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

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
from sklearn.datasets import fetch_openml
X, y = fetch_openml("Bank_marketing_data_set_UCI", return_X_y= True)
```

In [ ]:
```
X.head()
```

Out[ ]:

	age	job	marital	education	default	balance	housing	loan	contact	da
0	58	management	married	tertiary	no	2143	yes	no	unknown	
1	44	technician	single	secondary	no	29	yes	no	unknown	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	
4	33	unknown	single	unknown	no	1	no	no	unknown	

In [ ]:
```
y[:5]
```

Out[ ]:
```
0    no
1    no
2    no
3    no
4    no
Name: y, dtype: object
```

In [ ]:
```
y.value_counts(normalize=True).round(2)
```

Out[ ]:
```
y
no     0.88
yes    0.12
Name: proportion, dtype: float64
```

In [ ]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
dtypes: int64(7), object(9)
memory usage: 5.5+ MB
```

In [ ]:
```
def summary(df):
   tf=pd.DataFrame({'데이터 종류':df.dtypes,
                    '빈값':df.isnull().sum(),
                    '특별갯수':df.nunique(),
                    '많이 노온': [list(df[col].unique()[0:12]) for col in df.colu
                    )
   return tf
```

In [ ]:
```
summary(X)
```

Out[ ]:

	데이터 종류	빈 값	특별갯수	많이 노온
age	int64	0	77	[58, 44, 33, 47, 35, 28, 42, 43, 41, 29, 53, 57]
job	object	0	12	[management, technician, entrepreneur, blue-co...
marital	object	0	3	[married, single, divorced]
education	object	0	4	[tertiary, secondary, unknown, primary]
default	object	0	2	[no, yes]
balance	int64	0	7168	[2143, 29, 2, 1506, 1, 231, 447, 121, 593, 270...
housing	object	0	2	[yes, no]
loan	object	0	2	[no, yes]
contact	object	0	3	[unknown, cellular, telephone]
day	int64	0	31	[5, 6, 7, 8, 9, 12, 13, 14, 15, 16, 19, 20]
month	object	0	12	[may, jun, jul, aug, oct, nov, dec, jan, feb, ...
duration	int64	0	1573	[261, 151, 76, 92, 198, 139, 217, 380, 50, 55,...
campaign	int64	0	48	[1, 2, 3, 5, 4, 6, 7, 8, 9, 10, 11, 12]
pdays	int64	0	559	[-1, 151, 166, 91, 86, 143, 147, 89, 140, 176,...
previous	int64	0	41	[0, 3, 1, 4, 2, 11, 16, 6, 5, 10, 12, 7]
poutcome	object	0	4	[unknown, failure, other, success]

In [ ]:
```
def plotting_features(data):
    cols = data.columns

    nrows= int(np.ceil(len(cols)/2))
    fig, ax = plt.subplots(
                        nrows=nrows,
                        ncols=2,
                        figsize=(15,30),
                        constrained_layout=True)
    ax = ax.ravel()

    for i in range(len(cols)):
        if (data[cols[i]].dtypes == 'number'):
                sns.countplot(y = data[cols[i]], ax=ax[i])
                ax[i].set_title(f'{cols[i]} count')

        else:
            sns.histplot(x = data[cols[i]], ax=ax[i])
            ax[i].set_title(f'{cols[i]} distribution');
```

In [ ]:
```
X.shape
```

Out[ ]:
```
(45211, 16)
```

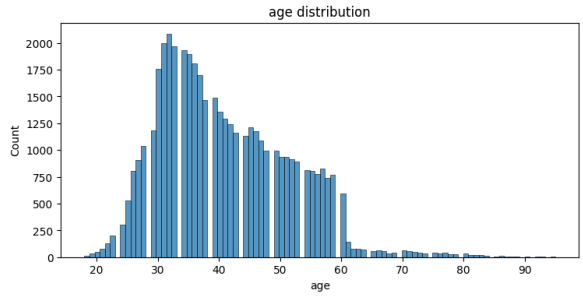
In [ ]:
```
obj=X.select_dtypes(include=[object])
```

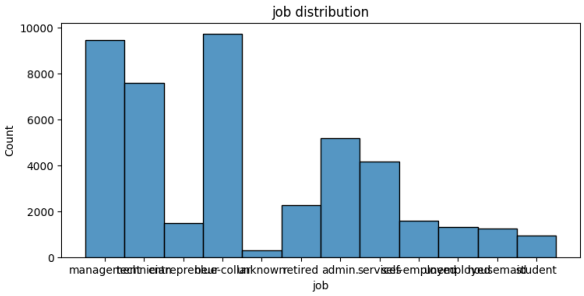
In [ ]:
```
obj[:3]
```

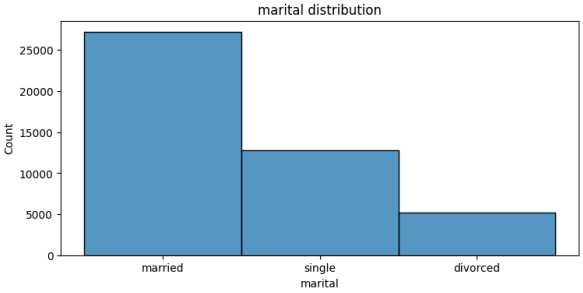
Out[ ]:

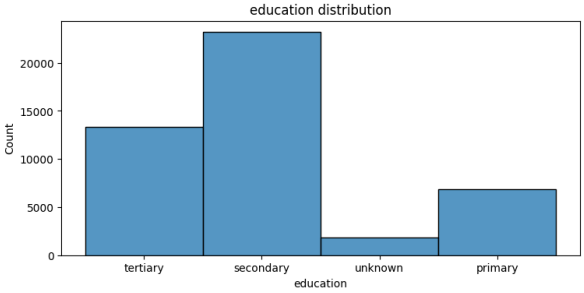
	job	marital	education	default	housing	loan	contact	month	poutcom
0	management	married	tertiary	no	yes	no	unknown	may	unknow
1	technician	single	secondary	no	yes	no	unknown	may	unknow
2	entrepreneur	married	secondary	no	yes	yes	unknown	may	unknow

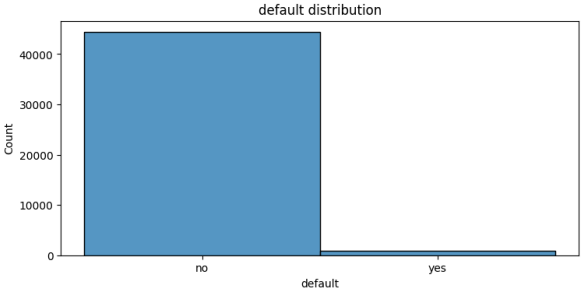
In [ ]:
```
plotting_features(X)
```

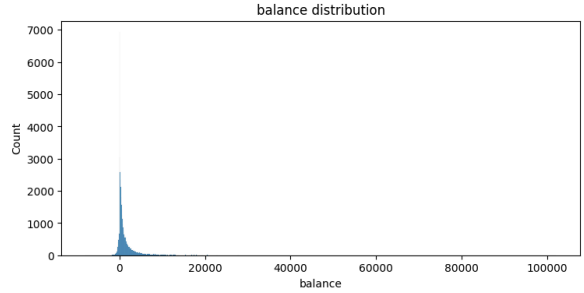

age distribution
Count
age

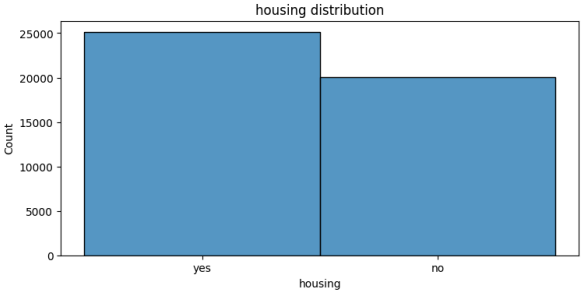

job distribution
Count
job

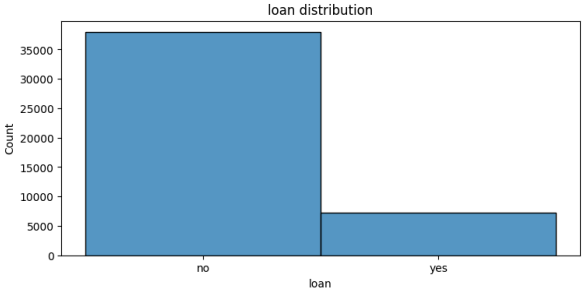

marital distribution
Count
married
single
divorced
marital

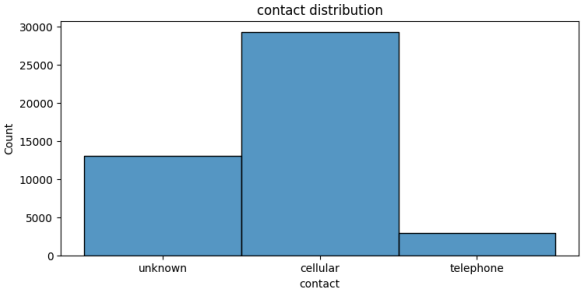

education distribution
Count
tertiary
secondary
unknown
primary
education

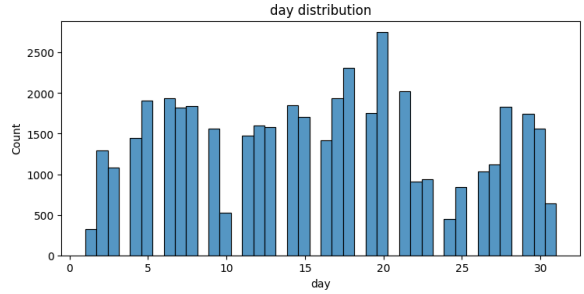

default distribution
Count
no
yes
default

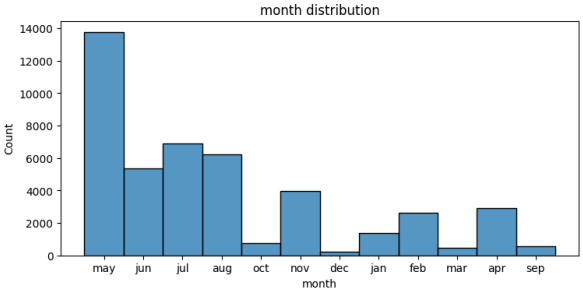

balance distribution
Count
balance

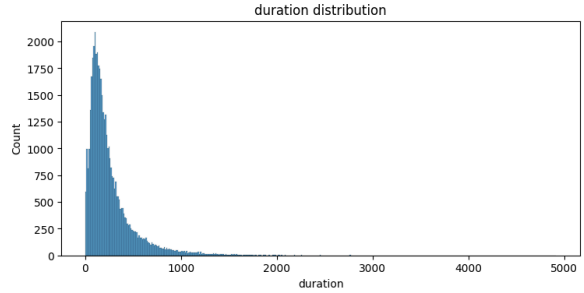

housing distribution
Count
yes
no
housing

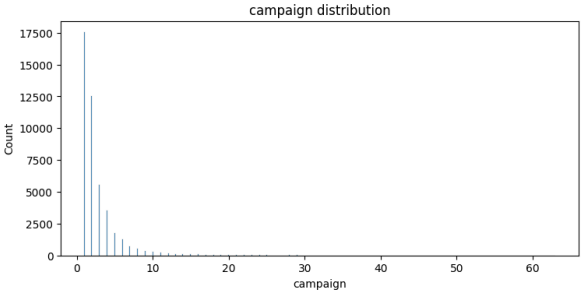

loan distribution
Count
no
yes
loan

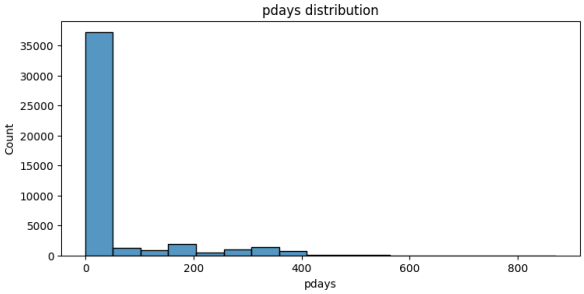

contact distribution
Count
unknown
cellular
telephone
contact

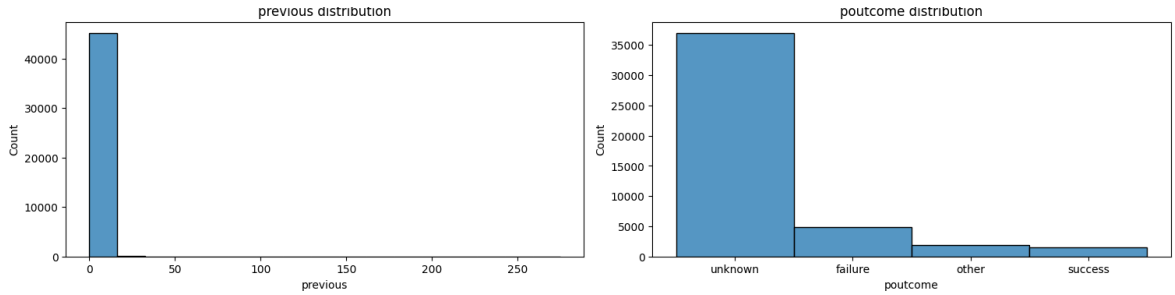

day distribution
Count
day


month distribution
Count
may
jun
jul
aug
oct
nov
dec
jan
feb
mar
apr
sep
month


duration distribution
Count
duration


campaign distribution
Count
campaign

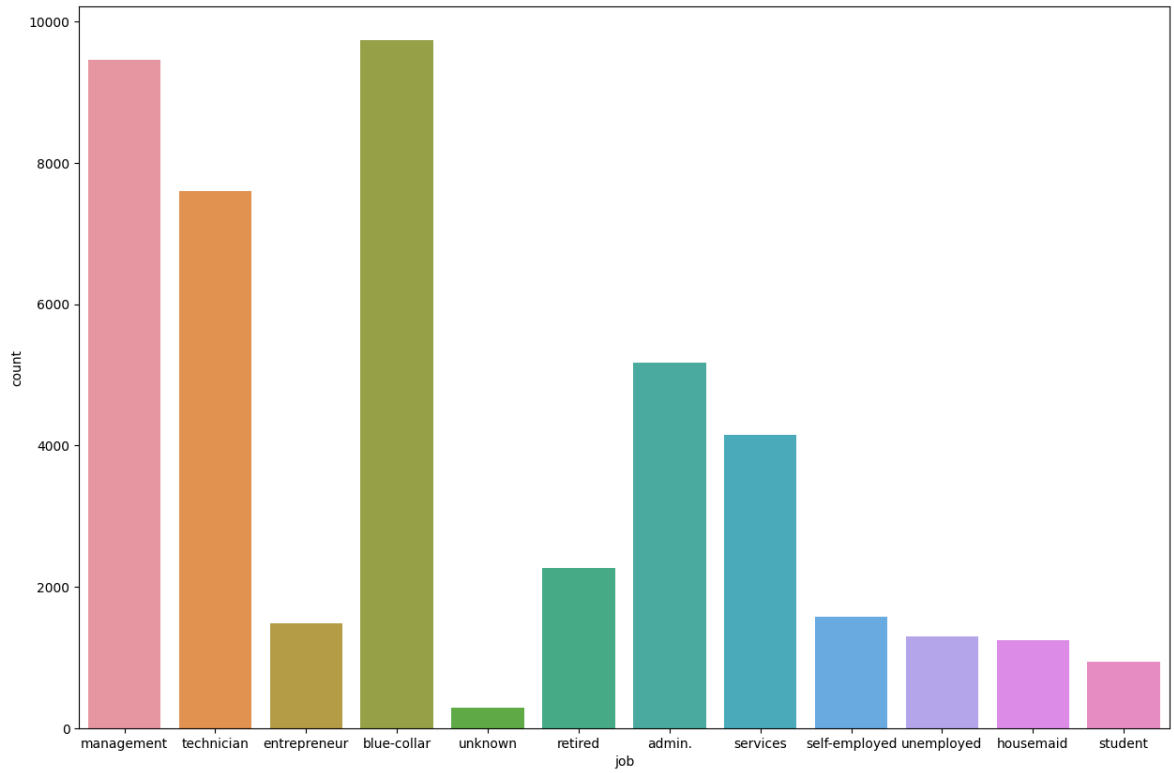

pdays distribution
Count
pdays



In [ ]:
```
plt.subplots(figsize=(15,10))
sns.countplot(x='job',data=X)
```

Out[ ]: (<Figure size 1500x1000 with 1 Axes>, <Axes: >)

Out[ ]: <Axes: xlabel='job', ylabel='count'>



In [ ]:
```
#불필요한 열 찾기
#poutcome[unknown 넘 많음], default(credit 있는지 없는지)[거의 다 'no'라고] , montl
```

In [ ]:
```
X_dr=X.drop(['poutcome','default','day','month'],axis=1)
```

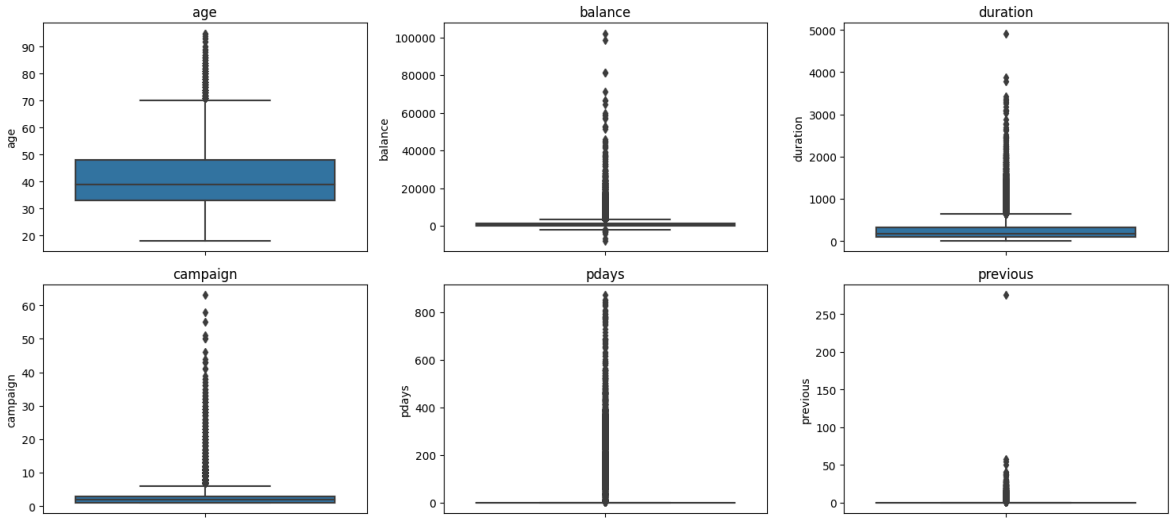
In [ ]:
```
X_dr
```

Out[ ]:

	age	job	marital	education	balance	housing	loan	contact	dura
0	58	management	married	tertiary	2143	yes	no	unknown	
1	44	technician	single	secondary	29	yes	no	unknown	
2	33	entrepreneur	married	secondary	2	yes	yes	unknown	
3	47	blue-collar	married	unknown	1506	yes	no	unknown	
4	33	unknown	single	unknown	1	no	no	unknown	
...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	825	no	no	cellular	
45207	71	retired	divorced	primary	1729	no	no	cellular	
45208	72	retired	married	secondary	5715	no	no	cellular	
45209	57	blue-collar	married	secondary	668	no	no	telephone	
45210	37	entrepreneur	married	secondary	2971	no	no	cellular	

45211 rows × 12 columns

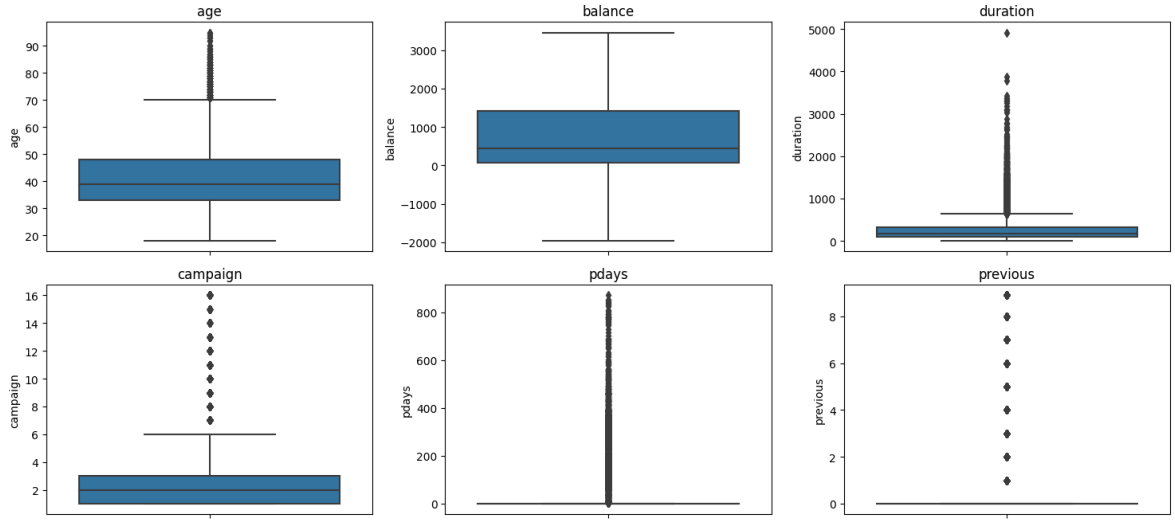
In [ ]:
```
plt.figure(figsize=(15, 10))
for i, col in enumerate(X_dr.select_dtypes(include=np.number), 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=X_dr[col])
    plt.title(col)
plt.tight_layout()
plt.show();
```



In [ ]:
```
#Balance 이상치 IQR
q1=X_dr.balance.quantile(.25)
q3=X_dr.balance.quantile(.75)
iqr=q3-q1
X_dr.balance=X_dr.balance.clip(lower=q1 - 1.5 * iqr, upper=q3 + 1.5 * iqr)
```

In [ ]:
```
#campaign and previous (only %1  is outlier)
X_dr['campaign'] = X_dr['campaign'].clip(upper=X_dr['campaign'].quantile(0.99))
X_dr['previous'] = X_dr['previous'].clip(upper=X_dr['previous'].quantile(0.99))
```

In [ ]:
```
plt.figure(figsize=(15, 10))
for i, col in enumerate(X_dr.select_dtypes(include=np.number), 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=X_dr[col])
    plt.title(col)
plt.tight_layout()
plt.show();
```



In [ ]:
```
X_dumy= pd.get_dummies(X_dr)
```

In [ ]:
```
X_dumy
```

Out[ ]:

	age	balance	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_
0	58	2143	261	1	-1	0.0	False	False	
1	44	29	151	1	-1	0.0	False	False	
2	33	2	76	1	-1	0.0	False	False	
3	47	1506	92	1	-1	0.0	False	True	
4	33	1	198	1	-1	0.0	False	False	
...	...	...	...	...	...	...	...	...	
45206	51	825	977	3	-1	0.0	False	False	
45207	71	1729	456	2	-1	0.0	False	False	
45208	72	3462	1127	5	184	3.0	False	False	
45209	57	668	508	4	-1	0.0	False	True	
45210	37	2971	361	2	188	8.9	False	False	

45211 rows × 32 columns

In [ ]:
```
X_dumy=X_dumy.replace(True,1)
X_dumy=X_dumy.replace(False,0)
```

In [ ]:
```
y=y.replace('no',0)
y=y.replace('yes',1)
```

In [ ]:
```
X_dumy
```

Out[ ]:

	age	balance	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_
0	58	2143	261	1	-1	0.0	0	0	
1	44	29	151	1	-1	0.0	0	0	
2	33	2	76	1	-1	0.0	0	0	
3	47	1506	92	1	-1	0.0	0	1	
4	33	1	198	1	-1	0.0	0	0	
...	...	...	...	...	...	...	...	...	
45206	51	825	977	3	-1	0.0	0	0	
45207	71	1729	456	2	-1	0.0	0	0	
45208	72	3462	1127	5	184	3.0	0	0	
45209	57	668	508	4	-1	0.0	0	1	
45210	37	2971	361	2	188	8.9	0	0	

45211 rows × 32 columns

In [ ]:
```
from sklearn.preprocessing import StandardScaler
numerical_features = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previo
scaler = StandardScaler()
X_dumy[numerical_features] = scaler.fit_transform(X_dumy[numerical_features])

X_dumy
```

Out[ ]:

	age	balance	duration	campaign	pdays	previous	job_admin.	job
0	1.606965	1.027653	0.011016	-0.654134	-0.411453	-0.359918	0	
1	0.288529	-0.768817	-0.416127	-0.654134	-0.411453	-0.359918	0	
2	-0.747384	-0.791761	-0.707361	-0.654134	-0.411453	-0.359918	0	
3	0.571051	0.486333	-0.645231	-0.654134	-0.411453	-0.359918	0	
4	-0.747384	-0.792611	-0.233620	-0.654134	-0.411453	-0.359918	0	
...	...	...	...	...	...	...	...	
45206	0.947747	-0.092379	2.791329	0.119347	-0.411453	-0.359918	0	
45207	2.831227	0.675837	0.768224	-0.267394	-0.411453	-0.359918	0	
45208	2.925401	2.148534	3.373797	0.892829	1.436189	1.697977	0	
45209	1.512791	-0.225797	0.970146	0.506088	-0.411453	-0.359918	0	
45210	-0.370689	1.731284	0.399328	-0.267394	1.476138	5.745170	0	

