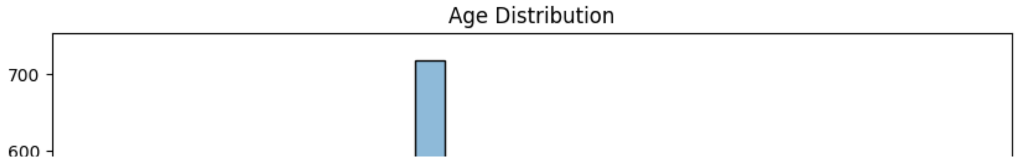
```
!pip install scikit-learn
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score

# Load the datasets
train_data = pd.read_csv('/content/Churn_TRAIN.csv')
test_data = pd.read_csv('/content/Churn_TEST.csv')

# Split the data into input features (X) and target variable (y)
X_train = train_data.drop('Churn', axis=1)
y_train = train_data['Churn']
X_test = test_data.drop('Churn', axis=1)
y_test = test_data['Churn']

# Train a random forest classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf.predict(X_test)

# Compute and print the classification report
print(classification_report(y_test, y_pred_rf))

# Compute and print the ROC AUC
roc_auc_rf = roc_auc_score(y_test, y_pred_rf)
print('ROC AUC (Random Forest):', roc_auc_rf)
```

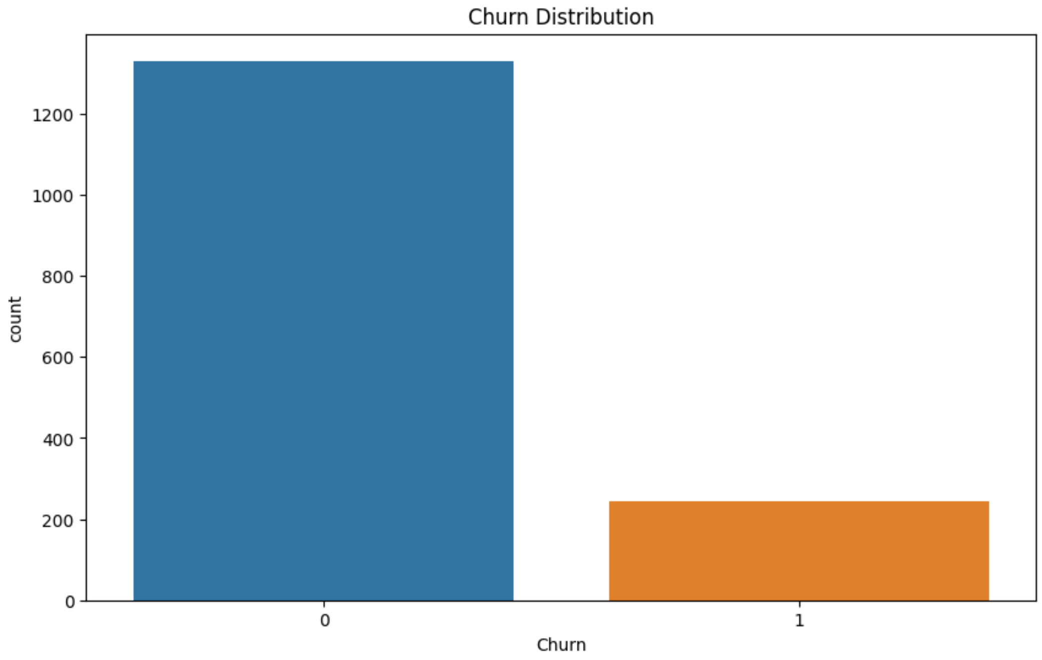
```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.1.0)
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1324
           1       0.89      0.78      0.83       251

    accuracy                           0.95      1575
   macro avg       0.92      0.88      0.90      1575
weighted avg       0.95      0.95      0.95      1575

ROC AUC (Random Forest): 0.8829891912711691
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(data=train_data, x='Age', bins=30, kde=True)
plt.title('Age Distribution')
plt.show()
```



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score
```

```
plt.figure(figsize=(10, 6))
sns.countplot(x='Churn', data=train_data)
plt.title('Churn Distribution')
plt.show()
```



```
X_train = train_data.drop('Churn', axis=1)
y_train = train_data['Churn']
X_test = test_data.drop('Churn', axis=1)
y_test = test_data['Churn']

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
print('ROC AUC:', roc_auc_score(y_test, y_pred))
```

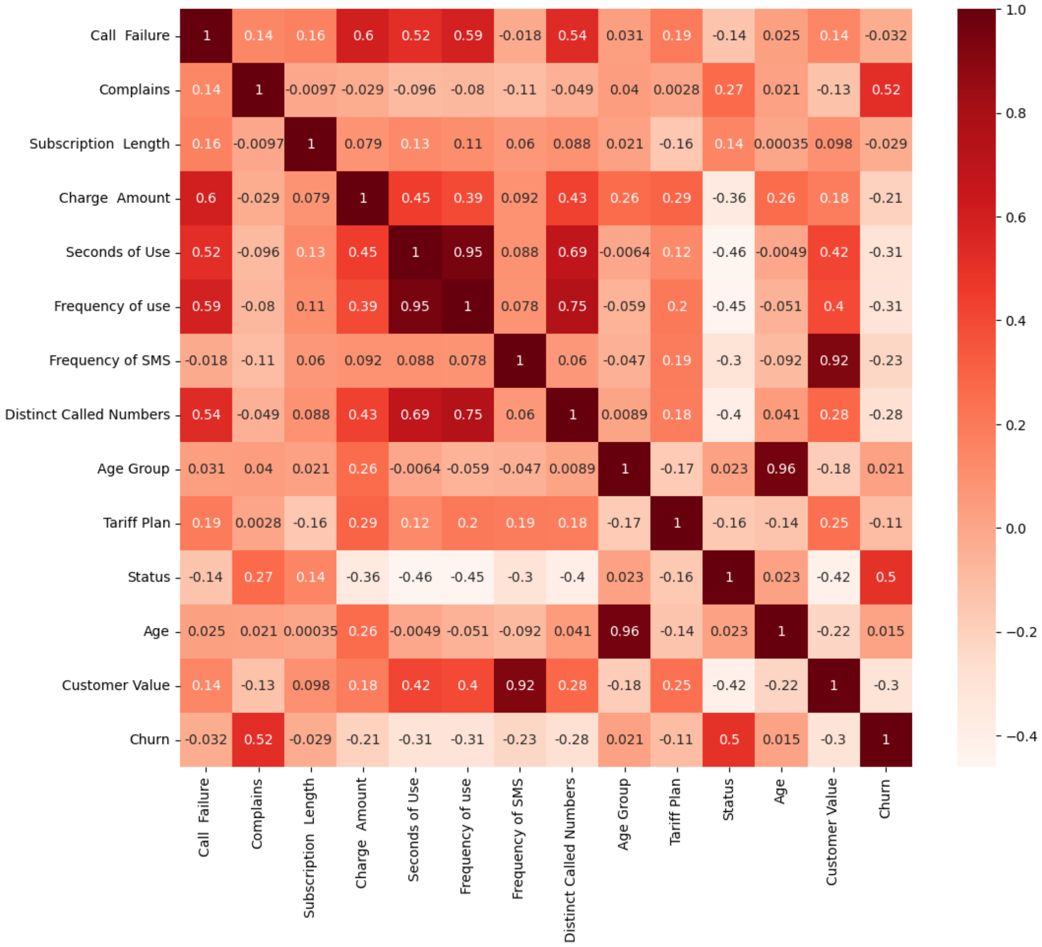
```
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      1324
           1       0.76      0.43      0.55       251

    accuracy                           0.89      1575
   macro avg       0.83      0.70      0.74      1575
weighted avg       0.88      0.89      0.87      1575

ROC AUC: 0.7022995630769973
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
from sklearn.model_selection import cross_val_score


# Preprocess your data
X_train = train_data.drop('Churn', axis=1)
y_train = train_data['Churn']
X_test = test_data.drop('Churn', axis=1)
y_test = test_data['Churn']

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)



# Train and evaluate different models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"Model: {name}")
    print(classification_report(y_test, y_pred))
    print('ROC AUC:', roc_auc_score(y_test, y_pred))
    print("------------------------")
```

```
Model: Logistic Regression
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      1324
           1       0.76      0.43      0.55       251

    accuracy                           0.89      1575
   macro avg       0.83      0.70      0.74      1575
weighted avg       0.88      0.89      0.87      1575

ROC AUC: 0.7022995630769973
------------------------
Model: Random Forest
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1324
           1       0.88      0.79      0.83       251

    accuracy                           0.95      1575
   macro avg       0.92      0.89      0.90      1575
weighted avg       0.95      0.95      0.95      1575

ROC AUC: 0.8858403245025938
------------------------
Model: Support Vector Machine
              precision    recall  f1-score   support

           0       0.92      0.99      0.95      1324
           1       0.94      0.54      0.68       251

    accuracy                           0.92      1575
   macro avg       0.93      0.77      0.82      1575
weighted avg       0.92      0.92      0.91      1575

ROC AUC: 0.7655255112480591
------------------------
```

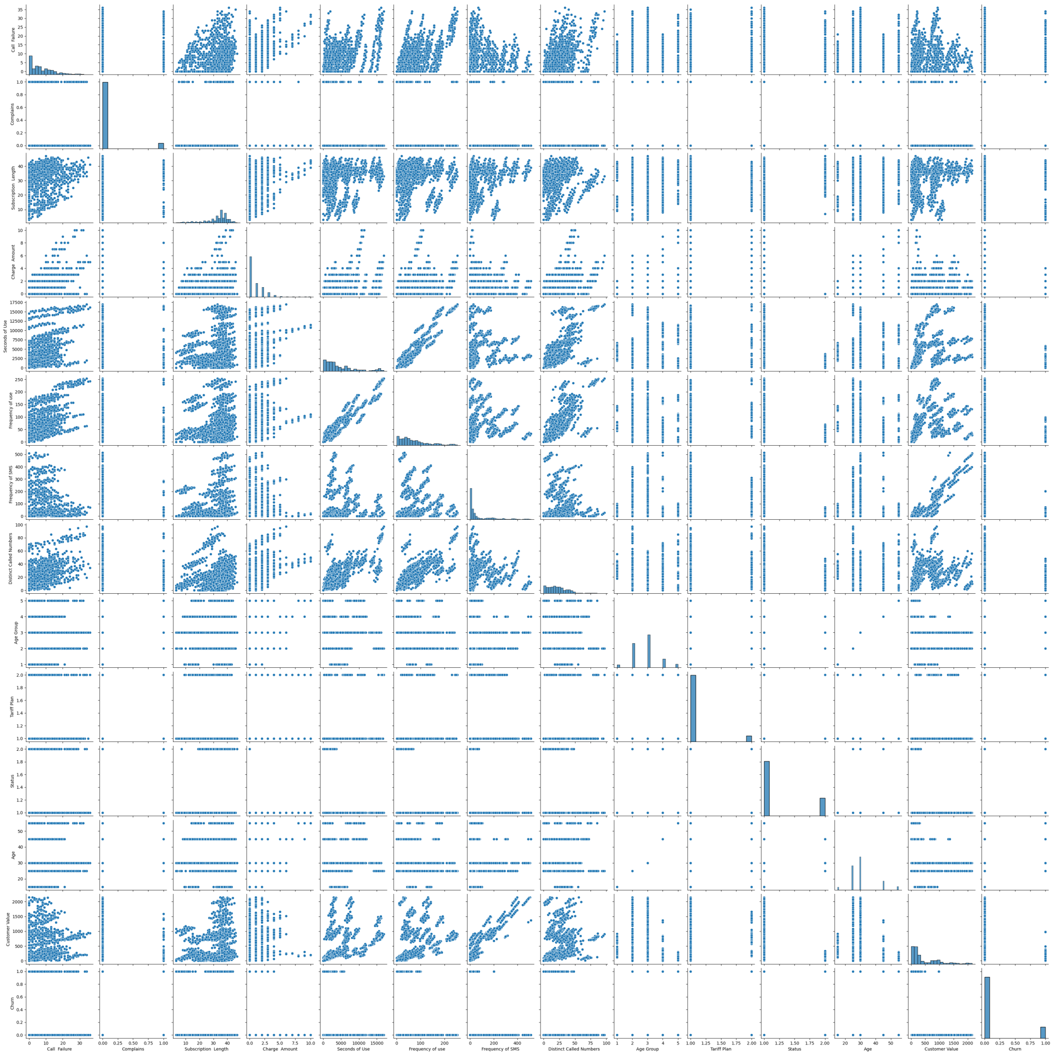
```
# Cross-validation
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='roc_auc')
    print(f"Model: {name}, ROC AUC: {scores.mean()}")

# Confusion Matrix
for name, model in models.items():
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt="d")
    plt.title(f"Confusion matrix for {name}")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

```
plt.figure(figsize=(12,10))
corr = train_data.corr()
sns.heatmap(corr, annot=True, cmap=plt.cm.Reds)
plt.show()
```

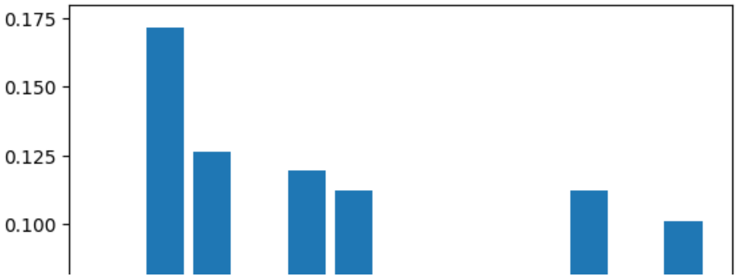


	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status	Age	Customer Value	Churn
Call Failure	1	0.14	0.16	0.6	0.52	0.59	-0.018	0.54	0.031	0.19	-0.14	0.025	0.14	-0.032
Complains	0.14	1	-0.0097	-0.029	-0.096	-0.08	-0.11	-0.049	0.04	0.0028	0.27	0.021	-0.13	0.52
Subscription Length	0.16	-0.0097	1	0.079	0.13	0.11	0.06	0.088	0.021	-0.16	0.14	0.00035	0.098	-0.029
Charge Amount	0.6	-0.029	0.079	1	0.45	0.39	0.092	0.43	0.26	0.29	-0.36	0.26	0.18	-0.21
Seconds of Use	0.52	-0.096	0.13	0.45	1	0.95	0.088	0.69	-0.0064	0.12	-0.46	-0.0049	0.42	-0.31
Frequency of use	0.59	-0.08	0.11	0.39	0.95	1	0.078	0.75	-0.059	0.2	-0.45	-0.051	0.4	-0.31
Frequency of SMS	-0.018	-0.11	0.06	0.092	0.088	0.078	1	0.06	-0.047	0.19	-0.3	-0.092	0.92	-0.23
Distinct Called Numbers	0.54	-0.049	0.088	0.43	0.69	0.75	0.06	1	0.0089	0.18	-0.4	0.041	0.28	-0.28
Age Group	0.031	0.04	0.021	0.26	-0.0064	-0.059	-0.047	0.0089	1	-0.17	0.023	0.96	-0.18	0.021
Tariff Plan	0.19	0.0028	-0.16	0.29	0.12	0.2	0.19	0.18	-0.17	1	-0.16	-0.14	0.25	-0.11
Status	-0.14	0.27	0.14	-0.36	-0.46	-0.45	-0.3	-0.4	0.023	-0.16	1	0.023	-0.42	0.5
Age	0.025	0.021	0.00035	0.26	-0.0049	-0.051	-0.092	0.041	0.96	-0.14	0.023	1	-0.22	0.015
Customer Value	0.14	-0.13	0.098	0.18	0.42	0.4	0.92	0.28	-0.18	0.25	-0.42	-0.22	1	-0.3
Churn	-0.032	0.52	-0.029	-0.21	-0.31	-0.31	-0.23	-0.28	0.021	-0.11	0.5	0.015	-0.3	1

```
sns.pairplot(train_data)
plt.show()
```



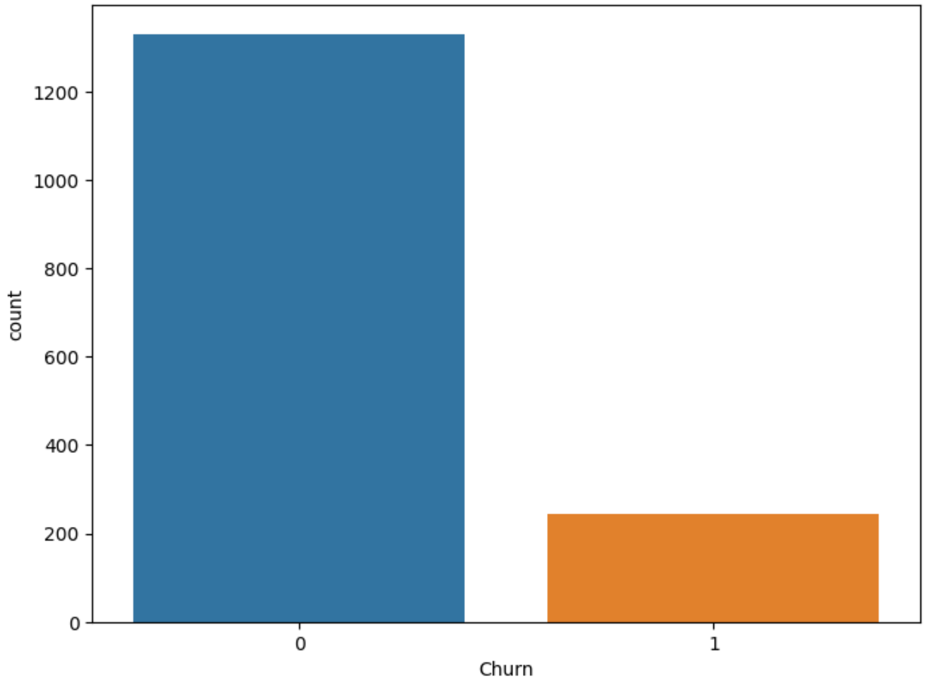
```
model = RandomForestClassifier()
model.fit(X_train, y_train)
importance = model.feature_importances_
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

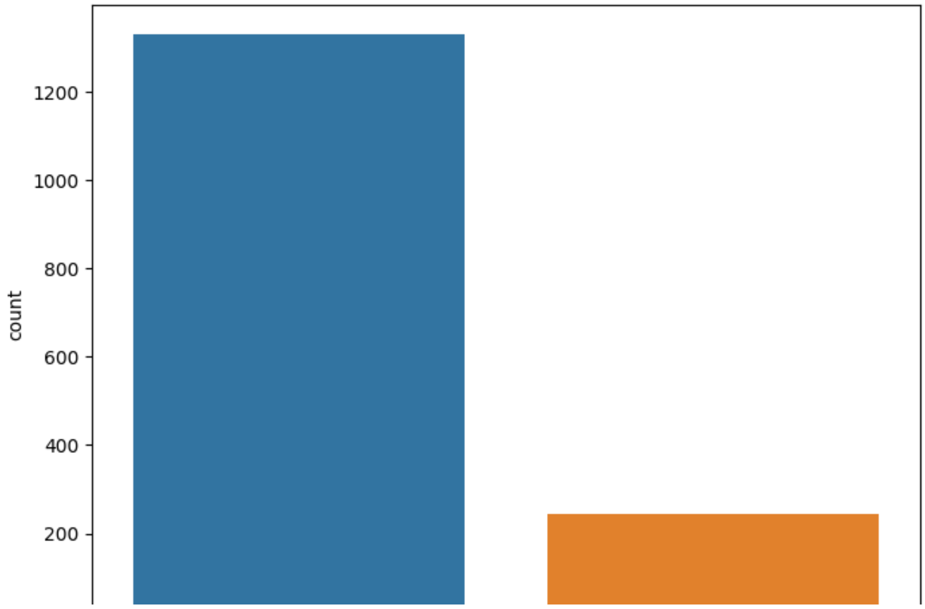


```
print(train_data.describe())
print(test_data.describe())
```

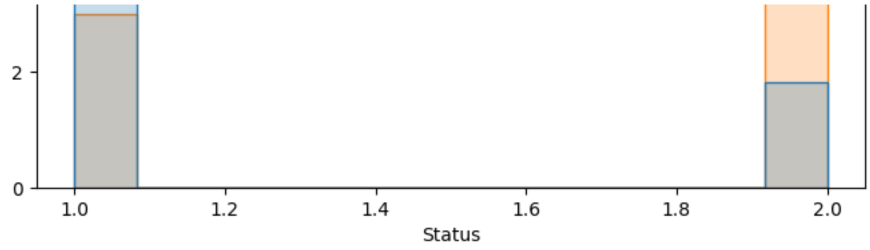
```
       Distinct Called Numbers    Age Group  Tariff Plan       Status  \
count              1575.000000  1575.000000  1575.000000  1575.000000   
mean                 23.537143     2.806349     1.079365     1.245079   
std                  17.718890     0.873853     0.270394     0.430271   
min                   0.000000     1.000000     1.000000     1.000000   
25%                  10.000000     2.000000     1.000000     1.000000   
50%                  21.000000     3.000000     1.000000     1.000000   
75%                  33.000000     3.000000     1.000000     1.000000   
max                  97.000000     5.000000     2.000000     2.000000   

               Age  Customer Value        Churn  
count  1575.000000     1575.000000  1575.000000  
mean     30.780952      480.649708     0.154921  
std       8.586189      515.837852     0.361944  
min      15.000000        0.000000     0.000000  
25%      25.000000      114.412500     0.000000  
50%      30.000000      237.915000     0.000000  
75%      30.000000      807.727500     0.000000  
max      55.000000     2149.280000     1.000000  
       Call  Failure    Complains  Subscription  Length  Charge  Amount  \
count     1575.000000  1575.000000           1575.000000     1575.000000   
mean         7.589841     0.075556             32.845079        0.944127   
std          7.213174     0.264370              8.494867        1.557268   
min          0.000000     0.000000              3.000000        0.000000   
25%          1.000000     0.000000             30.000000        0.000000   
50%          6.000000     0.000000             35.000000        0.000000   
75%         11.500000     0.000000             38.000000        1.000000   
max         36.000000     1.000000             46.000000       10.000000   

       Seconds of Use  Frequency of use  Frequency of SMS  \
count     1575.000000       1575.000000       1575.000000   
mean      4378.377778         68.247619         72.452063   
std       4078.714808         55.752091        113.154851   
min          0.000000          0.000000          0.000000   
25%       1366.500000         27.000000          7.000000   
50%       2970.000000         54.000000         20.000000   
75%       6480.000000         94.000000         81.500000   
max      17090.000000        255.000000        522.000000   

       Distinct Called Numbers    Age Group  Tariff Plan       Status  \
count              1575.000000  1575.000000  1575.000000  1575.000000   
mean                 23.482540     2.845714     1.076190     1.251429   
std                  16.706322     0.910726     0.265387     0.433972   
min                   0.000000     1.000000     1.000000     1.000000   
25%                  10.000000     2.000000     1.000000     1.000000   
50%                  21.000000     3.000000     1.000000     1.000000   
75%                  34.000000     3.000000     1.000000     2.000000   
max                  88.000000     5.000000     2.000000     2.000000   

               Age  Customer Value        Churn  
count  1575.000000     1575.000000  1575.000000  
mean     31.215873      461.296124     0.159365  
std       9.066905      518.173380     0.366132  
min      15.000000        0.000000     0.000000  
25%      25.000000      111.760000     0.000000  
50%      30.000000      221.715000     0.000000  
75%      30.000000      770.715000     0.000000  
max      55.000000     2165.280000     1.000000  
```

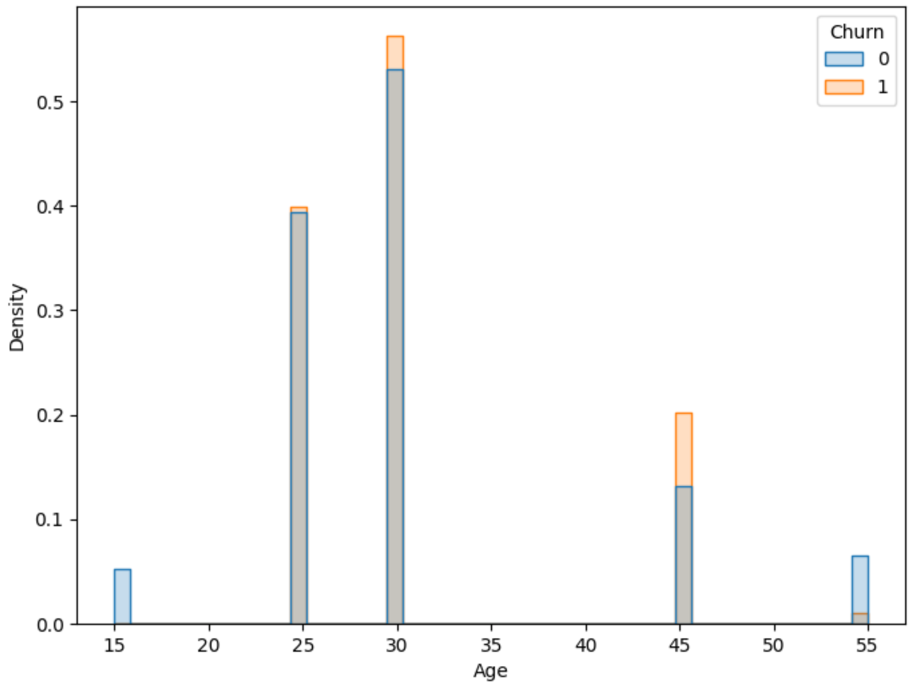
```
plt.figure(figsize=(8,6))
sns.countplot(x='Churn', data=train_data)
plt.show()
plt.figure(figsize=(8,6))
sns.countplot(x='Churn', data=train_data)
plt.show()
```

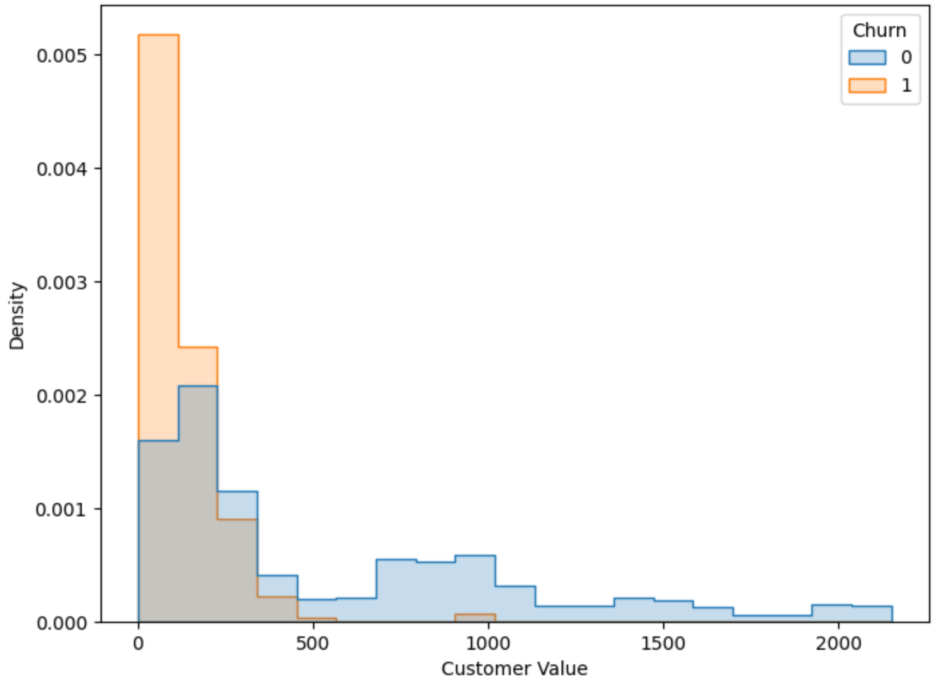




```
for column in train_data.columns:
    if column != 'Churn':
        plt.figure(figsize=(8,6))
        sns.histplot(data=train_data, x=column, hue='Churn', element='step', stat='density', common_norm=False)
        plt.show()
```


2
0
1.0
1.2
1.4
1.6
1.8
2.0
Status


Churn
0
1
Density
0.5
0.4
0.3
0.2
0.1
0.0
15
20
25
30
35
40
45
50
55
Age


Churn
0
1
Density
0.005
0.004
0.003
0.002
0.001
0.000
0
500
1000
1500
2000
Customer Value

```
from sklearn.model_selection import cross_val_score

for name, model in models.items():
    cv_scores = cross_val_score(model, X_train, y_train, cv=5)
    print(f"{name} Cross Validation Score: {cv_scores.mean()}")
```

```
Logistic Regression Cross Validation Score: 0.8920634920634921
Random Forest Cross Validation Score: 0.947936507936508
Support Vector Machine Cross Validation Score: 0.9117460317460317
```

```
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd


# Prepare the training data
X_train = train_data.drop('Churn', axis=1)
y_train = train_data['Churn']

# Prepare the test data
X_test = test_data.drop('Churn', axis=1)
y_test = test_data['Churn']

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Print a classification report
print(classification_report(y_test, y_pred))

# Print a confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      1324
           1       0.76      0.43      0.55       251

    accuracy                           0.89      1575
   macro avg       0.83      0.70      0.74      1575
weighted avg       0.88      0.89      0.87      1575

[[1290   34]
 [ 143  108]]
```

```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc


# Prepare the training data
X_train = train_data.drop('Churn', axis=1)
y_train = train_data['Churn']

# Prepare the test data
X_test = test_data.drop('Churn', axis=1)
y_test = test_data['Churn']

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```