1.Data Collection

```
[ ] import pandas as pd

    # Load your dataset
    df = pd.read_csv('/content/customer_churn_dataset.csv')
```

```
[ ] df.head()
```

	CustomerID	Age	Gender	Tenure	Usage Frequency	Support Calls	Payment Delay	Subscription Type	Contract Length	Total Spend	Last Interaction	Churn
0	2	30	Female	39	14	5	18	Standard	Annual	932.0	17.0	1.0
1	3	65	Female	49	1	10	8	Basic	Monthly	557.0	6.0	1.0
2	4	55	Female	14	4	6	18	Basic	Quarterly	185.0	3.0	1.0
3	5	58	Male	38	21	7	7	Standard	Monthly	396.0	29.0	1.0
4	6	23	Male	32	20	5	8	Basic	Monthly	617.0	20.0	1.0

Next steps: Generate code with df View recommended plots New interactive sheet

```
[ ] df.describe()
```

	CustomerID	Age	Tenure	Usage Frequency	Support Calls	Payment Delay	Total Spend	Last Interaction	Churn
count	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000	61857.000000	61857.000000	61857.000000
mean	31251.305926	41.568835	30.452439	15.469220	5.070242	15.098888	547.285821	15.548685	0.978531
std	18226.613415	13.905860	17.350471	8.686087	3.155061	8.956264	259.289130	8.642097	0.144942
min	2.000000	18.000000	1.000000	1.000000	0.000000	0.000000	100.000000	1.000000	0.000000
25%	15473.250000	29.000000	15.000000	8.000000	2.000000	7.000000	323.000000	8.000000	1.000000
50%	30937.500000	42.000000	30.000000	15.000000	5.000000	15.000000	546.000000	16.000000	1.000000
75%	47027.500000	54.000000	45.000000	23.000000	8.000000	23.000000	771.000000	23.000000	1.000000
max	63175.000000	65.000000	60.000000	30.000000	10.000000	30.000000	1000.000000	30.000000	1.000000

2. Data Preprocessing

Check for missing values in the training dataset

```
[ ] df.isnull().sum()
```

	0
CustomerID	0
Age	0
Gender	0
Tenure	0
Usage Frequency	0
Support Calls	0
Payment Delay	0
Subscription Type	0
Contract Length	1
Total Spend	1
Last Interaction	1
Churn	1

dtype: int64

Handle missing values for numeric columns only

```
[ ] numeric_columns = df.select_dtypes(include=['float', 'int']).columns
    df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].median()) # Fill missing values in numeric columns with their medians
```

Check for missing values again

```
[ ] df.isnull().sum()

    # Detect and handle outliers if necessary
    # For example, removing rows with extremely high or low values
```

	0
CustomerID	0

Age	0
Gender	0
Tenure	0
Usage Frequency	0
Support Calls	0
Payment Delay	0
Subscription Type	0
Contract Length	1
Total Spend	0
Last Interaction	0
Churn	0

dtype: int64

Data Normalization

```
[ ] from sklearn.preprocessing import MinMaxScaler

    # Strip whitespace from column names if necessary
    df.columns = df.columns.str.strip()
```

Adjust column names based on actual columns

```
[ ] numeric_columns = ['Tenure', 'Total Spend', 'Usage Frequency']  # Adjusted to match the dataset
```

Check if columns exist in DataFrame

```
[ ] missing_columns = [col for col in numeric_columns if col not in df.columns]
    if missing_columns:
        raise ValueError(f"Columns missing in DataFrame: {missing_columns}")
```

Normalize numeric columns

```
[ ] scaler = MinMaxScaler()
    df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
```

