breast-cancer-4

October 9, 2024

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
import missingno as msno
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import OrdinalEncoder
from scipy.stats import shapiro
enc = OrdinalEncoder(handle_unknown= 'use_encoded_value', unknown_value= -1)

from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split as tts
from sklearn.feature_selection import mutual_info_classif
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import svm
from sklearn.model_selection import cross_val_score
```

[2]:
```
df = pd.read_csv("C:\\Users\\prane\\Downloads\\Breast_Cancer (1).csv")
print(df.shape)
df.head
```

```
(4024, 16)
```

[2]: <bound method NDFrame.head of   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]:
```
# checking for null values
df.isnull().sum()

# we don't have any null values . Therefore Data Cleaning is not necessary
```

[3]:
```
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
```

[4]:
```
df.head()
```

[4]:
```
   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

               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
```

```
  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

   Survival Months Status
0                60  Alive
1                62  Alive
2                75  Alive
3                84  Alive
4                50  Alive
```

[5]:
```
df.tail()
```

[5]:
```
       Age   Race Marital Status T Stage  N Stage 6th Stage  \
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  \
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  \
4019            Positive                       1                      1
4020            Positive                      14                      8
4021            Negative                      11                      3
4022            Positive                      11                      1
4023            Positive                       7                      2

      Survival Months Status
4019                49  Alive
4020                69  Alive
4021                69  Alive
4022                72  Alive
4023               100  Alive
```

[6]:
```
# Number of unique columns
uni_col = df.columns.nunique()
```

```
# Number of unique rows
uni_row = df.nunique(axis=0)

print(f"Number of unique columns: {uni_col}")
print(f"Number of unique rows: {uni_row}")
```

```
Number of unique columns: 16
Number of unique rows: Age                        40
Race                         3
Marital Status               5
T Stage                      4
N Stage                      3
6th Stage                    5
differentiate                4
Grade                        4
A Stage                      2
Tumor Size                 110
Estrogen Status              2
Progesterone Status          2
Regional Node Examined      54
Reginol Node Positive       38
Survival Months            107
Status                       2
dtype: int64
```

[7]:
```
df.dtypes
```

[7]:
```
Age                         int64
Race                       object
Marital Status             object
T Stage                    object
N Stage                    object
6th Stage                  object
differentiate              object
Grade                      object
A Stage                    object
Tumor Size                  int64
Estrogen Status            object
Progesterone Status        object
Regional Node Examined      int64
Reginol Node Positive       int64
Survival Months             int64
Status                     object
dtype: object
```

[8]:
```
import pandas as pd
```

```
# Load your DataFrame
df = pd.read_csv('C:\\Users\\prane\\Downloads\\Breast_Cancer (1).csv')  #␣
 ↪Replace with your file path

# Calculate IQR for 'Tumor Size'
Q1 = df['Tumor Size'].quantile(0.25)
Q3 = df['Tumor Size'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace outliers with the median value
median_value = df['Tumor Size'].median()
#df.loc[df['Tumor Size'] < lower_bound, 'Tumor Size'] = median_value
df.loc[df['Tumor Size'] > upper_bound, 'Tumor Size'] = median_value

# Save the cleaned DataFrame, if needed
df.to_csv('C:\\Users\\prane\\Downloads\\Breast_Cancer_1.csv', index=False)
```

[9]:
```
### ------------  ENCODING   ----------------


import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming df is your DataFrame

# List of columns you want to encode
columns_to_encode = ['T Stage ', '6th Stage', 'N Stage', 'Race',␣
 ↪'differentiate', 'Marital Status', 'Grade', 'A Stage','Estrogen␣
 ↪Status','Progesterone Status','Status']

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder to each column
for col in columns_to_encode:
    df[col] = label_encoder.fit_transform(df[col])

# Display the first few rows of the updated DataFrame
print(df.head())
```

```
   Age  Race  Marital Status  T Stage   N Stage  6th Stage  differentiate  \
0   68     2               1         0        0          0              1
1   50     2               1         1        1          2              0
```

```
2   58     2                0         2         2          4              0
3   58     2                1         0         0          0              1
4   47     2                1         1         0          1              1

   Grade  A Stage  Tumor Size  Estrogen Status  Progesterone Status  \
0      3        1           4                1                    1
1      2        1          35                1                    1
2      2        1          63                1                    1
3      3        1          18                1                    1
4      3        1          41                1                    1

   Regional Node Examined  Reginol Node Positive  Survival Months  Status
0                      24                      1               60       0
1                      14                      5               62       0
2                      14                      7               75       0
3                       2                      1               84       0
4                       3                      1               50       0
```

[10]:
```
df.head()
```

[10]:
```
    Age  Race  Marital Status  T Stage   N Stage  6th Stage  differentiate  \
0   68     2                1         0        0          0              1
1   50     2                1         1        1          2              0
2   58     2                0         2        2          4              0
3   58     2                1         0        0          0              1
4   47     2                1         1        0          1              1

   Grade  A Stage  Tumor Size  Estrogen Status  Progesterone Status  \
0      3        1           4                1                    1
1      2        1          35                1                    1
2      2        1          63                1                    1
3      3        1          18                1                    1
4      3        1          41                1                    1

   Regional Node Examined  Reginol Node Positive  Survival Months  Status
0                      24                      1               60       0
1                      14                      5               62       0
2                      14                      7               75       0
3                       2                      1               84       0
4                       3                      1               50       0
```

[11]:
```
from scipy.stats import shapiro
data = df
for column in data.columns:
    if pd.api.types.is_numeric_dtype(data[column]):
        stat, p = shapiro(data[column].dropna())
        print('Statistics=%.3f, p=%.3f' % (stat, p))
```

```
        if p > 0.05:
            print('Sample looks Gaussian (normally distributed)')
        else:
            print('Sample does not look Gaussian (not normally distributed)')
```

```
Statistics=0.976, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.436, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.724, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.804, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.641, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.835, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.666, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.788, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.132, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.916, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.269, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.458, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.960, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.653, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.962, p=0.000
Sample does not look Gaussian (not normally distributed)
Statistics=0.431, p=0.000
Sample does not look Gaussian (not normally distributed)
```

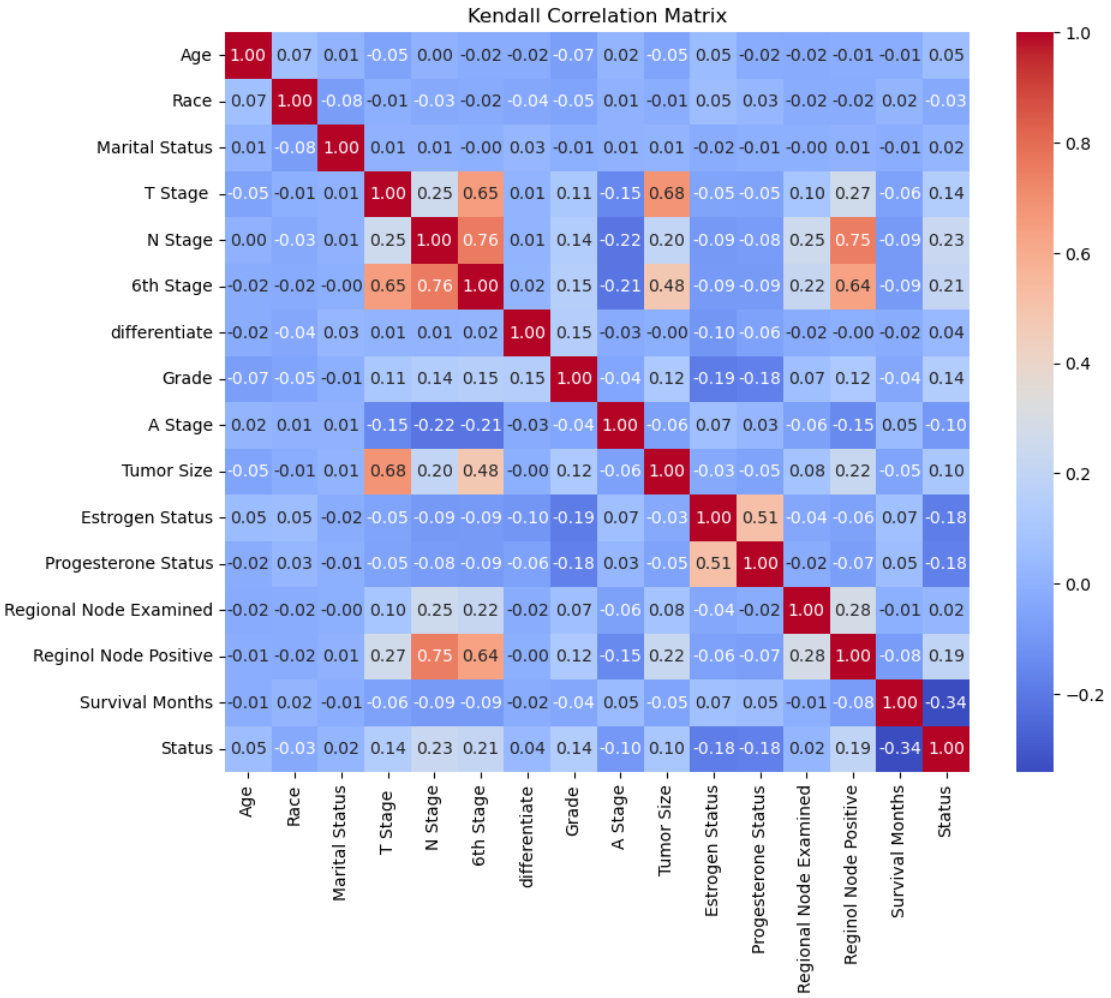
[12]:
```
# perform chi- square test for feature importance on Status

import pandas as pd
from scipy.stats import chi2_contingency


# Assuming df is your DataFrame and 'Status' is your target variable

# List of categorical columns (excluding 'Status')
```

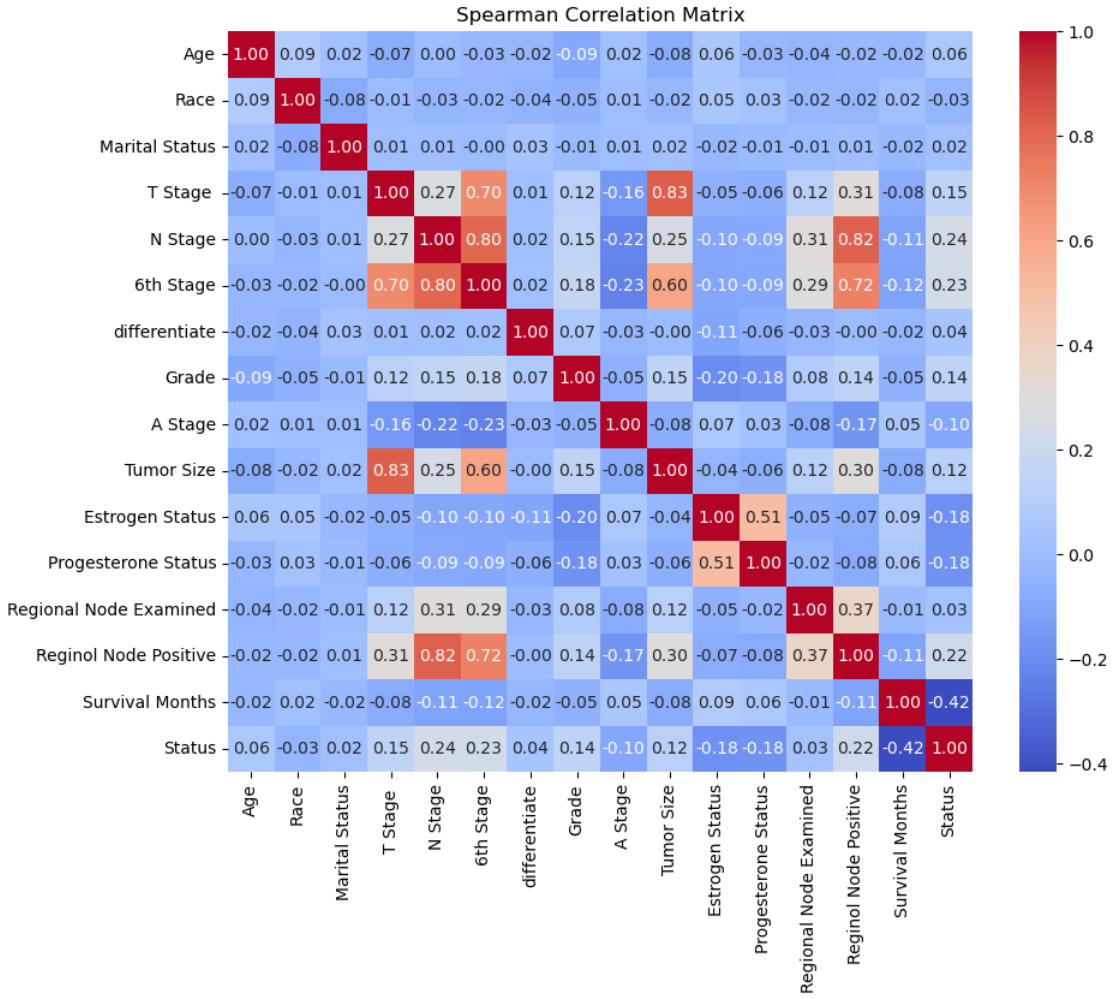
```
categorical_columns = ['Age', 'Race', 'Marital Status', 'T Stage ', 'N Stage',␣
 ↪'6th Stage',
       'differentiate', 'Grade', 'A Stage', 'Tumor Size', 'Estrogen Status',
       'Progesterone Status', 'Regional Node Examined',
       'Reginol Node Positive', 'Survival Months']

# Dictionary to hold p-values
p_values = {}

for col in categorical_columns:
    # Creating a contingency table
    contingency_table = pd.crosstab(df[col], df['Status'])

    # Performing the Chi-square test
    chi2, p, dof, ex = chi2_contingency(contingency_table)

    # Storing the p-value for the feature
    p_values[col] = p

# Sorting features by their p-value
sorted_p_values = {k: v for k, v in sorted(p_values.items(), key=lambda item:␣
 ↪item[1])}

# Display sorted features by importance
sorted_p_values
```

[12]:
```
{'Survival Months': 2.918381381477759e-256,
 '6th Stage': 9.830332296203994e-60,
 'N Stage': 2.430140625217663e-59,
 'Reginol Node Positive': 2.2926910134541016e-51,
 'Estrogen Status': 3.0526081181489177e-31,
 'Progesterone Status': 5.392079685518964e-29,
 'differentiate': 3.0913516733336542e-24,
 'Grade': 3.0913516733336542e-24,
 'T Stage ': 2.7790953099786567e-22,
 'A Stage': 2.2264262284984456e-09,
 'Race': 8.440928800112451e-07,
 'Marital Status': 1.1027694804532703e-05,
 'Age': 1.880215544112303e-05,
 'Tumor Size': 3.696964133377591e-05,
 'Regional Node Examined': 0.045010588788791626}
```

[13]:
```
kendall_corr = df.corr(method='kendall')
spearman_corr = df.corr(method='spearman')

plt.figure(figsize=(10, 8))  # Adjust the figure size as needed
sns.heatmap(kendall_corr, annot=True, fmt=".2f", cmap="coolwarm")
```

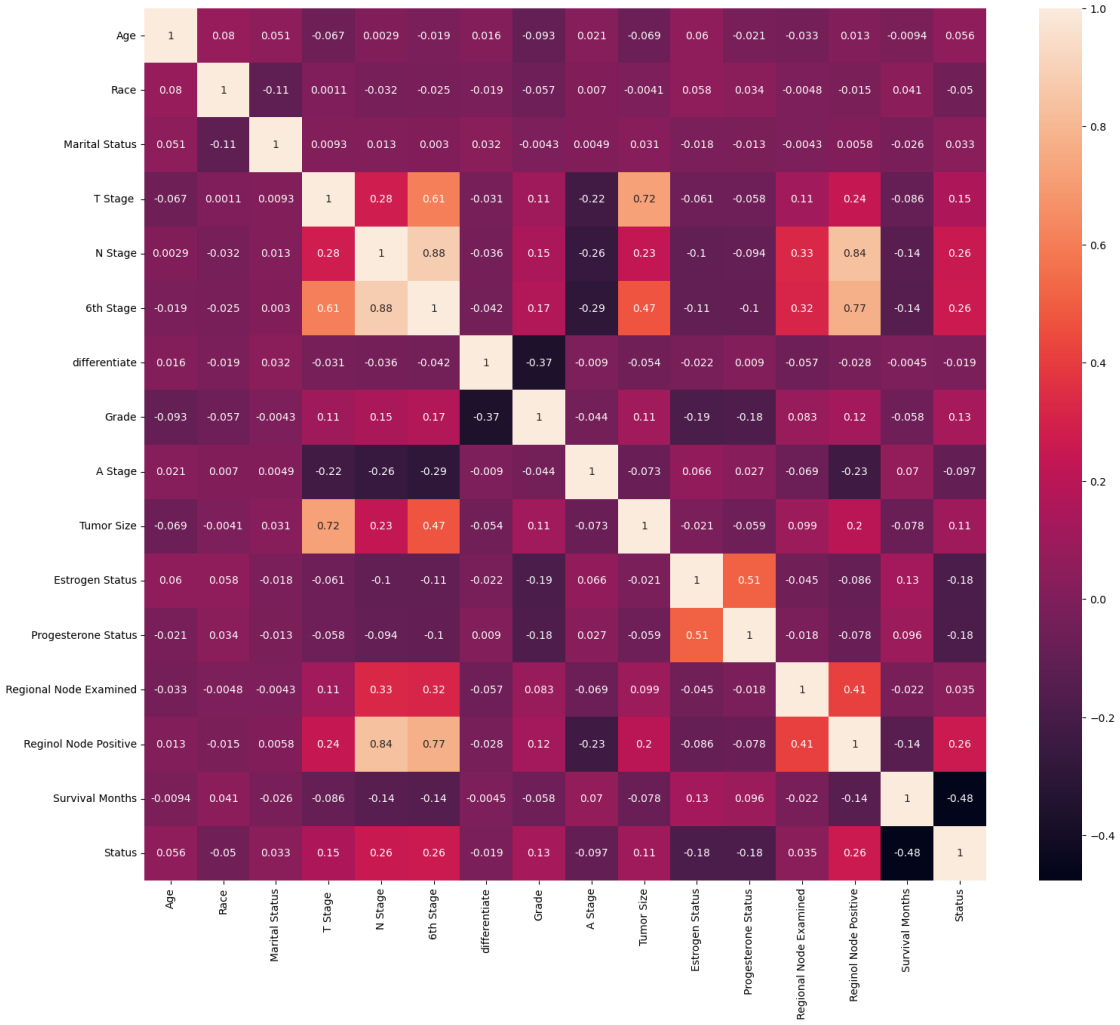
```
plt.title("Kendall Correlation Matrix")
plt.show()

plt.figure(figsize=(10, 8))  # Adjust the figure size as needed
sns.heatmap(spearman_corr, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Spearman Correlation Matrix")
plt.show()
```





[14]:
```
###  -----------   CORRELATION MATRIX ------------

corematrix = df.corr()
round(corematrix,2)
plt.figure(figsize=(18,15))
sns.heatmap(corematrix, annot=True)
```

[14]: <AxesSubplot:>



[15]:
```
# Print all column names to check
print(df.columns)
```

```
Index(['Age', 'Race', 'Marital Status', 'T Stage ', 'N Stage', '6th Stage',
       'differentiate', 'Grade', 'A Stage', 'Tumor Size', 'Estrogen Status',
       'Progesterone Status', 'Regional Node Examined',
       'Reginol Node Positive', 'Survival Months', 'Status'],
      dtype='object')
```

[16]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter
#Aim 2
#split into X/y
X = df[['Age', 'T Stage ', 'N Stage', '6th Stage',
       'differentiate', 'Grade', 'A Stage', 'Tumor Size', 'Estrogen Status',
       'Progesterone Status']]
```

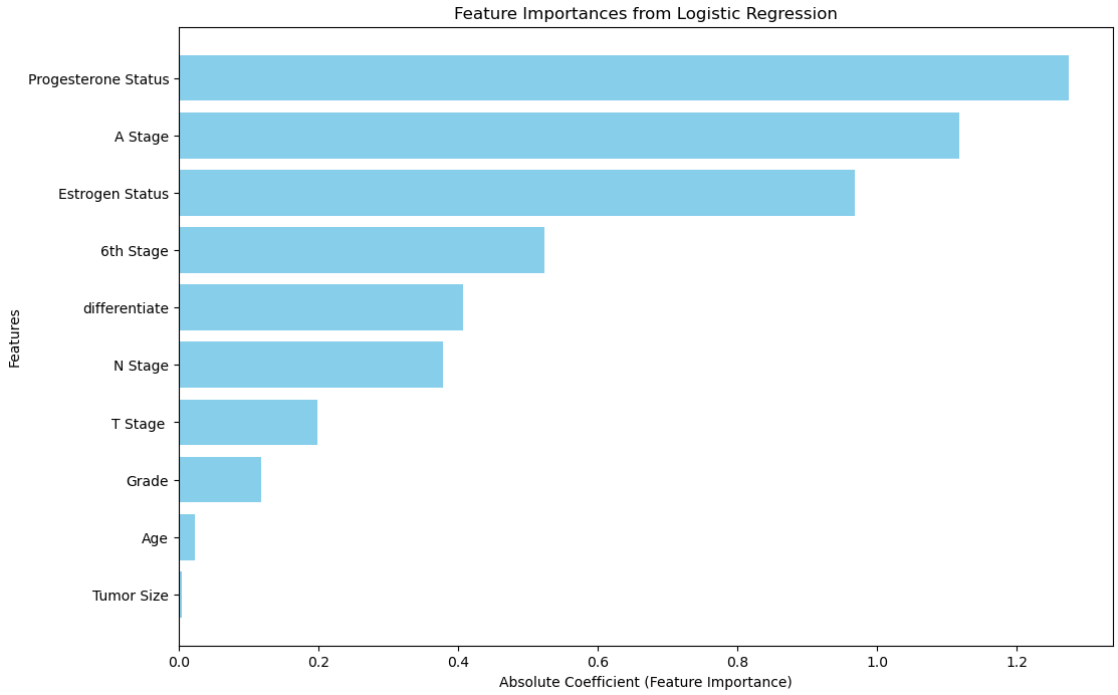
```
y = df["Status"]

# split into training and test
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.
 ↪2,random_state=42)


# Summarize class distribution
print("Before SMOTE:", Counter(y_train))

# Apply SMOTE
sm = SMOTE(random_state=42)
X_train, y_train = sm.fit_resample(X_train, y_train)

# Summarize new class distribution
print("After SMOTE:", Counter(y_train))
```

```
Before SMOTE: Counter({0: 2723, 1: 496})
After SMOTE: Counter({0: 2723, 1: 2723})
```

[17]:
```
### FEATURE IMPORTANCE ---------------    METHOD 2
###  FEATURE IMPORTANCE DERIVED FROM LOGISTIC REGRESSION


# Assuming X_train and y_train are already defined and LogisticRegression is␣
 ↪suitable for your dataset

# Train Logistic Regression model
lr_model = LogisticRegression(max_iter=1000)  # max_iter might need to be␣
 ↪increased if the algorithm doesn't converge
lr_model.fit(X_train, y_train)

# Get feature importances (coefficients)
lr_feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': np.abs(lr_model.coef_[0])
})

# Sort by importance
lr_feature_importances = lr_feature_importances.sort_values(by='Importance',␣
 ↪ascending=False)

# Display feature importances
print(lr_feature_importances)
```

```
# Visual Representation
plt.figure(figsize=(12, 8))
plt.barh(lr_feature_importances['Feature'],␣
 ↪lr_feature_importances['Importance'], color='skyblue')
plt.xlabel('Absolute Coefficient (Feature Importance)')
plt.ylabel('Features')
plt.title('Feature Importances from Logistic Regression')
plt.gca().invert_yaxis()  # To display the highest importance at the top
plt.show()
```

	Feature	Importance
9	Progesterone Status	1.273569
6	A Stage	1.116862
8	Estrogen Status	0.967466
3	6th Stage	0.523360
4	differentiate	0.407423
2	N Stage	0.378392
1	T Stage	0.198577
5	Grade	0.118567
0	Age	0.023676
7	Tumor Size	0.004033

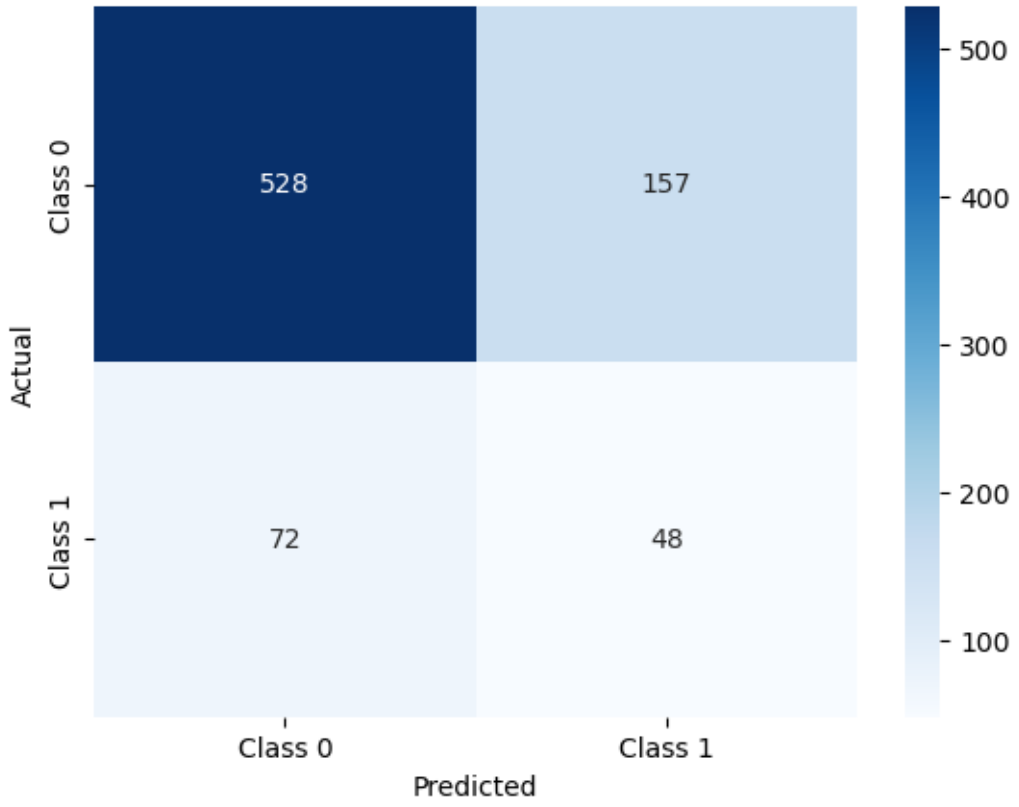


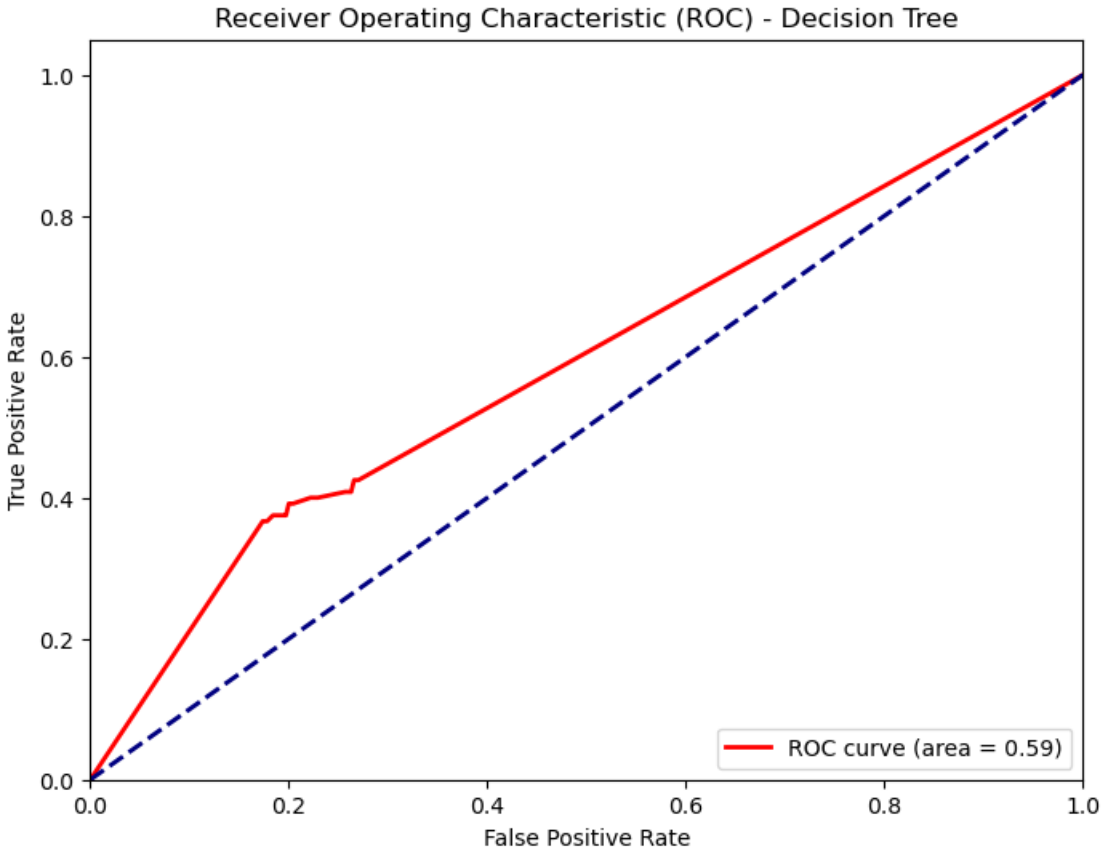
[18]:
```
##----------------    DECISION TREE CLASSIFIER MODEL -------------

import seaborn as sns
```

```
from sklearn.metrics import confusion_matrix
columns_consider= ['Age', 'T Stage ', 'N Stage', '6th Stage',
       'differentiate', 'Grade', 'A Stage', 'Tumor Size', 'Estrogen Status',
       'Progesterone Status']
# Model building using DECISION TREE
dtc = DecisionTreeClassifier(random_state = 606)
dtc.fit(X_train[columns_consider], y_train)
y_pred_dt= dtc.predict(X_test[columns_consider])

# calculating Accuracy
print("Accuracy: \t",accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))
print("Confusin Matrix: \n",confusion_matrix(y_test, y_pred_dt))

# Visual Representation of confusion matrix

cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0",␣
 ↪"Class 1"], yticklabels=["Class 0", "Class 1"])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()


# ROC for decision tree classifier
dtc = DecisionTreeClassifier(random_state=606)
dtc.fit(X_train[columns_consider], y_train)

y_scores_dt = dtc.predict_proba(X_test[columns_consider])[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_scores_dt)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Decision Tree')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy:        0.715527950310559
              precision    recall  f1-score   support
```

```
              0       0.88      0.77      0.82       685
              1       0.23      0.40      0.30       120

    accuracy                            0.72       805
   macro avg       0.56      0.59      0.56       805
weighted avg       0.78      0.72      0.74       805

Confusin Matrix:
 [[528 157]
 [ 72  48]]
```

