contacts performed during previous campaign and deposit statistics'}>
```



Looking at the diagrams above we can conclude that:

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