3. Exploratory Analysis

Statistical summary of the training dataset

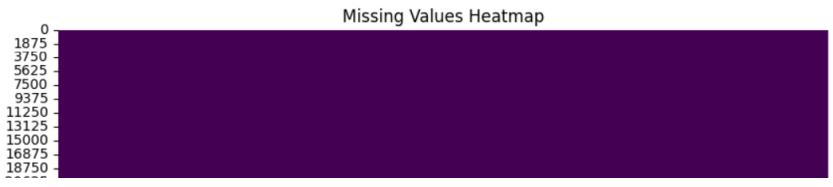
```
[ ] df.describe()
```

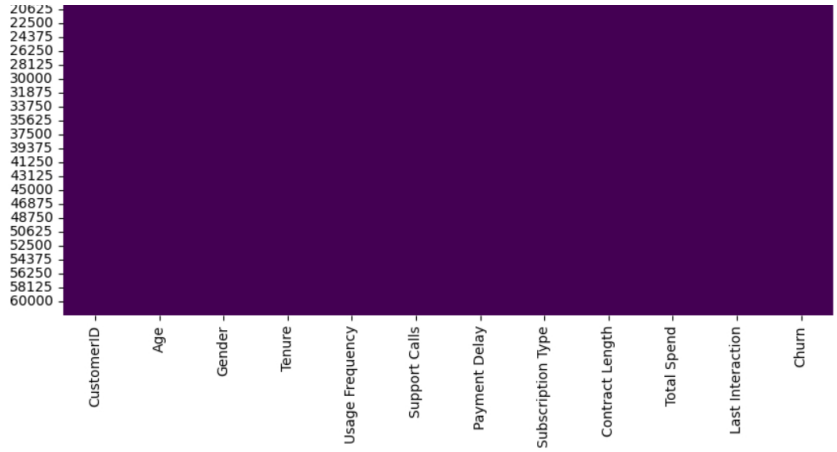
	CustomerID	Age	Tenure	Usage Frequency	Support Calls	Payment Delay	Total Spend	Last Interaction	Churn
count	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000	61858.000000
mean	31251.305926	41.568835	0.499194	0.498939	5.070242	15.098888	0.496984	15.548692	0.978531
std	18226.613415	13.905860	0.294076	0.299520	3.155061	8.956264	0.288097	8.642028	0.144941
min	2.000000	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
25%	15473.250000	29.000000	0.237288	0.241379	2.000000	7.000000	0.247778	8.000000	1.000000
50%	30937.500000	42.000000	0.491525	0.482759	5.000000	15.000000	0.495556	16.000000	1.000000
75%	47027.500000	54.000000	0.745763	0.758621	8.000000	23.000000	0.745556	23.000000	1.000000
max	63175.000000	65.000000	1.000000	1.000000	10.000000	30.000000	1.000000	30.000000	1.000000

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

    # Define numeric columns based on provided column names
    numeric_columns = ['Tenure', 'Usage Frequency', 'Total Spend']  # Adjusted based on your columns
```

Visualize missing values heatmap

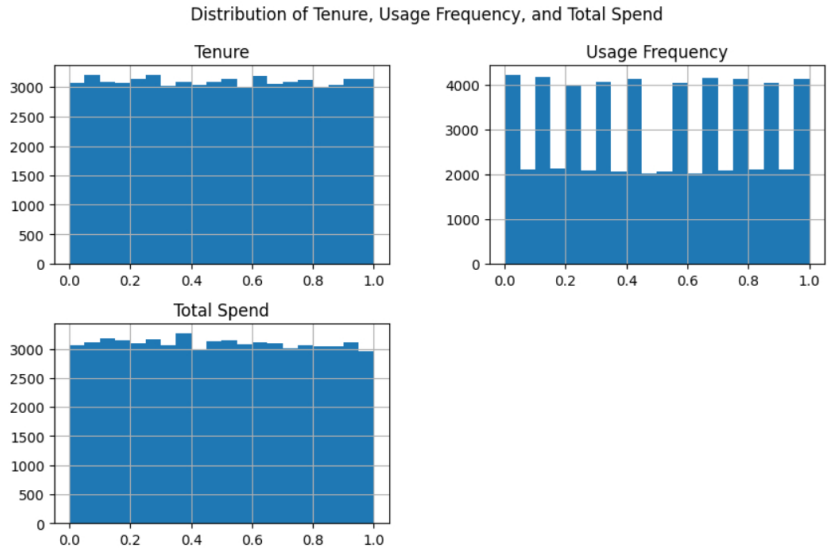
```
[ ] plt.figure(figsize=(10, 6))
    sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
    plt.title('Missing Values Heatmap')
    plt.show()
```





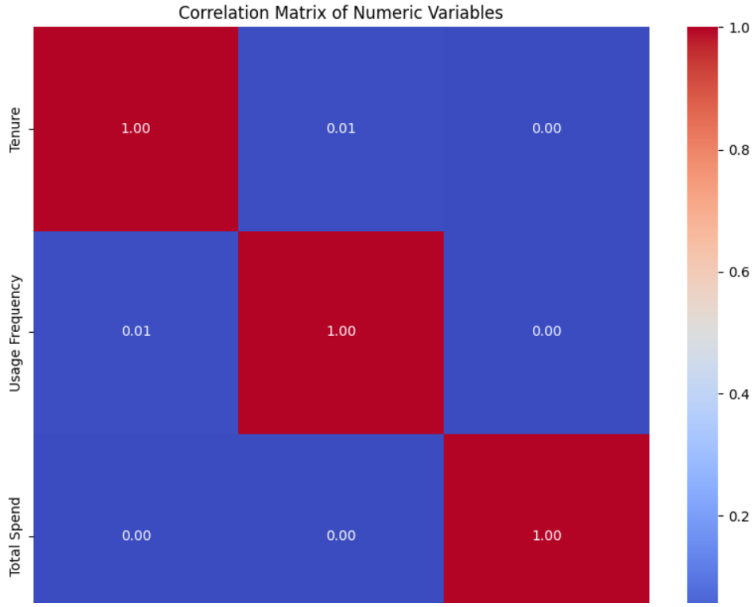
Histograms for numerical columns

```
[ ] df[numeric_columns].hist(bins=20, figsize=(10, 6))
    plt.suptitle('Distribution of Tenure, Usage Frequency, and Total Spend')
    plt.show()
```



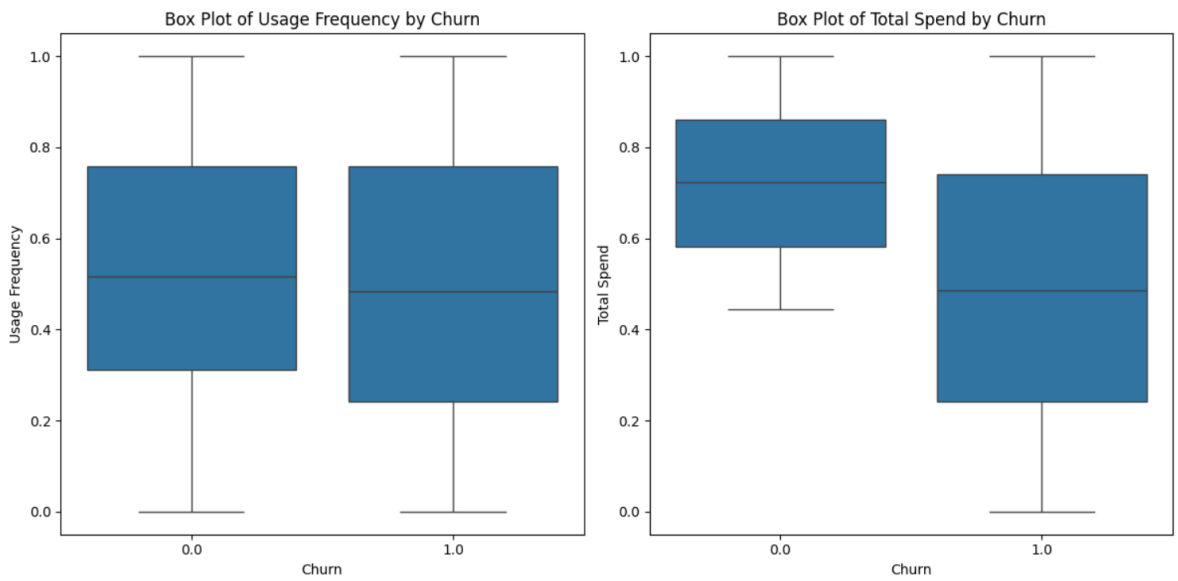
Correlation matrix

```
[ ] corr = df[numeric_columns].corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
    plt.title('Correlation Matrix of Numeric Variables')
    plt.show()
```



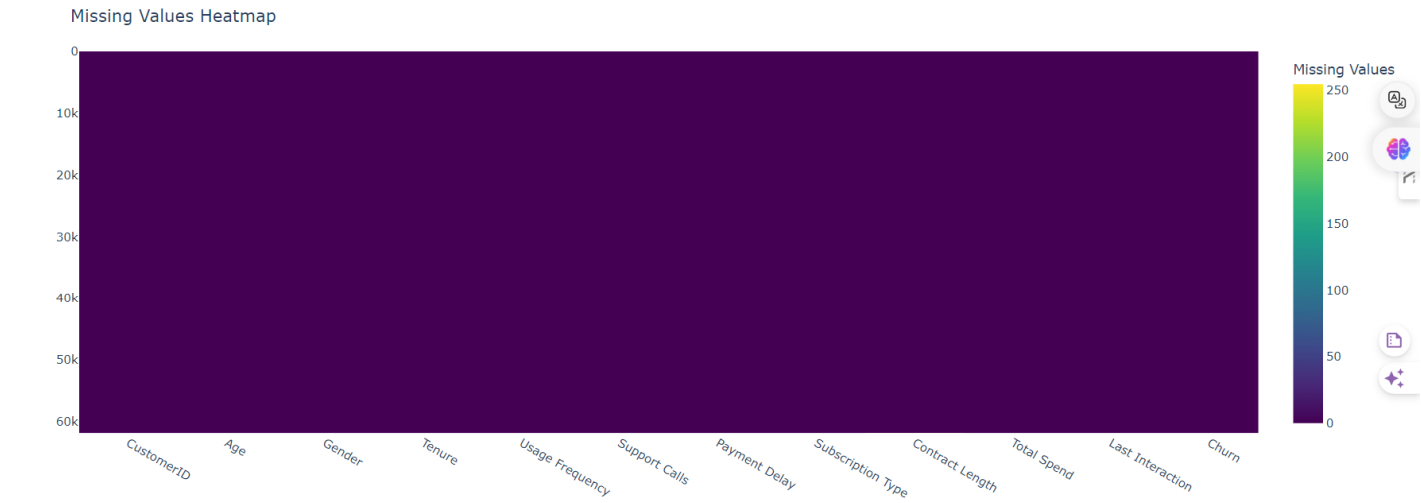
Box plot of Usage Frequency and Total Spend by Churn

```
[ ] plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    sns.boxplot(x='Churn', y='Usage Frequency', data=df)
    plt.title('Box Plot of Usage Frequency by Churn')

    plt.subplot(1, 2, 2)
    sns.boxplot(x='Churn', y='Total Spend', data=df)
    plt.title('Box Plot of Total Spend by Churn')

    plt.tight_layout()
    plt.show()
```



Interactive heatmap for missing values

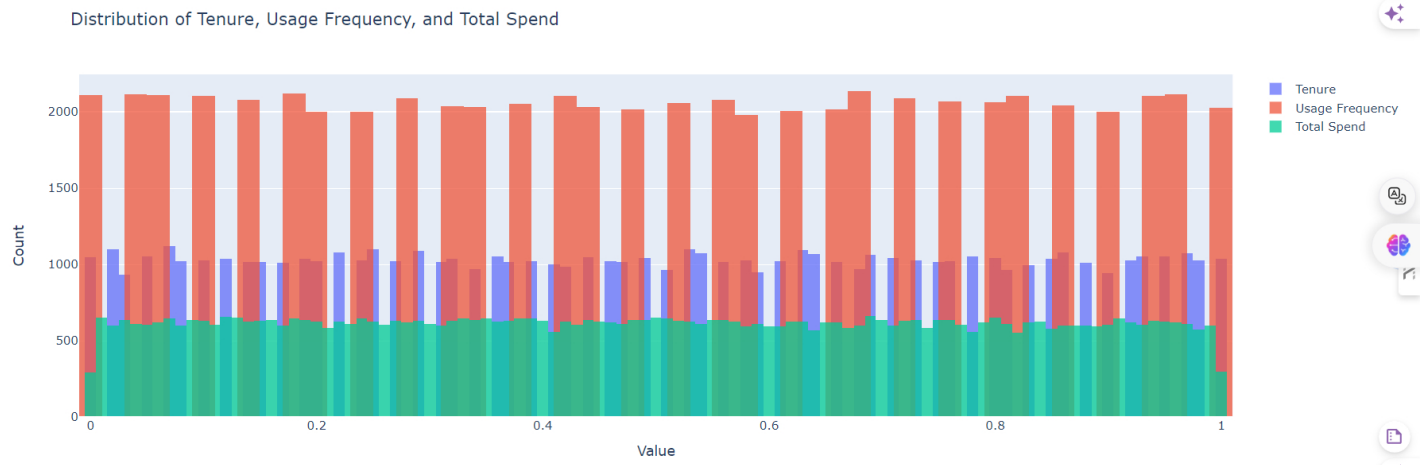
```
import plotly.express as px
import plotly.graph_objects as go

fig = px.imshow(df.isnull(), color_continuous_scale='viridis', labels={'color': 'Missing Values'})
fig.update_layout(title='Missing Values Heatmap')
fig.show()
```



Interactive histograms

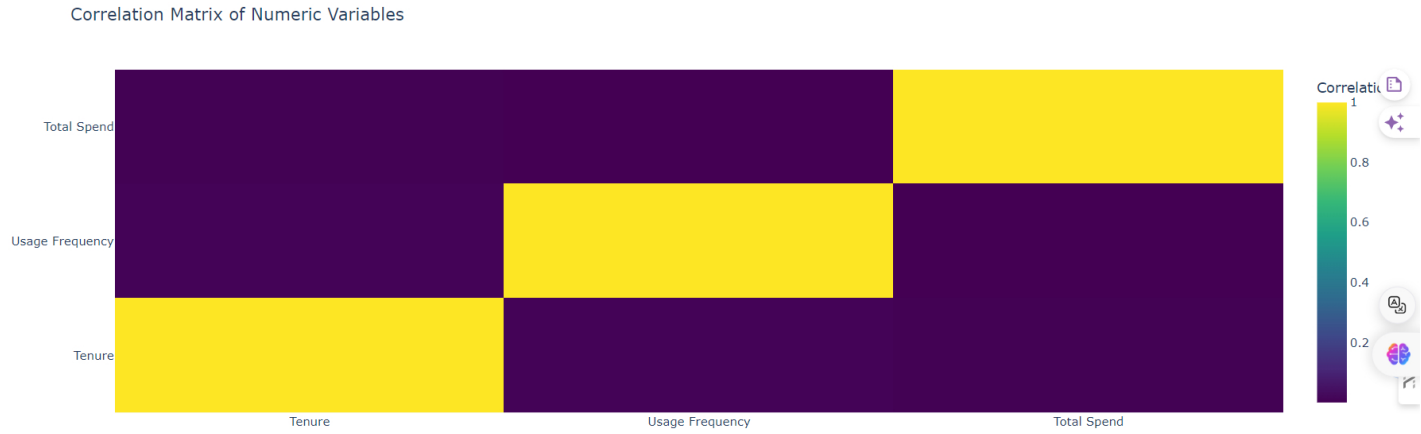
```
[ ] fig = go.Figure()

    for col in numeric_columns:
        fig.add_trace(go.Histogram(x=df[col], name=col, opacity=0.75))

    fig.update_layout(title='Distribution of Tenure, Usage Frequency, and Total Spend',
                      barmode='overlay', xaxis_title='Value', yaxis_title='Count')
```

```
fig.show()
```



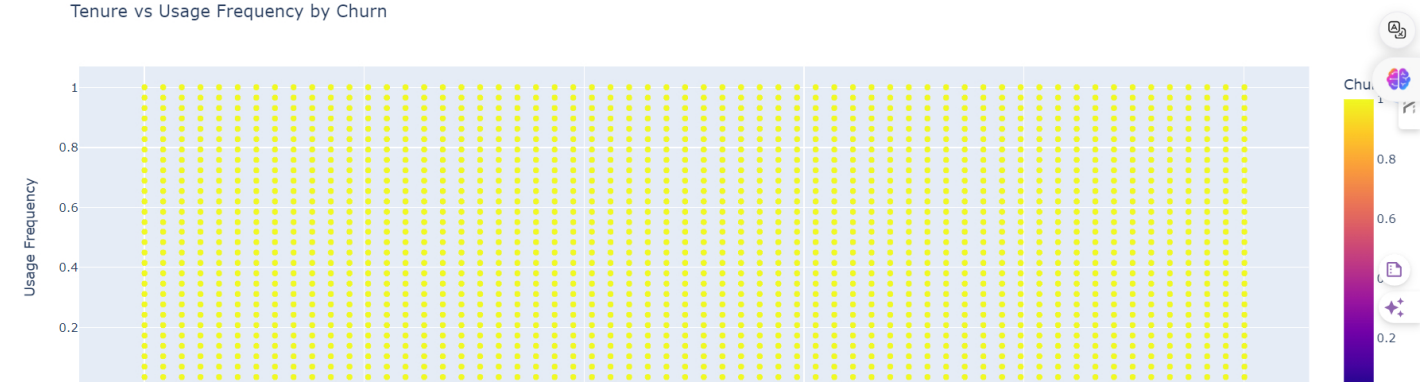
Interactive correlation matrix

```
[ ] corr = df[numeric_columns].corr()
    fig = go.Figure(data=go.Heatmap(z=corr.values,
                                    x=corr.columns,
                                    y=corr.columns,
                                    colorscale='Viridis', # Change to a valid colorscale name
                                    colorbar=dict(title='Correlation')))

    fig.update_layout(title='Correlation Matrix of Numeric Variables')
    fig.show()
```



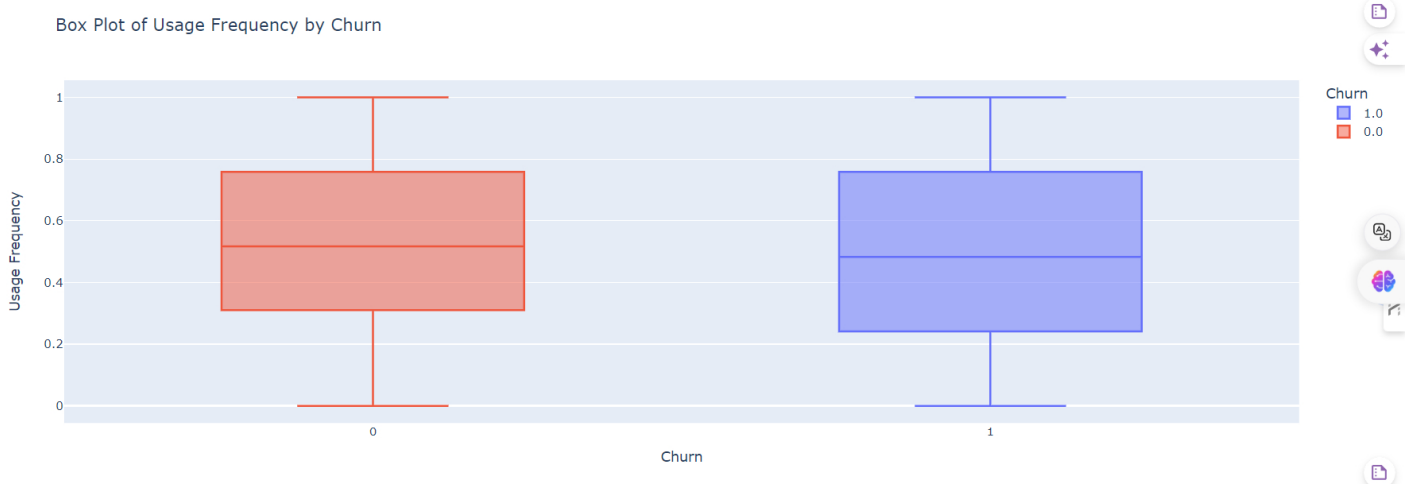
Interactive scatter plot

```
[ ] fig = px.scatter(df, x='Tenure', y='Usage Frequency', color='Churn',
                     labels={'Tenure': 'Tenure', 'Usage Frequency': 'Usage Frequency'},
                     title='Tenure vs Usage Frequency by Churn')
    fig.show()
```



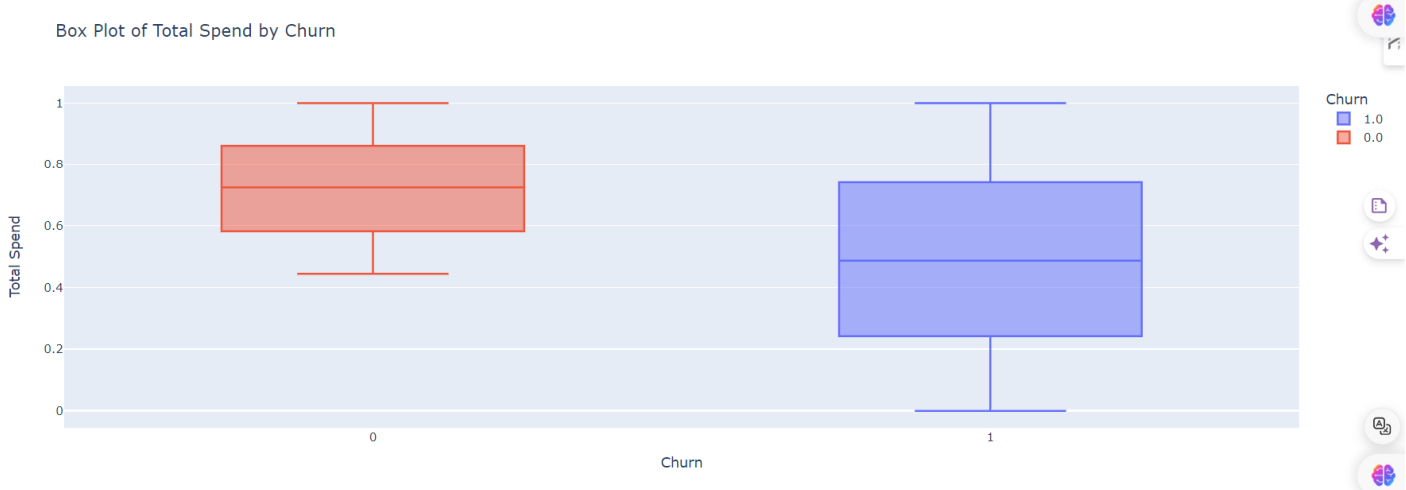
Interactive box plot for Usage Frequency

```
[ ] fig = px.box(df, x='Churn', y='Usage Frequency', color='Churn',
                 labels={'Churn': 'Churn', 'Usage Frequency': 'Usage Frequency'},
                 title='Box Plot of Usage Frequency by Churn')
    fig.show()
```



Interactive box plot for Total Spend

```
[ ] fig = px.box(df, x='Churn', y='Total Spend', color='Churn',
                 labels={'Churn': 'Churn', 'Total Spend': 'Total Spend'},
                 title='Box Plot of Total Spend by Churn')
    fig.show()
```

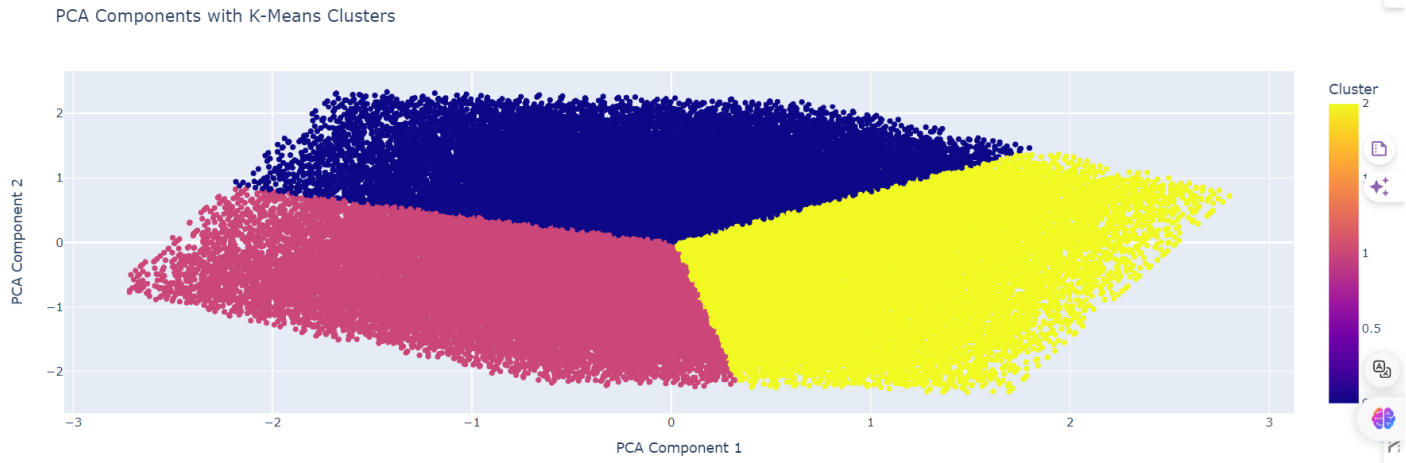


PCA and K-Means Clustering

```
[ ] import pandas as pd
    import numpy as np
    import plotly.express as px
    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.cluster import KMeans
    from sklearn.pipeline import Pipeline
    from sklearn.impute import SimpleImputer # Import SimpleImputer to handle NaNs


    # Define numeric columns based on your dataset
    numeric_columns = ['Tenure', 'Usage Frequency', 'Total Spend']  # Update if needed

    # Preprocessing: Standardize the data
    scaler = StandardScaler()
    # Impute missing values with the mean of each column
    imputer = SimpleImputer(strategy='mean') # Create an imputer object
    df_imputed = imputer.fit_transform(df[numeric_columns]) # Impute NaNs
```

```
df_scaled = scaler.fit_transform(df_imputed) # Scale the imputed data

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)  # Reduce to 2 dimensions for visualization
df_pca = pca.fit_transform(df_scaled)

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)  # Change number of clusters as needed
clusters = kmeans.fit_predict(df_pca)

# Add PCA components and cluster labels to the DataFrame
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]
df['Cluster'] = clusters

# Interactive scatter plot of PCA components colored by cluster
fig = px.scatter(df, x='PCA1', y='PCA2', color='Cluster',
                 labels={'PCA1': 'PCA Component 1', 'PCA2': 'PCA Component 2'},
                 title='PCA Components with K-Means Clusters')
fig.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```



Double-click (or enter) to edit

✓ 2s completed at 10:24 PM