```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

url = "C:/test.csv"  # Update the path
data = pd.read_csv(url)

print(data.head())
print(data.info())
print(data.describe())
```

```
   id state  account_length      area_code international_plan
voice_mail_plan  \
0   1    KS             128  area_code_415                 no
yes
1   2    AL             118  area_code_510                yes
no
2   3    IA              62  area_code_415                 no
no
3   4    VT              93  area_code_510                 no
no
4   5    NE             174  area_code_415                 no
no

   number_vmail_messages  total_day_minutes  total_day_calls  \
0                     25              265.1              110
1                      0              223.4               98
2                      0              120.7               70
3                      0              190.7              114
4                      0              124.3               76

   total_day_charge  total_eve_minutes  total_eve_calls
total_eve_charge  \
0             45.07              197.4               99
16.78
1             37.98              220.6              101
18.75
2             20.52              307.2               76
26.11
3             32.42              218.2              111
18.55
4             21.13              277.1              112
23.55

   total_night_minutes  total_night_calls  total_night_charge  \
0                244.7                 91               11.01
1                203.9                118                9.18
2                203.0                 99                9.14
3                129.6                121                5.83
```

```
4                250.7                 115                11.28

   total_intl_minutes  total_intl_calls  total_intl_charge  \
0                10.0                 3               2.70
1                 6.3                 6               1.70
2                13.1                 6               3.54
3                 8.1                 3               2.19
4                15.5                 5               4.19

   number_customer_service_calls
0                              1
1                              0
2                              4
3                              3
4                              3
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              750 non-null    int64
 1   state                           750 non-null    object
 2   account_length                  750 non-null    int64
 3   area_code                       750 non-null    object
 4   international_plan              750 non-null    object
 5   voice_mail_plan                 750 non-null    object
 6   number_vmail_messages           750 non-null    int64
 7   total_day_minutes               750 non-null    float64
 8   total_day_calls                 750 non-null    int64
 9   total_day_charge                750 non-null    float64
 10  total_eve_minutes               750 non-null    float64
 11  total_eve_calls                 750 non-null    int64
 12  total_eve_charge                750 non-null    float64
 13  total_night_minutes             750 non-null    float64
 14  total_night_calls               750 non-null    int64
 15  total_night_charge              750 non-null    float64
 16  total_intl_minutes              750 non-null    float64
 17  total_intl_calls                750 non-null    int64
 18  total_intl_charge               750 non-null    float64
 19  number_customer_service_calls   750 non-null    int64
dtypes: float64(8), int64(8), object(4)
memory usage: 117.3+ KB
None
              id  account_length  number_vmail_messages
total_day_minutes  \
count  750.00000      750.000000             750.000000
750.000000
mean   375.50000      100.385333               8.454667
180.454933
```

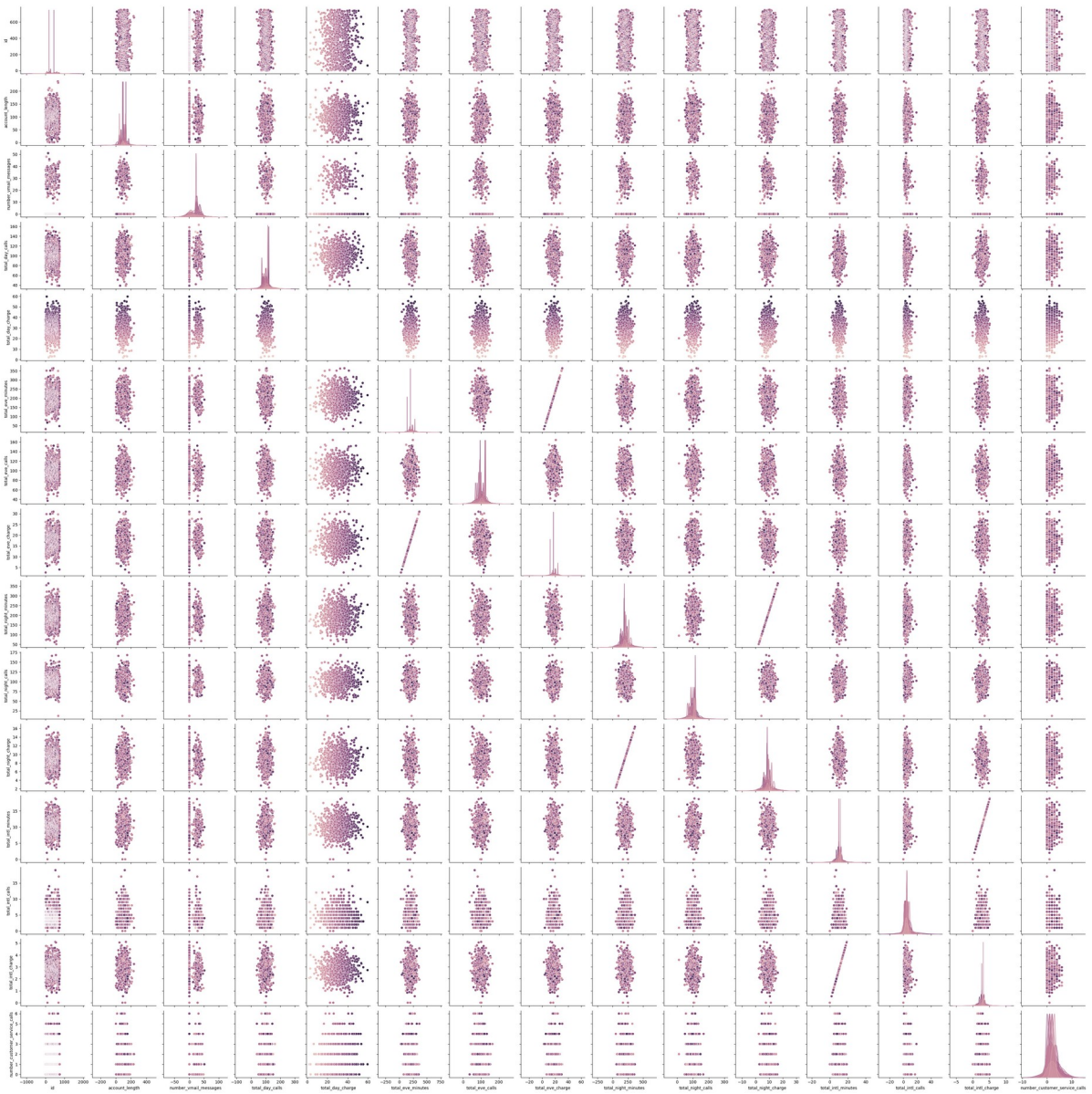
```
std    216.65064       39.699029               14.123712
53.258337
min      1.00000        1.000000                0.000000
12.500000
25%    188.25000       74.000000                0.000000
146.625000
50%    375.50000      101.000000                0.000000
178.200000
75%    562.75000      126.000000               21.000000
215.975000
max    750.00000      238.000000               51.000000
350.800000

       total_day_calls  total_day_charge  total_eve_minutes
total_eve_calls  \
count       750.000000        750.000000         750.000000
750.000000
mean        100.721333         30.677920         203.258267
100.273333
std          19.718539          9.053756          52.185471
19.367535
min          39.000000          2.130000          31.200000
37.000000
25%          88.000000         24.925000         166.800000
87.000000
50%         101.000000         30.295000         203.350000
101.000000
75%         114.000000         36.715000         235.975000
113.000000
max         163.000000         59.640000         363.700000
164.000000

       total_eve_charge  total_night_minutes  total_night_calls  \
count        750.000000           750.000000         750.000000
mean          17.277080           199.619467         100.370667
std            4.435638            51.531351          19.185238
min            2.650000            50.900000          12.000000
25%           14.177500           164.475000          88.000000
50%           17.285000           199.450000         100.500000
75%           20.057500           234.800000         113.000000
max           30.910000           364.300000         168.000000

       total_night_charge  total_intl_minutes  total_intl_calls  \
count          750.000000          750.000000        750.000000
mean             8.982827           10.294133          4.485333
std              2.318920            2.770340          2.421901
min              2.290000            0.000000          0.000000
25%              7.402500            8.525000          3.000000
50%              8.975000           10.300000          4.000000
75%             10.565000           12.100000          6.000000
```

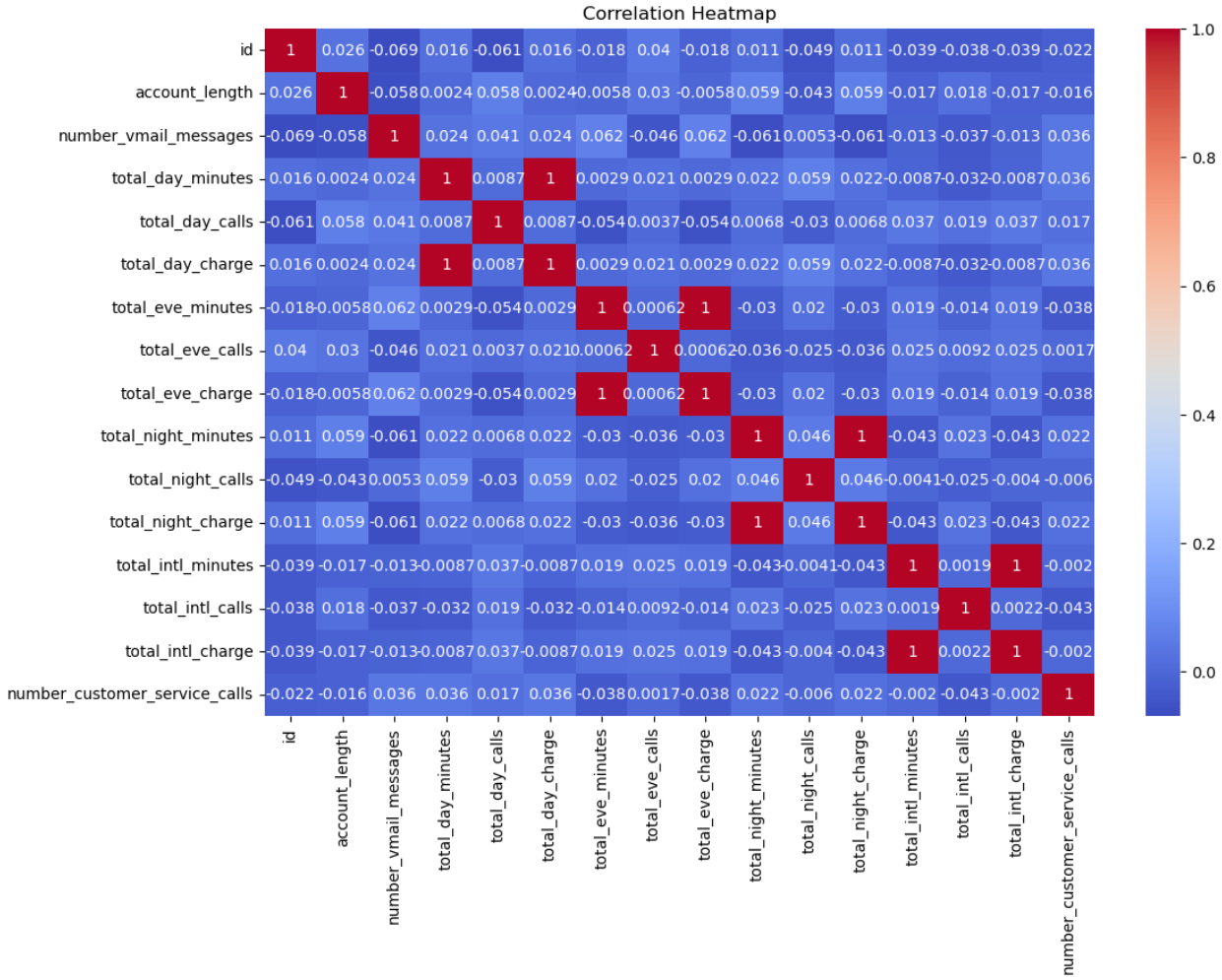
```
max              16.390000           18.900000         19.000000
```