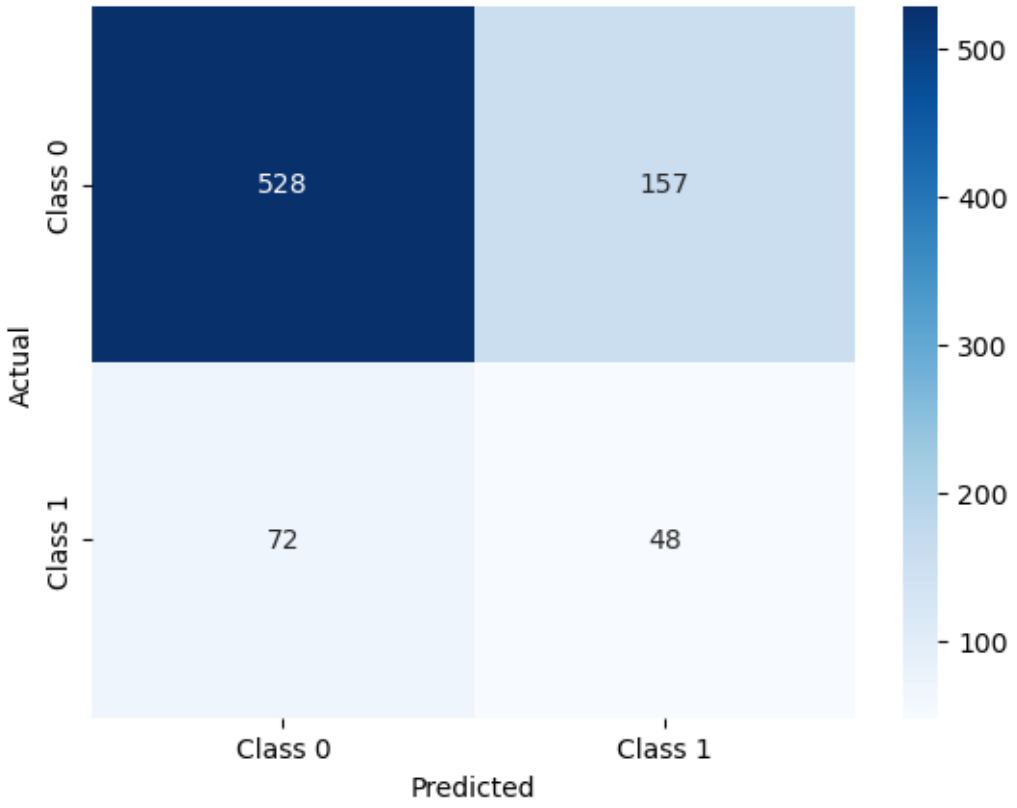


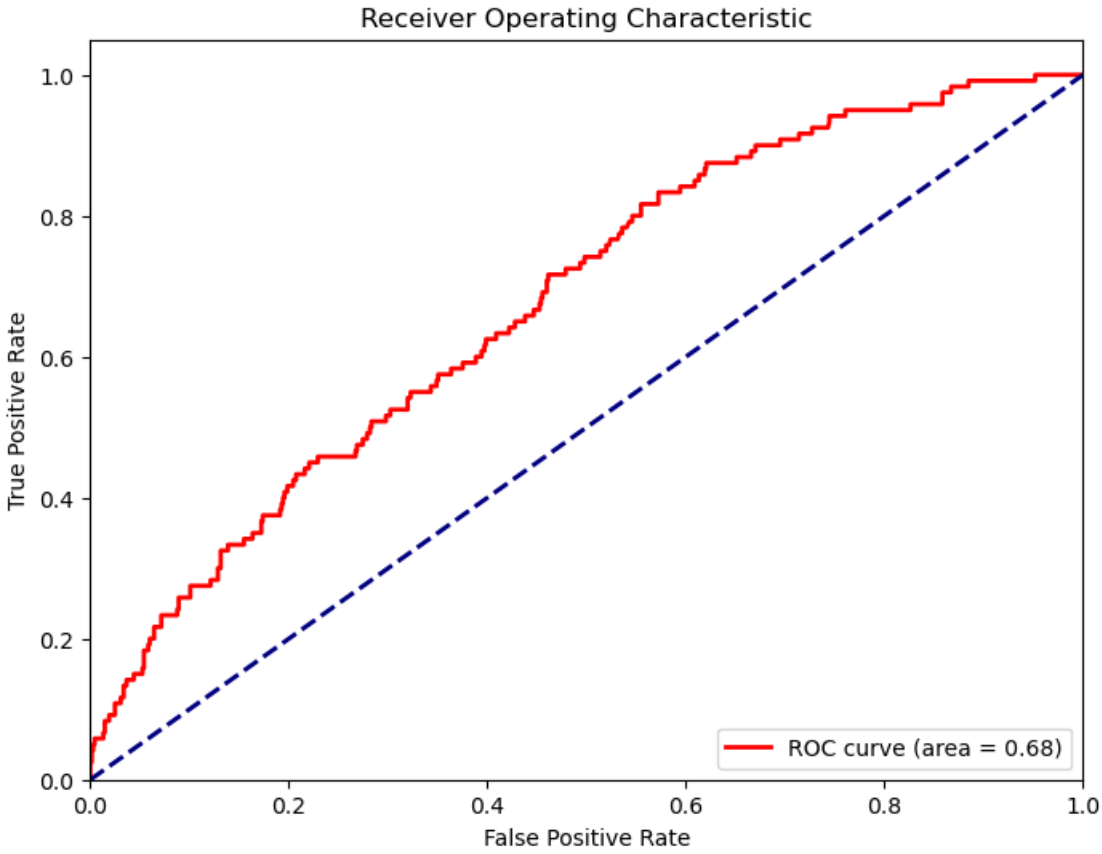


[19]:
```
## -------------------------  XBBOOST CLASSIFIER MODEL ␣
 ↪--------------------------------

# Model building using XGBOOST
columns_required=columns_consider
xgbc = xgb.XGBClassifier(random_state = 606)
xgbc.fit(X_train[columns_required], y_train)
y_pred_xg= xgbc.predict(X_test[columns_required])

#calculating Accuracy
print("Accuracy: \t",accuracy_score(y_test, y_pred_xg))
print(classification_report(y_test, y_pred_xg))
print("Confusin Matrix: \n",confusion_matrix(y_test, y_pred_xg))

# Visual Representation of confusion matrix

cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0",␣
 ↪"Class 1"], yticklabels=["Class 0", "Class 1"])
plt.ylabel('Actual')
```

```
plt.xlabel('Predicted')
plt.show()

# ROC for XG Boost classifier
y_pred_xg = xgbc.predict_proba(X_test[columns_required])[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_xg)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy:         0.7416149068322981
              precision    recall  f1-score   support

           0       0.89      0.80      0.84       685
           1       0.27      0.42      0.32       120

    accuracy                           0.74       805
   macro avg       0.58      0.61      0.58       805
weighted avg       0.79      0.74      0.76       805

Confusin Matrix:
 [[547 138]
 [ 70  50]]
```

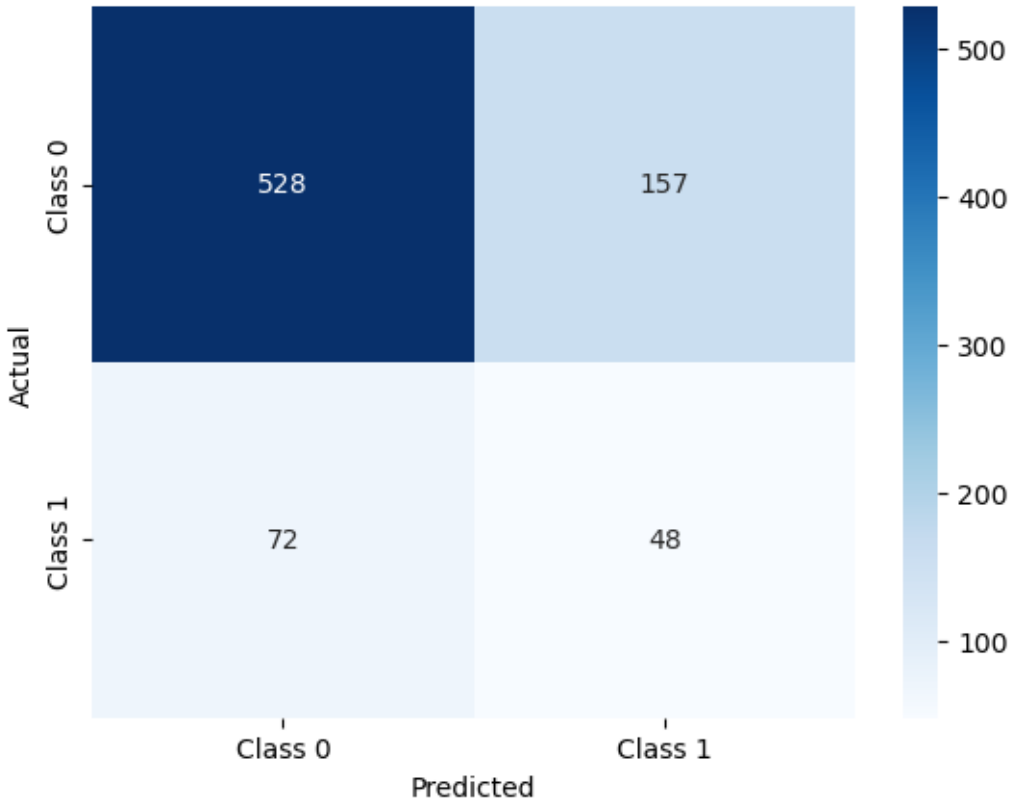

528
157
72
48
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
500
400
300
200
100

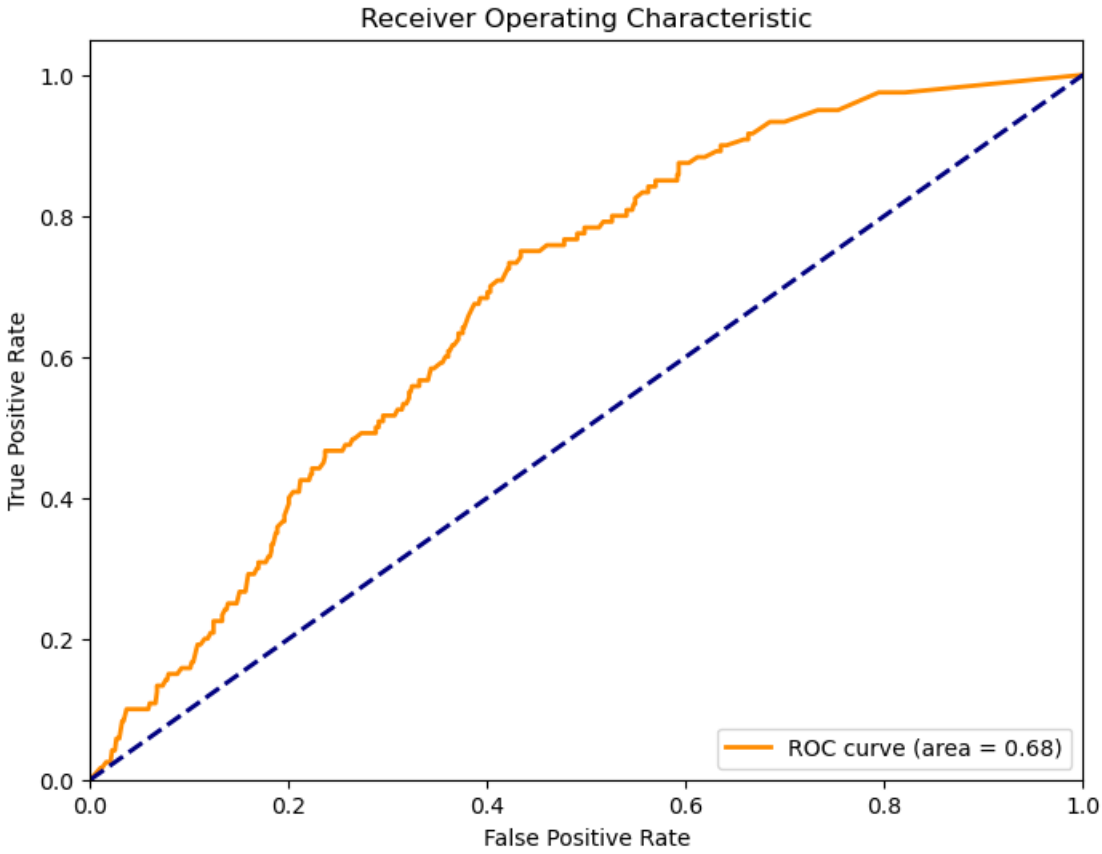


[20]:
```
# Model building using RANDOMFOREST
rfc = RandomForestClassifier(random_state = 606)
rfc.fit(X_train[columns_required], y_train)
y_pred= rfc.predict(X_test[columns_required])
print("Accuracy: \t",accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("Confusin Matrix: \n",confusion_matrix(y_test, y_pred))

# Visual Representation of confusion matrix

cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0",␣
 ↪"Class 1"], yticklabels=["Class 0", "Class 1"])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# ROC for Random Forest Classifier

rfc = RandomForestClassifier(random_state = 606)
```

```
rfc.fit(X_train[columns_required], y_train)
y_pred = rfc.predict(X_test[columns_required])

y_pred_prob = rfc.predict_proba(X_test[columns_required])[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy:         0.7267080745341615
              precision    recall  f1-score   support

           0       0.89      0.78      0.83       685
           1       0.25      0.42      0.32       120

    accuracy                           0.73       805
   macro avg       0.57      0.60      0.57       805
weighted avg       0.79      0.73      0.75       805

Confusin Matrix:
 [[534 151]
 [ 69  51]]
```

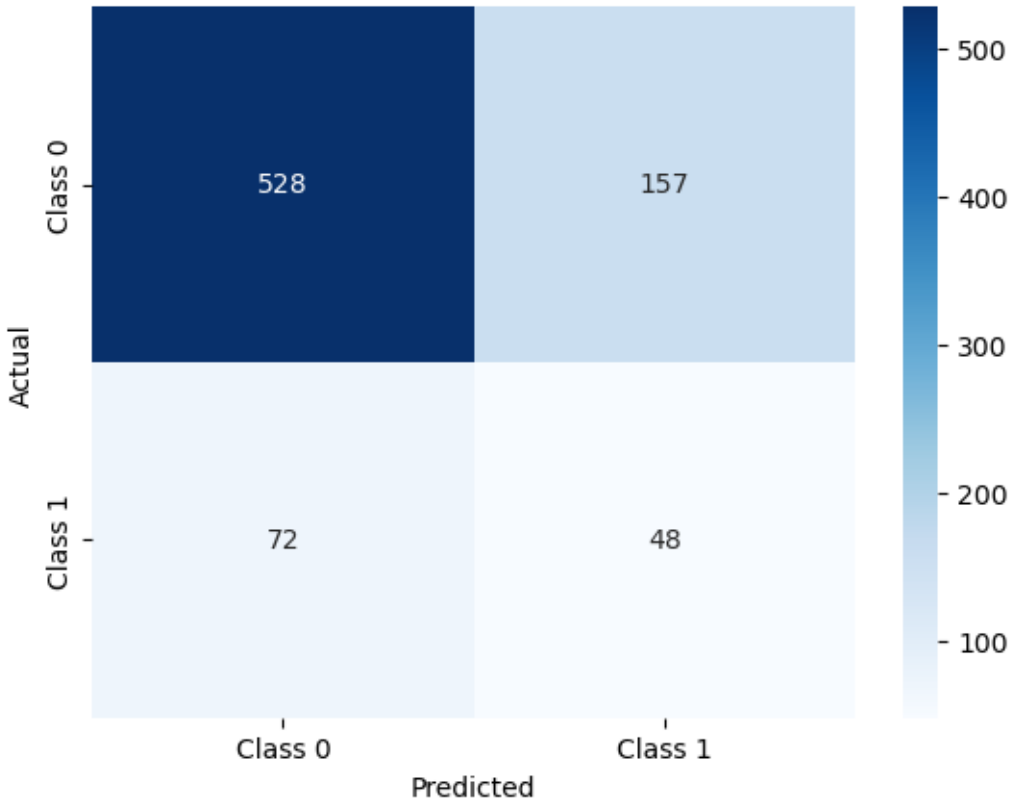

528
157
72
48
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
500
400
300
200
100



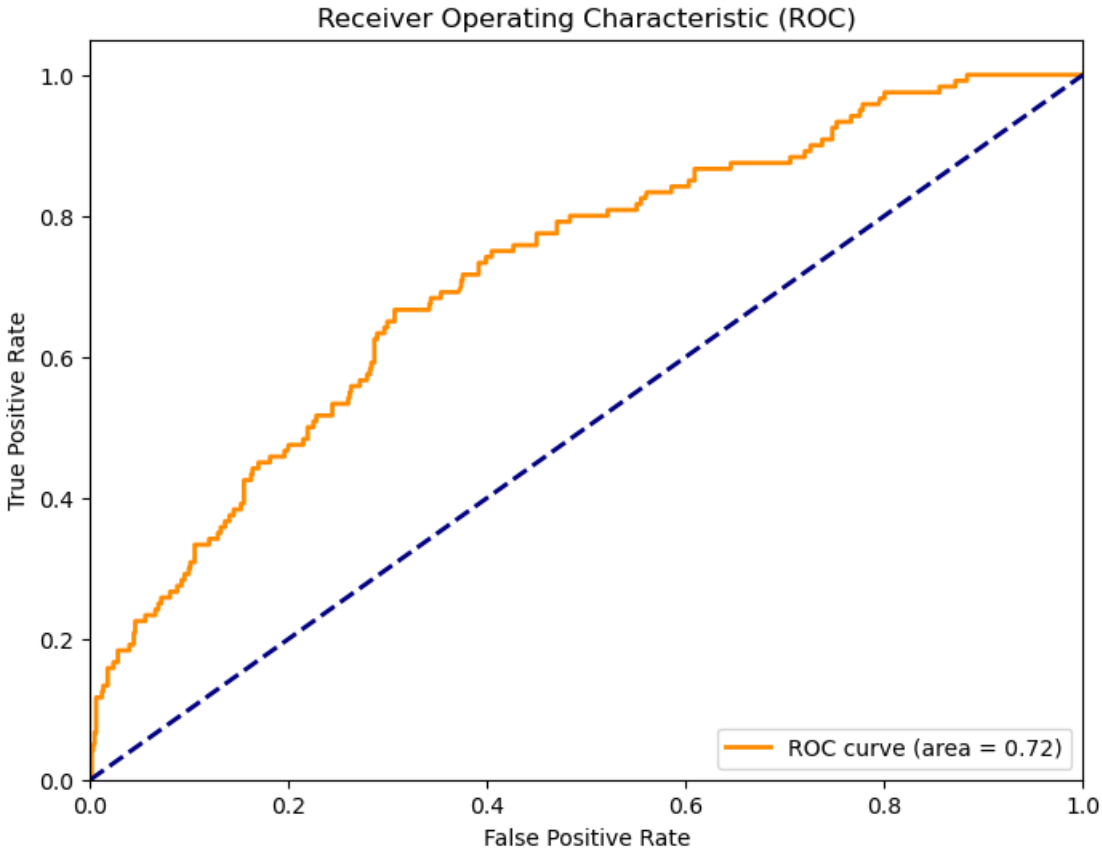
[21]:
```
# Model building using Logistic Regression
lrc = LogisticRegression()
lrc.fit(X_train[columns_required], y_train)
y_pred_lr= lrc.predict(X_test[columns_required])


#calculating Accuracy
print("Accuracy: \t",accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
print("Confusin Matrix: \n",confusion_matrix(y_test, y_pred_lr))

# Visual Representation of confusion matrix

cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0",␣
 ↪"Class 1"], yticklabels=["Class 0", "Class 1"])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
# ROC FOR LOGISTIC REGRESSION

y_pred_prob = lrc.predict_proba(X_test[columns_required])[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
```

```
Accuracy:        0.7329192546583851
              precision    recall  f1-score   support

           0       0.90      0.77      0.83       685
           1       0.28      0.52      0.37       120

    accuracy                           0.73       805
   macro avg       0.59      0.64      0.60       805
weighted avg       0.81      0.73      0.76       805

Confusin Matrix:
 [[528 157]
 [ 58  62]]
```



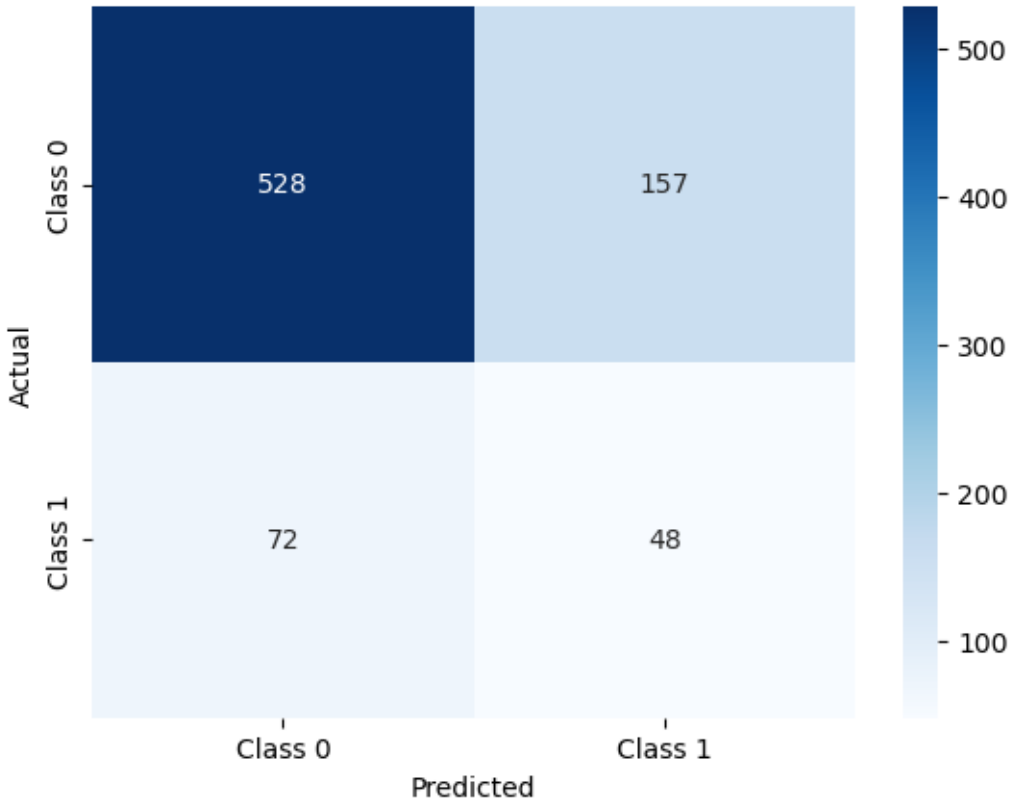
[21]: <matplotlib.legend.Legend at 0x1b9659dc550>

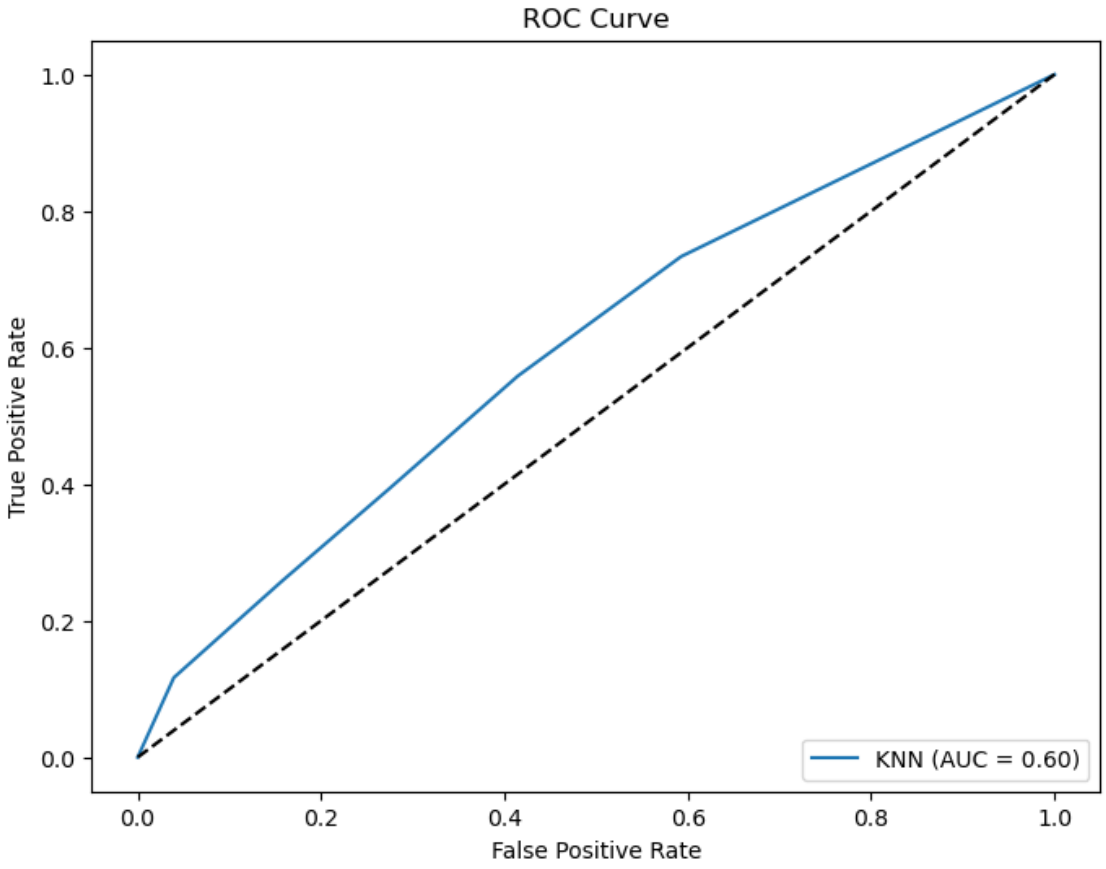


[22]:
```
# Model building using KNN Classifier
knc = KNeighborsClassifier( metric='minkowski')
knc.fit(X_train[columns_required], y_train)
y_pred_knc= knc.predict(X_test[columns_required])
#Calculating Accuracy
print("Accuracy: \t",accuracy_score(y_test, y_pred_knc))
print(classification_report(y_test, y_pred_knc))
print("Confusin Matrix: \n",confusion_matrix(y_test, y_pred_knc))

# Visual Representation of confusion matrix

cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0",␣
 ↪"Class 1"], yticklabels=["Class 0", "Class 1"])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

y_pred_prob_knc = knc.predict_proba(X_test[columns_required])[:, 1]
```

```
# Calculate ROC Curve
fpr_knc, tpr_knc, thresholds_knc = roc_curve(y_test, y_pred_prob_knc)

# Calculate AUC (Area under the ROC Curve)
auc_knc = auc(fpr_knc, tpr_knc)

# Plotting ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_knc, tpr_knc, label=f'KNN (AUC = {auc_knc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

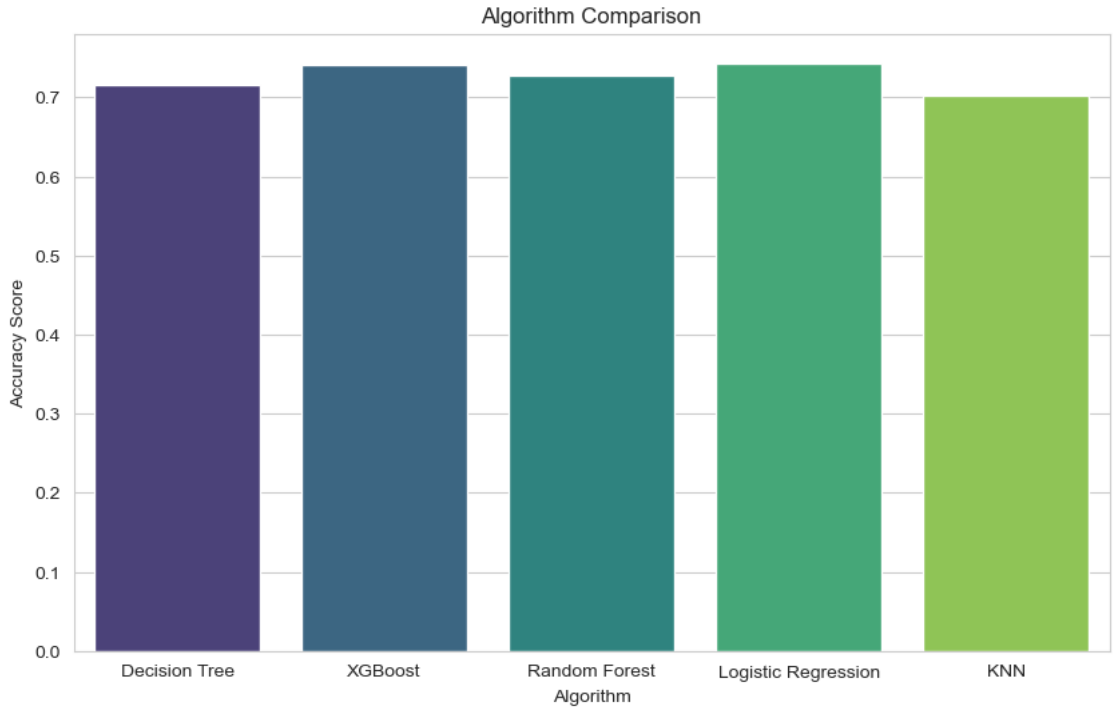
```
Accuracy:         0.6770186335403726
              precision    recall  f1-score   support

           0       0.87      0.73      0.79       685
           1       0.20      0.39      0.27       120

    accuracy                           0.68       805
   macro avg       0.54      0.56      0.53       805
weighted avg       0.77      0.68      0.71       805

Confusin Matrix:
 [[498 187]
 [ 73  47]]
```


528
157
72
48
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
500
400
300
200
100



[23]:
```
# Assuming X_train, X_test, y_train, y_test are already defined and␣
 ↪'columns_consider' is your list of features

# Decision Tree
dtc = DecisionTreeClassifier(random_state=606)
dtc.fit(X_train[columns_consider], y_train)
y_pred_dt = dtc.predict(X_test[columns_consider])
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# XGBoost
xgbc = xgb.XGBClassifier(random_state=606)
xgbc.fit(X_train[columns_consider], y_train)
y_pred_xg = xgbc.predict(X_test[columns_consider])
accuracy_xgb = accuracy_score(y_test, y_pred_xg)

# Random Forest
rfc = RandomForestClassifier(random_state=606)
rfc.fit(X_train[columns_consider], y_train)
y_pred_rf = rfc.predict(X_test[columns_consider])
```

```
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Logistic Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train[columns_consider])
X_test_scaled = scaler.transform(X_test[columns_consider])

lrc = LogisticRegression()
lrc.fit(X_train_scaled, y_train)
y_pred_lr = lrc.predict(X_test_scaled)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# KNN Classifier
knc = KNeighborsClassifier(metric='minkowski')
knc.fit(X_train_scaled, y_train)
y_pred_knc = knc.predict(X_test_scaled)
accuracy_knn = accuracy_score(y_test, y_pred_knc)

# Create a dictionary of accuracy scores
accuracy_scores = {
    'Decision Tree': accuracy_dt,
    'XGBoost': accuracy_xgb,
    'Random Forest': accuracy_rf,
    'Logistic Regression': accuracy_lr,
    'KNN': accuracy_knn
}

# Convert dictionary to lists for plotting
algorithms = list(accuracy_scores.keys())
scores = list(accuracy_scores.values())

# Create bar plot
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
sns.barplot(x=algorithms, y=scores, palette="viridis")

# Adding the aesthetics
plt.title('Algorithm Comparison')
plt.xlabel('Algorithm')
plt.ylabel('Accuracy Score')

# Show the plot
plt.show()
```



[24]:
```
#Building machine learning model
model = LogisticRegression()

model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[24]: 0.7329192546583851

[25]:
```
# Cross validation of machine learning
scores = cross_val_score(model, X, y, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

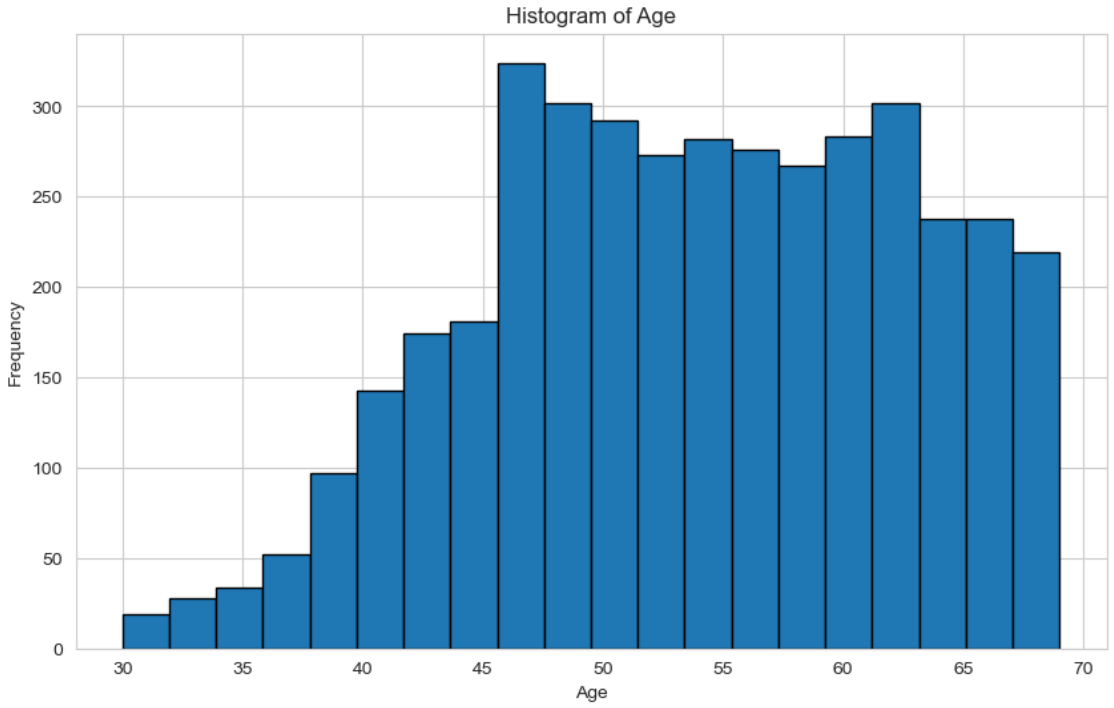
```
Accuracy: 0.85 (+/- 0.01)
```

[26]:
```
##-------------- PERFORM T TEST -------------------------
##----- CHECKING THE MEAN DIFFERENCE BETWEEN STATUS AND EACH COLUMN , IF THEY␣
 ↪ARE SIGNIFICANTLY DIFFERENT.
##  IF P<0.05 , THERE IS A RELATION


import scipy.stats as stats
import pandas as pd
```

```
# Assuming df is your DataFrame and it includes 'Status' and columns in␣
 ↪'columns_consider'
columns_consider = ["Age", "Race", "Marital Status", "T Stage ", "N Stage",␣
 ↪"6th Stage",
                    "differentiate", "Grade", "A Stage", "Tumor Size",␣
 ↪"Estrogen Status",
                    "Progesterone Status", "Regional Node Examined", "Reginol␣
 ↪Node Positive",
                    'Survival Months']
group1_value = 0
group2_value = 1

t_test_results = pd.DataFrame(columns=['Feature', 'T-Statistic', 'P-Value'])

for column in columns_consider:
    # Perform t-test for each numeric column
    group1 = df[df['Status'] == group1_value][column]
    group2 = df[df['Status'] == group2_value][column]

    # Check if the column is numeric before performing t-test
    if pd.api.types.is_numeric_dtype(df[column]):
        t_stat, p_value = stats.ttest_ind(group1, group2, equal_var=False,␣
 ↪nan_policy='omit')
        t_test_results = t_test_results.append({'Feature': column,␣
 ↪'T-Statistic': t_stat, 'P-Value': p_value}, ignore_index=True)
    else:
        print(f"Skipped t-test for {column} as it is not numeric.")

print(t_test_results)
```

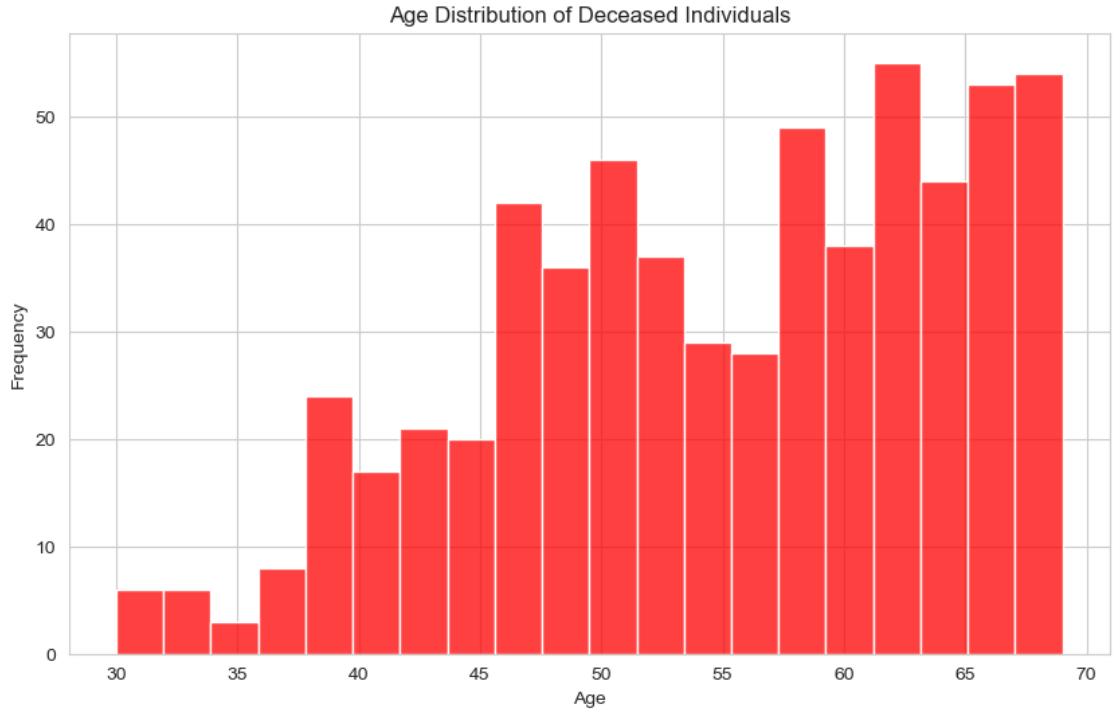
	Feature	T-Statistic	P-Value
0	Age	-3.322896	9.309078e-04
1	Race	2.760770	5.903773e-03
2	Marital Status	-1.957499	5.063470e-02
3	T Stage	-9.129611	5.520297e-19
4	N Stage	-13.796728	9.931253e-39
5	6th Stage	-14.834571	5.296667e-44
6	differentiate	1.416397	1.569583e-01
7	Grade	-8.319897	3.593153e-16
8	A Stage	4.180761	3.282492e-05
9	Tumor Size	-6.670446	4.649658e-11
10	Estrogen Status	8.128191	2.036183e-15
11	Progesterone Status	9.351384	9.918444e-20
12	Regional Node Examined	-2.122260	3.411297e-02
13	Reginol Node Positive	-11.977077	3.440363e-30
14	Survival Months	29.702853	1.192571e-129

[27]:
```
### -------------  Visualizaztions --------------------

##   Distribution of People Age in the Data

# Assuming your DataFrame is named 'data' and has a column 'Age'
plt.figure(figsize=(10, 6))

# Create a histogram for the 'Age' column
plt.hist(df['Age'], bins=20, edgecolor='black')

# Add title and labels
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```



[28]:
```
###  ---------- Age Distribution of Breast Cancer Deceased -----------------

# Assuming your DataFrame is named 'data', 'Age' is the age column, and␣
 ↪'Status' is the status column
# Let's assume '1' in 'Status' indicates death
```

```
# Filter the DataFrame for individuals who died
deceased = df[df['Status'] == 1]

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

# Create a histogram for the 'Age' of deceased individuals
sns.histplot(deceased['Age'], bins=20, kde=False, color='red')

# Add title and labels
plt.title('Age Distribution of Deceased Individuals')
plt.xlabel('Age')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```



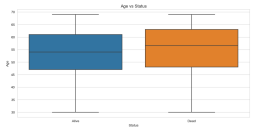
[29]:
```
tumor_size_summary = df.describe()

# Print the summary
print(tumor_size_summary)
```

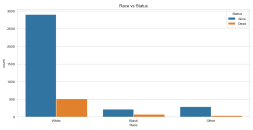
```
                Age         Race  Marital Status     T Stage       N Stage  \
count   4024.000000  4024.000000     4024.000000  4024.000000  4024.000000
mean      53.972167     1.775845        1.371272     0.784791     0.438370
```

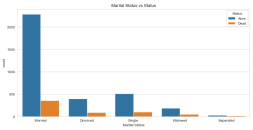
```
std       8.963134     0.564465        1.063637     0.765531     0.693479
min      30.000000     0.000000        0.000000     0.000000     0.000000
25%      47.000000     2.000000        1.000000     0.000000     0.000000
50%      54.000000     2.000000        1.000000     1.000000     0.000000
75%      61.000000     2.000000        1.000000     1.000000     1.000000
max      69.000000     2.000000        4.000000     3.000000     2.000000

          6th Stage  differentiate        Grade      A Stage   Tumor Size  \
count  4024.000000    4024.000000  4024.000000  4024.000000  4024.000000
mean      1.321819       0.690358     2.131710     0.977137    26.813370
std       1.266624       1.016418     0.642398     0.149485    14.647782
min       0.000000       0.000000     0.000000     0.000000     1.000000
25%       0.000000       0.000000     2.000000     1.000000    16.000000
50%       1.000000       0.000000     2.000000     1.000000    25.000000
75%       2.000000       1.000000     3.000000     1.000000    35.000000
max       4.000000       3.000000     3.000000     1.000000    70.000000

       Estrogen Status  Progesterone Status  Regional Node Examined  \
count      4024.000000          4024.000000             4024.000000
mean          0.933151             0.826541               14.357107
std           0.249791             0.378691                8.099675
min           0.000000             0.000000                1.000000
25%           1.000000             1.000000                9.000000
50%           1.000000             1.000000               14.000000
75%           1.000000             1.000000               19.000000
max           1.000000             1.000000               61.000000

       Reginol Node Positive  Survival Months       Status
count            4024.000000      4024.000000  4024.000000
mean                4.158052        71.297962     0.153082
std                 5.109331        22.921430     0.360111
min                 1.000000         1.000000     0.000000
25%                 1.000000        56.000000     0.000000
50%                 2.000000        73.000000     0.000000
75%                 5.000000        90.000000     0.000000
max                46.000000       107.000000     1.000000
```

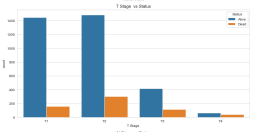
[30]:
```python
df = pd.read_csv("C:\\Users\\prane\\Downloads\\Breast_Cancer_1.csv")
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns

# Assuming 'Status' is your target column
target_col = 'Status'

# Define the number of rows needed for subplots
num_rows = len(df.columns) - 1  # excluding the target column
```

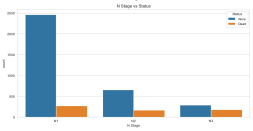
```
# Create subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=1, figsize=(10, 5 * num_rows))

# Iterate through columns and create plots
for i, column in enumerate(df.columns):
    if column == target_col:
        continue  # Skip the target column

    ax = axes[i] if num_rows > 1 else axes  # Handle case of single subplot

    if column in numeric_cols:
        # For numeric columns, use a boxplot or scatterplot
        sns.boxplot(x=target_col, y=column, data=df, ax=ax)


    elif column in categorical_cols:
        # For categorical columns, use a countplot or barplot
        sns.countplot(x=column, hue=target_col, data=df, ax=ax)
        # Alternatively, for barplot: sns.barplot(x=target_col, y=column,␣
 ↪data=df, ax=ax)

    ax.set_title(f'{column} vs {target_col}')

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

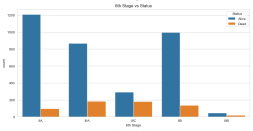

Age vs Status
Age
Status
Alive
Dead

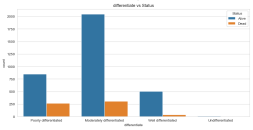

Race vs Status
Status
Alive
Dead
count
Race
White
Black
Other

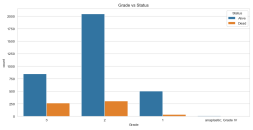

Marital Status vs Status
Status
Alive
Dead
count
Married
Divorced
Single
Widowed
Separated
Marital Status

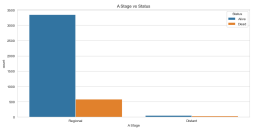

T Stage vs Status
Status
Alive
Dead
count
T1
T2
T3
T4
T Stage

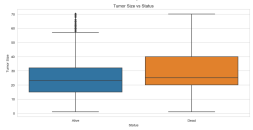

N Stage vs Status
Status
Alive
Dead
count
N1
N2
N3
N Stage

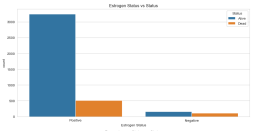

6th Stage vs Status
Status
Alive
Dead
count
IIIA
IIIB
IIIC
IIIA
IIIB
6th Stage

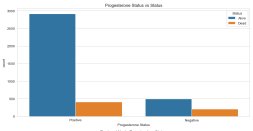

differentiate vs Status
Status
Alive
Dead
count
Poorly differentiated
Moderately differentiated
differentiate
Undifferentiated

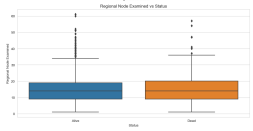

Grade vs Status
Status
Alive
Dead
count
Grade

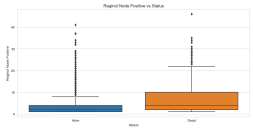

A Stage vs Status
Status
Alive
Dead
count
Regional
Distant
A Stage

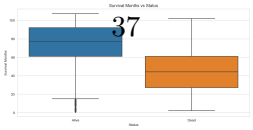

Tumor Size vs Status
Tumor Size
Status
Alive
Dead


Estrogen Status vs Status
Status
Alive
Dead
count
Positive
Negative
Estrogen Status

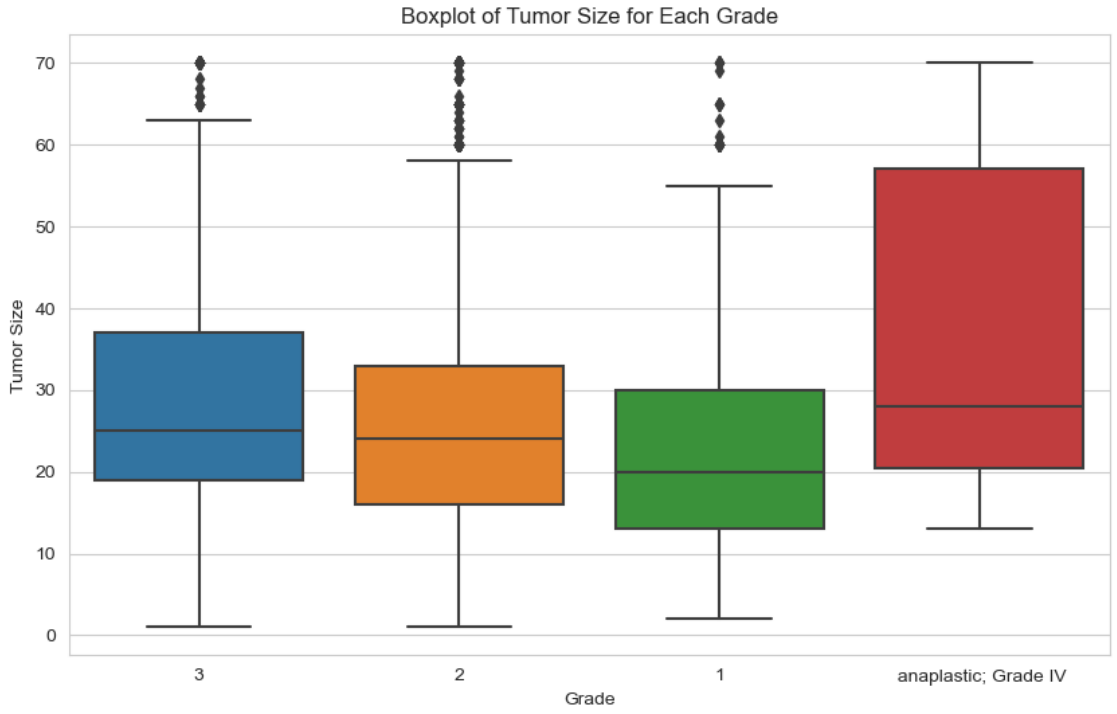

Progesterone Status vs Status
Status
Alive
Dead
count
Positive
Negative
Progesterone Status


Regional Node Examined vs Status
Regional Node Examined
Status
Alive
Dead

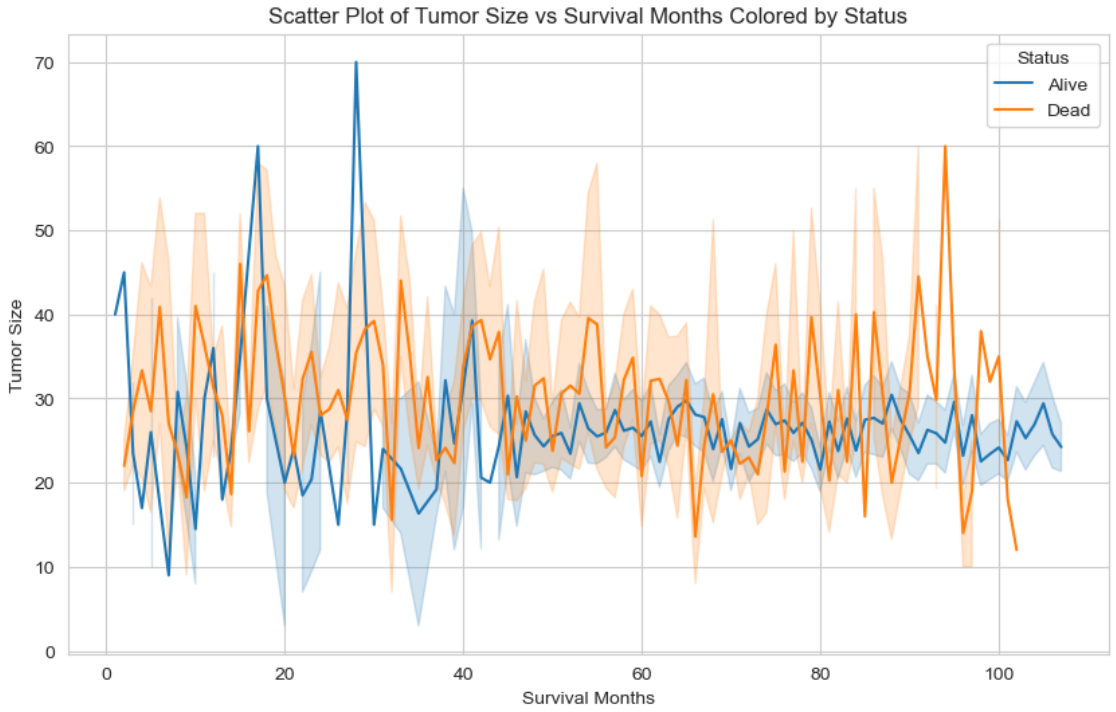

Regional Node Positive vs Status
Regional Node Positive
Status
Alive
Dead


Survival Months vs Status
Survival Months
Status
Alive
Dead

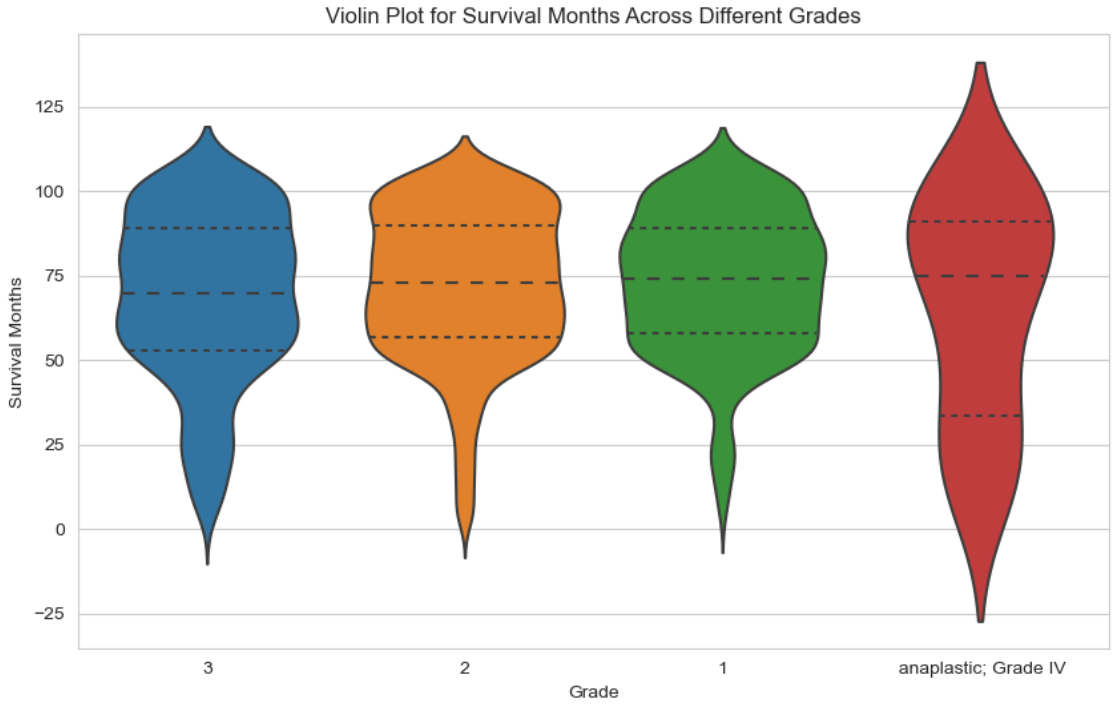
[31]:
```
# Visualization codes
data=df
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Grade', y='Tumor Size')
plt.title('Boxplot of Tumor Size for Each Grade')
plt.xlabel('Grade')
plt.ylabel('Tumor Size')
plt.show()
```



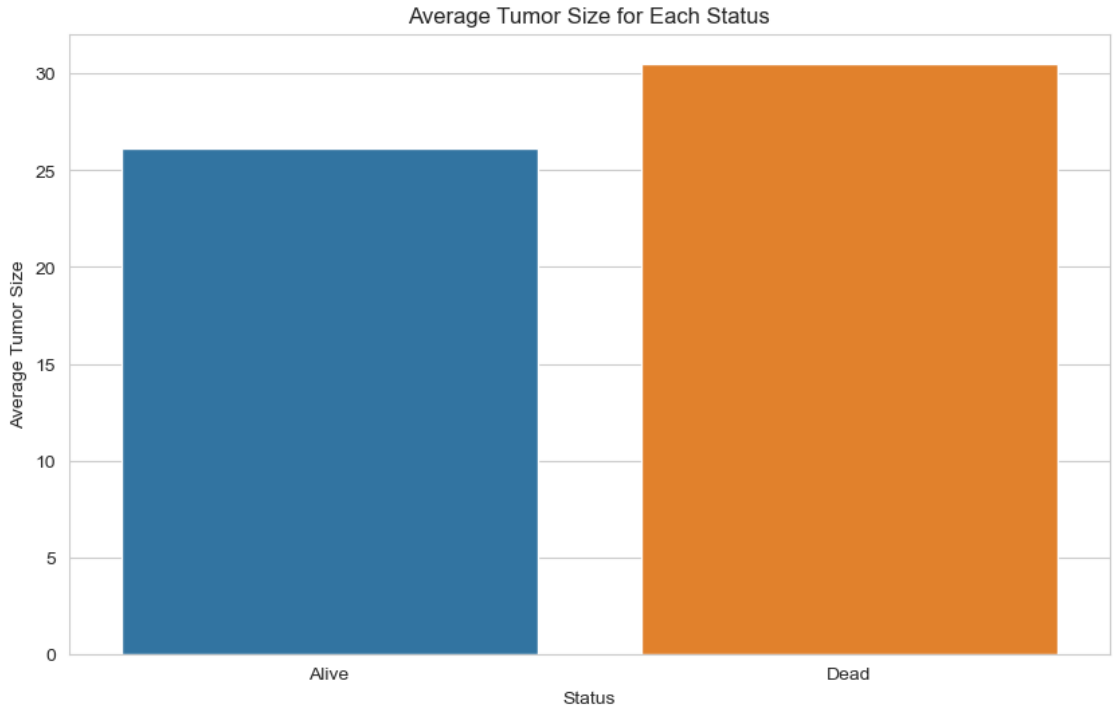
[32]:
```
plt.figure(figsize=(10, 6))
sns.lineplot(data=data, x='Survival Months', y='Tumor Size', hue='Status')
plt.title('Scatter Plot of Tumor Size vs Survival Months Colored by Status')
plt.xlabel('Survival Months')
plt.ylabel('Tumor Size')
plt.show()
```



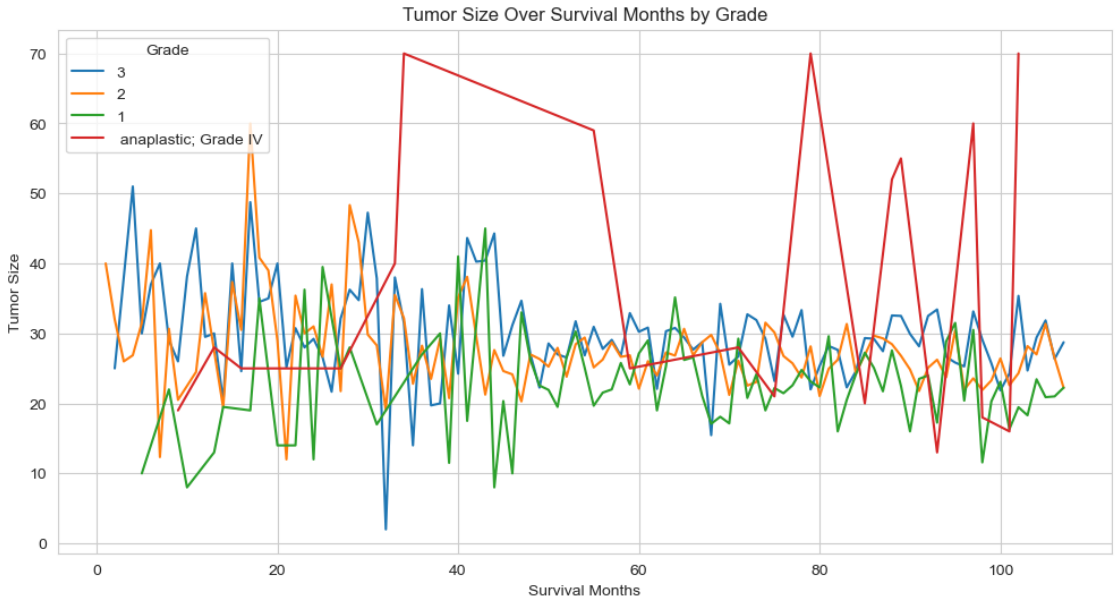
[33]:
```
plt.figure(figsize=(10, 6))
sns.violinplot(data=data, x='Grade', y='Survival Months', inner='quartile')
plt.title('Violin Plot for Survival Months Across Different Grades')
plt.xlabel('Grade')
plt.ylabel('Survival Months')
plt.show()
```



[34]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=data, x='Status', y='Tumor Size', ci=None)
plt.title('Average Tumor Size for Each Status')
plt.xlabel('Status')
plt.ylabel('Average Tumor Size')
plt.show()
```



[35]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(data=data, x='Survival Months', y='Tumor Size', hue='Grade',␣
 ↪ci=None)
plt.title('Tumor Size Over Survival Months by Grade')
plt.xlabel('Survival Months')
plt.ylabel('Tumor Size')
plt.legend(title='Grade')
plt.show()
```


Tumor Size Over Survival Months by Grade
Grade
3
2
1
anaplastic; Grade IV
Tumor Size
Survival Months
0
10
20
30
40
50
60
70
0
20
40
60
80
100

[ ]: