```
from google.colab import files
uploaded=files.upload()
```

Choose Files Telco-Custo…r-Churn.csv

- **Telco-Customer-Churn.csv**(text/csv) - 977501 bytes, last modified: 5/8/2025 - 100% done

```
Saving Telco-Customer-Churn.csv to Telco-Customer-Churn.csv
```

```
import pandas as pd
df=pd.read_csv("Telco-Customer-Churn.csv")
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServ
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber

5 rows × 21 columns

```
#Data Exploration
df.info()
df.describe()
df.columns
df.shape
```

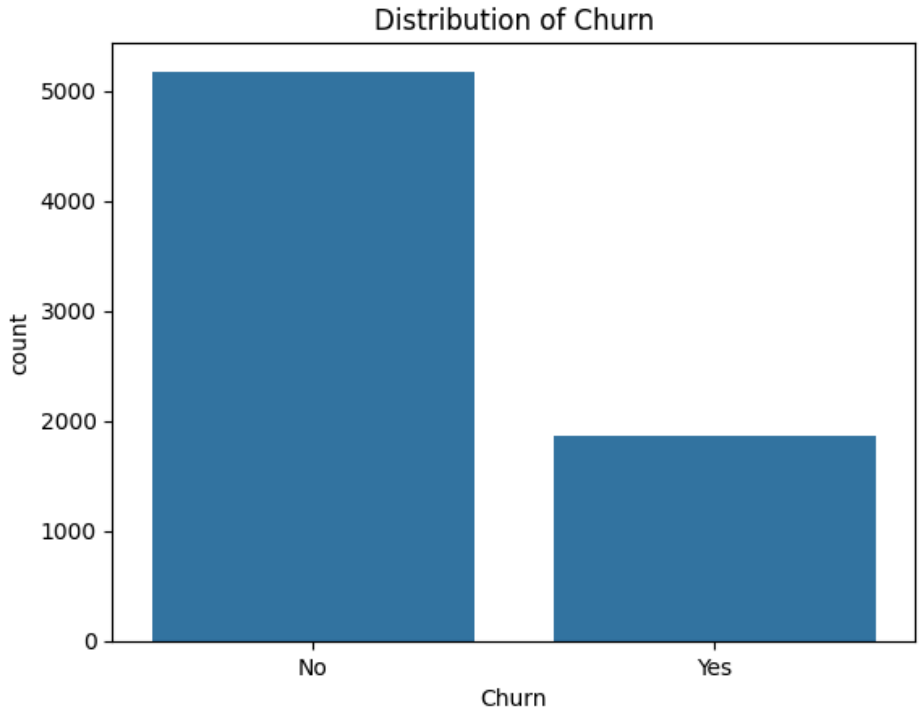
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
(7043, 21)
```

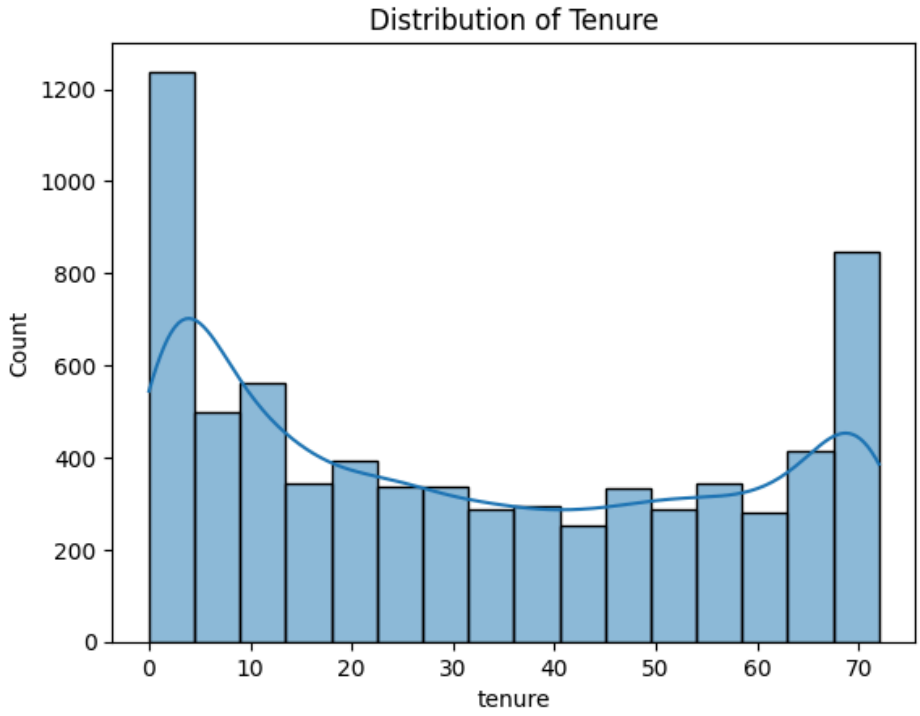
```
#checking missing value and duplicates
print(df.isnull().sum())
```

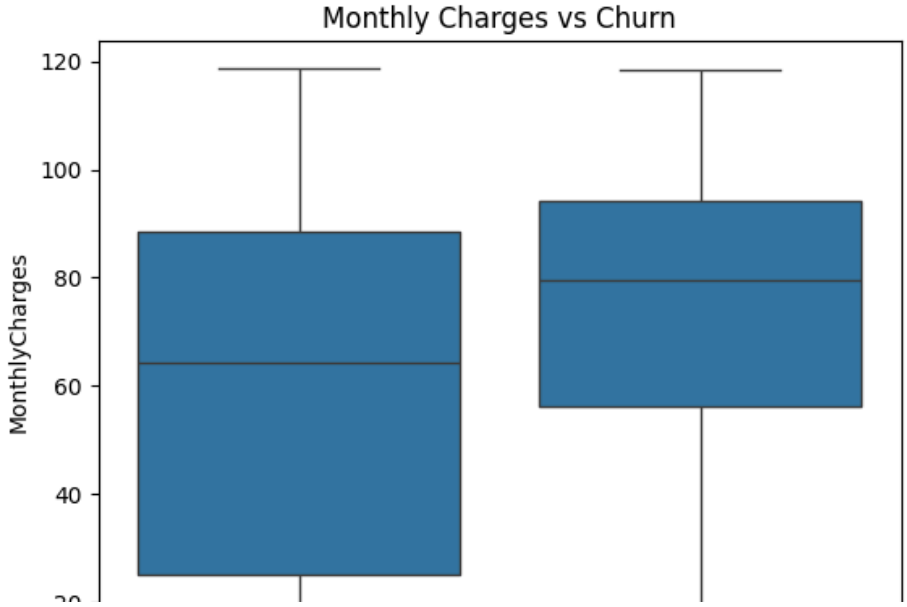
```
print(f"Duplicted Rows :{df.duplicated().sum()}")
```

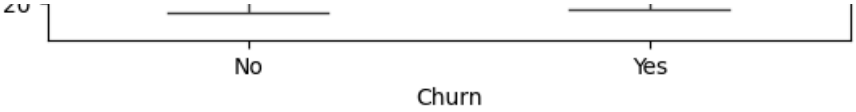
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
Duplicted Rows :0
```

```
#visualize features
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='Churn', data=df)
plt.title('Distribution of Churn')
plt.show()

sns.histplot(df['tenure'],kde=True)
plt.title('Distribution of Tenure')
plt.show()

sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
plt.title('Monthly Charges vs Churn')
plt.show()
```


Distribution of Churn
count
5000
4000
3000
2000
1000
0
No
Yes
Churn


Distribution of Tenure
Count
1200
1000
800
600
400
200
0
0
10
20
30
40
50
60
70
tenure


Monthly Charges vs Churn
MonthlyCharges
120
100
80
60
40




```
#identifing target and features
target='Churn'
features=df.drop(columns=[target]).columns.tolist()


#convert catgo to numeric

df['Totalcharges']=pd.to_numeric(df['TotalCharges'],errors='coerce')
df.dropna(inplace=True)


#one-hot encode
df_encoded=pd.get_dummies(df,drop_first=True)



#featue scaling
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaled_features=scaler.fit_transform(df_encoded.drop(columns=['Churn_Yes']))
X=pd.DataFrame(scaled_features,columns=df_encoded.drop(columns=['Churn_Yes']).columns)
y=df_encoded['Churn_Yes']



#train-test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)


#model building
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_model = RandomForestClassifier(class_weight='balanced')  # Automatically adjust weights for imbalanced data


# Logistic Regression
lr_model = LogisticRegression(max_iter=1000)  # Increased iterations for convergence
lr_model.fit(X_train,y_train)
lr_model = LogisticRegression(class_weight='balanced')  # Adjust class weight for logistic regression




#evalution
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Random Forest Evaluation
print("Random Forest Results:")
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)

print(confusion_matrix(y_test, rf_preds))
print(classification_report(y_test, rf_preds))
print("RF Accuracy:", accuracy_score(y_test, rf_preds))

# Logistic Regression Evaluation
print("\nLogistic Regression Results:")
lr_model.fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)
```

```
print(confusion_matrix(y_test, lr_preds))
print(classification_report(y_test, lr_preds))
print("LR Accuracy:", accuracy_score(y_test,lr_preds))
```

```
Random Forest Results:
[[925 108]
 [185 189]]
              precision    recall  f1-score   support

       False       0.83      0.90      0.86      1033
        True       0.64      0.51      0.56       374

    accuracy                           0.79      1407
   macro avg       0.73      0.70      0.71      1407
weighted avg       0.78      0.79      0.78      1407

RF Accuracy: 0.7917555081734187

Logistic Regression Results:
[[539 494]
 [ 40 334]]
              precision    recall  f1-score   support

       False       0.93      0.52      0.67      1033
        True       0.40      0.89      0.56       374

    accuracy                           0.62      1407
   macro avg       0.67      0.71      0.61      1407
weighted avg       0.79      0.62      0.64      1407

LR Accuracy: 0.6204690831556503
```

```
# Compare accuracies and choose the better model
if accuracy_score(y_test, rf_preds) > accuracy_score(y_test, lr_preds):
    model = rf_model
    print("Selected RF model")
else:
    model = lr_model
    print("Selected LR model")
```

```
Selected RF model
```

```
#prediction
new_data=X_test.iloc[0:1]
model.predict(new_data)
```

```
array([False])
```

```
import joblib
joblib.dump(rf_model,'churn_prediction_model.pkl')
joblib.dump(X.columns.tolist(),'columns.pkl')

files.download("churn_prediction_model.pkl")
files.download("columns.pkl")
```

```

```

```
code='''
import streamlit as st
import pandas as pd
import joblib

model=joblib.load('churn_prediction_model.pkl')
columns=joblib.load('columns.pkl')

st.title('Customer Churn Prediction')
```

```
st.title("Customer Churn Prediction")
st.write("Enter customer details to predict churn ")
gender = st.selectbox('Gender', ['Female', 'Male'])
senior_citizen = st.selectbox('Senior Citizen', ['No', 'Yes'])
partner = st.selectbox('Partner', ['No', 'Yes'])
dependents = st.selectbox('Dependents', ['No', 'Yes'])
tenure = st.slider('Tenure (months)', 0, 72)
monthly=st.number_input('Monthly Charges', min_value=0.0)
total=st.number_input('Total Charges', min_value=0.0)
phone_service = st.selectbox('Phone Service', ['No', 'Yes'])
multiple_lines = st.selectbox('Multiple Lines', ['No phone service', 'No', 'Yes'])
internet_service = st.selectbox('Internet Service', ['DSL', 'Fiber optic', 'No'])

input_data = {
  'gender': gender,
  'SeniorCitizen': senior_citizen,
  'partner': partner,
  'dependents': dependents,
  'tenure': tenure,
  'MonthlyCharges': monthly,
  'TotalCharges': total,
  'PhoneService': phone_service,
  'MultipleLines': multiple_lines,
  'InternetService': internet_service
}

def predict_chrun(data):
  df_input=pd.DataFrame([data])
  df_encoded=pd.get_dummies(df_input).reindex(columns=columns,fill_value=0)
  prediction=model.predict(df_encoded)
  return "Churn" if prediction[0]==1 else "No Churn"

if st.button('Predict'):
  result=predict_chrun(input_data)
  st.write(f'Prediction: {result}')
'''
with open("app.py","w") as f:
  f.write(code)

print("All Done. Download churn_prediction_model.pkl, columns.pkl, and app.py files to run the Streamlit app loca
```

```
All Done. Download churn_prediction_model.pkl, columns.pkl, and app.py files to run the Streamlit app local
```

```
df['Churn'].value_counts()
```

```
        count
Churn
```