```
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	InternetService	OnlineSecurity	...
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...

5 rows × 21 columns

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

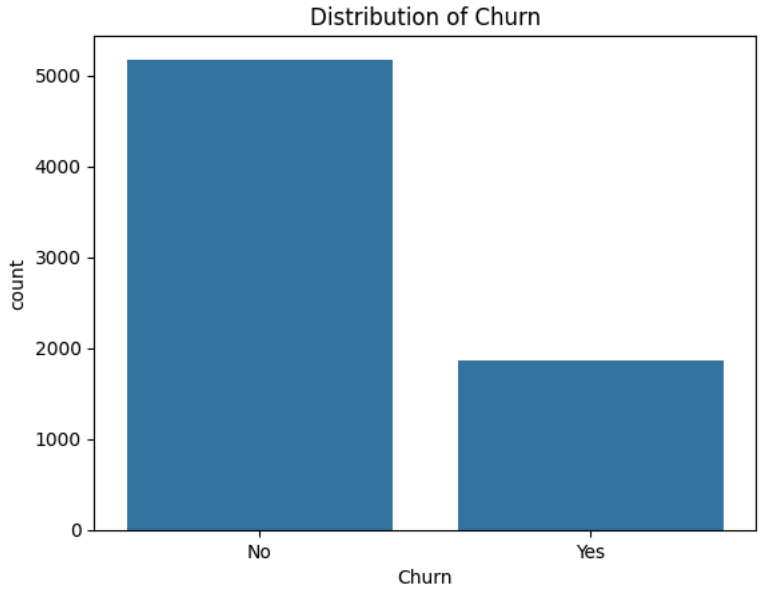
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     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)
```

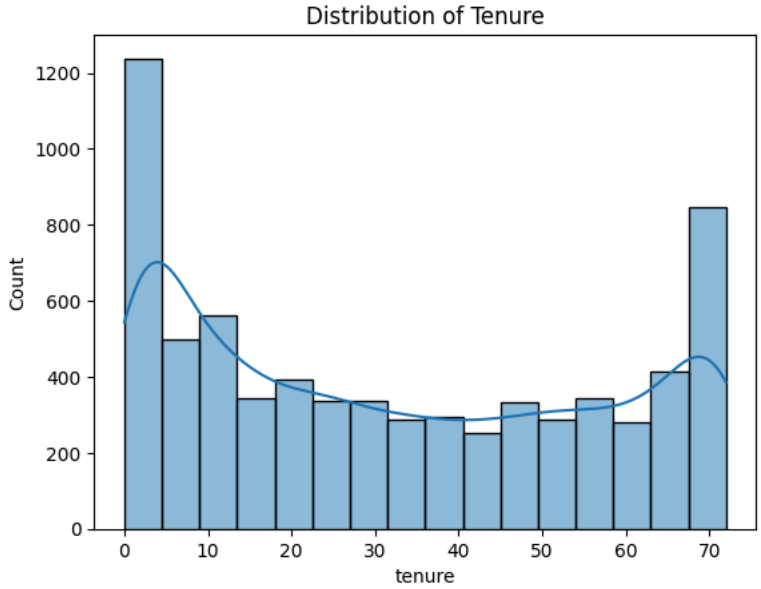
```
#checking missing value and duplicates
print(df.isnull().sum())
print(f"Duplicted Rows :{df.duplicated().sum()}")
```

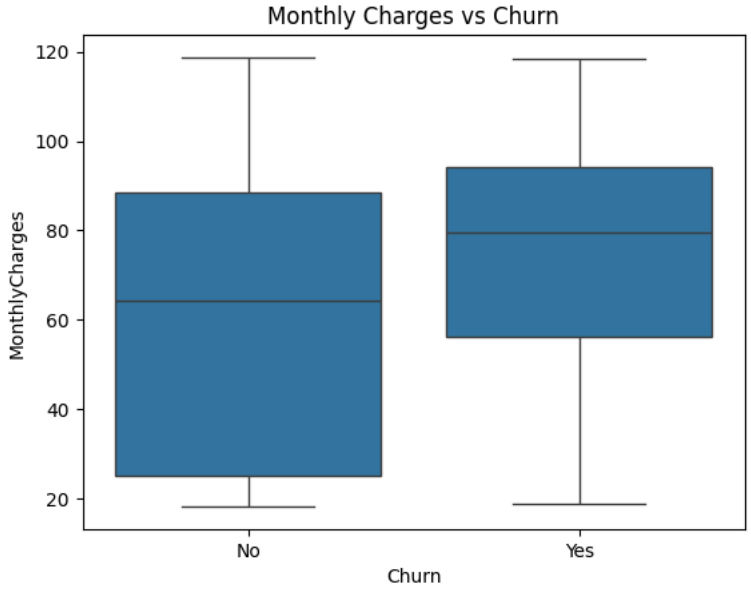
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         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()
```








```
#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)


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

Churn	count
No	5163
Yes	1869

dtype: int64

```
# Apply SMOTE to balance the dataset
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)


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

# Random Forest (balanced)
rf_model = RandomForestClassifier(class_weight='balanced')
rf_model.fit(X_resampled, y_resampled)

# Logistic Regression (balanced)
lr_model = LogisticRegression(class_weight='balanced', max_iter=1000)
lr_model.fit(X_resampled, y_resampled)
```

```
▾ LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=1000)
```

```
#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:
[[931 102]
 [191 183]]
```

```
                  precision    recall  f1-score   support

           False       0.83      0.90      0.86      1033
            True       0.64      0.49      0.56       374

        accuracy                           0.79      1407
       macro avg       0.74      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
```

```
print(f"After SMOTE - Churn Distribution: {y_resampled.value_counts()}")
```

```
After SMOTE - Churn Distribution: Churn_Yes
True     4130
False    4130
Name: count, dtype: int64
```

```
# 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
```

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

```
['columns.pkl']
```

```
!pip install gradio pandas joblib --quiet

import gradio as gr
import pandas as pd
import joblib

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

def predict_churn(gender, senior_citizen, partner, dependents, tenure, monthly, total, phone_service, multiple_lines, internet_service):
    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
    }

    df_input = pd.DataFrame([input_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"
```

```
# Create Gradio interface
iface = gr.Interface(
    fn=predict_churn,
    inputs=[
        gr.Dropdown(['Female', 'Male'], label="Gender"),
        gr.Dropdown(['No', 'Yes'], label="Senior Citizen"),
        gr.Dropdown(['No', 'Yes'], label="Partner"),
        gr.Dropdown(['No', 'Yes'], label="Dependents"),
        gr.Slider(0, 72, label="Tenure (months)"),
        gr.Number(minimum=0.0, label="Monthly Charges"),
        gr.Number(minimum=0.0, label="Total Charges"),
        gr.Dropdown(['No', 'Yes'], label="Phone Service"),
        gr.Dropdown(['No phone service', 'No', 'Yes'], label="Multiple Lines"),
        gr.Dropdown(['DSL', 'Fiber optic', 'No'], label="Internet Service")
    ],
    outputs="text",
    title="Customer Churn Prediction",
    description="Enter customer details to predict churn."
)

# Launch the interface and share it publicly
iface.launch(share=True)
```