a -

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

b -

In [2]:
```
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

In [ ]:
```
c -
```

In [3]:
```
nb_of_features = df.shape[1]
nb_of_observations = df.shape[0]
print("There are ",nb_of_features, "features and ",nb_of_observations,"observations in this dataset" )
```

```
There are  21 features and  7043 observations in this dataset
```

In [ ]:
```
d -
```

In [4]:
```
# Calculate the percentage of missing values in each column
missing_values = df.isnull().sum()
total_values = df.shape[0]
missing_values_pct = (missing_values / total_values) * 100

# Print the percentage of missing values for each column
print(missing_values_pct)
```

```
customerID          0.0
gender              0.0
SeniorCitizen       0.0
Partner             0.0
Dependents          0.0
tenure              0.0
PhoneService        0.0
MultipleLines       0.0
InternetService     0.0
OnlineSecurity      0.0
OnlineBackup        0.0
DeviceProtection    0.0
TechSupport         0.0
StreamingTV         0.0
StreamingMovies     0.0
Contract            0.0
PaperlessBilling    0.0
PaymentMethod       0.0
MonthlyCharges      0.0
TotalCharges        0.0
Churn               0.0
dtype: float64
```

e -

In [6]:
```
dependent_variable = df['Churn']

# number of instances in each class of that variable
class_counts = dependent_variable.value_counts()


# the percentage of instances in each class
class_percentages = class_counts / df.shape[0] * 100

print(class_percentages)
```

```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

In [ ]:
```
f -
```

In [14]:
```
# check of missing values
print(df.isnull().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
```

In [15]:
```
#describe numerical features
df.describe()
```

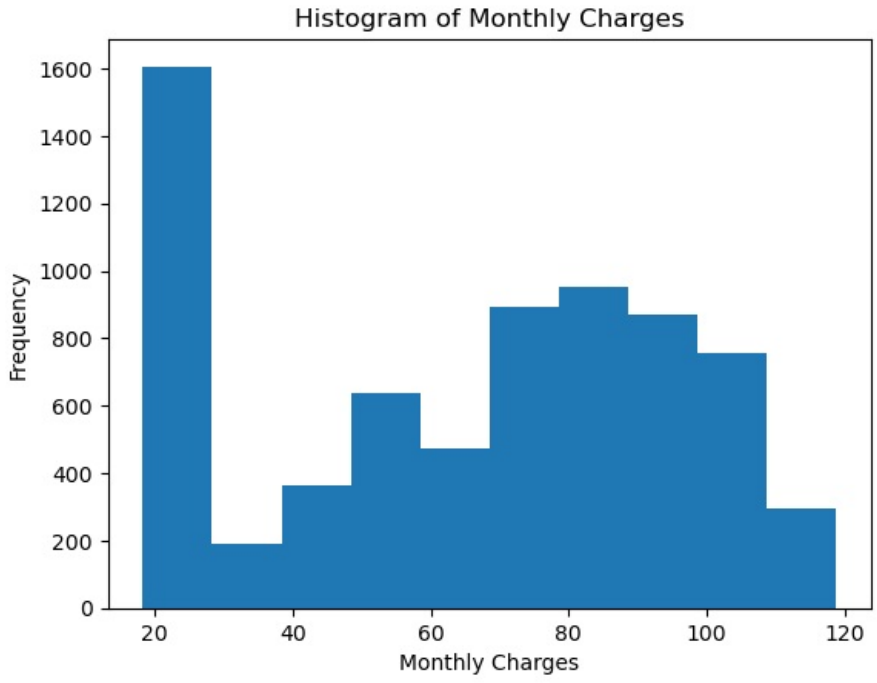
Out[15]:

	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

In [16]:
```
# Explore the distribution of the outcome variable
print(df['Churn'].value_counts())
```

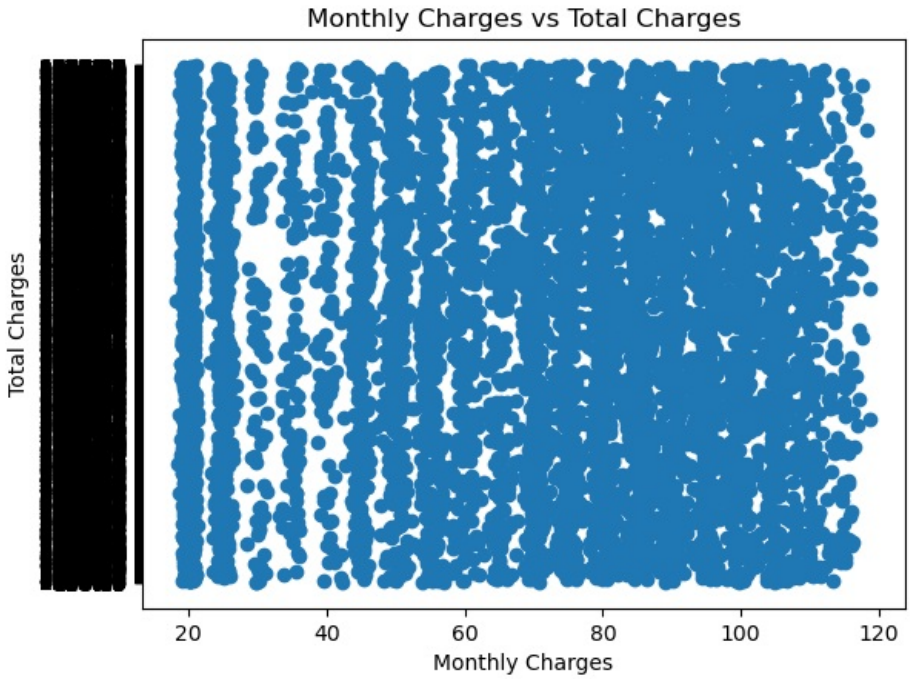
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [17]:
```
# Create a histogram of monthly charges
plt.hist(df['MonthlyCharges'])
plt.xlabel('Monthly Charges')
plt.ylabel('Frequency')
plt.title('Histogram of Monthly Charges')
plt.show()
```



In [19]:
```
# A scatter plot of monthly charges vs total charges
plt.scatter(df['MonthlyCharges'], df['TotalCharges'])
plt.xlabel('Monthly Charges')
```

```
plt.ylabel('Total Charges')
plt.title('Monthly Charges vs Total Charges')
plt.show()
```



In [20]:
```
# Analyze the categorical features
for feature in df.columns:
    if df[feature].dtypes == 'object':
        print(df[feature].value_counts())
        print('-' * 20)
```

```
7590-VHVEG    1
3791-LGQCY    1
6008-NAIXK    1
5956-YHHRX    1
5365-LLFYV    1
             ..
9796-MVYXX    1
2637-FKFSY    1
1552-AAGRX    1
4304-TSPVK    1
3186-AJIEK    1
Name: customerID, Length: 7043, dtype: int64
--------------------
Male      3555
Female    3488
Name: gender, dtype: int64
--------------------
No     3641
Yes    3402
Name: Partner, dtype: int64
--------------------
No     4933
Yes    2110
Name: Dependents, dtype: int64
--------------------
Yes    6361
No      682
Name: PhoneService, dtype: int64
--------------------
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64
--------------------
Fiber optic    3096
DSL            2421
No             1526
Name: InternetService, dtype: int64
--------------------
No                     3498
Yes                    2019
No internet service    1526
Name: OnlineSecurity, dtype: int64
--------------------
No                     3088
Yes                    2429
No internet service    1526
Name: OnlineBackup, dtype: int64
--------------------
No                     3095
```

```
Yes                     2422
No internet service     1526
Name: DeviceProtection, dtype: int64
--------------------
No                      3473
Yes                     2044
No internet service     1526
Name: TechSupport, dtype: int64
--------------------
No                      2810
Yes                     2707
No internet service     1526
Name: StreamingTV, dtype: int64
--------------------
No                      2785
Yes                     2732
No internet service     1526
Name: StreamingMovies, dtype: int64
--------------------
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
--------------------
Yes    4171
No     2872
Name: PaperlessBilling, dtype: int64
--------------------
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: PaymentMethod, dtype: int64
--------------------
          11
20.2      11
19.75      9
20.05      8
19.9       8
          ..
6849.4     1
692.35     1
130.15     1
3211.9     1
6844.5     1
Name: TotalCharges, Length: 6531, dtype: int64
--------------------
No     5174
Yes    1869
Name: Churn, dtype: int64
--------------------
```

g -

Preprocessing of numerical features

In [47]:
```
# Checking for missing values in "tenure" column and replacing them if any with the median or median of the fea
#"tenure"
missing_values = df['tenure'].isnull().sum()
if missing_values.any():
    for df['tenure'] in df.columns:
        if df['tenure'].dtypes != 'object' and missing_values[df['tenure']] > 0:
            if df['tenure'].skew() < 0.5:
                df['tenure'].fillna(df['tenure'].mean(), inplace=True)
            else:
                df['tenure'].fillna(df['tenure'].median(), inplace=True)
print(df['tenure'])
```

```
0        1
1       34
2        2
3       45
4        2
       ..
7038    24
7039    72
7040    11
7041     4
7042    66
Name: tenure, Length: 7043, dtype: int64
```

In [46]:
```
# Checking for missing values in "tenure" column and replacing them if any with the median or median of the fea
#"tenure"
missing_values = df['MonthlyCharges'].isnull().sum()
if missing_values.any():
    for feature in df['MonthlyCharges'].columns:
        if df['MonthlyCharges'].dtypes != 'object' and missing_values[df['MonthlyCharges']] > 0:
            if df[feature].skew() < 0.5:
```

```
            df['MonthlyCharges'].fillna(df['MonthlyCharges'].mean(), inplace=True)
        else:
            df['MonthlyCharges'].fillna(df['MonthlyCharges'].median(), inplace=True)
print(df['MonthlyCharges'])
```

```
0        29.85
1        56.95
2        53.85
3        42.30
4        70.70
        ...
7038     84.80
7039    103.20
7040     29.60
7041     74.40
7042    105.65
Name: MonthlyCharges, Length: 7043, dtype: float64
```

Preprocessing of some categorical values by representing them as numerical values using binary or one-encoding depending on their number of categories

In [51]:
```
for feature in df.columns:
    if df['Churn'].dtypes == 'object':
        if len(df['Churn'].unique()) <= 10:
            # One-hot encode the feature using the OneHotEncoder class from scikit-learn
            from sklearn.preprocessing import OneHotEncoder
            encoder = OneHotEncoder()

            # Convert the feature to a NumPy array
            feature_array = df['Churn'].to_numpy()

            # Reshape the feature into a 2D array
            feature_array_reshaped = feature_array.reshape(-1, 1)

            # Encode the feature
            encoded_features = encoder.fit_transform(feature_array_reshaped)

            # Convert the encoded features to a DataFrame and add it to the original DataFrame
            encoded_features_df = pd.DataFrame(encoded_features.toarray(), columns=encoder.get_feature_names([f
            df = pd.concat([df, encoded_features_df], axis=1)

            # Drop the original feature
            df.drop('Churn', axis=1, inplace=True)
        else:
            # Label encode the feature using the LabelEncoder class from scikit-learn
            from sklearn.preprocessing import LabelEncoder
            encoder = LabelEncoder()
            df['Churn'] = encoder.fit_transform(df['Churn'])
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_na
mes is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_na
mes_out instead.
  warnings.warn(msg, category=FutureWarning)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17544\680002280.py in <module>
     16
     17             # Convert the encoded features to a DataFrame and add it to the original DataFrame
---> 18             encoded_features_df = pd.DataFrame(encoded_features.toarray(), columns=encoder.get_feature_
names([feature_array_reshaped]))
     19             df = pd.concat([df, encoded_features_df], axis=1)
     20

~\anaconda3\lib\site-packages\pandas\core\frame.py in __init__(self, data, index, columns, dtype, copy)
    692                 )
    693             else:
--> 694                 mgr = ndarray_to_mgr(
    695                     data,
    696                     index,

~\anaconda3\lib\site-packages\pandas\core\internals\construction.py in ndarray_to_mgr(values, index, columns, d
type, copy, typ)
    345
    346     # _prep_ndarray ensures that values.ndim == 2 at this point
--> 347     index, columns = _get_axes(
    348         values.shape[0], values.shape[1], index=index, columns=columns
    349     )

~\anaconda3\lib\site-packages\pandas\core\internals\construction.py in _get_axes(N, K, index, columns)
    759         columns = default_index(K)
    760     else:
--> 761         columns = ensure_index(columns)
    762     return index, columns
    763

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in ensure_index(index_like, copy)
   7058             return Index._with_infer(index_like, copy=copy, tupleize_cols=False)
   7059     else:
-> 7060         return Index._with_infer(index_like, copy=copy)
   7061
   7062

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in _with_infer(cls, *args, **kwargs)
    678         with warnings.catch_warnings():
    679             warnings.filterwarnings("ignore", ".*the Index constructor", FutureWarning)
--> 680             result = cls(*args, **kwargs)
    681
    682         if result.dtype == _dtype_obj and not result._is_multi:

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in __new__(cls, data, dtype, copy, name, tupleize_col
s, **kwargs)
    492
    493                 if dtype is None:
--> 494                     arr = _maybe_cast_data_without_dtype(
    495                         arr, cast_numeric_deprecated=True
    496                     )

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in _maybe_cast_data_without_dtype(subarr, cast_numeri
c_deprecated)
   7139     """
   7140
-> 7141     result = lib.maybe_convert_objects(
   7142         subarr,
   7143         convert_datetime=True,

~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx in pandas._libs.lib.maybe_convert_objects()

ValueError: Buffer has wrong number of dimensions (expected 1, got 3)
```

The following justifies the preprocessing choices: Numerical features: Impute missing values: It's critical to impute missing values prior to model training because they can significantly affect how well machine learning models perform. Depending on how skewed the feature distribution is, missing values may be imputed using the feature's mean or median. Categorical features: Use binary encoding or one-hot encoding to encode category features: For machine learning models to understand categorical data, they must be converted into a numerical format using one-hot encoding, also known as label encoding. When the categories of a categorical feature are not ordered, one-hot encoding is usually employed; when the categories are ordered, label encoding is usually used.