In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

In [2]:
```
df = pd.read_csv('startup data.csv')
df.head()
```

Out[2]:

nding_year	age_first_milestone_year	age_last_milestone_year	relationships	funding_rounds	funding_total_usd	milestones	state_code.1	is
3.0027	4.6685	6.7041	3	3	375000	3	CA	
9.9973	7.0055	7.0055	9	4	40100000	1	CA	
1.0329	1.4575	2.2055	5	1	2600000	2	CA	
5.3151	6.0027	6.0027	5	3	40000000	1	CA	
1.6685	0.0384	0.0384	2	2	1300000	1	CA	

Data Preprocessing Part 1

In [3]:
```
df.drop(columns = ['Unnamed: 0', 'id', 'Unnamed: 6', 'name'], inplace=True)
df.shape
```

Out[3]: (923, 45)

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

Out[4]:
```
state_code           35
zip_code            382
city                221
founded_at          217
closed_at           202
first_funding_at    585
last_funding_at     680
state_code.1         35
category_code        35
object_id           922
status                2
dtype: int64
```

In [5]:
```
# Drop column with unique value higher than 100 except for date time column
df.drop(columns = ['zip_code', 'city', 'object_id'], inplace = True)
df.shape
```

Out[5]: (923, 42)

In [7]:
```
# Extract only year from all of date time column and change the datatype into integer except null value
df['founded_at'] = df['founded_at'].apply(lambda x: int(x[-4:].lstrip('0')) if isinstance(x, str) else np.nan)
df['closed_at'] = df['closed_at'].apply(lambda x: int(x[-4:].lstrip('0')) if isinstance(x, str) else np.nan)
df['first_funding_at'] = df['first_funding_at'].apply(lambda x: int(x[-4:].lstrip('0')) if isinstance(x, str) 
df['last_funding_at'] = df['last_funding_at'].apply(lambda x: int(x[-4:].lstrip('0')) if isinstance(x, str) el
```

In [8]:
```
df.head()
```

Out[8]:

	state_code	latitude	longitude	labels	founded_at	closed_at	first_funding_at	last_funding_at	age_first_funding_year	age_last_fu
0	CA	42.358880	-71.056820	1	NaN	NaN	2009	2010	2.2493	
1	CA	37.238916	-121.973718	1	NaN	NaN	2005	2009	5.1260	
2	CA	32.901049	-117.192656	1	NaN	NaN	2010	2010	1.0329	
3	CA	37.320309	-122.050040	1	NaN	NaN	2005	2007	3.1315	
4	CA	37.779281	-122.419236	0	NaN	2012.0	2010	2012	0.0000	

In [9]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[9]:
```
state_code       35
state_code.1     35
category_code    35
status            2
dtype: int64
```

Segment the Category Code into Smaller Unique Value

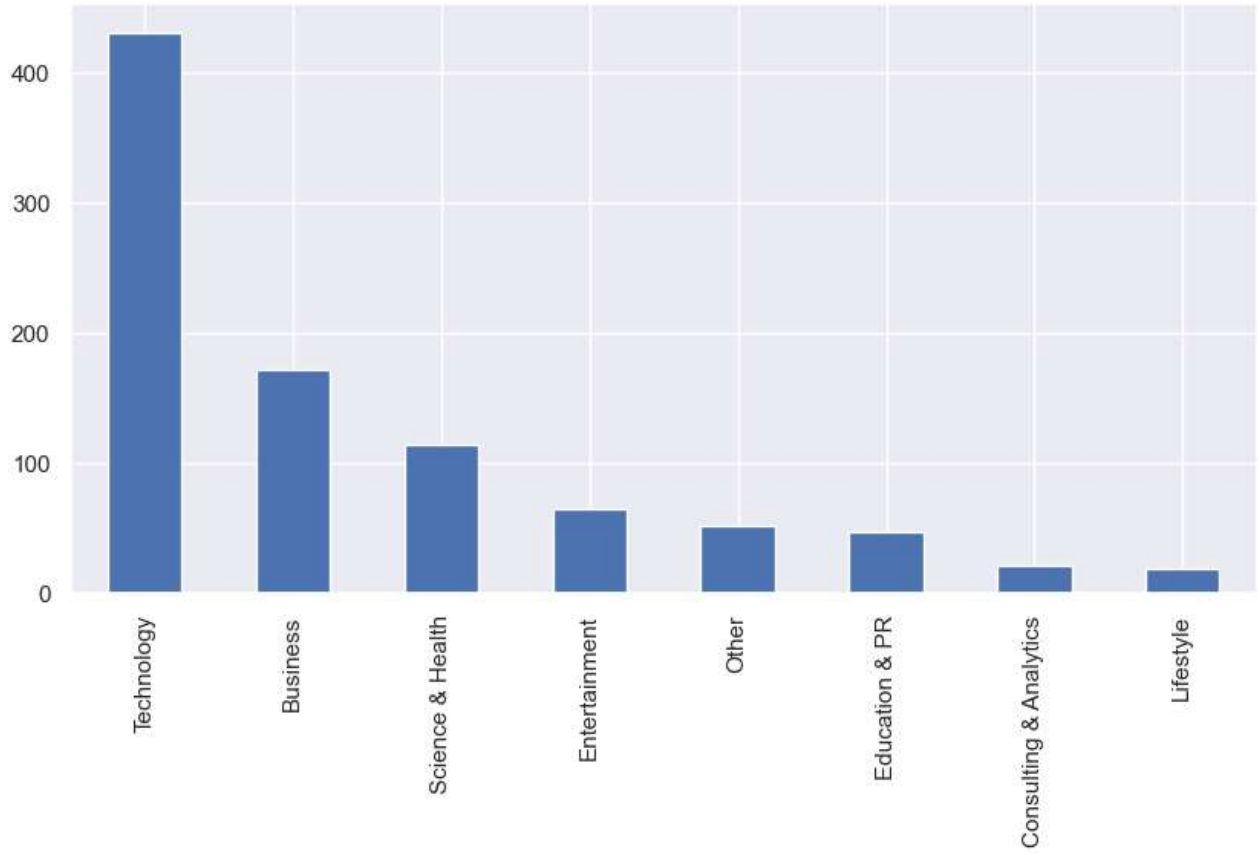
In [10]:
```
df.category_code.unique()
```

Out[10]:
```
array(['music', 'enterprise', 'web', 'software', 'games_video',
       'network_hosting', 'finance', 'mobile', 'education',
       'public_relations', 'security', 'other', 'photo_video', 'hardware',
       'ecommerce', 'advertising', 'travel', 'fashion', 'analytics',
       'consulting', 'biotech', 'cleantech', 'search', 'semiconductor',
       'social', 'medical', 'automotive', 'messaging', 'manufacturing',
       'hospitality', 'news', 'transportation', 'sports', 'real_estate',
       'health'], dtype=object)
```

In [11]:
```
def segment_category(category):
    if category in ['music', 'games_video', 'photo_video', 'entertainment']:
        return 'Entertainment'
    elif category in ['enterprise', 'web', 'software', 'network_hosting', 'hardware', 'tech']:
        return 'Technology'
    elif category in ['finance', 'mobile', 'ecommerce', 'advertising', 'business']:
        return 'Business'
    elif category in ['education', 'public_relations', 'security']:
        return 'Education & PR'
    elif category in ['travel', 'fashion', 'hospitality', 'transportation']:
        return 'Lifestyle'
    elif category in ['analytics', 'consulting']:
        return 'Consulting & Analytics'
    elif category in ['biotech', 'cleantech', 'search', 'semiconductor', 'medical', 'health']:
        return 'Science & Health'
    else:
        return 'Other'


# Apply the segmentation function to each category code
df['category_segment'] = df['category_code'].apply(segment_category)
```

In [13]:
```
plt.figure(figsize=(10,5))
df['category_segment'].value_counts().plot(kind='bar')
```

Out[13]: <AxesSubplot:>



In [14]:
```
df.head()
```

Out[14]:

nds	funding_total_usd	milestones	state_code.1	is_CA	is_NY	is_MA	is_TX	is_otherstate	category_code	is_software	is_web	is_mobile
3	375000	3	CA	1	0	0	0	0	music	0	0	0
4	40100000	1	CA	1	0	0	0	0	enterprise	0	0	0
1	2600000	2	CA	1	0	0	0	0	web	0	1	0
3	40000000	1	CA	1	0	0	0	0	software	1	0	0
2	1300000	1	CA	1	0	0	0	0	games_video	0	0	0

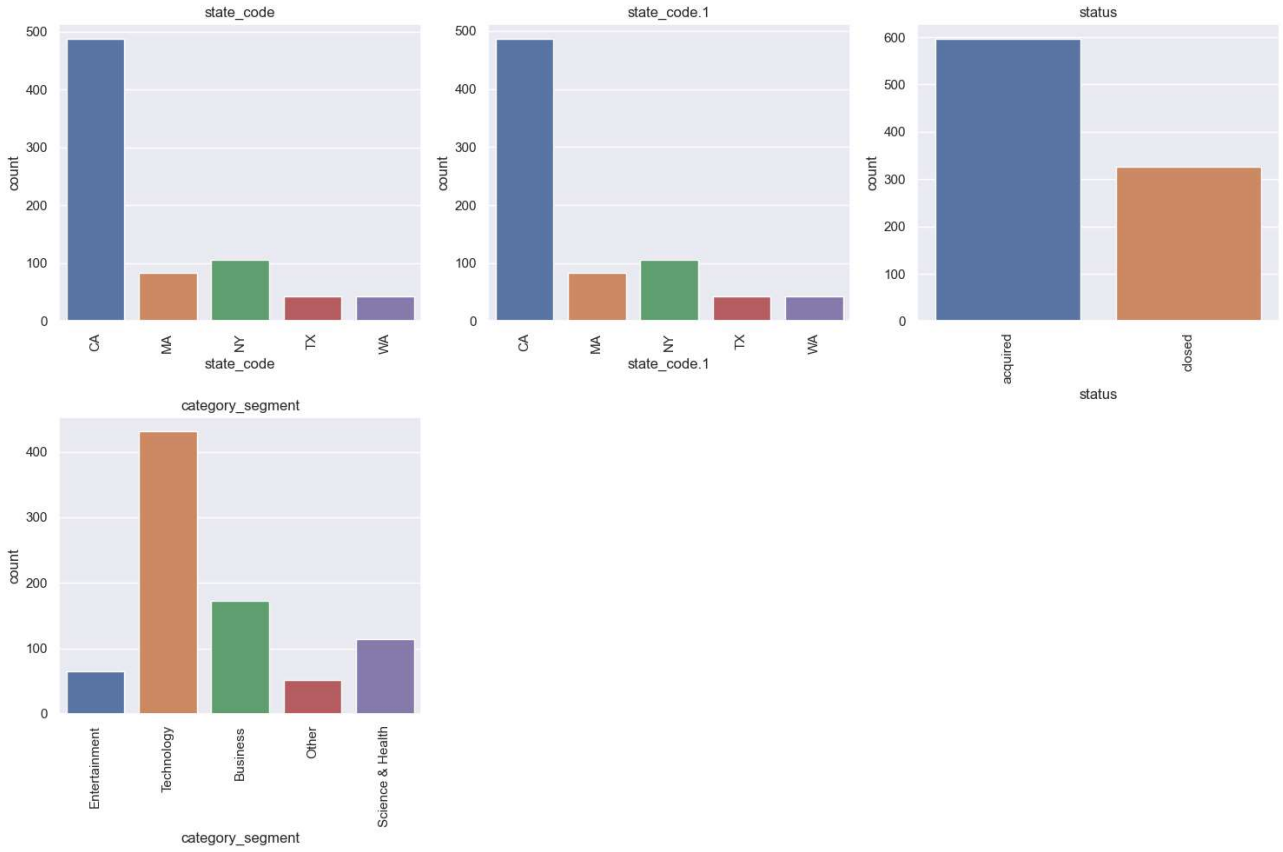
In [15]:
```
# Drop category_code column
df.drop(columns = 'category_code', inplace=True)
df.head()
```

Out[15]:

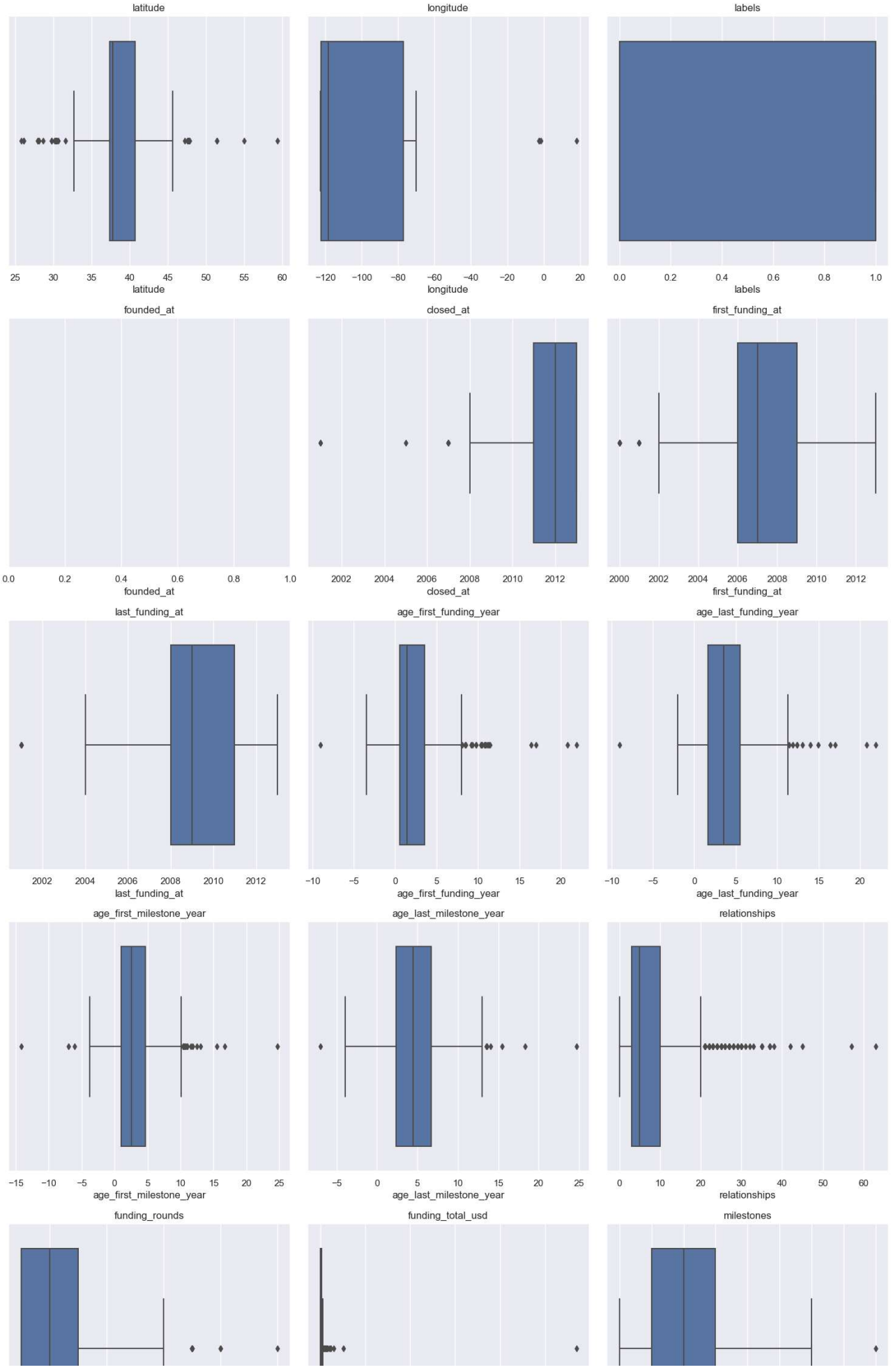
_CA	is_NY	is_MA	is_TX	is_otherstate	is_software	is_web	is_mobile	is_enterprise	is_advertising	is_gamesvideo	is_ecommerce	is_bi
1	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	1	0	0	0	
1	0	0	0	0	0	1	0	0	0	0	0	
1	0	0	0	0	1	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	1	0	

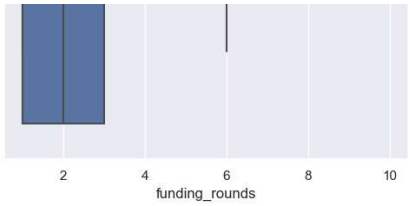
Exploratory Data Analysis

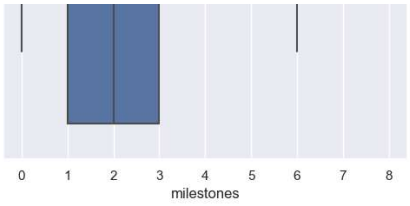
In [17]:
```
# Get the names of all columns with data type 'object' (categorical columns)
cat_vars = df.select_dtypes(include='object').columns.tolist()

# 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 Seaborn
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(5).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()
```



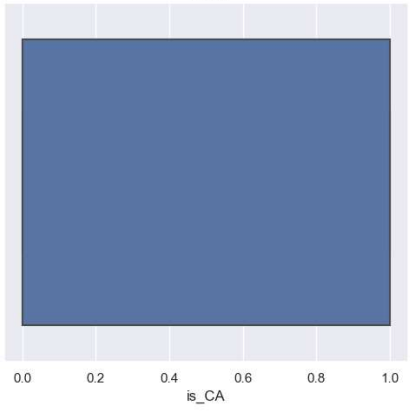
In [19]:
```
# Get the names of all columns with data type 'int' or 'float'
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# 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()
```


latitude
25 30 35 40 45 50 55 60
latitude
longitude
−120 −100 −80 −60 −40 −20 0 20
longitude
labels
0.0 0.2 0.4 0.6 0.8 1.0
labels
founded_at
0.0 0.2 0.4 0.6 0.8 1.0
founded_at
closed_at
2002 2004 2006 2008 2010 2012
closed_at
first_funding_at
2000 2002 2004 2006 2008 2010 2012
first_funding_at
last_funding_at
2002 2004 2006 2008 2010 2012
last_funding_at
age_first_funding_year
−10 −5 0 5 10 15 20
age_first_funding_year
age_last_funding_year
−10 −5 0 5 10 15 20
age_last_funding_year
age_first_milestone_year
−15 −10 −5 0 5 10 15 20 25
age_first_milestone_year
age_last_milestone_year
−5 0 5 10 15 20 25
age_last_milestone_year
relationships
0 10 20 30 40 50 60
relationships
funding_rounds
funding_total_usd
milestones

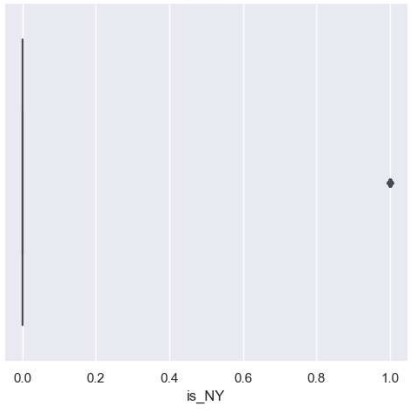




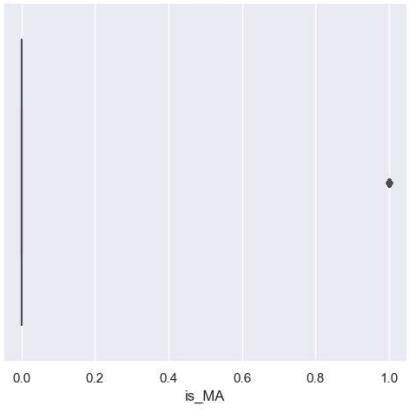
is_CA



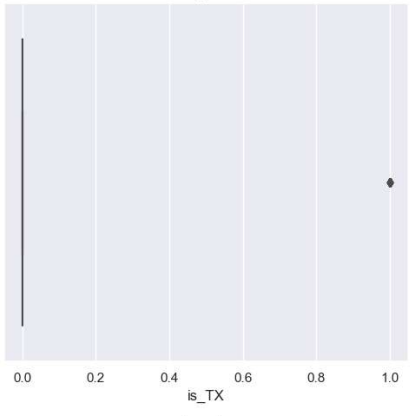
is_NY



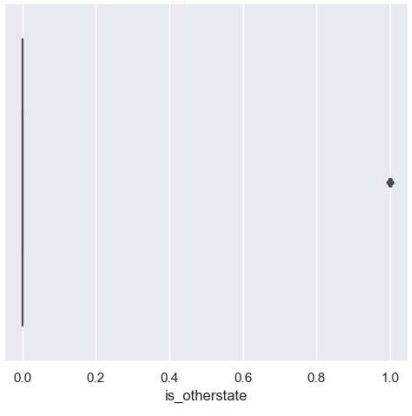
is_MA



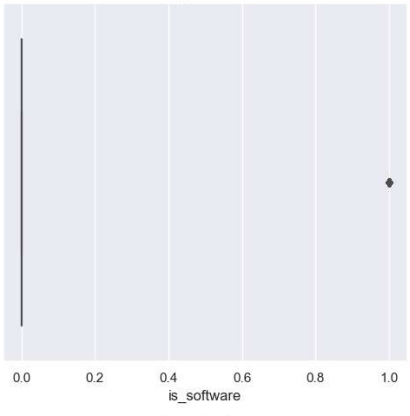
is_TX



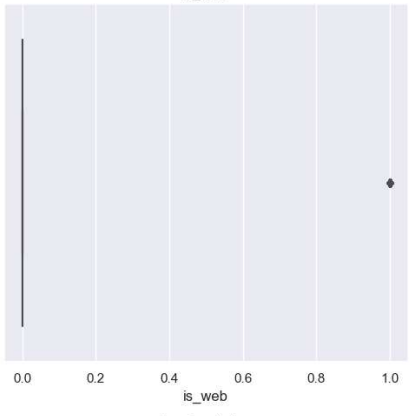
is_otherstate



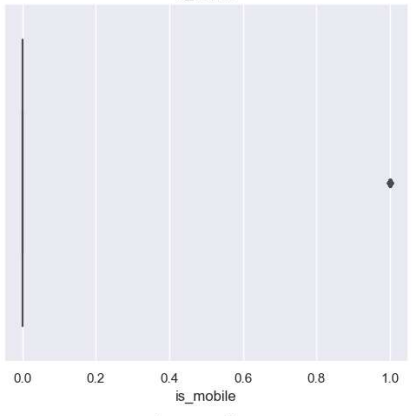
is_software



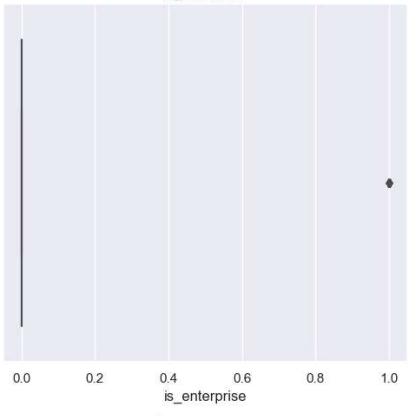
is_web



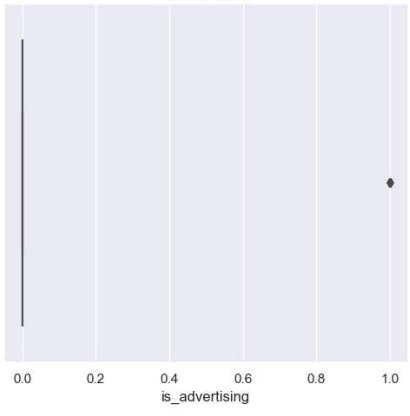
is_mobile



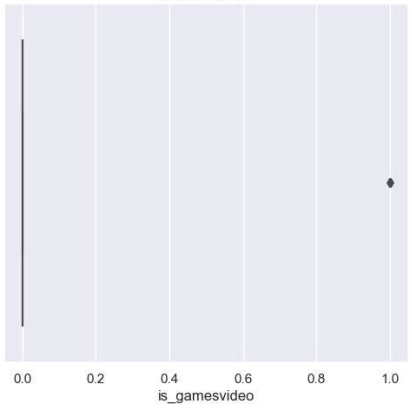
is_enterprise



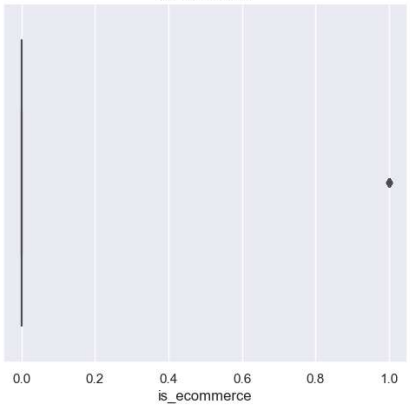
is_advertising



is_gamesvideo



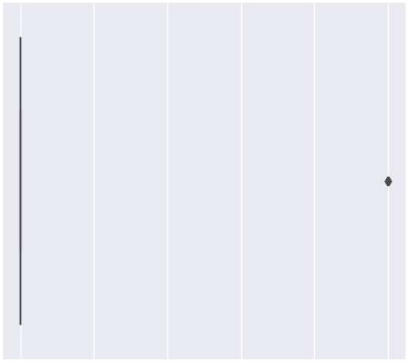
is_ecommerce

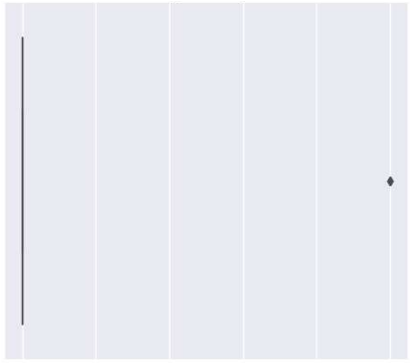


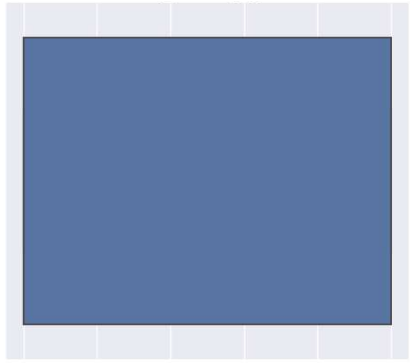
is_biotech

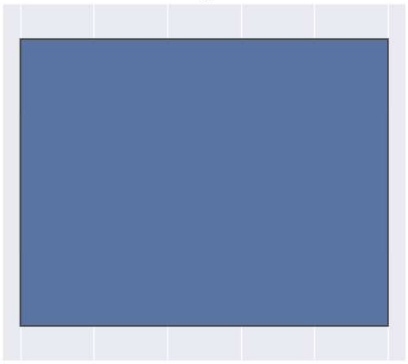
is_consulting

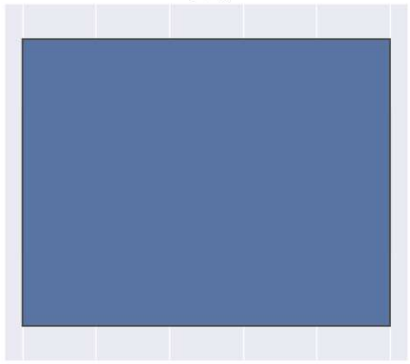
is_othercategory

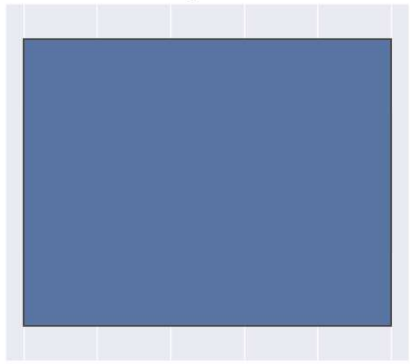

0.0 0.2 0.4 0.6 0.8 1.0
is_biotech

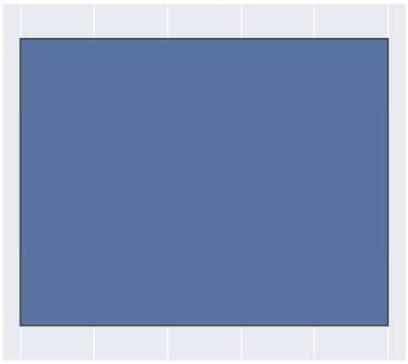

0.0 0.2 0.4 0.6 0.8 1.0
is_consulting

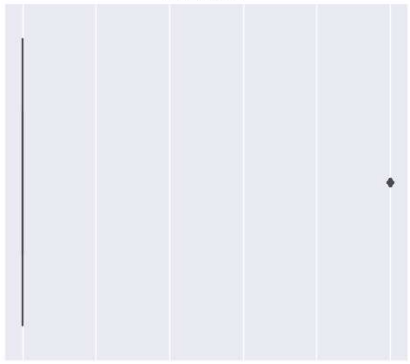

0.0 0.2 0.4 0.6 0.8 1.0
is_othercategory

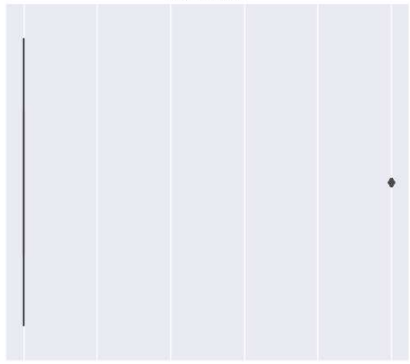

has_VC
0.0 0.2 0.4 0.6 0.8 1.0
has_VC

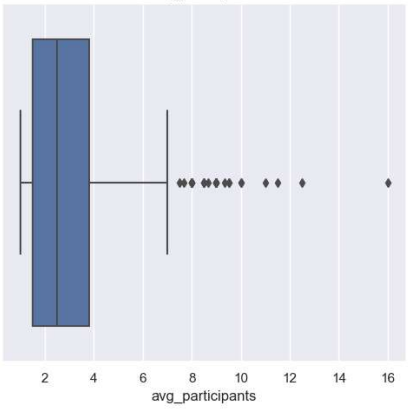

has_angel
0.0 0.2 0.4 0.6 0.8 1.0
has_angel

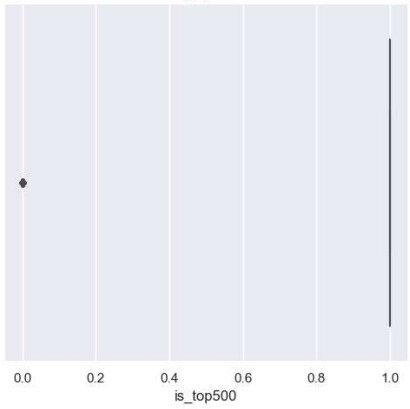

has_roundA
0.0 0.2 0.4 0.6 0.8 1.0
has_roundA


has_roundB
0.0 0.2 0.4 0.6 0.8 1.0
has_roundB

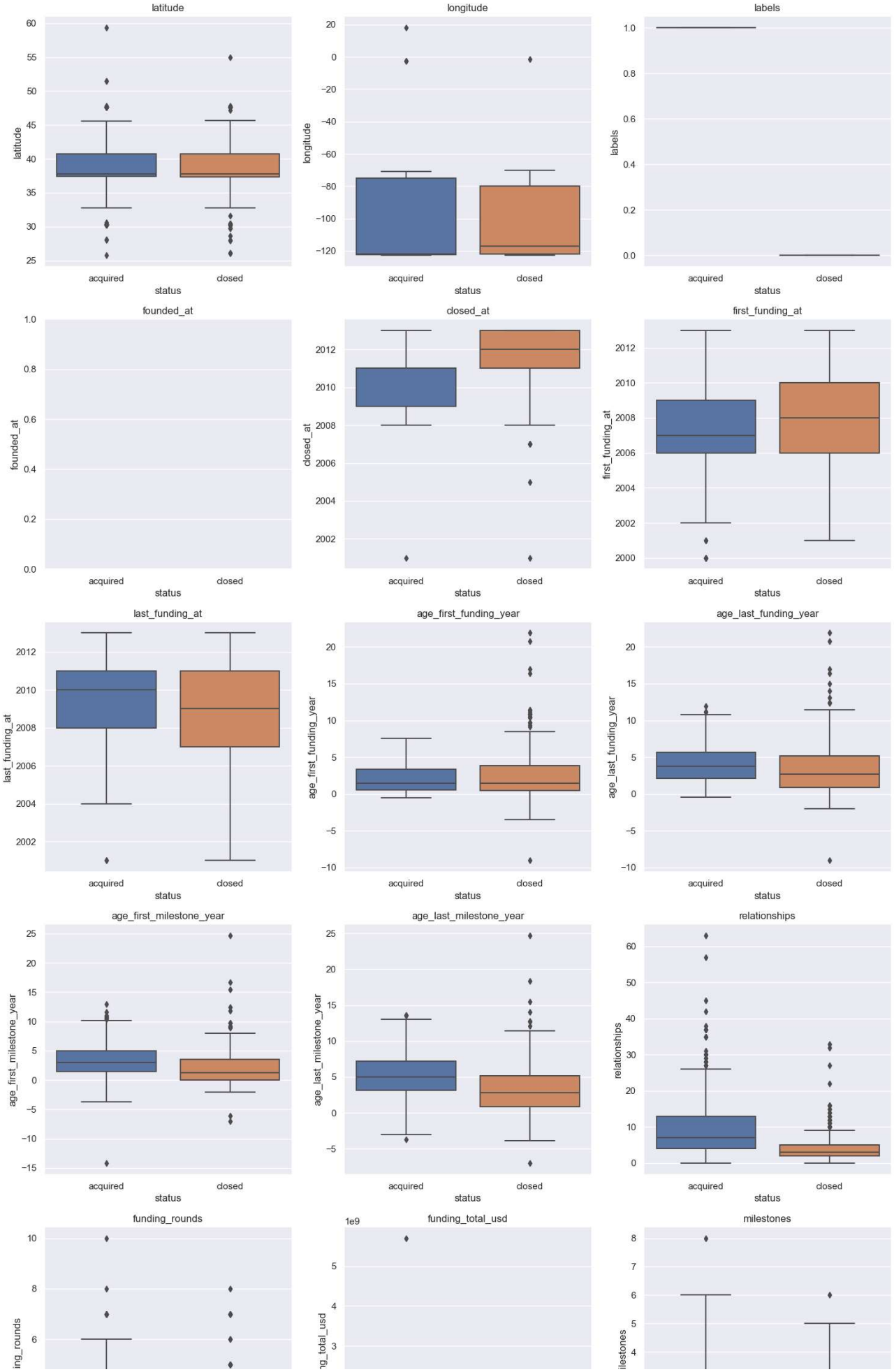

has_roundC
0.0 0.2 0.4 0.6 0.8 1.0
has_roundC

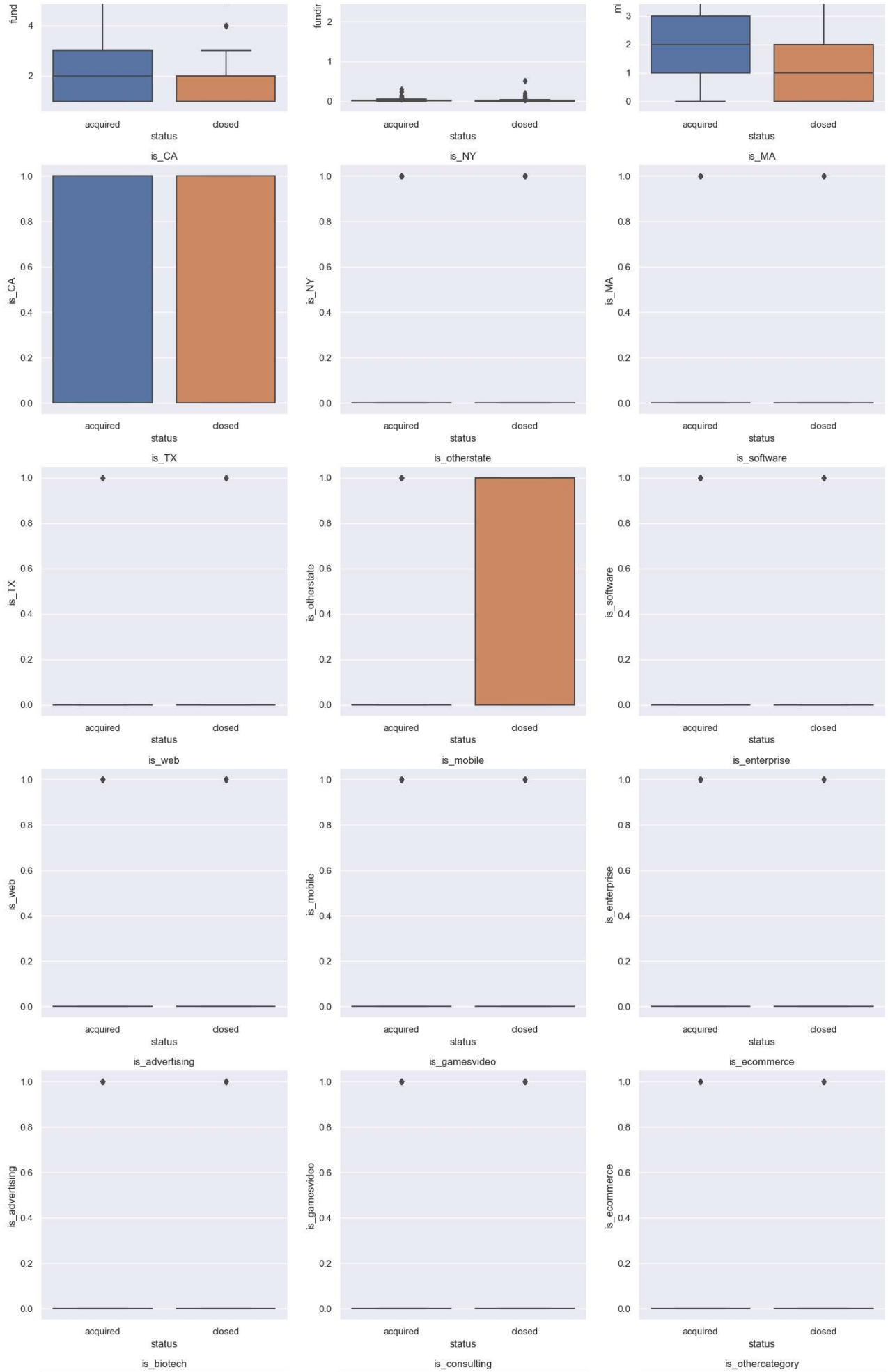

has_roundD
0.0 0.2 0.4 0.6 0.8 1.0
has_roundD

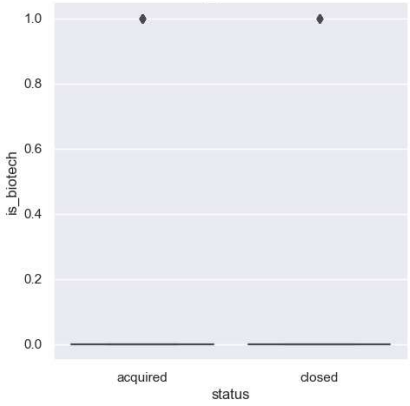

avg_participants
2 4 6 8 10 12 14 16
avg_participants

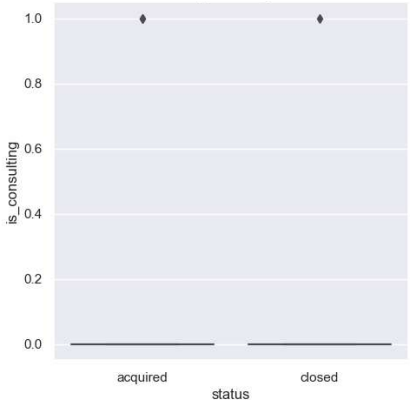

is_top500
0.0 0.2 0.4 0.6 0.8 1.0
is_top500

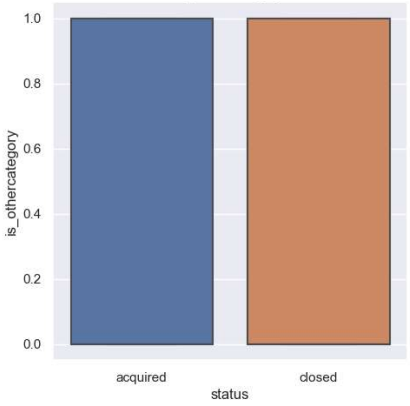
In [20]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# 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 subplots
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='attrition'
for i, var in enumerate(int_vars):
    sns.boxplot(y=var, x='status', 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()
```

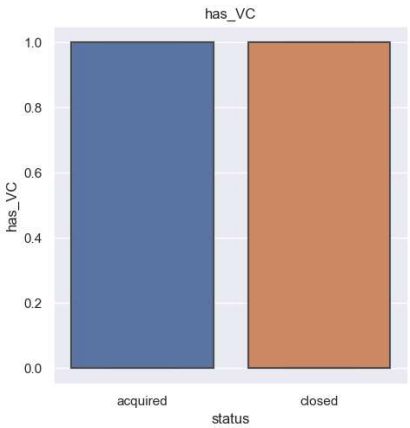

latitude
latitude
status
acquired
closed
longitude
longitude
labels
labels
founded_at
founded_at
closed_at
closed_at
first_funding_at
first_funding_at
last_funding_at
last_funding_at
age_first_funding_year
age_first_funding_year
age_last_funding_year
age_last_funding_year
age_first_milestone_year
age_first_milestone_year
age_last_milestone_year
age_last_milestone_year
relationships
relationships
funding_rounds
funding_total_usd
1e9
milestones

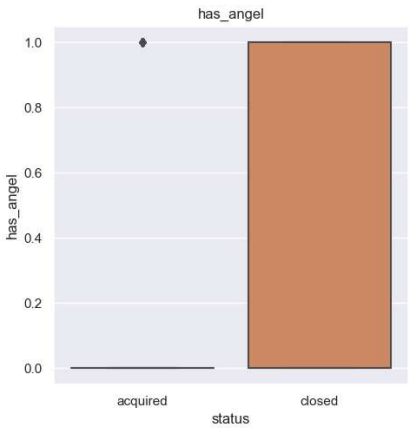

fund
fundi
m
status
acquired
closed
is_CA
is_NY
is_MA
is_TX
is_otherstate
is_software
is_web
is_mobile
is_enterprise
is_advertising
is_gamesvideo
is_ecommerce
is_biotech
is_consulting
is_othercategory

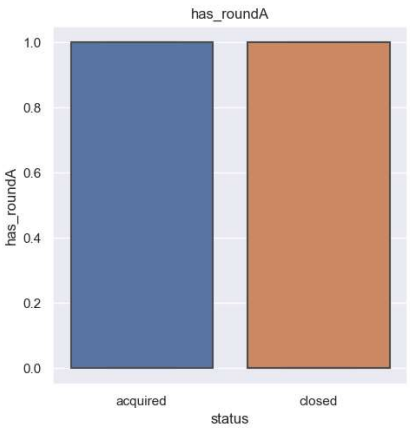

is_biotech
status
acquired
closed

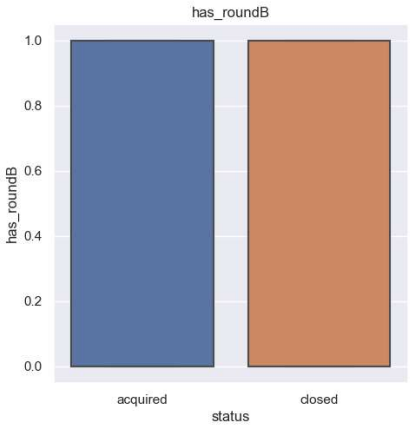

is_consulting
status
acquired
closed

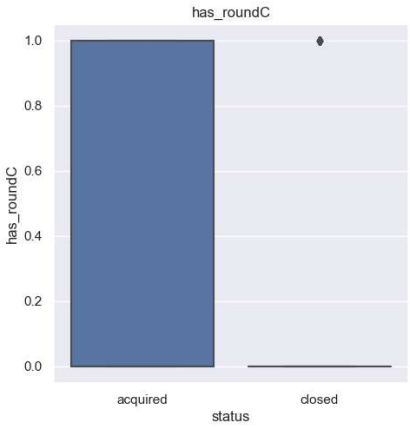

is_othercategory
status
acquired
closed

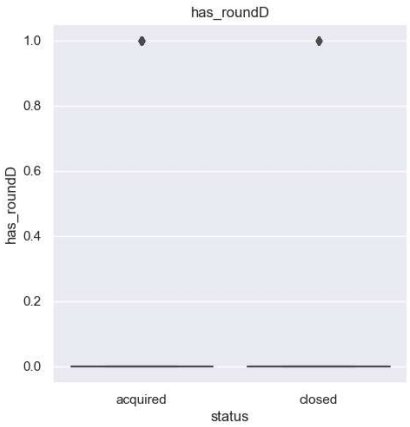

has_VC
has_VC
status
acquired
closed

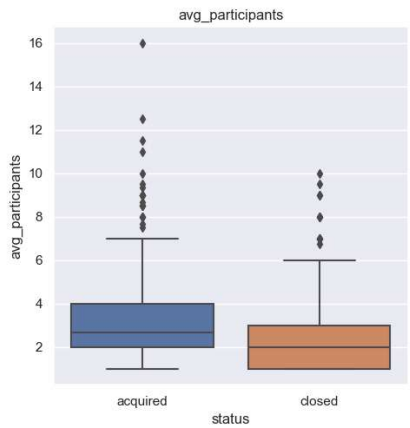

has_angel
has_angel
status
acquired
closed

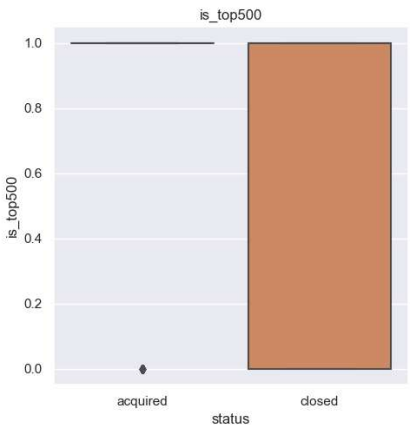

has_roundA
has_roundA
status
acquired
closed


has_roundB
has_roundB
status
acquired
closed

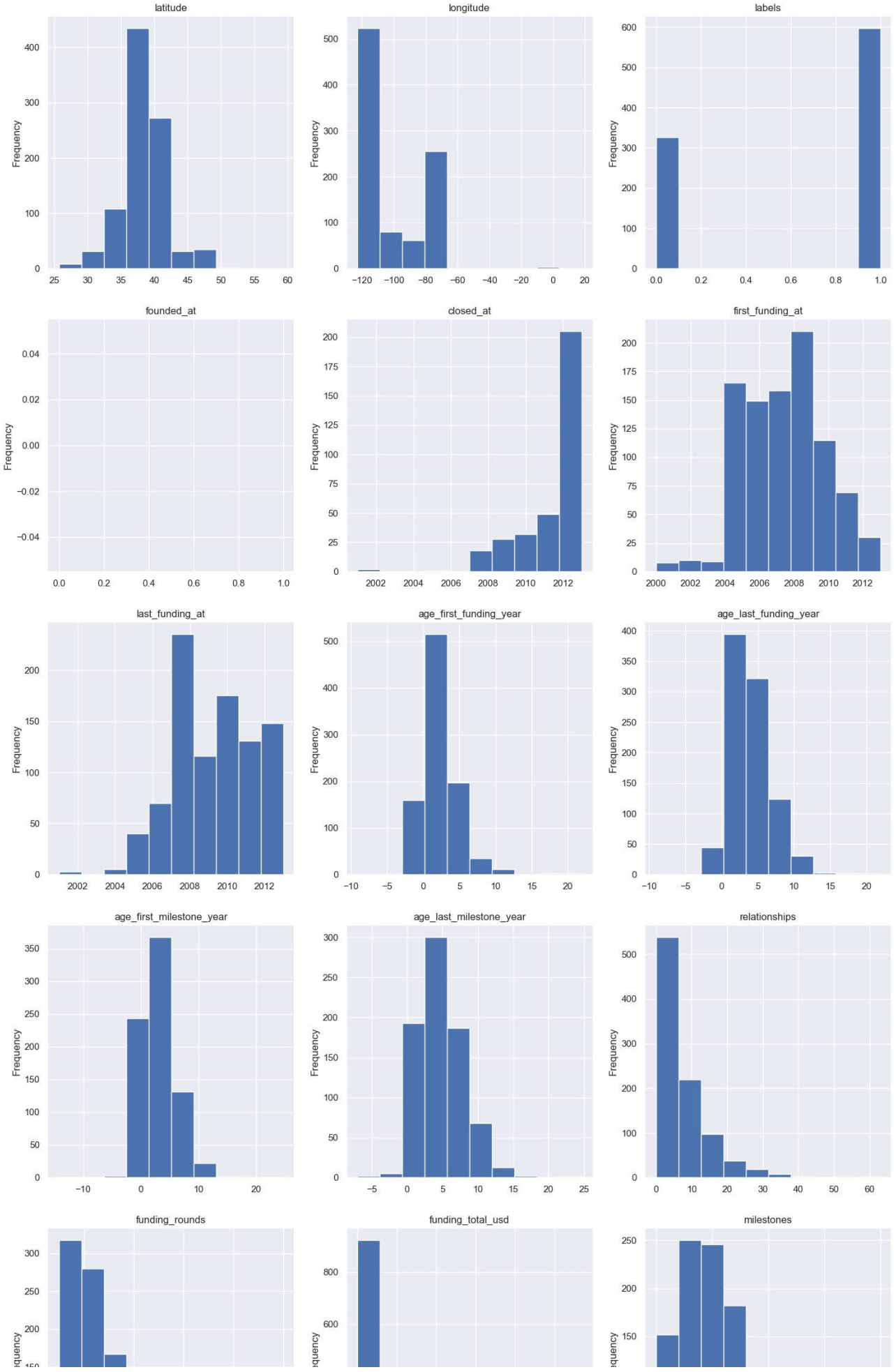

has_roundC
has_roundC
status
acquired
closed

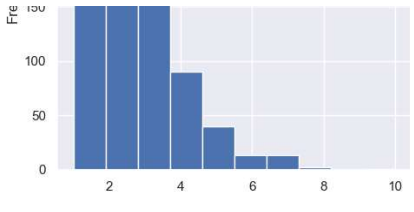

has_roundD
has_roundD
status
acquired
closed

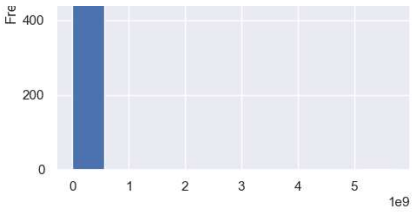

avg_participants
avg_participants
status
acquired
closed

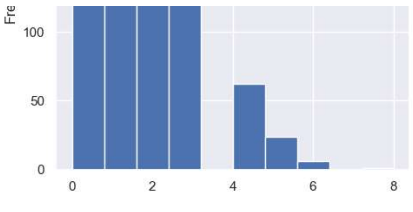

is_top500
is_top500
status
acquired
closed

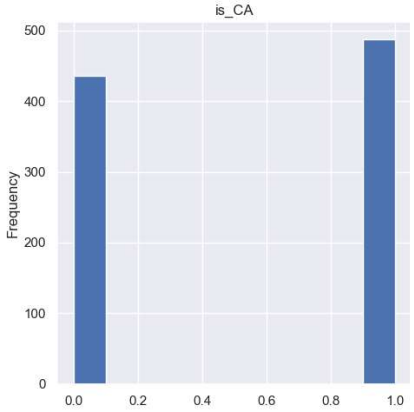
In [21]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# 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 subplots
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()
```

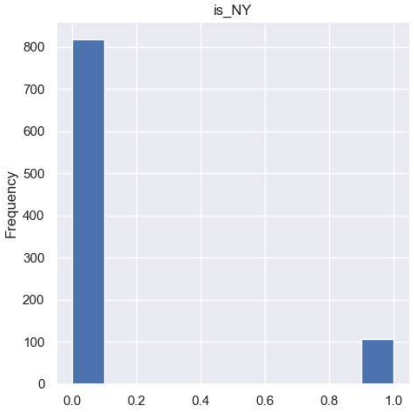

latitude
longitude
labels
founded_at
closed_at
first_funding_at
last_funding_at
age_first_funding_year
age_last_funding_year
age_first_milestone_year
age_last_milestone_year
relationships
funding_rounds
funding_total_usd
milestones
Frequency

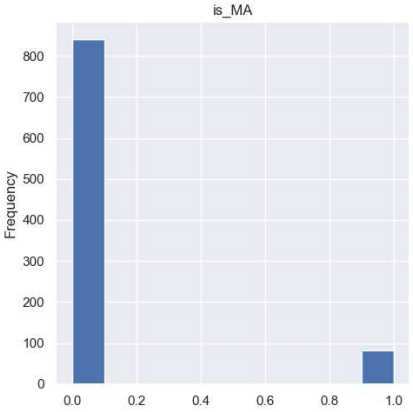


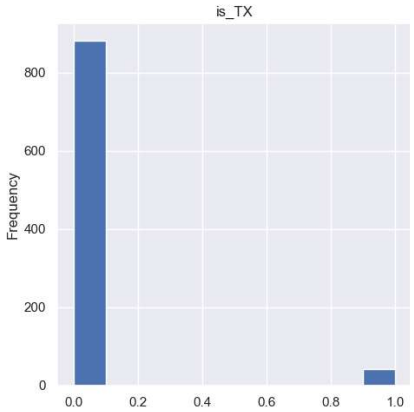

1e9

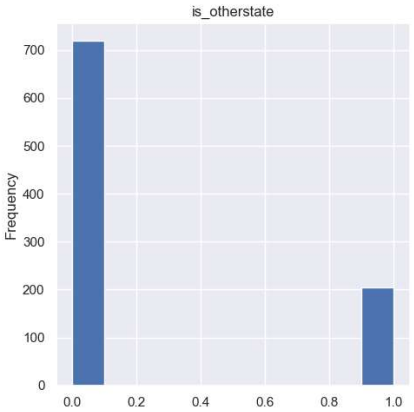


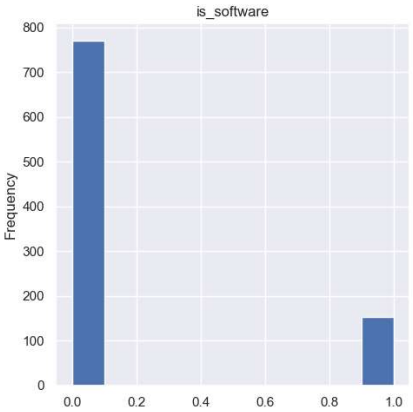

is_CA
Frequency

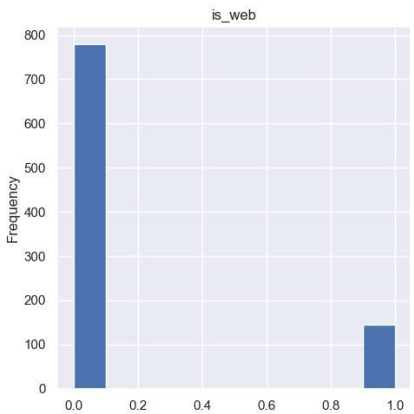

is_NY
Frequency

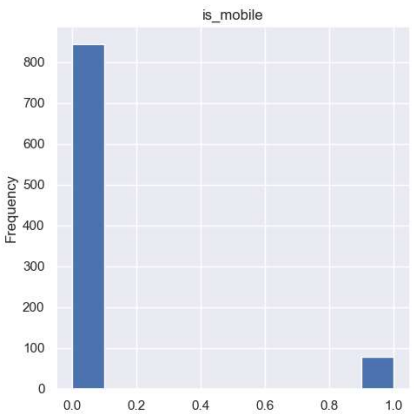

is_MA
Frequency

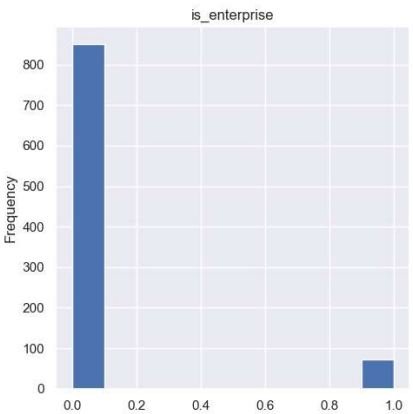

is_TX
Frequency

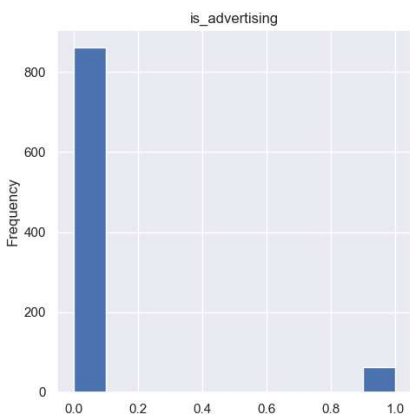

is_otherstate
Frequency

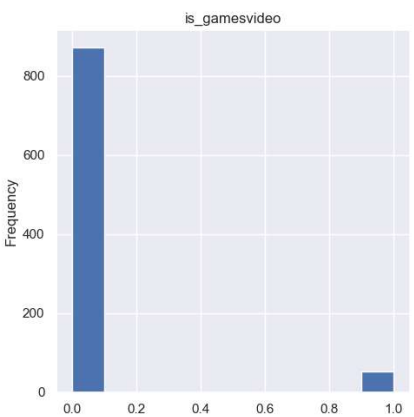

is_software
Frequency

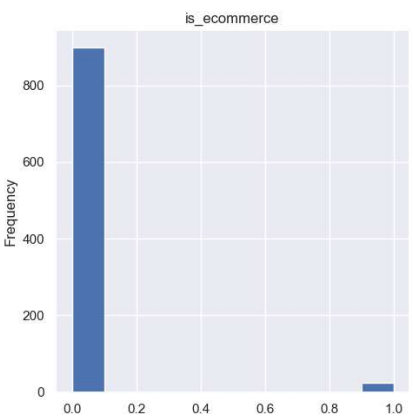

is_web
Frequency

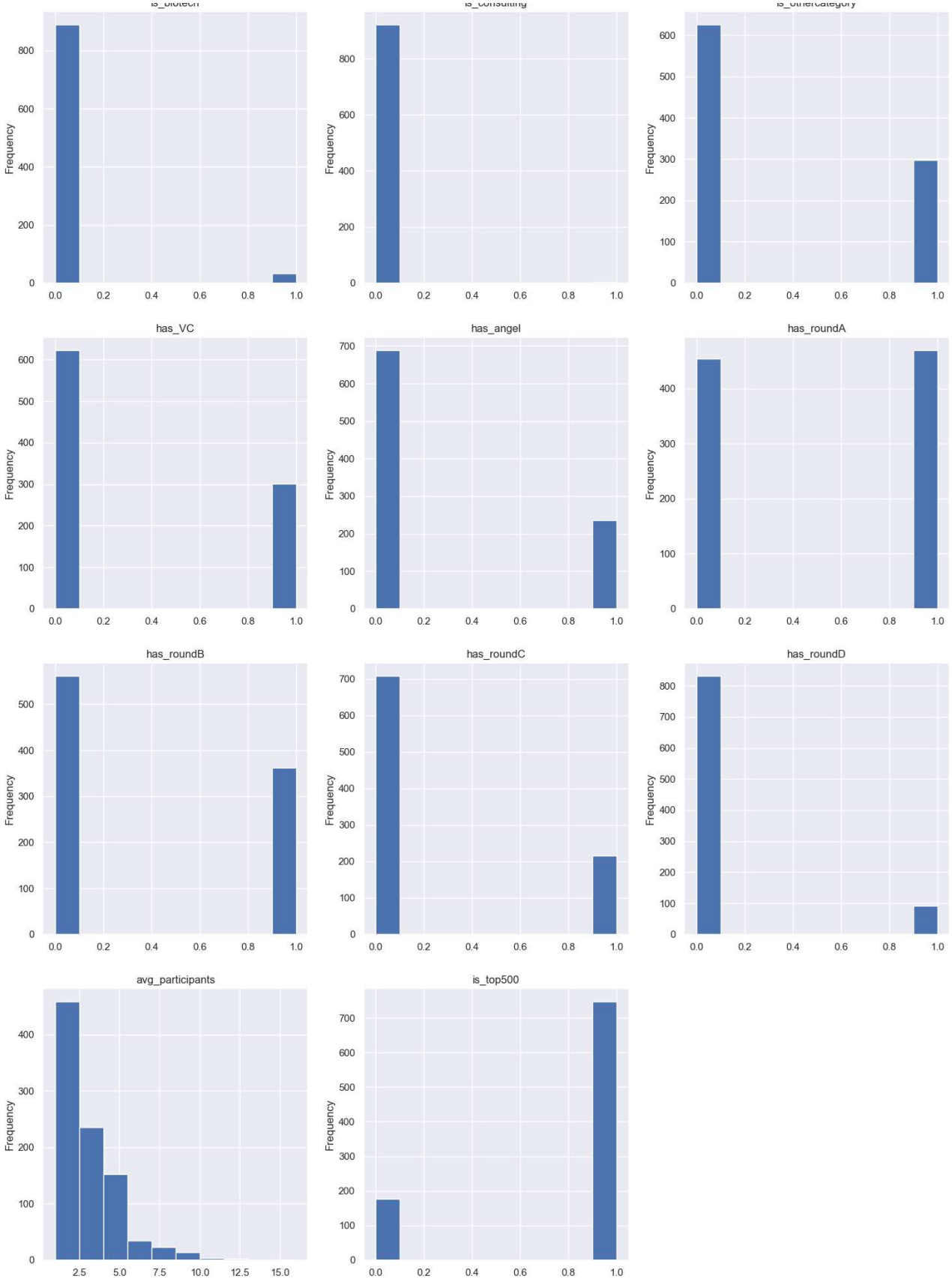

is_mobile
Frequency


is_enterprise
Frequency

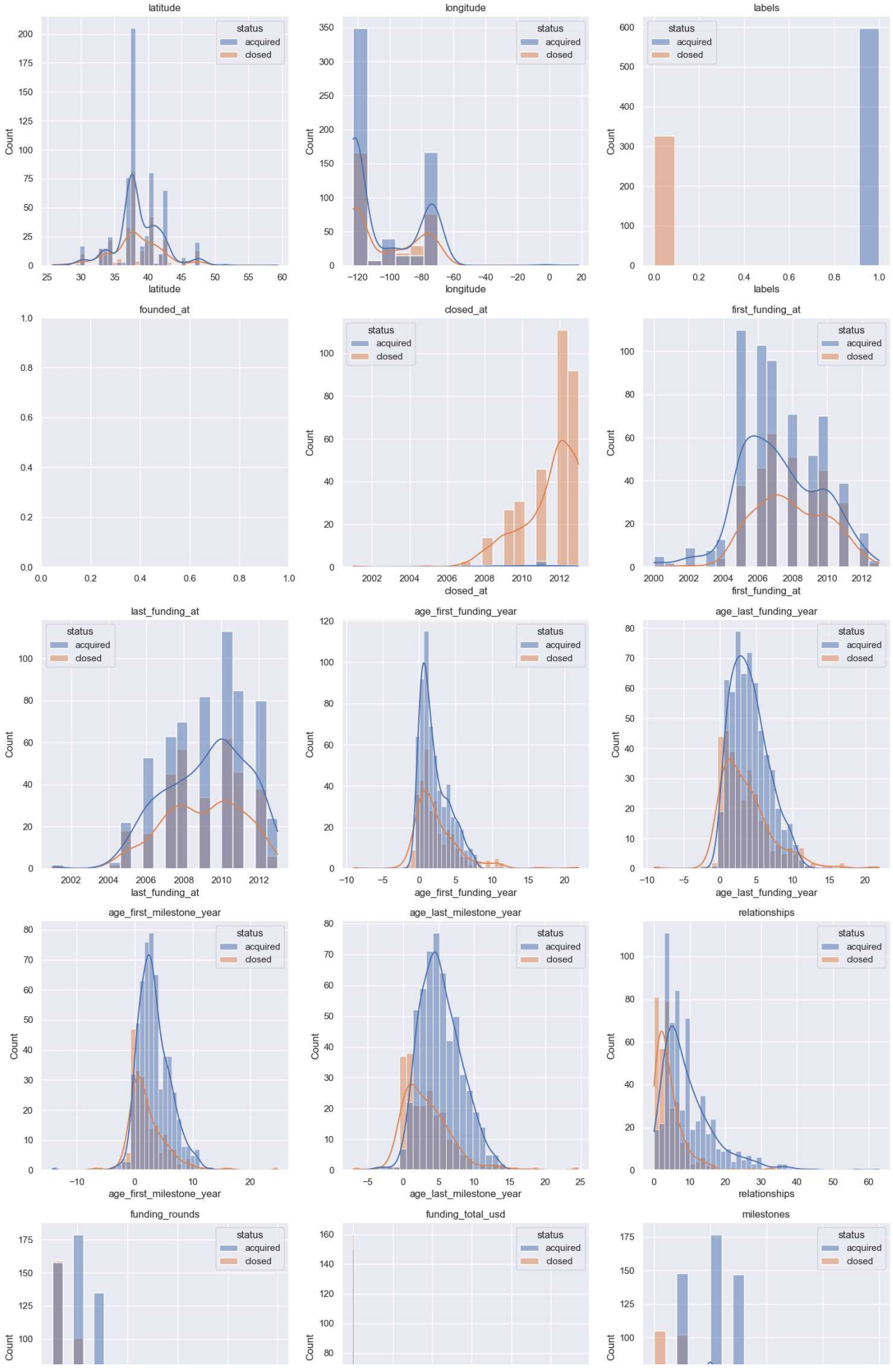

is_advertising
Frequency

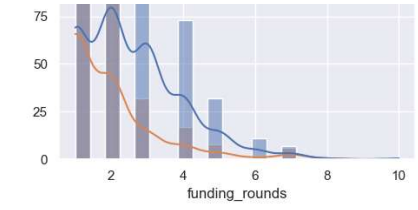

is_gamesvideo
Frequency

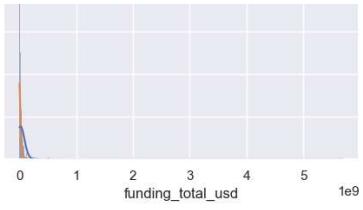

is_ecommerce
Frequency

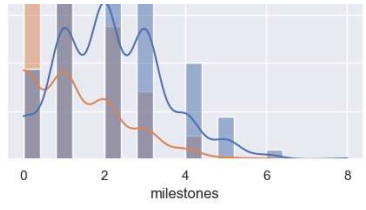

is_biotech
is_consulting
is_othercategory
has_VC
has_angel
has_roundA
has_roundB
has_roundC
has_roundD
avg_participants
is_top500
Frequency

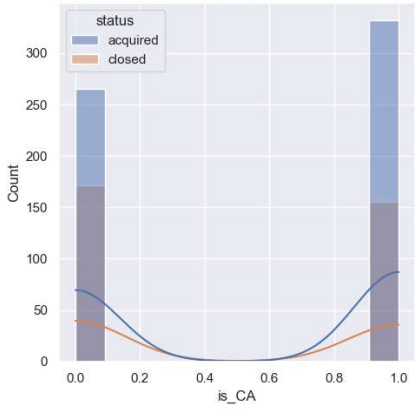
In [22]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# 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 subplots
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='status', 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()
```

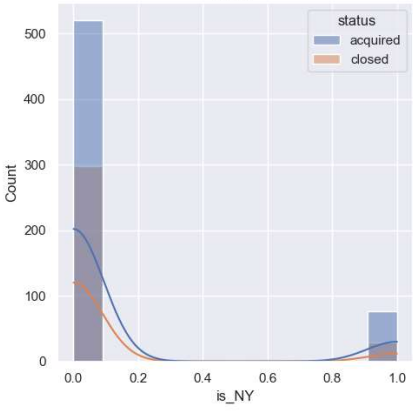

latitude
longitude
labels
founded_at
closed_at
first_funding_at
last_funding_at
age_first_funding_year
age_last_funding_year
age_first_milestone_year
age_last_milestone_year
relationships
funding_rounds
funding_total_usd
milestones
status
acquired
closed
Count

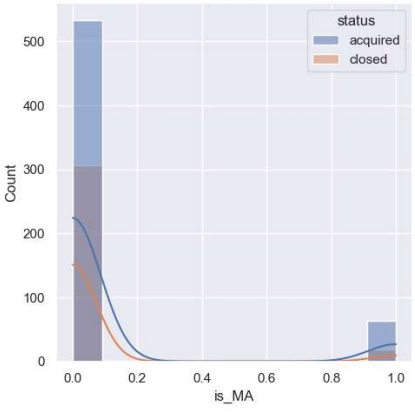


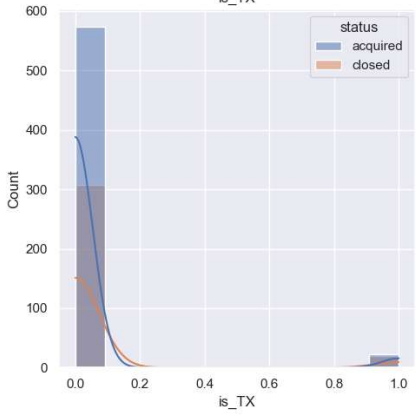


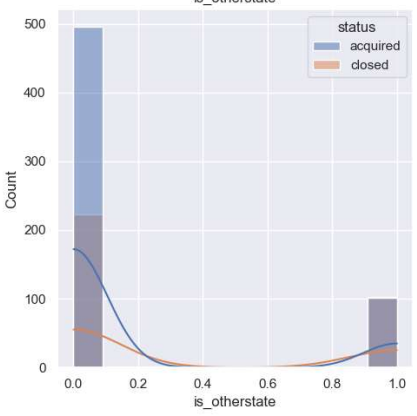


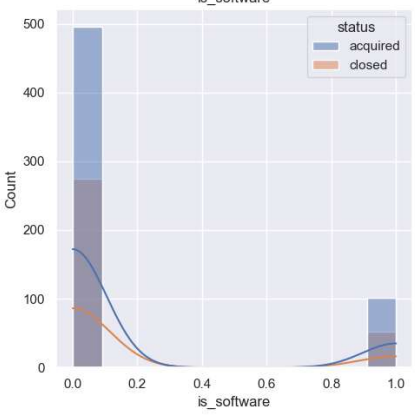


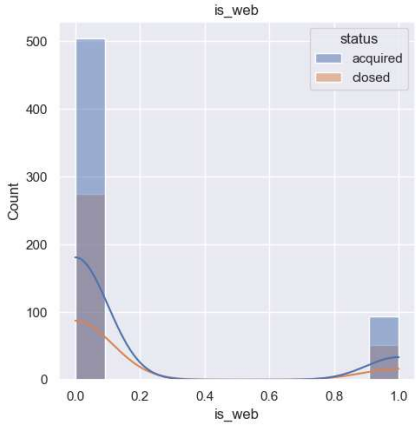


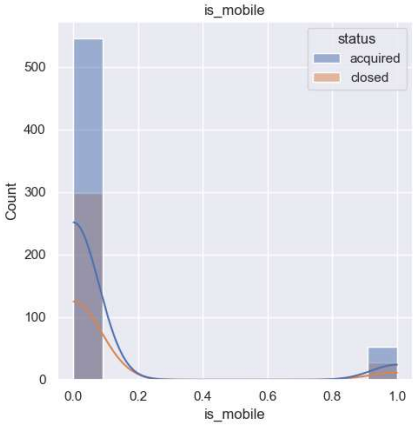


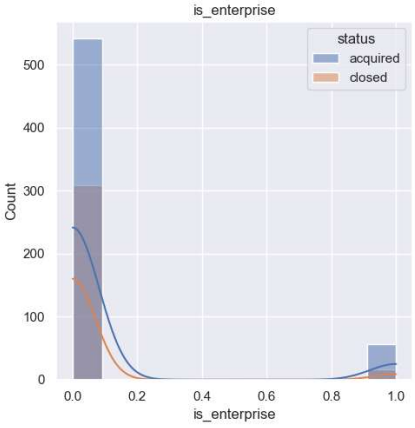


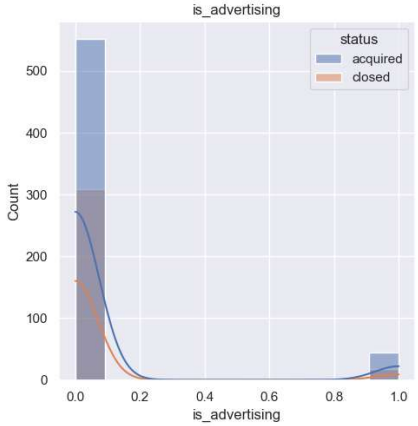


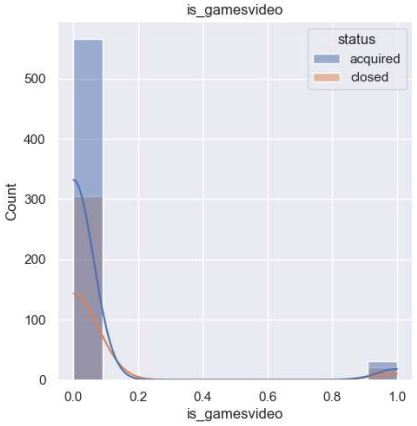


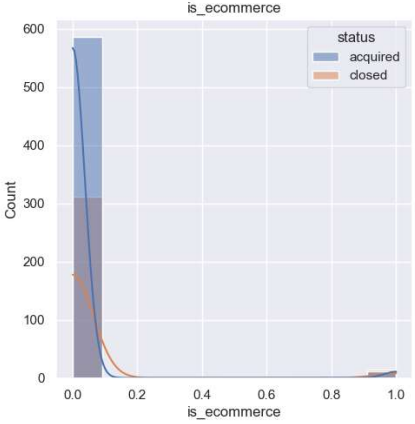




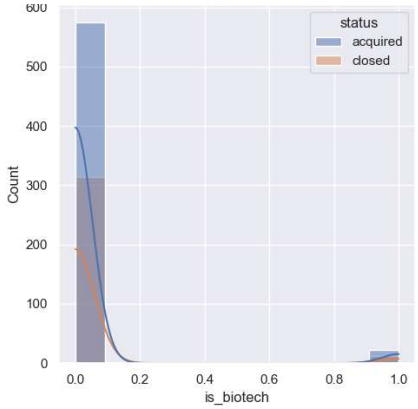


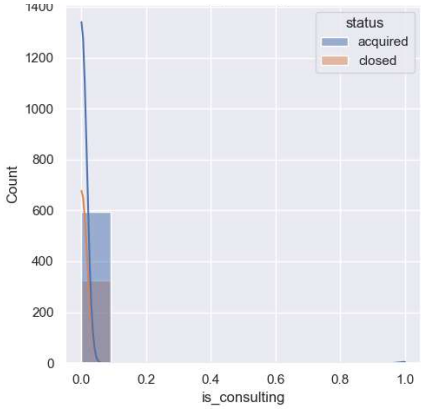


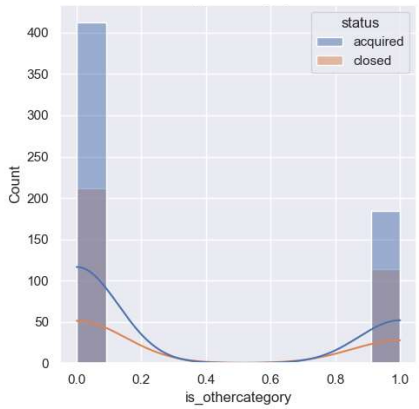


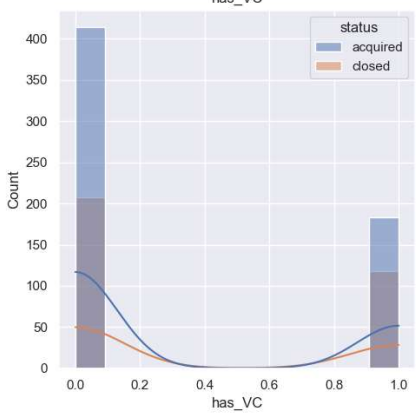


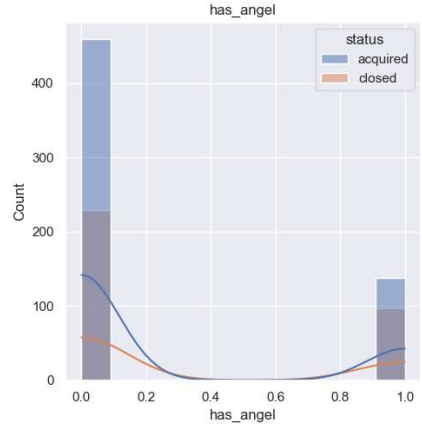
is_biotech is_consulting is_othercategory

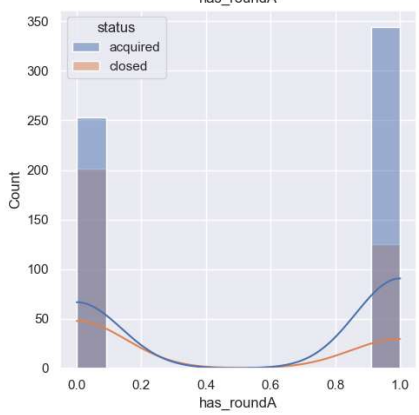

status
acquired
closed
Count
is_biotech

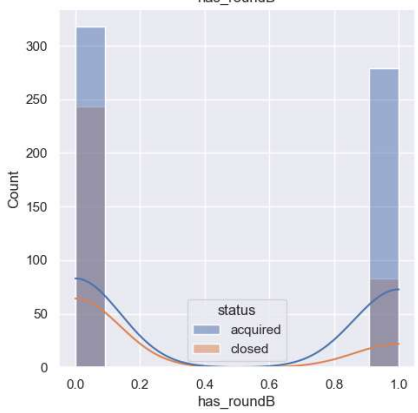

status
acquired
closed
Count
is_consulting

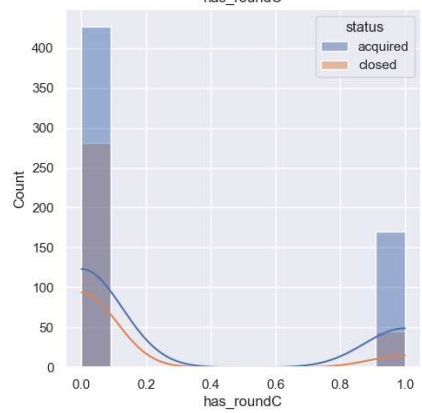

status
acquired
closed
Count
is_othercategory

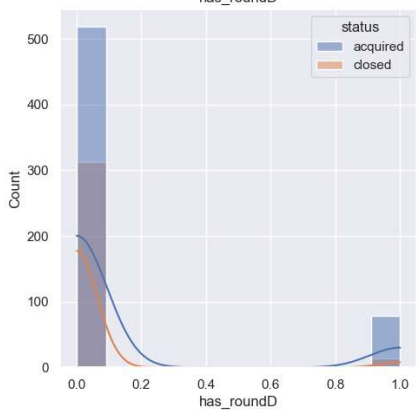

status
acquired
closed
Count
has_VC

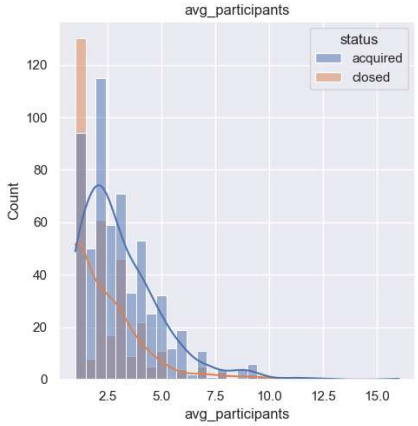

status
acquired
closed
Count
has_angel

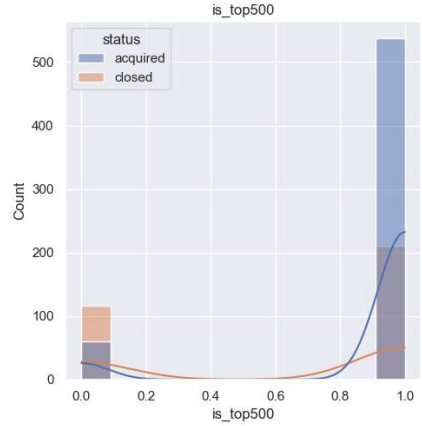

status
acquired
closed
Count
has_roundA


status
acquired
closed
Count
has_roundB


status
acquired
closed
Count
has_roundC


status
acquired
closed
Count
has_roundD

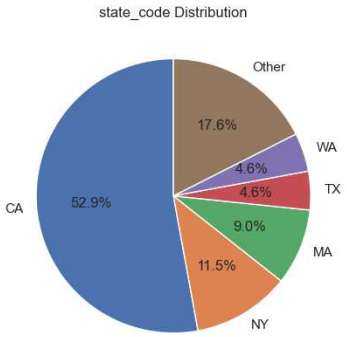

status
acquired
closed
Count
avg_participants

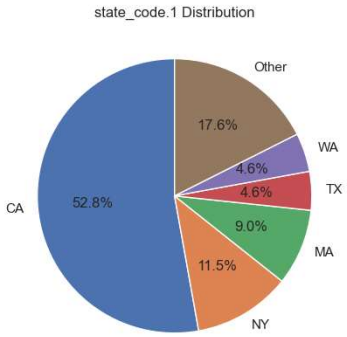

status
acquired
closed
Count
is_top500

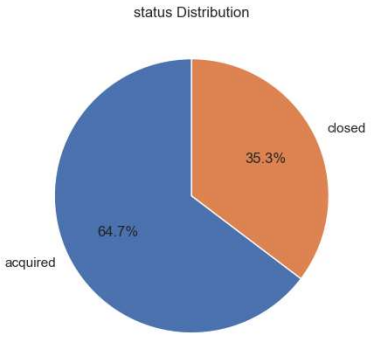
In [24]:

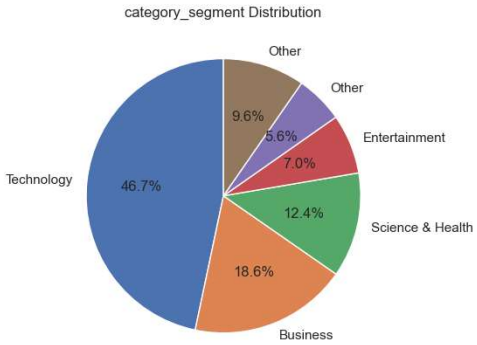
```
# Specify the maximum number of categories to show individually
max_categories = 5

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

# 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(), index=['Other'])
            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%%', startangle=90)
        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()
```

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_916\2023955865.py:25: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_916\2023955865.py:25: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_916\2023955865.py:25: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```









Data Preprocessing Part 2

In [25]:
```
# 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[25]:
```
founded_at                 100.000000
closed_at                   63.705309
age_first_milestone_year    16.468039
age_last_milestone_year     16.468039
state_code.1                 0.108342
dtype: float64
```

In [26]:
```
# Remove founded_at and closed_at column because the null value is higher than 30%
df.drop(columns = ['founded_at', 'closed_at'], inplace=True)
```

In [27]:
```
# Drop null value in state_code.1 column because the null value only 0.1%
df.dropna(subset='state_code.1', inplace=True)


# 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[27]:
```
age_first_milestone_year    16.4859
age_last_milestone_year     16.4859
dtype: float64
```

In [28]:
```
# Fill age_first_milestone_year and age_last_milestone_year with mean because the bar chart diagram is balance
df['age_first_milestone_year'].fillna(df['age_first_milestone_year'].mean(), inplace=True)
df['age_last_milestone_year'].fillna(df['age_last_milestone_year'].mean(), inplace=True)
```

In [29]:
```
# 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[29]:
```
Series([], dtype: float64)
```

Label Encoding for Object Datatypes

In [30]:
```
# 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()}")
```

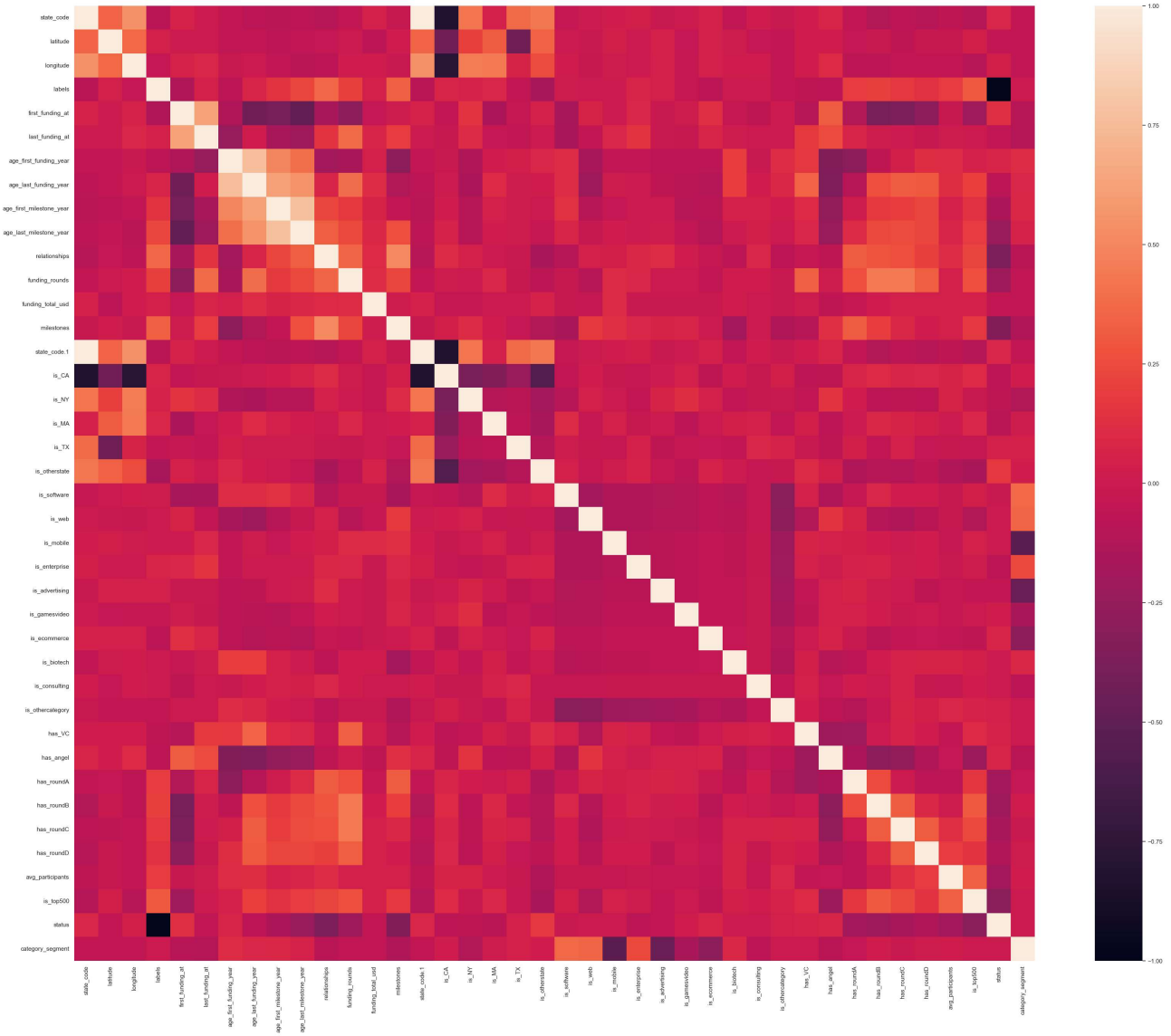
```
state_code: ['CA' 'MA' 'KY' 'NY' 'CO' 'VA' 'TX' 'WA' 'IL' 'NC' 'PA' 'GA' 'NH' 'MO'
 'FL' 'NJ' 'WV' 'MI' 'DC' 'CT' 'MD' 'OH' 'TN' 'MN' 'RI' 'OR' 'UT' 'ME'
 'NV' 'NM' 'IN' 'AZ' 'ID' 'AR' 'WI']
state_code.1: ['CA' 'MA' 'KY' 'NY' 'CO' 'VA' 'TX' 'WA' 'IL' 'NC' 'PA' 'GA' 'NH' 'MO'
 'FL' 'NJ' 'WV' 'MI' 'DC' 'CT' 'MD' 'OH' 'TN' 'MN' 'RI' 'OR' 'UT' 'ME'
 'NV' 'NM' 'IN' 'AZ' 'ID' 'AR' 'WI']
status: ['acquired' 'closed']
category_segment: ['Entertainment' 'Technology' 'Business' 'Education & PR' 'Other'
 'Lifestyle' 'Consulting & Analytics' 'Science & Health']
```

In [31]:
```
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()}")
```

```
state_code: [ 2 12 11 23  3 31 29 32  9 18 26  7 19 17  6 20 34 15  5  4 13 24 28 16
 27 25 30 14 22 21 10  1  8  0 33]
state_code.1: [ 2 12 11 23  3 31 29 32  9 18 26  7 19 17  6 20 34 15  5  4 13 24 28 16
 27 25 30 14 22 21 10  1  8  0 33]
status: [0 1]
category_segment: [3 7 0 2 5 4 1 6]
```

In [32]:
```
# Correlation Heatmap
plt.figure(figsize=(40, 32))
sns.heatmap(df.corr(), fmt='.2g', annot=False)
```

Out[32]: <AxesSubplot:>



In [33]:
```
df.drop(columns = ['state_code.1', 'labels'], inplace=True)
df.shape
```

Out[33]: (922, 38)

Train Test Split

In [34]:
```
X = df.drop('status', axis=1)
y = df['status']
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)
```

Remove Outlier from Train Data using Z-Score

In [36]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['first_funding_at', 'age_first_milestone_year', 'age_last_milestone_year',
                    'age_first_funding_year', 'age_last_funding_year', 'relationships', 'funding_total_usd',
                    'avg_participants']

# 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 [37]:
```
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': 4, 'min_samples_leaf': 2, 'min_samples_split': 2, 'random_state': 0}
```

In [38]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=4, min_samples_leaf=2, min_samples_split=2, class_wei
dtree.fit(X_train, y_train)
```

Out[38]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=4, min_samples_leaf=2,
                       random_state=0)
```

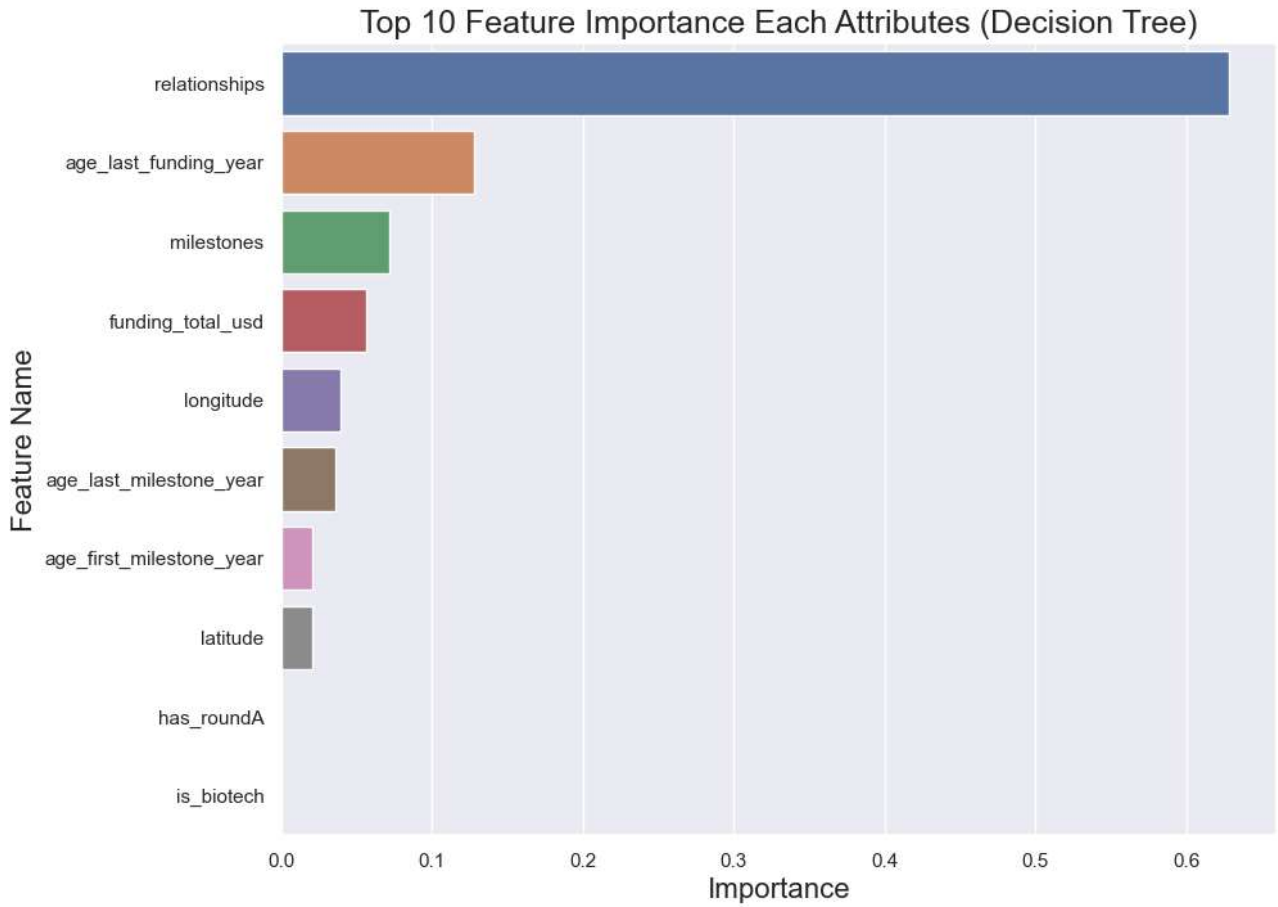
In [39]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 73.51 %
```

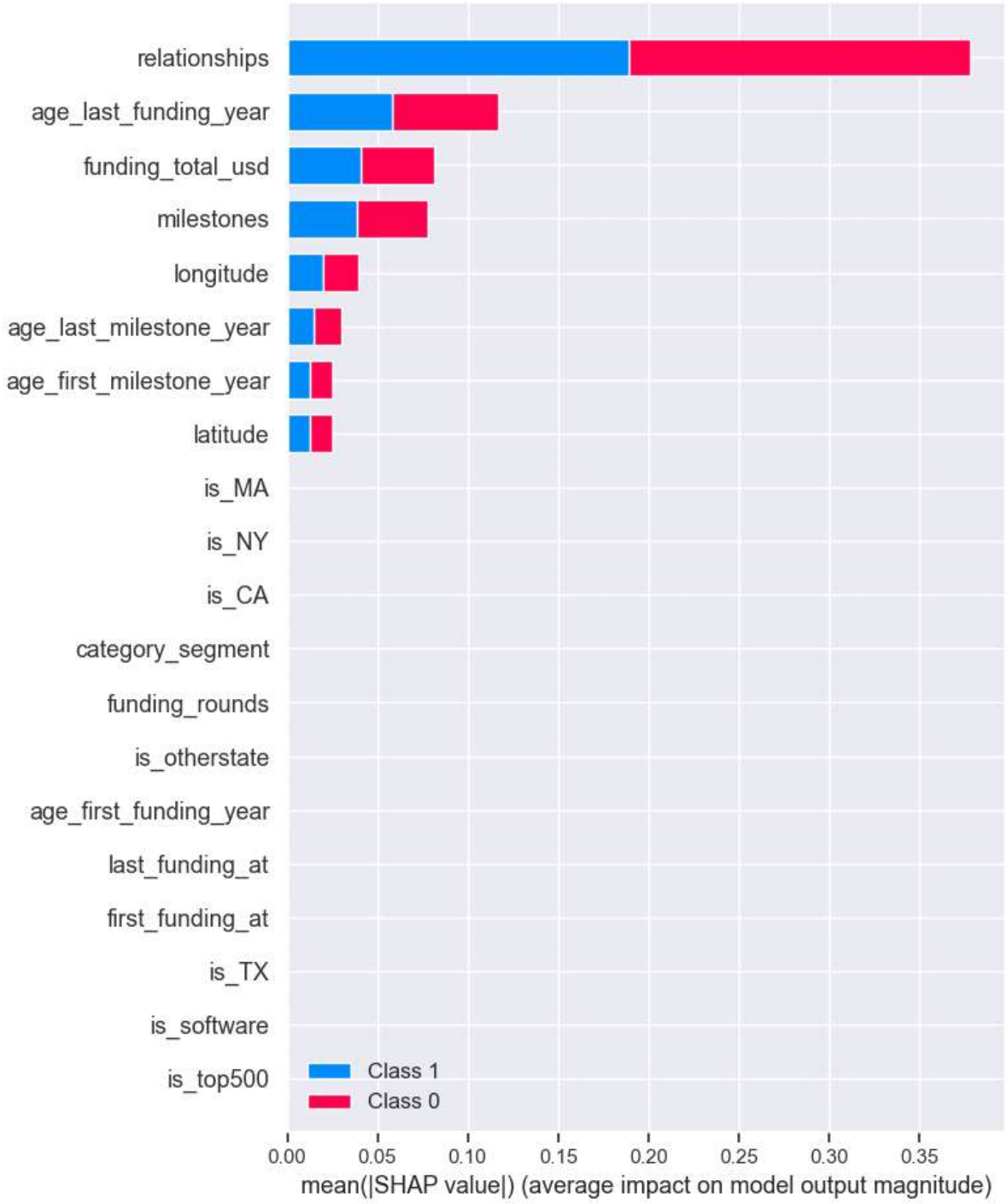
In [40]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, 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 :  0.7351351351351352
Precision Score :  0.7351351351351352
Recall Score :  0.7351351351351352
Jaccard Score :  0.5811965811965812
Log Loss :  9.148207751846043
```

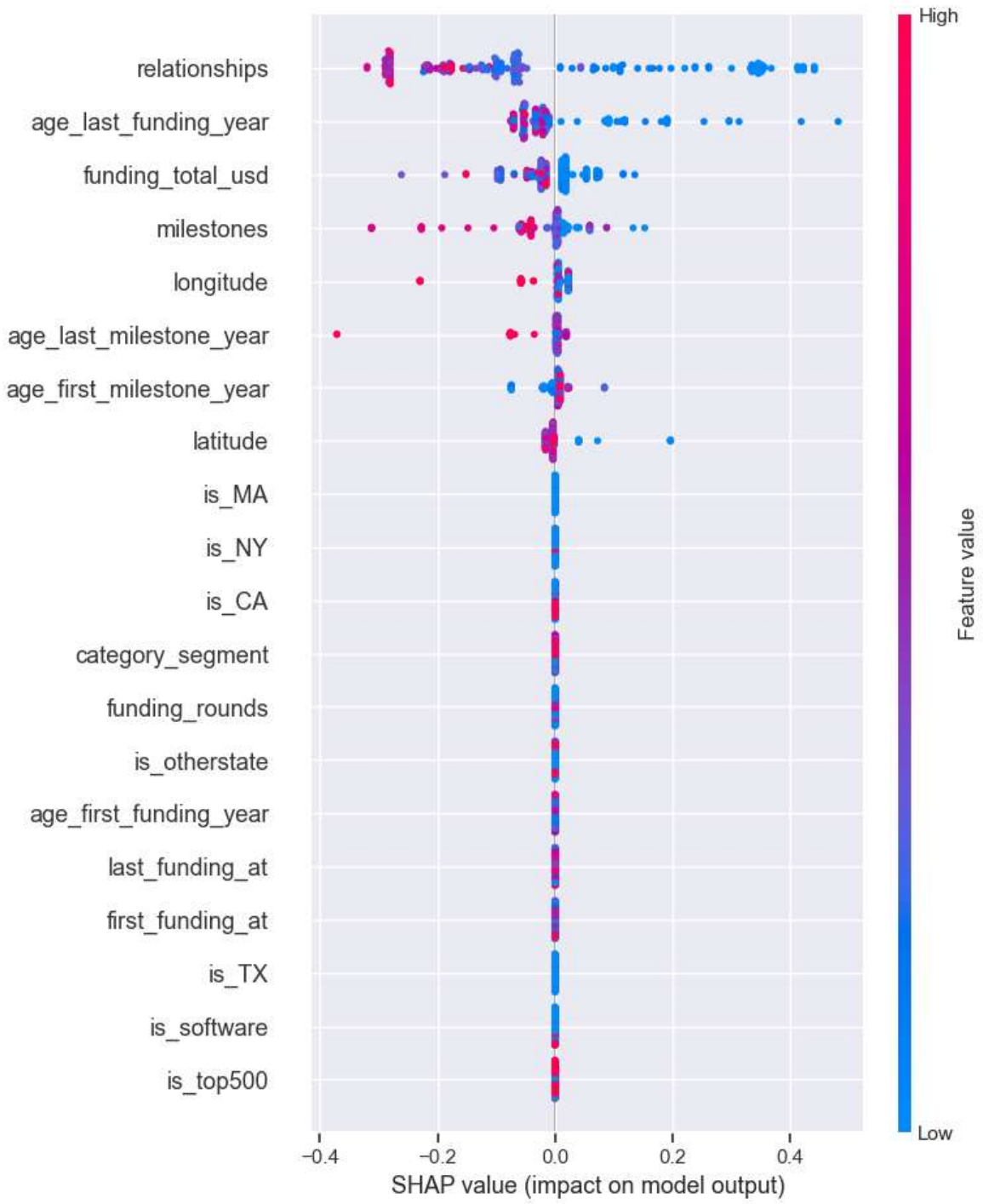
In [41]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [42]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

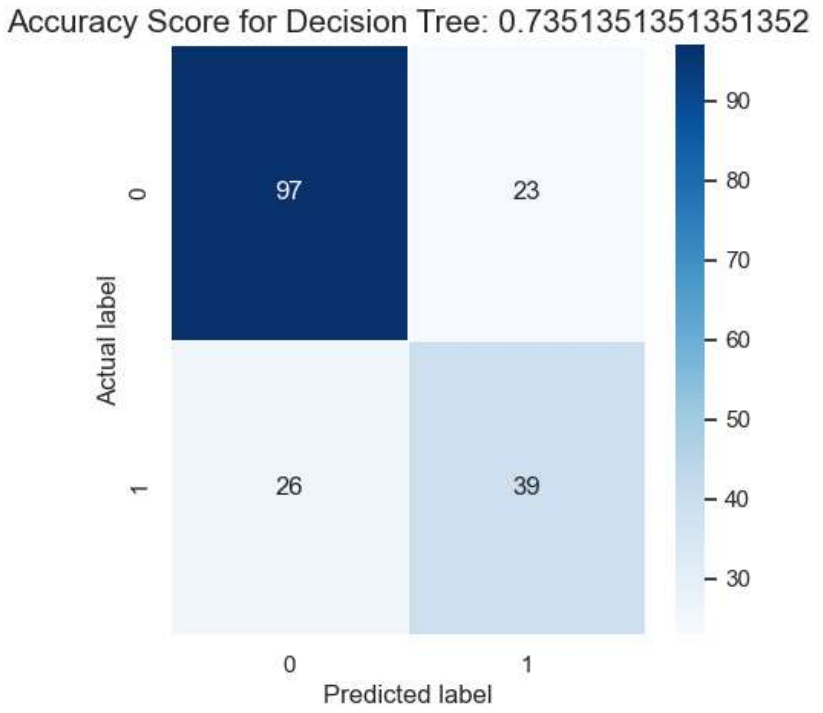


In [43]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



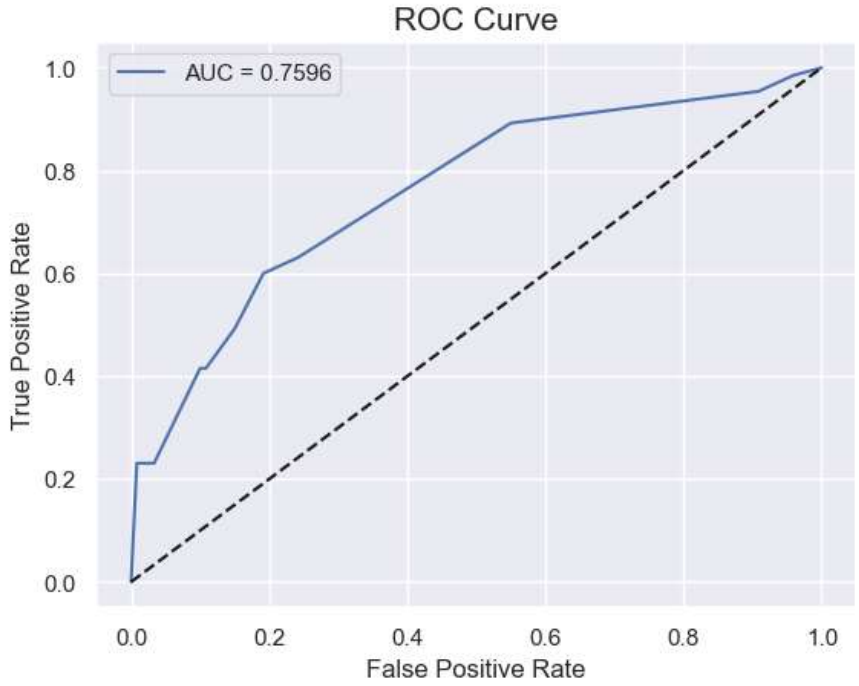
In [44]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[44]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.7351351351351352')
```



In [45]:
```
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_actual']), pd.DataFrame(y_pred_prol
df_actual_predicted.index = y_test.index

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

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[45]: <matplotlib.legend.Legend at 0x27702eda310>



Random Forest Classifier

In [46]:
```
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': 10, 'max_features': None, 'n_estimators': 100, 'random_state': 42}
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_depth=10, max_features=None, n_estimators=100, class_weight=
rfc.fit(X_train, y_train)
```

Out[47]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10, max_features=None,
                       random_state=42)
```

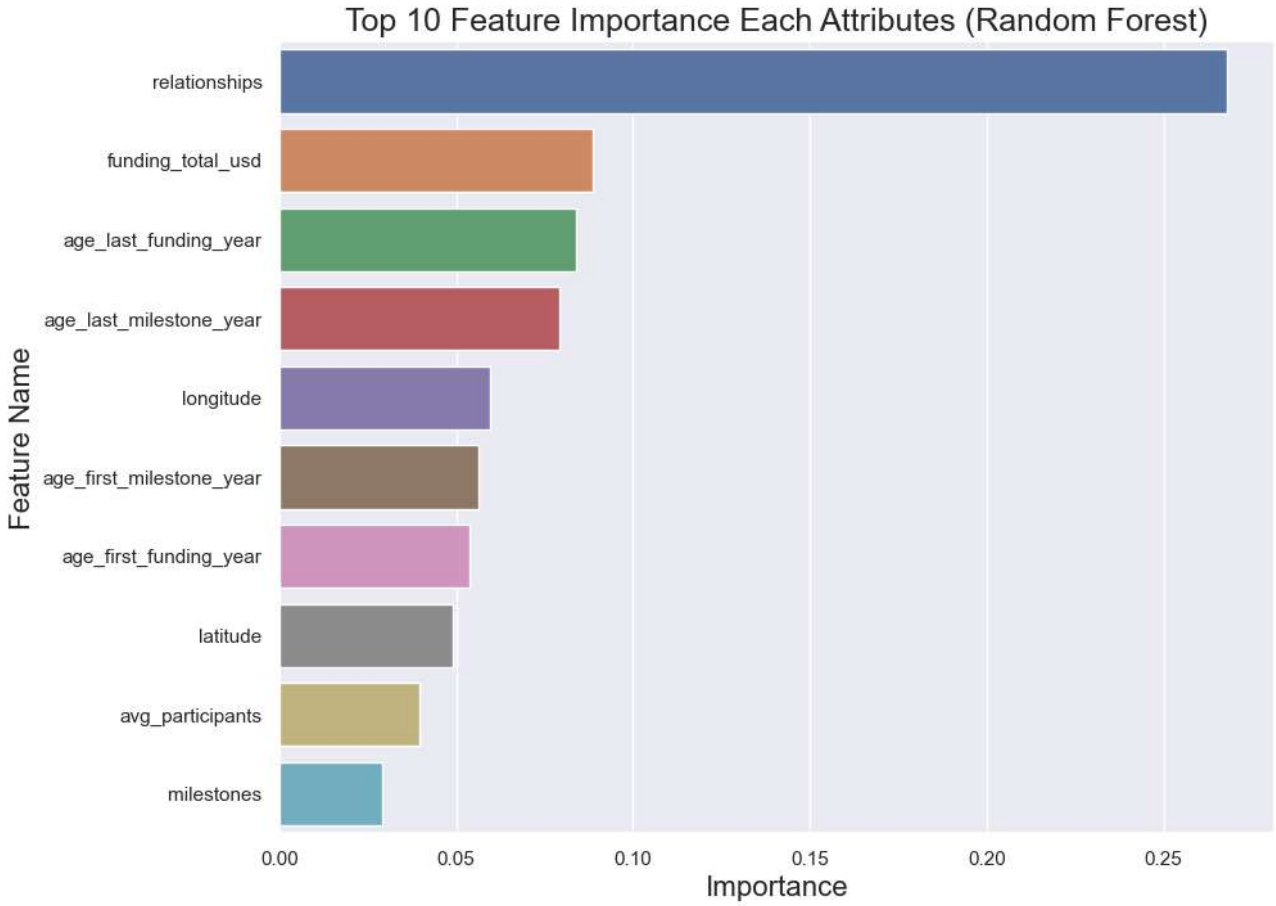
In [48]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 77.3 %
```

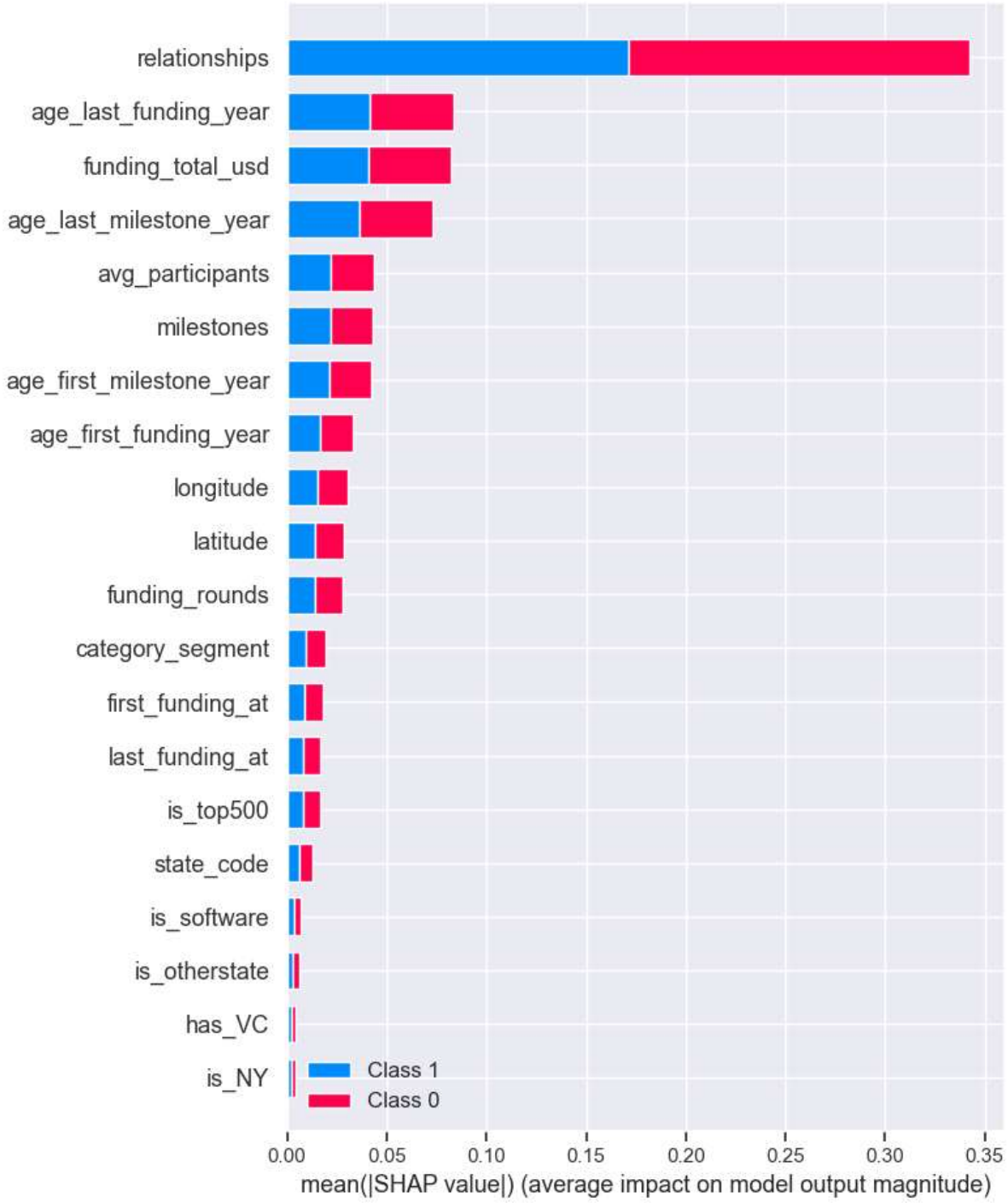
In [49]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, 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 :  0.772972972972973
Precision Score :  0.772972972972973
Recall Score :  0.772972972972973
Jaccard Score :  0.6299559471365639
Log Loss :  7.841287587867087
```

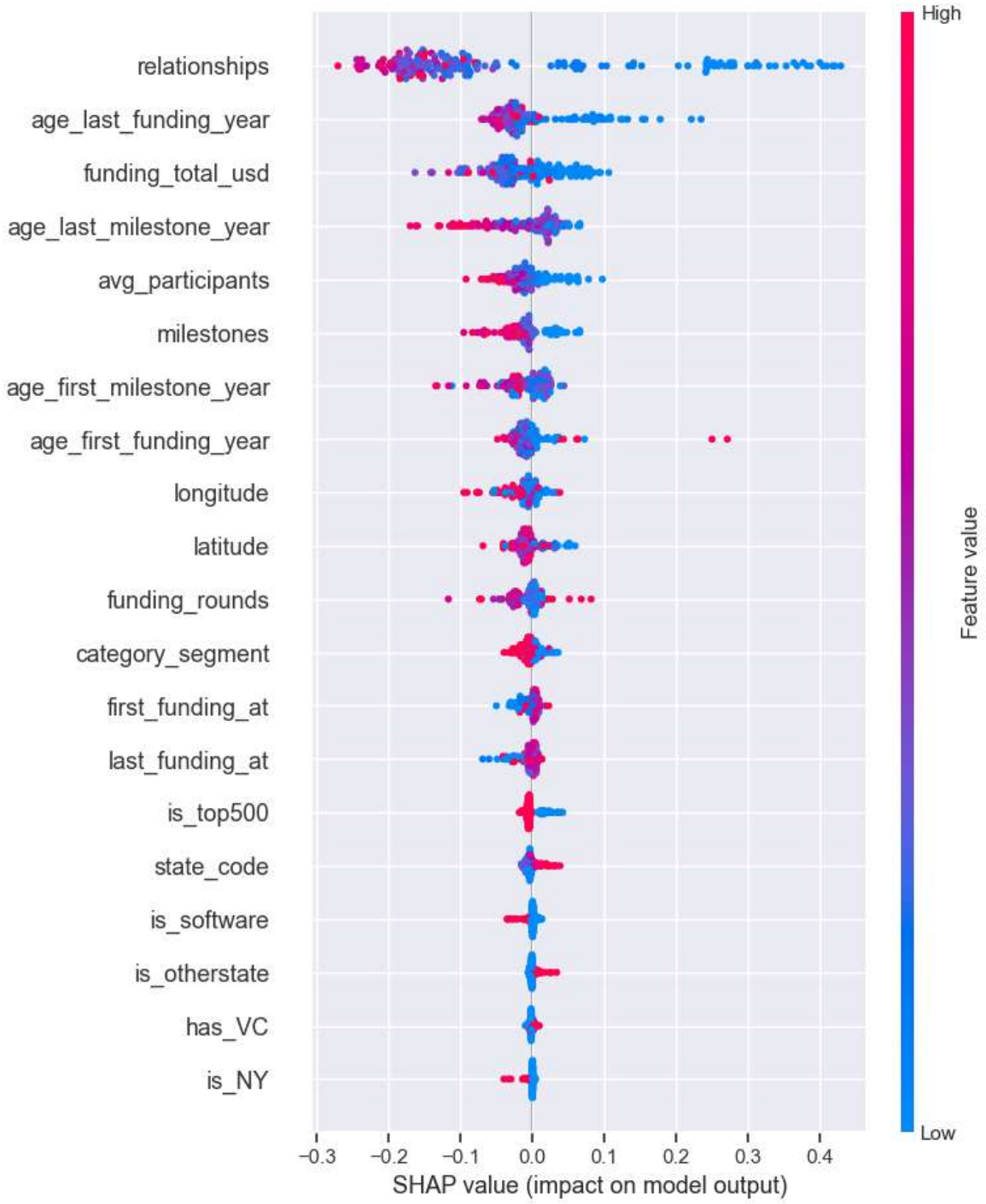
In [50]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [51]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

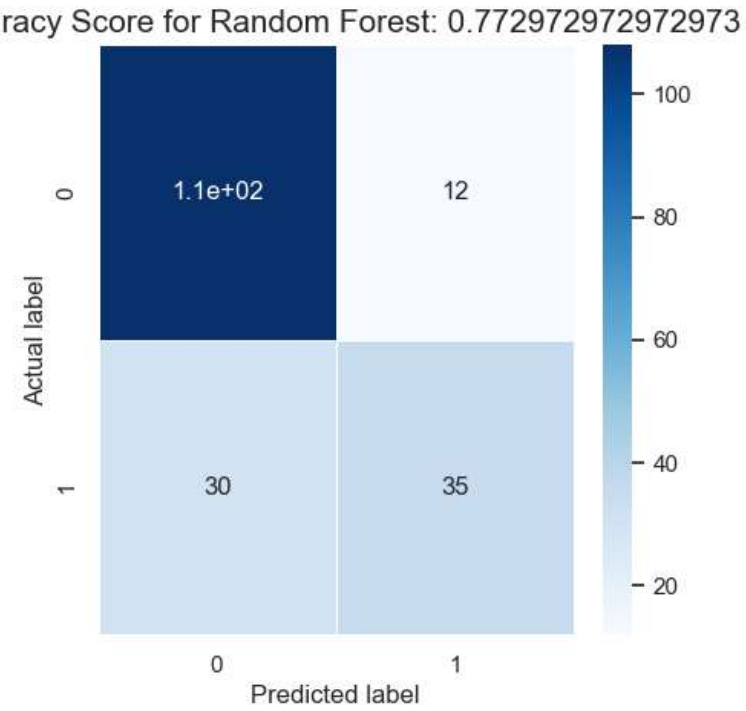


In [52]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



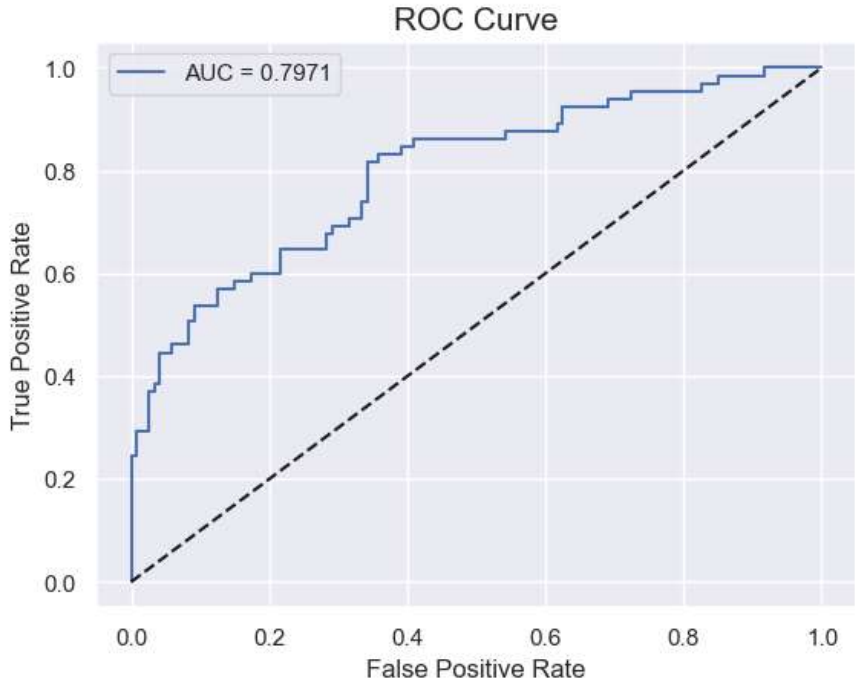
In [53]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[53]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.772972972972973')
```



In [54]:
```
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_actual']), pd.DataFrame(y_pred_prol
df_actual_predicted.index = y_test.index

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

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[54]: <matplotlib.legend.Legend at 0x27705917a60>



XGBoost Classifier

In [55]:
```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

# Create an XGBoost classifier
xgb = XGBClassifier()

# Define the parameter grid for grid search
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'gamma': [0, 0.1, 0.2]
}

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

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

```
{'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100}
```

In [56]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(gamma=0.2, learning_rate=0.1, max_depth=7, n_estimators=100)
xgb.fit(X_train, y_train)
```

Out[56]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=7, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [57]:
```
from sklearn.metrics import accuracy_score
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 74.05 %
```

In [58]:
```
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)))
```

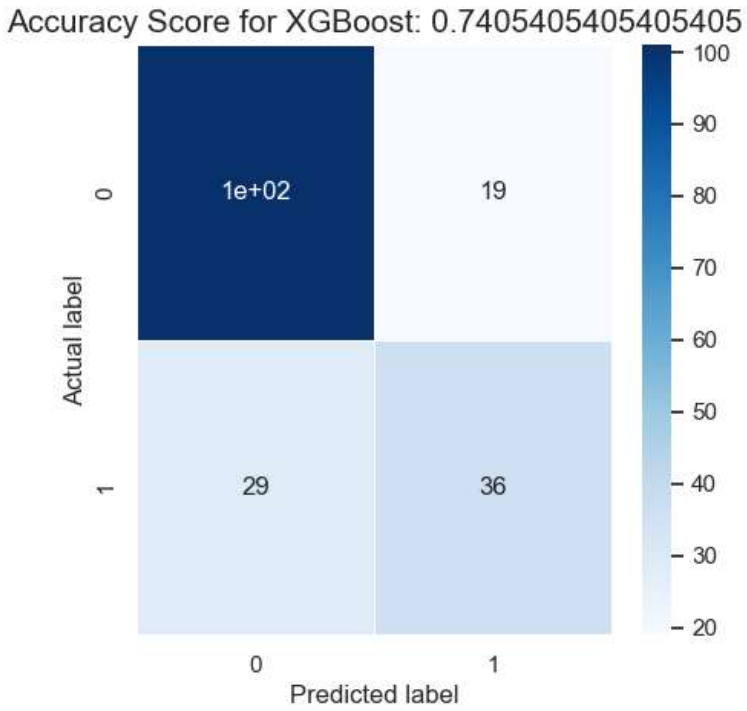
```
F-1 Score :  0.7405405405405405
Precision Score :  0.7405405405405405
Recall Score :  0.7405405405405405
Jaccard Score :  0.5879828326180258
Log Loss :  8.961494374631819
```

In [59]:
```
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

In [60]:
```
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 XGBoost: {0}'.format(xgb.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[60]: Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.7405405405405405')



In [61]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = xgb.predict_proba(X_test)[:][:,1]

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

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

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[61]: <matplotlib.legend.Legend at 0x2770069fd00>