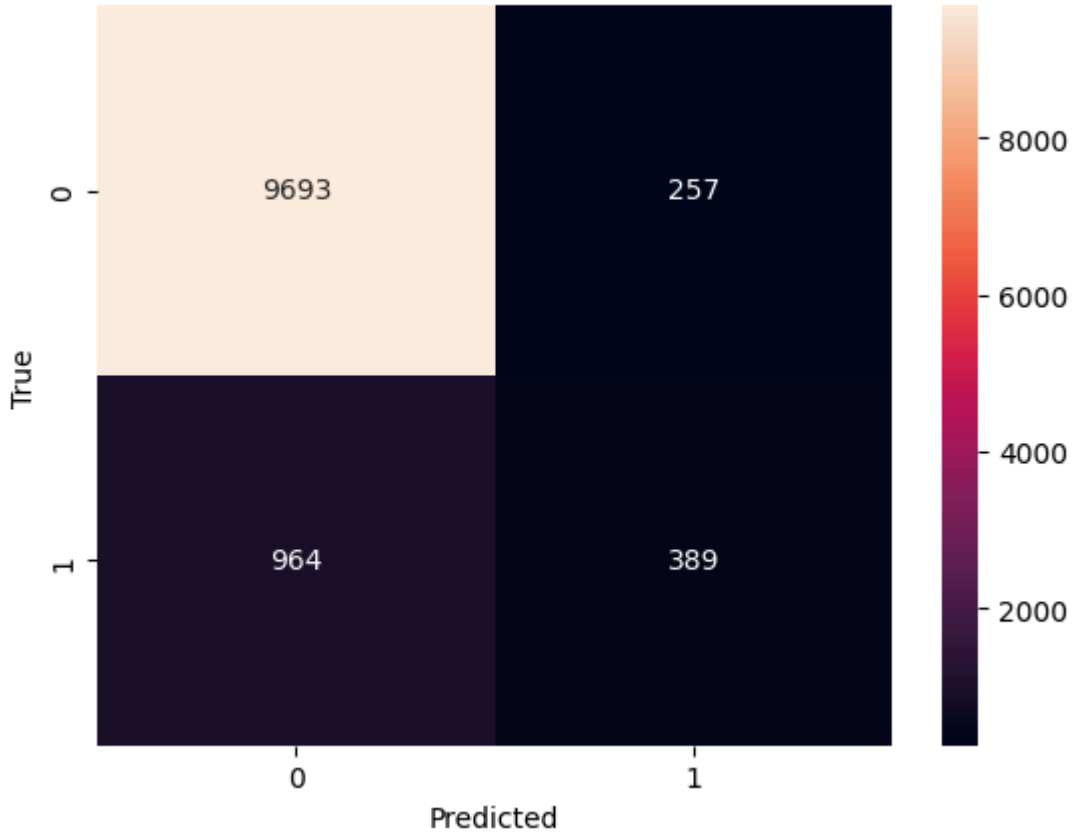
45211 rows × 32 columns

Class Weight Control

In [ ]:

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,recall_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
model=RandomForestClassifier(random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_dumy, y, random_state=42)
# Calculate class weights
classes = np.unique(y_train)
class_weights = compute_class_weight(class_weight='balanced', classes=classes, y
class_weights_dict = dict(zip(classes, class_weights))

# You can pass this dictionary to the model (if it supports it)
# Example with RandomForest
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(class_weight=class_weights_dict, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(classification_report(y_test,y_pred,target_names=['No','Yes']))
recall_score(y_test,y_pred)
confusion_matrix(y_test, y_pred)
cm=confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True,fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True');
```

```
              precision    recall  f1-score   support

          No       0.91      0.97      0.94      9950
         Yes       0.60      0.29      0.39      1353

    accuracy                           0.89     11303
   macro avg       0.76      0.63      0.66     11303
weighted avg       0.87      0.89      0.87     11303
```



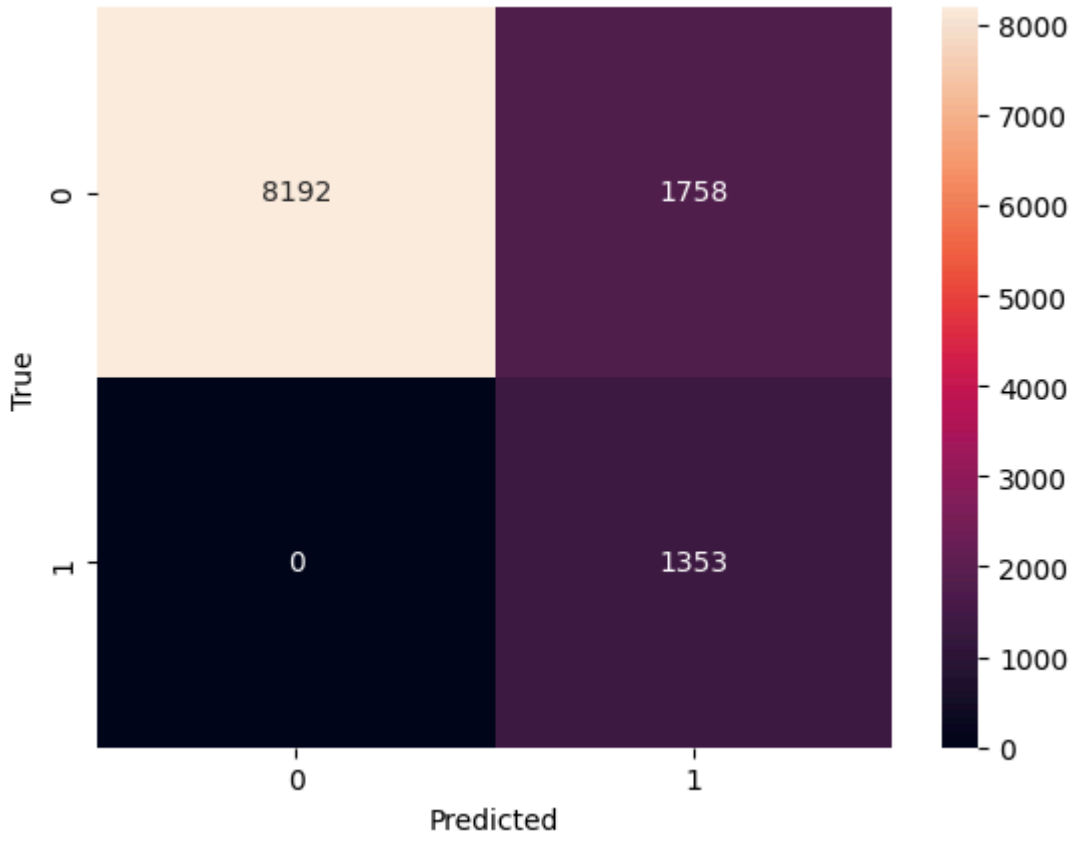
Undersampling

In [ ]:

```
from sklearn.datasets import fetch_openml
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

import numpy as np

train_data=pd.concat([X_dumy,y],axis=1)

majority_class = train_data[train_data.y == 0]
minority_class = train_data[train_data.y == 1]

# Downsample majority class
majority_class_downsampled = majority_class.sample(n=len(minority_class), random

# Combine minority class with downsampled majority class
downsampled_data = pd.concat([minority_class, majority_class_downsampled])

# Separate features and target
X_train_resampled = downsampled_data.drop('y', axis=1)
y_train_resampled = downsampled_data['y']
```

```
# Train a classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train_resampled, y_train_resampled)
# Predict on the test set
y_pred = model.predict(X_test)


print(classification_report(y_test,y_pred,target_names=['No','Yes']))
recall_score(y_test,y_pred)
confusion_matrix(y_test, y_pred)
cm=confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True,fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True');
```

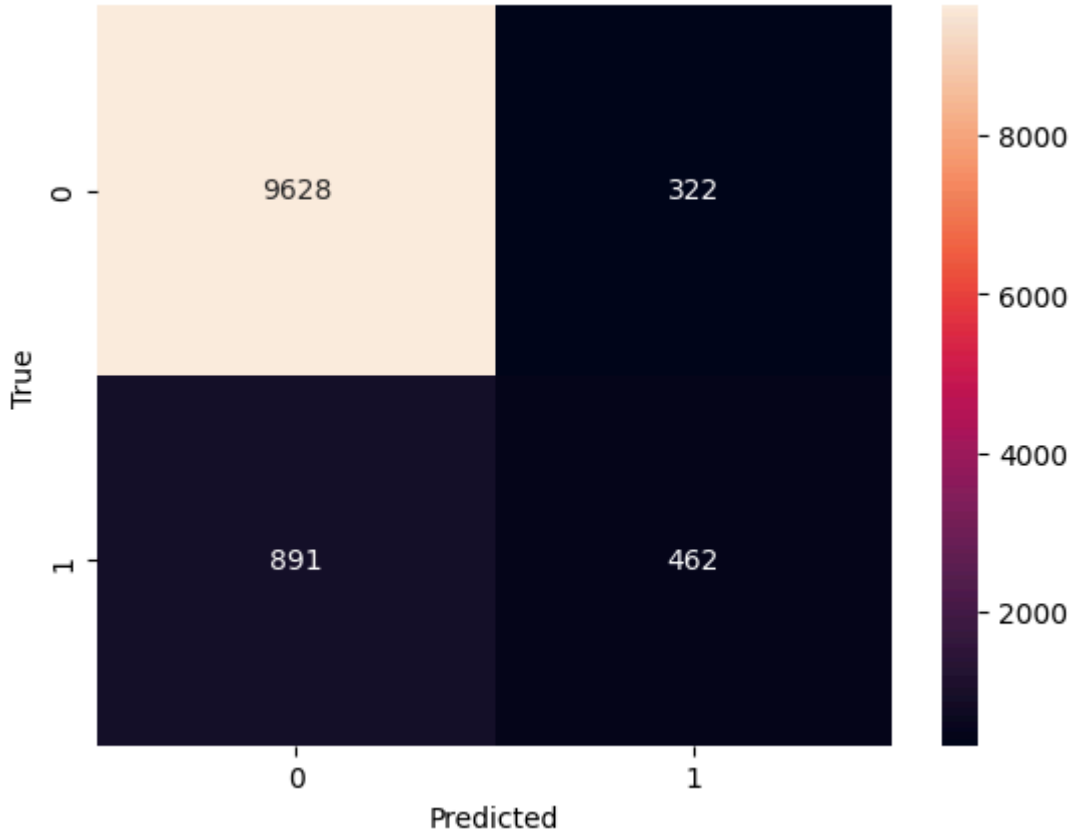
```
              precision    recall  f1-score   support

          No       1.00      0.82      0.90      9950
         Yes       0.43      1.00      0.61      1353

    accuracy                           0.84     11303
   macro avg       0.72      0.91      0.75     11303
weighted avg       0.93      0.84      0.87     11303
```



In [ ]:
```
# Tuning 안 하고 ML 확인
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,recall_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
model=RandomForestClassifier(random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_dumy, y, random_state=42)
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

```
accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred,target_names=['No','Yes']))
recall_score(y_test,y_pred)
confusion_matrix(y_test, y_pred)
cm=confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True,fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True');
```

```
              precision    recall  f1-score   support

          No       0.92      0.97      0.94      9950
         Yes       0.59      0.34      0.43      1353

    accuracy                           0.89     11303
   macro avg       0.75      0.65      0.69     11303
weighted avg       0.88      0.89      0.88     11303
```



In [ ]:
```
#pozitif olanlarin gercekte de ne kadarinin pozitif oldugu presision 462/ 322+ 4
#gercek pozitiflerin ne kadarinin bildigimize recall 462 / 462 + 891
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from scipy.stats import randint
```

In [ ]:
```
rf = RandomForestClassifier()
```

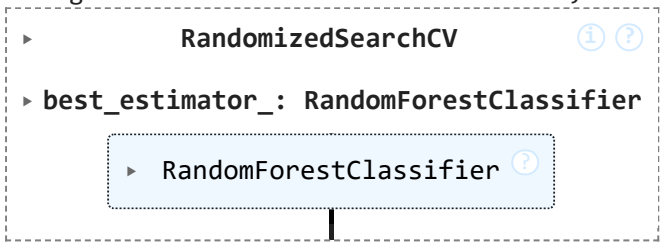
In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_dumy, y, random_state=42)
```

In [ ]:
```
param_distributions ={
    'n_estimators': randint(100, 1000),
    'max_depth': randint(1, 50),
```

```
    'min_samples_split': randint(2, 50),
    'min_samples_leaf': randint(1, 50),
    'max_features': randint(1,10),
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy'],
    'max_leaf_nodes':randint(10,100)
}

# now create a searchCV object and fit it to the data
search = RandomizedSearchCV(estimator=rf,
                                    param_distributions=param_distributions,
                                    n_iter=50,
                                    cv=3,
                                    verbose=2,
                                    random_state=42,
                                    n_jobs=-1
                                    )

search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[ ]:



In [ ]:

```
pd.DataFrame(search.cv_results_).sort_values('rank_test_score')
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_bootstrap	para
34	16.878517	0.523355	0.701458	0.068950	True	
48	26.399893	0.513529	0.432852	0.042912	False	
35	40.673534	0.920311	1.344099	0.040471	True	
30	23.482690	0.182988	1.000697	0.058127	True	
27	29.332383	0.128267	1.263619	0.148669	True	
7	7.002935	0.084675	0.211769	0.008709	False	
21	23.536376	1.285002	1.016613	0.111814	True	
24	40.153105	0.328346	1.505971	0.096430	True	
14	15.328000	0.332872	0.617347	0.050547	True	
10	31.691562	0.550389	1.233367	0.143330	False	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_bootstrap	para
31	28.072911	0.066415	0.648928	0.023910	False	
0	9.321733	0.255976	0.436830	0.066999	True	
3	31.745266	0.733120	0.743677	0.012222	False	
1	12.833006	0.345042	0.499996	0.099054	True	
15	37.578482	0.714068	1.173194	0.116906	True	
12	8.023871	0.269212	0.212102	0.014570	False	
41	6.829400	0.345045	0.283574	0.012624	False	
47	7.265239	0.495649	0.303027	0.007332	False	
5	10.398975	0.127870	0.406910	0.043090	True	
25	12.361599	1.109572	0.556512	0.125778	False	
29	9.800631	0.205022	0.437819	0.050114	False	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_bootstrap	para
16	7.391892	0.078498	0.254322	0.040249	False	
49	15.988572	0.566355	0.465090	0.066142	False	
36	12.739258	0.584453	0.482043	0.044277	False	
18	11.315067	0.343376	0.560833	0.028495	True	
11	21.258801	0.100918	1.216745	0.081318	True	
20	18.563012	0.287955	1.022930	0.153268	False	
38	7.272215	0.671248	0.454784	0.021593	True	
42	15.570350	0.167227	1.108368	0.033776	True	
22	13.220634	0.341332	0.844742	0.144203	False	
13	13.492575	0.243527	0.420874	0.018408	True	
32	5.688450	0.170243	0.412231	0.007298	True	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_bootstrap	para
19	7.458714	0.258959	0.539224	0.023606	True	
43	2.855030	0.160009	0.182845	0.005544	False	
17	13.879540	0.333704	1.016613	0.080253	False	
26	10.387439	0.680053	0.506107	0.032591	False	
46	2.941798	0.071282	0.117352	0.007115	False	
45	3.952428	0.237299	0.235370	0.021219	False	
44	6.402873	0.055081	0.750990	0.035411	False	
9	17.759827	0.539222	1.641608	0.047336	False	
23	5.159533	0.298811	0.507311	0.015064	True	
39	17.530107	0.206582	1.253981	0.103279	True	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_bootstrap	para
2	14.742404	0.206710	0.539889	0.076426	False	
33	13.908463	1.028296	1.454443	0.070992	False	
28	3.232686	0.062306	0.131314	0.003291	False	
8	15.749537	0.199292	0.675859	0.015286	False	
6	9.417476	0.146129	0.346407	0.017698	False	
40	2.923181	0.038298	0.152924	0.008476	False	
4	20.998013	0.137283	1.300189	0.094132	False	
37	12.157812	0.721481	0.753651	0.014844	True	

In [ ]:
```
search.best_params_
```

Out[ ]:
```
{'bootstrap': True,
 'criterion': 'gini',
 'max_depth': 28,
 'max_features': 9,
 'max_leaf_nodes': 98,
 'min_samples_leaf': 23,
 'min_samples_split': 32,
 'n_estimators': 397}
```

In [ ]:
```
search.score(X_test, y_test)
```

Out[ ]:
```
0.892240997965142
```

In [ ]:
```
y_pred_random=search.predict(X_test)
recall_score(y_test,y_pred_random)
```

Out[ ]:
```
0.2660753880266075
```

In [ ]:
```
a=confusion_matrix(y_test, y_pred_random)
b=confusion_matrix(y_test, y_pred)
a,b
a-b
```

Out[ ]:
```
(array([[9725,  225],
        [ 993,  360]], dtype=int64),
 array([[9628,  322],
        [ 891,  462]], dtype=int64))
```

Out[ ]:
```
array([[  97,  -97],
       [ 102, -102]], dtype=int64)
```

In [ ]:
```
search.best_score_
```

Out[ ]:
```
0.8965436857688566
```

In [ ]:
```
print(classification_report(y_test,y_pred_random,target_names=['No','Yes']))
```

```
              precision    recall  f1-score   support

          No       0.91      0.98      0.94      9950
         Yes       0.62      0.27      0.37      1353

    accuracy                           0.89     11303
   macro avg       0.76      0.62      0.66     11303
weighted avg       0.87      0.89      0.87     11303
```

월래 기본값들 **default params**와 하면 **no recall= 0.97 yes recall = 0.34** 튜닝 다음에 **no recall 0.98 yes recall 0.27** 로 됐습니다.

- https://github.com/FurkanBeyazit/mage-ai
- https://github.com/FurkanBeyazit/mage-ai/blob/main/data_loaders/bank_load.py
- https://github.com/FurkanBeyazit/mage-ai/blob/main/transformers/bank_transformer.py
- https://github.com/FurkanBeyazit/mage-ai/blob/main/custom/bank_ml.py