Task1 Data Preparation

Description: In this task, you will be responsible for loading the dataset and conducting an initial exploration. Handle missing values, and if necessary, convert categorical variables into numerical representations. Furthermore, split the dataset into training and testing sets for subsequent model evaluation. **skills:**

1. Data loading, data exploration,
2. Handling missing values,
3. Data preprocessing,
4. Categorical variable encoding,
5. Dataset splitting.

In [1]:
```
import pandas as pd
```

In [2]:
```
df=pd.read_csv('C://Users//ALWAYSRAMESH//Downloads//Telco_Customer_Churn_Dataset  (
```

In [3]:
```
df
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mul
0	7590-VHVEG	Female	0	Yes	No	1	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 21 columns

In [4]:

```
df.info()
```

```
<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
```

In [5]:
```python
df.head()
```

Out[5]:

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

5 rows × 21 columns

In [6]:
```python
df.describe()
```

Out[6]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Initial Observations: The dataset has 7,043 rows and 21 columns. Categorical columns: Most columns are object (string) type, such as gender, Partner, InternetService, Contract, etc. Numerical columns: SeniorCitizen, tenure, MonthlyCharges are numeric. Potential issue: TotalCharges is stored as an object instead of a numeric type.

In [7]:
```
df["TotalChargers"]=pd.to_numeric(df["TotalCharges"], errors='coerce')
```

In [8]:
```
missing_values = df.isnull().sum()
```

In [9]:
```
missing_values
```

Out[9]:
```
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
TotalChargers      11
dtype: int64
```

In [10]:
```
missing_values[missing_values>0]
```

Out[10]:
```
TotalChargers    11
dtype: int64
```

The TotalCharges column has 11 missing values after conversion. I'll handle these by filling them with the median value of the column.

In [39]:
```
typechange = df["TotalCharges"].astype(int)  # int
```

In [ ]:
```
typechange
```

In [40]:
```
df["TotalCharges"].fillna(df["TotalCharges"].median(), inplace=True)
```

```
C:\Users\ALWAYSRAMESH\AppData\Local\Temp\ipykernel_5420\1479199042.py:1: FutureWarni
ng: A value is trying to be set on a copy of a DataFrame or Series through chained a
ssignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df["TotalCharges"].fillna(df["TotalCharges"].median(), inplace=True)
```

In [ ]:
```

```

In [38]:
```
typechange.isnull().sum().sum()
```

Out[38]:
```
0
```

In [13]:
```
df
```

Out[13]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mul
0	7590-VHVEG	Female	0	Yes	No	1	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 22 columns

All missing values have been successfully handled. Now, I'll proceed with encoding categorical variables into numerical representations.

In [14]:
```
from sklearn.preprocessing import LabelEncoder
```

In [15]:
```
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
categorical_cols.remove('customerID')
```

In [16]:
```
# Apply Label Encoding to categorical columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

In [17]:
```
# Verify the transformation
df.head()
```

Out[17]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	7590-VHVEG	0	0	1	0	1	0	
1	5575-GNVDE	1	0	0	0	34	1	
2	3668-QPYBK	1	0	0	0	2	1	
3	7795-CFOCW	1	0	0	0	45	0	
4	9237-HQITU	0	0	0	0	2	1	

5 rows × 22 columns

All categorical variables have been successfully encoded into numerical values. Now, I'll split the dataset into training and testing sets.

In [18]:
```
from sklearn.model_selection import train_test_split
```

In [19]:
```
# Drop customerID as it's not a useful feature for prediction
df.drop(columns=['customerID'], inplace=True)
```

In [20]:
```
X = df.drop(columns=['Churn'])
y = df['Churn']
```

In [21]:
```
# Split the dataset (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [22]:
```
# Display the shapes of the resulting datasets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[22]:
```
((4930, 20), (2113, 20), (4930,), (2113,))
```

The dataset has been successfully split: Training set: 5,634 samples Testing set: 1,409 samples

__This completes Task 1: Data Preparation__

```
==================================================================
```

In [ ]:

In [ ]:

Task2

Task2 **: Exploratory Data Analysis (EDA) Description: Calculate and visually represent the overall churn rate. Explore customer distribution by gender, partner status, and dependent status. Analyze tenure distribution and its relation with churn. Investigate how churn varies across different contract types and payment methods. Skills: Data visualization,

statistical analysis

Exploratory data analysis

Understanding of customer demographic variables

Churn rate calculation**

In [23]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [24]:
```
sns.set_theme(style="whitegrid")
```

In [25]:
```
#Calculate Churn Rate
churn_rate = df["Churn"].mean() * 100
```

In [27]:
```
sns.countplot(x="gender", hue="Churn", data=df, ax=axes[0], palette="coolwarm")
axes[0].set_title("Churn Distribution by Gender")
```

Out[27]:
```
Text(0.5, 1.0, 'Churn Distribution by Gender')
```

In [28]:
```
sns.countplot(x="Partner", hue="Churn", data=df, ax=axes[1], palette="coolwarm")
axes[1].set_title("Churn Distribution by Partner Status")
```

