```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# Ignoring warnings
warnings.filterwarnings('ignore')

# Ensuring plots are displayed inline
%matplotlib inline
```

```
df = pd.read_csv("bank.csv",delimiter=';')
df.rename(columns={'y':'deposit'}, inplace=True)
df.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	p
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	...	2	999	0	no
1	39	services	single	high.school	no	no	no	telephone	may	fri	...	4	999	0	no
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	...	1	999	0	no
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	...	3	999	0	no
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	...	1	999	0	no

5 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4119 entries, 0 to 4118
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             4119 non-null   int64  
 1   job             4119 non-null   object 
 2   marital         4119 non-null   object 
 3   education       4119 non-null   object 
 4   default         4119 non-null   object 
 5   housing         4119 non-null   object 
 6   loan            4119 non-null   object 
 7   contact         4119 non-null   object 
 8   month           4119 non-null   object 
 9   day_of_week     4119 non-null   object 
 10  duration        4119 non-null   int64  
 11  campaign        4119 non-null   int64  
 12  pdays           4119 non-null   int64  
 13  previous        4119 non-null   int64  
 14  poutcome        4119 non-null   object 
 15  emp.var.rate    4119 non-null   float64
 16  cons.price.idx  4119 non-null   float64
 17  cons.conf.idx   4119 non-null   float64
 18  euribor3m       4119 non-null   float64
 19  nr.employed     4119 non-null   float64
 20  deposit         4119 non-null   object 
dtypes: float64(5), int64(5), object(11)
memory usage: 675.9+ KB
```

```
df.tail()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	po
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	...	1	999	0	none
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	...	1	999	0	none
4116	27	student	single	high.school	no	no	no	cellular	may	mon	...	2	999	1	
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	...	1	999	0	none
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	...	1	999	0	none

5 rows × 21 columns

```
df.tail()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	po
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	...	1	999	0	non
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	...	1	999	0	non
4116	27	student	single	high.school	no	no	no	cellular	may	mon	...	2	999	1	
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	...	1	999	0	non
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	...	1	999	0	non

5 rows × 21 columns

```
df.tail()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	po
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	...	1	999	0	non
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	...	1	999	0	non
4116	27	student	single	high.school	no	no	no	cellular	may	mon	...	2	999	1	
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	...	1	999	0	non
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	...	1	999	0	non

5 rows × 21 columns

```
df.dtypes
```

```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
deposit            object
dtype: object
```

```
df.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'deposit'],
      dtype='object')
```

```
df.dtypes
```

```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
```

```
nr.employed       float64
deposit            object
dtype: object
```

```
df.dtypes.value_counts()
```

```
object     11
int64       5
float64     5
Name: count, dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

