# breast_cancer_classification

August 31, 2024

## 1 BREAST CANCER CLASSIFICATION PROJECT

### 1.1 Business Problem

Death among women has various factors and one of the leading causes being `Breast Cancer`. A good way of improving the survival rates and giving out personalized treatments depends on whether or not it can be detected early or predicted in advance. The intricacy and diversity of patient data, including age, tumor size, hormone receptor status, and staging, can make it difficult for medical professionals to assess the severity and course of the illness.

A hospital that deals with breast cancer patients want a solution that can be used to classify patients based on their breast cancer characteristics and tell if they are likely to experience a more aggressive cancer. then secondly they would like to be able to predict a patients survival time based oon various factors both medical and demographic. The company can increase the precision of diagnosis and treatment strategies by using machine learning to predict survival months and classify the cancer status (e.g., aggressive vs. non-aggressive). The ultimate goal is to increase survival rates, reduce patient suffering, and optimize the overall cost-effectiveness of breast cancer care.

### 1.2 Business Understanding

#### 1.2.1 Objectives

1. Create a predictive model to estimate survival time based on medical and demographic factors.

2. Develop a machine learning model to classify patients into categories such as aggressive vs. non-aggressive cancer.

### 1.2.2 Stakeholders:

1. Patients: Directly affected by the treatment and survival outcomes.
2. Medical Professionals: Need accurate tools to make informed treatment decisions.
3. Hospital Administrators: Focused on improving patient outcomes and optimizing operational costs.

Assumption: The quality and completeness of the data provided are sufficient for building accurate models.

## 2 Step 1: Data Exploration

```
[1]: import pandas as pd
     import numpy as np
     from sklearn.preprocessing import StandardScaler
     import statsmodels.api as sm
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report,accuracy_score,␣
       ↪confusion_matrix, roc_curve, auc, roc_auc_score
     from sklearn.tree import DecisionTreeClassifier
     from imblearn.over_sampling import SMOTE
     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline
```

```
[2]: #Load the dataset
     data = pd.read_csv('Breast_Cancer.csv')
     data
```

```
[2]:       Age   Race Marital Status T Stage  N Stage 6th Stage  \
     0      68  White        Married       T1      N1       IIA
     1      50  White        Married       T2      N2      IIIA
     2      58  White       Divorced       T3      N3      IIIC
     3      58  White        Married       T1      N1       IIA
     4      47  White        Married       T2      N1       IIB
     …     …     …             …       …       …        …
     4019   62  Other        Married       T1      N1       IIA
     4020   56  White       Divorced       T2      N2      IIIA
     4021   68  White        Married       T2      N1       IIB
     4022   58  Black       Divorced       T2      N1       IIB
     4023   46  White        Married       T2      N1       IIB

                     differentiate Grade   A Stage  Tumor Size Estrogen Status  \
     0         Poorly differentiated     3  Regional           4        Positive
```

```
1     Moderately differentiated     2  Regional          35        Positive
2     Moderately differentiated     2  Regional          63        Positive
3         Poorly differentiated     3  Regional          18        Positive
4         Poorly differentiated     3  Regional          41        Positive
…                          …   …        …         …             …
4019  Moderately differentiated     2  Regional           9        Positive
4020  Moderately differentiated     2  Regional          46        Positive
4021  Moderately differentiated     2  Regional          22        Positive
4022  Moderately differentiated     2  Regional          44        Positive
4023  Moderately differentiated     2  Regional          30        Positive

     Progesterone Status  Regional Node Examined  Reginol Node Positive  \
0               Positive                      24                      1
1               Positive                      14                      5
2               Positive                      14                      7
3               Positive                       2                      1
4               Positive                       3                      1
…                    …                      …                     …
4019            Positive                       1                      1
4020            Positive                      14                      8
4021            Negative                      11                      3
4022            Positive                      11                      1
4023            Positive                       7                      2

      Survival Months Status
0                  60  Alive
1                  62  Alive
2                  75  Alive
3                  84  Alive
4                  50  Alive
…                 …    …
4019               49  Alive
4020               69  Alive
4021               69  Alive
4022               72  Alive
4023              100  Alive

[4024 rows x 16 columns]
```

[3]:
```
#Look at the basic info of the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4024 entries, 0 to 4023
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     4024 non-null   int64
 1   Race                    4024 non-null   object
 2   Marital Status          4024 non-null   object
 3   T Stage                 4024 non-null   object
 4   N Stage                 4024 non-null   object
 5   6th Stage               4024 non-null   object
 6   differentiate           4024 non-null   object
 7   Grade                   4024 non-null   object
 8   A Stage                 4024 non-null   object
 9   Tumor Size              4024 non-null   int64
 10  Estrogen Status         4024 non-null   object
 11  Progesterone Status     4024 non-null   object
 12  Regional Node Examined  4024 non-null   int64
 13  Reginol Node Positive   4024 non-null   int64
 14  Survival Months         4024 non-null   int64
 15  Status                  4024 non-null   object
dtypes: int64(5), object(11)
memory usage: 503.1+ KB
```

[4]:
```
#Check for missing values
data.isnull().sum()
```

[4]:
```
Age                       0
Race                      0
Marital Status            0
T Stage                   0
N Stage                   0
6th Stage                 0
differentiate             0
Grade                     0
A Stage                   0
Tumor Size                0
Estrogen Status           0
Progesterone Status       0
Regional Node Examined    0
Reginol Node Positive     0
Survival Months           0
Status                    0
dtype: int64
```

[5]:
```
#Check for any duplications in the dataset
data.duplicated().sum()
```

[5]:
```
1
```

[6]:
```
#Drop the duplicate values found
data.drop_duplicates(inplace=True)
#check if there still any duplicates left
data.duplicated().sum()
```

```
[6]: 0
```

```
[7]: #Look at the total number of unique values within the dataset
     for column in data.columns:
         print(f'{column} has {data[column].nunique()} unique values')
```

```
Age has 40 unique values
Race has 3 unique values
Marital Status has 5 unique values
T Stage  has 4 unique values
N Stage has 3 unique values
6th Stage has 5 unique values
differentiate has 4 unique values
Grade has 4 unique values
A Stage has 2 unique values
Tumor Size has 110 unique values
Estrogen Status has 2 unique values
Progesterone Status has 2 unique values
Regional Node Examined has 54 unique values
Reginol Node Positive has 38 unique values
Survival Months has 107 unique values
Status has 2 unique values
```

```
[8]: data.describe()
```

```
[8]:                Age   Tumor Size  Regional Node Examined  \
     count  4023.000000  4023.000000             4023.000000
     mean     53.969923    30.477007               14.358439
     std       8.963118    21.121253                8.100241
     min      30.000000     1.000000                1.000000
     25%      47.000000    16.000000                9.000000
     50%      54.000000    25.000000               14.000000
     75%      61.000000    38.000000               19.000000
     max      69.000000   140.000000               61.000000

            Reginol Node Positive  Survival Months
     count            4023.000000      4023.000000
     mean                4.158837        71.301765
     std                 5.109724        22.923009
     min                 1.000000         1.000000
     25%                 1.000000        56.000000
     50%                 2.000000        73.000000
     75%                 5.000000        90.000000
     max                46.000000       107.000000
```

## 3 Step 2: Exploratory Data Analysis

[9]:
```
#here we are going to group the data to help us later on when performing␣
↪regression and classification
numerical_cols = ['Age', 'Tumor Size', 'Regional Node Examined', 'Reginol Node␣
↪Positive', 'Survival Months']
categorical_cols = ['Race', 'Marital Status', 'T Stage ', 'N Stage', '6th␣
↪Stage', 'differentiate', 'Grade', 'A Stage', 'Estrogen Status',␣
↪'Progesterone Status', 'Status']
```

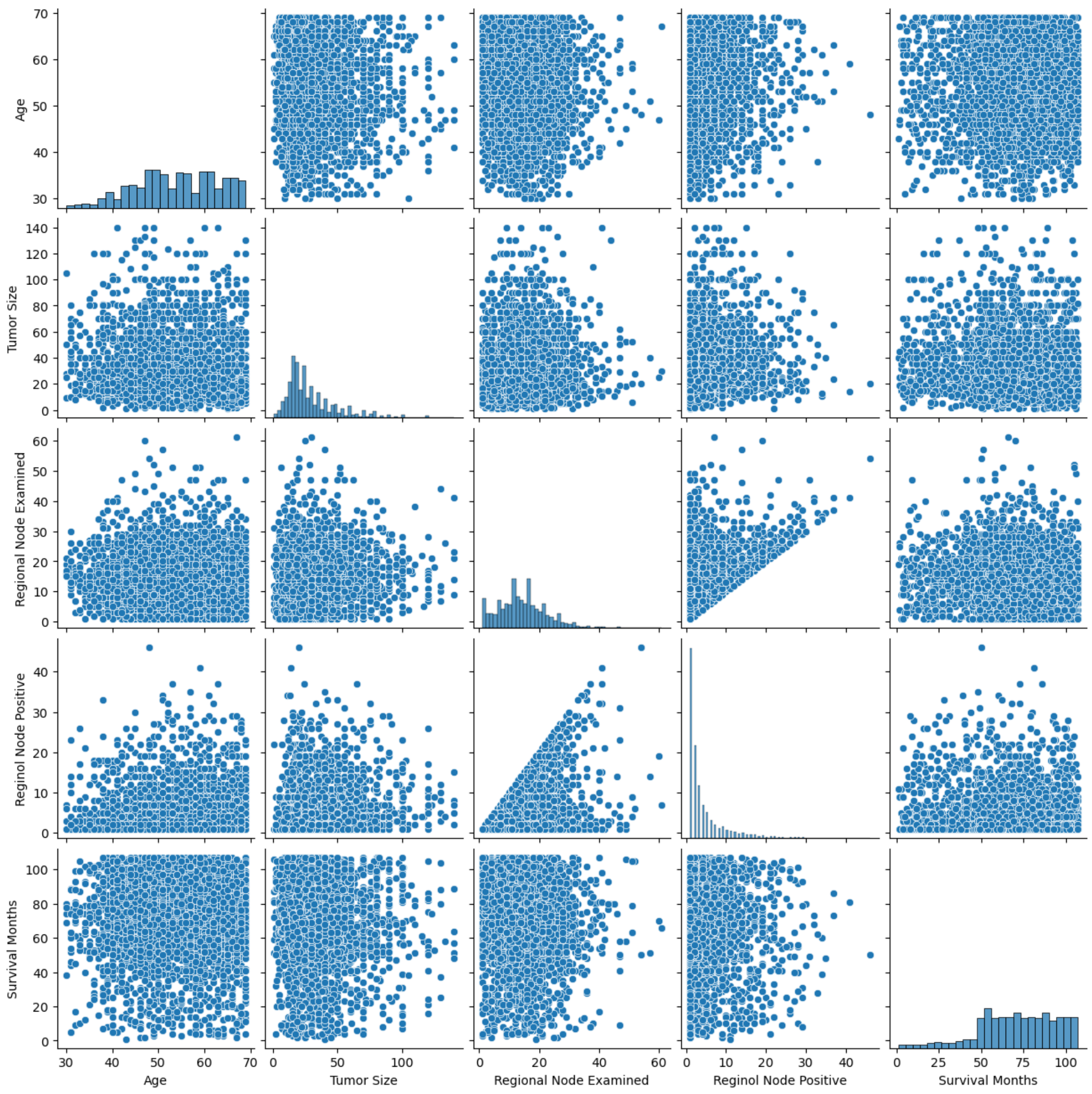
### 3.1 Linear Regression

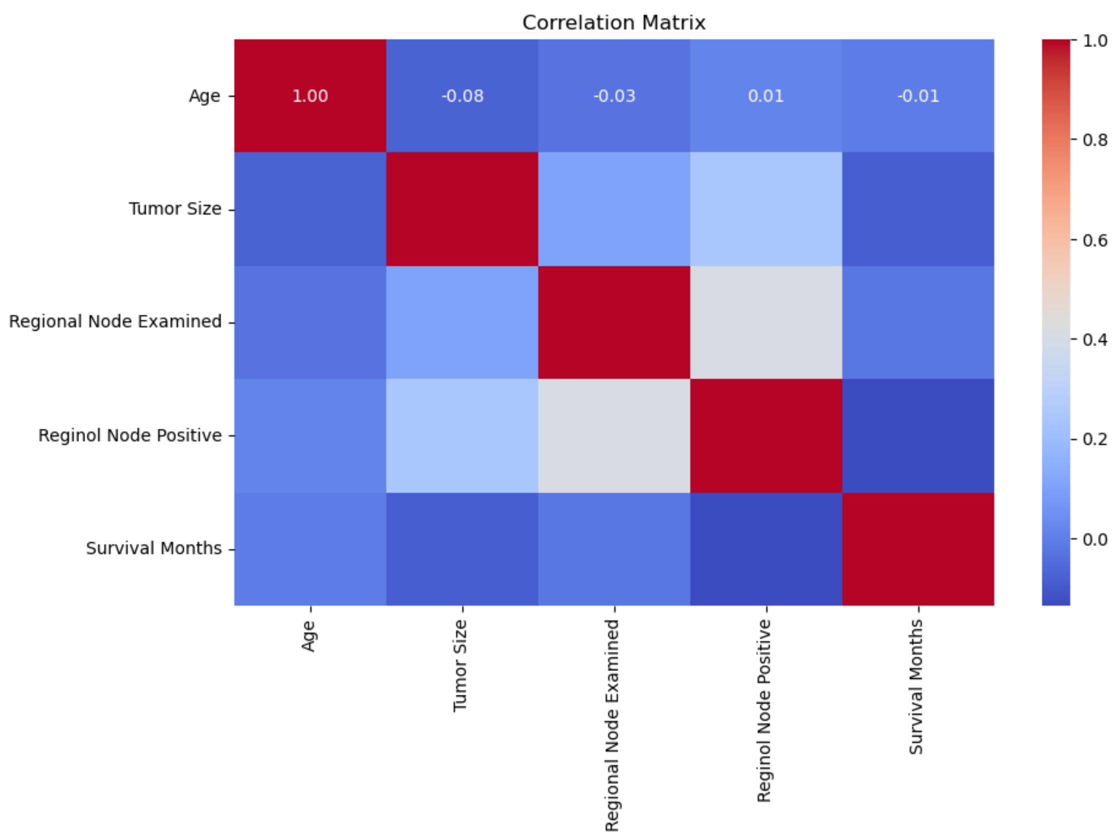
for this part we want to create a simple model that will see the numerical data being fitted into a linear regression in order to predict the number of survival months

[10]:
```
#Use the numerical_cols as the dataframe to use for creating the Linear␣
↪regression
data_numerals = data[numerical_cols]
#create a pairplot for the dataframe
sns.pairplot(data_numerals)
plt.suptitle('Pair Plot of data_numerals Columns', y=1.02)
plt.show()
# Correlation Matrix
plt.figure(figsize=(10, 6))
sns.heatmap(data_numerals.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

```
c:\Users\User\.conda\envs\learn-env\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\User\.conda\envs\learn-env\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\User\.conda\envs\learn-env\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\User\.conda\envs\learn-env\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\User\.conda\envs\learn-env\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Pair Plot of data_numerals Columns





```
[11]: #Initialize the scaler
      scaler = StandardScaler()

      #Select the feature and target variables
      #define the target
      y_lin = data_numerals['Survival Months']

      #fit and transform the features
      X_lin = scaler.fit_transform(data_numerals.drop('Survival Months', axis = 1))
```

```
[12]: #Now let's create a linear regression model that will cater for the numerical␣
       ↪data that we have in the data set
      lg_model = sm.OLS(y_lin, sm.add_constant(X_lin))
      lg_results = lg_model.fit()
      lg_results.summary()
```

[12]:

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Survival Months | **R-squared:** | 0.023 |
| **Model:** | OLS | **Adj. R-squared:** | 0.022 |
| **Method:** | Least Squares | **F-statistic:** | 23.57 |
| **Date:** | Sat, 31 Aug 2024 | **Prob (F-statistic):** | 2.73e-19 |
| **Time:** | 04:04:39 | **Log-Likelihood:** | -18262. |
| **No. Observations:** | 4023 | **AIC:** | 3.653e+04 |
| **Df Residuals:** | 4018 | **BIC:** | 3.657e+04 |
| **Df Model:** | 4 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P> \|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 71.3018 | 0.357 | 199.491 | 0.000 | 70.601 | 72.003 |
| **x1** | -0.2444 | 0.359 | -0.681 | 0.496 | -0.948 | 0.459 |
| **x2** | -1.3458 | 0.370 | -3.641 | 0.000 | -2.071 | -0.621 |
| **x3** | 0.9211 | 0.393 | 2.347 | 0.019 | 0.152 | 1.691 |
| **x4** | -3.1517 | 0.402 | -7.832 | 0.000 | -3.941 | -2.363 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 169.896 | **Durbin-Watson:** | 1.987 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 191.623 |
| **Skew:** | -0.535 | **Prob(JB):** | 2.45e-42 |
| **Kurtosis:** | 2.980 | **Cond. No.** | 1.66 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

1. Model Summary Dep. Variable (Dependent Variable): Survival Months

The variable that the model is trying to predict. R-squared: 0.023

R-squared is the proportion of the variance in the dependent variable that is predictable from the independent variables. An R-squared of 0.023 means that only 2.3% of the variability in Survival Months is explained by the model. This is a very low value, indicating that the model doesn't explain much of the variability in survival months. Adj. R-squared: 0.022

The adjusted R-squared accounts for the number of predictors in the model and adjusts for the potential overfitting. It's slightly lower than the R-squared, indicating that adding more predictors didn't improve the model much. F-statistic: 23.57

The F-statistic tests whether at least one of the predictors is statistically significant. A higher F-statistic indicates that the model is a better fit than a model with no predictors. Prob (F-statistic): 2.73e-19

This is the p-value associated with the F-statistic. A very small p-value (close to zero) indicates that the model as a whole is statistically significant.

2. Coefficients and Statistical Significance

Const (Intercept): 71.3018

This is the expected value of Survival Months when all independent variables are zero. The constant term is highly significant with a p-value of 0.000.

x1, x2, x3, x4 (Independent Variables):

x1 Coefficient: -0.2444

The relationship between x1 and Survival Months is negative, but with a p-value of 0.496, it is not statistically significant. This suggests that x1 does not have a meaningful impact on Survival Months.

x2 Coefficient: -1.3458

The relationship between x2 and Survival Months is negative and statistically significant (p-value = 0.000). For every unit increase in x2, Survival Months decreases by approximately 1.35 months.

x3 Coefficient: 0.9211

The relationship between x3 and Survival Months is positive and statistically significant (p-value = 0.019). This suggests that as x3 increases, Survival Months increases by approximately 0.92 months.

x4 Coefficient: -3.1517

The relationship between x4 and Survival Months is strongly negative and highly significant (p-value = 0.000). For every unit increase in x4, Survival Months decreases by approximately 3.15 months.

Standard Error (std err):

Reflects the accuracy of the coefficient estimates. Smaller standard errors indicate more precise estimates.

t-statistic (t) and P>|t| (p-value):

The t-statistic tests whether a coefficient is significantly different from zero. The p-value tells you the probability that the coefficient is actually zero (i.e., not significant).

Coefficients with p-values less than 0.05 are generally considered statistically significant.

The condition number tests for multicollinearity (high correlation between independent variables). A value below 30 suggests no serious multicollinearity issues.

Conclusion

Model Fit: The R-squared and adjusted R-squared values are low, indicating that the model does not explain much of the variability in Survival Months.

Significance: While the model overall is statistically significant (based on the F-statistic), not all variables are contributing meaningfully (e.g., x1 is not significant).

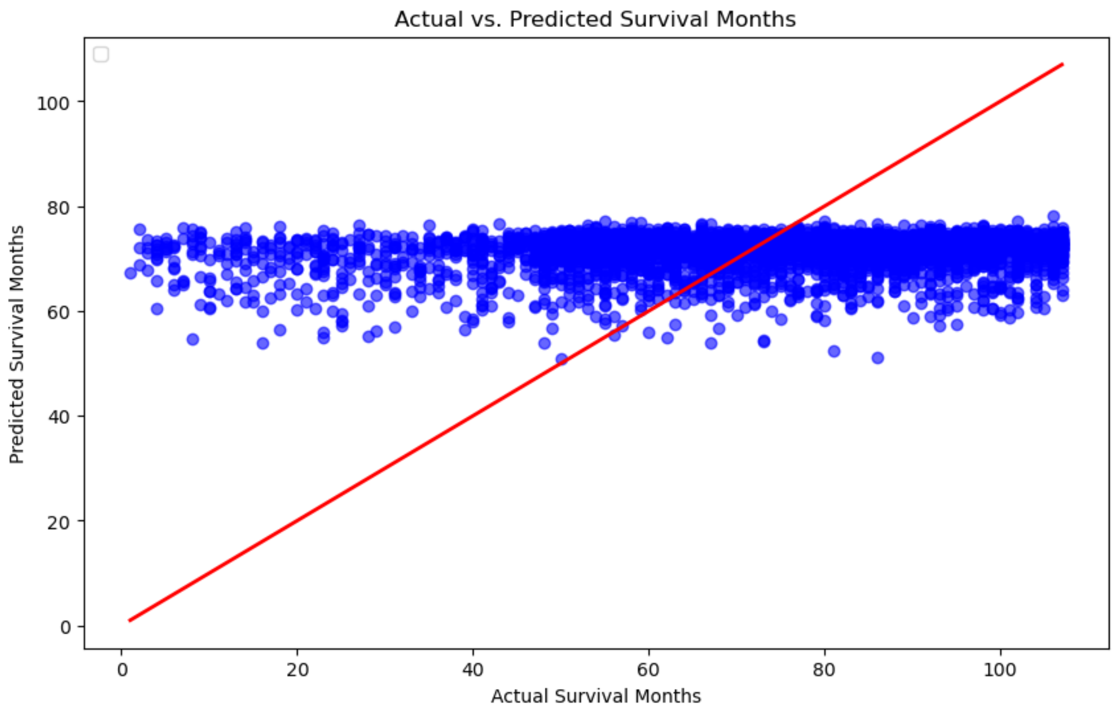
Coefficients: Some variables (x2, x3, x4) have significant coefficients, suggesting they do impact survival months, but the effect sizes vary.

Assumptions: The normality tests suggest that the residuals may not be normally distributed, which could affect the validity of the model's inferences.

Given these results, the model might require refinement, such as exploring non-linear relationships, interaction effects, or considering other variables that might better explain the variation in survival months.

[13]:
```python
# Predicted values
y_pred = lg_results.predict(sm.add_constant(X_lin))

plt.figure(figsize=(10, 6))

plt.scatter(y_lin, y_pred, color='blue', alpha=0.6)
plt.plot([y_lin.min(), y_lin.max()], [y_lin.min(), y_lin.max()], color='red',␣
 ↪linewidth=2)
plt.xlabel('Actual Survival Months')
plt.ylabel('Predicted Survival Months')
plt.title('Actual vs. Predicted Survival Months')
plt.legend(loc='upper left')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



Interpretation:

Points on the Red Line: Data points on this line indicate perfect predictions.

Points Above the Red Line: Data points above the line indicate overestimations by the model (the predicted survival is higher than the actual survival).

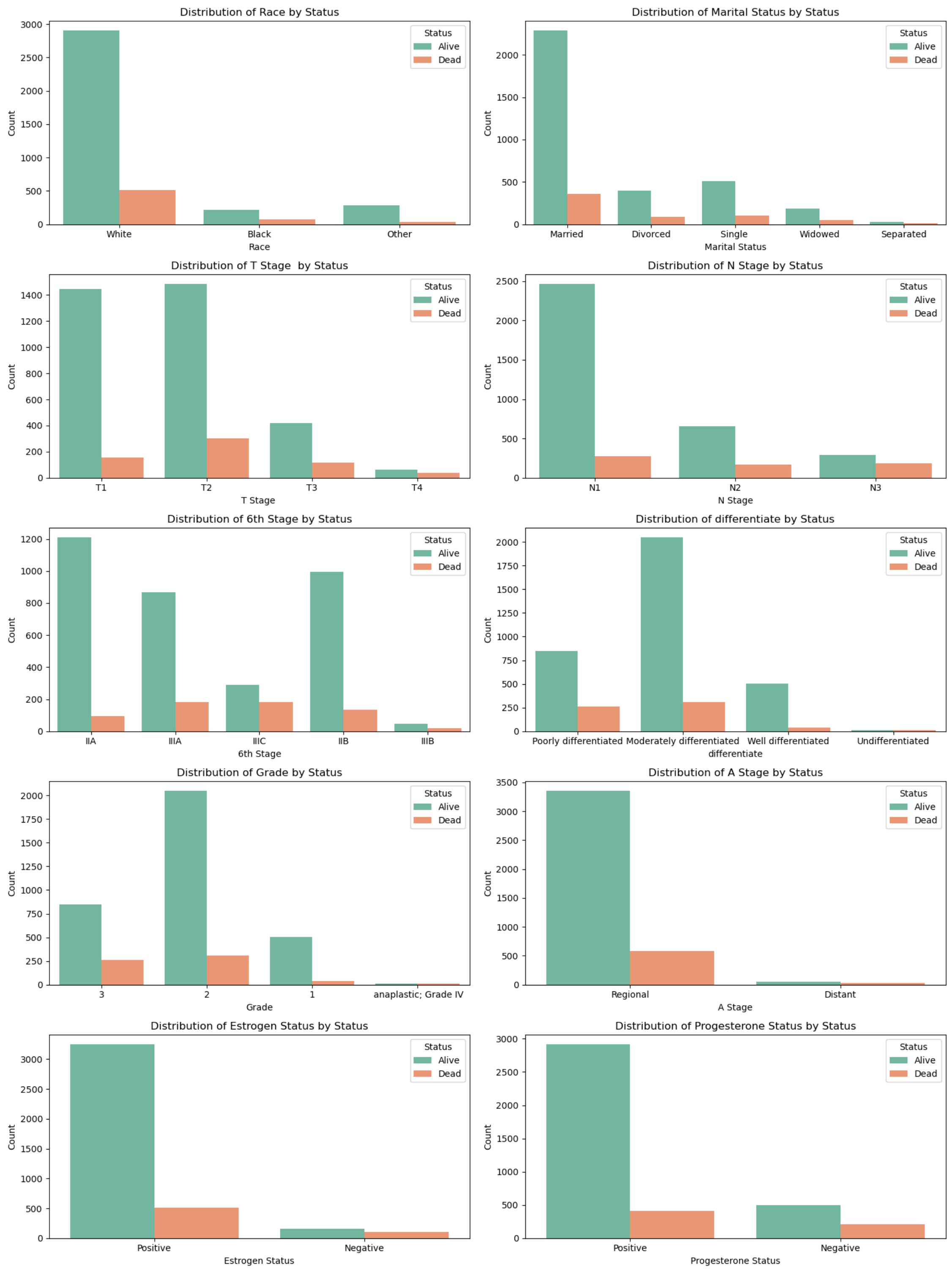
Points Below the Red Line: Data points below the line indicate underestimations by the model (the predicted survival is lower than the actual survival).

Cluster of Blue Dots: Most of the data points are clustered horizontally between 20 and 80 on the x-axis, showing a concentration of actual survival months in that range. However, these points spread vertically, indicating that the model's predictions vary significantly from the actual survival months in this range.

Overall Insight:
The graph shows that while the model predicts survival months, the predictions deviate quite a bit from the actual values, as seen by the spread of points around the red line. The red line serves as a reference for ideal predictions, but most of the points do not lie on this line, suggesting that the model may not be very accurate in predicting the exact survival months.

## 4 CLASSIFICATION MODELS

[14]:
```
#here we create a plot of the distribution of the categorical columns with␣
↪repsect to status
columns_to_plot = data[categorical_cols].drop(columns='Status',axis=1)
plt.figure(figsize=(15, 20))
for i, col in enumerate(columns_to_plot, 1):
    plt.subplot(5, 2, i)  # Adjust based on the number of plots (5 rows, 2␣
↪columns here)
    sns.countplot(x=col, hue='Status', data=data[categorical_cols],␣
↪palette='Set2')
    plt.title(f'Distribution of {col} by Status')
    plt.xlabel(col)
    plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



```
[15]: #Define the data we are using for classification i.e. the categorical_colss
      X = data[categorical_cols]
```

```
#One-Hot encode the variables
X=pd.get_dummies(X, drop_first=True, dtype=int)
X.head(25)
```

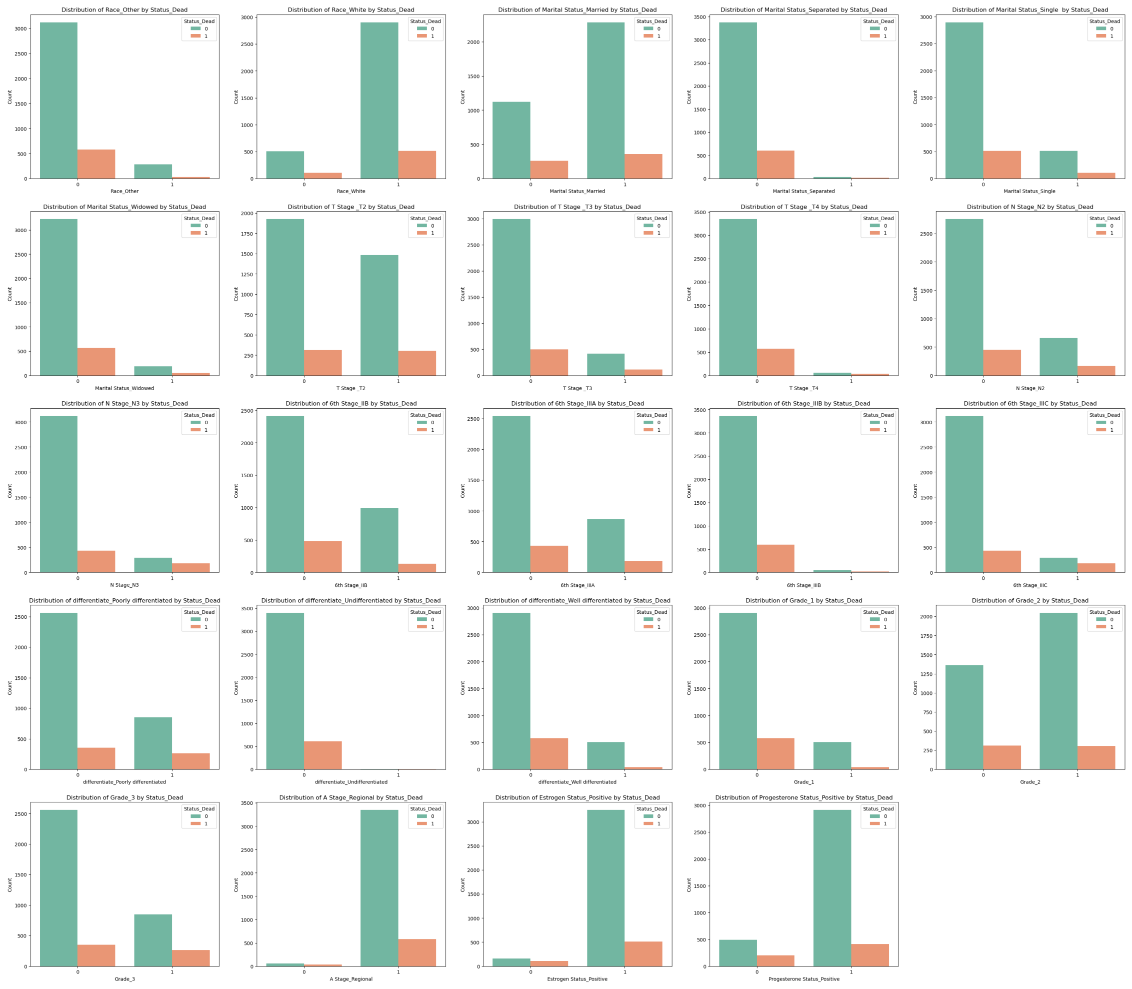
```
[15]:     Race_Other  Race_White  Marital Status_Married  Marital Status_Separated  \
      0            0           1                       1                         0
      1            0           1                       1                         0
      2            0           1                       0                         0
      3            0           1                       1                         0
      4            0           1                       1                         0
      5            0           1                       0                         0
      6            0           1                       1                         0
      7            0           1                       1                         0
      8            0           1                       0                         0
      9            0           1                       1                         0
      10           0           1                       0                         0
      11           0           1                       1                         0
      12           0           1                       1                         0
      13           0           1                       1                         0
      14           0           1                       0                         0
      15           0           1                       1                         0
      16           0           1                       0                         0
      17           0           1                       1                         0
      18           0           0                       0                         0
      19           0           1                       0                         0
      20           1           0                       1                         0
      21           0           1                       1                         0
      22           0           1                       0                         0
      23           0           1                       1                         0
      24           0           1                       1                         0

          Marital Status_Single  Marital Status_Widowed  T Stage _T2  T Stage _T3  \
      0                       0                       0            0            0
      1                       0                       0            1            0
      2                       0                       0            0            1
      3                       0                       0            0            0
      4                       0                       0            1            0
      5                       1                       0            0            0
      6                       0                       0            0            0
      7                       0                       0            1            0
      8                       0                       0            0            0
      9                       0                       0            0            0
      10                      0                       1            0            0
      11                      0                       0            0            1
      12                      0                       0            1            0
      13                      0                       0            0            0
      14                      0                       0            1            0
```

```
15                        0                        0            0            0
16                        1                        0            1            0
17                        0                        0            1            0
18                        0                        0            1            0
19                        0                        0            1            0
20                        0                        0            1            0
21                        0                        0            1            0
22                        1                        0            1            0
23                        0                        0            0            1
24                        0                        0            0            0

    T Stage _T4  N Stage_N2  …  differentiate_Poorly differentiated  \
0             0           0  …                                    1
1             0           1  …                                    0
2             0           0  …                                    0
3             0           0  …                                    1
4             0           0  …                                    1
5             0           0  …                                    0
6             0           0  …                                    0
7             0           0  …                                    0
8             1           0  …                                    1
9             1           0  …                                    0
10            0           0  …                                    0
11            0           0  …                                    1
12            0           0  …                                    1
13            0           1  …                                    1
14            0           0  …                                    0
15            0           0  …                                    0
16            0           0  …                                    0
17            0           0  …                                    0
18            0           0  …                                    0
19            0           0  …                                    0
20            0           0  …                                    0
21            0           1  …                                    0
22            0           0  …                                    1
23            0           0  …                                    1
24            0           0  …                                    1

    differentiate_Undifferentiated  differentiate_Well differentiated  \
0                                0                                  0
1                                0                                  0
2                                0                                  0
3                                0                                  0
4                                0                                  0
5                                0                                  0
6                                0                                  1
7                                0                                  0
```

```
8                                 0                                    0
9                                 0                                    1
10                                0                                    0
11                                0                                    0
12                                0                                    0
13                                0                                    0
14                                0                                    0
15                                0                                    0
16                                0                                    0
17                                0                                    0
18                                0                                    0
19                                0                                    0
20                                0                                    1
21                                0                                    0
22                                0                                    0
23                                0                                    0
24                                0                                    0

    Grade_1  Grade_2  Grade_3  A Stage_Regional  Estrogen Status_Positive  \
0         0        0        1                 1                         1
1         0        1        0                 1                         1
2         0        1        0                 1                         1
3         0        0        1                 1                         1
4         0        0        1                 1                         1
5         0        1        0                 1                         1
6         1        0        0                 1                         1
7         0        1        0                 1                         1
8         0        0        1                 1                         1
9         1        0        0                 0                         1
10        0        1        0                 1                         1
11        0        0        1                 1                         0
12        0        0        1                 1                         1
13        0        0        1                 1                         1
14        0        1        0                 1                         1
15        0        1        0                 1                         1
16        0        1        0                 1                         1
17        0        1        0                 1                         1
18        0        1        0                 1                         1
19        0        1        0                 1                         0
20        1        0        0                 1                         1
21        0        1        0                 1                         1
22        0        0        1                 1                         1
23        0        0        1                 1                         1
24        0        0        1                 1                         1

    Progesterone Status_Positive  Status_Dead
0                              1            0
```

```
1                              1            0
2                              1            0
3                              1            0
4                              1            0
5                              1            0
6                              1            0
7                              1            1
8                              1            0
9                              1            0
10                             1            1
11                             0            0
12                             1            0
13                             1            0
14                             1            0
15                             1            0
16                             1            0
17                             1            0
18                             1            0
19                             0            0
20                             1            0
21                             1            0
22                             1            0
23                             0            0
24                             1            0

[25 rows x 25 columns]
```

[16]:
```
#   Visualize the encoded variables above

plt.figure(figsize=(40, 35))
for i, col in enumerate(X.columns[:-1], 1):  # Exclude 'Status_Dead'
    plt.subplot(5, 5, i)  # Adjust based on the number of plots (5 rows, 5␣
↪columns here)
    sns.countplot(x=col, hue='Status_Dead', data=X, palette='Set2')
    plt.title(f'Distribution of {col} by Status_Dead')
    plt.xlabel(col)
    plt.ylabel('Count')

plt.show()
```



[17]:
```
#select the features(X_cat) and the target(y_cat)
y_cat = X['Status_Dead']
X_cat = X.drop(columns='Status_Dead',axis = 1)
```

[18]:
```
#create the test and train sets
X_train, X_test, y_train, y_test = train_test_split(X_cat, y_cat, test_size=0.
 ↪3, random_state=1)
```

## 4.1 The Logistic Regression Model

[19]:
```
#create the logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

[19]:
```
LogisticRegression()
```

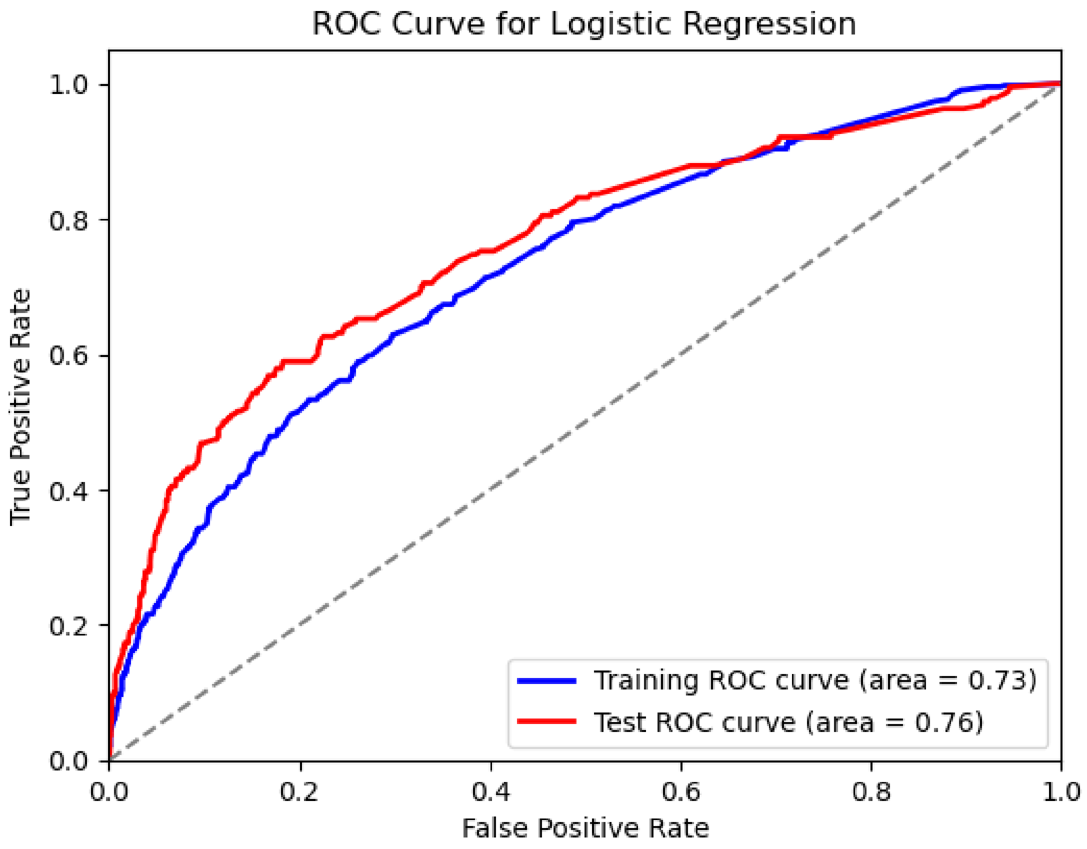
[20]:
```
#make predictions for the model
y_pred_train = logreg.predict(X_train)
y_pred_test = logreg.predict(X_test)
```

[21]:
```
#Model Evaluation
#Creating a classification report
train_class_report = classification_report(y_train, y_pred_train)
test_class_report = classification_report(y_test, y_pred_test)
#Display the results of the classification report
print('The outcome of the training classification report is:')
print(train_class_report)
print('\n\n')
print('The outcome of the test classification report is:')
print(test_class_report)
print('\n\n')

#Let's create a confusion matrix that can helpprovide a summary of prediction␣
 ↪results
train_conf_mat = confusion_matrix(y_train, y_pred_train)
test_conf_mat = confusion_matrix(y_test, y_pred_test)
#Display the confusion matrix results for the train and test sets
print('The confusion matric for the train set is:')
print(train_conf_mat)
print('\n\n')
print('The confusion matric for the test set is:')
print(test_conf_mat)
print('\n\n')

#Now let's look at the accuracy score for both sets
train_acc = accuracy_score(y_train, y_pred_train)
test_acc = accuracy_score(y_test, y_pred_test)
#Display the results of the accuracy score
print('The accuracy score for the train set is:', train_acc)
print('The accuracy score for the test set is:', test_acc)

#Fianlly we should consider the ROC curve and the AUC
#start by getting probability estimates of the positive class
y_train_prob = logreg.predict_proba(X_train)[:, 1]
y_test_prob = logreg.predict_proba(X_test)[:, 1]
#ROC curve for the training data
fpr_train, tpr_train, threshold_train = roc_curve(y_train, y_train_prob)
roc_auc_train = auc(fpr_train, tpr_train)
#ROC curve for the test set
fpr_test, tpr_test, threshold_test = roc_curve(y_test, y_test_prob)
roc_auc_test = auc(fpr_test, tpr_test)
#plot the ROC curve
plt.figure()
plt.plot(fpr_train, tpr_train, color='blue', lw=2, label=f'Training ROC curve␣
↪(area = {roc_auc_train:.2f})')
plt.plot(fpr_test, tpr_test, color='red', lw=2, label=f'Test ROC curve (area =␣
↪{roc_auc_test:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```

```
The outcome of the training classification report is:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      2390
           1       0.59      0.10      0.17       426

    accuracy                           0.85      2816
   macro avg       0.73      0.54      0.54      2816
weighted avg       0.82      0.85      0.81      2816




The outcome of the test classification report is:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1017
           1       0.70      0.14      0.23       190

    accuracy                           0.86      1207
   macro avg       0.78      0.56      0.57      1207
weighted avg       0.84      0.86      0.81      1207




The confusion matric for the train set is:
[[2362   28]
 [ 385   41]]



The confusion matric for the test set is:
[[1006   11]
```

```
 [ 164   26]]
```

```
The accuracy score for the train set is: 0.8533380681818182
The accuracy score for the test set is: 0.8550124275062138
```



For the training set:

The model has high precision (0.86) for class 0, meaning it correctly predicted most instances labeled as 0. The recall for class 1 is low (0.10), indicating that the model missed many actual positive instances. The overall accuracy is 0.85, which is reasonably good. For the testing set:

The precision and recall for class 0 have improved slightly. The precision for class 1 has also improved, but the recall remains low. The overall accuracy has increased slightly to 0.86. Confusion Matrices:

A confusion matrix shows the number of instances predicted for each class compared to their actual classes. For the training set:

Many instances of class 0 were correctly predicted (2362), but there were 28 false positives. Many instances of class 1 were incorrectly predicted as class 0 (385), and only 41 were correctly predicted. For the testing set:

Similar patterns are observed, with more correct predictions for class 0 and fewer correct predictions for class 1. Accuracy Scores:

The accuracy score measures the overall proportion of correct predictions. Overall:

The model is performing reasonably well on both training and testing sets, with higher accuracy for the testing set. However, the model struggles to correctly predict instances of class 1, especially in the training set. Further analysis and potential improvements might be necessary to address the class imbalance and improve the model's performance on class 1.

Graph Interpretation:

Shape of the curve: A ROC curve ideally should be as close to the top-left corner as possible. This indicates that the model has high TPR and low FPR, which is desirable.

Comparison of curves: In this graph, the test ROC curve is slightly above the training ROC curve, suggesting that the model might be slightly underfitting the training data. However, both curves are significantly above the random classifier line, indicating that the model is performing better than random guessing.

Area Under the Curve (AUC): The AUC is a metric that quantifies the overall performance of a model. A higher AUC indicates better performance. In this case, the AUC for both training and testing curves is above 0.5, indicating that the model is performing better than random guessing. The test AUC is slightly higher than the training AUC, which is a good sign.

Overall, the ROC curve suggests that the logistic regression model is performing reasonably well on both training and testing data, and is significantly better than a random classifier. The slight underfitting indicated by the difference between the training and testing curves might be addressed by adjusting the model's hyperparameters or collecting more data.

## 4.2 Decision tree model

### 4.2.1 Vanilla Model of the Decision Tree

[22]:
```
#Initiate the classifier
clf = DecisionTreeClassifier(random_state=42)

#fit it on the training data
clf.fit(X_train, y_train)
```

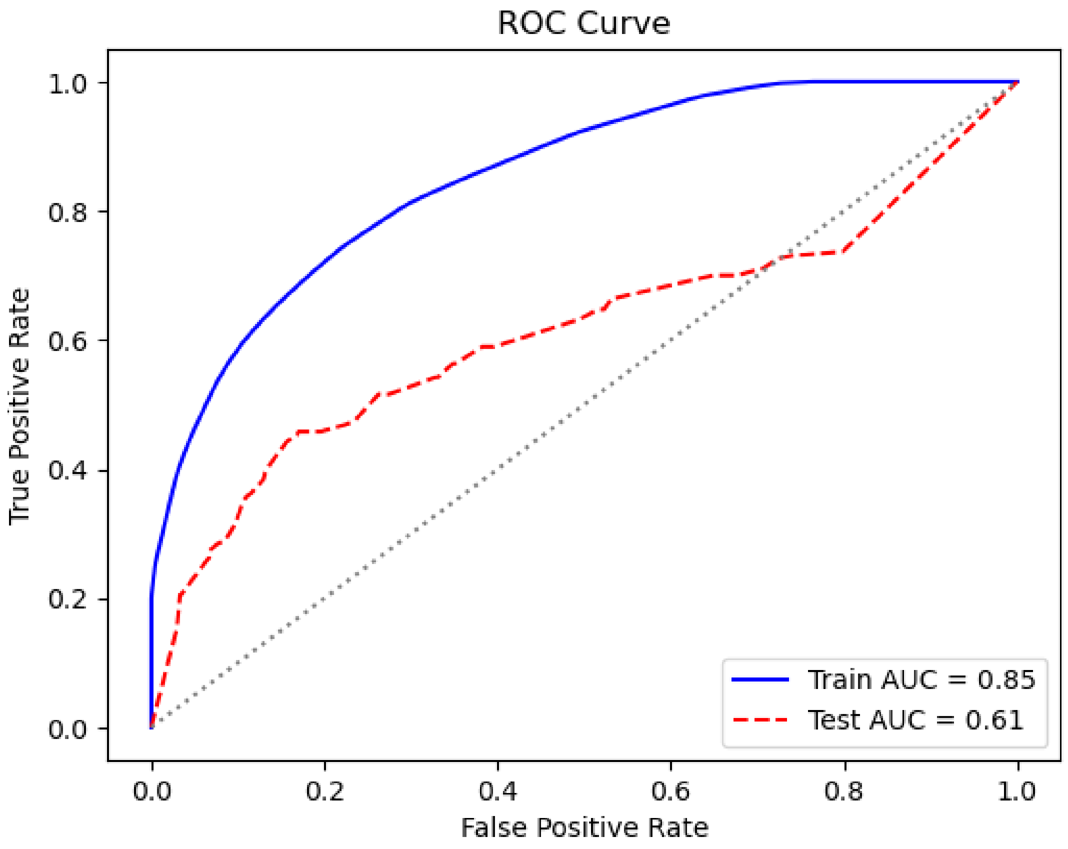
[22]:
```
DecisionTreeClassifier(random_state=42)
```

[23]:
```
# Make predictions
y_tr_pred = clf.predict(X_train)
y_te_pred = clf.predict(X_test)
```

[24]:
```
# Evaluate the model
#Look at the classification report
print('The classification report for the training set is:')
print(classification_report(y_train, y_tr_pred))
print('\n')
print('The classification report for the test set is:')
```

```python
print(classification_report(y_test, y_te_pred))
print('\n\\n')
#Show accuracy for both sets
print('the accuracy for the training set is:',accuracy_score(y_train,␣
↪y_tr_pred))
print('th eaccuracy for the test set is:', accuracy_score(y_test, y_te_pred))
print('\n')
#Create the ROC curve and AUC for both sets
#start by getting the probability for each od the sets
clf_train_prob = clf.predict_proba(X_train)[:, 1]
clf_test_prob = clf.predict_proba(X_test)[:, 1]
#calculate the ROC and auc
train_fpr, train_tpr, train_threshold = roc_curve(y_train, clf_train_prob)
train_auc = auc(train_fpr, train_tpr)
test_fpr, test_tpr, test_threshold = roc_curve(y_test, clf_test_prob)
test_auc = auc(test_fpr, test_tpr)
#plot the ROC roc_curve
plt.figure()
plt.plot(train_fpr, train_tpr, color='blue', label=f'Train AUC = {train_auc:.
↪2f}')
plt.plot(test_fpr, test_tpr, color='red', linestyle='--', label=f'Test AUC =␣
↪{test_auc:.2f}')
plt.plot([0, 1], [0, 1], color='grey', linestyle=':')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
The classification report for the training set is:
              precision    recall  f1-score   support

           0       0.88      0.99      0.94      2390
           1       0.89      0.26      0.40       426

    accuracy                           0.88      2816
   macro avg       0.89      0.63      0.67      2816
weighted avg       0.88      0.88      0.85      2816




The classification report for the test set is:
              precision    recall  f1-score   support

           0       0.87      0.97      0.91      1017
           1       0.53      0.21      0.30       190
    accuracy                           0.85      1207
   macro avg       0.70      0.59      0.60      1207
weighted avg       0.81      0.85      0.82      1207


\n
the accuracy for the training set is: 0.8831676136363636
th eaccuracy for the test set is: 0.8458989229494615
```



### Insights from the results 1. Precision, Recall and F1_score class 0 (Majority Class):

```
Training Set: Precision = 0.88, Recall = 0.99, F1-Score = 0.94
Test Set: Precision = 0.87, Recall = 0.97, F1-Score = 0.91
The model performs very well for class 0, with high precision, recall, and F1-score in both the

Class 1 (Minority Class):

Training Set: Precision = 0.89, Recall = 0.26, F1-Score = 0.40
Test Set: Precision = 0.53, Recall = 0.21, F1-Score = 0.30
```

```
The performance for class 1 is notably weaker, particularly in terms of recall and F1-score. Th
```

2. Imbalance in Performance

   The disparity between the performance on class 0 and class 1 suggests that the model is biased towards predicting the majority class. This is common in imbalanced datasets where one class significantly outnumbers the other. The model's high accuracy is driven by its performance on class 0, but its ability to detect class 1 is poor.

3. Conclusion

   While the model performs well overall, its ability to detect the minority class (class 1) is lacking.There is aneed to address the class imbalance in the data.

### 4.2.2 Addressing class imbalance

There are several ways in whiv=ch the class imbalance can be addressed and the one of the methods we are going to consider first is the use of SMOTE. This is a resampling technique where we use oversampling for the minority class.

[25]:
```
#Apply SMOTE to the training dta
smote = SMOTE(random_state=42)
smote_X_train, smote_y_train = smote.fit_resample(X_train,y_train)
```

[26]:
```
#Train a model
clf.fit(smote_X_train, smote_y_train)
```

[26]:
```
DecisionTreeClassifier(random_state=42)
```

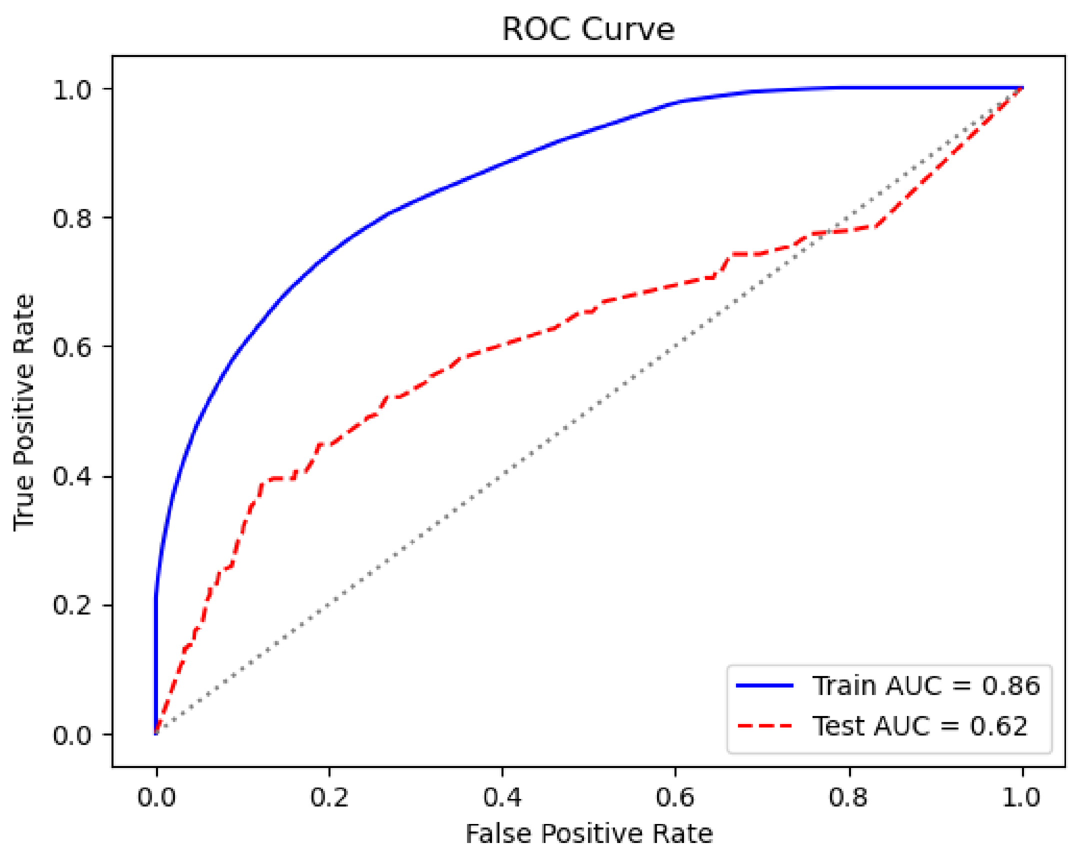
[27]:
```
#create predctions for train and test sets
smote_train_pred = clf.predict(smote_X_train)
smote_test_pred = clf.predict(X_test)
```

[28]:
```
print("Classification Report for Training Set:")
print(classification_report(smote_y_train, smote_train_pred))
print("Classification Report for Test Set:")
print(classification_report(y_test, smote_test_pred))
print('\n\n')
#Show accuracy for both sets
print('the accuracy for the training set is:',accuracy_score(smote_y_train,␣
 ↪smote_train_pred))
print('th eaccuracy for the test set is:', accuracy_score(y_test,␣
 ↪smote_test_pred))
print('\n')
#Create the ROC curve and AUC for both sets
#start by getting the probability for each od the sets
smote_train_prob = clf.predict_proba(smote_X_train)[:, 1]
smote_test_prob = clf.predict_proba(X_test)[:, 1]
#calculate the ROC and auc
```

```
smote_train_fpr, smote_train_tpr, smote_train_threshold =␣
 ↪roc_curve(smote_y_train, smote_train_prob)
train_auc = auc(smote_train_fpr, smote_train_tpr)
smote_test_fpr, smote_test_tpr, smote_test_threshold = roc_curve(y_test,␣
 ↪smote_test_prob)
test_auc = auc(smote_test_fpr, smote_test_tpr)
#plot the ROC roc_curve
plt.figure()
plt.plot(smote_train_fpr, smote_train_tpr, color='blue', label=f'Train AUC =␣
 ↪{train_auc:.2f}')
plt.plot(smote_test_fpr, smote_test_tpr, color='red', linestyle='--',␣
 ↪label=f'Test AUC = {test_auc:.2f}')
plt.plot([0, 1], [0, 1], color='grey', linestyle=':')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Classification Report for Training Set:
              precision    recall  f1-score   support

           0       0.76      0.79      0.78      2390
           1       0.78      0.75      0.77      2390

    accuracy                           0.77      4780
   macro avg       0.77      0.77      0.77      4780
weighted avg       0.77      0.77      0.77      4780

Classification Report for Test Set:
              precision    recall  f1-score   support

           0       0.89      0.74      0.81      1017
           1       0.27      0.49      0.35       190

    accuracy                           0.71      1207
   macro avg       0.58      0.62      0.58      1207
weighted avg       0.79      0.71      0.74      1207
```

```
the accuracy for the training set is: 0.7713389121338912
th eaccuracy for the test set is: 0.7050538525269263
```



Now while the use of SMOTE has addressed the issue of class imbalance , the model may still need further refinement to generalize better on unseen data. Here below are some key highlights supporting this:

1. ROC Curve Analysis:

   Train AUC: 0.86 Test AUC: 0.62 The train AUC has increased slightly from 0.83 to 0.86, indicating that the model is better at distinguishing between classes on the training data after SMOTE was applied. The test AUC remains the same at 0.62, suggesting that the model's ability to generalize to unseen data hasn't improved significantly.

2. Classification Report Summary:

   Training Set (After SMOTE): Accuracy: 0.77 Precision for class 1: 0.78 Recall for class 1: 0.75 F1-score for class 1: 0.77

   Test Set (After SMOTE): Accuracy: 0.71 Precision for class 1: 0.27 Recall for class 1: 0.49 F1-score for class 1: 0.35 Interpretation:

3. Impact of SMOTE:

   SMOTE has balanced the classes in the training data, leading to a more balanced precision and recall for both classes in the training set. For the test set, the recall for class 1 (the minority class) improved from 0.21 to 0.49, indicating the model's improved ability to identify positive cases. However, the precision for class 1 decreased from 0.53 to 0.27, showing that the model is now making more false positive errors.

4. Accuracy Decrease:

   The overall accuracy on both training and test sets has decreased compared to before applying SMOTE (from 0.88 to 0.77 on the training set and from 0.85 to 0.71 on the test set). This is typical when using SMOTE as the model is forced to learn more about the minority class, which can reduce accuracy but improve the balance of errors between classes. Model Generalization:

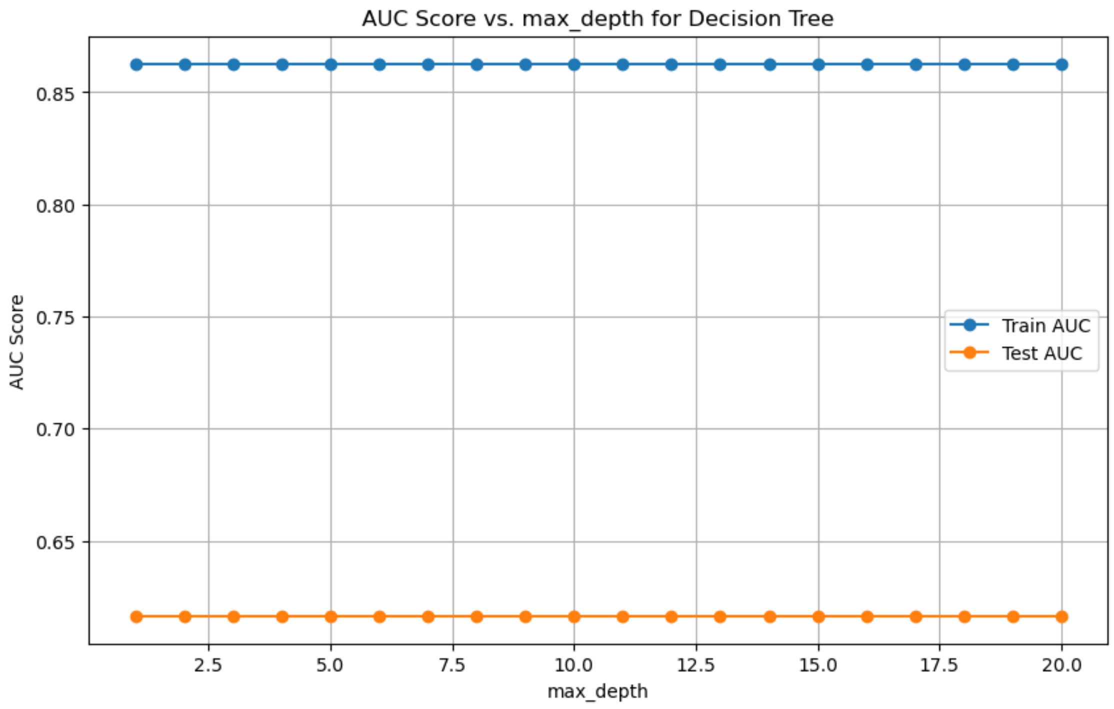
The test AUC remains unchanged at 0.62, which suggests that while the model's ability to handle class imbalance has improved, its overall discriminatory power on unseen data hasn't.

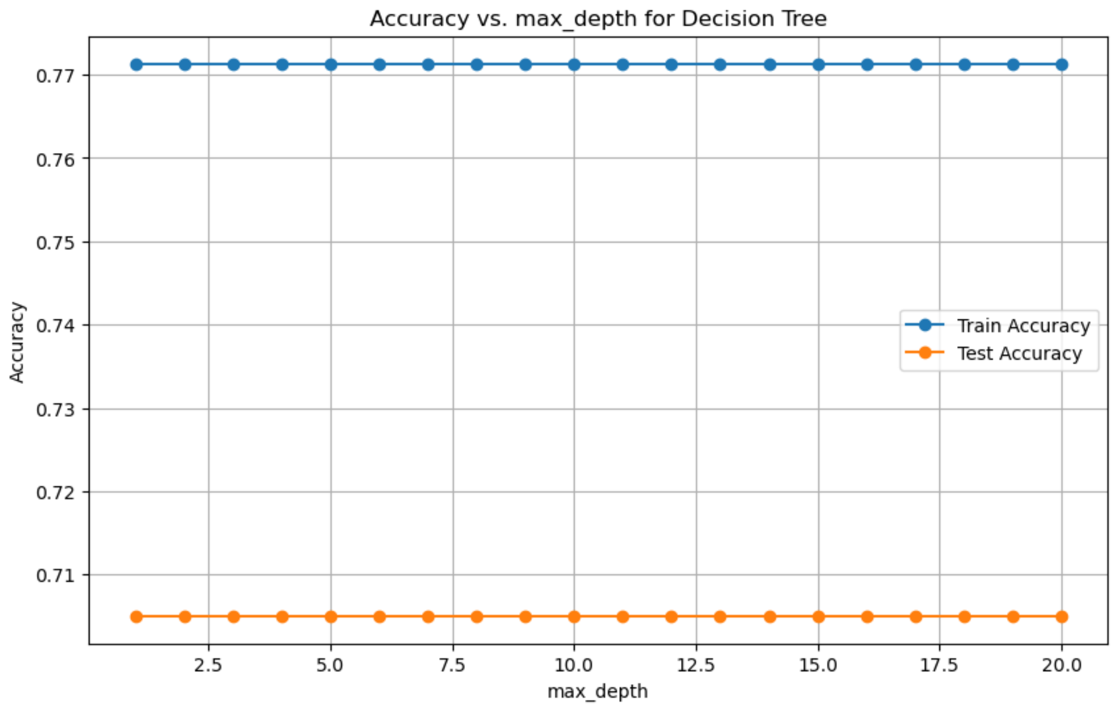
Now we want to try and tune this model by trying to fine tune the hyperparameters and then have a final model.

### 4.2.3 Hyperparaameter tuning I

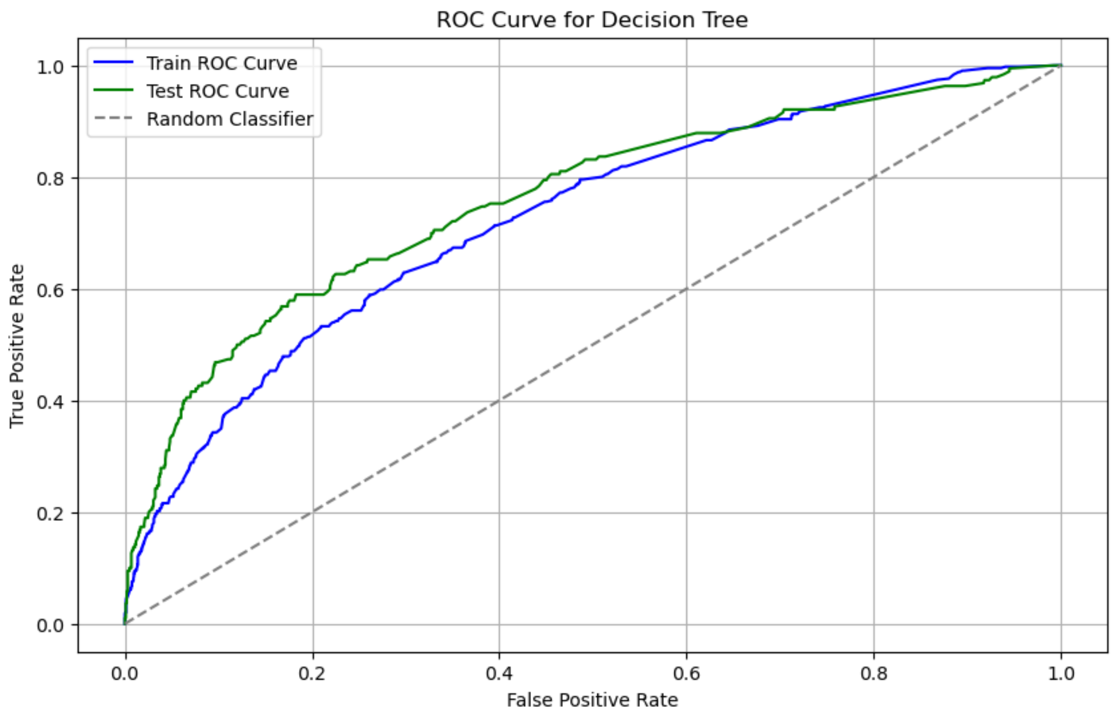
We want to start by looking for the maximum depth of the model

[29]:
```python
# Initialize lists to store results for different depths
depths = range(1, 21)
train_aucs = []
test_aucs = []
train_accuracies = []
test_accuracies = []

# Loop over different values of max_depth
for max_depth in depths:
    # Calculate ROC AUC scores for both sets
    train_auc = roc_auc_score(smote_y_train, smote_train_prob)
    test_auc = roc_auc_score(y_test, smote_test_prob)

    # Calculate accuracy for both sets
    train_accuracy = clf.score(smote_X_train, smote_y_train)
    test_accuracy = clf.score(X_test, y_test)

    # Store the results
    train_aucs.append(train_auc)
    test_aucs.append(test_auc)
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

# Step 5: Plot AUC scores for different max_depth values
plt.figure(figsize=(10, 6))
plt.plot(depths, train_aucs, marker='o', label='Train AUC')
plt.plot(depths, test_aucs, marker='o', label='Test AUC')
plt.xlabel('max_depth')
plt.ylabel('AUC Score')
plt.title('AUC Score vs. max_depth for Decision Tree')
plt.legend()
plt.grid(True)
plt.show()

# Step 6: Plot Accuracy for different max_depth values
plt.figure(figsize=(10, 6))
plt.plot(depths, train_accuracies, marker='o', label='Train Accuracy')
plt.plot(depths, test_accuracies, marker='o', label='Test Accuracy')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. max_depth for Decision Tree')
plt.legend()
plt.grid(True)
plt.show()

# Step 7: Output classification report for the best max_depth
best_depth = depths[test_aucs.index(max(test_aucs))]
print(f"Best max_depth based on Test AUC: {best_depth}")

best_clf = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
best_clf.fit(smote_X_train, smote_y_train)

smote_train_pred = best_clf.predict(smote_X_train)
smote_test_pred = best_clf.predict(X_test)

print("Classification Report for Training Set:")
print(classification_report(smote_y_train, smote_train_pred))

print("Classification Report for Test Set:")
print(classification_report(y_test, smote_test_pred))

train_auc = roc_auc_score(smote_y_train, best_clf.predict_proba(smote_X_train)[:
 ↪, 1])
test_auc = roc_auc_score(y_test, best_clf.predict_proba(X_test)[:, 1])

print(f"Train AUC: {train_auc:.2f}")
print(f"Test AUC: {test_auc:.2f}")

# Plot ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr_train, tpr_train, color='blue', label='Train ROC Curve')
plt.plot(fpr_test, tpr_test, color='green', label='Test ROC Curve')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random␣
 ↪Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree')
plt.legend()
plt.grid(True)
plt.show()
```





```
Best max_depth based on Test AUC: 1
Classification Report for Training Set:
              precision    recall  f1-score   support

           0       0.56      0.91      0.69      2390
           1       0.75      0.27      0.40      2390

    accuracy                           0.59      4780
   macro avg       0.66      0.59      0.54      4780
weighted avg       0.66      0.59      0.54      4780

Classification Report for Test Set:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      1017
           1       0.43      0.33      0.37       190

    accuracy                           0.83      1207
   macro avg       0.66      0.62      0.64      1207
weighted avg       0.81      0.83      0.82      1207

Train AUC: 0.59
Test AUC: 0.62
```



Graph Interpretation:

Shape of the curve: A ROC curve ideally should be as close to the top-left corner as possible. This indicates that the model has high TPR and low FPR, which is desirable.

Comparison of curves: In this graph, the train ROC curve is slightly above the test ROC curve, suggesting that the model might be slightly overfitting the training data. However, both curves are significantly above the random classifier line, indicating that the model is performing better than random guessing.

Area Under the Curve (AUC): The AUC is a metric that quantifies the overall performance of a model. A higher AUC indicates better performance. In this case, the AUC for both train and test curves would likely be higher than 0.5, indicating that the model is performing better than random guessing.

Overall, the ROC curve suggests that the decision tree model is performing reasonably well on both training and testing data, and is significantly better than a random classifier. However, there might be some overfitting, as indicated by the slight difference between the train and test curves. To improve the model's performance, techniques like regularization or hyperparameter tuning could be considered.

Here are some key take aways from the results above: 1. The model's simplicity: The ideal max_depth will be 1suggesting the model is very simple, reflected in the low performance of themetrics especially for class 1 2. The low recall and F1-scores for class 1 in both the training and test sets suggest that the model struggles to detect the minority class, which is a common issue in imbalanced datasets. 3. Although there isn't a large gap between Train and Test AUC, the overall performance is poor, suggesting that a deeper or more complex model may be necessary to capture the patterns in the data, though care must be taken to avoid overfitting

From the data above it is clear that when we use that data that has been oversampled that all the test performance does not increase accross all samples hence there is no positive effect on the test performance.

It is safe to make the assumption that whether we include a different approach of tuning the decion tree the results will be fairly the same in that there willl be no change in the test performance of the model.

## 4.3 LOGISTIC REGRESSION vs DECISION TREE

### 4.3.1 Analysis of the two models

Choosing the better model between the logistic regression and decision tree (with SMOTE) depends on your specific goals and the trade-offs you're willing to make. Here's a comparison based on the provided results:

### 4.3.2 Logistic Regression

- **Training Accuracy**: 85.33%
- **Test Accuracy**: 85.50%
- **Precision and Recall**:

  - **Class 0**: High precision (0.86), recall (0.99), and f1-score (0.92) on both training and test sets.
  - **Class 1**: Lower precision (0.59 train, 0.70 test) and very low recall (0.10 train, 0.14 test).

#### 4.3.3 Decision Tree tuned (max_depth) after SMOTE

- **Training Accuracy**: 59% (This seems lower compared to the logistic regression, possibly due to overfitting or the model complexity.)
- **Test Accuracy**: 83%
- **Precision and Recall**:
  - **Class 0**: Precision (0.88 test), recall (0.92 test), and f1-score (0.90 test) are high.
  - **Class 1**: Lower precision (0.43 test), recall (0.33 test), and f1-score (0.37 test).

#### 4.3.4 Comparative Analysis

1. **Overall Accuracy**: Logistic regression has higher accuracy on both the training and test sets compared to the tuned decision tree.
2. **Class 0 Performance**: Both models perform well on Class 0, but logistic regression maintains consistency across both sets.
3. **Class 1 Performance**:
   - The tuned decision tree shows some improvement in recall and precision for Class 1 compared to the original logistic regression, but the metrics are still relatively low.
   - Logistic regression's performance on Class 1 is quite poor, but the decision tree performs better in recall, suggesting it may better identify some of the minority class samples.

#### 4.3.5 Suitability

- **Logistic Regression**:
  - **Pros**: Consistently high performance on Class 0, high overall accuracy.
  - **Cons**: Struggles with Class 1, leading to a potential under representation of the minority class.
- **Decision Tree with SMOTE**:
  - **Pros**: Improved handling of the minority class (Class 1) due to SMOTE.
  - **Cons**: Lower overall accuracy and training accuracy, potentially overfitting or too simplistic due to max_depth of 1.

#### 4.3.6 Recommendation

If the primary concern is overall accuracy and robust performance on the majority class (Class 0), **logistic regression** is better. However, if improving performance on the minority class (Class 1) is critical and there is a posibility of experimenting with more complex models, the **decision tree with SMOTE** might offer better potential with further tuning.

Trying more advanced models like Random Forests, Gradient Boosting, or Support Vector Machines should be considered, which can potentially offer a better balance between precision and recall for both classes.

To explore further adjustments or other models, the choice depends on the specific application and which metric (precision, recall, F1-score) is more important for the hospital.