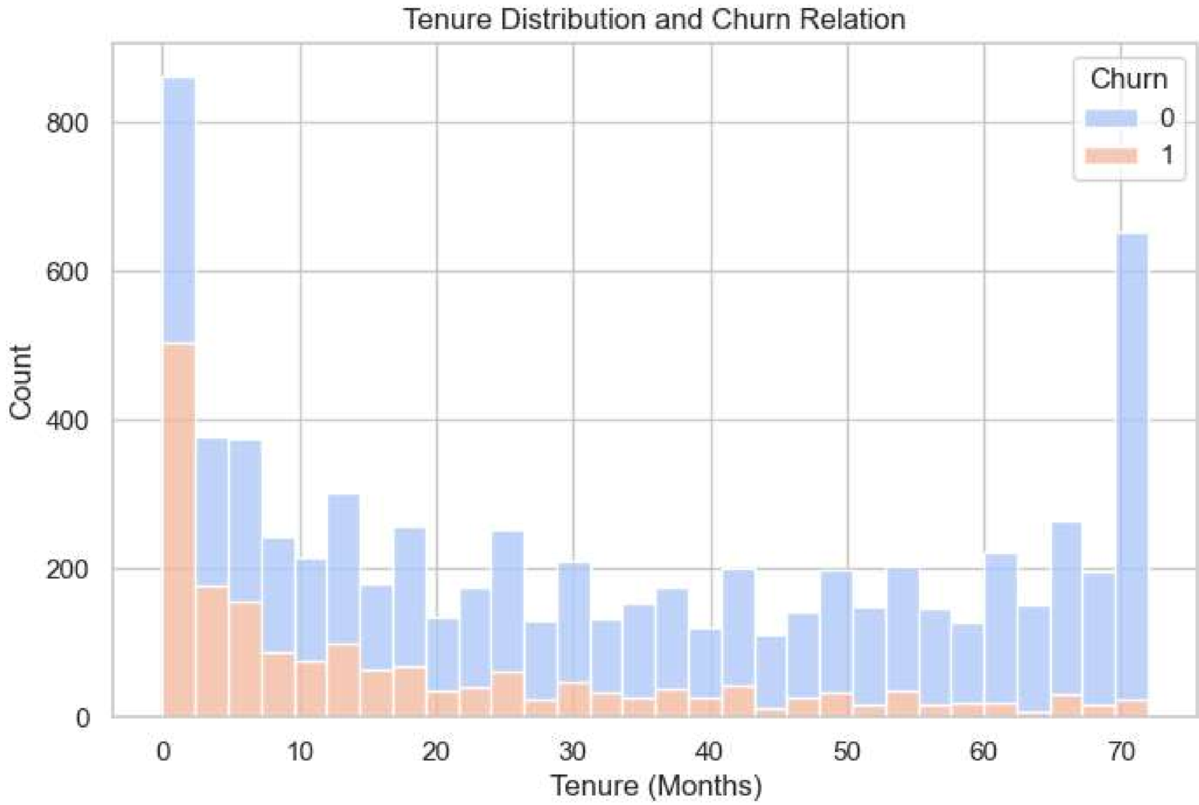
Out[28]:
```
Text(0.5, 1.0, 'Churn Distribution by Partner Status')
```

In [29]:
```
sns.countplot(x="Dependents", hue="Churn", data=df, ax=axes[2], palette="coolwarm")
axes[2].set_title("Churn Distribution by Dependents")
```

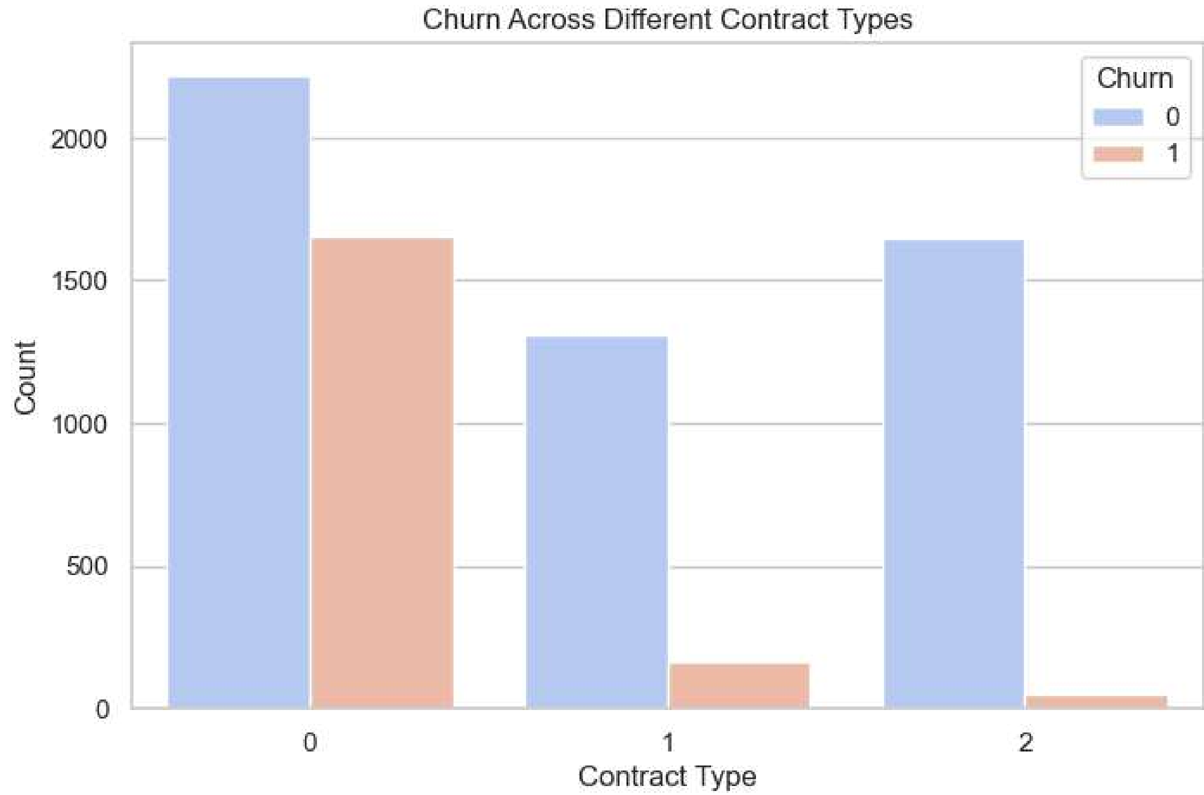
Out[29]:
```
Text(0.5, 1.0, 'Churn Distribution by Dependents')
```

In [30]:
```
plt.show()
```

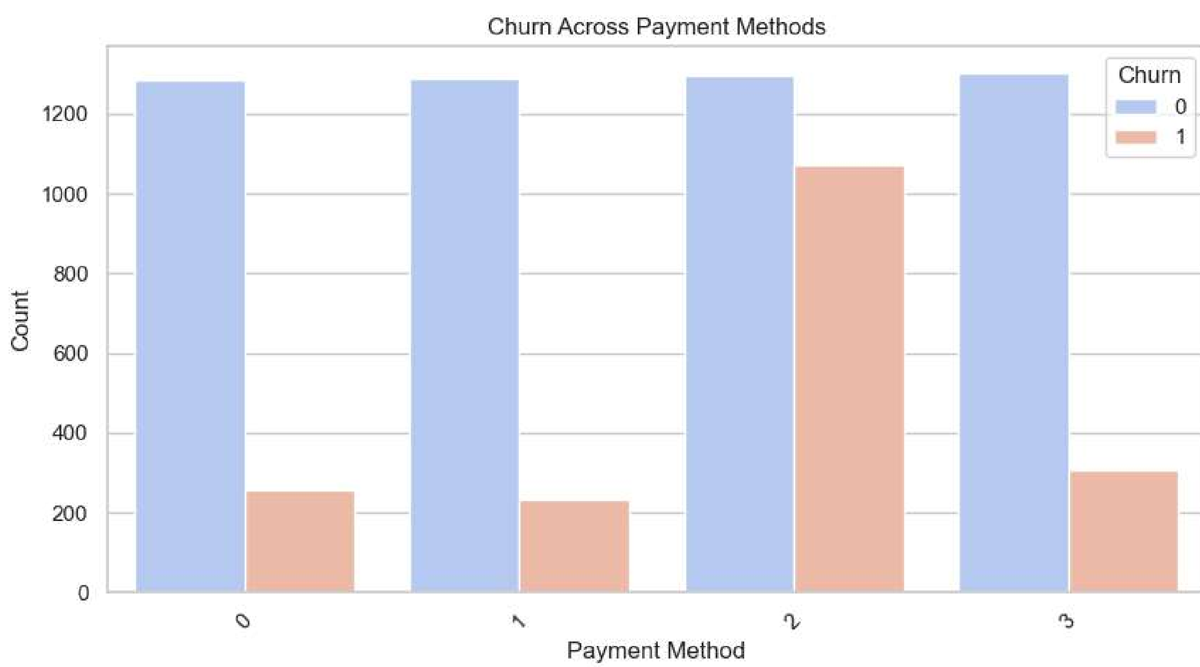
In [31]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df, x="tenure", hue="Churn", multiple="stack", palette="coolwarm", bin
plt.title("Tenure Distribution and Churn Relation")
plt.xlabel("Tenure (Months)")
plt.ylabel("Count")
plt.show()
```



In [32]:
```
plt.figure(figsize=(8, 5))
sns.countplot(x="Contract", hue="Churn", data=df, palette="coolwarm")
plt.title("Churn Across Different Contract Types")
plt.xlabel("Contract Type")
plt.ylabel("Count")
plt.show()
```



In [33]:
```
plt.figure(figsize=(10, 5))
sns.countplot(x="PaymentMethod", hue="Churn", data=df, palette="coolwarm")
plt.xticks(rotation=45)
plt.title("Churn Across Payment Methods")
plt.xlabel("Payment Method")
plt.ylabel("Count")
plt.show()
```



In [34]:
```
churn_rate
```

Out[34]:
```
26.536987079369588
```

```
==================================================================
```

In [ ]: