Loading the data.

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('/Users/cantu/Documents/Data Science Masters/Applied DS DSC68(
df
```

Out[2]:

	Patient ID	Age	Gender	Tumor Type	Tumor Grade	Tumor Location	Treatment	Treatment Outcome	Time Recurren (mont
0	1	45	Male	Glioblastoma	IV	Frontal lobe	Surgery	Partial response	1
1	2	55	Female	Meningioma	I	Parietal lobe	Surgery	Complete response	N
2	3	60	Male	Astrocytoma	III	Occipital lobe	Surgery + Chemotherapy	Progressive disease	1
3	4	50	Female	Glioblastoma	IV	Temporal lobe	Surgery + Radiation therapy	Complete response	N
4	5	65	Male	Astrocytoma	II	Frontal lobe	Surgery + Radiation therapy	Partial response	2
...	...	...	...	...	...	...	...	...	
1995	1996	49	Female	Meningioma	I	Parietal lobe	Radiation	Progressive disease	1
1996	1997	57	Male	Glioblastoma	IV	Occipital lobe	Surgery	Complete response	N
1997	1998	45	Female	Meningioma	I	Temporal lobe	Chemotherapy	Partial response	2
1998	1999	62	Male	Astrocytoma	III	Frontal lobe	Radiation	Stable disease	2
1999	2000	53	Female	Glioblastoma	IV	Parietal lobe	Chemotherapy + Radiation	Progressive disease	1

2000 rows × 11 columns

Initial discovery

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 11 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Patient ID                    2000 non-null   int64  
 1   Age                           2000 non-null   int64  
 2   Gender                        2000 non-null   object 
 3   Tumor Type                    2000 non-null   object 
 4   Tumor Grade                   2000 non-null   object 
 5   Tumor Location                2000 non-null   object 
 6   Treatment                     2000 non-null   object 
 7   Treatment Outcome             2000 non-null   object 
 8   Time to Recurrence (months)   1438 non-null   float64
 9   Recurrence Site               1438 non-null   object 
 10  Survival Time (months)        2000 non-null   int64  
dtypes: float64(1), int64(3), object(7)
memory usage: 172.0+ KB
```

In [5]:
```
print(df.isnull().sum())
```

```
Patient ID                       0
Age                              0
Gender                           0
Tumor Type                       0
Tumor Grade                      0
Tumor Location                   0
Treatment                        0
Treatment Outcome                0
Time to Recurrence (months)    562
Recurrence Site                562
Survival Time (months)           0
dtype: int64
```

In [6]:
```
# replace spaces in column names with underscores
df.columns = df.columns.str.replace(' ', '_')

# display the data with updated column names
print(df.head())
```

```
   Patient_ID  Age  Gender    Tumor_Type Tumor_Grade  Tumor_Location  \
0           1   45    Male  Glioblastoma          IV    Frontal lobe   
1           2   55  Female    Meningioma           I   Parietal lobe   
2           3   60    Male   Astrocytoma         III  Occipital lobe   
3           4   50  Female  Glioblastoma          IV   Temporal lobe   
4           5   65    Male   Astrocytoma          II    Frontal lobe   

                    Treatment    Treatment_Outcome  \
0                     Surgery     Partial response   
1                     Surgery    Complete response   
2       Surgery + Chemotherapy  Progressive disease   
3  Surgery + Radiation therapy    Complete response   
4  Surgery + Radiation therapy     Partial response   

   Time_to_Recurrence_(months) Recurrence_Site  Survival_Time_(months)  
0                         10.0   Temporal lobe                      18  
1                          NaN             NaN                      36  
2                         14.0    Frontal lobe                      22  
3                          NaN             NaN                      12  
4                         24.0    Frontal lobe                      48  
```

In [7]:
```
def roman_to_int(roman):
    """
    convert roman numeral to an integer.

    args:
        roman (str): Roman numeral to convert.

    returns:
        int: Numeric value of the Roman numeral.
    """
    values = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000
    }
    result = 0
    prev_value = 0
    for letter in roman:
        value = values[letter]
        if value > prev_value:
            result += value - 2 * prev_value
        else:
            result += value
        prev_value = value
    return result
df['Tumor_Grade'] = df['Tumor_Grade'].apply(roman_to_int)
df['Tumor_Grade'] = pd.to_numeric(df['Tumor_Grade'])
```

In [8]:
```
from skimpy import skim
skim(df)
```

skimpy summary

Data Summary

dataframe	Values
Number of rows	2000
Number of columns	11

Data Types

Column Type	Count
string	6
int64	4
float64	1

number

column_name	NA	NA %	mean	sd	p0	p25
Patient_ID	0	0	1000	577.5	1	50
Age	0	0	56.15	6.078	42	
Tumor_Grade	0	0	2.433	1.261	1	
Time_to_Recurrence_(months)	562	28.1	16.1	3.128	6	
Survival_Time_(months)	0	0	34.27	8.606	9	

string

column_name	NA	NA %	words per ro
Gender	0	0	
Tumor_Type	0	0	
Tumor_Location	0	0	
Treatment	0	0	
Treatment_Outcome	0	0	
Recurrence_Site	562	28.1	

End

EDA

In [ ]:

The only null values exist within time to reocurence and reocurence site, could this be due to all these events are the result of a complete response of surgery outcome and the patient no longer has tumor?

In [11]:
```
complete_response_data = df[df['Treatment_Outcome'] == 'Complete response']
complete_response_data
#complete_response_data.head()
```

Out[11]:

	Patient_ID	Age	Gender	Tumor_Type	Tumor_Grade	Tumor_Location	Treatment	Tre
1	2	55	Female	Meningioma	1	Parietal lobe	Surgery	C
3	4	50	Female	Glioblastoma	4	Temporal lobe	Surgery + Radiation therapy	C
6	7	55	Female	Meningioma	1	Parietal lobe	Surgery	C
8	9	50	Female	Glioblastoma	4	Temporal lobe	Surgery + Radiation	C
11	12	48	Male	Meningioma	1	Frontal lobe	Surgery + Radiation	C
...	...	...	...	...	...	...	...	
1983	1984	59	Female	Meningioma	1	Parietal lobe	Surgery + Radiation	C
1986	1987	58	Male	Glioblastoma	4	Parietal lobe	Surgery + Chemotherapy	C
1990	1991	56	Female	Meningioma	1	Occipital lobe	Surgery + Chemotherapy	C
1993	1994	57	Male	Glioblastoma	4	Parietal lobe	Surgery + Radiation	C
1996	1997	57	Male	Glioblastoma	4	Occipital lobe	Surgery	C

561 rows × 11 columns

In [12]:
```
complete_response_and_not_null = df[(df['Treatment_Outcome'] == 'Complete resp
complete_response_and_not_null.head()
```

Out[12]:

	Patient_ID	Age	Gender	Tumor_Type	Tumor_Grade	Tumor_Location	Treatment	Trea
812	813	52	Female	Glioblastoma	4	Frontal lobe	Surgery + Chemotherapy	Co
828	829	52	Female	Glioblastoma	4	Frontal lobe	Surgery + Chemotherapy	Co
844	845	52	Female	Glioblastoma	4	Frontal lobe	Surgery + Chemotherapy	Co
860	861	52	Female	Glioblastoma	4	Frontal lobe	Surgery + Chemotherapy	Co
876	877	52	Female	Glioblastoma	4	Frontal lobe	Surgery + Chemotherapy	Co

the prelimanry hypothesis was wrong as some cases where treament was a complete response did experience regrowth of a tumor this was paticulary evident in the frontal lobe.

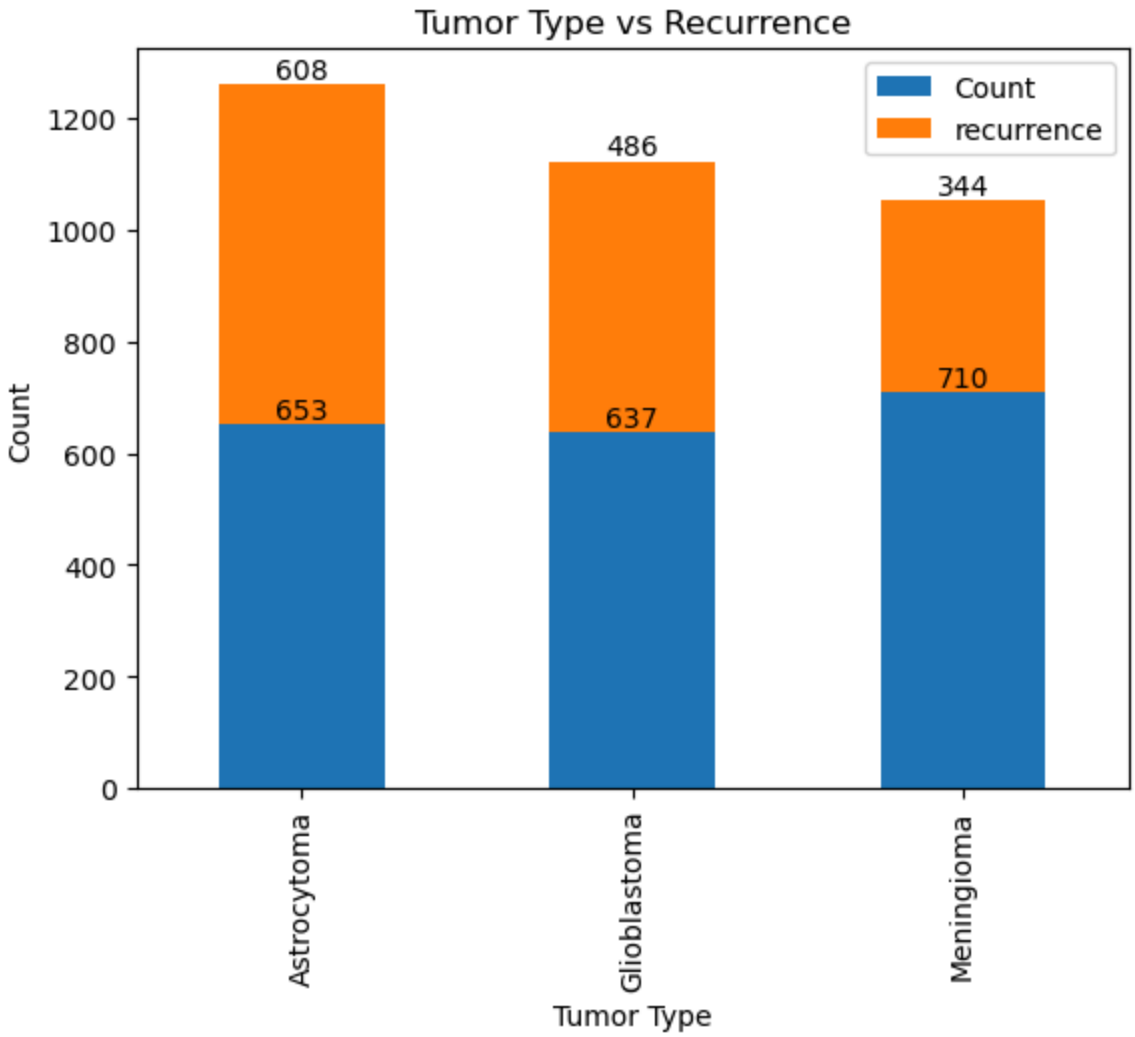
Look at Recurrence

In [15]:
```
# add aboolian value for reourence
```

```
# add a new column "reoccurrence" based on the condition
df['recurrence'] = ~df['Time_to_Recurrence_(months)'].isnull()

# set maximum number of rows to display
# pd.set_option('display.max_rows', None)
#df
print(df[['Tumor_Type', 'recurrence']])
```

```
        Tumor_Type  recurrence
0     Glioblastoma        True
1       Meningioma       False
2      Astrocytoma        True
3     Glioblastoma       False
4      Astrocytoma        True
...            ...         ...
1995    Meningioma        True
1996  Glioblastoma       False
1997    Meningioma        True
1998   Astrocytoma        True
1999  Glioblastoma        True

[2000 rows x 2 columns]
```

In [16]:
```
# Group by Tumor Type and count the occurrences, also count the occurrences of
grouped_df = df.groupby('Tumor_Type').agg({'Tumor_Type': 'count', 'recurrence'
grouped_df.columns = ['Count', 'recurrence']

# plot
ax = grouped_df.plot(kind='bar', stacked=True)
#grouped_df.plot(kind='bar', stacked=True)
plt.xlabel('Tumor Type')
plt.ylabel('Count')
plt.title('Tumor Type vs Recurrence')

# add actual numbers on top of the bars
for i in range(len(grouped_df)):
    ax.text(i, grouped_df.iloc[i]['Count'],
            str(int(grouped_df.iloc[i]['Count'])), ha='center', va='bottom')
    ax.text(i, grouped_df.iloc[i]['Count'] + grouped_df.iloc[i]['recurrence'],
            str(int(grouped_df.iloc[i]['recurrence'])), ha='center', va='botto
plt.show()
```



In [17]:
```
# calculate the sum of occurrences for each tumor type
tumor_type_count = df['Tumor_Type'].value_counts()

# calculate the sum of occurrences where recurrence is true for each tumor type
reoccurrence_true_count = df[df['Time_to_Recurrence_(months)'].notna()]['Tumor_

print("Sum of occurrences for each tumor type:")
print(tumor_type_count)
print("\nSum of occurrences where recurrence is true for each tumor type:")
print(reoccurrence_true_count)
```

```
Sum of occurrences for each tumor type:
Tumor_Type
Meningioma      710
Astrocytoma     653
Glioblastoma    637
Name: count, dtype: int64

Sum of occurrences where recurrence is true for each tumor type:
Tumor_Type
Astrocytoma     608
Glioblastoma    486
Meningioma      344
Name: count, dtype: int64
```

Cox Proportional Hazard Regression analysis

In [19]:
```
df['Treatment_Outcome'].unique()
```

Out[19]:
```
array(['Partial response', 'Complete response', 'Progressive disease',
       'Stable disease'], dtype=object)
```

In [20]:
```
#  dictionary to map each response to a numerical value
response_mapping = {
    'Partial response': 1,
    'Complete response': 2,
    'Progressive disease': 3,
    'Stable disease': 4
}

# create a new column with numerical values based on the mapping
df['Treatment_Outcome_numeric_map'] = df['Treatment_Outcome'].map(response_mapp

# display the DataFrame with the new column
df.head()
```

Out[20]:

	Patient_ID	Age	Gender	Tumor_Type	Tumor_Grade	Tumor_Location	Treatment	Treatm
0	1	45	Male	Glioblastoma	4	Frontal lobe	Surgery	Pa
1	2	55	Female	Meningioma	1	Parietal lobe	Surgery	Comp
2	3	60	Male	Astrocytoma	3	Occipital lobe	Surgery + Chemotherapy	Progr
3	4	50	Female	Glioblastoma	4	Temporal lobe	Surgery + Radiation therapy	Comp
4	5	65	Male	Astrocytoma	2	Frontal lobe	Surgery + Radiation therapy	Pa

In [21]:
```
from lifelines import CoxPHFitter

# load your dataset into a DataFrame (let's assume it's named df)

# drop rows with missing values for simplicity
df.dropna(inplace=True)

# fit the Cox Regression model
cph = CoxPHFitter()
cph.fit(df, duration_col='Survival_Time_(months)', formula='Age + Gender + Tum

# summarize the results
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'Survival_Time_(months)'
baseline estimation	breslow
number of observations	1438
number of events observed	1438
partial log-likelihood	-8849.41
time fit was run	2024-06-04 19:42:45 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%
Age	-0.00	1.00	0.00	-0.01	0.01	0.99	1.0
Gender[T.Male]	-0.03	0.97	0.06	-0.16	0.09	0.86	1.1
Tumor_Type[T.Glioblastoma]	0.42	1.53	0.16	0.11	0.73	1.12	2.0
Tumor_Type[T.Meningioma]	0.15	1.16	0.15	-0.14	0.44	0.87	1.5
Tumor_Grade	0.01	1.01	0.09	-0.16	0.19	0.85	1.2
Tumor_Location[T.Occipital lobe]	-0.27	0.76	0.09	-0.44	-0.10	0.64	0.9
Tumor_Location[T.Parietal lobe]	-0.08	0.92	0.08	-0.24	0.08	0.79	1.0
Tumor_Location[T.Temporal lobe]	-0.83	0.43	0.08	-1.00	-0.67	0.37	0.5
Treatment[T.Chemotherapy + Radiation]	0.05	1.05	0.72	-1.36	1.46	0.26	4.3
Treatment[T.Radiation]	0.47	1.60	0.17	0.14	0.80	1.15	2.2
Treatment[T.Surgery]	0.81	2.24	0.17	0.47	1.14	1.60	3.1
Treatment[T.Surgery + Chemotherapy]	0.63	1.87	0.12	0.39	0.86	1.48	2.3
Treatment[T.Surgery + Radiation]	0.54	1.72	0.12	0.30	0.78	1.36	2.1
Treatment[T.Surgery + Radiation therapy]	-1.36	0.26	1.02	-3.35	0.63	0.04	1.8
Treatment_Outcome[T.Partial response]	-1.30	0.27	0.21	-1.71	-0.89	0.18	0.4
Treatment_Outcome[T.Progressive disease]	-0.84	0.43	0.21	-1.24	-0.44	0.29	0.6
Treatment_Outcome[T.Stable disease]	-0.96	0.38	0.21	-1.37	-0.56	0.25	0.5

Concordance	0.67
Partial AIC	17732.82
log-likelihood ratio test	345.71 on 17 df
-log2(p) of ll-ratio test	207.34

Age: Age does not significantly affect survival time (p = 0.86).

Gender: Being male is not significantly associated with survival time compared to being female (p = 0.63).

Tumor Type: Patients with Glioblastoma have a significantly higher hazard ratio (HR = 1.53, p < 0.005) compared to other tumor types.

Tumor Grade: Tumor grade does not significantly affect survival time (p = 0.87).

Tumor Location: Tumor location significantly affects survival time. Specifically, tumors located in the temporal lobe have the lowest hazard ratio (HR = 0.43, p < 0.005) compared to other locations.

Treatment: Different treatments have varying effects on survival time: Surgery alone (HR = 2.24, p < 0.005) and Surgery combined with Chemotherapy (HR = 1.87, p < 0.005) or Radiation (HR = 1.72, p < 0.005) are associated with significantly higher hazard ratios compared to no treatment. Treatment with Radiation alone (HR = 1.60, p < 0.005) also leads to a higher hazard ratio. Notably, treatment with Surgery combined with Radiation therapy has a significantly lower hazard ratio (HR = 0.26, p = 0.18) compared to other treatments.

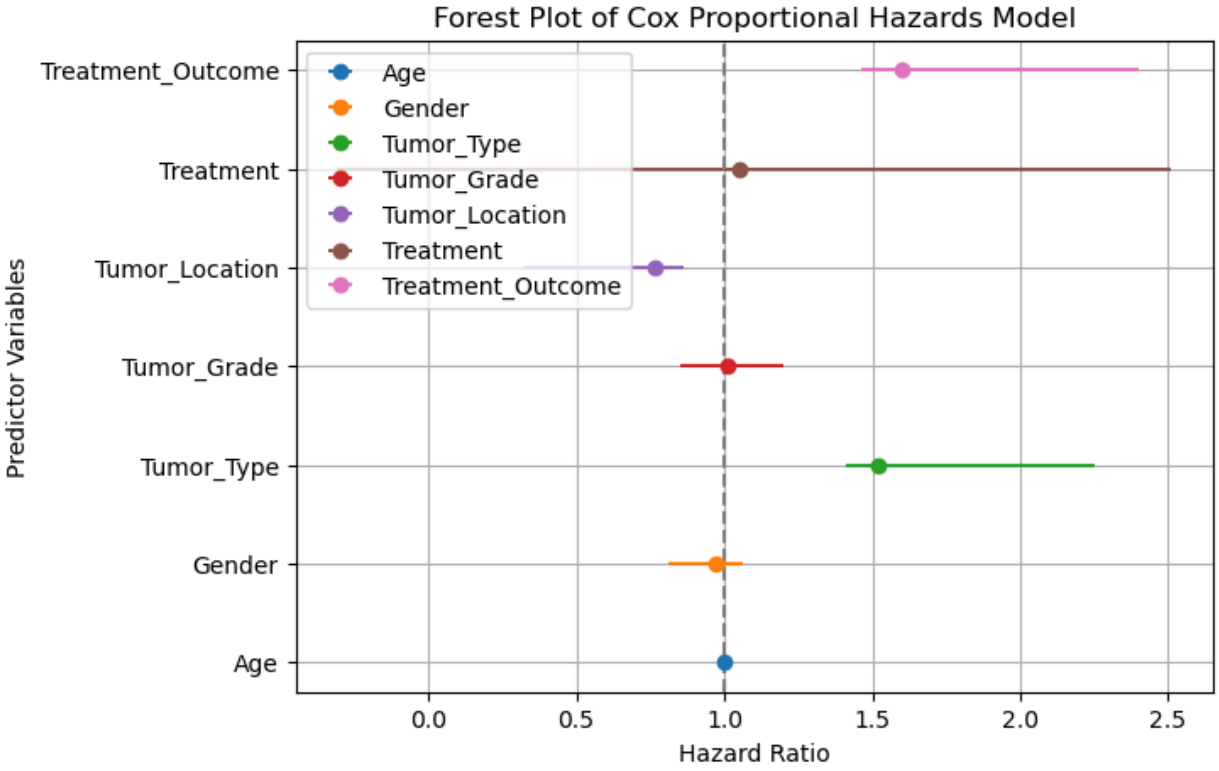
Treatment Outcome: Treatment outcomes significantly impact survival time: Patients with Partial response (HR = 0.27, p < 0.005), Progressive disease (HR = 0.43, p < 0.005), or Stable disease (HR = 0.38, p < 0.005) have lower hazard ratios compared to patients with no response.

Overall, the results suggest that tumor type, tumor location, treatment type, and treatment outcome are significant predictors of survival time, while age, gender, and tumor grade do not have significant effects.

In [ ]:

In [61]:
```
import numpy as np

#  predictor variables
predictors = ['Age', 'Gender', 'Tumor_Type', 'Tumor_Grade', 'Tumor_Location',

#  coefficients, lower and upper bounds
coefficients = [0, -0.03, 0.42, 0.01, -0.27, 0.05, 0.47, 0.81, 0.63, 0.54, -1.
lower_bound = [-0.01, -0.16, 0.11, -0.16, -0.44, -1.36, 0.14, 0.47, 0.39, 0.3,
upper_bound = [0.01, 0.09, 0.73, 0.19, -0.1, 1.46, 0.8, 1.14, 0.86, 0.78, 0.63

#  hazard ratios
hazard_ratios = np.exp(coefficients)

# set up the plot
plt.figure(figsize=(7, 5))

# plot each predictor variable with its coefficient and confidence interval
for i, predictor in enumerate(predictors):
    xerr = np.abs([[lower_bound[i]], [upper_bound[i]]])  # Adjusted xerr to en
```

```
    plt.errorbar(hazard_ratios[i], i, xerr=xerr, fmt='o', label=predictor)

# add vertical line at hazard ratio of 1
plt.axvline(x=1, linestyle='--', color='gray')

# customize y-axis labels
plt.yticks(range(len(predictors)), predictors)

# add labels and title
plt.xlabel('Hazard Ratio')
plt.ylabel('Predictor Variables')
plt.title('Forest Plot of Cox Proportional Hazards Model')

# add legend
plt.legend()

# plot
plt.grid(True)
plt.show()
```



Tesing other Models

In [25]:
```
df.columns
```

Out[25]:
```
Index(['Patient_ID', 'Age', 'Gender', 'Tumor_Type', 'Tumor_Grade',
       'Tumor_Location', 'Treatment', 'Treatment_Outcome',
       'Time_to_Recurrence_(months)', 'Recurrence_Site',
       'Survival_Time_(months)', 'recurrence',
       'Treatment_Outcome_numeric_map'],
      dtype='object')
```

In [48]:
```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
```

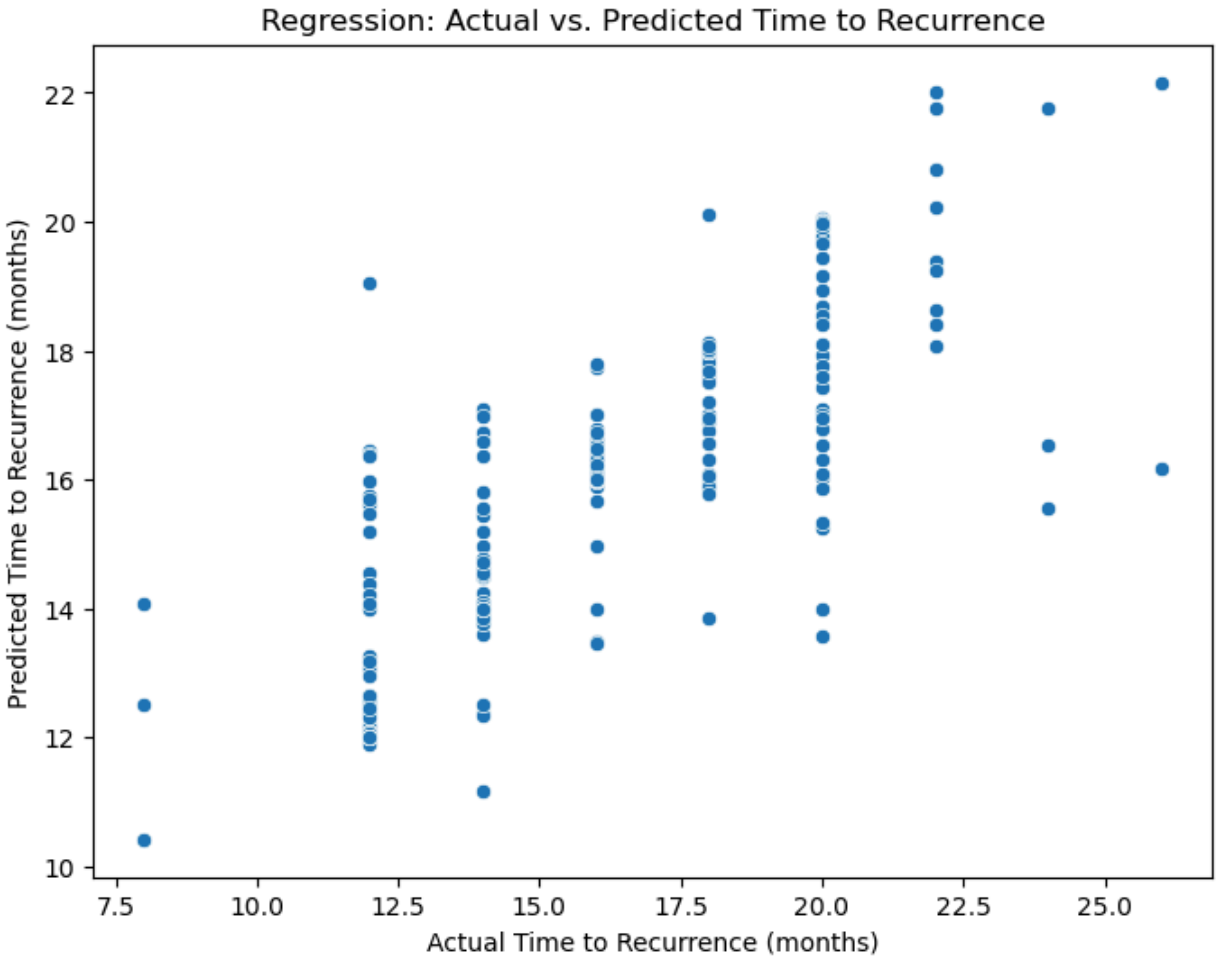
```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.pipeline import Pipeline


# separate features and target variables
X = df.drop(['Time_to_Recurrence_(months)', 'Recurrence_Site', 'Survival_Time_
y_time_to_recurrence = df['Time_to_Recurrence_(months)']
y_recurrence_site = df['Recurrence_Site']
y_survival_time = df['Survival_Time_(months)']

# split the data into training and testing sets
X_train, X_test, y_train_time_to_recurrence, y_test_time_to_recurrence = train_
X_train, X_test, y_train_recurrence_site, y_test_recurrence_site = train_test_s
X_train, X_test, y_train_survival_time, y_test_survival_time = train_test_spli

# preprocessing
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Model  predicting Time_to_Recurrence_(months)
model_time_to_recurrence = Pipeline(steps=[('preprocessor', preprocessor),
                                           ('regressor', RandomForestRegresso
model_time_to_recurrence.fit(X_train, y_train_time_to_recurrence)
y_pred_time_to_recurrence = model_time_to_recurrence.predict(X_test)
mse_time_to_recurrence = mean_squared_error(y_test_time_to_recurrence, y_pred_
print("Mean Squared Error for Time_to_Recurrence_(months):", mse_time_to_recur

# model for predicting Recurrence_Site
model_recurrence_site = Pipeline(steps=[('preprocessor', preprocessor),
                                        ('classifier', RandomForestClassifier(
model_recurrence_site.fit(X_train, y_train_recurrence_site)
y_pred_recurrence_site = model_recurrence_site.predict(X_test)
accuracy_recurrence_site = accuracy_score(y_test_recurrence_site, y_pred_recur
print("Accuracy Score for Recurrence_Site:", accuracy_recurrence_site)

# model for predicting Survival_Time_(months)
model_survival_time = Pipeline(steps=[('preprocessor', preprocessor),
                                      ('regressor', RandomForestRegressor())])
model_survival_time.fit(X_train, y_train_survival_time)
y_pred_survival_time = model_survival_time.predict(X_test)
```

```
mse_survival_time = mean_squared_error(y_test_survival_time, y_pred_survival_t
print("Mean Squared Error for Survival_Time_(months):", mse_survival_time)
```

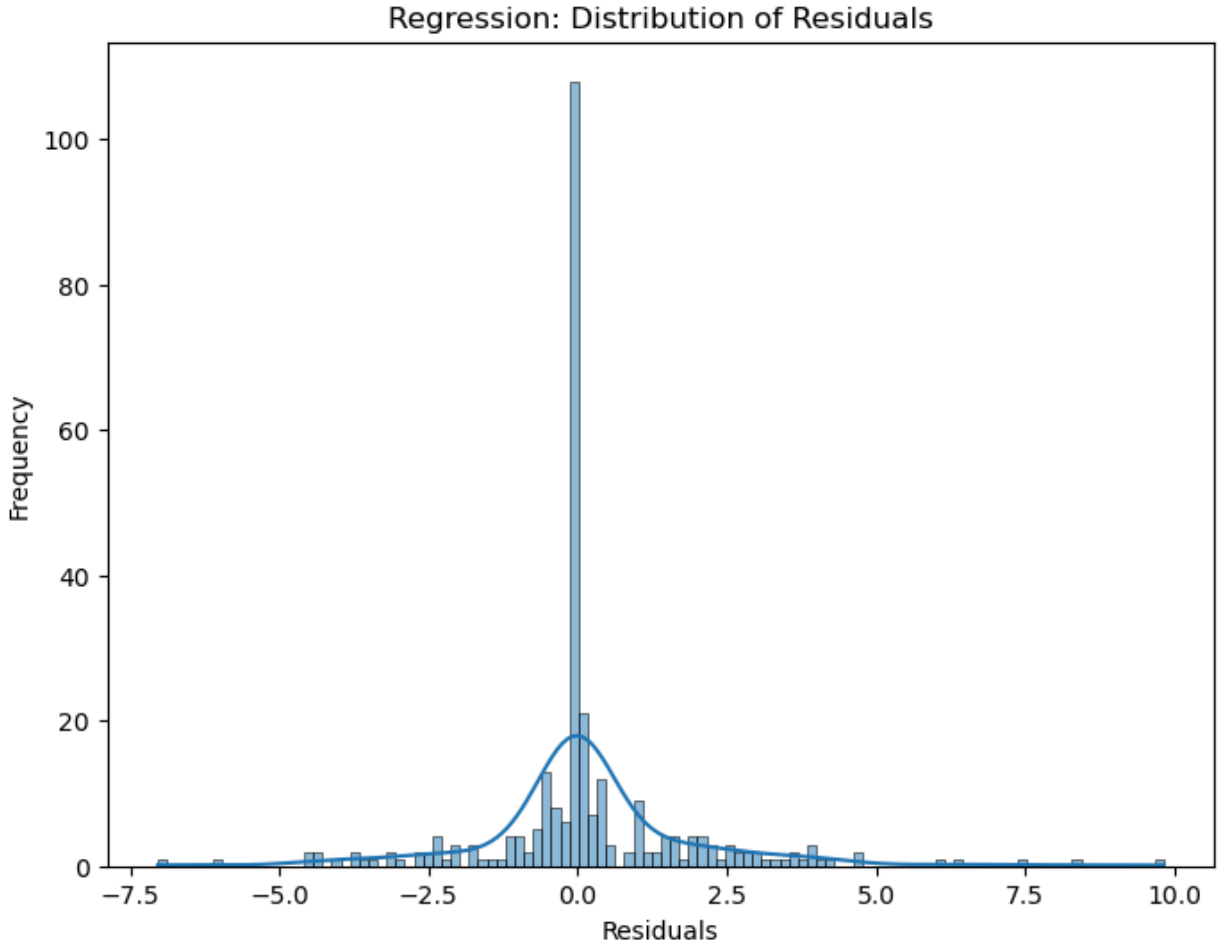
```
Mean Squared Error for Time_to_Recurrence_(months): 3.5700246527777773
Accuracy Score for Recurrence_Site: 0.9965277777777778
Mean Squared Error for Survival_Time_(months): 14.445002083333337
```

In [54]:

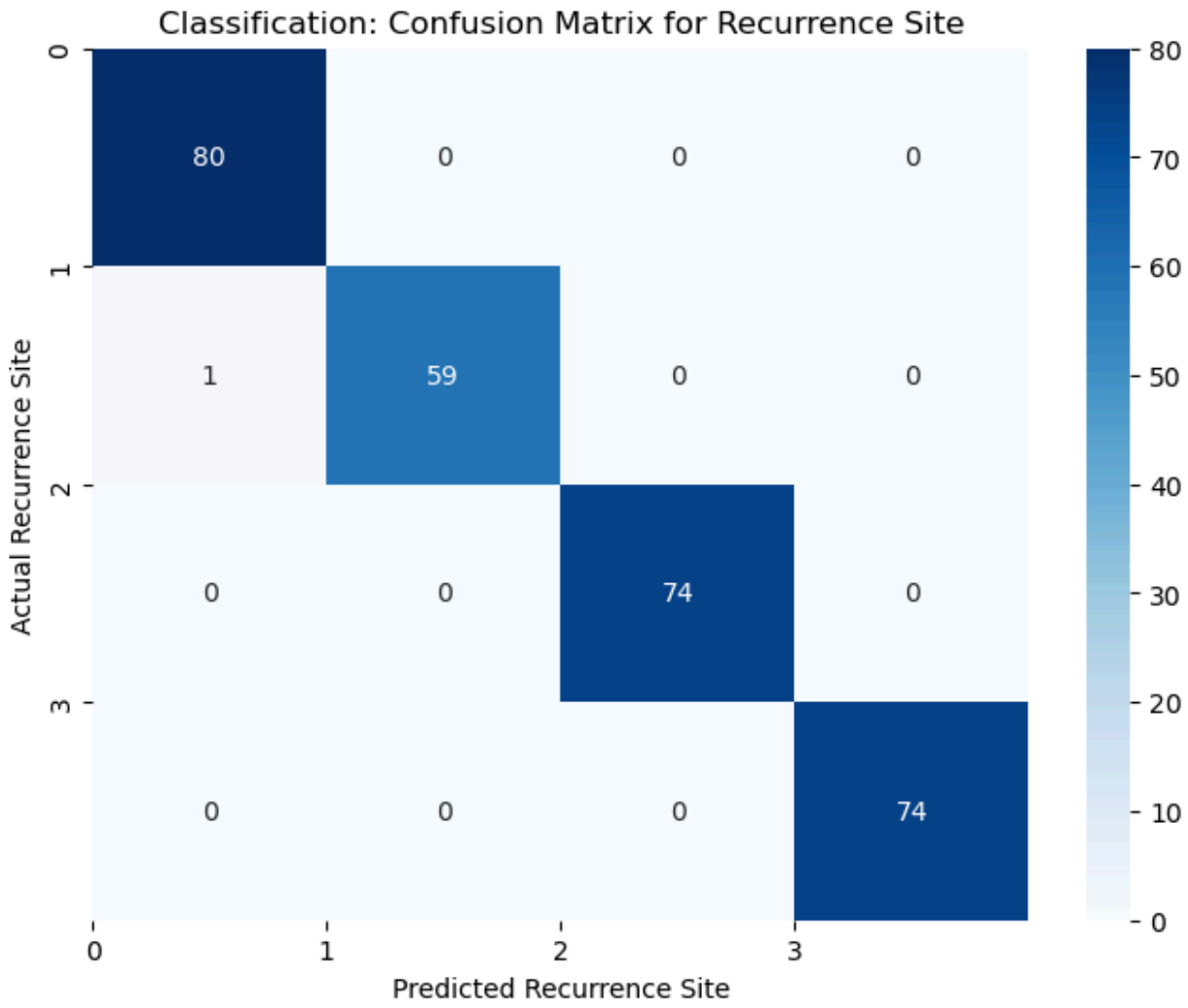
```
import matplotlib.pyplot as plt
import seaborn as sns

# scatter plot comparing actual vs. predicted values of Time_to_Recurrence_(mo
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test_time_to_recurrence, y=y_pred_time_to_recurrence)
plt.xlabel('Actual Time to Recurrence (months)')
plt.ylabel('Predicted Time to Recurrence (months)')
plt.title('Regression: Actual vs. Predicted Time to Recurrence')
plt.show()

# distribution plot of residuals
residuals_time_to_recurrence = y_test_time_to_recurrence - y_pred_time_to_recu
plt.figure(figsize=(8, 6))
sns.histplot(residuals_time_to_recurrence, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Regression: Distribution of Residuals')
plt.show()
```

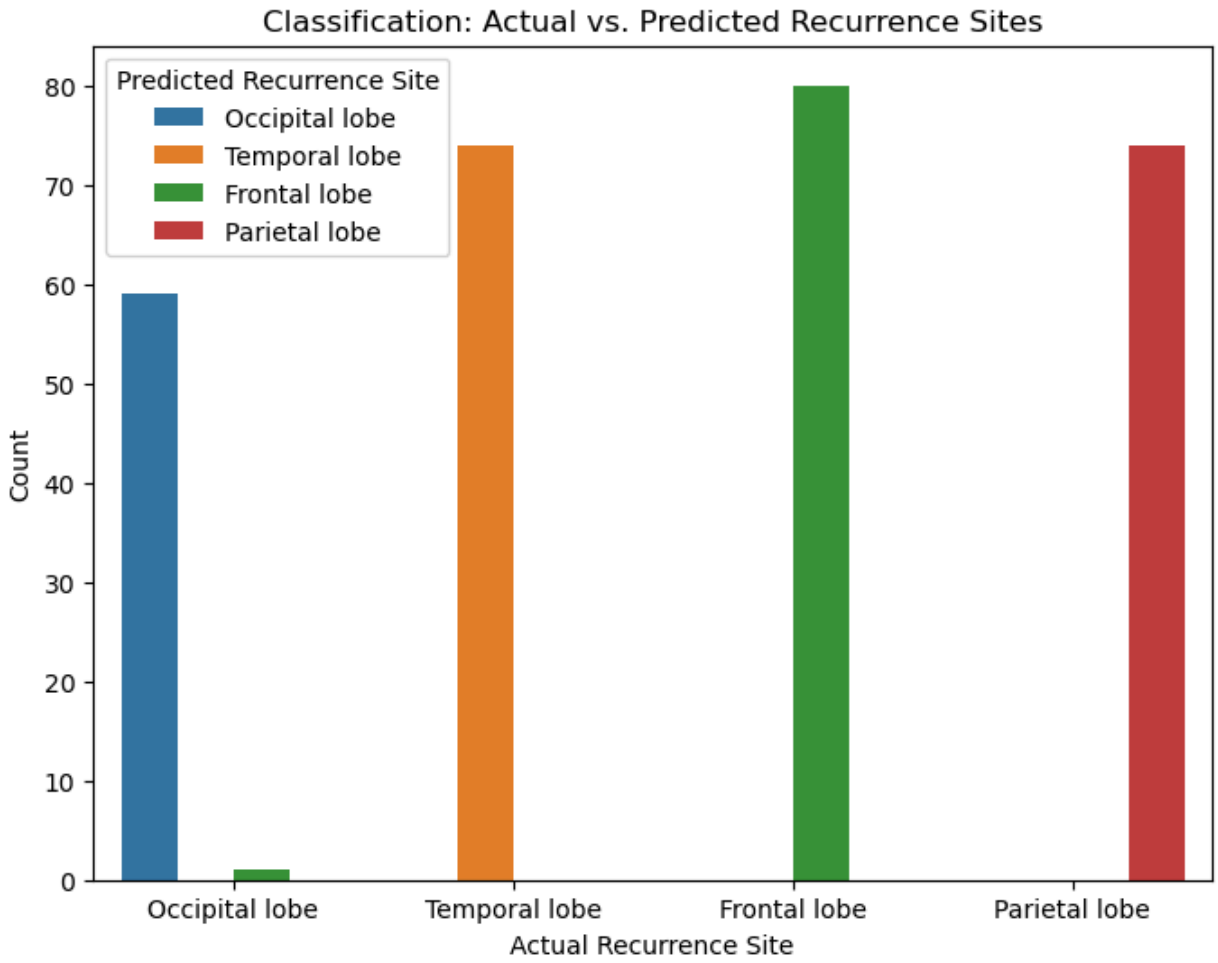


```
/Users/cantu/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a fut
ure version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

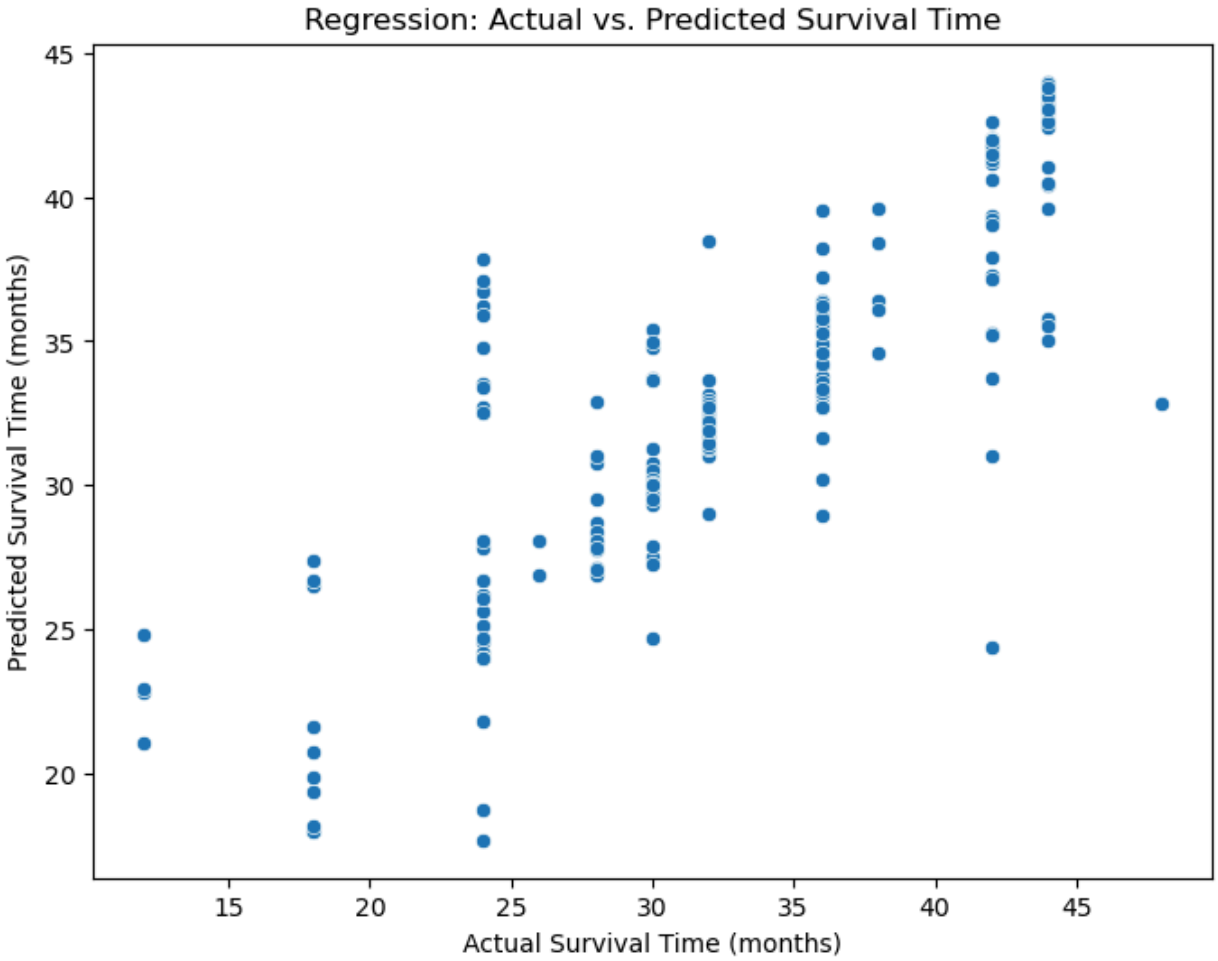


In [52]:
```python
from sklearn.metrics import confusion_matrix
import numpy as np

# confusion matrix
conf_matrix = confusion_matrix(y_test_recurrence_site, y_pred_recurrence_site)

# Plot  matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Recurrence Site')
plt.ylabel('Actual Recurrence Site')
plt.title('Classification: Confusion Matrix for Recurrence Site')
plt.xticks(ticks=np.arange(len(conf_matrix)), labels=np.arange(len(conf_matrix
plt.yticks(ticks=np.arange(len(conf_matrix)), labels=np.arange(len(conf_matrix
plt.show()

# Bar chart showing distribution of actual vs. predicted recurrence sites
plt.figure(figsize=(8, 6))
sns.countplot(x=y_test_recurrence_site, hue=y_pred_recurrence_site)
plt.xlabel('Actual Recurrence Site')
plt.ylabel('Count')
plt.title('Classification: Actual vs. Predicted Recurrence Sites')
plt.legend(title='Predicted Recurrence Site')
plt.show()
```


Classification: Confusion Matrix for Recurrence Site
Actual Recurrence Site
Predicted Recurrence Site
0
1
2
3
80
0
0
0
1
59
0
0
0
0
74
0
0
0
0
74
80
70
60
50
40
30
20
10
0



In [56]:
```
# scatter plot comparing actual vs. predicted values of Survival_Time_(months)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test_survival_time, y=y_pred_survival_time)
plt.xlabel('Actual Survival Time (months)')
plt.ylabel('Predicted Survival Time (months)')
plt.title('Regression: Actual vs. Predicted Survival Time')
plt.show()
```


Regression: Actual vs. Predicted Survival Time
Predicted Survival Time (months)
Actual Survival Time (months)
45
40
35
30
25
20
15
20
25
30
35
40
45

In [ ]: