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 [2]:
```
df = pd.read_csv('Assignment-2_Data.csv')
df.head()
```

Out[2]:

	Id	age	job	marital	education	default	balance	housing	loan	contact	day
0	1001	999.0	management	married	tertiary	no	2143.0	yes	no	unknown	5
1	1002	44.0	technician	single	secondary	no	29.0	yes	no	unknown	5
2	1003	33.0	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5
3	1004	47.0	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5
4	1005	33.0	unknown	single	unknown	no	1.0	no	no	unknown	5

Data Preprocessing Part 1

In [3]:
```
# Drop identifier column
df.drop(columns = 'Id', inplace=True)
df.head()
```

Out[3]:

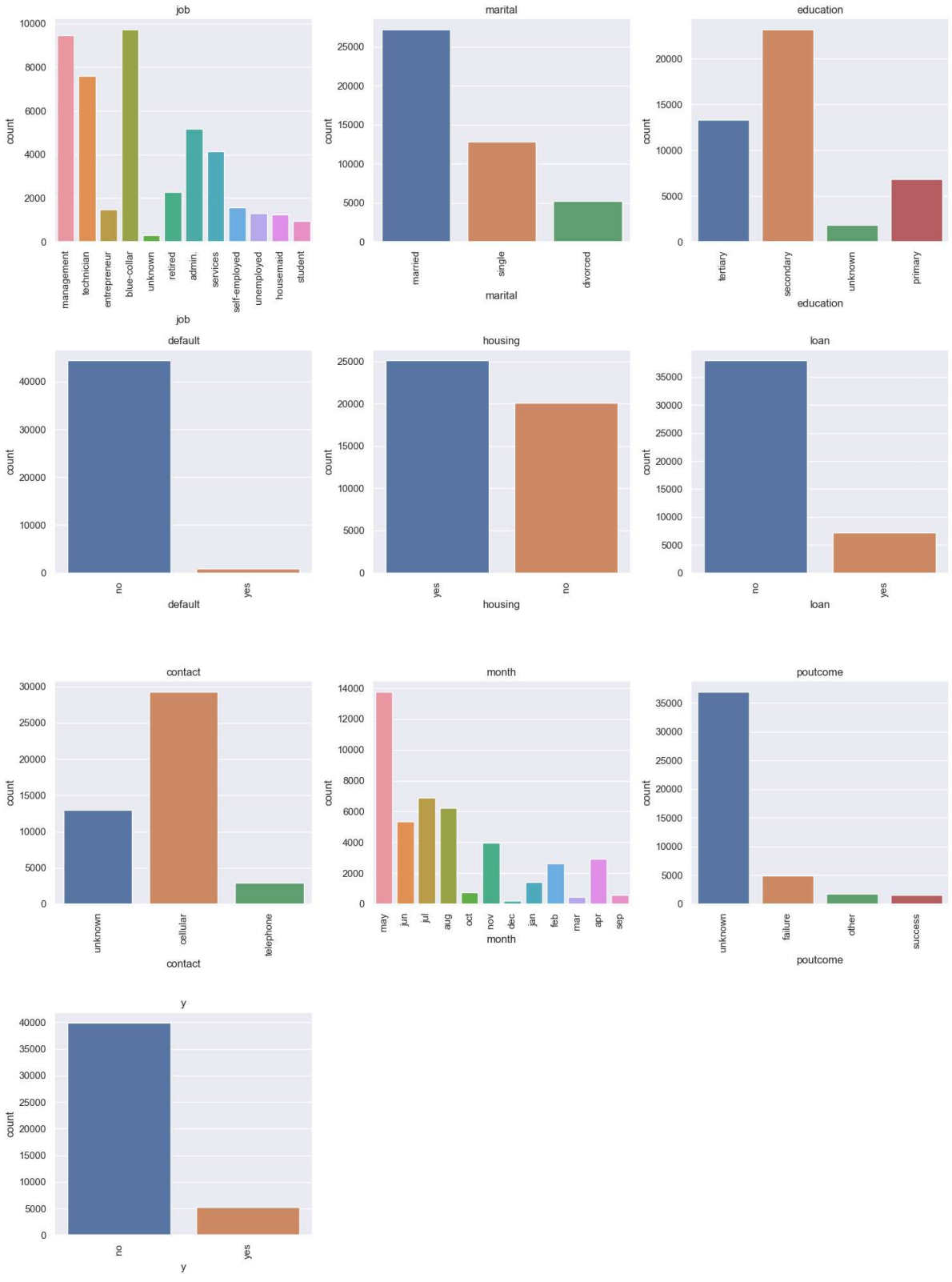
	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	999.0	management	married	tertiary	no	2143.0	yes	no	unknown	5	may
1	44.0	technician	single	secondary	no	29.0	yes	no	unknown	5	may
2	33.0	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5	may
3	47.0	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5	may
4	33.0	unknown	single	unknown	no	1.0	no	no	unknown	5	may

In [4]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

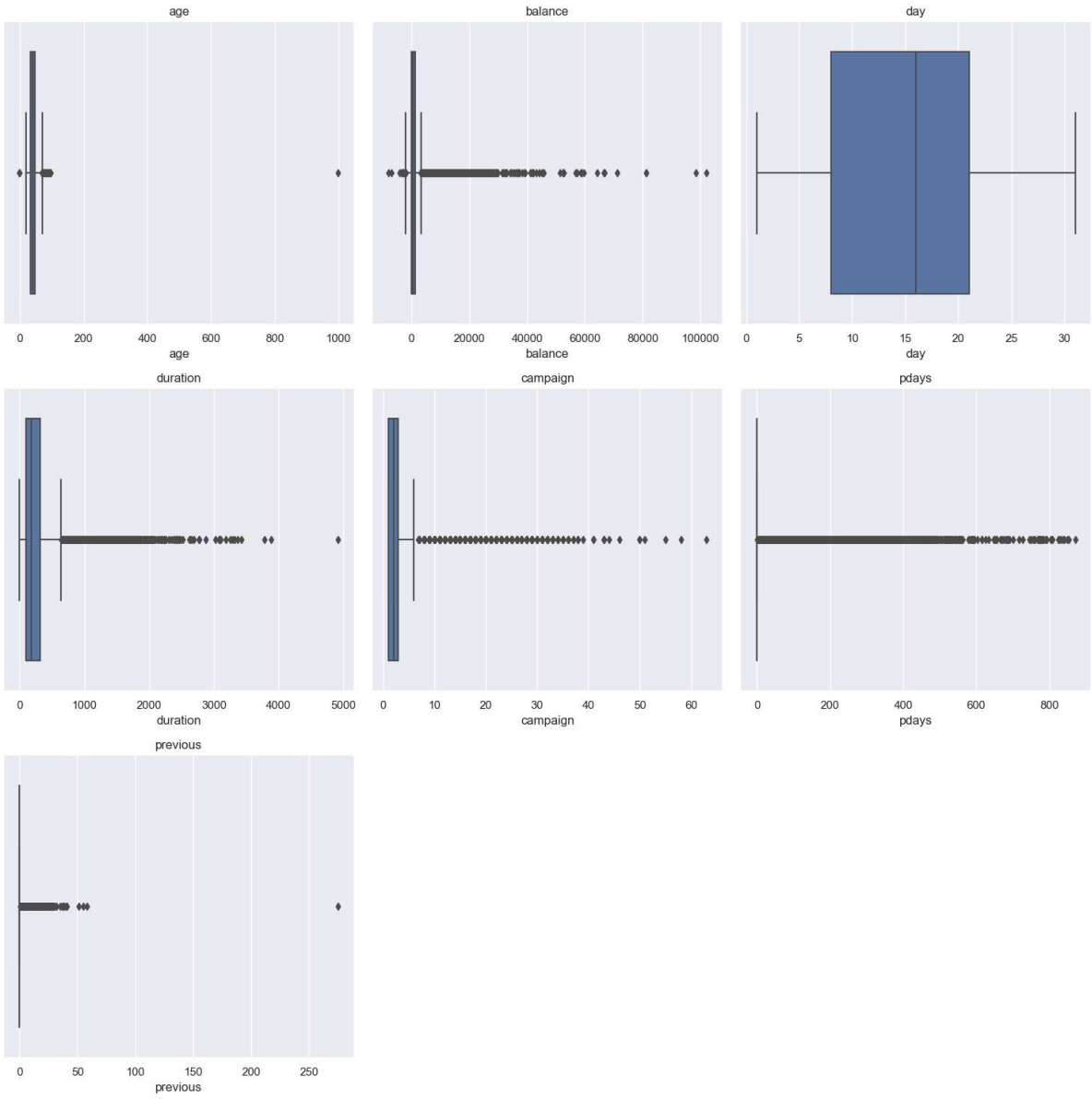
Out[4]:
```
job          12
marital       3
education     4
default       2
housing       2
loan          2
contact       3
month        12
poutcome      4
y             2
dtype: int64
```

Exploratory Data Analysis

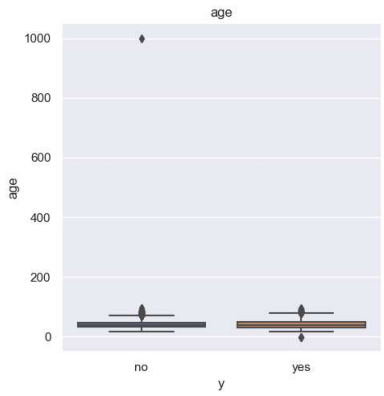
In [5]:
```
# Get the names of all columns with data type 'object' (categorical columns)
cat_vars = df.select_dtypes(include='object').columns.tolist()

# Create a figure with subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a countplot for the top 5 values of each categorical variable using Se
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(12).index
    filtered_df = df[df[var].isin(top_values)]
    sns.countplot(x=var, data=filtered_df, ax=axs[i])
    axs[i].set_title(var)
    axs[i].tick_params(axis='x', rotation=90)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

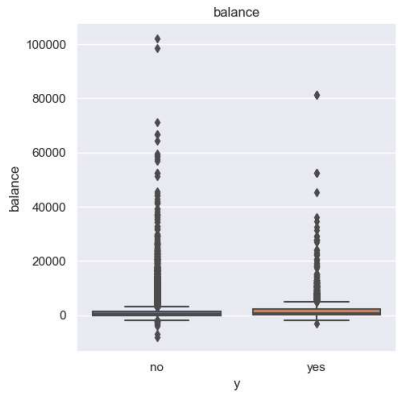

job
count
management
technician
entrepreneur
blue-collar
unknown
retired
admin.
services
self-employed
unemployed
housemaid
student
job
marital
married
single
divorced
education
tertiary
secondary
unknown
primary
default
no
yes
housing
loan
contact
unknown
cellular
telephone
month
may
jun
jul
aug
oct
nov
dec
jan
feb
mar
apr
sep
poutcome
failure
other
sucess
y

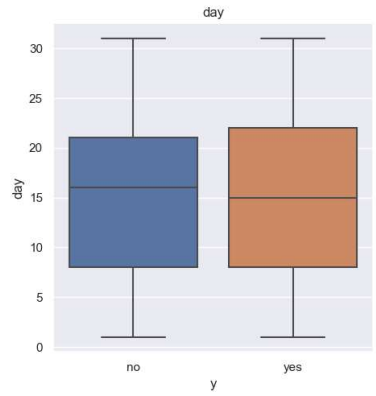
In [6]:
```
# Get the names of all columns with data type 'int' or 'float'
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(num_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each numerical variable using Seaborn
for i, var in enumerate(num_vars):
    sns.boxplot(x=df[var], ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

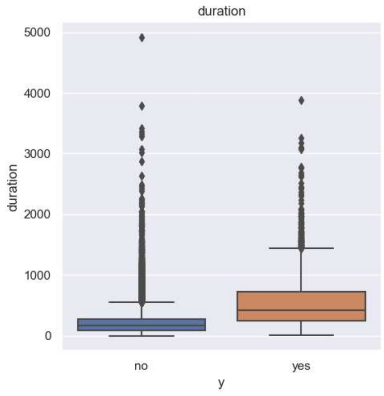

age
0
200
400
600
800
1000
age
balance
0
20000
40000
60000
80000
100000
balance
day
0
5
10
15
20
25
30
day
duration
0
1000
2000
3000
4000
5000
duration
campaign
0
10
20
30
40
50
60
campaign
pdays
0
200
400
600
800
pdays
previous
0
50
100
150
200
250
previous

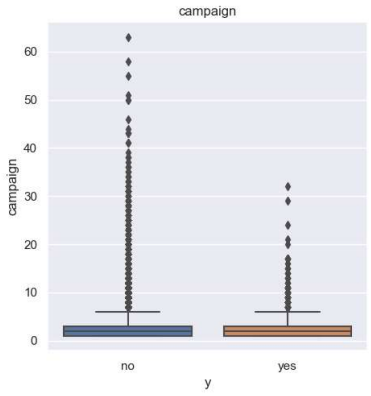
In [7]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each integer variable using Seaborn with hue='attritio
for i, var in enumerate(int_vars):
    sns.boxplot(y=var, x='y', data=df, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

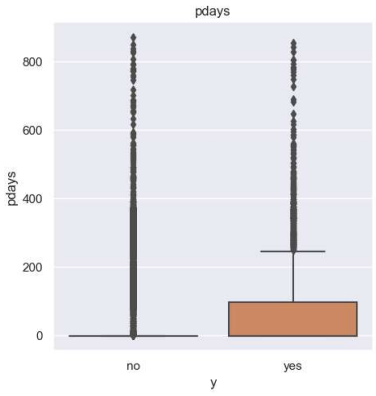

age
1000
800
600
400
200
0
age
no
yes
y

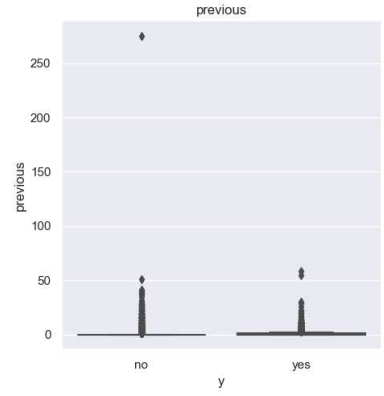

balance
100000
80000
60000
40000
20000
0
balance
no
yes
y


day
30
25
20
15
10
5
0
day
no
yes
y

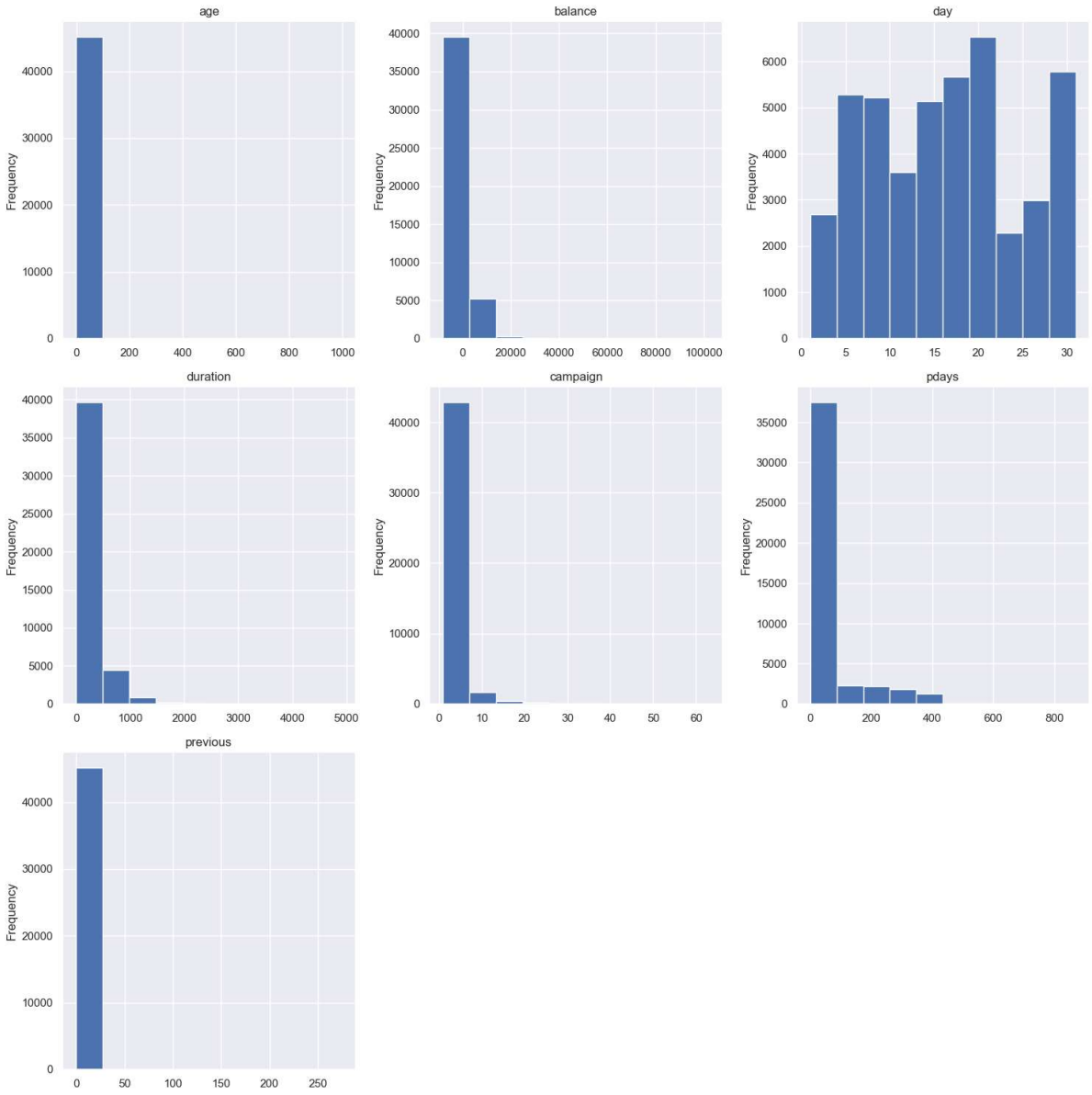

duration
5000
4000
3000
2000
1000
0
duration
no
yes
y


campaign
60
50
40
30
20
10
0
campaign
no
yes
y

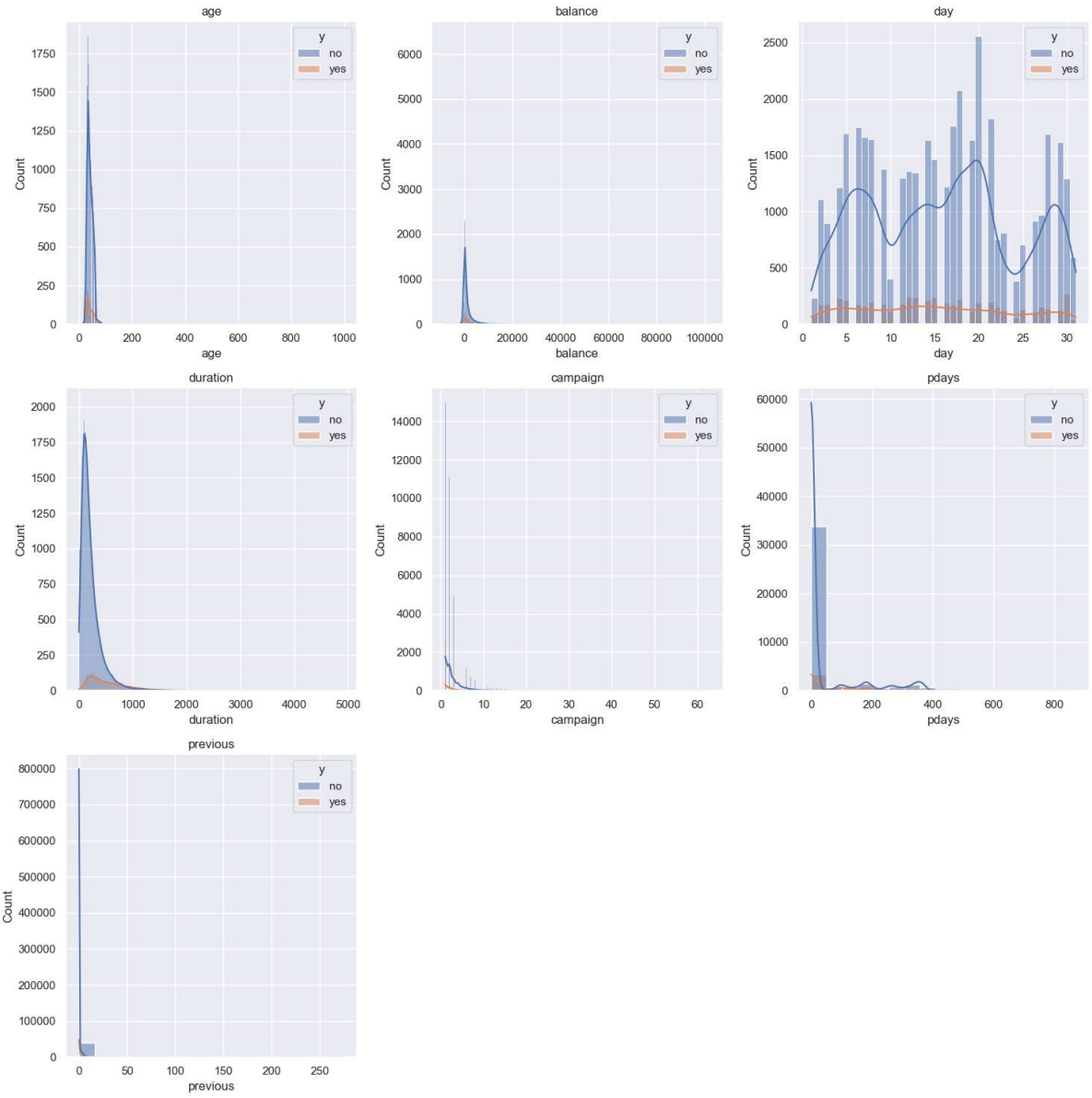

pdays
800
600
400
200
0
pdays
no
yes
y


previous
250
200
150
100
50
0
previous
no
yes
y

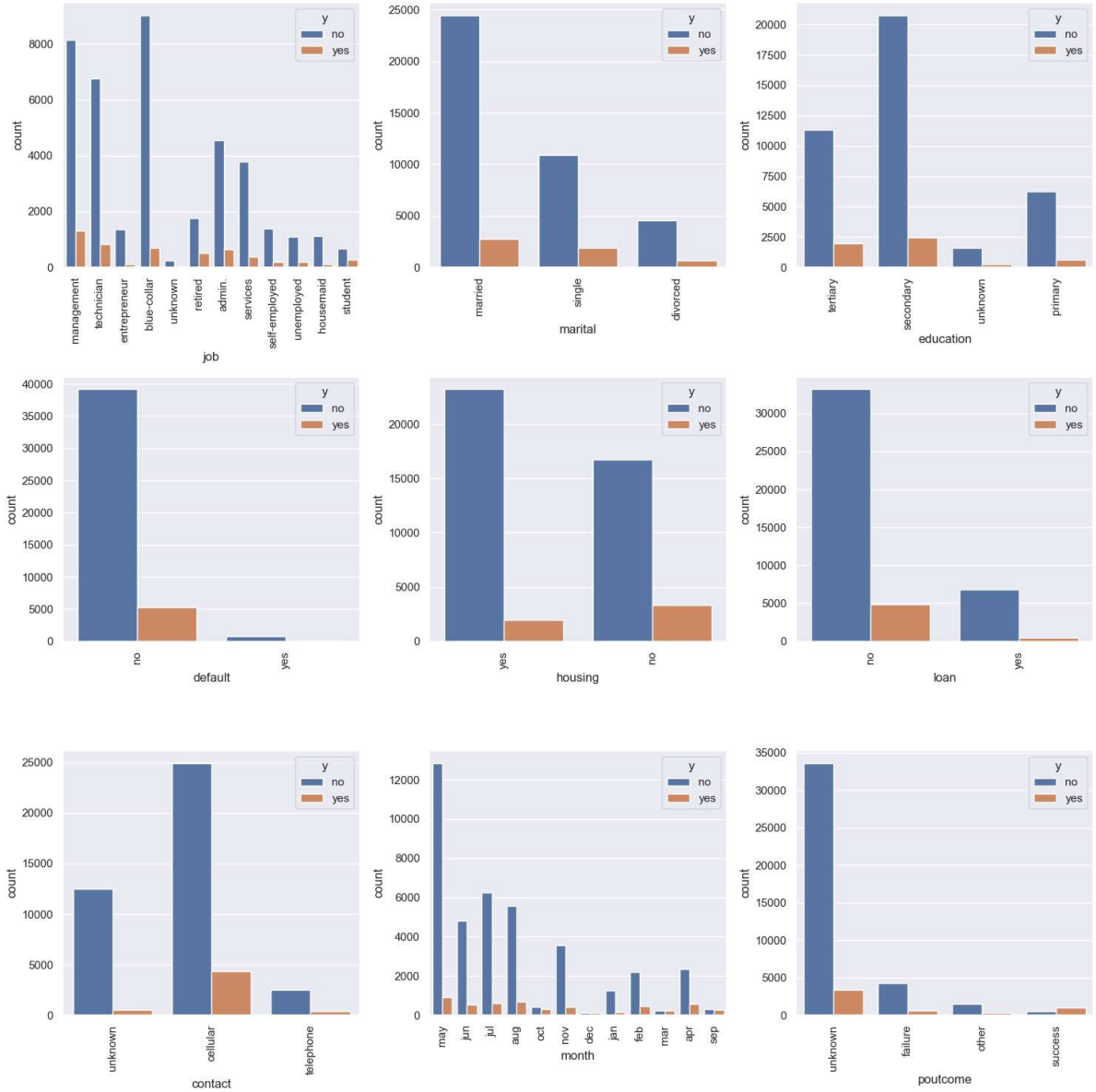
In [8]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable
for i, var in enumerate(int_vars):
    df[var].plot.hist(ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


age
Frequency
0
10000
20000
30000
40000
0
200
400
600
800
1000
balance
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
0
20000
40000
60000
80000
100000
day
Frequency
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25
30
duration
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
0
1000
2000
3000
4000
5000
campaign
Frequency
0
10000
20000
30000
40000
0
10
20
30
40
50
60
pdays
Frequency
0
5000
10000
15000
20000
25000
30000
35000
0
200
400
600
800
previous
Frequency
0
10000
20000
30000
40000
0
50
100
150
200
250

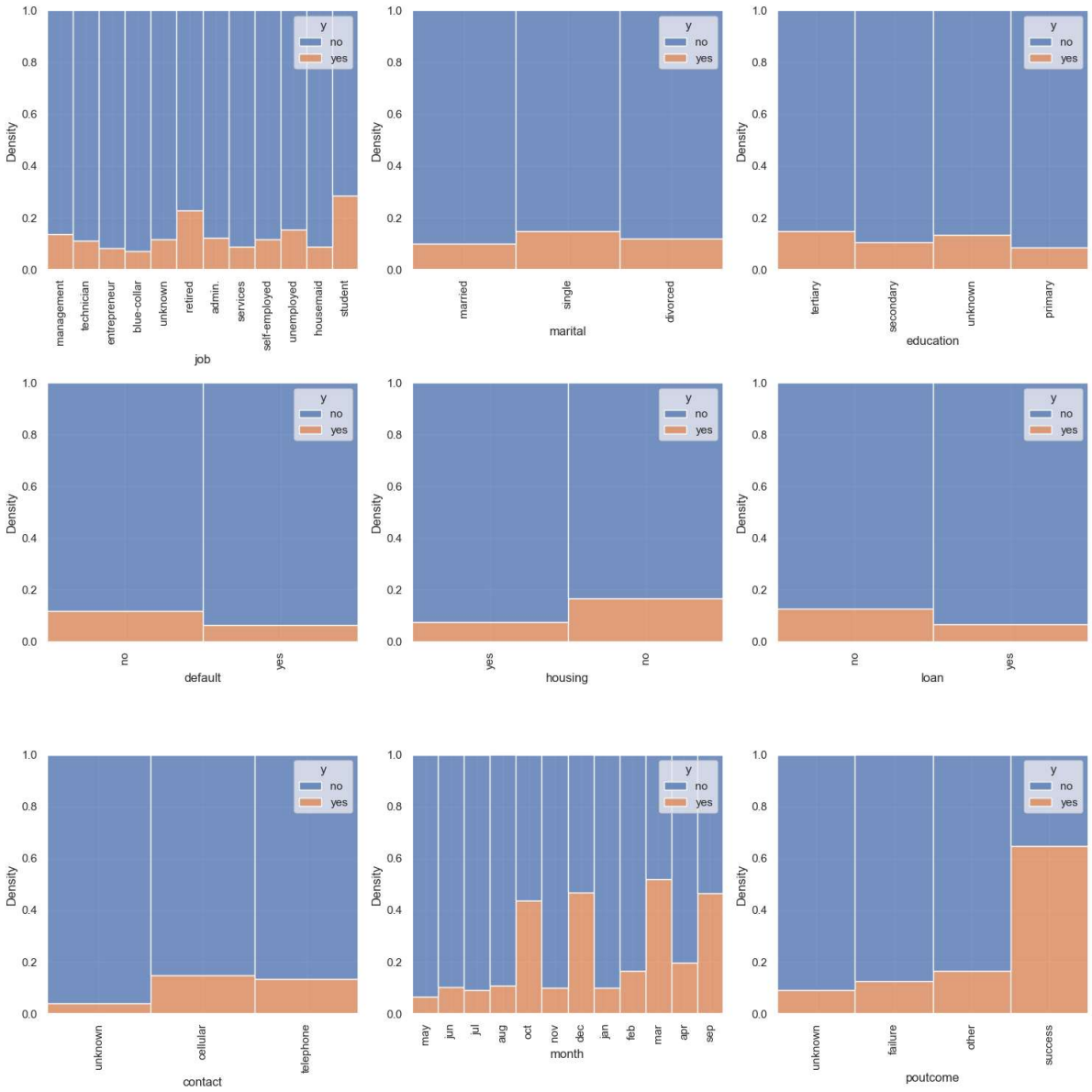
In [9]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the s
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable with hue='Attrition'
for i, var in enumerate(int_vars):
    sns.histplot(data=df, x=var, hue='y', kde=True, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


age
balance
day
duration
campaign
pdays
previous
y
no
yes
Count

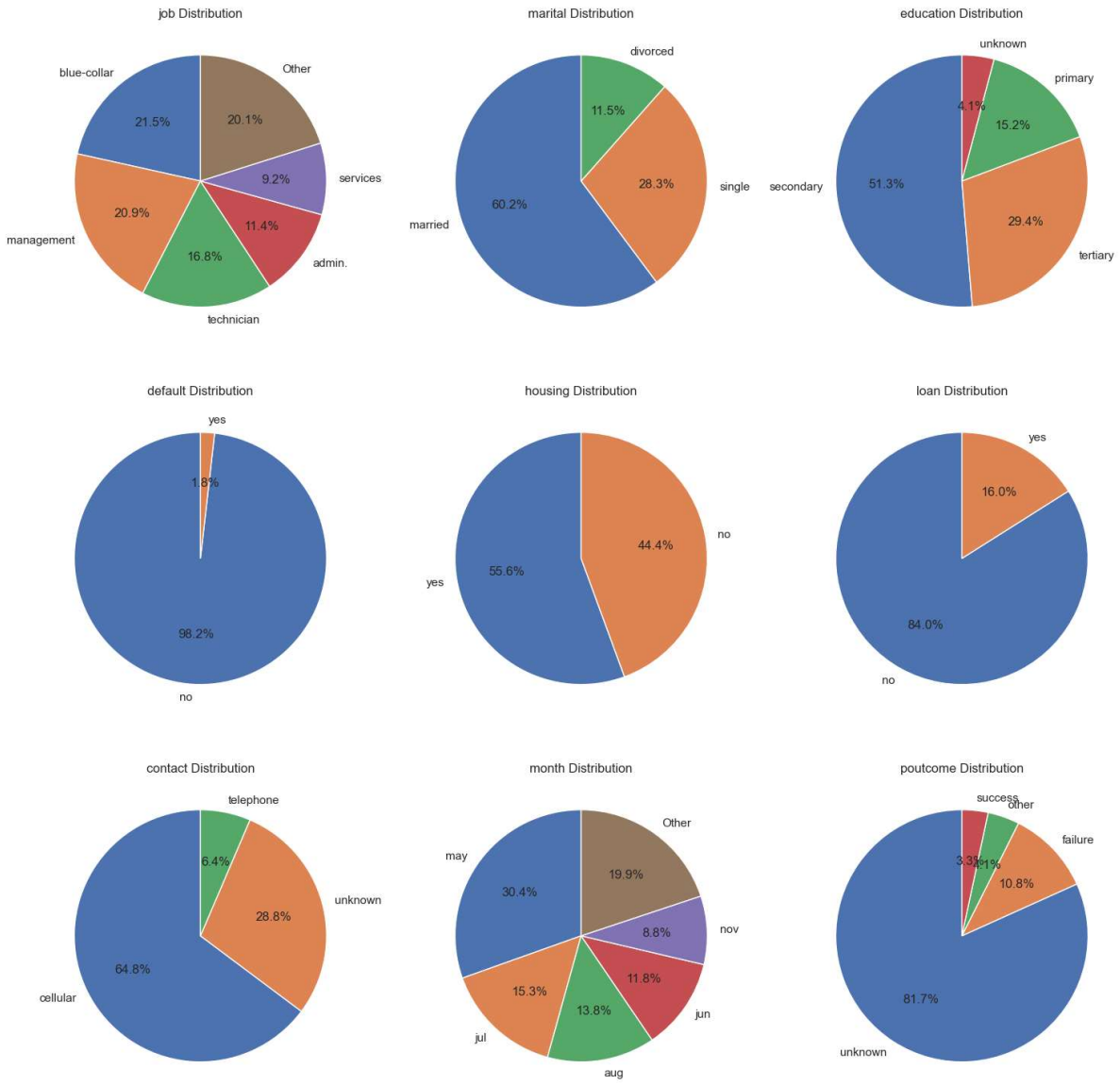
In [10]:
```
# list of categorical variables to plot
cat_vars = ['job', 'marital', 'education', 'default',
            'housing', 'loan', 'contact', 'month', 'poutcome']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='y', data=df, 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 [11]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['job', 'marital', 'education', 'default',
            'housing', 'loan', 'contact', 'month', 'poutcome']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axs = axs.flatten()

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

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```


y
no
yes
Density
job
management
technician
entrepreneur
blue-collar
unknown
retired
admin.
services
self-employed
unemployed
housemaid
student
marital
married
single
divorced
education
tertiary
secondary
unknown
primary
default
no
yes
housing
yes
no
loan
no
yes
contact
unknown
cellular
telephone
month
may
jun
jul
aug
oct
nov
dec
jan
feb
mar
apr
sep
poutcome
unknown
failure
other
success

In [12]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

# Filter categorical columns with 'object' data type
cat_cols = [col for col in df.columns if col != 'y' and df[col].dtype == 'obje

# Create a figure with subplots
num_cols = len(cat_cols)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))

# Flatten the axs array for easier indexing
axs = axs.flatten()

# Create a pie chart for each categorical column
for i, col in enumerate(cat_cols):
    if i < len(axs):  # Ensure we don't exceed the number of subplots
        # Count the number of occurrences for each category
        cat_counts = df[col].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), in
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', sta
        axs[i].set_title(f'{col} Distribution')

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```



Data Preprocessing Part 2

In [13]:
```
# Check the amounnt of missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[13]:
```
age        0.019907
balance    0.006636
dtype: float64
```

In [14]:
```
# Drop null value because the amount of null value is very small
df.dropna(inplace=True)
```

In [15]:
```
# Check the amounnt of missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[15]:
```
Series([], dtype: float64)
```

In [16]:
```
df.shape
```

Out[16]:
```
(45200, 17)
```

Label Encoding for Object Datatypes

In [17]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

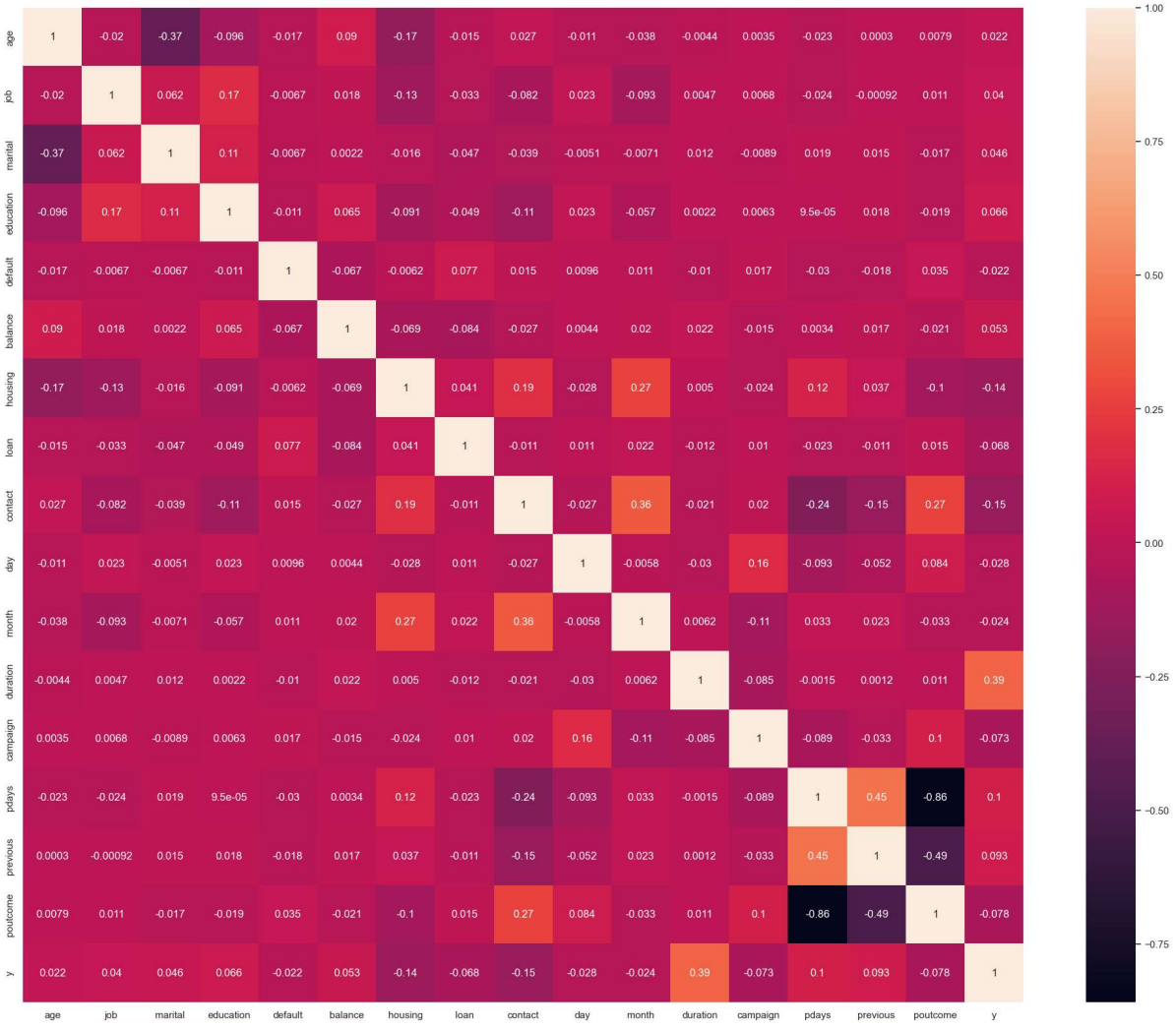
```
job: ['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown' 'admi
n.'
 'services' 'retired' 'self-employed' 'unemployed' 'housemaid' 'student']
marital: ['married' 'single' 'divorced']
education: ['tertiary' 'secondary' 'unknown' 'primary']
default: ['no' 'yes']
housing: ['yes' 'no']
loan: ['no' 'yes']
contact: ['unknown' 'cellular' 'telephone']
month: ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'se
p']
poutcome: ['unknown' 'failure' 'other' 'success']
y: ['no' 'yes']
```

In [18]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
job: [ 4  9  2  1 11  0  7  5  6 10  3  8]
marital: [1 2 0]
education: [2 1 3 0]
default: [0 1]
housing: [1 0]
loan: [0 1]
contact: [2 0 1]
month: [ 8  6  5  1 10  9  2  4  3  7  0 11]
poutcome: [3 0 1 2]
y: [0 1]
```

In [19]:
```
# Correlation Heatmap
plt.figure(figsize=(25, 20))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[19]: <AxesSubplot:>



Train Test Split

In [20]:
```
X = df.drop('y', axis=1)
y = df['y']
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_
```

Remove Outlier from Train Data using Z-Score

In [21]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['age', 'balance', 'duration',
                    'campaign', 'pdays', 'previous']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree Classifier

In [22]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier(class_weight='balanced')
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split': 2, 'random_stat
e': 0}
```

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=7, min_samples_leaf=2
dtree.fit(X_train, y_train)
```

Out[23]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=7, min_samples_leaf
=2,
                       random_state=0)
```

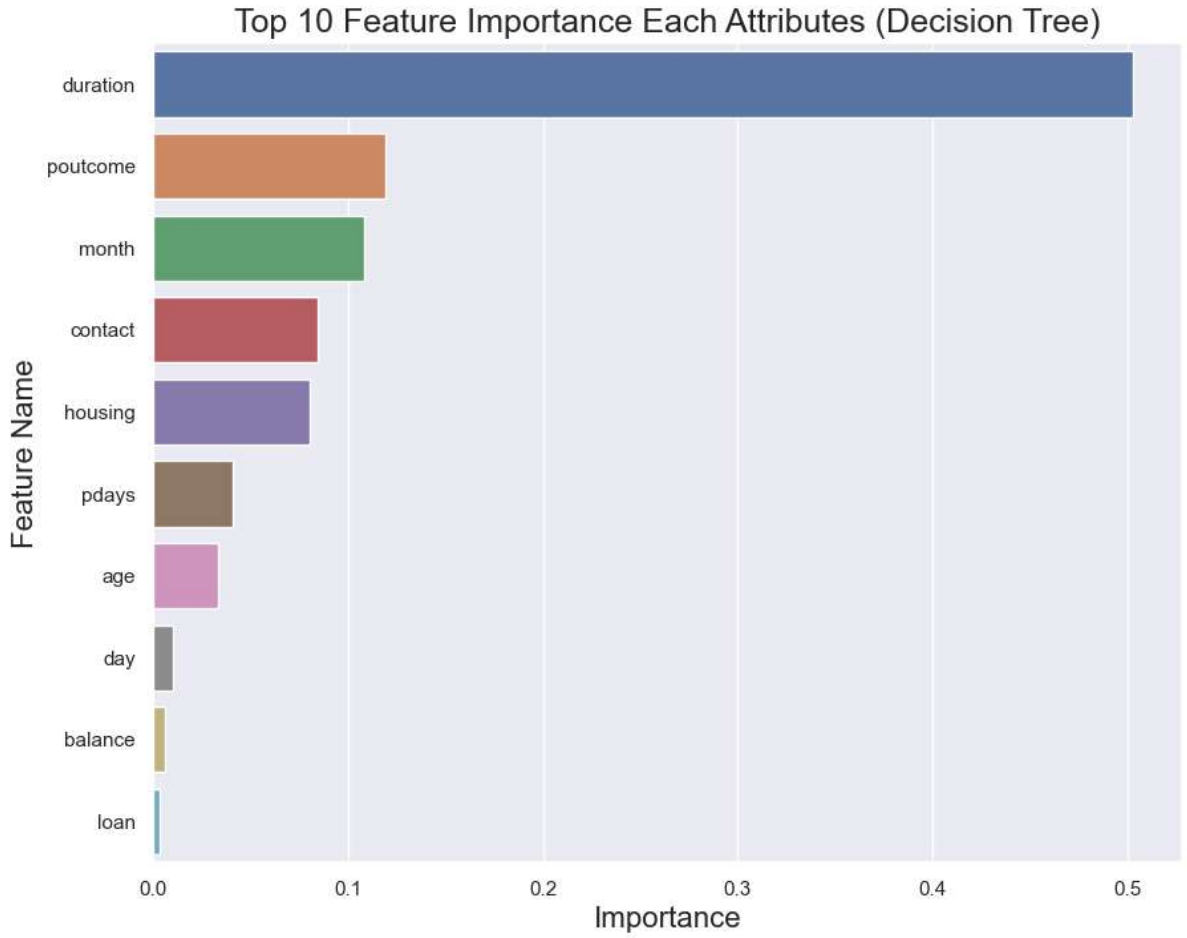
In [24]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 83.66 %
```

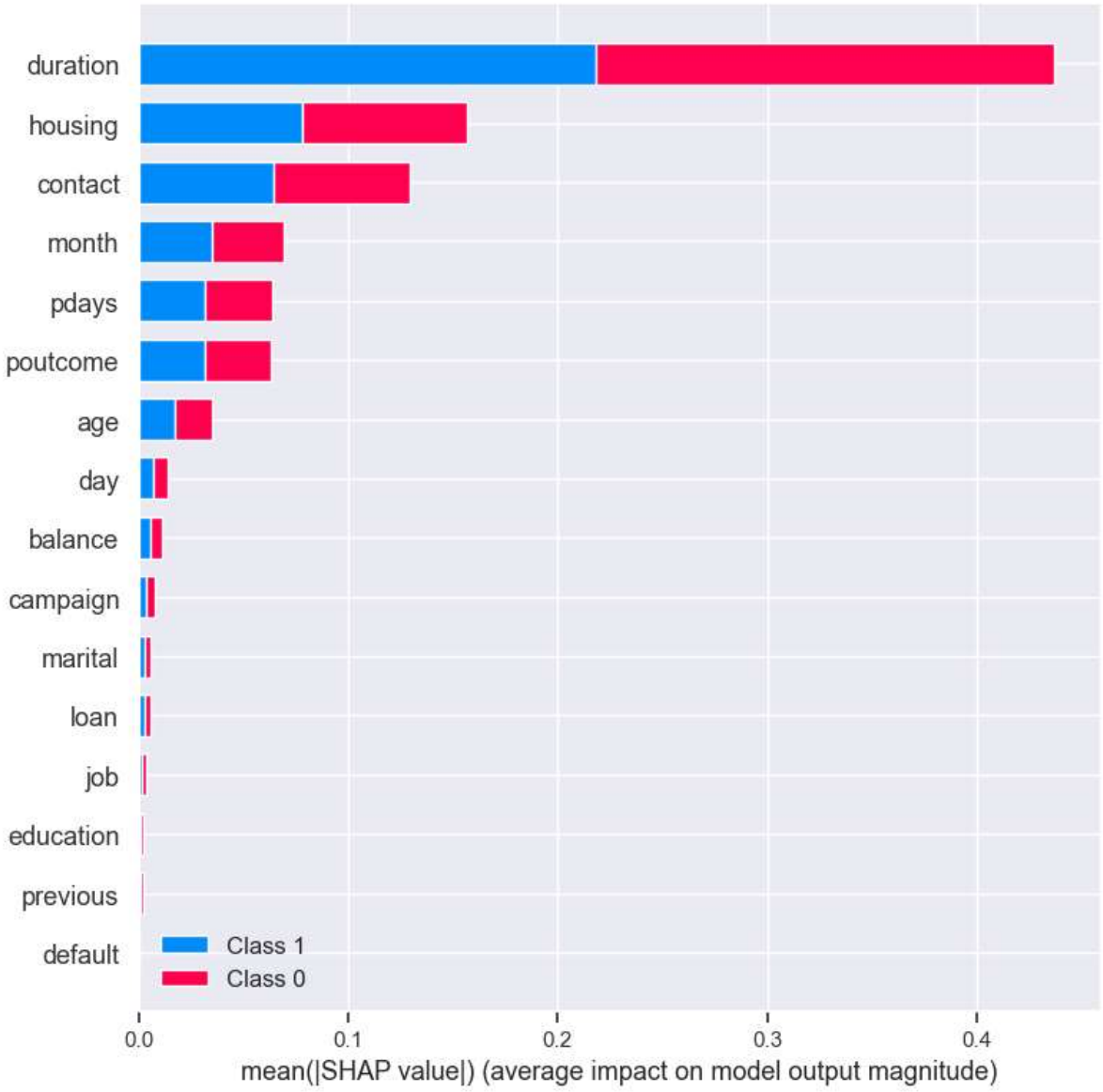
In [25]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8366150442477877
Precision Score :  0.8366150442477877
Recall Score :  0.8366150442477877
Jaccard Score :  0.7191214224588761
Log Loss :  5.643226928370951
```

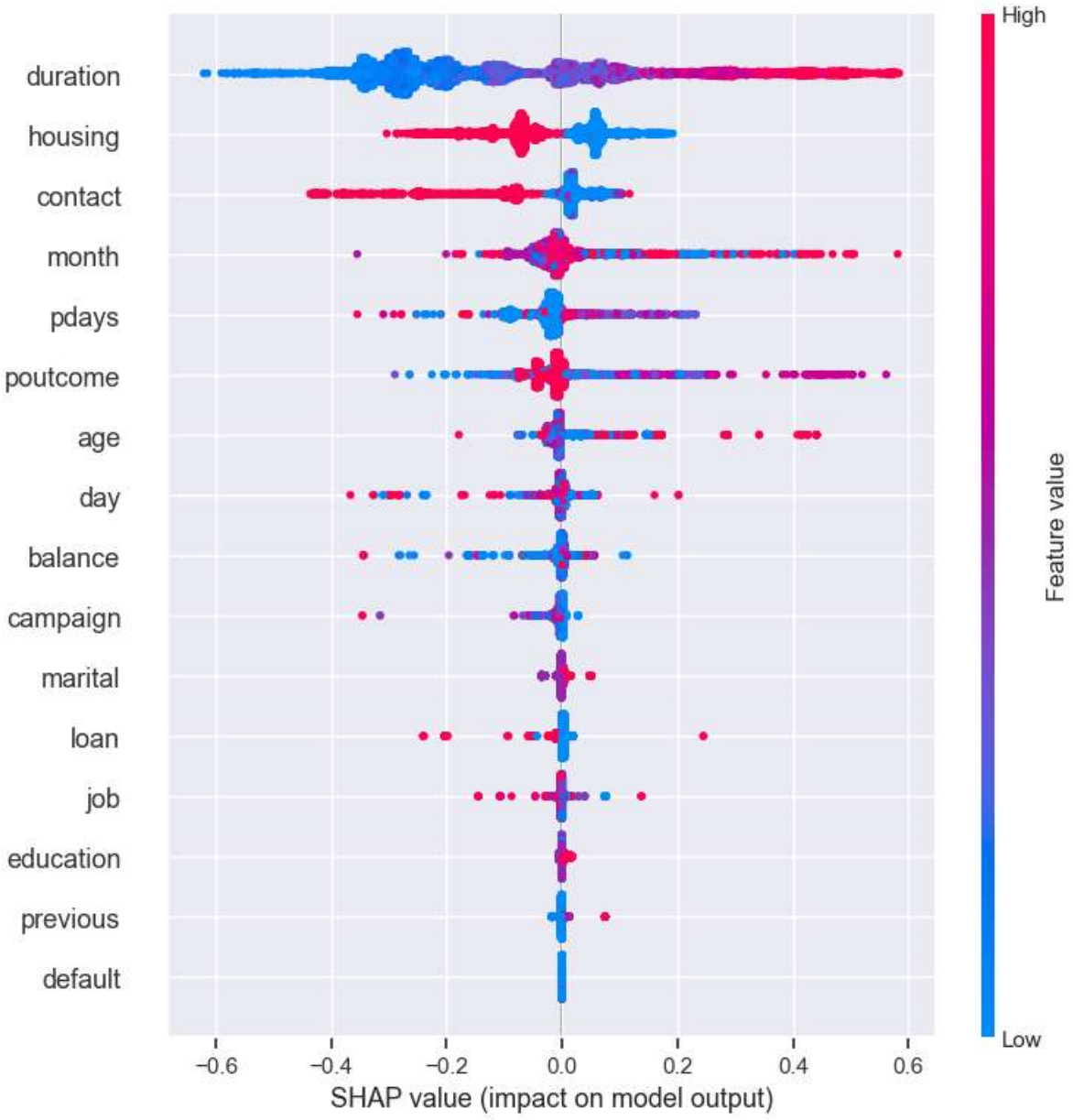
In [26]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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



In [27]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

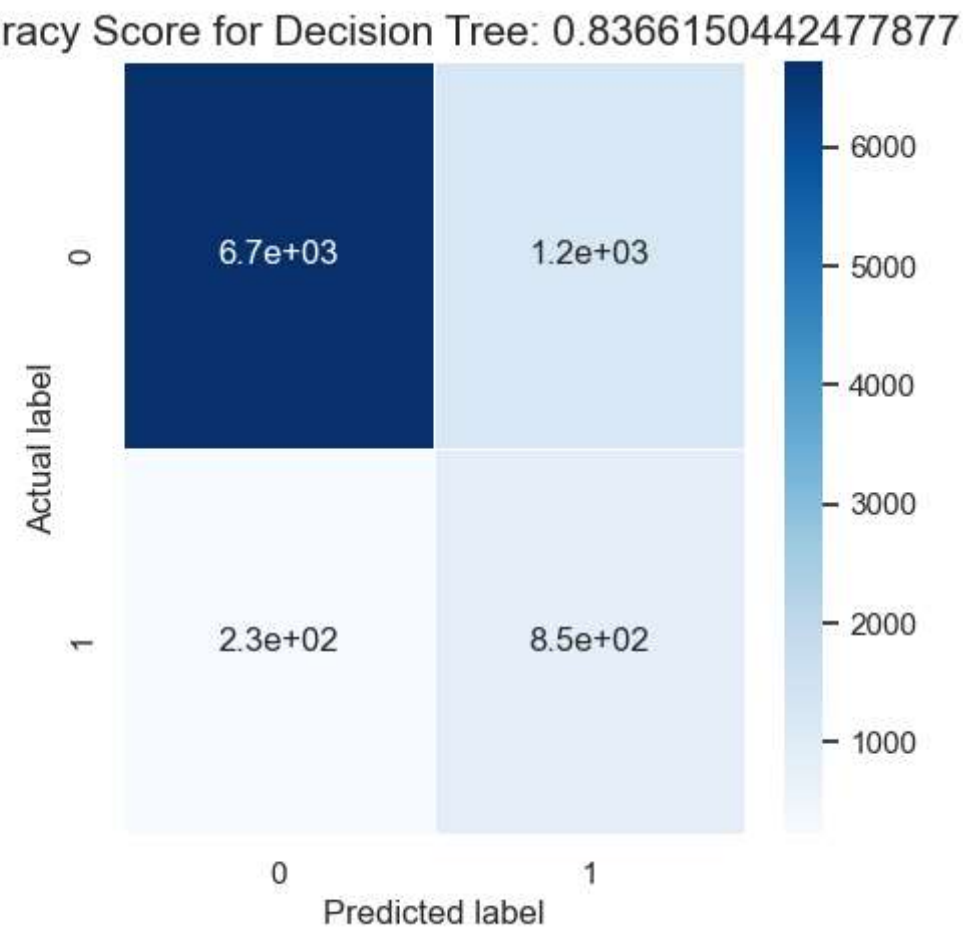


In [28]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns
```



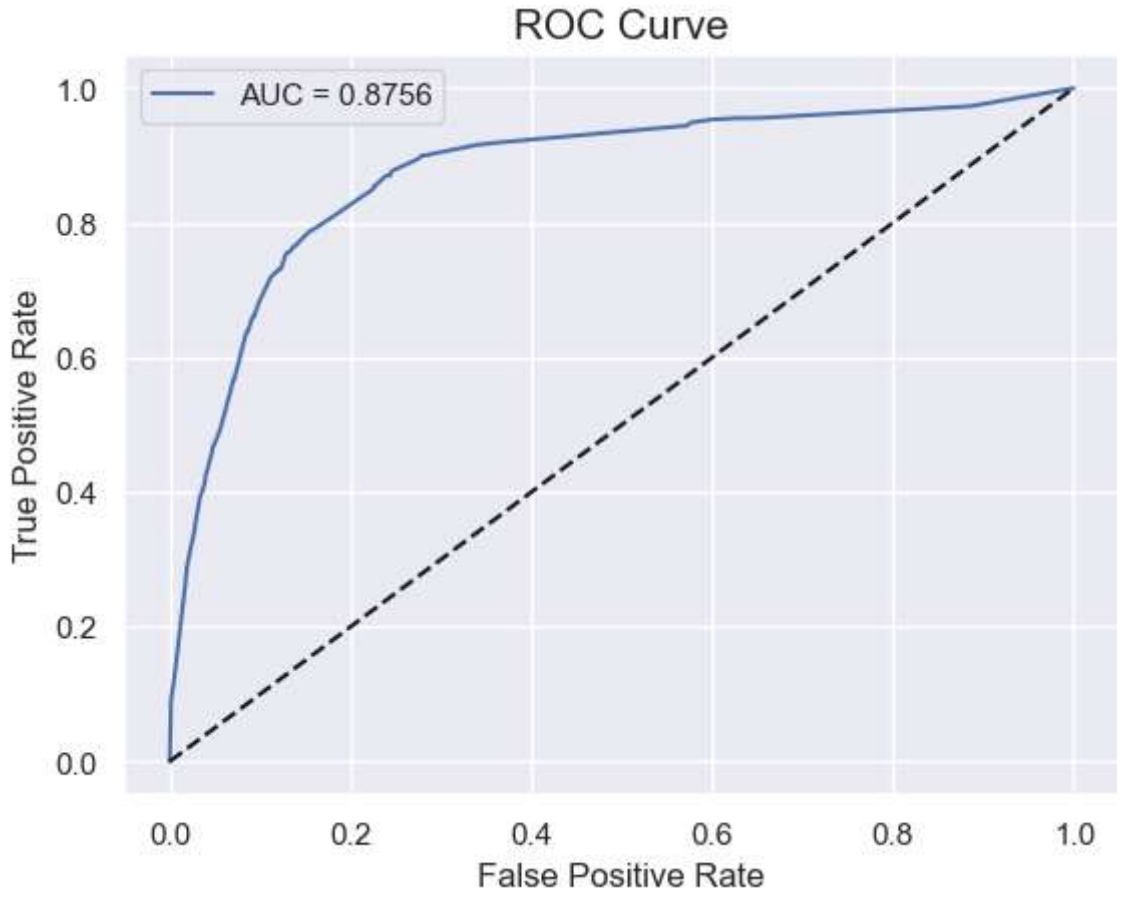
In [29]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(
plt.title(all_sample_title, size = 15)
```

Out[29]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8366150442477877')



In [30]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_ac
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted[
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pr

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[30]: <matplotlib.legend.Legend at 0x19dce969fd0>



Random Forest Classifier

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(class_weight='balanced')
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200, 'random_stat
e': 42}
```

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_depth=None, max_features='sq
rfc.fit(X_train, y_train)
```

Out[32]:
```
RandomForestClassifier(class_weight='balanced', max_features='sqrt',
                       n_estimators=200, random_state=42)
```

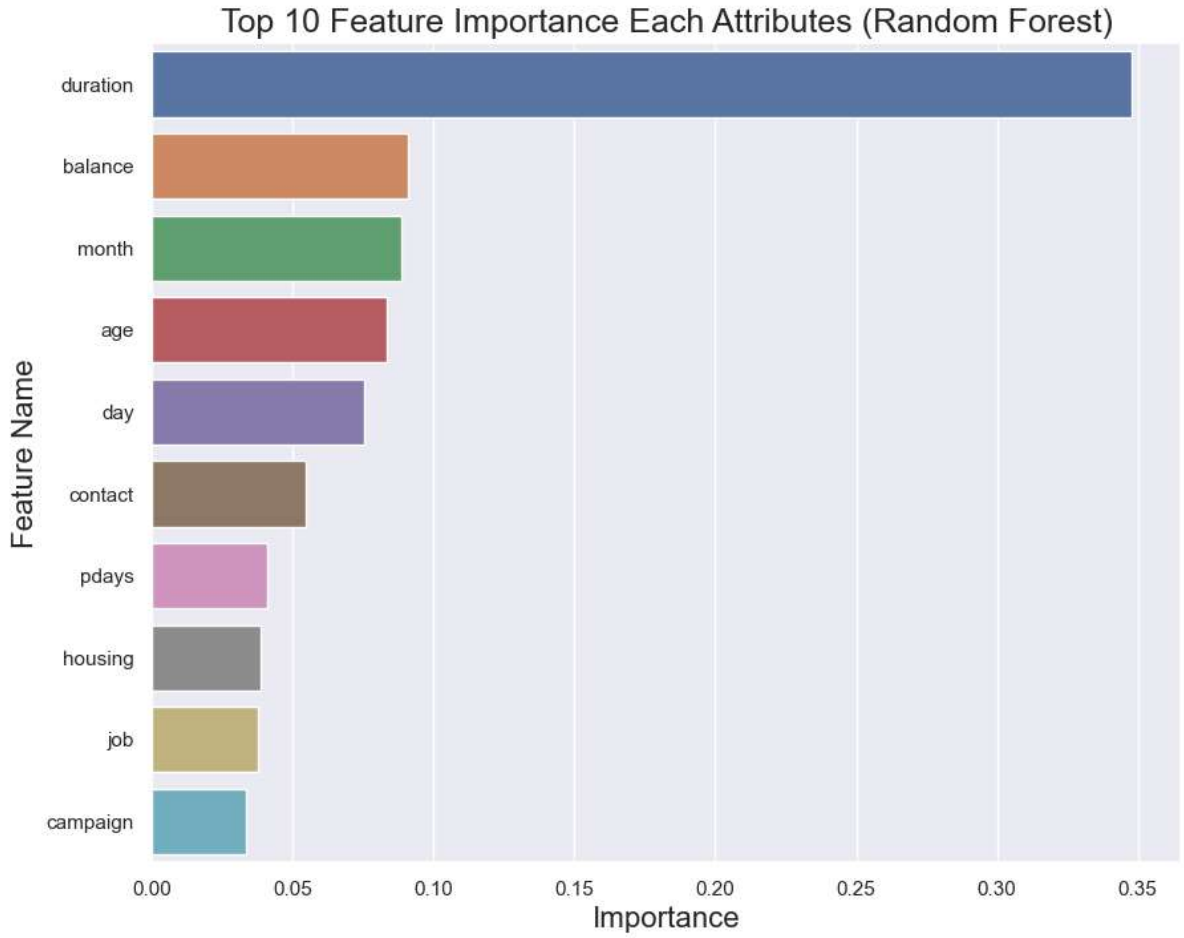
In [33]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 89.87 %
```

In [34]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
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)))
```

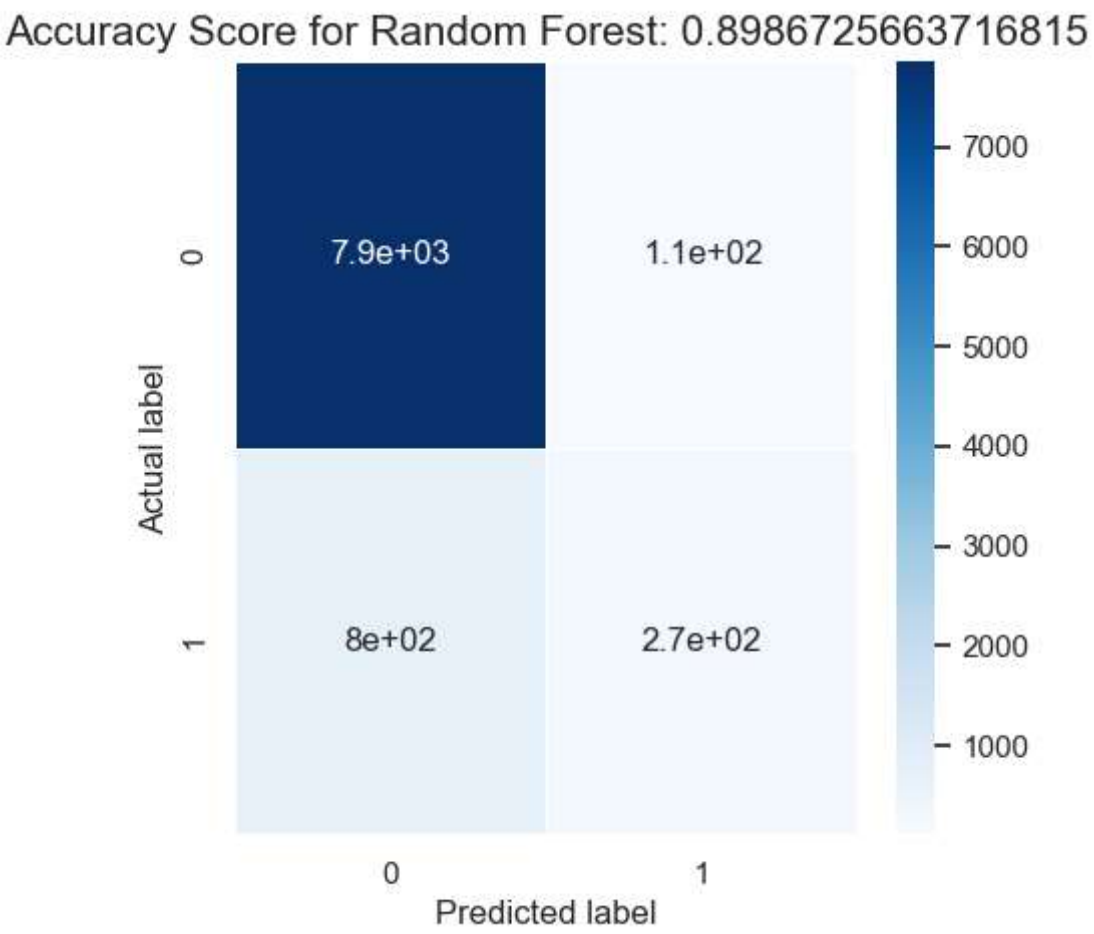
```
F-1 Score :  0.8986725663716815
Precision Score :  0.8986725663716815
Recall Score :  0.8986725663716815
Jaccard Score :  0.8159903575733226
Log Loss :  3.4997354792754836
```

In [35]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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



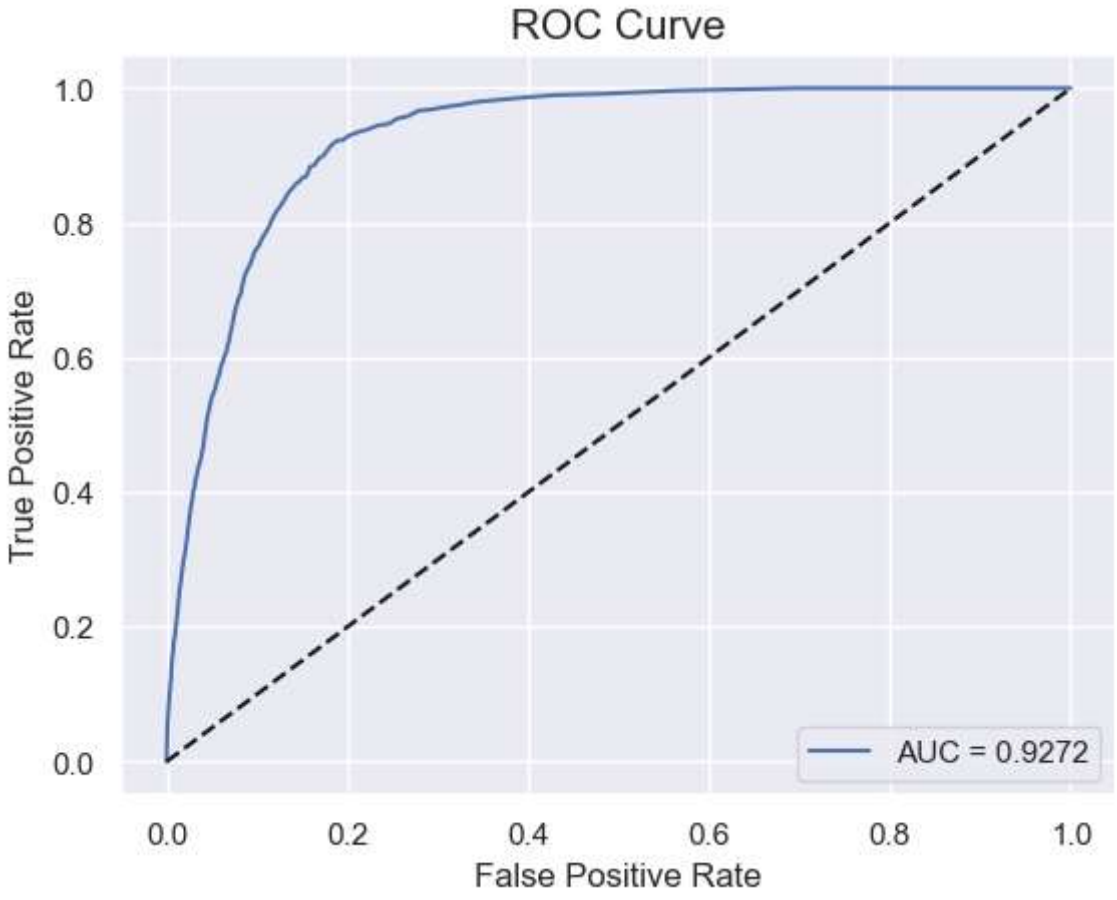
In [36]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_
plt.title(all_sample_title, size = 15)
```

Out[36]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.8986725663716815')
```



In [37]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_ac
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted[
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pr

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[37]: <matplotlib.legend.Legend at 0x19dcd19b6a0>



In [ ]: