

Bank Telemarketing Classification using XGBoost and Oversampling

- The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).
- The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

The task at hand is a binary classification problem, where we need to predict whether a client will subscribe to a bank term deposit or not, based on the features related to the marketing campaign. Here's a high-level approach to building a classification algorithm for this task:

- Data Preprocessing
- Exploratory Data Analysis (EDA)
- Model Selection
- Error Analysis
- Model Tuning
- Result

```
! pip install xgboost
! pip install imbalanced-learn
```

Collecting xgboost

Downloading xgboost-1.7.4-py3-none-manylinux2014_x86_64.whl (193.6 MB)

193.6/193.6 MB 12.5 MB/s eta 0:00:00

Requirement already satisfied: scipy in /shared-lib/python3.9/py/lib/python3.9/site-packages (fr

Requirement already satisfied: numpy in /shared-lib/python3.9/py/lib/python3.9/site-packages (fr

Installing collected packages: xgboost

Successfully installed xgboost-1.7.4

WARNING: You are using pip version 22.0.4; however, version 23.0.1 is available.

You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command

Collecting imbalanced-learn

Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)

226.0/226.0 KB 27.9 MB/s eta 0:00:00

Requirement already satisfied: joblib>=1.1.1 in /shared-lib/python3.9/py/lib/python3.9/site-packa

Requirement already satisfied: scipy>=1.3.2 in /shared-lib/python3.9/py/lib/python3.9/site-packag

02/04/2023, 18:34


Bank Classification

Requirement already satisfied: numpy>=1.17.3 in /shared-libs/python3.9/py/lib/python3.9/site-packa
Requirement already satisfied: scikit-learn>=1.0.2 in /shared-libs/python3.9/py/lib/python3.9/site
Requirement already satisfied: threadpoolctl>=2.0.0 in /shared-libs/python3.9/py/lib/python3.9/sit
Installing collected packages: imbalanced-learn
Successfully installed imbalanced-learn-0.10.1
WARNING: You are using pip version 22.0.4; however, version 23.0.1 is available.
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command

```
# Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from imblearn.over_sampling import RandomOverSampler
```

Date Preprocessing

- Load the data and perform any necessary data cleaning and preprocessing steps such as handling missing values, converting categorical variables to numerical variables, and scaling numerical variables.

SQL	Saved to variable df					
<pre>SELECT * FROM 'bank-additional-full.csv'</pre>						
	<div>age int64</div> <div>17 - 98</div> <div></div>	<div>job object</div> <div>admin. 25.3%</div> <div>blue-collar 22.5%</div> <div>10 others 52.2%</div>	<div>marital object</div>	<div>education object</div>	<div>default object</div>	<div>household object</div>
0	56	housemaid	married	basic.4y	no	no
1	57	services	married	high.school	unknown	no
2	37	services	married	high.school	no	yes
3	40	admin.	married	basic.6y	no	no
4	56	services	married	high.school	no	no
5	45	services	married	basic.9y	unknown	no
6	59	admin.	married	professional.course	no	no
7	41	blue-collar	married	unknown	unknown	no
8	24	technician	single	professional.course	no	yes

9	25	services	single	high.school	no	yes
---	----	----------	--------	-------------	----	-----

Check the dimensions of the dataset. Row and columns present
df.shape

(41188, 21)

View the first few rows of the dataset and get an idea of what the data looks like.
df.head()

	age int64	job object	marital object	education object	default object	household
0	56	housemaid	married	basic.4y	no	no
1	57	services	married	high.school	unknown	no
2	37	services	married	high.school	no	yes
3	40	admin.	married	basic.6y	no	no
4	56	services	married	high.school	no	no

Get some summary statistics for the numerical variables in the dataset.
df.describe()

	age float64	duration float64	campaign float64	pdays float64	previous float64	employed
count	41188.0	41188.0	41188.0	41188.0	41188.0	
mean	40.02406040594348	258.2850101971448	2.567592502670681	962.4754540157328	0.17296299893172767	0.08061575793895108
std	10.421249980934048	259.2792488364648	2.7700135429023276	186.9109073447418	0.4949010798392897	1.571754687112677
min	17.0	0.0	1.0	0.0	0.0	
25%	32.0	102.0	1.0	999.0	0.0	
50%	38.0	180.0	2.0	999.0	0.0	
75%	47.0	319.0	3.0	999.0	0.0	
max	98.0	4918.0	56.0	999.0	7.0	

```
# Check if any of the columns have missing data.  
df.isnull().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  
y            0  
dtype: int64
```

```
# Explore the distribution of the target variable 'y' to see how many samples belong to each  
df['y'].value_counts()
```

```
no      36548  
yes     4640  
Name: y, dtype: int64
```

Note :

- There are no missing values.
- There are substantially more 'No' than 'Yes' creating an unbalanced dataset. Will come back to deal with it later.

Exploratory Data Analysis (EDA)

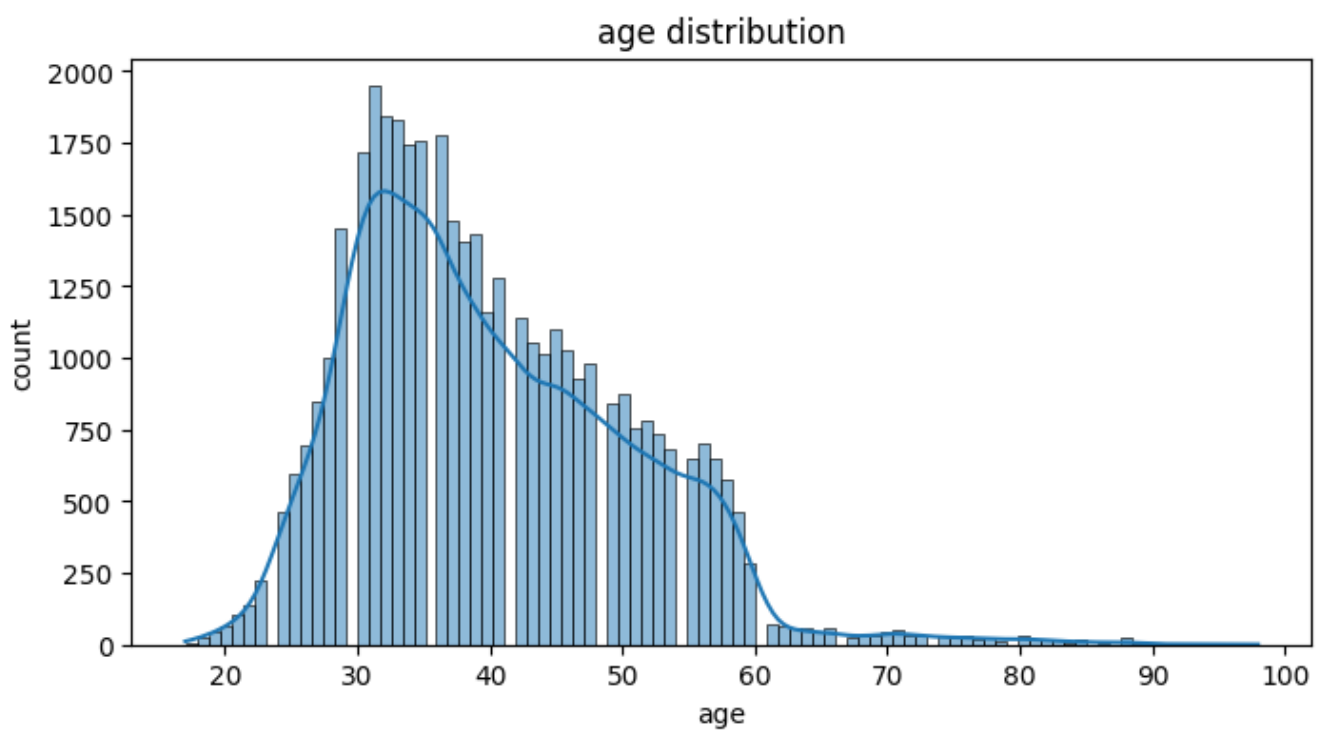
- to perform some exploratory data analysis (EDA) to understand the distribution of the data and identify any patterns or trends.

```
# Create a list of numerical column names  
num_cols = ['age', 'duration', 'campaign', 'pdays', 'previous',  
            'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',  
            'euribor3m', 'nr.employed']
```

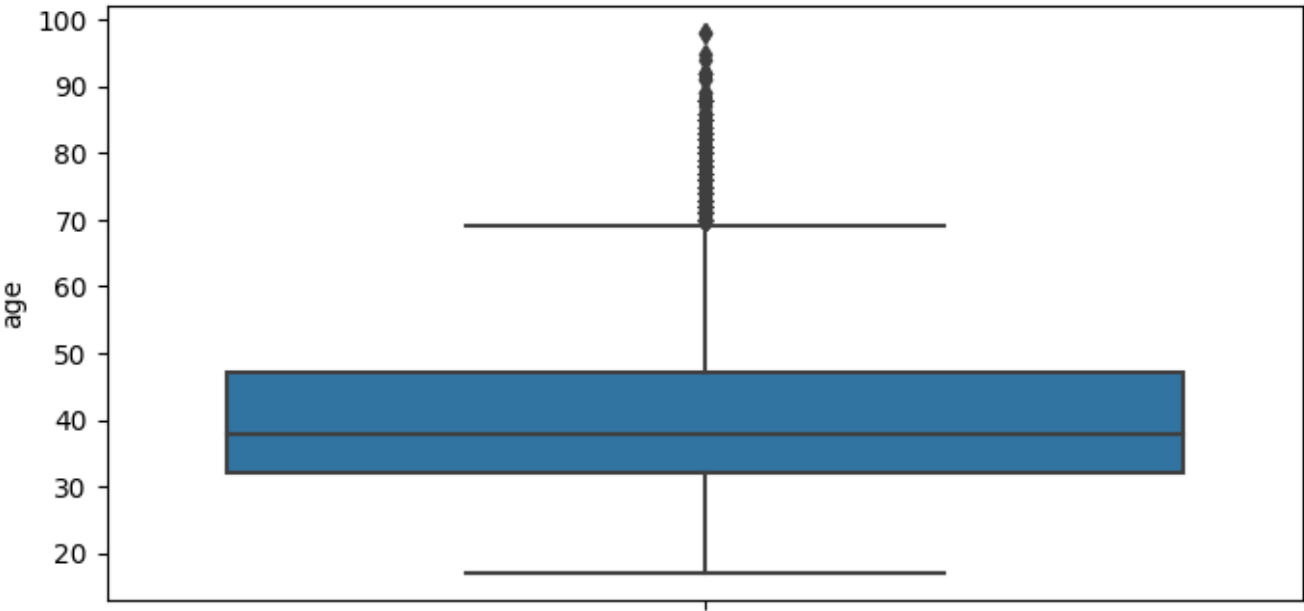
Loop over each numerical column and plot a histogram and boxplot

```
for col in num_cols:
    # Histogram
    plt.figure(figsize=(8, 4))
    sns.histplot(data=df, x=col, kde=True)
    plt.title(f'{col} distribution')
    plt.xlabel(col)
    plt.ylabel('count')
    plt.show()

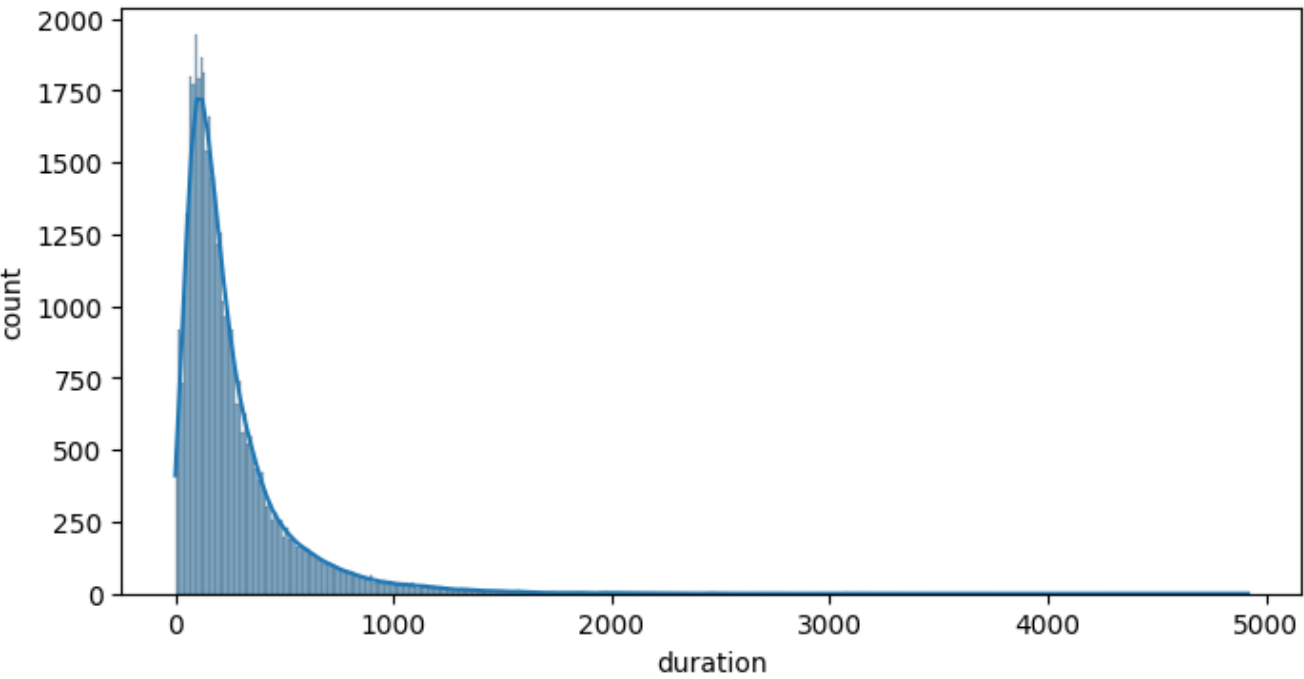
    # Boxplot
    plt.figure(figsize=(8, 4))
    sns.boxplot(data=df, y=col)
    plt.title(f'{col} distribution')
    plt.ylabel(col)
    plt.show()
```



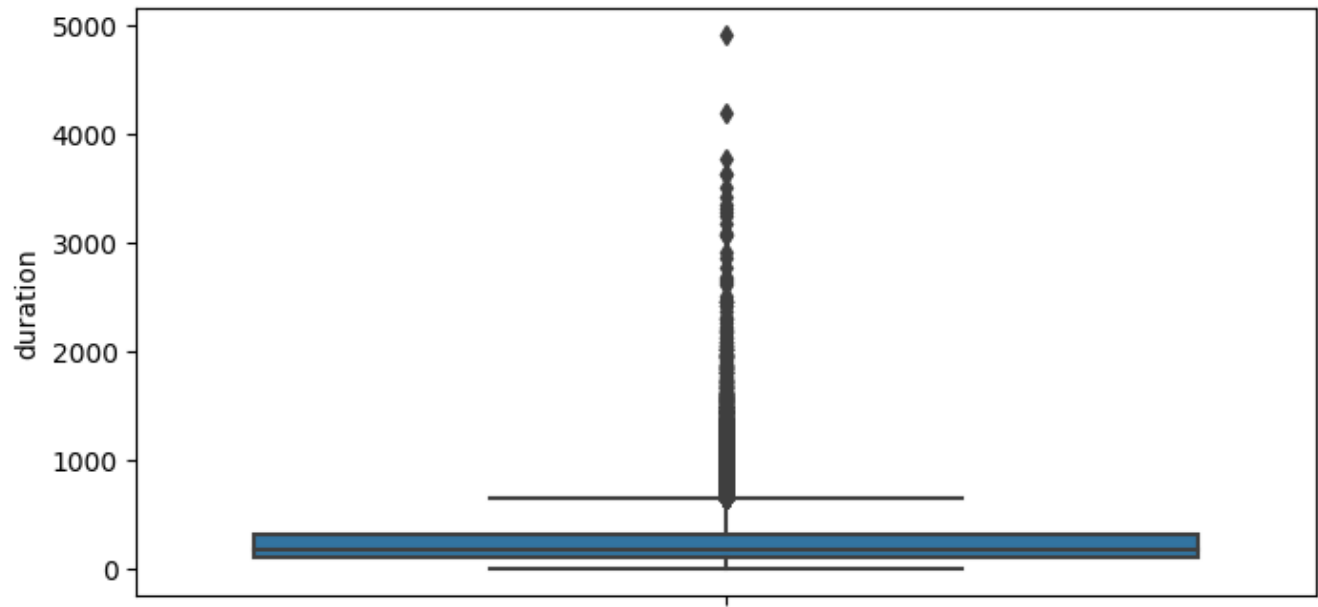
age distribution



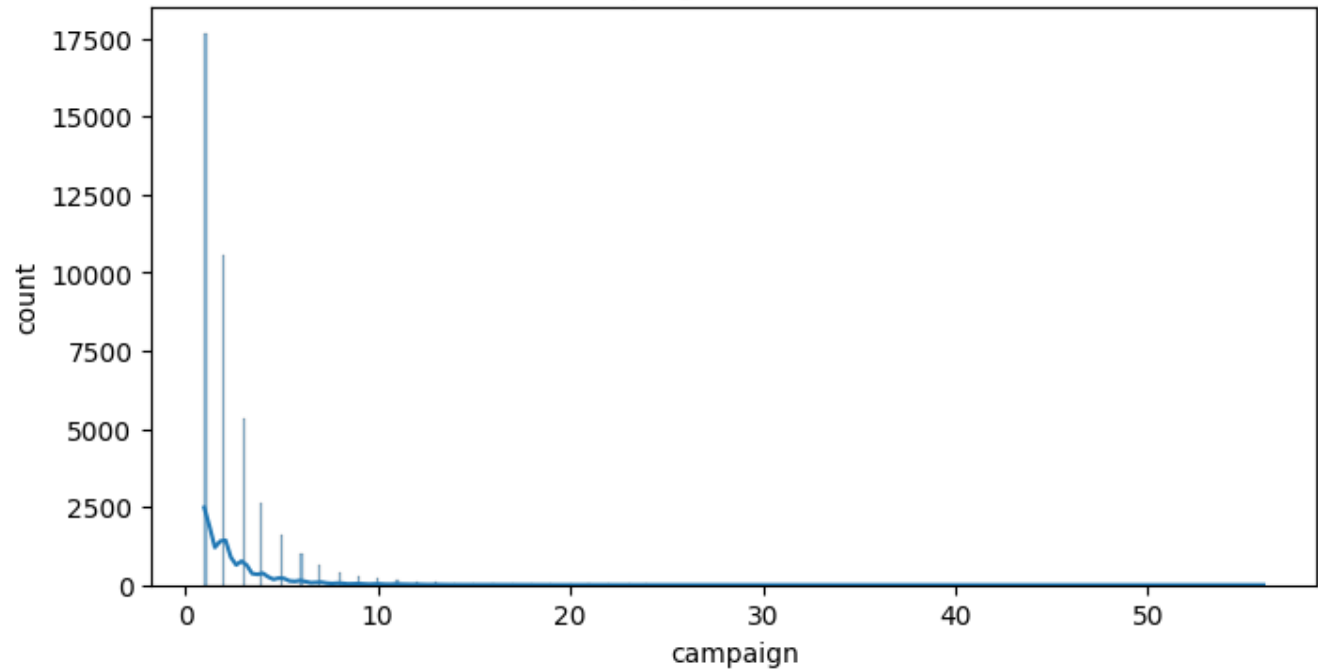
duration distribution

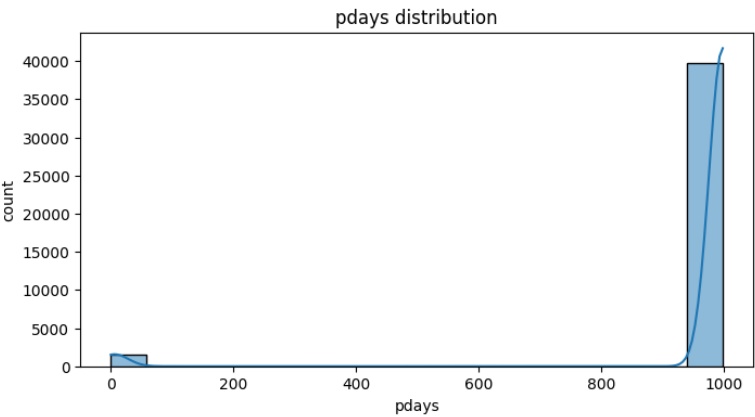
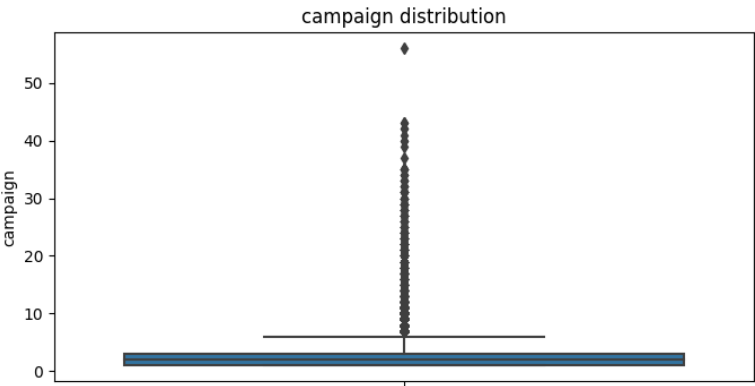


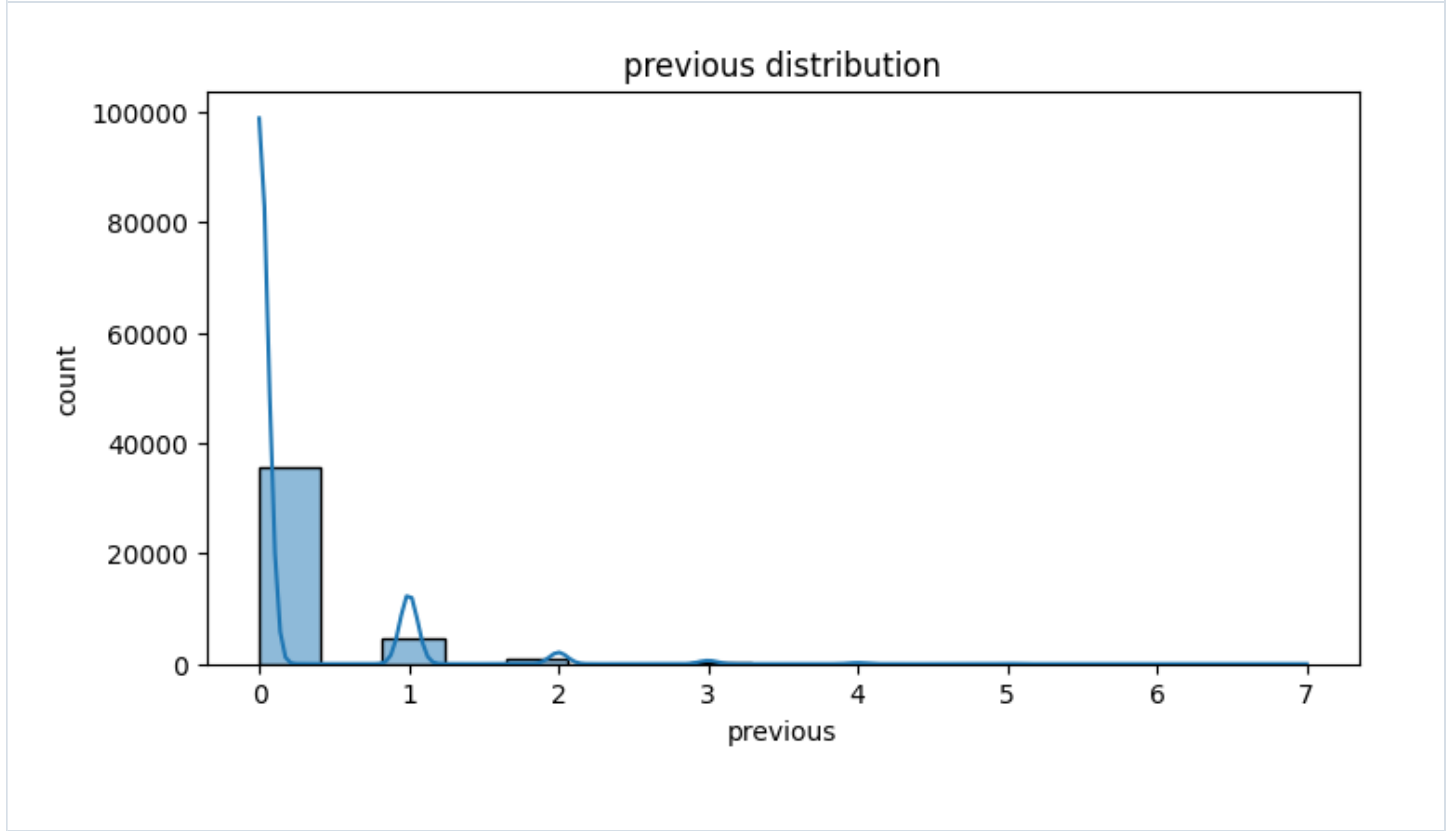
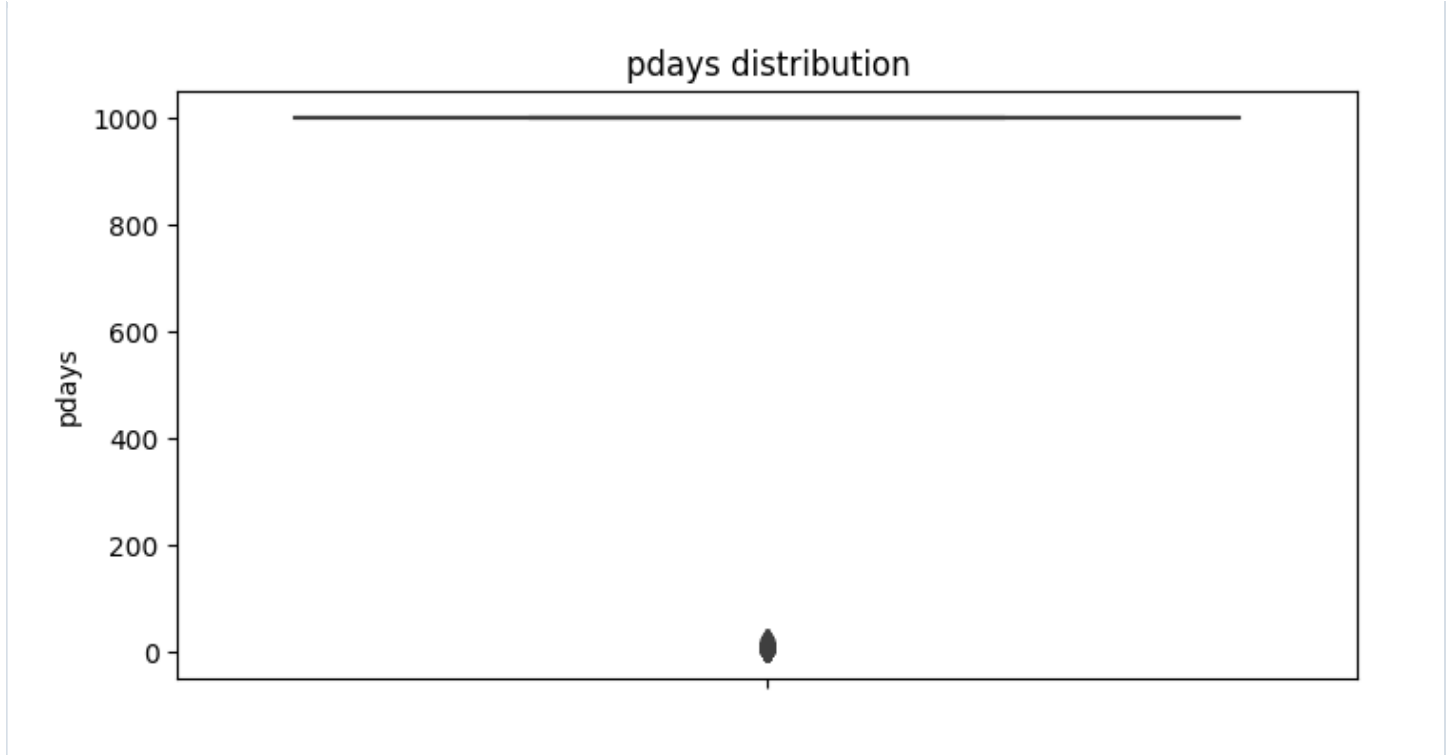
duration distribution

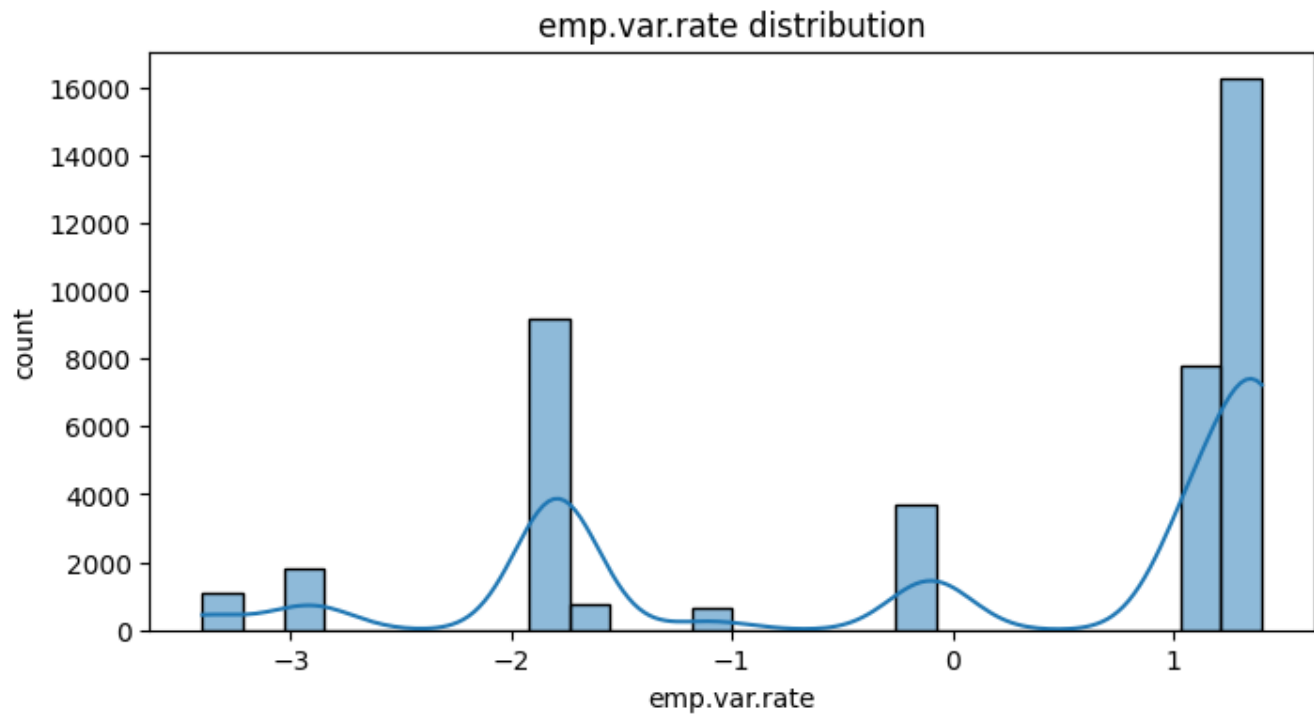
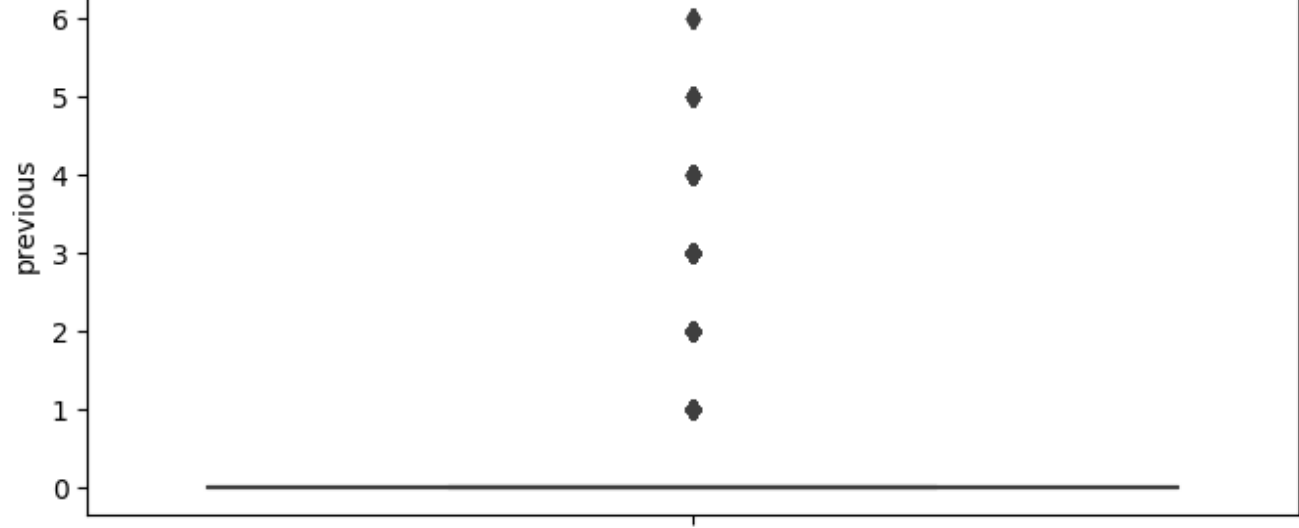


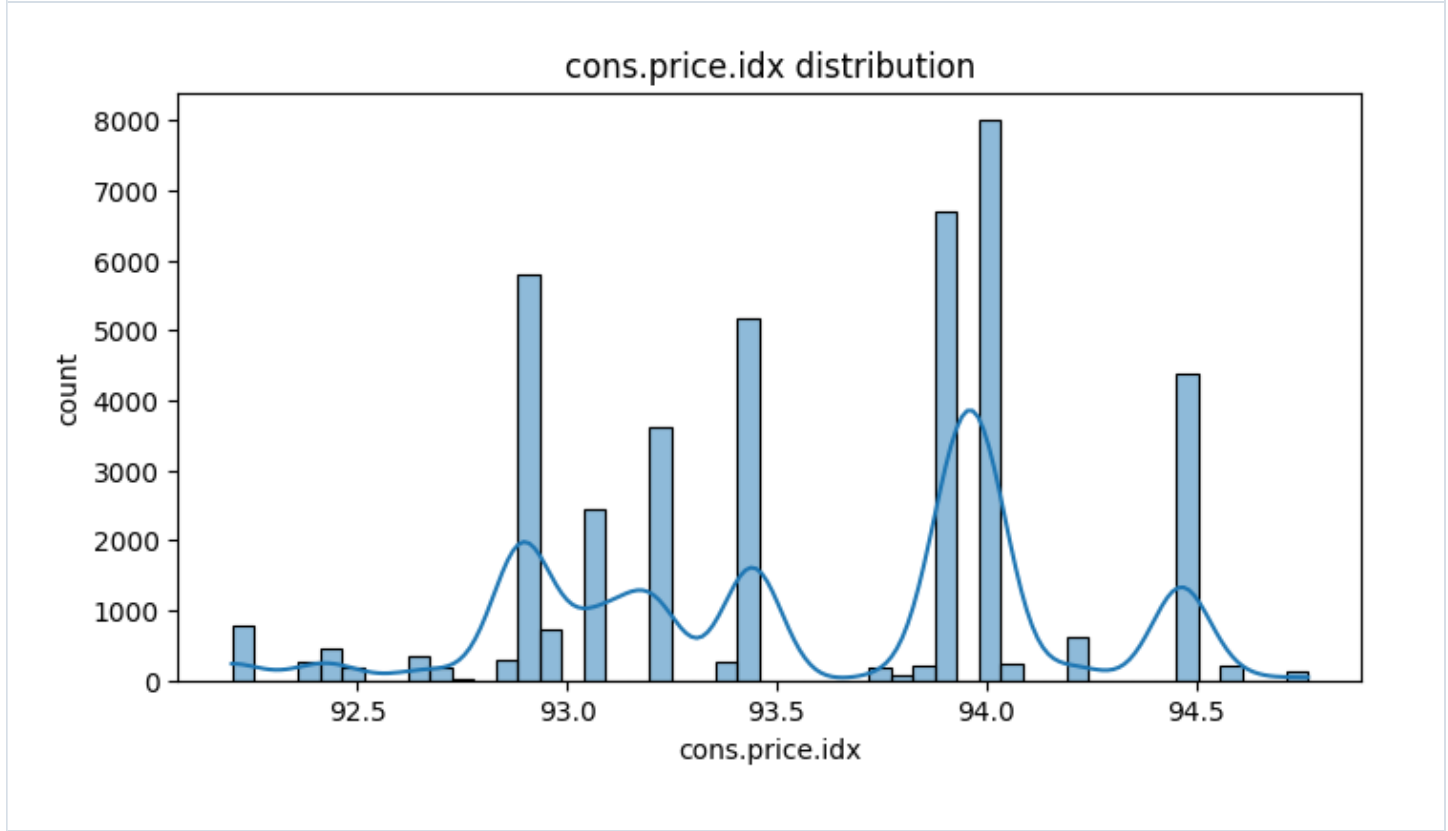
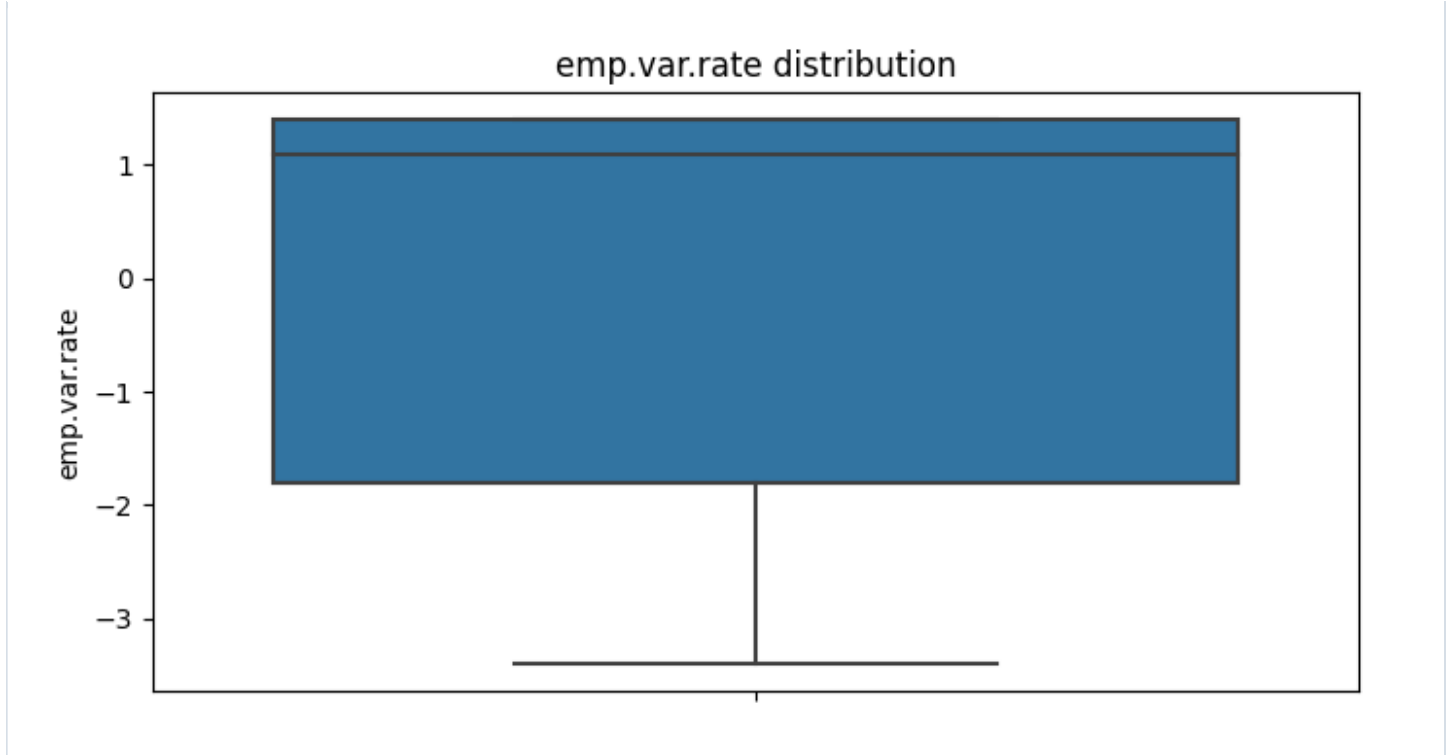
campaign distribution

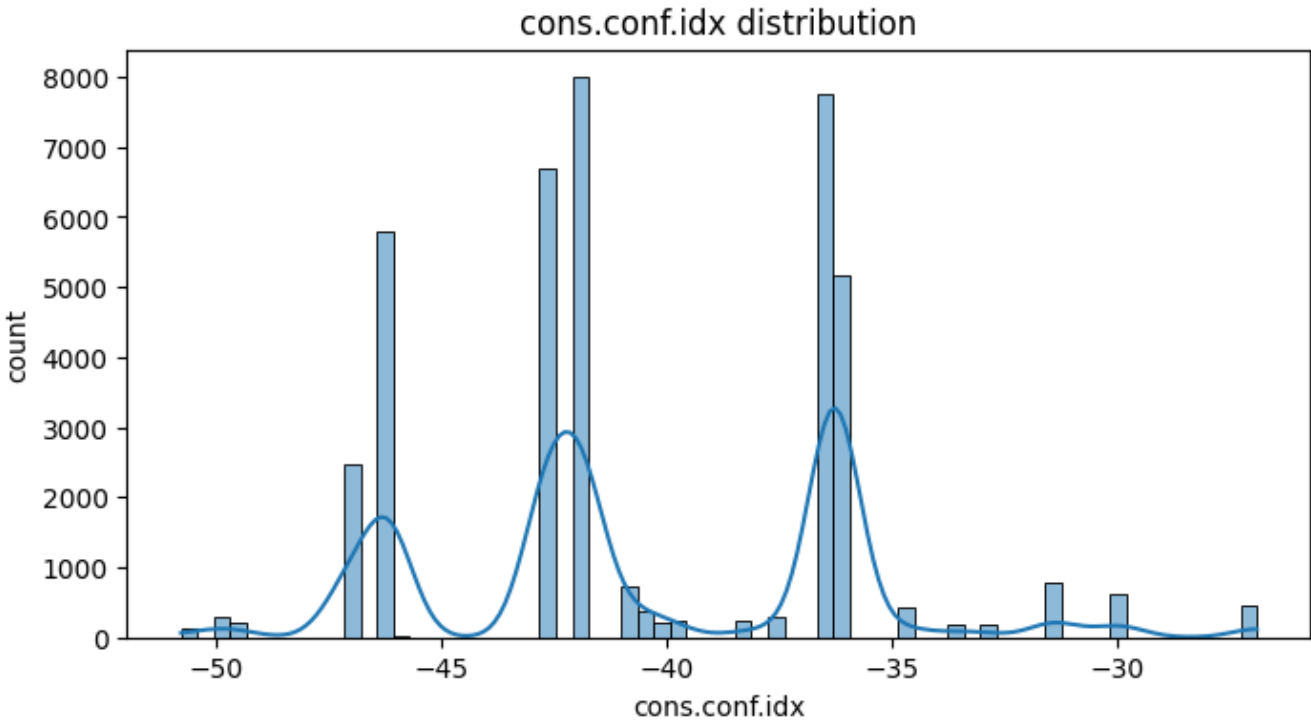
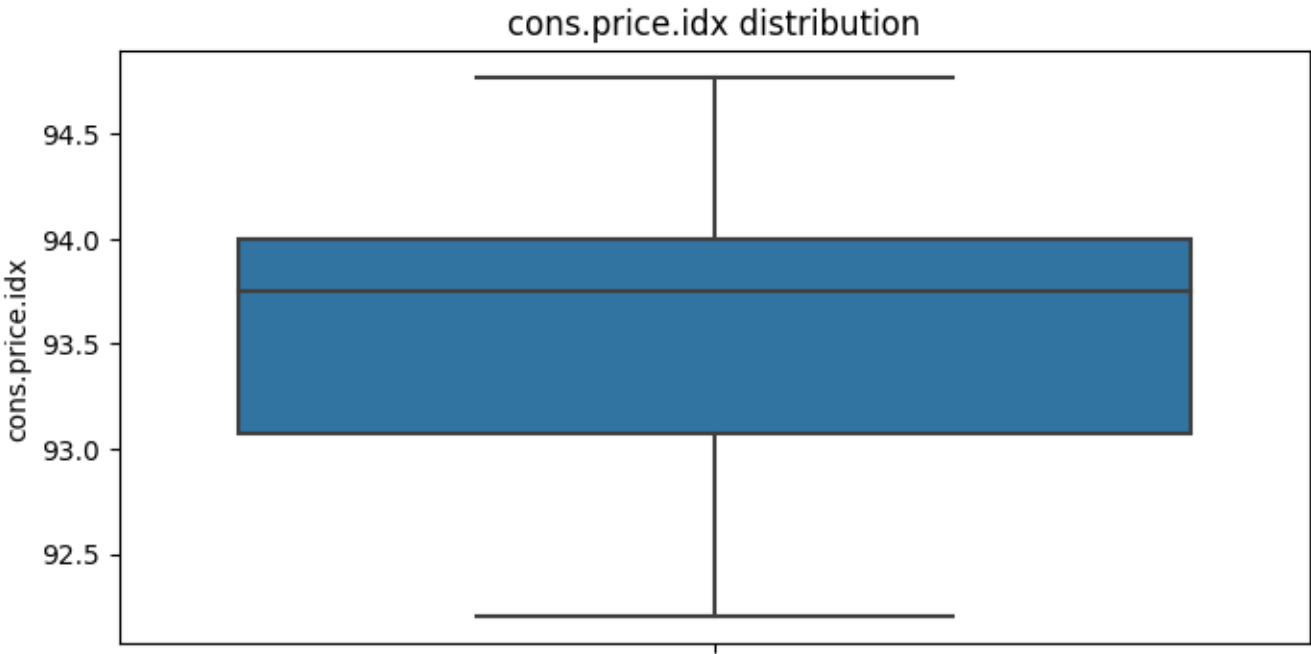


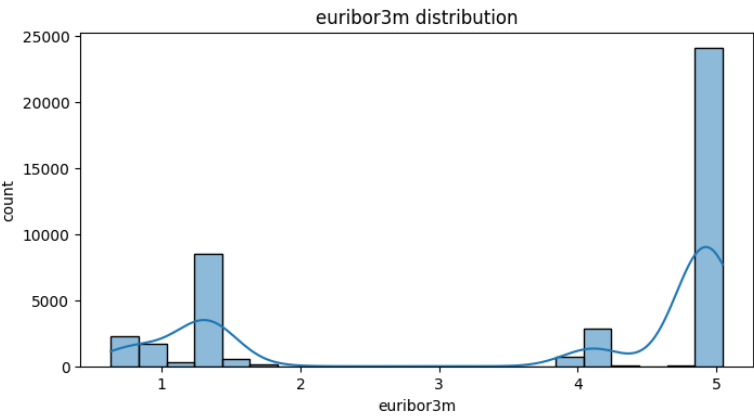
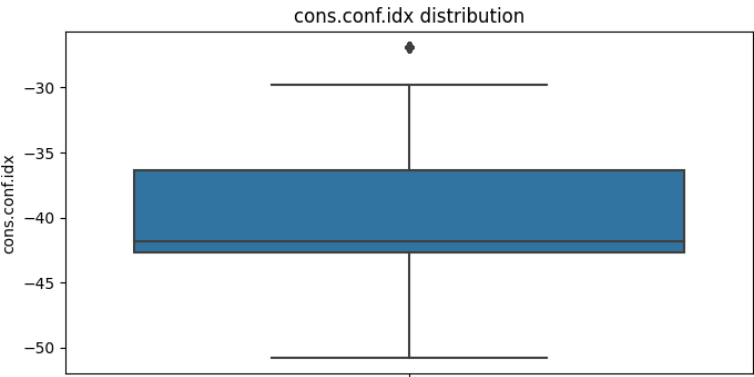


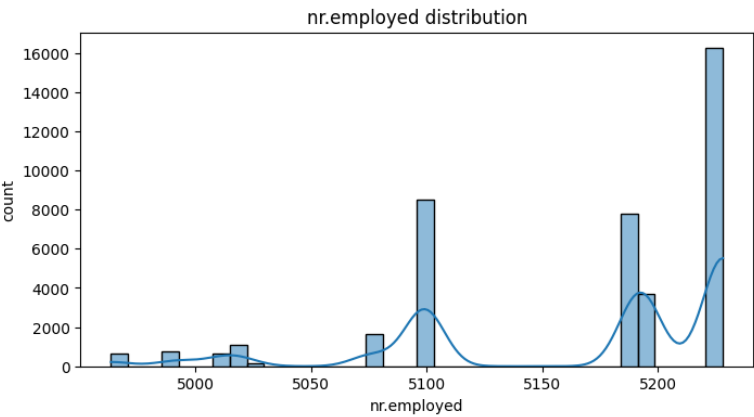
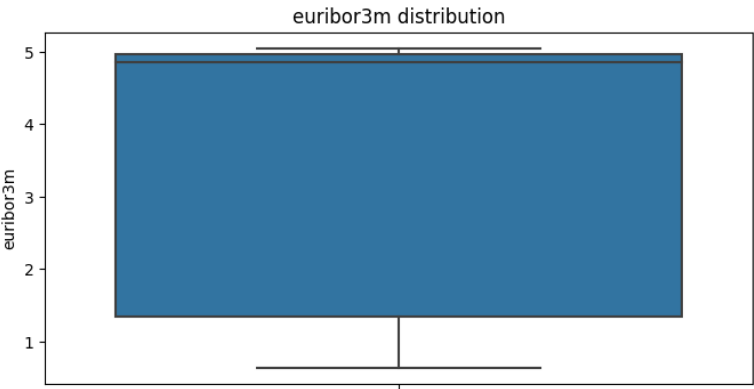


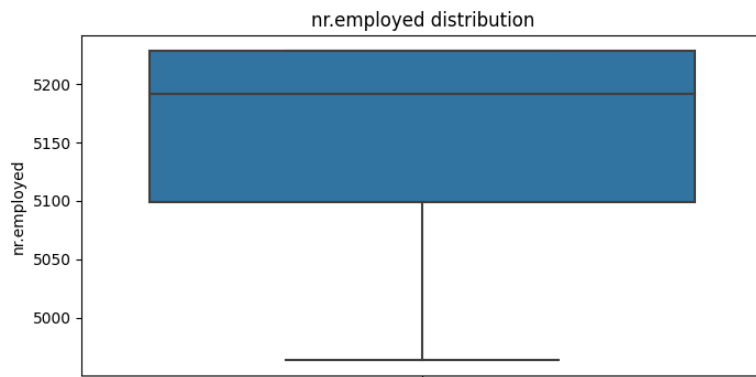












Note:

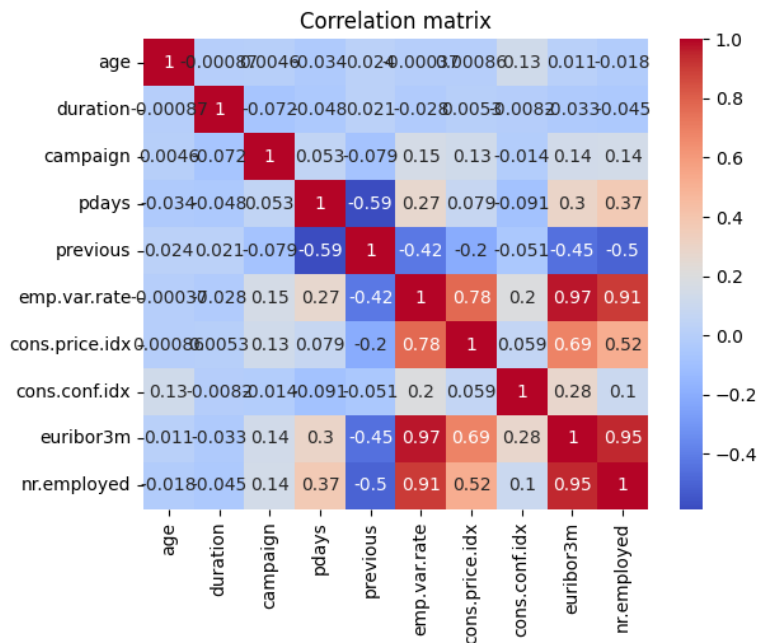
The histogram shows the distribution of the numerical variable, while the boxplot shows the median, interquartile range, and any outliers or extreme values. These plots can help identify any skewness or unusual values in the data, which may need to be addressed during preprocessing.

- 'age' : Looks like a normal distribution but there's a lot of values outside of Q3.
- 'duration' : exhibits a exponential distribution.

```
# Create a subset of the dataframe with only the numerical variables
num_cols = ['age', 'duration', 'campaign', 'pdays', 'previous',
            'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
            'euribor3m', 'nr.employed']
num_df = df[num_cols]

# Calculate the correlation matrix
corr_matrix = num_df.corr()

# Plot a heatmap to visualize the correlations
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation matrix')
plt.show()
```



Note :

The heatmap allows to visualize the strength and direction of the correlations between the numerical variables. Positive correlations are shown in shades of red, while negative correlations are shown in shades of blue. The darker the color, the stronger the correlation. This information can be useful for feature selection and identifying any multicollinearity between the variables.

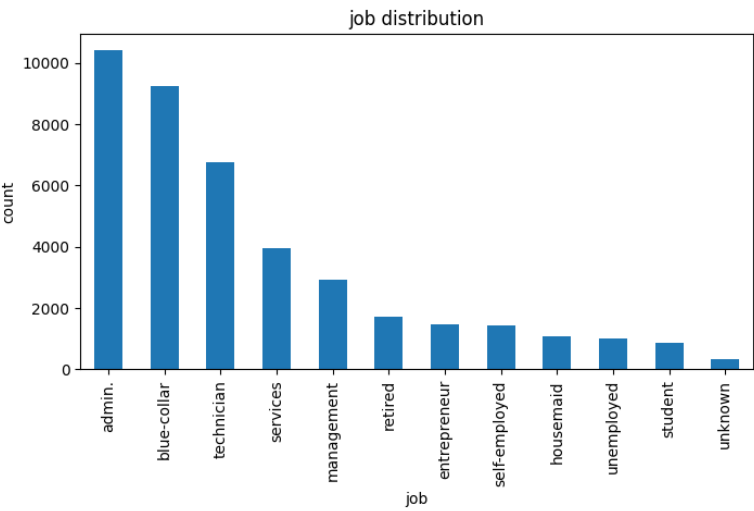
- There's a high degree of correlation between euribor3m and nr.employed, euribor3m and emp.var.rate, nr.employed and emp.var rate
- Basically, the final 5 features has some high degree of correlation that have to be consider.

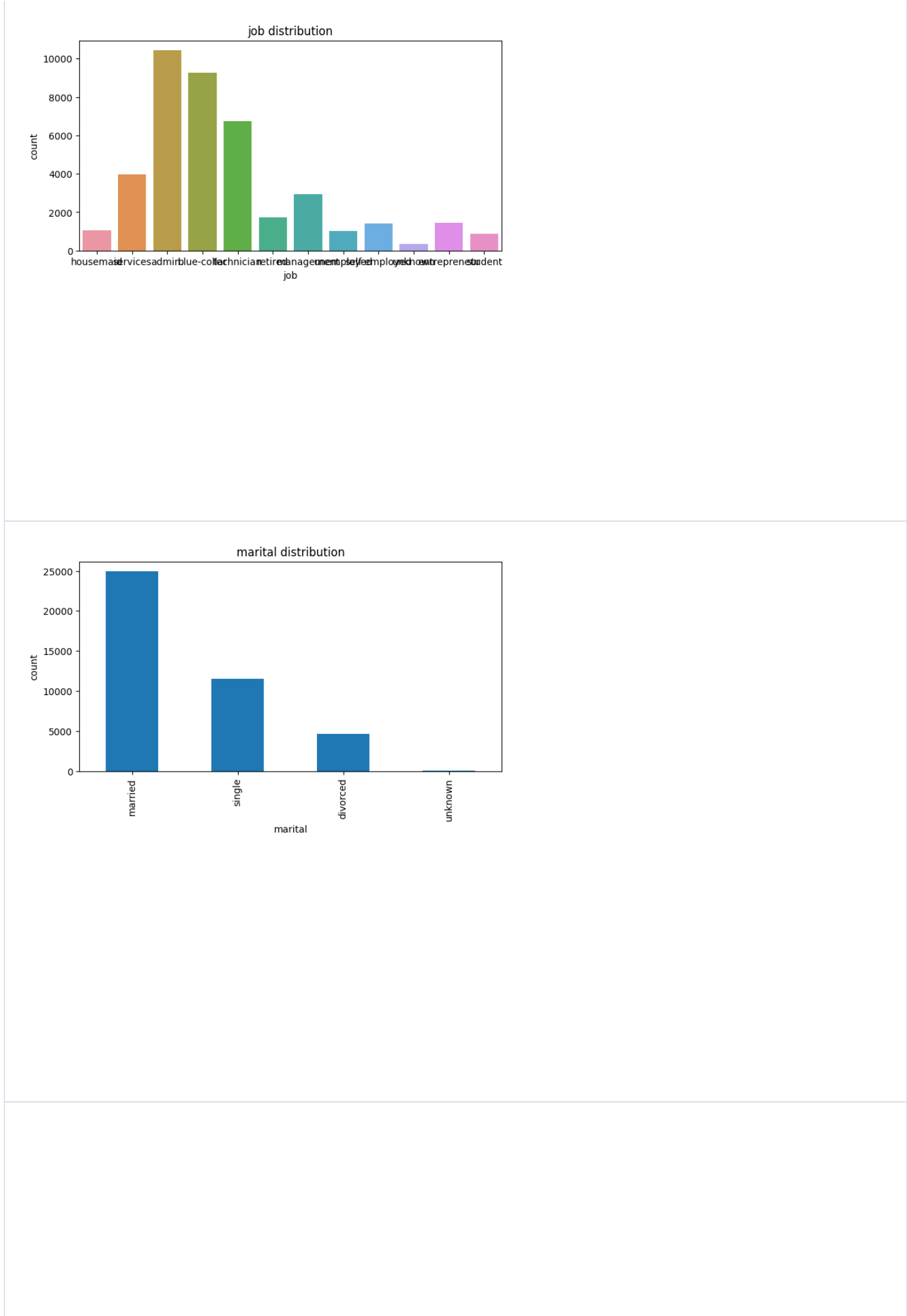
```
# Create a list of categorical column names
cat_cols = ['job', 'marital', 'education', 'default', 'housing',
            'loan', 'contact', 'month', 'day_of_week', 'poutcome']

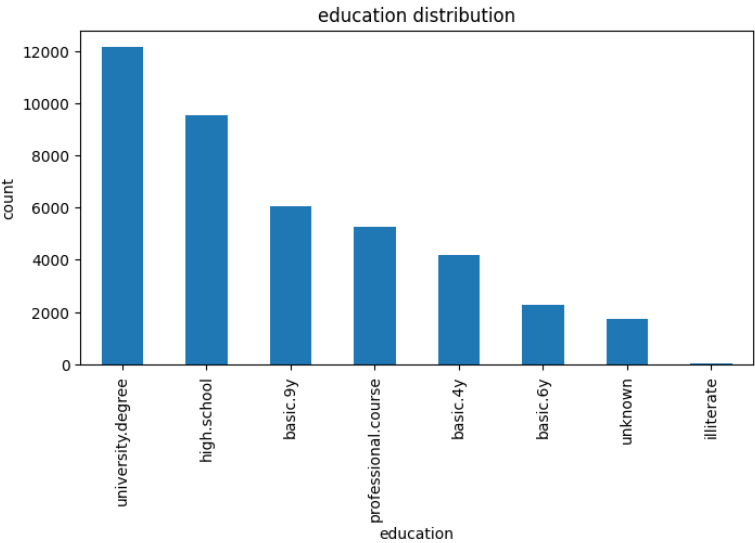
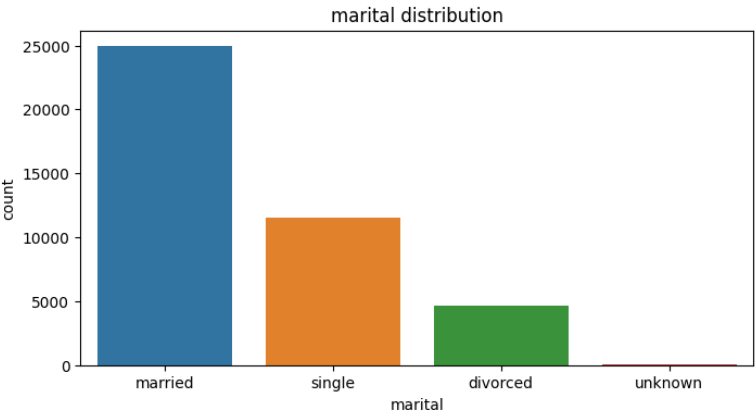
# Loop over each categorical column and plot a bar chart and count plot
for col in cat_cols:
    # Bar chart
    plt.figure(figsize=(8, 4))
    df[col].value_counts().plot(kind='bar')
    plt.title(f'{col} distribution')
    plt.xlabel(col)
    plt.ylabel('count')
    plt.show()
```

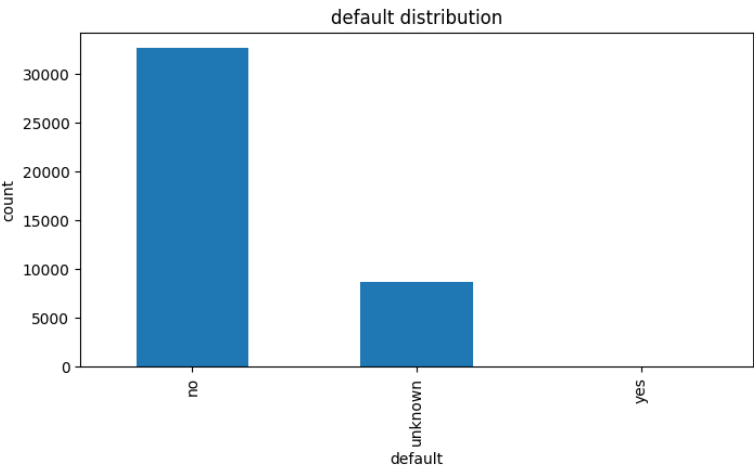
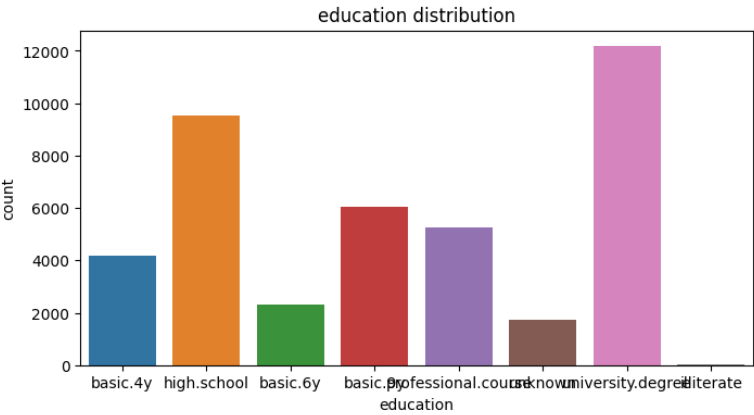


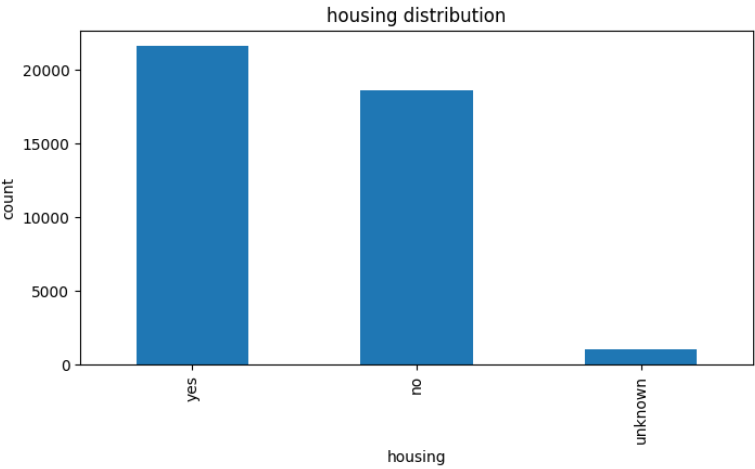
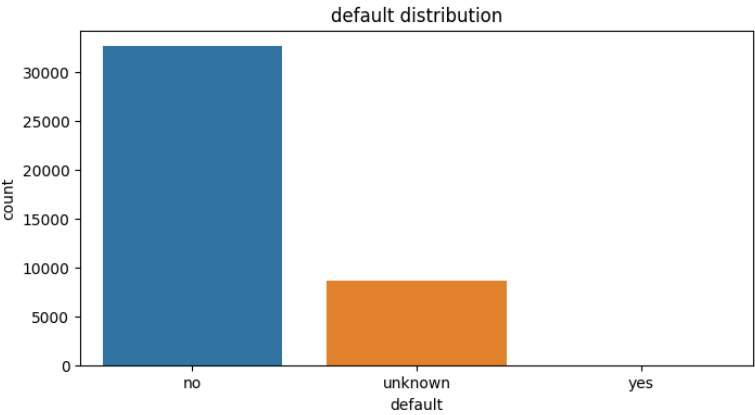
```
# Count plot
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x=col)
plt.title(f'{col} distribution')
plt.xlabel(col)
plt.ylabel('count')
plt.show()
```

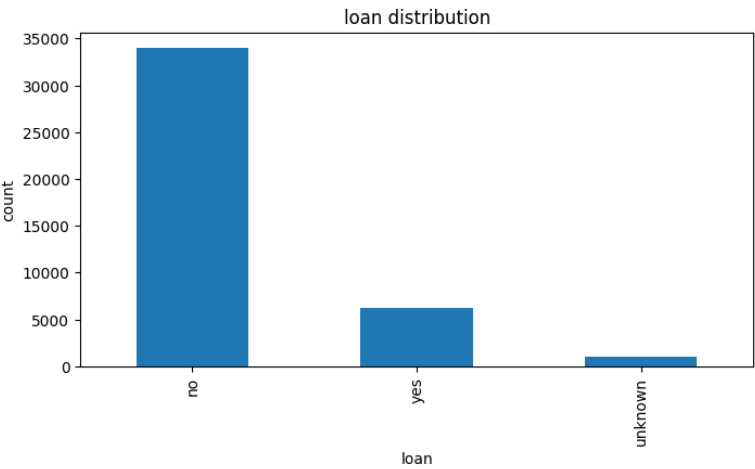
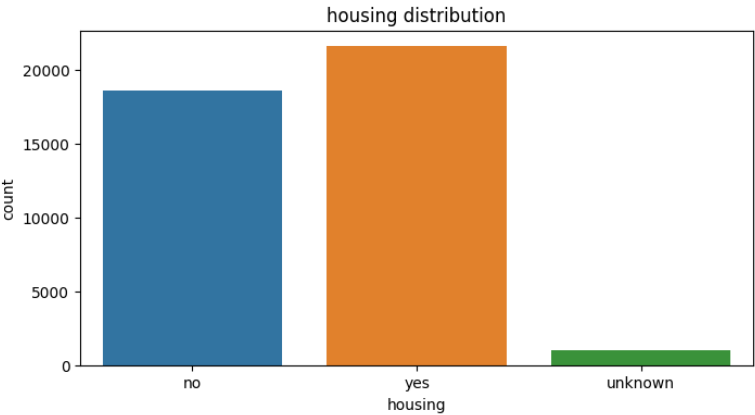


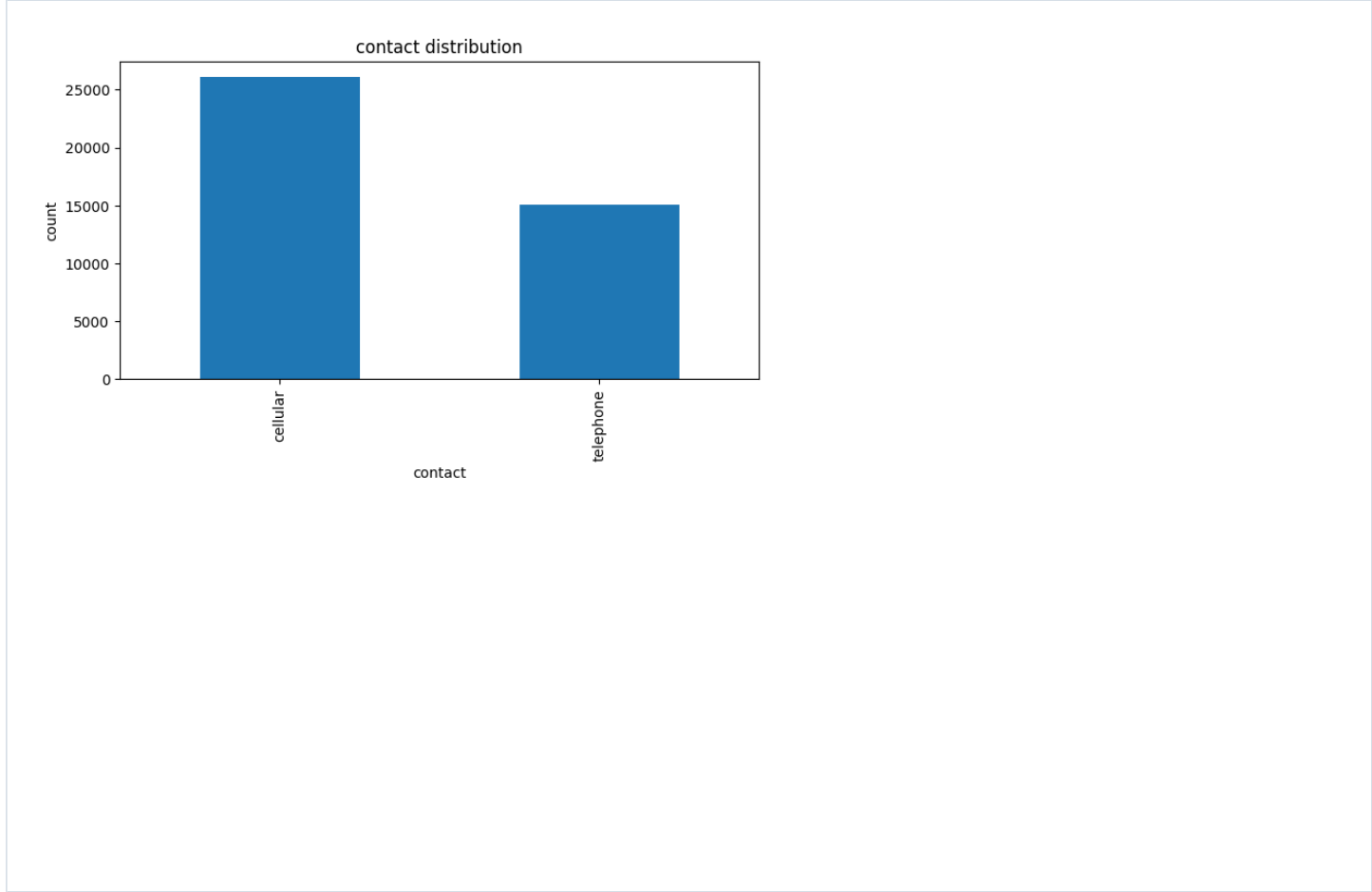
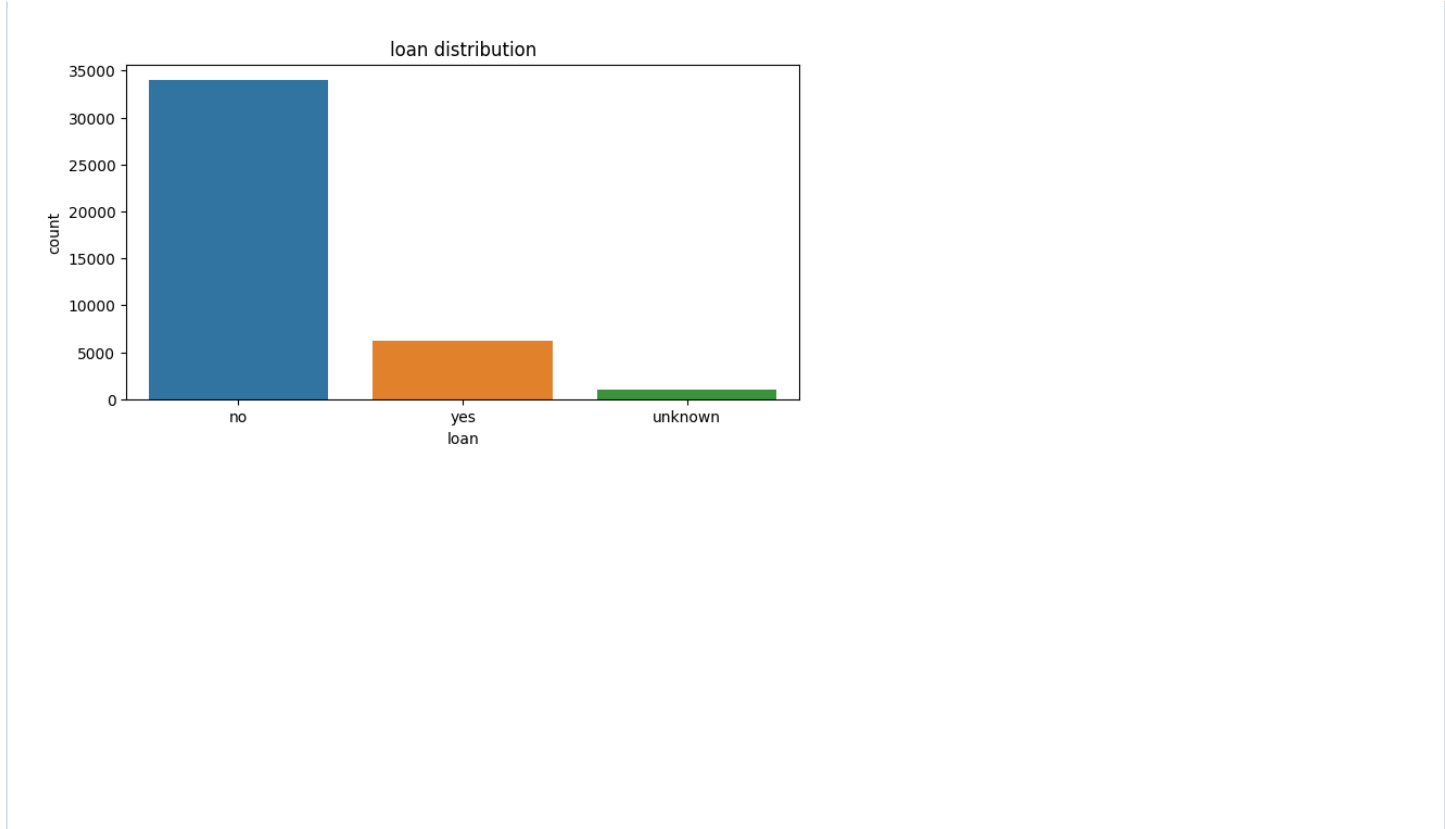


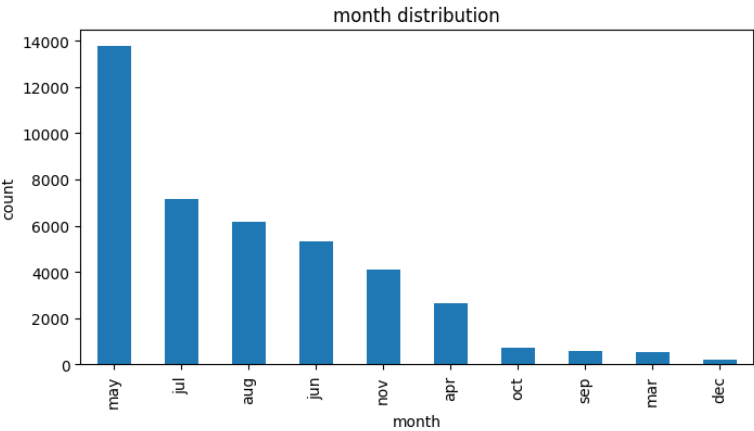
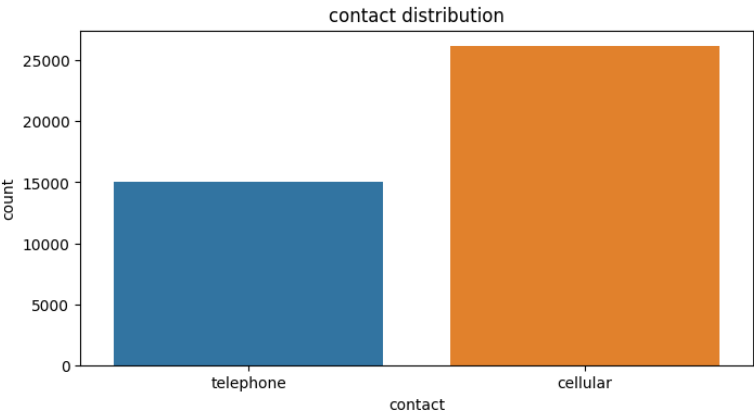


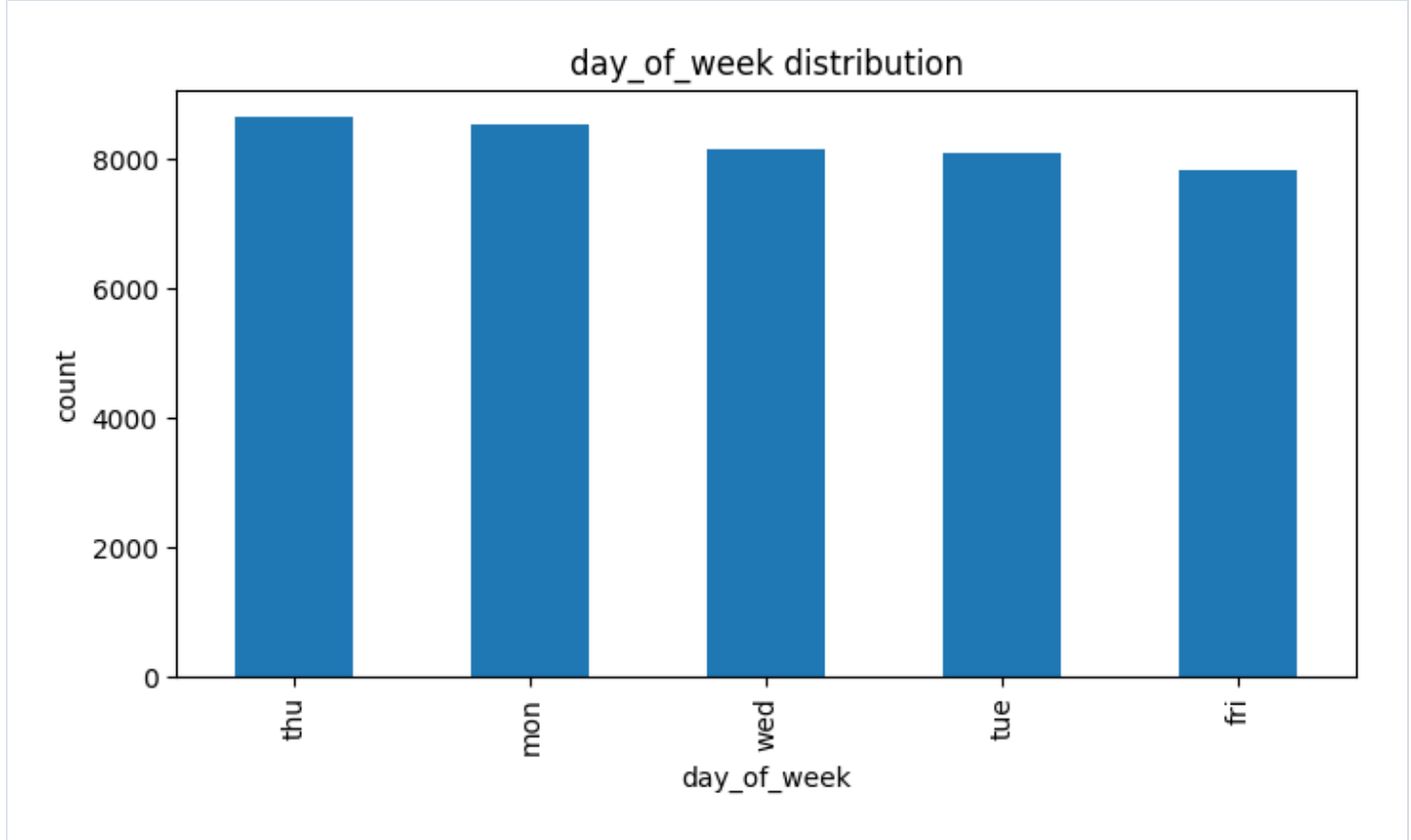
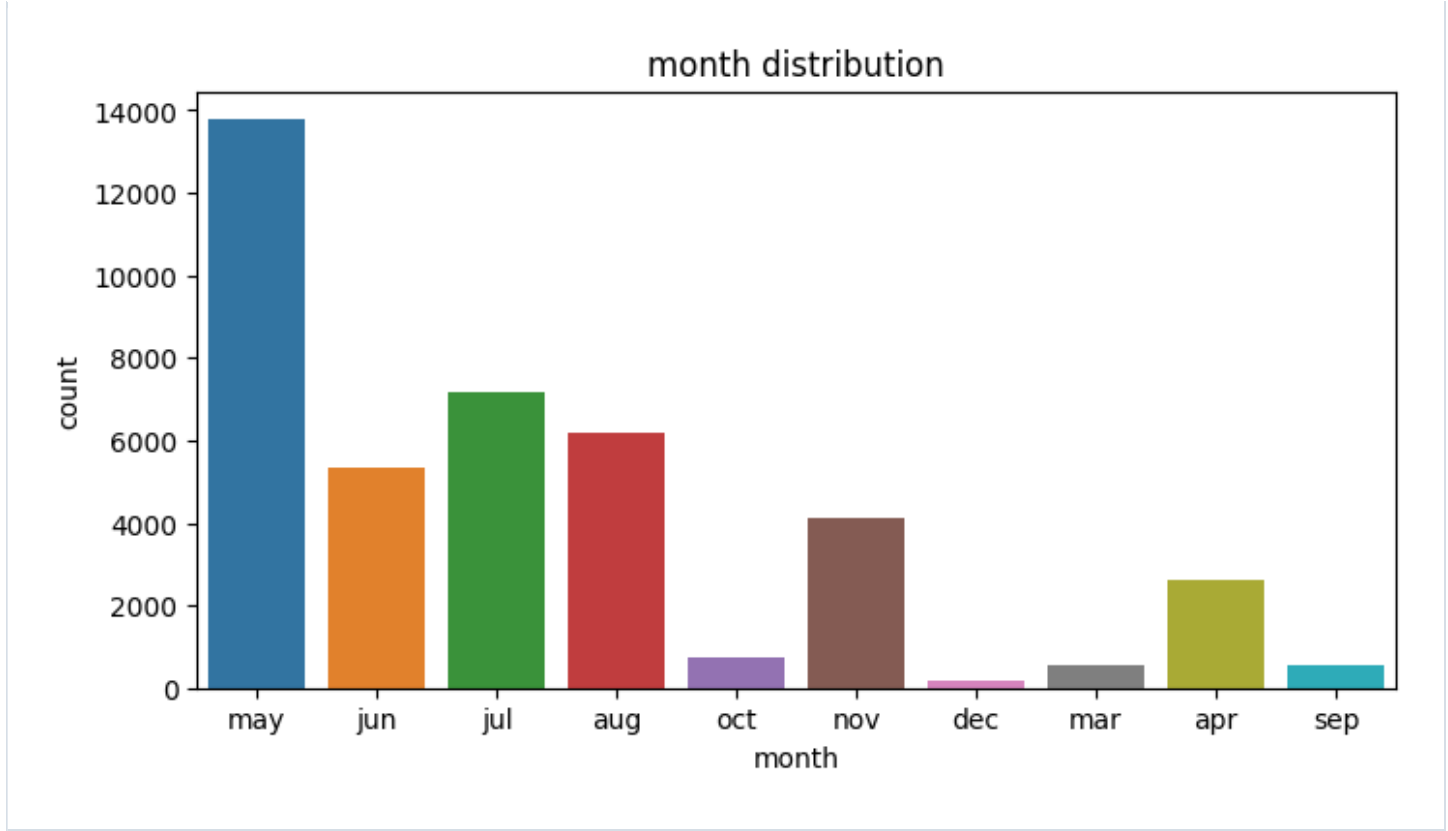


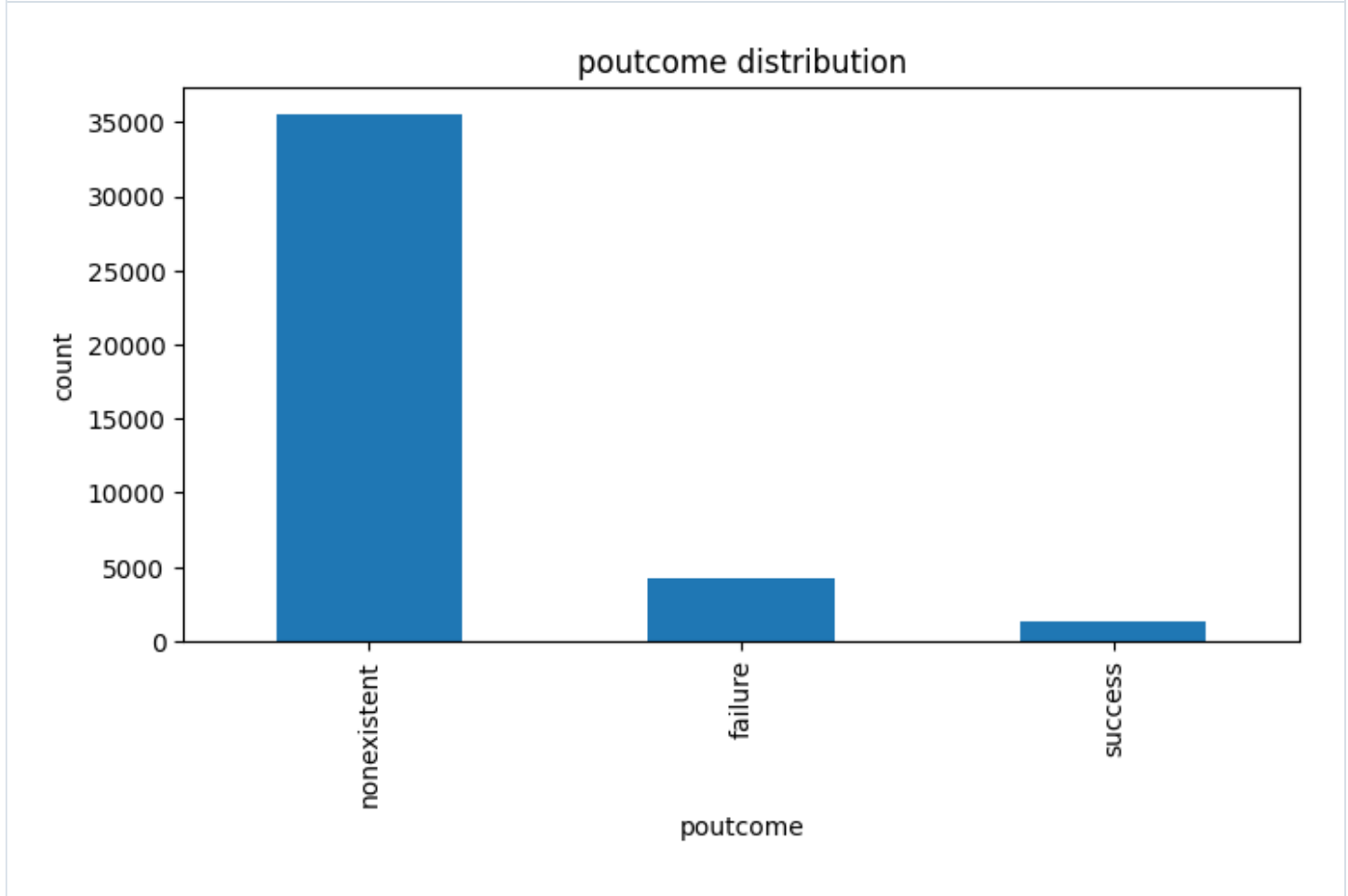
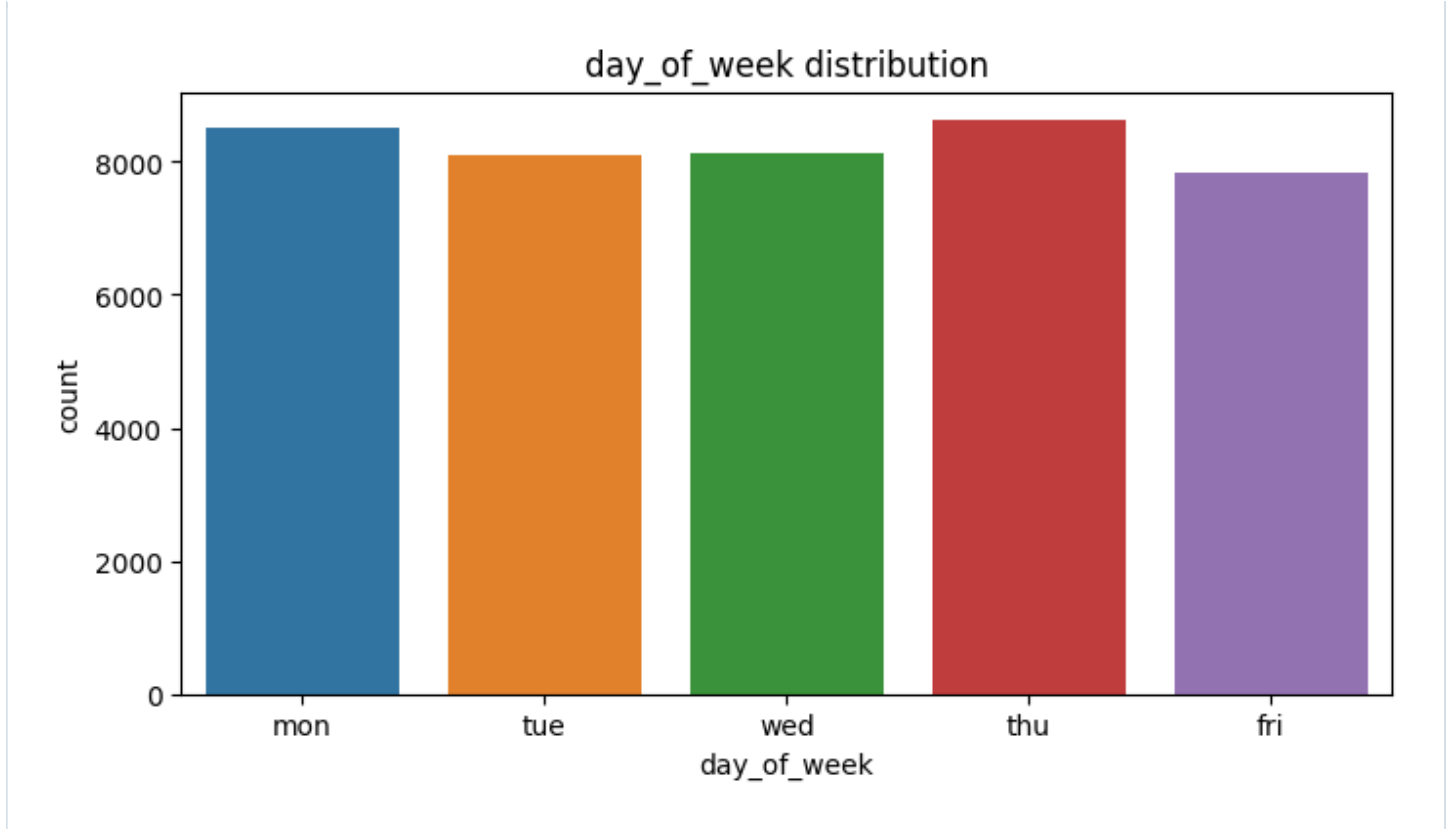


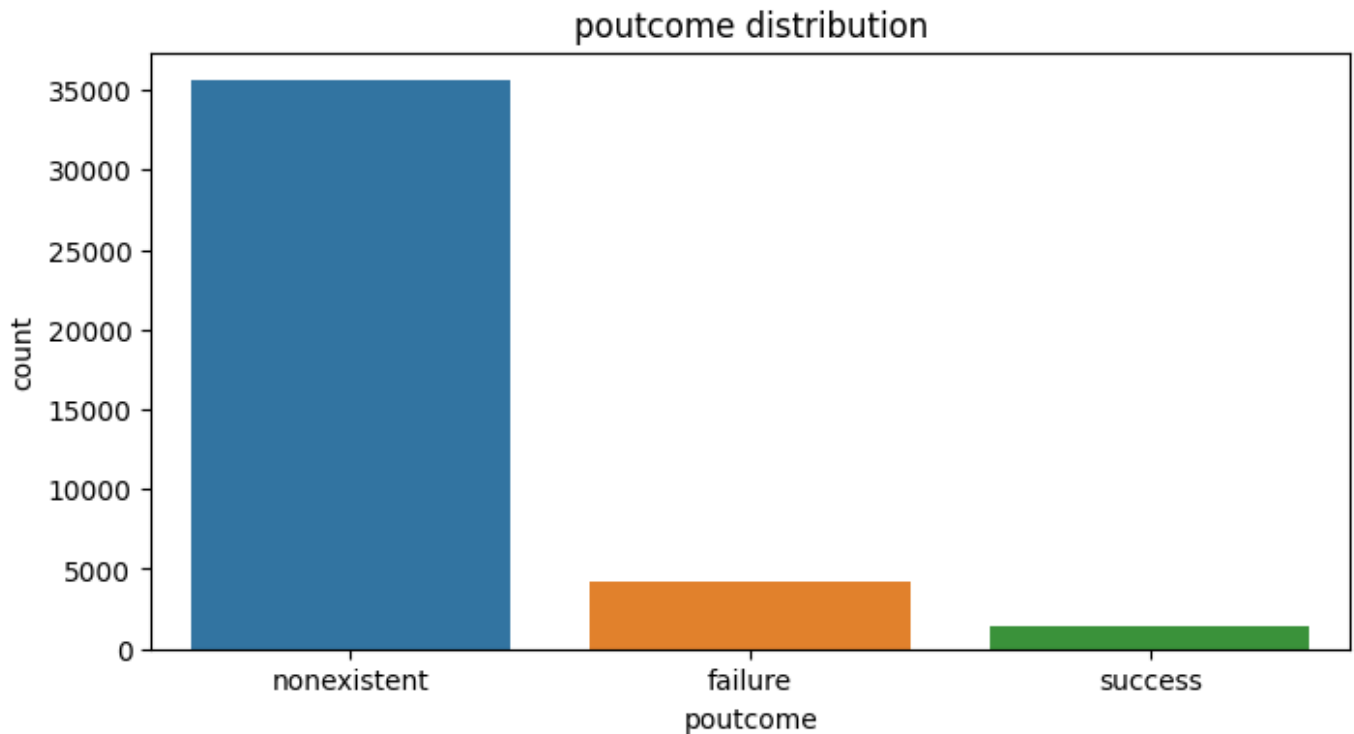












Note :

The bar chart shows the distribution of each categorical variable, while the count plot shows the count of each category in the variable. These plots can help understand the distribution of the categorical variables and identify any imbalances or patterns that may be useful for modeling.

- There quite a lot of instances with admin, blue collar job, are married, has university degree/ high school, have not defaulted, have no loan, poutcome is nonexistent.
- No too drastic imbalance here.

Model Selection

- Select a set of candidate classification algorithms to evaluate on the dataset. Popular algorithms for binary classification like Logistic Regression.

One-Hot Encoding

```
# Separate the features (X) and target variable (y)
X = df.drop('y', axis=1)
y = df['y']

# Select the numerical columns
numerical_cols = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.p
X_encoded = X[numerical_cols]

# One-hot encoding
for col in cat_cols:
    # Create a one-hot encoder object
    enc = OneHotEncoder(handle_unknown='ignore')
```

```
# Fit and transform the data
enc_df = pd.DataFrame(enc.fit_transform(df[[col]]).toarray())

# Rename the columns and concatenate with the original dataframe
enc_df = enc_df.add_prefix(col+'_')
X_encoded = pd.concat([X_encoded, enc_df], axis=1)

# Display the result
X_encoded.head()
```

	age int64	duration int64	campaign int64	pdays int64	previous int64	emp
0	56	261	1	999	0	
1	57	149	1	999	0	
2	37	226	1	999	0	
3	40	151	1	999	0	
4	56	307	1	999	0	

Label Encoding

```
# Encode the binary variable using label encoding
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
y
```

```
array([0, 0, 0, ..., 0, 1, 0])
```

Logistic Regression

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Use the trained model to make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the performance of the model using the test set
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print('Precision:', precision_score(y_test, y_pred))
```

```
print('Recall:', recall_score(y_test, y_pred))
print('F1 score:', f1_score(y_test, y_pred))
```

Accuracy: 0.911750424860403

Precision: 0.6682847896440129

Recall: 0.44171122994652406

F1 score: 0.531873792659369

/shared-libs/python3.9/py/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

XGBoost

```
import xgboost as xgb
from sklearn.model_selection import train_test_split

# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# convert the data to xgboost's DMatrix format
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# set the hyperparameters for the XGBoost model
params = {
    'max_depth': 6,
    'eta': 0.3,
    'objective': 'binary:logistic',
    'eval_metric': 'auc'
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the test data
y_pred = xgb_model.predict(dtest)

# convert the predictions to binary values
y_pred_binary = [1 if p >= 0.5 else 0 for p in y_pred]

# evaluate the performance of the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('Accuracy:', accuracy_score(y_test, y_pred_binary))
print('Precision:', precision_score(y_test, y_pred_binary))
print('Recall:', recall_score(y_test, y_pred_binary))
print('F1 score:', f1_score(y_test, y_pred_binary))
```

```
Accuracy: 0.9139354212187424
Precision: 0.64125
Recall: 0.5486631016042781
F1 score: 0.5913544668587897
```

Note:

- Since we concluded before that the target value is highly imbalanced.
- We should not use accuracy as the metric to evaluate model performance
- Instead, we use F1 score.
- Here, we can conclude that our baseline F1 score is 0.53 and 0.59 which is a fairly low F1 Score, indicating the model is not doing well.

Error Analysis

- Next, we will use oversampling to oversample the underrepresented target value to create a balanced dataset.

```
# Create a random oversampler object
oversampler = RandomOverSampler(random_state=42)

# Fit and transform the data
X_resampled, y_resampled = oversampler.fit_resample(X_train, y_train)
```

Model Tuning

```
# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2,

# convert the data to xgboost's DMatrix format
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# set the hyperparameters for the XGBoost model
params = {
    'max_depth': 6,
    'eta': 0.3,
    'objective': 'binary:logistic',
    'eval_metric': 'auc'
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the test data
y_pred = xgb_model.predict(dtest)

# convert the predictions to binary values
```

```
y_pred_binary = [1 if p >= 0.5 else 0 for p in y_pred]

# evaluate the performance of the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('Accuracy:', accuracy_score(y_test, y_pred_binary))
print('Precision:', precision_score(y_test, y_pred_binary))
print('Recall:', recall_score(y_test, y_pred_binary))
print('F1 score:', f1_score(y_test, y_pred_binary))
```

```
Accuracy: 0.9352880834330655
Precision: 0.897658551713444
Recall: 0.9835202174651716
F1 score: 0.938629914876368
```

Note :

- By using the oversampling method and training it on XGBoost model, we have increased the F1 score from 0.59 to 0.93, indicating a model improvement of 57% with a good F1-score.

Future improvement

- Feature scaling could be used to improve the model performance since some of the numerical features do not exhibit Gaussian distribution.
- Feature selection could also be done since the correlation analysis indicate there might be issue with multicollinearity, challenging model assumptions that need statistical independence.
- Hyperparameter tuning could also be done to improve the XGBoost model performance.

Overall,

- By using encoding techniques, oversampling technique and XGBoost, the model on this imbalanced dataset has improved by 57% achieving a F1 Score of 0.93 which is considered a useful model for the application in bank telemarketing.