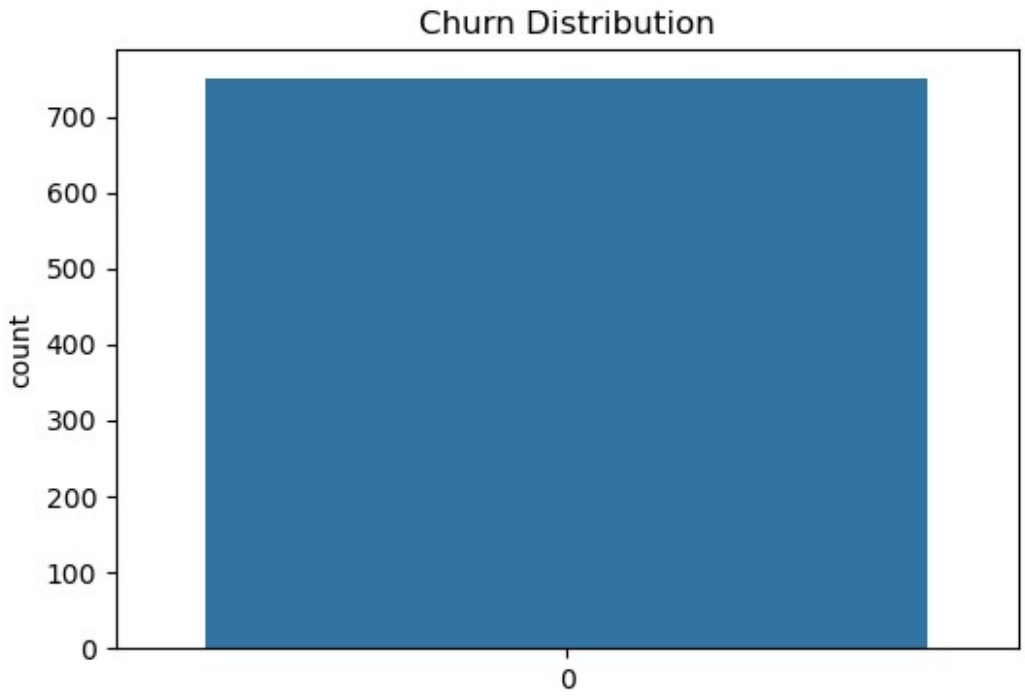
```
       total_intl_charge  number_customer_service_calls
count         750.000000                     750.000000
mean            2.779933                       1.634667
std             0.747704                       1.276207
min             0.000000                       0.000000
25%             2.305000                       1.000000
50%             2.780000                       1.000000
75%             3.270000                       2.000000
max             5.100000                       6.000000
```

```
# Pairplot for general overview
sns.pairplot(data, hue='total_day_minutes')  # Assuming
'international_plan' is the target column

# Correlation heatmap
correlation_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

# Distribution of Churn
plt.figure(figsize=(6, 4))
sns.countplot(data['total_day_minutes'])  # Assuming
'international_plan' is the churn indicator
plt.title("Churn Distribution")
plt.show()
```

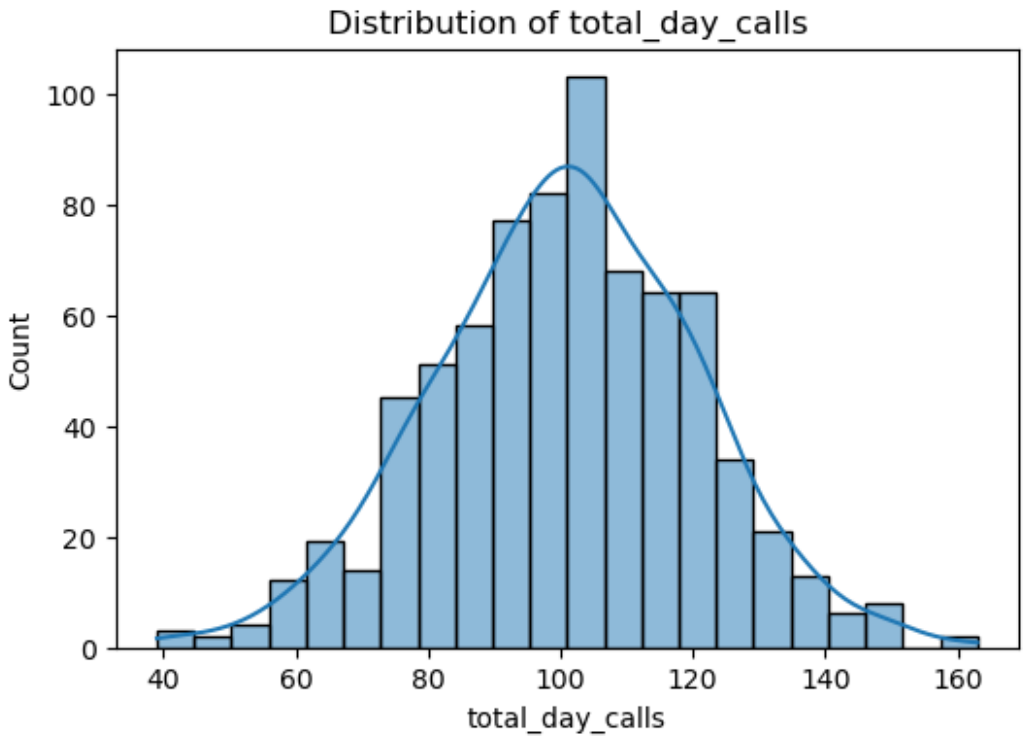
```
C:\Users\nishi\AppData\Local\Temp\ipykernel_2948\92574793.py:5:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this
warning.
  correlation_matrix = data.corr()
```

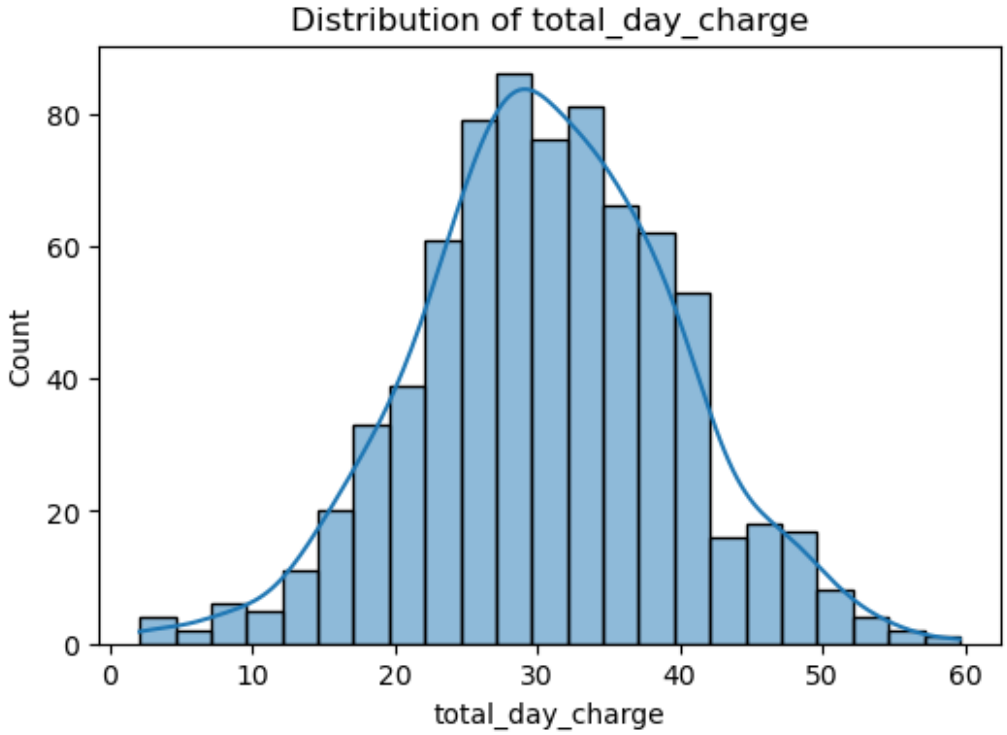

id
account_length
number_vmail_messages
total_day_calls
total_day_charge
total_eve_minutes
total_eve_calls
total_eve_charge
total_night_minutes
total_night_calls
total_night_charge
total_intl_minutes
total_intl_calls
total_intl_charge
number_customer_service_calls
total_day_minutes
60
120
180
240
300