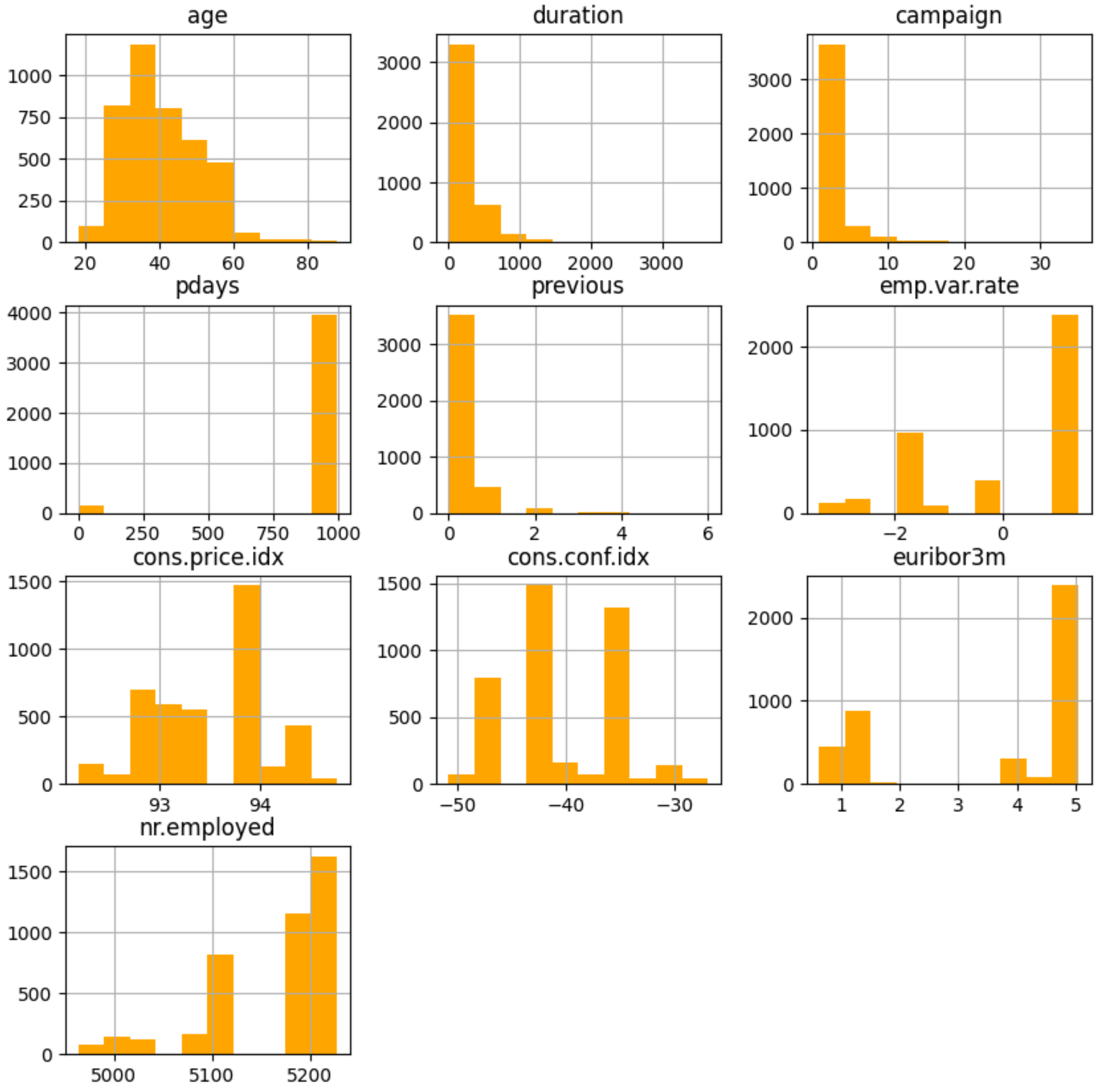
```
df.isna().sum()
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
deposit           0
dtype: int64
```

```
cat_cols = df.select_dtypes(include='object').columns
print(cat_cols)

num_cols = df.select_dtypes(exclude='object').columns
print(num_cols)
```

```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'day_of_week', 'poutcome', 'deposit'],
      dtype='object')
Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed'],
      dtype='object')
```

```
df.hist(figsize=(10,10), color='orange')
plt.show()
```



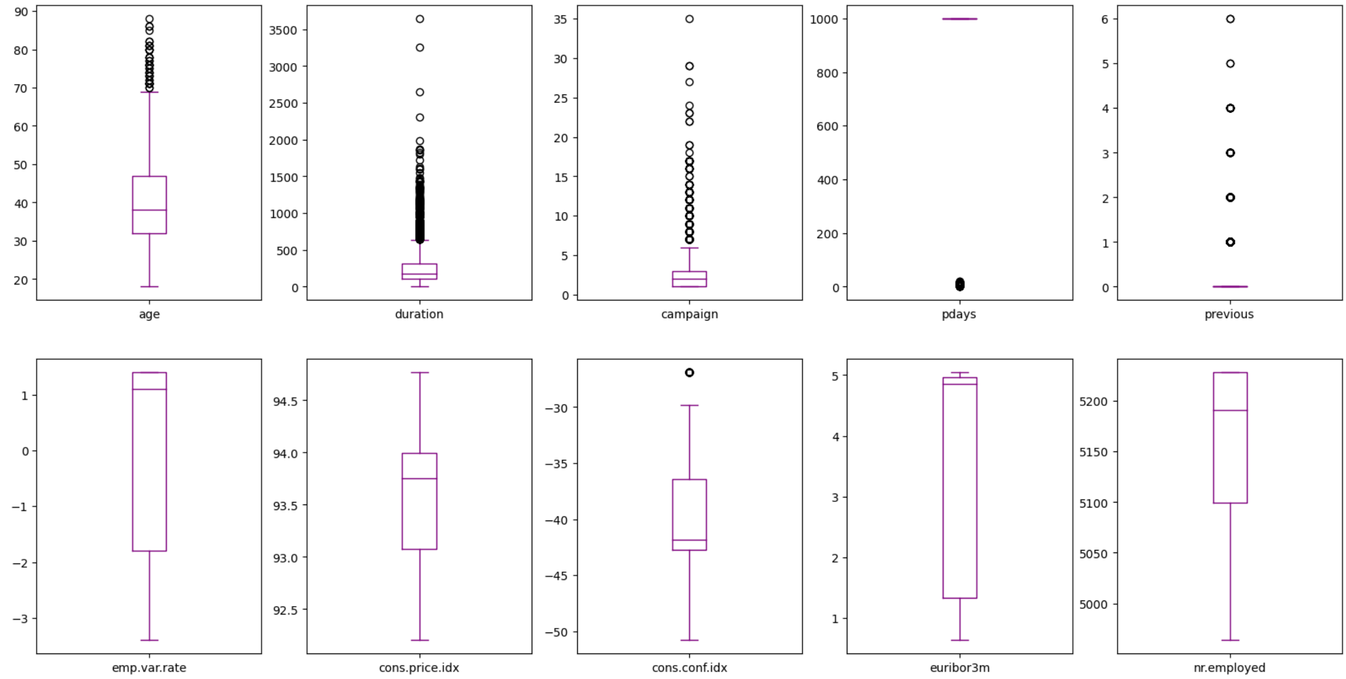

```
df.describe(include='object')
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	deposit
count	4119	4119	4119	4119	4119	4119	4119	4119	4119	4119	4119
unique	12	4	8	3	3	3	2	10	5	3	2
top	admin.	married	university.degree	no	yes	no	cellular	may	thu	nonexistent	no
freq	1012	2509	1264	3315	2175	3349	2652	1378	860	3523	3668

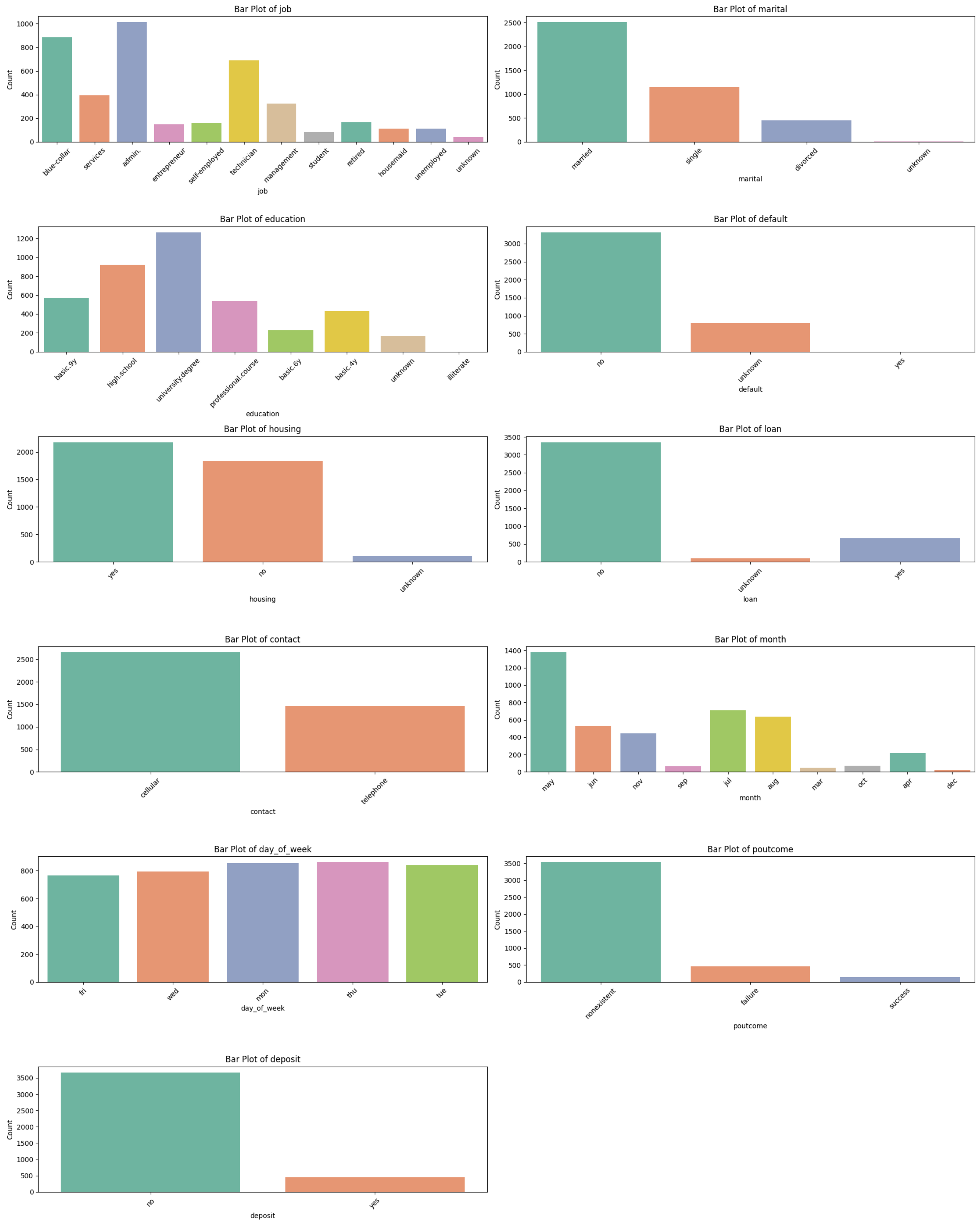
```
df.describe()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.em
count	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.
mean	40.113620	256.788055	2.537266	960.422190	0.190337	0.084972	93.579704	-40.499102	3.621356	5166.
std	10.313362	254.703736	2.568159	191.922786	0.541788	1.563114	0.579349	4.594578	1.733591	73.
min	18.000000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.635000	4963.
25%	32.000000	103.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.334000	5099.
50%	38.000000	181.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.
75%	47.000000	317.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.
max	88.000000	3643.000000	35.000000	999.000000	6.000000	1.400000	94.767000	-26.900000	5.045000	5228.

```
df.plot(kind='box', subplots=True, layout=(2,5), figsize=(20,10), color='purple')
plt.show()
```

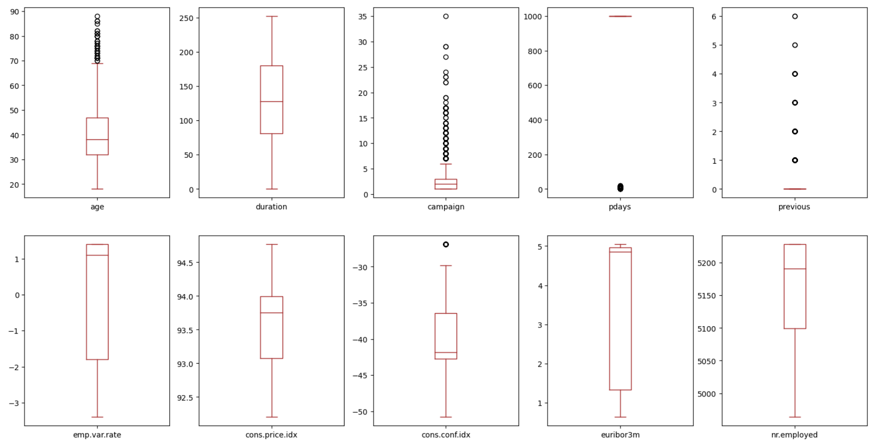



```
# Calculate the number of rows and columns for subplots
num_plots = len(cat_cols)
num_rows = (num_plots + 1) // 2  # Add 1 and divide by 2 to round up for odd numbers
num_cols = 2

# Create a new figure
plt.figure(figsize=(20, 25))  # Adjust the figure size as needed

# Loop through each feature and create a countplot
for i, feature in enumerate(cat_cols, 1):
    plt.subplot(num_rows, num_cols, i)
    sns.countplot(x=feature, data=df, palette='Set2')  # Changed palette to 'Set2'
    plt.title(f'Bar Plot of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.xticks(rotation=45)

# Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()
```


Bar Plot of job
Count
job
blue-collar services admin. entrepreneur self-employed technician management student retired housemaid unemployed unknown
Bar Plot of marital
Count
marital
married single divorced unknown
Bar Plot of education
Count
education
basic.9y high.school university.degree professional.course basic.6y basic.4y unknown illiterate
Bar Plot of default
Count
default
no unknown yes
Bar Plot of housing
Count
housing
yes no unknown
Bar Plot of loan
Count
loan
no unknown yes
Bar Plot of contact
Count
contact
cellular telephone
Bar Plot of month
Count
month
may jun nov sep jul aug mar oct apr dec
Bar Plot of day_of_week
Count
day_of_week
fri wed mon thu tue
Bar Plot of poutcome
Count
poutcome
nonexistent failure success
Bar Plot of deposit
Count
deposit
no yes

```
# Assigning selected columns to a new variable
columns_to_check = df[['age', 'campaign', 'duration']]

# Calculating quartiles and IQR
q1 = np.percentile(columns_to_check, 25)
q3 = np.percentile(columns_to_check, 75)
iqr = q3 - q1

# Calculating lower and upper bounds
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Filtering out values beyond the bounds
df[['age', 'campaign', 'duration']] = columns_to_check[(columns_to_check > lower_bound) & (columns_to_check < upper_bound)]
```

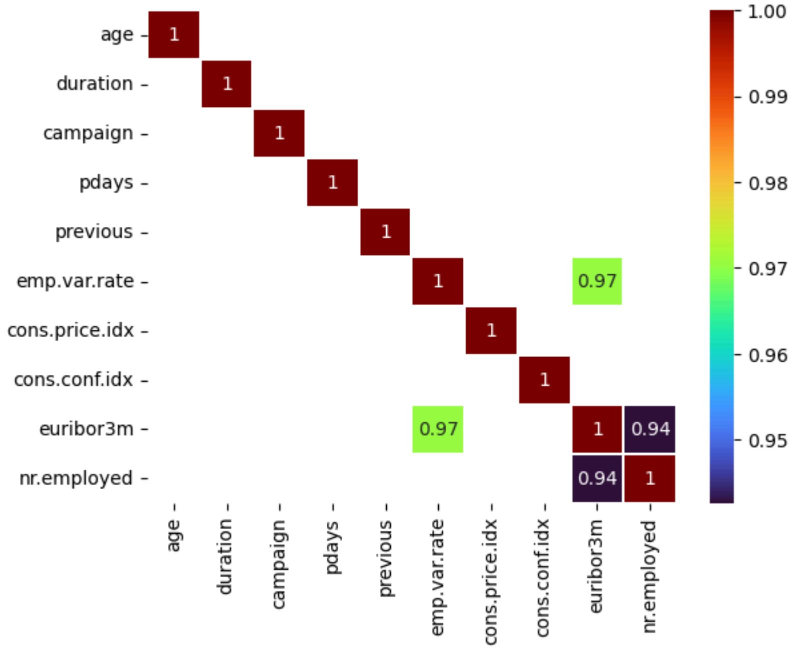
```
df.plot(kind='box', subplots=True, layout=(2,5),figsize=(20,10),color='brown')
plt.show()
```




```
## Exclude non-numeric columns
numeric_df = df.drop(columns=cat_cols)

# Compute the correlation matrix
corr = numeric_df.corr()

# Print the correlation matrix
print(corr)

# Filter correlations with absolute value >= 0.90
corr = corr[abs(corr) >= 0.90]
```

```
sns.heatmap(corr,annot=True,cmap='turbo',linewidths=0.2)
plt.show()
```

```
cons.price.idx -0.000482 -0.013338  0.145021  0.058472 -0.164922
cons.conf.idx   0.098135  0.045889  0.007882 -0.092090 -0.051420
euribor3m      -0.015033 -0.067815  0.159435  0.301478 -0.458851
nr.employed    -0.041936 -0.097339  0.161037  0.381983 -0.514853

                emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
age                -0.019192       -0.000482       0.098135  -0.015033
duration           -0.063870       -0.013338       0.045889  -0.067815
campaign            0.176079        0.145021       0.007882   0.159435
pdays               0.270684        0.058472      -0.092090   0.301478
previous           -0.415238       -0.164922      -0.051420  -0.458851
emp.var.rate        1.000000        0.755155       0.195022   0.970308
cons.price.idx      0.755155        1.000000       0.045835   0.657159
cons.conf.idx       0.195022        0.045835       1.000000   0.276595
euribor3m           0.970308        0.657159       0.276595   1.000000
nr.employed         0.897173        0.472560       0.107054   0.942589

                nr.employed
age               -0.041936
duration          -0.097339
campaign           0.161037
pdays              0.381983
previous          -0.514853
emp.var.rate       0.897173
cons.price.idx     0.472560
cons.conf.idx      0.107054
euribor3m          0.942589
nr.employed        1.000000
```




```
high_corr_cols = ['emp.var.rate','euribor3m','nr.employed']
```

```
df1 = df.copy()
df1.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'deposit'],
      dtype='object')
```

```
df1.drop(high_corr_cols,inplace=True,axis=1)  # axis=1 indicates columns
df1.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
```

```
       'previous', 'poutcome', 'cons.price.idx', 'cons.conf.idx', 'deposit'],
      dtype='object')
```

```
df1.shape
```

```
(4119, 18)
```

```
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()
df_encoded = df1.apply(lb.fit_transform)
df_encoded
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_wee
0	12	1	1	2	0	2	0	0	6	
1	21	7	2	3	0	0	0	1	6	
2	7	7	1	3	0	2	0	1	4	
3	20	7	1	2	0	1	1	1	4	
4	29	0	1	6	0	2	0	0	7	
...	...	...	...	...	...	...	...	...	...	
4114	12	0	1	1	0	2	2	0	3	
4115	21	0	1	3	0	2	0	1	3	
4116	9	8	2	3	0	0	0	0	6	
4117	40	0	1	3	0	0	0	0	1	
4118	16	4	2	3	0	2	0	0	7	

4119 rows × 18 columns

Next steps: Generate code with df_encoded View recommended plots

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

```
deposit
0    3668
1     451
Name: count, dtype: int64
```

```
x = df_encoded.drop('deposit',axis=1)  # independent variable
y = df_encoded['deposit']              # dependent variable
print(x.shape)
print(y.shape)
print(type(x))
print(type(y))
```

```
(4119, 17)
(4119,)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=1)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3089, 17)
(1030, 17)
(3089,)
(1030,)
```

```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

def eval_model(y_test,y_pred):
    acc = accuracy_score(y_test,y_pred)
    print('Accuracy_Score',acc)
    cm = confusion_matrix(y_test,y_pred)
    print('Confusion Matrix\n',cm)
    print('Classification Report\n',classification_report(y_test,y_pred))

def mscore(model):
    train_score = model.score(x_train,y_train)
```

```
    test_score = model.score(x_test,y_test)
    print('Training Score',train_score)
    print('Testing Score',test_score)
```

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(criterion='gini',max_depth=5,min_samples_split=10)
dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_split=10)
```

```
mscore(dt)
```

```
Training Score 0.9148591777274199
Testing Score 0.8990291262135922
```

```
ypred_dt = dt.predict(x_test)
print(ypred_dt)
```

```
[0 0 1 ... 0 0 0]
```

```
eval_model(y_test,ypred_dt)
```

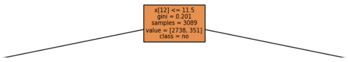
```
Accuracy_Score 0.8990291262135922
Confusion Matrix
 [[905  25]
 [ 79  21]]
Classification Report
               precision    recall  f1-score   support

           0       0.92      0.97      0.95       930
           1       0.46      0.21      0.29       100

    accuracy                           0.90      1030
   macro avg       0.69      0.59      0.62      1030
weighted avg       0.87      0.90      0.88      1030
```

```
from sklearn.tree import plot_tree
```

```
cn = ['no','yes']
fn = x_train.columns
print(fn)
print(cn)
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'cons.price.idx', 'cons.conf.idx'],
      dtype='object')
['no', 'yes']
```

```
plt.figure(figsize=(30,10))
plot_tree(dt,class_names=cn,filled=True)
plt.show()
```



```
dt1 = DecisionTreeClassifier(criterion='entropy',max_depth=4,min_samples_split=15)
dt1.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_split=15)
```

```
mscore(dt1)
```

```
Training Score 0.9080608611201036
Testing Score 0.9048543689320389
```

```
ypred_dt1 = dt1.predict(x_test)
```

```
eval_model(y_test,ypred_dt1)
```

```
Accuracy_Score 0.9048543689320389
Confusion Matrix
 [[915  15]
 [ 83  17]]
Classification Report
               precision    recall  f1-score   support

           0       0.92      0.98      0.95       930
           1       0.53      0.17      0.26       100

    accuracy                           0.90      1030
   macro avg       0.72      0.58      0.60      1030
weighted avg       0.88      0.90      0.88      1030
```

```
plt.figure(figsize=(40,20))
plot_tree(dt1,class_names=cn,filled=True)
plt.show()
```