In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set_theme(color_codes=True)
```

In [6]:
```
df = pd.read_csv('bank-additional-full.csv', delimiter=';')
pd.set_option('display.max_columns', None)
df.head()
```

Out[6]:

ɟ	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
ɔ	no	telephone	may	mon	261	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0
ɔ	no	telephone	may	mon	149	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0
s	no	telephone	may	mon	226	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0
ɔ	no	telephone	may	mon	151	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0
ɔ	yes	telephone	may	mon	307	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

Exploratory Data Analysis

In [9]:
```
# select categorial data
df_categorial = df[['job','marital','education','default','housing',
                    'loan','contact','month','day_of_week','poutcome','y']]
df_categorial.head()
```

Out[9]:

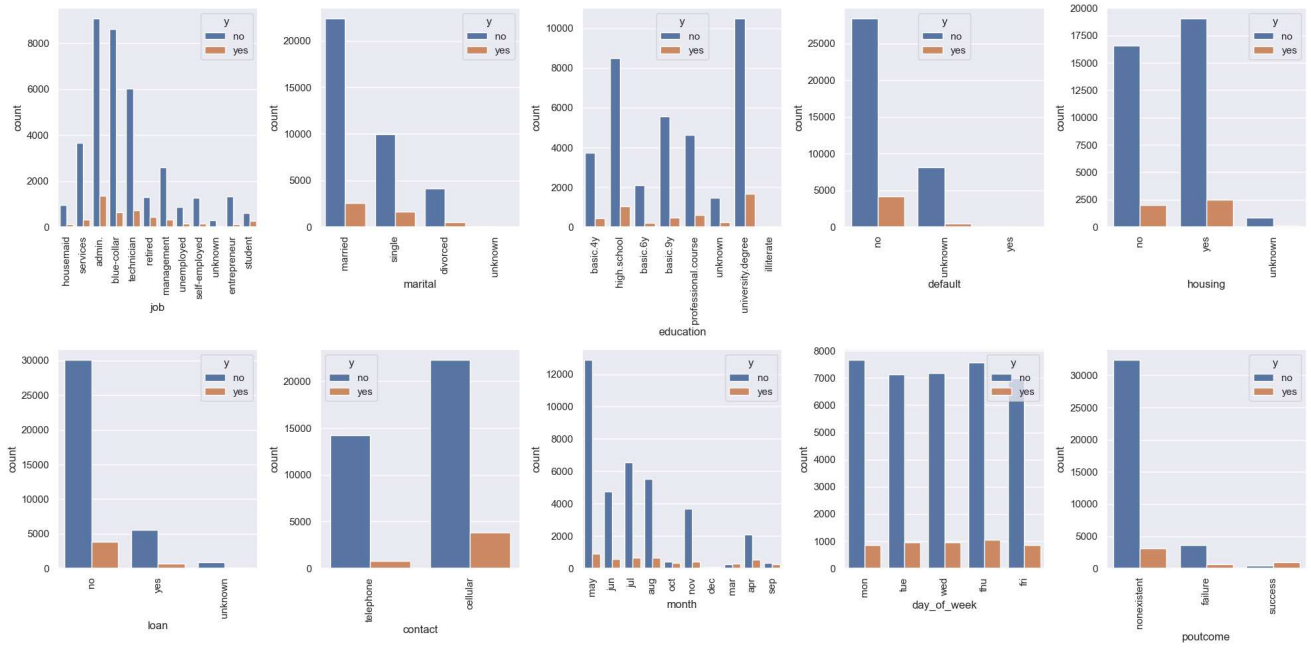
	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	y
0	housemaid	married	basic.4y	no	no	no	telephone	may	mon	nonexistent	no
1	services	married	high.school	unknown	no	no	telephone	may	mon	nonexistent	no
2	services	married	high.school	no	yes	no	telephone	may	mon	nonexistent	no
3	admin.	married	basic.6y	no	no	no	telephone	may	mon	nonexistent	no
4	services	married	high.school	no	no	yes	telephone	may	mon	nonexistent	no

In [11]:
```
#select numeric data
df_numeric = df[['age','duration','campaign','pdays','previous','emp.var.rate',
                 'cons.price.idx','cons.conf.idx','euribor3m','nr.employed']]
df_numeric.head()
```

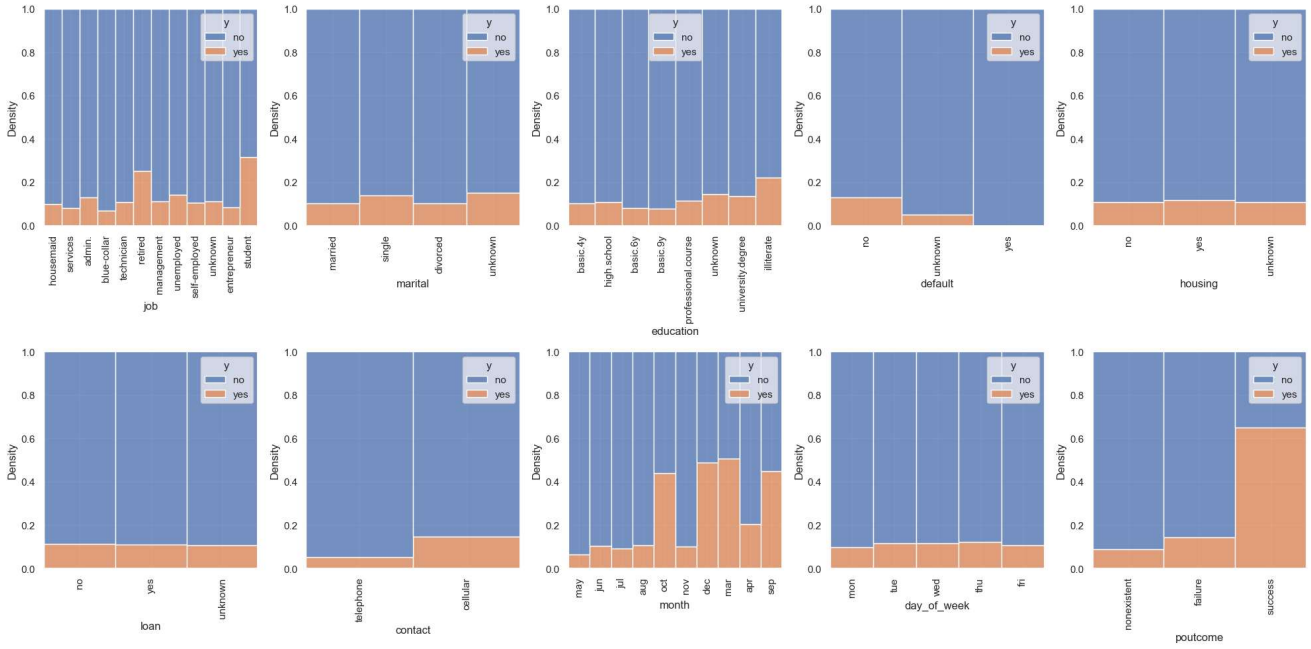
Out[11]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0
1	57	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0
2	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0
3	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0
4	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0

In [22]:
```
cat_vars = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

# create countplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='y', data=df_categorial, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



In [34]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='y', data=df_categorial, ax=axs[i], multiple="fill", kde=False, element="bars", fill=True, stat='de
    axs[i].set_xticklabels(df_categorial[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```

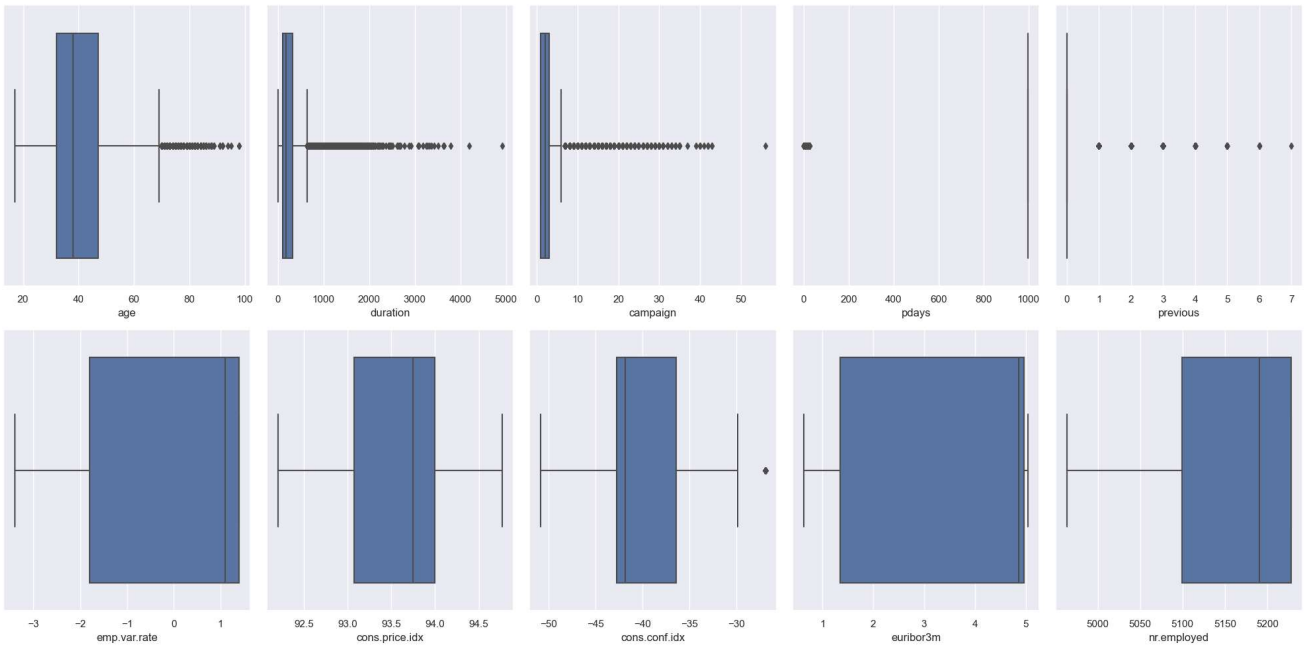


1. Most people that subsribed the bank term deposit are : retired and student

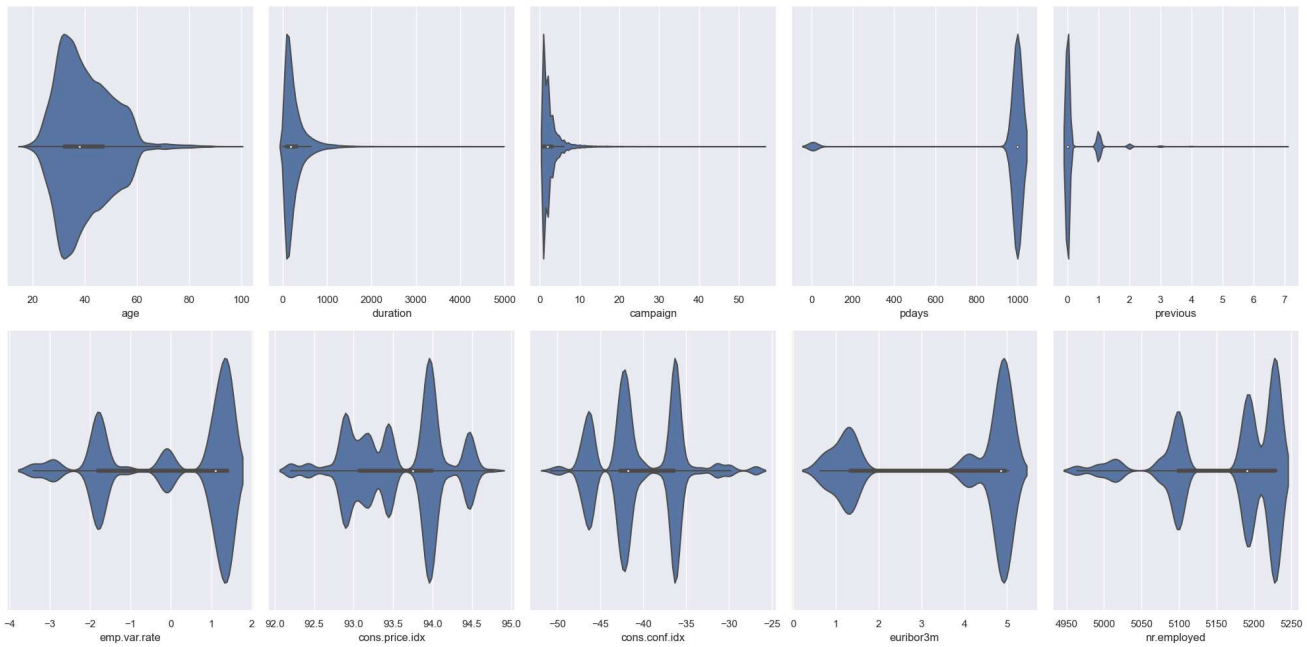
2. Most people that subsribed the bank term deposit are last contact in : October, December, March, September

3. Most people that subsribed the bank term deposit are succesfull previous marketing campaign

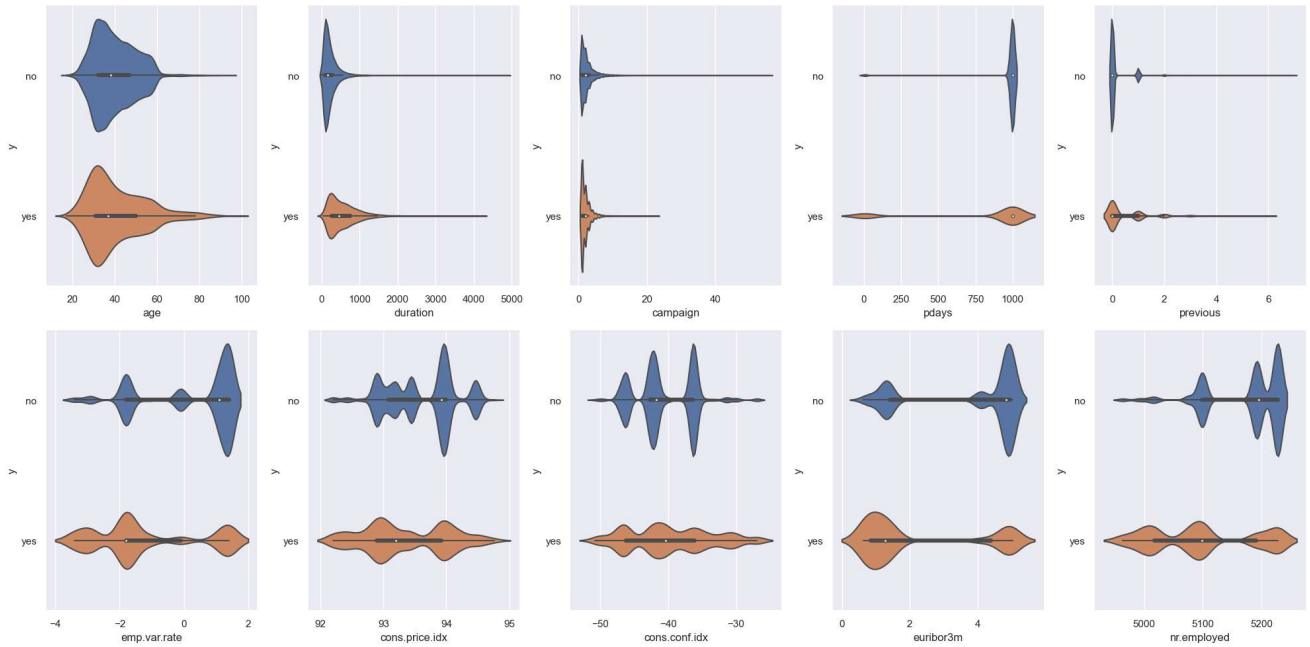
In [35]:
```
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



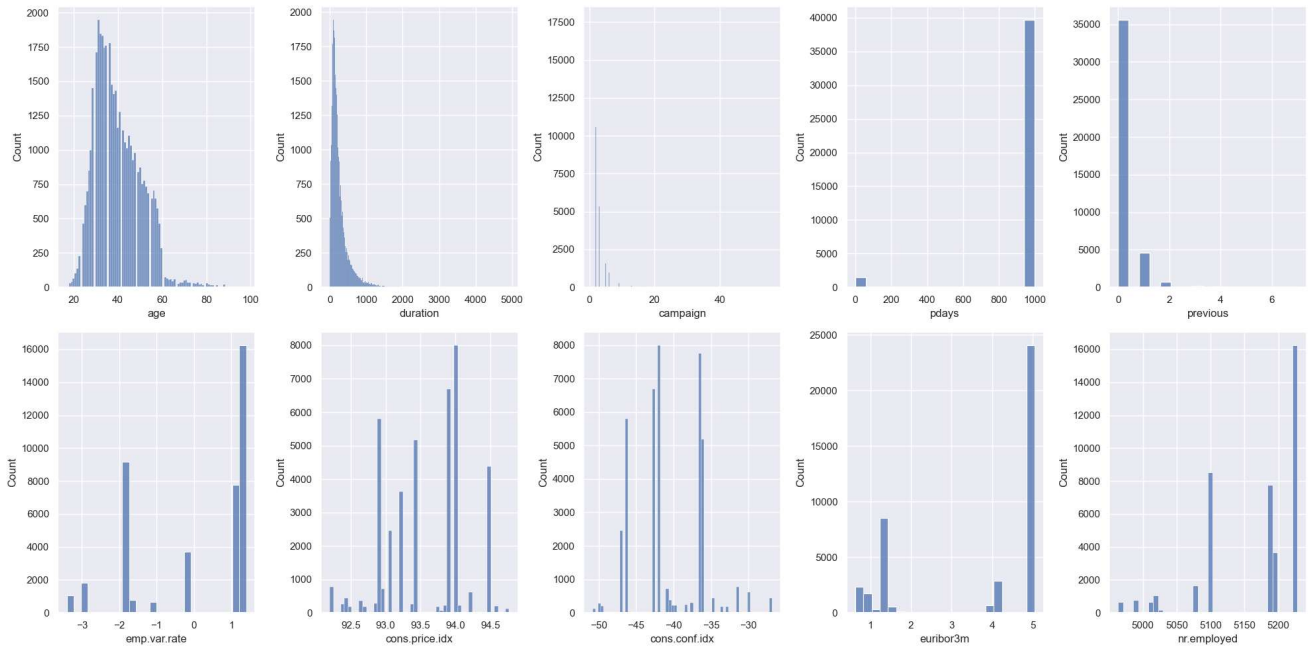
In [45]:
```
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



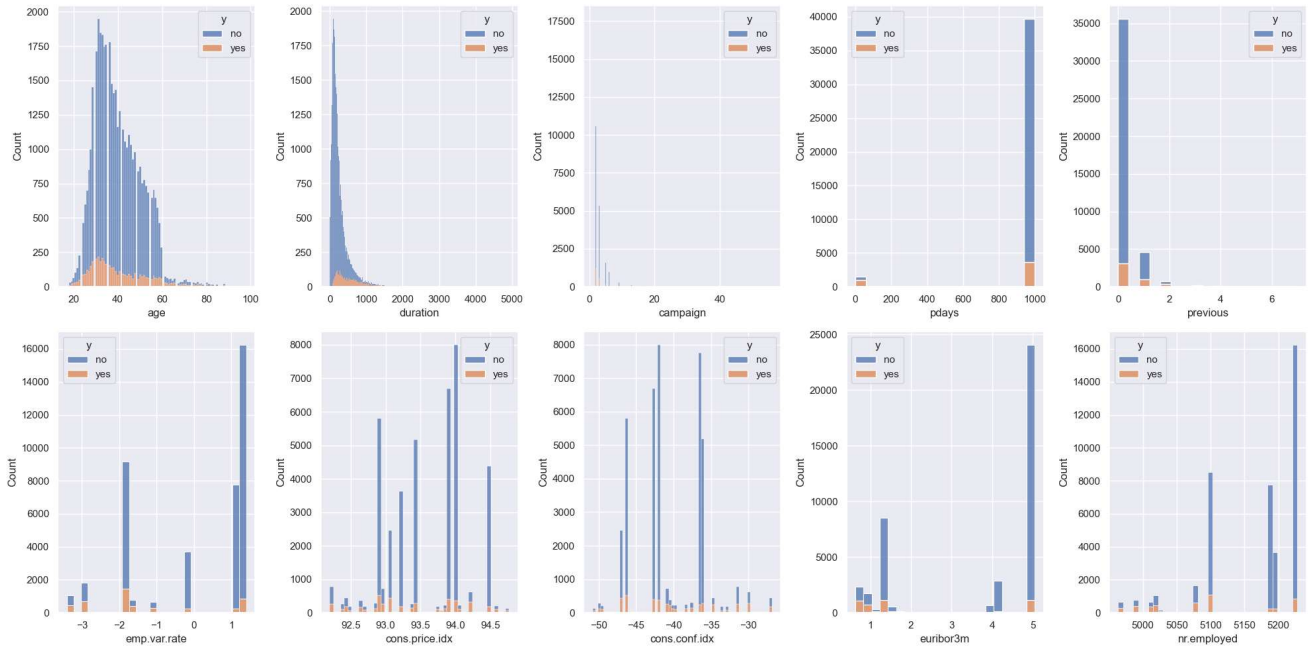
In [44]:
```
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, y='y', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [46]:
```
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```

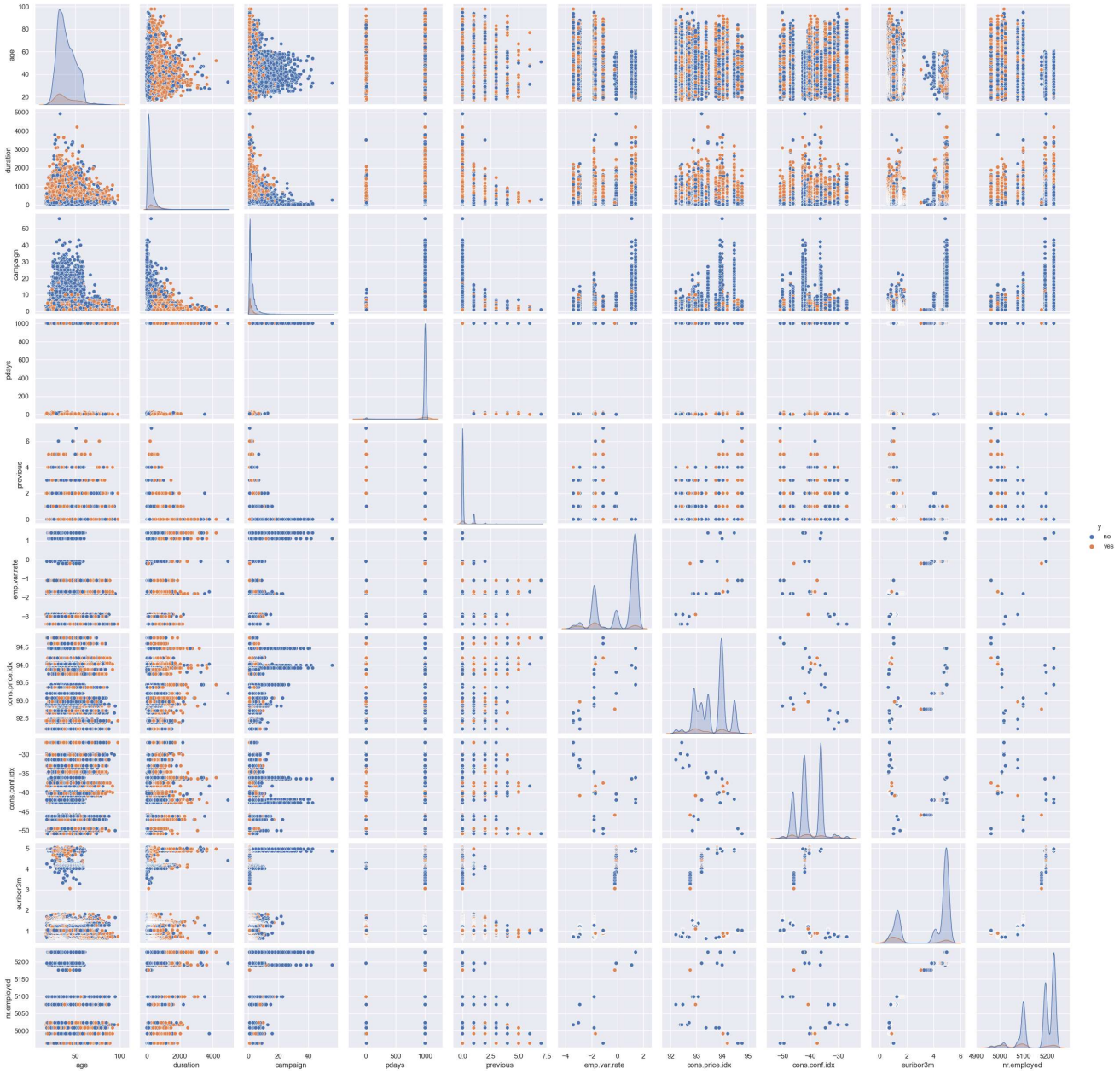


In [48]:
```
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',

fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, hue='y', data=df, ax=axs[i], multiple="stack")

fig.tight_layout()

plt.show()
```



In [50]:
```
# get list of numerical variables
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',

# create scatter plot matrix
sns.pairplot(df, hue='y')
```

Out[50]:
```
<seaborn.axisgrid.PairGrid at 0x1bbebbb90d0>
```



Data Preprocessing

In [51]:
```
df['job'].unique()
```

Out[51]:
```
array(['housemaid', 'services', 'admin.', 'blue-collar', 'technician',
       'retired', 'management', 'unemployed', 'self-employed', 'unknown',
       'entrepreneur', 'student'], dtype=object)
```

In [52]:
```
df['marital'].unique()
```

Out[52]:
```
array(['married', 'single', 'divorced', 'unknown'], dtype=object)
```

In [53]:
```
df['education'].unique()
```

Out[53]:
```
array(['basic.4y', 'high.school', 'basic.6y', 'basic.9y',
       'professional.course', 'unknown', 'university.degree',
       'illiterate'], dtype=object)
```

In [54]:
```
df['default'].unique()
```

Out[54]:
```
array(['no', 'unknown', 'yes'], dtype=object)
```

In [55]:
```
df['housing'].unique()
```

Out[55]:
```
array(['no', 'yes', 'unknown'], dtype=object)
```

In [56]:
```
df['loan'].unique()
```

Out[56]:
```
array(['no', 'yes', 'unknown'], dtype=object)
```

In [57]:
```
df['contact'].unique()
```

Out[57]:
```
array(['telephone', 'cellular'], dtype=object)
```

In [58]:
```
df['month'].unique()
```

Out[58]:
```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'mar', 'apr',
       'sep'], dtype=object)
```

In [59]:
```
df['day_of_week'].unique()
```

Out[59]:
```
array(['mon', 'tue', 'wed', 'thu', 'fri'], dtype=object)
```

In [60]:
```
df['poutcome'].unique()
```

Out[60]:
```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

In [61]:
```
df['y'].unique()
```

Out[61]:
```
array(['no', 'yes'], dtype=object)
```

In [62]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['job']= label_encoder.fit_transform(df['job'])
df['job'].unique()
```

Out[62]:
```
array([ 3,  7,  0,  1,  9,  5,  4, 10,  6, 11,  2,  8])
```

In [63]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['marital']= label_encoder.fit_transform(df['marital'])
df['marital'].unique()
```

Out[63]:
```
array([1, 2, 0, 3])
```

In [64]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['education']= label_encoder.fit_transform(df['education'])
df['education'].unique()
```

Out[64]:
```
array([0, 3, 1, 2, 5, 7, 6, 4])
```

In [65]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['default']= label_encoder.fit_transform(df['default'])
df['default'].unique()
```

Out[65]:
```
array([0, 1, 2])
```

In [66]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['default']= label_encoder.fit_transform(df['default'])
df['default'].unique()
```

Out[66]:
```
array([0, 1, 2], dtype=int64)
```

In [67]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['housing']= label_encoder.fit_transform(df['housing'])
df['housing'].unique()
```

Out[67]: array([0, 2, 1])

In [68]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['loan']= label_encoder.fit_transform(df['loan'])
df['loan'].unique()
```

Out[68]: array([0, 2, 1])

In [69]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['contact']= label_encoder.fit_transform(df['contact'])
df['contact'].unique()
```

Out[69]: array([1, 0])

In [71]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['month']= label_encoder.fit_transform(df['month'])
df['month'].unique()
```

Out[71]: array([6, 4, 3, 1, 8, 7, 2, 5, 0, 9])

In [72]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['day_of_week']= label_encoder.fit_transform(df['day_of_week'])
df['day_of_week'].unique()
```

Out[72]: array([1, 3, 4, 2, 0])

In [73]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['poutcome']= label_encoder.fit_transform(df['poutcome'])
df['poutcome'].unique()
```

Out[73]: array([1, 0, 2])

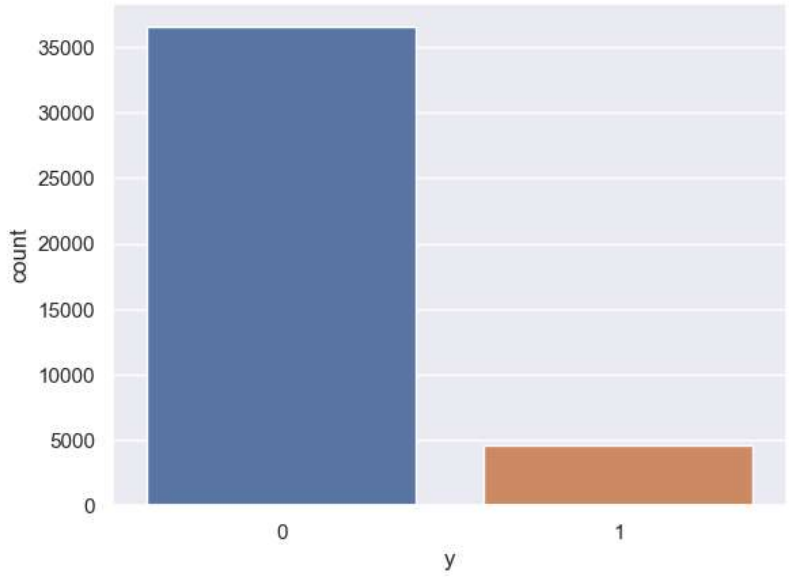
In [74]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['y']= label_encoder.fit_transform(df['y'])
df['y'].unique()
```

Out[74]: array([0, 1])

Balance the Label : 'y' label

In [75]:
```
sns.countplot(df['y'])
df['y'].value_counts()
```

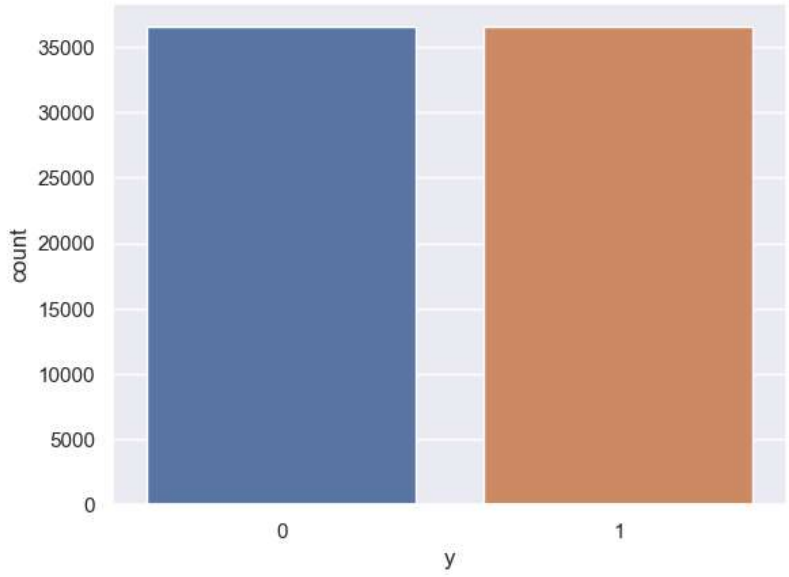
Out[75]:
```
0    36548
1     4640
Name: y, dtype: int64
```



In [76]:
```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df[(df['y']==0)]
df_minority = df[(df['y']==1)]
# upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,    # sample with replacement
                                 n_samples= 36548, # to match majority class
                                 random_state=0)  # reproducible results
# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])
```

In [77]:
```
sns.countplot(df_upsampled['y'])
df_upsampled['y'].value_counts()
```

Out[77]:
```
1    36548
0    36548
Name: y, dtype: int64
```



Remove outlier using IQR, because there are extreme value in some variables

In [79]:
```
# define a function to remove outliers using IQR
def remove_outliers_iqr(df, columns):
    for col in columns:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# specify the columns to remove outliers from
columns_to_check = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'eur

# call the function to remove outliers using IQR
df_clean = remove_outliers_iqr(df_upsampled, columns_to_check)

# print the resulting dataframe
df_clean.head()
```

Out[79]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	co
37017	25	8	2	7	1	2	0	0	3	3	371	1	999	0	1	-2.9	
36682	51	9	2	6	0	0	0	0	4	0	657	1	999	0	1	-2.9	
29384	45	7	2	7	0	0	0	1	0	0	541	1	999	0	1	-1.8	
21998	29	9	2	3	1	0	0	0	1	4	921	3	999	0	1	1.4	
16451	37	10	2	2	1	2	2	0	3	4	633	1	999	0	1	1.4	

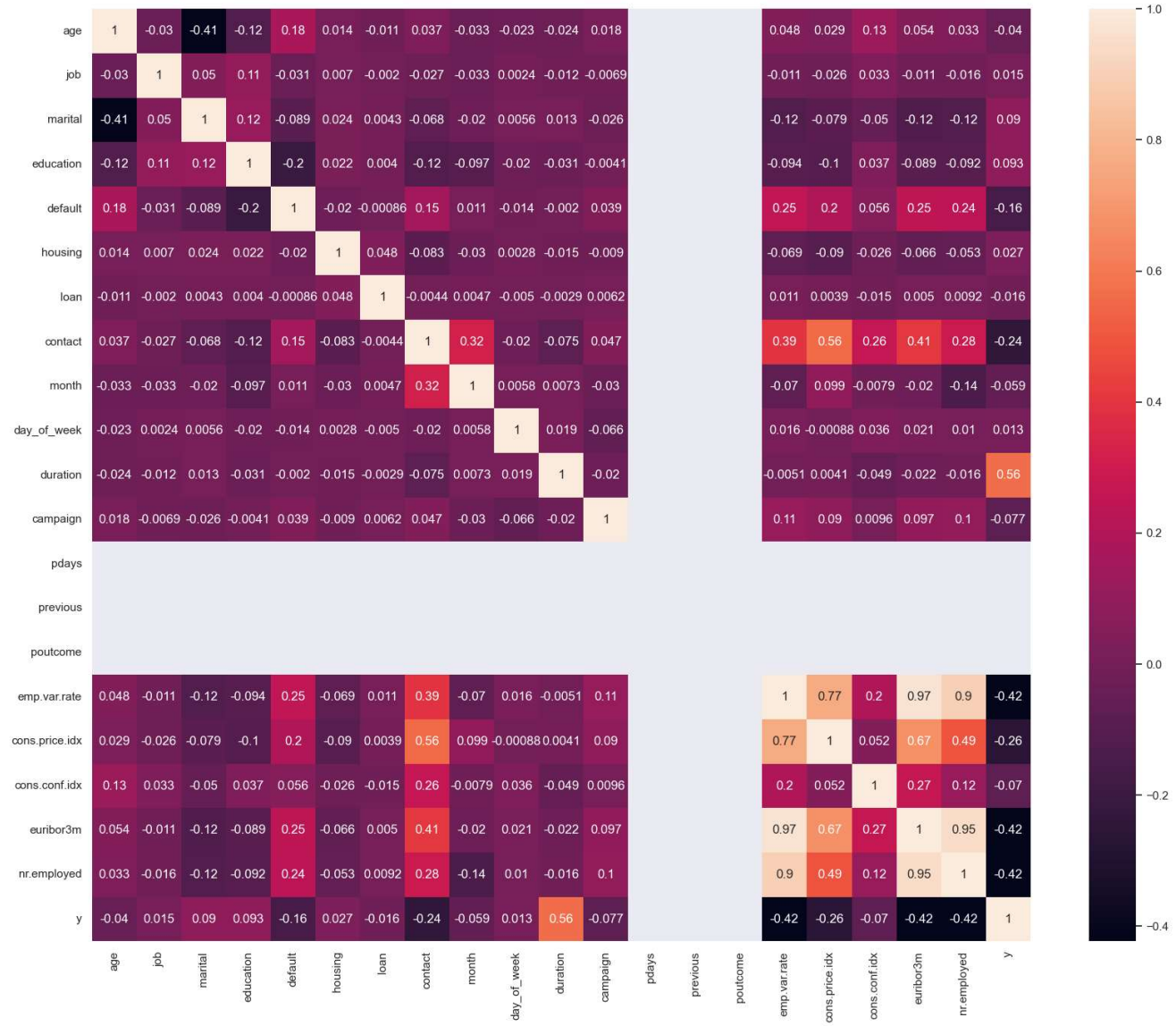
In [80]:
```
df_clean.shape
```

Out[80]:
```
(49702, 21)
```

Heatmap Correlation

In [82]:
```
plt.figure(figsize=(20, 16))
sns.heatmap(df_clean.corr(), fmt='.2g', annot=True)
```

Out[82]: <AxesSubplot:>



Machine Learning Model Building

In [83]:
```
X = df_clean.drop('y', axis=1)
y = df_clean['y']
```

In [84]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Decision Tree

In [85]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X_train, y_train)
```

Out[85]: DecisionTreeClassifier(random_state=0)

In [86]:
```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 96.49 %
```

In [87]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.9648928679207324
Precision Score :  0.9648928679207324
Recall Score :  0.9648928679207324
Jaccard Score :  0.9321671525753158
Log Loss :  1.2125851345877936
```

Random Forest

In [92]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_depth=6)
rfc.fit(X_train, y_train)
```

Out[92]:
```
RandomForestClassifier(max_depth=6, random_state=0)
```

In [93]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 88.09 %
```

In [94]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8808972940348053
Precision Score :  0.8808972940348053
Recall Score :  0.8808972940348053
Jaccard Score :  0.7871460674157303
Log Loss :  4.113732431115118
```

Logistic Regression

In [95]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
```

Out[95]:
```
LogisticRegression(random_state=0)
```

In [96]:
```
y_pred = lr.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 84.44 %
```

In [97]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8443818529323006
Precision Score :  0.8443818529323006
Recall Score :  0.8443818529323006
Jaccard Score :  0.7306754874651811
Log Loss :  5.37492054942205
```

Support Vector Machine

In [98]:
```
from sklearn import svm
support = svm.SVC()
support.fit(X_train, y_train)
```

Out[98]:
```
SVC()
```

In [99]:
```
y_pred = support.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 83.16 %
```

In [100]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8316064782215069
Precision Score :  0.8316064782215069
Recall Score :  0.8316064782215069
Jaccard Score :  0.7117520447696943
Log Loss :  5.816170140331706
```

K Nearest Neighbor

In [101]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
```

Out[101]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [102]:
```
y_pred = neigh.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 94.4 %
```

In [103]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.9439694195754954
Precision Score :  0.9439694195754954
Recall Score :  0.9439694195754954
Jaccard Score :  0.8938845494379882
Log Loss :  1.9352711230839517
```

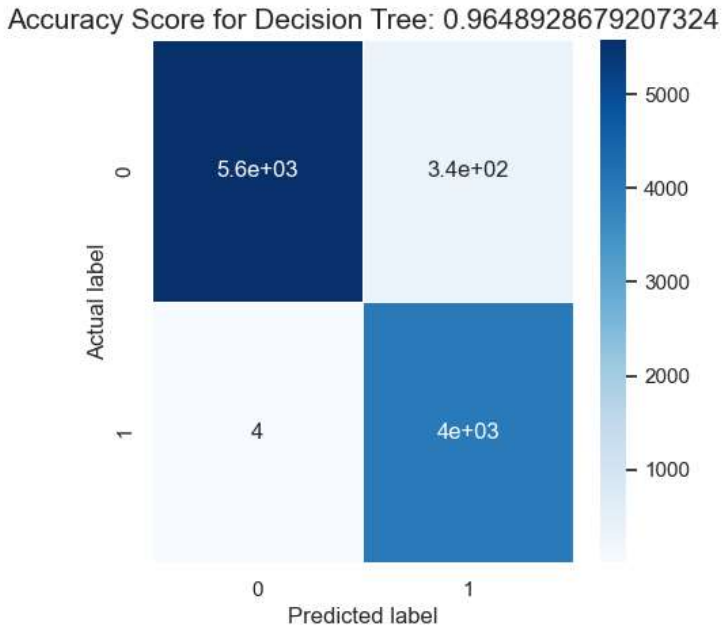
Confusion Matrix for Each Algorithm

Decision Tree

In [104]:
```
dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X_train, y_train)
y_pred = dtree.predict(X_test)
```

In [105]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[105]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.9648928679207324')
```

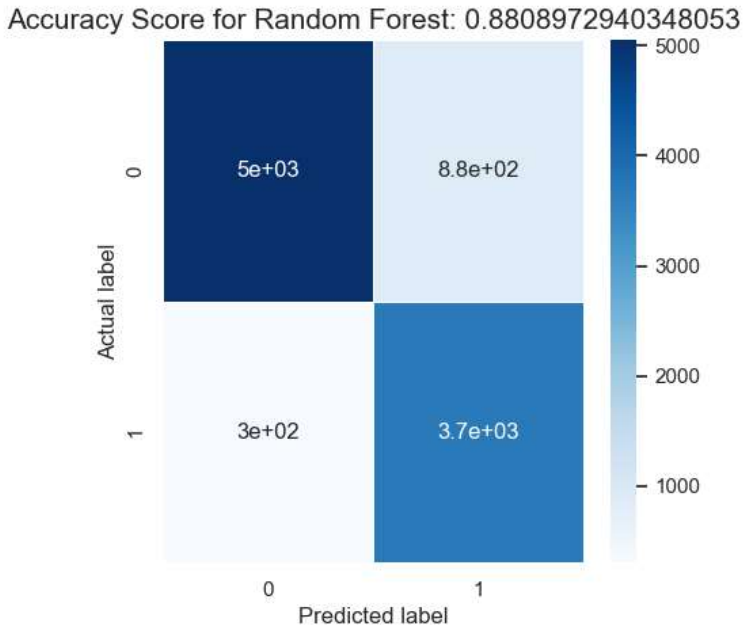


Random Forest

In [106]:
```
rfc = RandomForestClassifier(random_state=0, max_depth=6)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
```

In [108]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[108]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.8808972940348053')
```

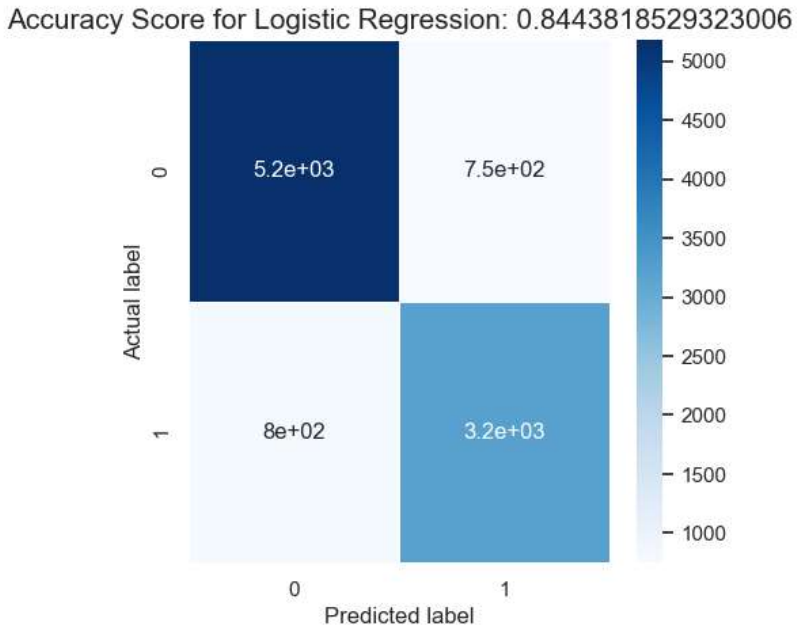


Logistic Regression

In [109]:
```
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

In [111]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Logistic Regression: {0}'.format(lr.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[111]:
```
Text(0.5, 1.0, 'Accuracy Score for Logistic Regression: 0.8443818529323006')
```

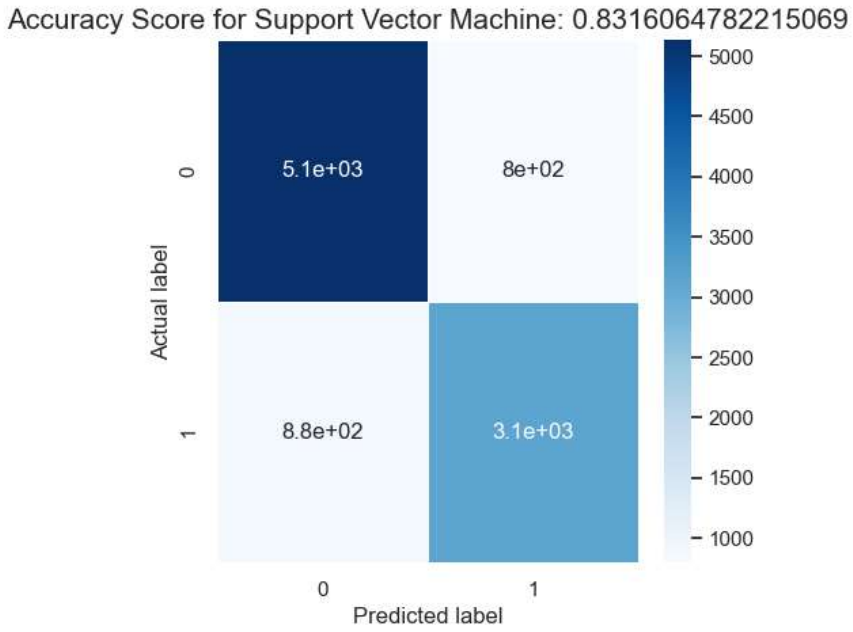


Support Vector Machine

In [112]:
```
support = svm.SVC()
support.fit(X_train, y_train)
y_pred = support.predict(X_test)
```

In [113]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Support Vector Machine: {0}'.format(support.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[113]:
```
Text(0.5, 1.0, 'Accuracy Score for Support Vector Machine: 0.8316064782215069')
```

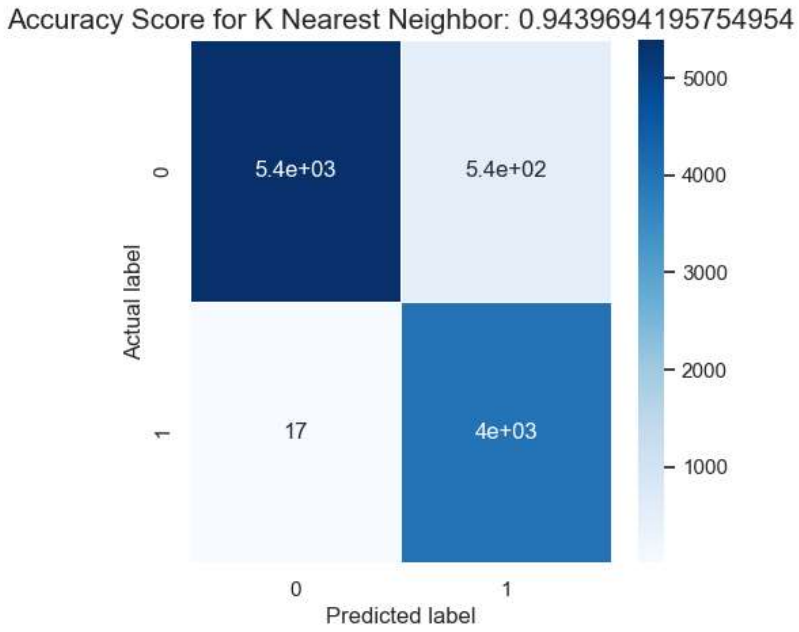


K Nearest Neighbor

In [114]:
```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
```

In [115]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for K Nearest Neighbor: {0}'.format(neigh.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[115]: Text(0.5, 1.0, 'Accuracy Score for K Nearest Neighbor: 0.9439694195754954')



Feature Importances

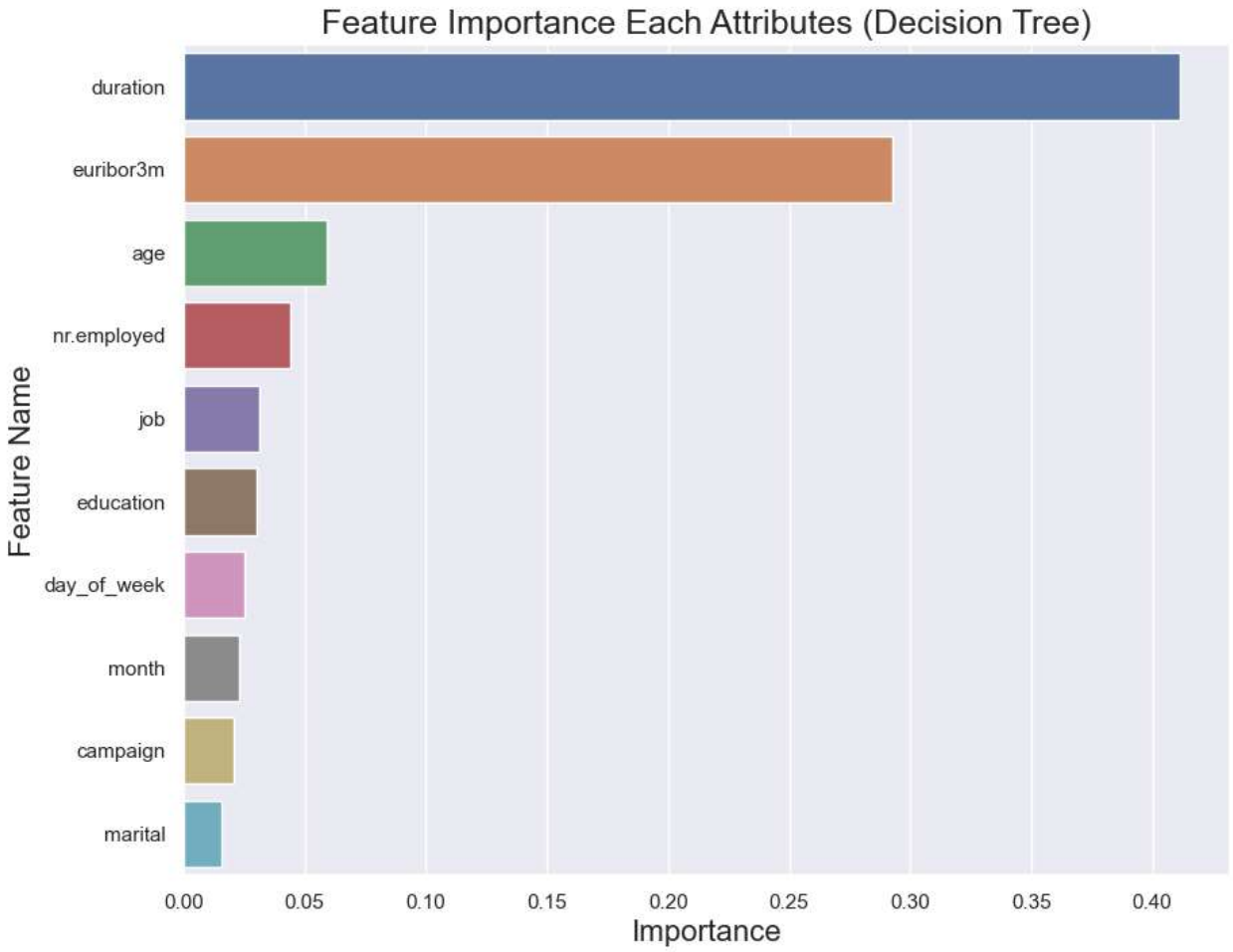
Decision Tree

In [116]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)
fi
```

Out[116]:

	Feature Name	Importance
10	duration	0.411272
18	euribor3m	0.292738
0	age	0.058949
19	nr.employed	0.044276
1	job	0.031216
3	education	0.030205
9	day_of_week	0.025408
8	month	0.022781
11	campaign	0.020640
2	marital	0.015413
5	housing	0.011015
16	cons.price.idx	0.007870
6	loan	0.007486
17	cons.conf.idx	0.006360
4	default	0.004892
7	contact	0.004843
15	emp.var.rate	0.004636
12	pdays	0.000000
13	previous	0.000000
14	poutcome	0.000000

In [117]:
```
fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [118]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)
fi
```

Out[118]:

	Feature Name	Importance
10	duration	0.460042
18	euribor3m	0.162768
19	nr.employed	0.109897
15	emp.var.rate	0.091251
17	cons.conf.idx	0.043510
8	month	0.041889
16	cons.price.idx	0.041883
7	contact	0.014444
4	default	0.009906
0	age	0.008962
3	education	0.004092
9	day_of_week	0.003498
11	campaign	0.002661
1	job	0.002191
2	marital	0.002081
6	loan	0.000529
5	housing	0.000396
12	pdays	0.000000
13	previous	0.000000
14	poutcome	0.000000

In [119]:
```
fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```