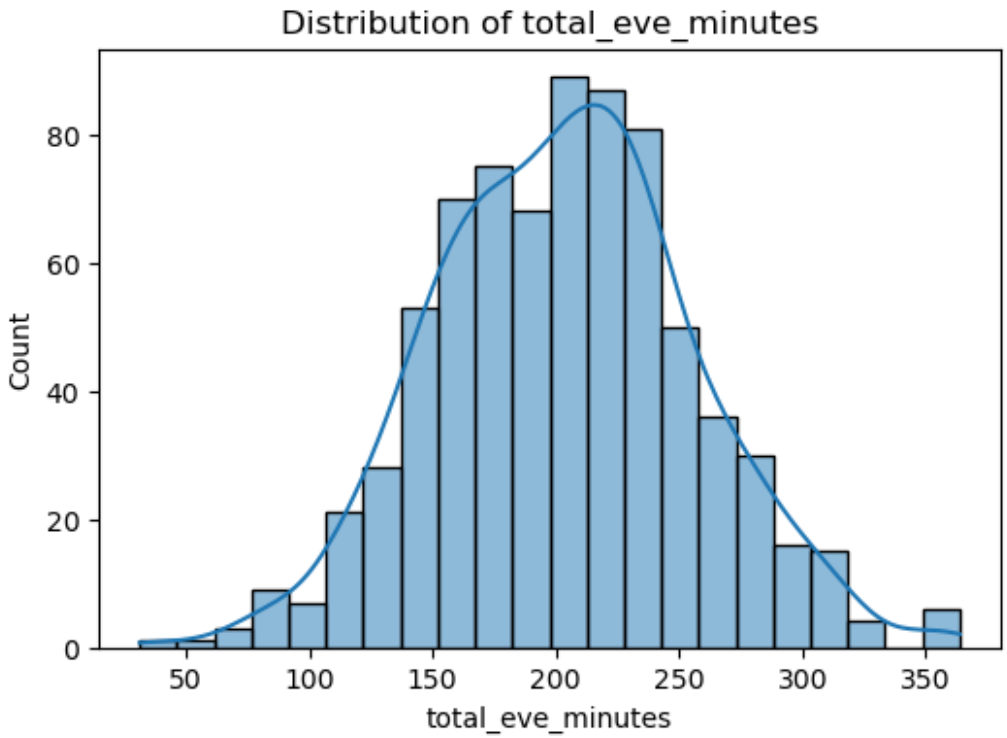

Correlation Heatmap
id 1 0.026 -0.069 0.016 -0.061 0.016 -0.018 0.04 -0.018 0.011 -0.049 0.011 -0.039 -0.038 -0.039 -0.022
account_length 0.026 1 -0.058 0.0024 0.058 0.0024 0.0058 0.03 -0.0058 0.059 -0.043 0.059 -0.017 0.018 -0.017 -0.016
number_vmail_messages -0.069 -0.058 1 0.024 0.041 0.024 0.062 -0.046 0.062 -0.061 0.0053 -0.061 -0.013 -0.037 -0.013 0.036
total_day_minutes 0.016 0.0024 0.024 1 0.0087 1 0.0029 0.021 0.0029 0.022 0.059 0.022 -0.0087 -0.032 -0.0087 0.036
total_day_calls -0.061 0.058 0.041 0.0087 1 0.0087 -0.054 0.0037 -0.054 0.0068 -0.03 0.0068 0.037 0.019 0.037 0.017
total_day_charge 0.016 0.0024 0.024 1 0.0087 1 0.0029 0.021 0.0029 0.022 0.059 0.022 -0.0087 -0.032 -0.0087 0.036
total_eve_minutes -0.018 -0.0058 0.062 0.0029 -0.054 0.0029 1 0.00062 1 -0.03 0.02 -0.03 0.019 -0.014 0.019 -0.038
total_eve_calls 0.04 0.03 -0.046 0.021 0.0037 0.021 0.00062 1 0.00062 -0.036 -0.025 -0.036 0.025 0.0092 0.025 0.0017
total_eve_charge -0.018 -0.0058 0.062 0.0029 -0.054 0.0029 1 0.00062 1 -0.03 0.02 -0.03 0.019 -0.014 0.019 -0.038
total_night_minutes 0.011 0.059 -0.061 0.022 0.0068 0.022 -0.03 -0.036 -0.03 1 0.046 1 -0.043 0.023 -0.043 0.022
total_night_calls -0.049 -0.043 0.0053 0.059 -0.03 0.059 0.02 -0.025 0.02 0.046 1 0.046 -0.0041 -0.025 -0.004 -0.006
total_night_charge 0.011 0.059 -0.061 0.022 0.0068 0.022 -0.03 -0.036 -0.03 1 0.046 1 -0.043 0.023 -0.043 0.022
total_intl_minutes -0.039 -0.017 -0.013 -0.0087 0.037 -0.0087 0.019 0.025 0.019 -0.043 -0.0041 -0.043 1 0.0019 1 -0.002
total_intl_calls -0.038 0.018 -0.037 -0.032 0.019 -0.032 -0.014 0.0092 -0.014 0.023 -0.025 0.023 0.0019 1 0.0022 -0.043
total_intl_charge -0.039 -0.017 -0.013 -0.0087 0.037 -0.0087 0.019 0.025 0.019 -0.043 -0.004 -0.043 1 0.0022 1 -0.002
number_customer_service_calls -0.022 -0.016 0.036 0.036 0.017 0.036 -0.038 0.0017 -0.038 0.022 -0.006 0.022 -0.002 -0.043 -0.002 1
id
account_length
number_vmail_messages
total_day_minutes
total_day_calls
total_day_charge
total_eve_minutes
total_eve_calls
total_eve_charge
total_night_minutes
total_night_calls
total_night_charge
total_intl_minutes
total_intl_calls
total_intl_charge
number_customer_service_calls
1.0
0.8
0.6
0.4
0.2
0.0



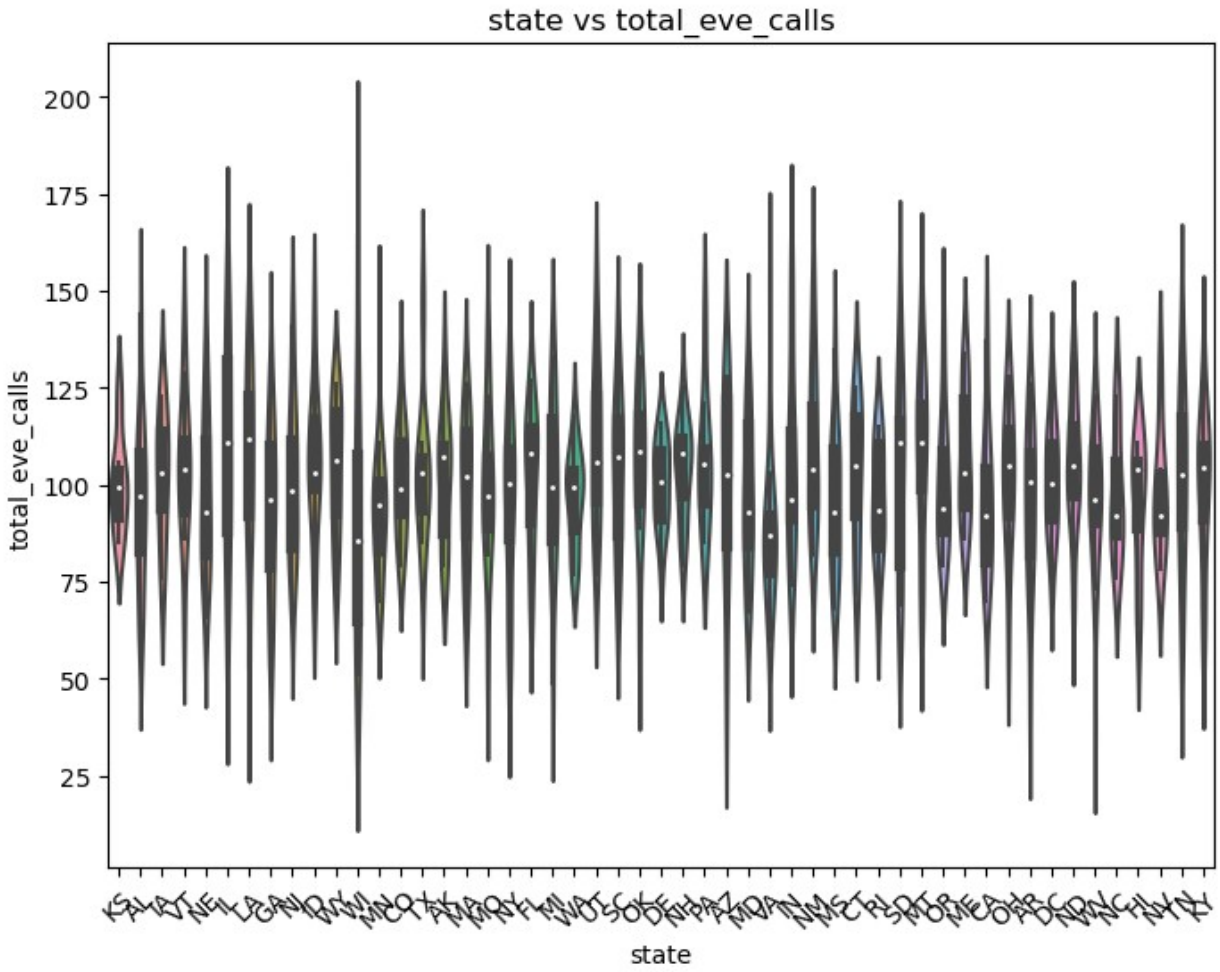

```
numerical_features = ['total_day_calls', 'total_day_charge',
'total_eve_minutes']  # Update with your features
for feature in numerical_features:
    plt.figure(figsize=(6, 4))
    sns.histplot(data[feature], kde=True)
    plt.title(f"Distribution of {feature}")
    plt.show()
```

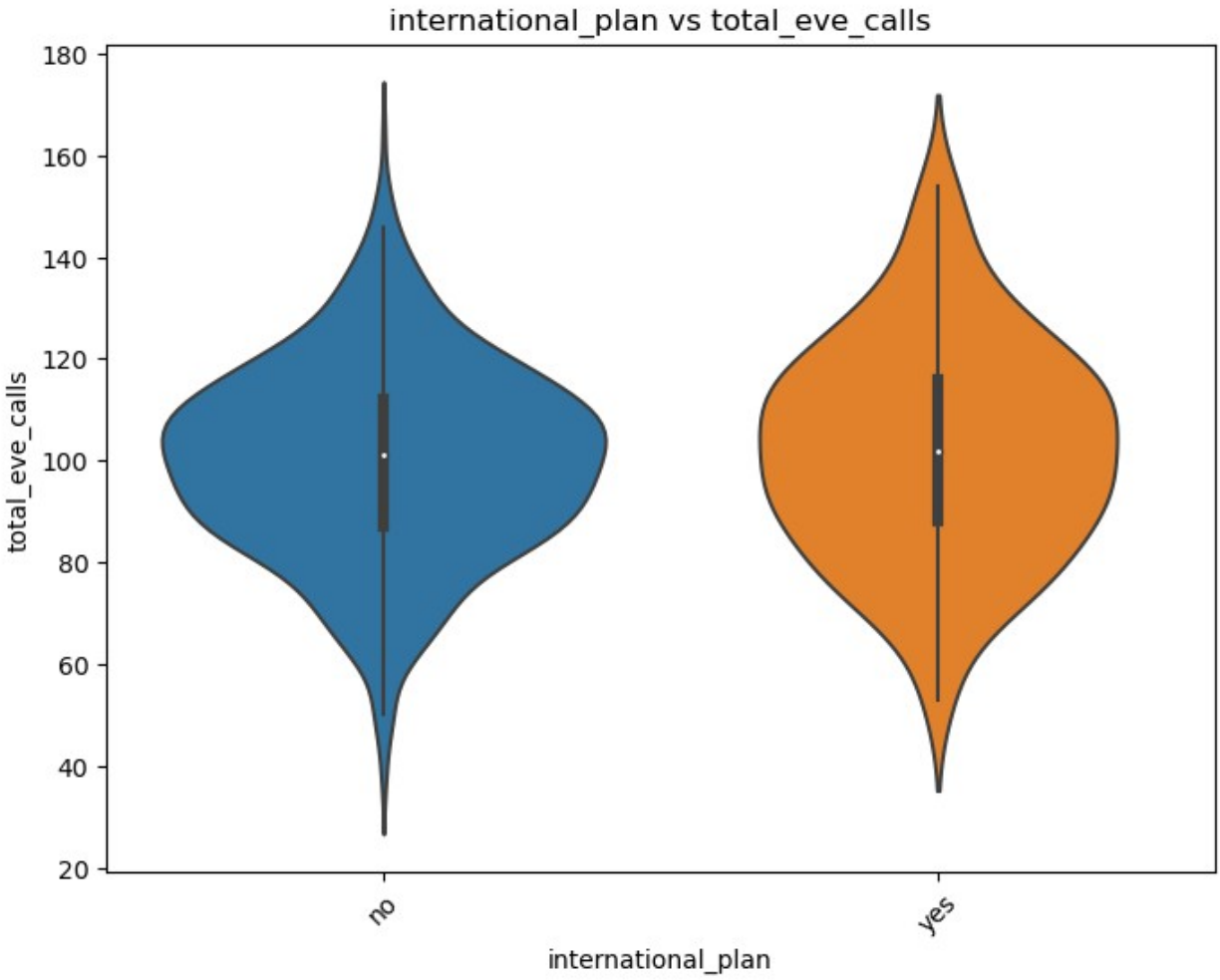

Distribution of total_day_calls
Count
total_day_calls
0
20
40
60
80
100
40
60
80
100
120
140
160

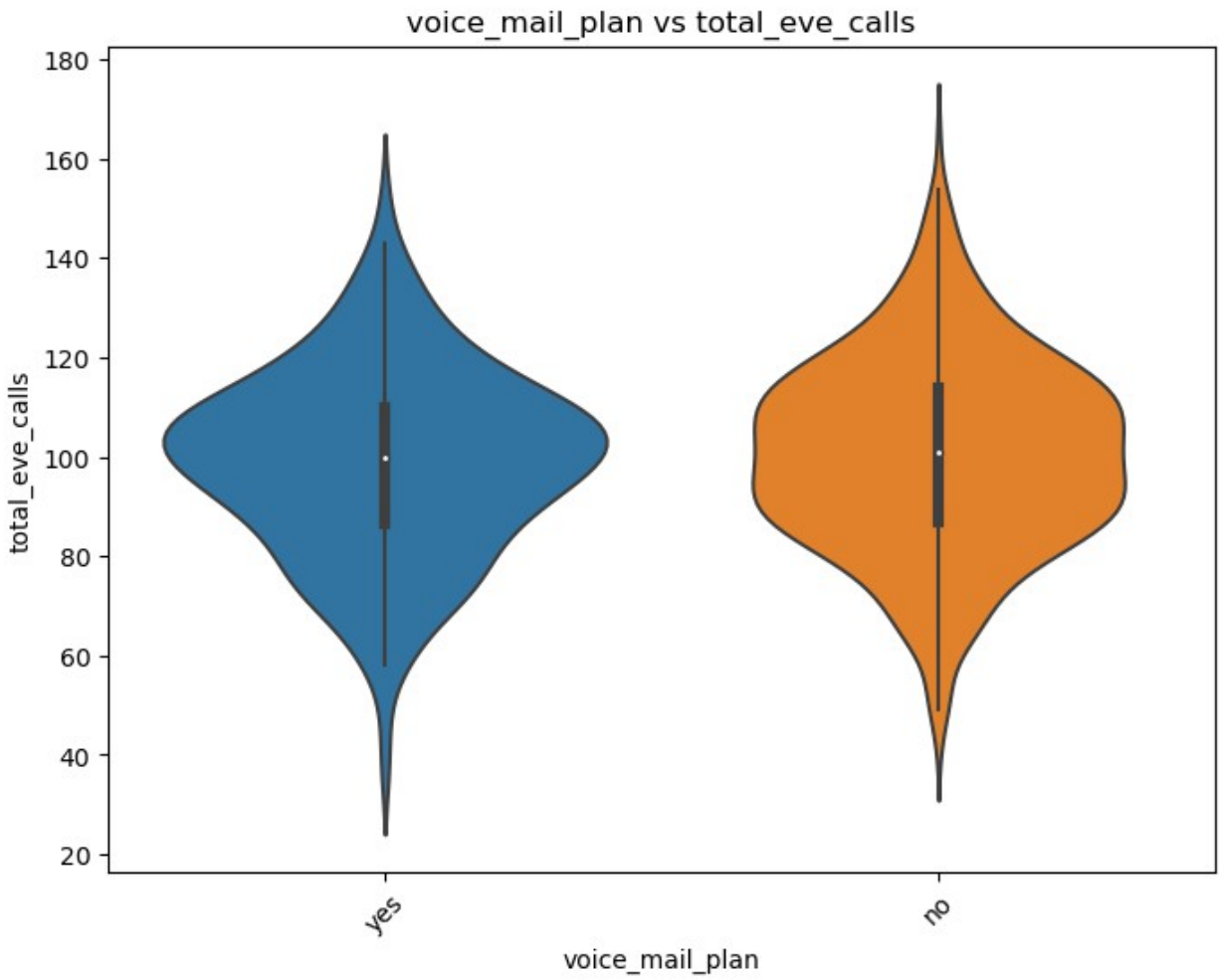

Distribution of total_day_charge
Count
total_day_charge
0
20
40
60
80
0
10
20
30
40
50
60




```
categorical_features = ['state', 'international_plan',
'voice_mail_plan']  # Update with your features
for feature in categorical_features:
    plt.figure(figsize=(8, 6))
    sns.violinplot(x=feature, y='total_eve_calls', data=data)
    plt.title(f"{feature} vs total_eve_calls")
    plt.xticks(rotation=45)
    plt.show()
```


state vs total_eve_calls
total_eve_calls
200
175
150
125
100
75
50
25
state


international_plan vs total_eve_calls
total_eve_calls
180
160
140
120
100
80
60
40
20
no
yes
international_plan




```
print(data.head())
```

```
   id state  account_length      area_code international_plan
voice_mail_plan  \
0   1    KS             128  area_code_415                 no
yes
1   2    AL             118  area_code_510                yes
no
2   3    IA              62  area_code_415                 no
no
3   4    VT              93  area_code_510                 no
no
4   5    NE             174  area_code_415                 no
no

   number_vmail_messages  total_day_minutes  total_day_calls  \
0                     25              265.1              110
1                      0              223.4               98
2                      0              120.7               70
3                      0              190.7              114
```

```
4                       0              124.3                  76

   total_day_charge  total_eve_minutes  total_eve_calls
total_eve_charge  \
0             45.07              197.4               99
16.78
1             37.98              220.6              101
18.75
2             20.52              307.2               76
26.11
3             32.42              218.2              111
18.55
4             21.13              277.1              112
23.55

   total_night_minutes  total_night_calls  total_night_charge  \
0                244.7                 91               11.01
1                203.9                118                9.18
2                203.0                 99                9.14
3                129.6                121                5.83
4                250.7                115               11.28

   total_intl_minutes  total_intl_calls  total_intl_charge  \
0                10.0                 3               2.70
1                 6.3                 6               1.70
2                13.1                 6               3.54
3                 8.1                 3               2.19
4                15.5                 5               4.19

   number_customer_service_calls
0                              1
1                              0
2                              4
3                              3
4                              3
```

```python
import pandas as pd

# Assuming you have a DataFrame named 'df' and want to delete columns
'column1' and 'column2'
columns_to_delete = ['area_code','state']

# Use the drop method to delete the specified columns
data.drop(columns=columns_to_delete, inplace=True)

# The 'inplace=True' argument modifies the DataFrame in place, so you
don't need to reassign it.

print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              750 non-null    int64  
 1   account_length                  750 non-null    int64  
 2   number_vmail_messages           750 non-null    int64  
 3   total_day_minutes               750 non-null    float64
 4   total_day_calls                 750 non-null    int64  
 5   total_day_charge                750 non-null    float64
 6   total_eve_minutes               750 non-null    float64
 7   total_eve_calls                 750 non-null    int64  
 8   total_eve_charge                750 non-null    float64
 9   total_night_minutes             750 non-null    float64
 10  total_night_calls               750 non-null    int64  
 11  total_night_charge              750 non-null    float64
 12  total_intl_minutes              750 non-null    float64
 13  total_intl_calls                750 non-null    int64  
 14  total_intl_charge               750 non-null    float64
 15  number_customer_service_calls  750 non-null    int64  
dtypes: float64(8), int64(8)
memory usage: 93.9 KB
None
```

```
print(data.isnull().sum())
# Depending on the columns with missing values, you can use methods
like imputation.
```

```
id                              0
state                           0
account_length                  0
area_code                       0
number_vmail_messages           0
total_day_minutes               0
total_day_calls                 0
total_day_charge                0
total_eve_minutes               0
total_eve_calls                 0
total_eve_charge                0
total_night_minutes             0
total_night_calls               0
total_night_charge              0
total_intl_minutes              0
total_intl_calls                0
total_intl_charge               0
number_customer_service_calls    0
dtype: int64
```

```
from sklearn.model_selection import train_test_split

X = data.drop(columns=["total_intl_calls"])
y = data["total_intl_calls"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print(X.shape, X_train.shape, X_test.shape)
```

```
(750, 15) (600, 15) (150, 15)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train[numerical_features] =
scaler.fit_transform(X_train[numerical_features])
X_test[numerical_features] =
scaler.transform(X_test[numerical_features])

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)# Import LogisticRegression
#from sklearn.tree import DecisionTreeClassifier
#from sklearn.ensemble import RandomForestClassifier
#from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
# Preprocess the categorical variables using one-hot encoding
#data = pd.get_dummies(data, columns=["Geography", "Gender"])

# Split into features (X) and target (y)
X = data.drop("number_customer_service_calls", axis=1)
y = data["number_customer_service_calls"]

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    #"Decision Tree": DecisionTreeClassifier(),
    #"Random Forest": RandomForestClassifier(),
    #"Support Vector Machine": SVC()
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted',
zero_division=0)  # Specify 'weighted' for multiclass
    recall = recall_score(y_test, y_pred, average='weighted')  #
Specify 'weighted' for multiclass
    f1 = f1_score(y_test, y_pred, average='weighted')  # Specify
'weighted' for multiclass

    results[name] = {"Accuracy": accuracy, "Precision": precision,
"Recall": recall, "F1": f1}

# Display evaluation results
print("Model Evaluation Results:")
for name, metrics in results.items():
    print(f"Model: {name}")
    print(f"Accuracy: {metrics['Accuracy']:.2f}")
    print(f"Precision: {metrics['Precision']:.2f}")
    print(f"Recall: {metrics['Recall']:.2f}")
    print(f"F1 Score: {metrics['F1']:.2f}")
    print()
```

```
Model Evaluation Results:
Model: Logistic Regression
Accuracy: 0.34
Precision: 0.17
Recall: 0.34
F1 Score: 0.21


C:\Users\nishi\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:460: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
print(data.head())
```

```
   id  account_length  number_vmail_messages  total_day_minutes  \
0   1             128                     25              265.1
1   2             118                      0              223.4
2   3              62                      0              120.7
```

```
3   4              93                       0              190.7
4   5             174                       0              124.3

   total_day_calls  total_day_charge  total_eve_minutes
total_eve_calls  \
0              110             45.07              197.4
99
1               98             37.98              220.6
101
2               70             20.52              307.2
76
3              114             32.42              218.2
111
4               76             21.13              277.1
112

   total_eve_charge  total_night_minutes  total_night_calls  \
0             16.78                244.7                 91
1             18.75                203.9                118
2             26.11                203.0                 99
3             18.55                129.6                121
4             23.55                250.7                115

   total_night_charge  total_intl_minutes  total_intl_calls  \
0               11.01                10.0                 3
1                9.18                 6.3                 6
2                9.14                13.1                 6
3                5.83                 8.1                 3
4               11.28                15.5                 5

   total_intl_charge  number_customer_service_calls
0               2.70                              1
1               1.70                              0
2               3.54                              4
3               2.19                              3
4               4.19                              3
```

```
best_model = max(results, key=lambda k: results[k]["F1"])
print("Best Performing Model:", best_model)
```

```
Best Performing Model: Logistic Regression
```

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Input data
input_data = (175, 258, 587, 123, 456, 258, 236, 147, 159, 368, 157,
147, 142, 124, 152)

# Changing the input_data to a numpy array
```

```
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)

# Assuming you have a DataFrame named 'data' and want to predict
'number_customer_service_calls'
# Replace 'X' and 'y' with your actual features and target variable
X = data.drop("number_customer_service_calls", axis=1)
y = data["number_customer_service_calls"]

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

# Create and train a Logistic Regression model# Create and train a
Logistic Regression model with increased max_iter
logistic_regression = LogisticRegression(max_iter=1000)
logistic_regression.fit(X_train, y_train)


# Now, you can make predictions with the trained model
prediction = logistic_regression.predict(input_data_reshaped)

if prediction[0] == 0:
    print("Churn: No")
else:
    print("Churn: Yes")
```

```
Churn: Yes

C:\Users\nishi\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:460: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\nishi\anaconda3\Lib\site-packages\sklearn\base.py:464:
UserWarning: X does not have valid feature names, but
LogisticRegression was fitted with feature names
  warnings.warn(
```