**BREAST CANCER ANALYISIS** EDA AND PREDICTION

Breast cancer is a cancer that forms in the cells of the breast. Breast cancer can occur in women and rarely in men. It requires a medical diagnosis, through lab tests or imaging.

We have data on patients . The main goal of this notebook is to build a machine learning model, that will be able to predict whether a patient will survive breast cancer or not.

# Data understanding

This dataset consists of a group of breast cancer patients, who had surgery to remove the tumour. The dataset consists of these variables:

**Patient\_ID**: unique identifier id of a patient

**Age**: Age of patient (Years) **Gender**: Male/Female.

**Protein1, Protein2, Protein3, Protein4**: These expression levels are biomarker for the dectection of breast cancer.

**Tumour\_Stage**: I, II, III. It is a form of staging to describe the extensiveness of breast cancer, based on the size of the tumor and its spread to the lymph nodes.

**Histology**: Infiltrating Ductal Carcinoma, Infiltrating Lobular Carcinoma, Mucinous Carcinoma.

**ER status**: Positive/Negative:(Estrogen receptor) the hormone receptor determines the type of breast cancer to determine treatment.

**PR status**: Positive/Negative:(Progesterone receptor)the hormone receptor determines the type of breast cancer to determine treatment.

**HER2 status**: Positive/Negative.(human epidermal growth factor receptor2). This protein promotes the growth of cancer cells .

**Surgery\_type**: Lumpectomy, Simple Mastectomy, Modified Radical Mastectomy, Other surgery type.

**Date of Surgery**: the date the surgery was performed (in DD-MON-YY).

**Date of Last\_Visit**: Date of last visit (in DD-MON-YY) to the hospital.

**Patient\_Status**: If Alive or Dead

## Import libraries

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

from sklearn.model\_selection import GridSearchCV

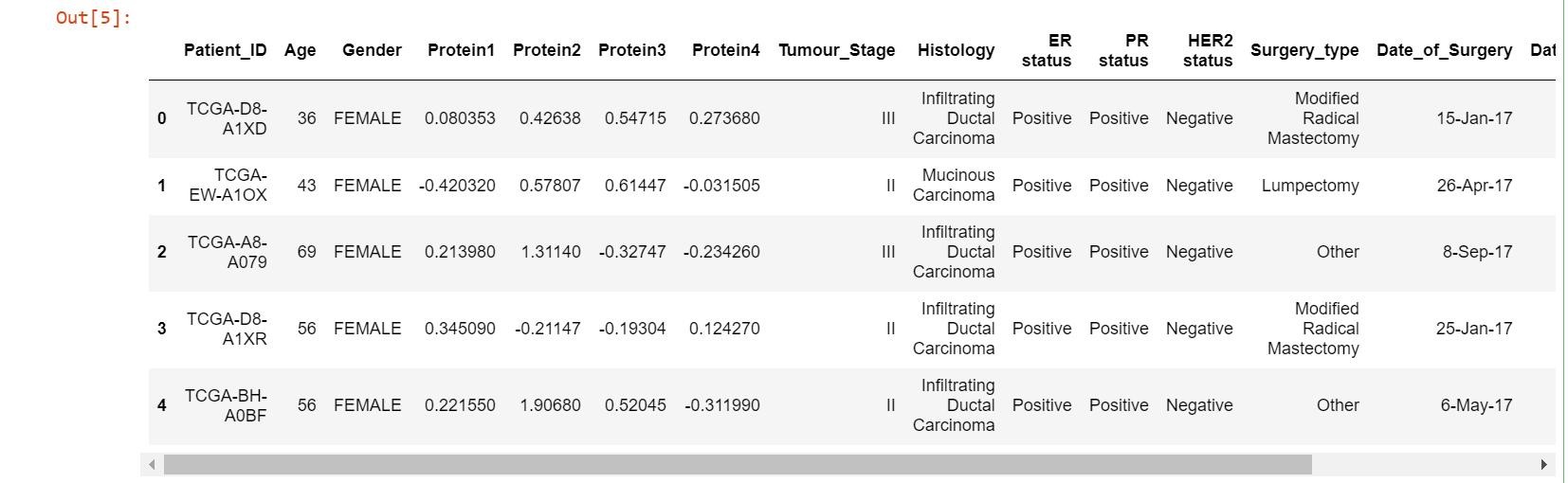
**# importing the dataset**

data=pd.read\_csv("C:\\Users\\AkankshaAdase\\Downloads\\BRCA.csv")

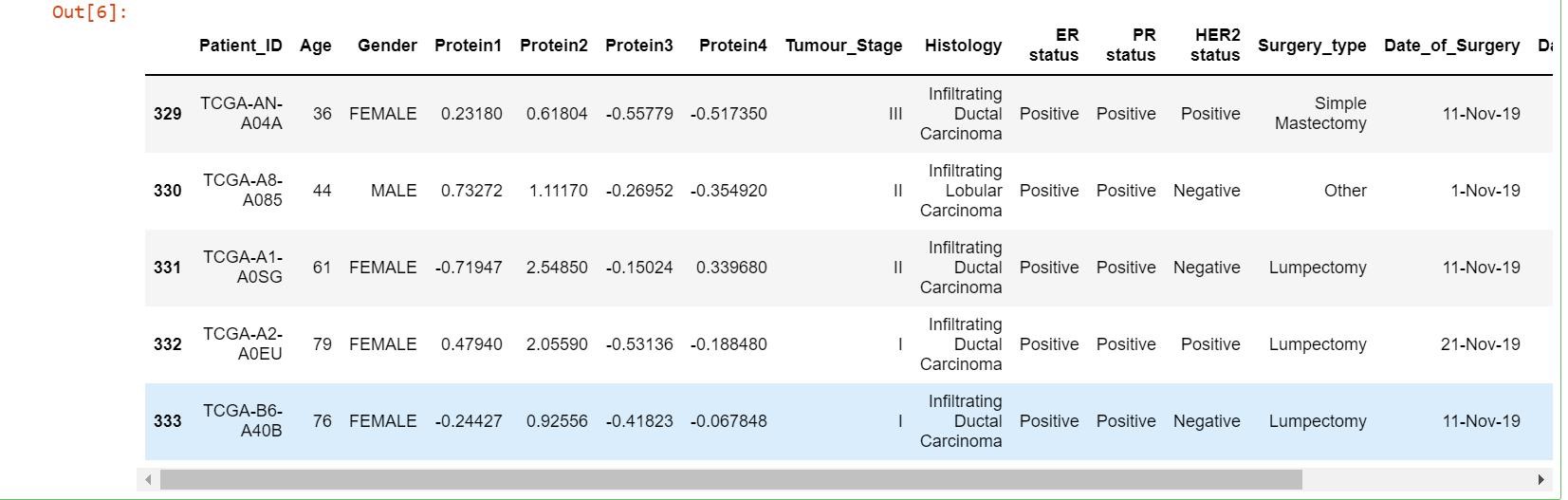
### # inspecting the dataset

data.head()

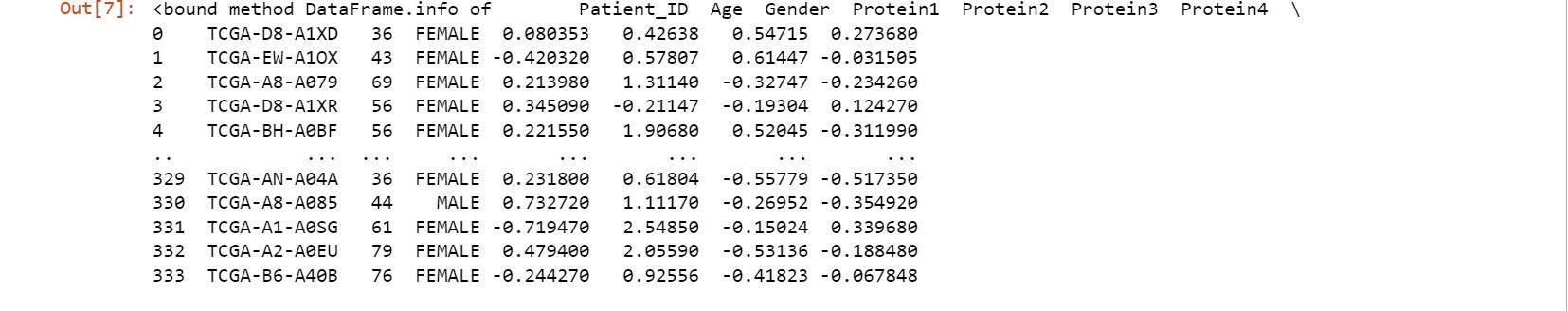
Printing the 1st 5 columns



data.tail()

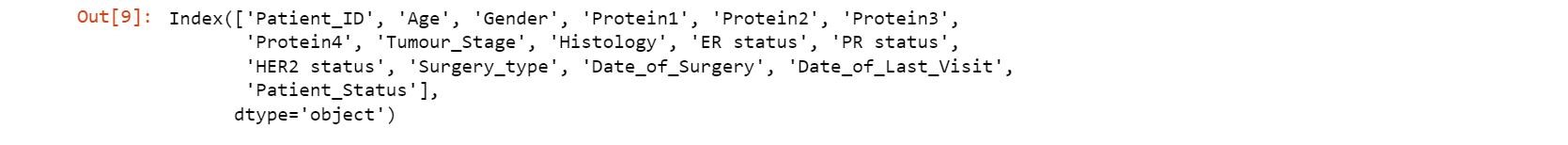
# Printing the dimenions of data

data.info()

#Printing information of data

data.columns

#Viewing the column heading



data.Patient\_Status.value\_counts()

#Inspecting target variable



1. The dataset includes information about 321 patients.
2. There are 8 features. Even though all of them are numeric type, only 5 are really numerical, the rest are categorical.

categorical features : Tumour\_stage, Histology, Surgery\_type, Pateint\_Status

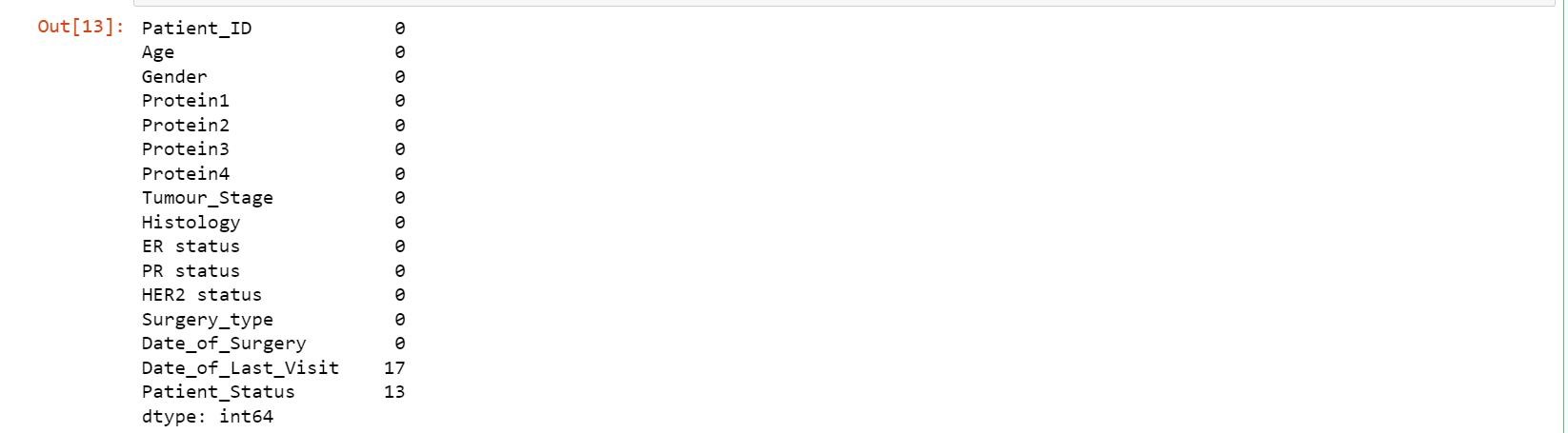
numerical features : Age, Protein1, Protein2, Protein3, Protein4

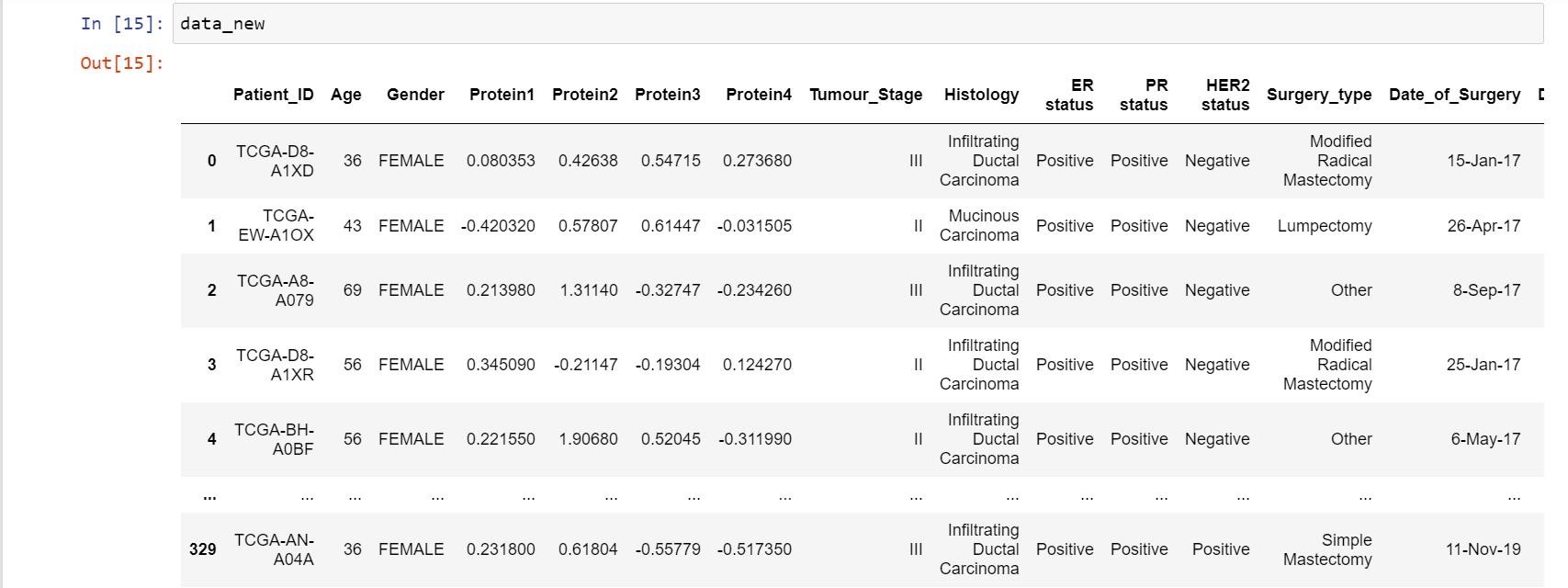
1. The target is output. 0 is Alive, 1 is Dead.

4)There are no missing values.

## Data Cleaning

data.isnull().sum()

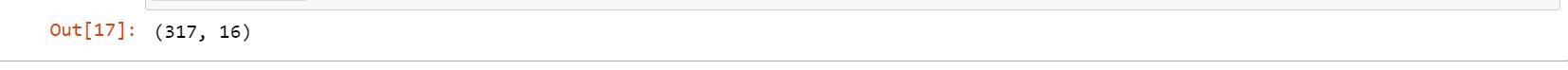
# Checking if any NULL values are present in the dataset data\_new=data.dropna()

#Removing NULL values

data\_new.isnull().sum()

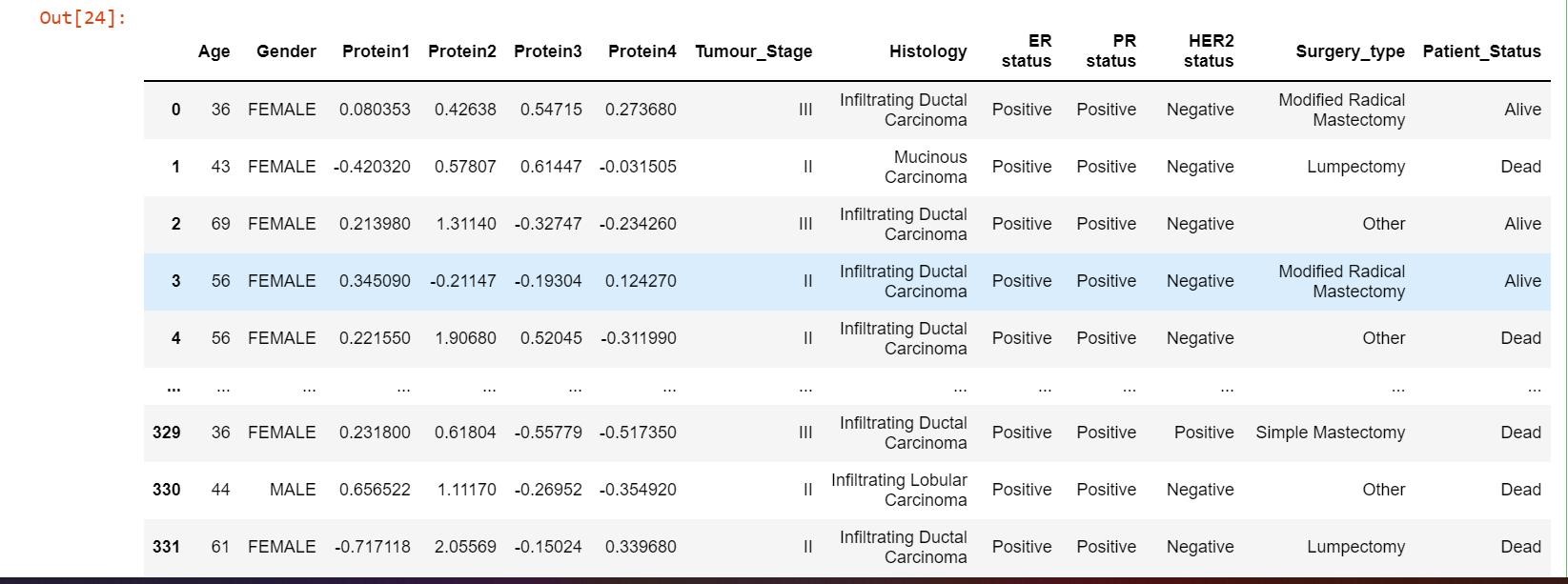
#Checking whether the NULL values are removed.

data\_new.shape

#Printing new shape of dataset

data.drop(columns[ "Patient\_ID","Date\_of\_Surgery","Date\_of\_Last\_Visi t"], inplace = True)

#Removing unrelevent and unwanted columns



data[data.duplicated()]

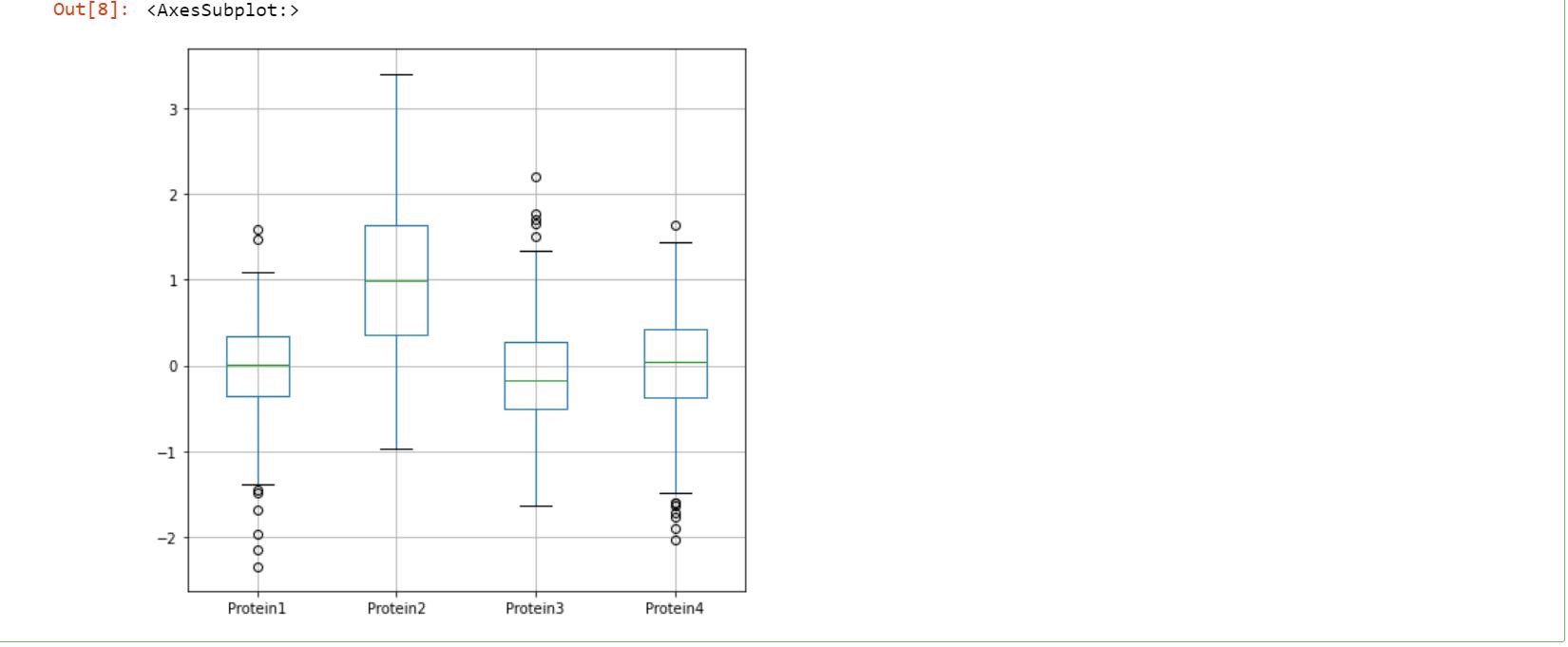
#Checking whether any duplicate values is present



No duplicate data is present in the dataset.

### #OUTLIERS

plt.figure(figsize=(7,7))

data.boxplot(column=[ 'Protein1', 'Protein2', 'Protein3', 'Protein4']) # Checking for the presence of outliers in numeric cols

# Outliers are present.So we need to get the exact values of outliers using IQR method.

# Finding values of outliers(IQR method)

def detect\_outliers\_iqr(data): outlier\_list = []

data = sorted(data)

q1 = np.percentile(data, 25)

q3 = np.percentile(data, 75)

#print("The Val of Q1 and Q2",q1, q3)

IQR = q3-q1 lwr\_bound = q1-(1.5\*IQR) upr\_bound = q3+(1.5\*IQR)

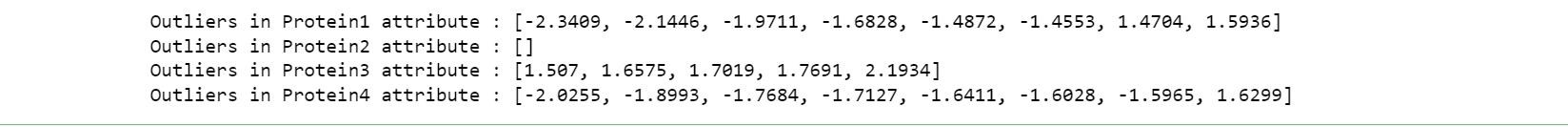
#print("The lower & Upper Bound",lwr\_bound, upr\_bound) for i in data:

if (i<lwr\_bound or i>upr\_bound): outlier\_list.append(i)

return outlier\_list # Driver code

for i in ['Protein1', 'Protein2', 'Protein3', 'Protein4']:

outliers = detect\_outliers\_iqr(data[i]) print("Outliers in",i,"attribute :", outliers)



# No Outliers in Protein2

# Outliers in Protein1 attribute : [-2.3409, -2.1446, -1.9711, -1.6828, -1.4872, -1.4553, 1.4704, 1.5936]

# Outliers in Protein3 attribute : [1.507, 1.6575, 1.7019, 1.7691, 2.1934]

# Outliers in Protein4 attribute : [-2.0255, -1.8993, -1.7684, -1.7127, -1.6411, -1.6028, -1.5965, 1.6299]

#Now we need to handle these outlier using Quantile based flooring and capping method

def handle\_outliers(data):

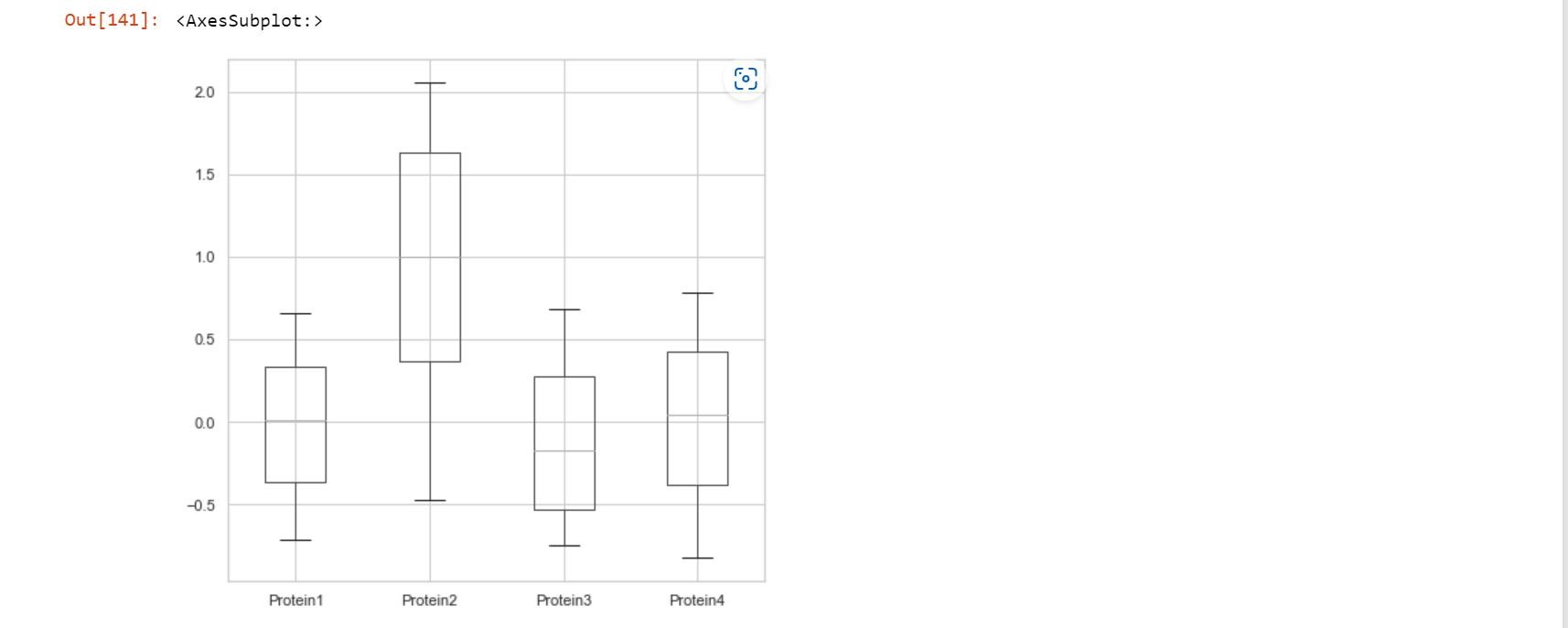
tenth\_percentile = np.percentile(data, 10) ninetieth\_percentile = np.percentile(data, 90) b = np.where(data<tenth\_percentile, tenth\_percentile, data) b1 = np.where(b>ninetieth\_percentile, ninetieth\_percentile, b) return b1

for i in ['Protein1', 'Protein2', 'Protein3', 'Protein4']:

data[i]=handle\_outliers(data[i])

# verifying again with boxplot

plt.figure(figsize=(7,7)) data.boxplot(column=[ 'Protein1', 'Protein2', 'Protein3','Protein4'])



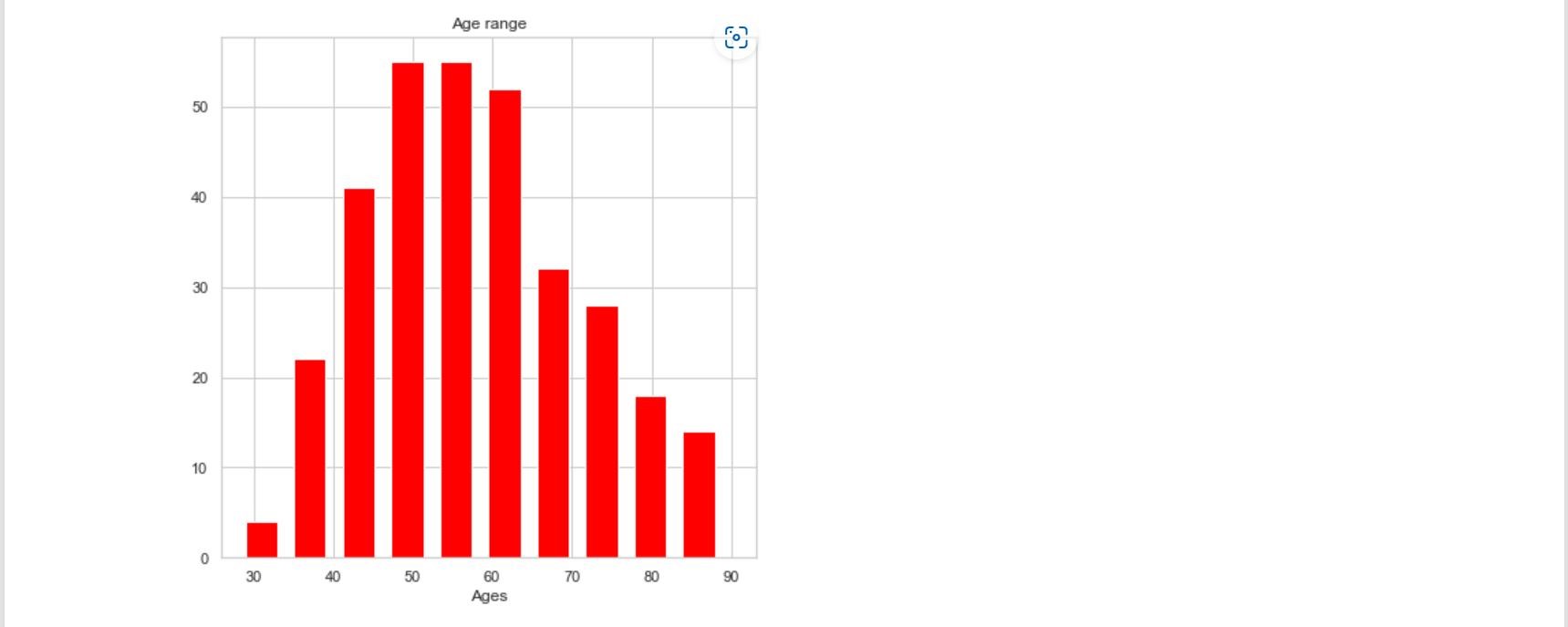
### Data Visualization

In this section, we will get an idea about our patients by considering all the features separately.

#Univariate Analysis

plt.figure(figsize=(12,9))

plt.hist(data['Age'].astype(int),width=4, align='mid',color='red') plt.xlabel('Ages') plt.title('Age range') plt.show()



# Analyzing the target variable

sns.set(style='whitegrid')

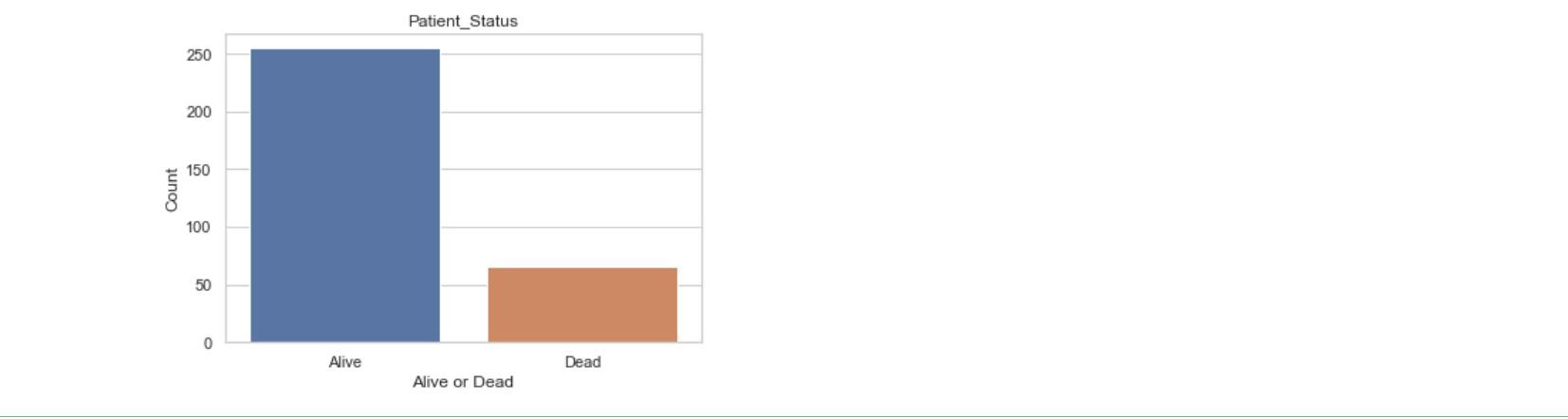
plt.title('Patient\_Status')

sns.countplot(data['Patient\_Status'])

plt.xlabel('Alive or Dead')

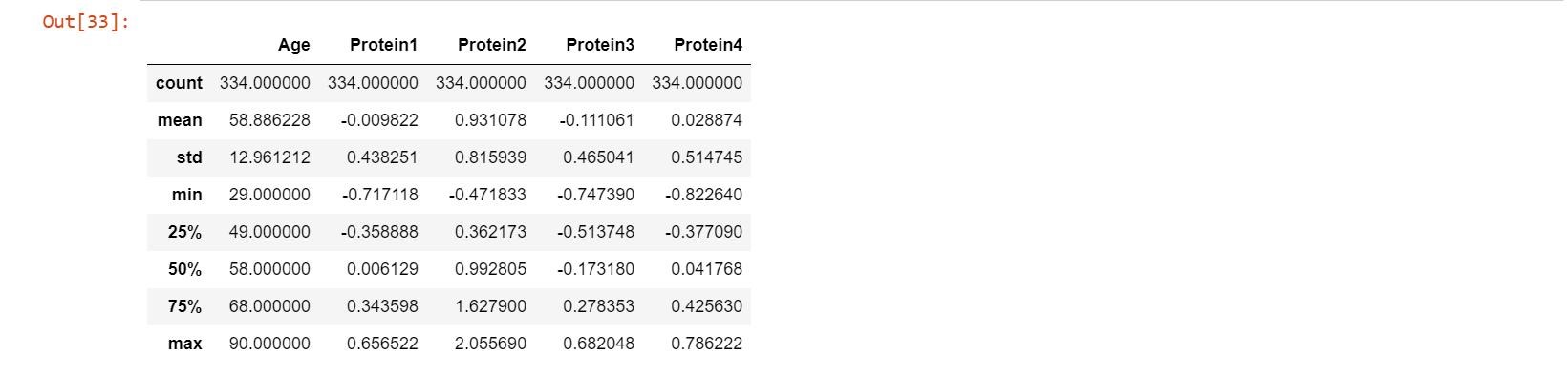
plt.ylabel('Count')

plt.show()



#For statistical description

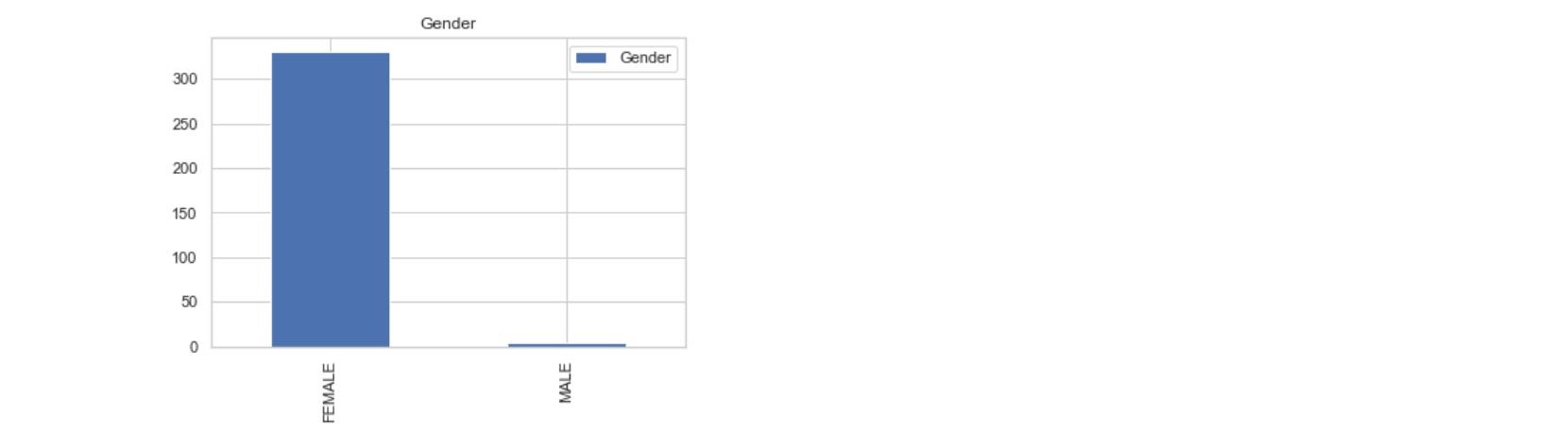
data.describe()



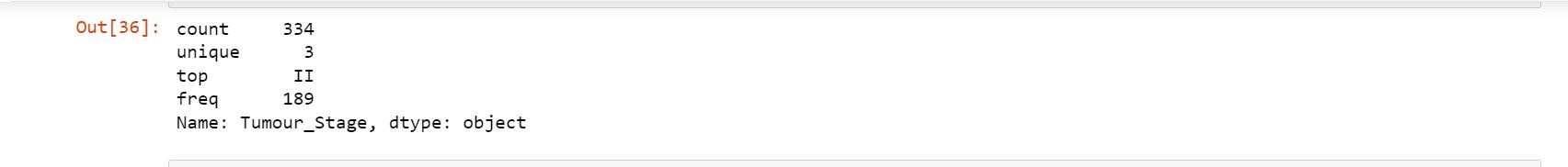
data.Gender.describe()



data.Gender.value\_counts().plot(kind='bar',title='Gender',legend=True,) plt.show()



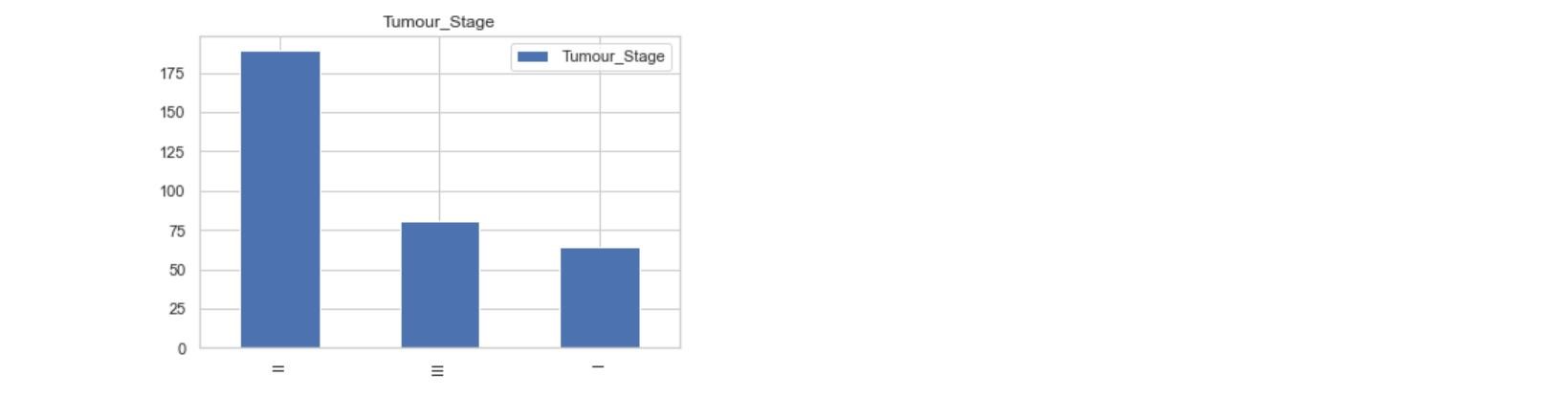
data["Tumour\_Stage"].describe()



data["Tumour\_Stage"].value\_counts()



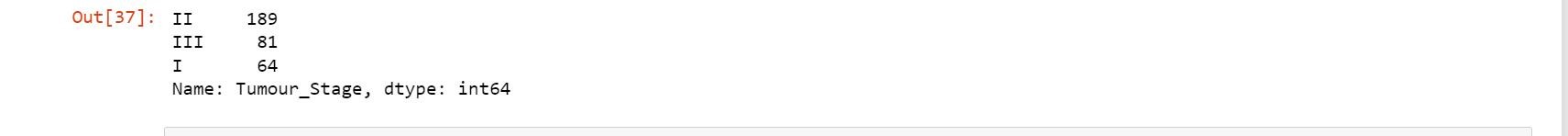
data.Tumour\_Stage.value\_counts().plot(kind='bar',title='Tumour\_Stage',l egend=True,) plt.show()



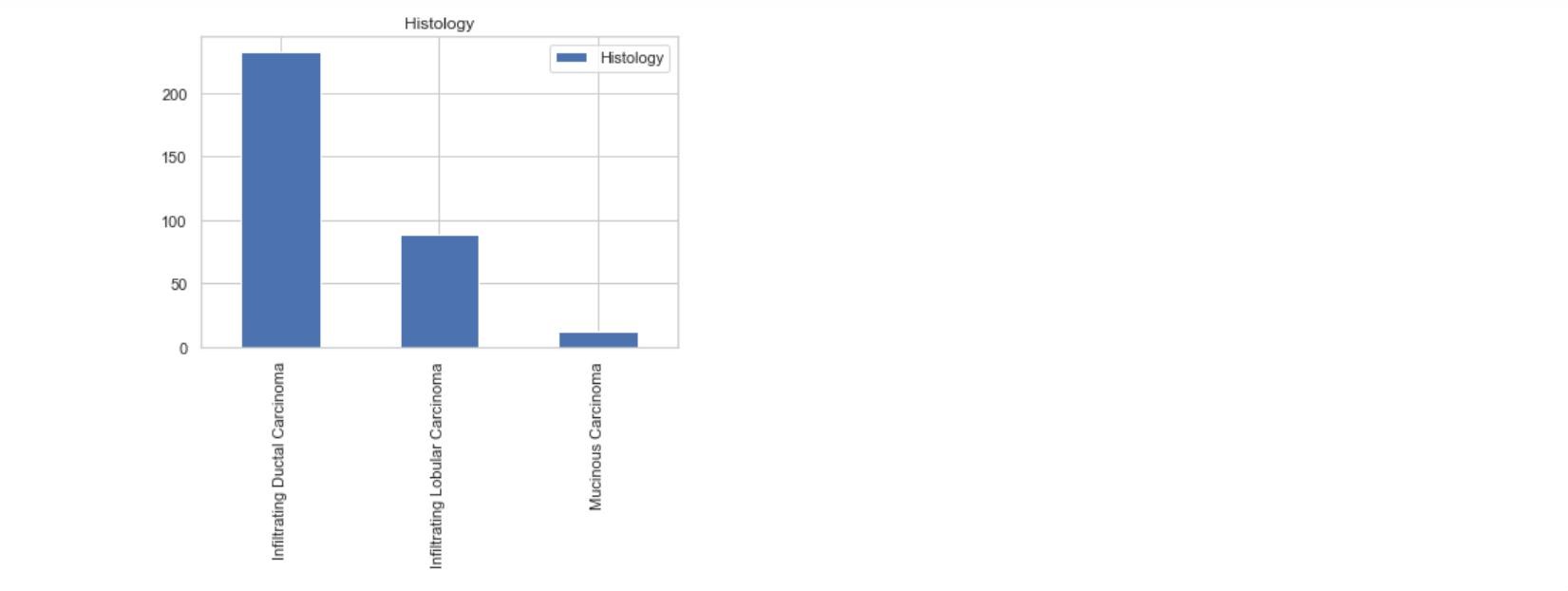
data["Histology"].describe()



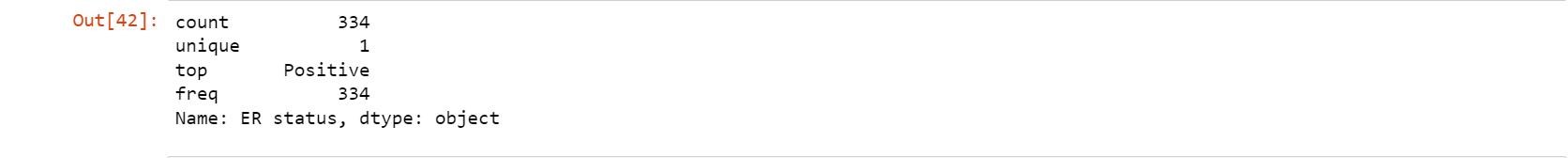
data["Histology"].value\_counts()



data.Histology.value\_counts().plot(kind='bar',title='Histology',legend=Tr ue,) plt.show()



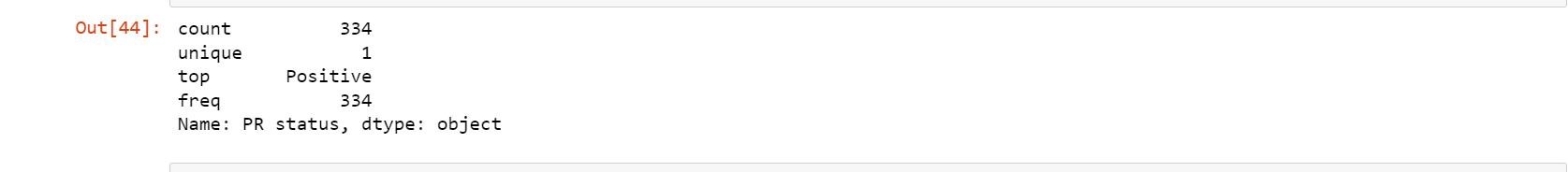
data['ER status'].describe()



data["ER status"].value\_counts()



data['PR status'].describe()



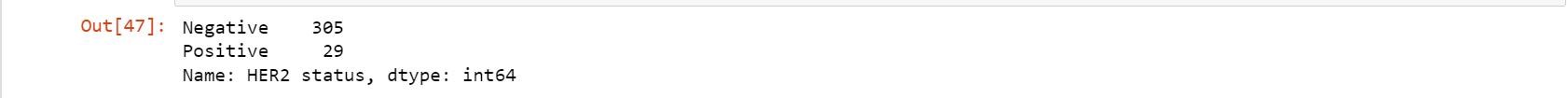
data["PR status"].value\_counts()



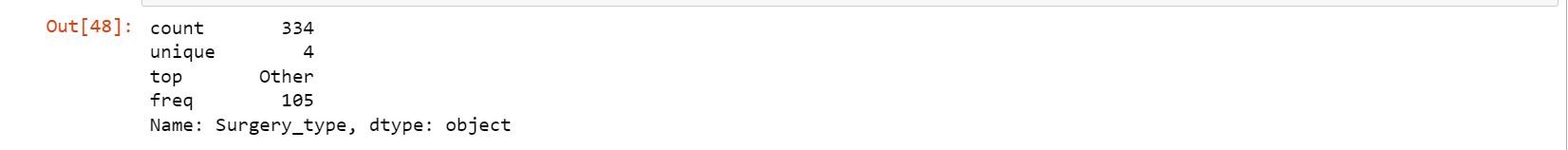
data['HER2 status'].describe()



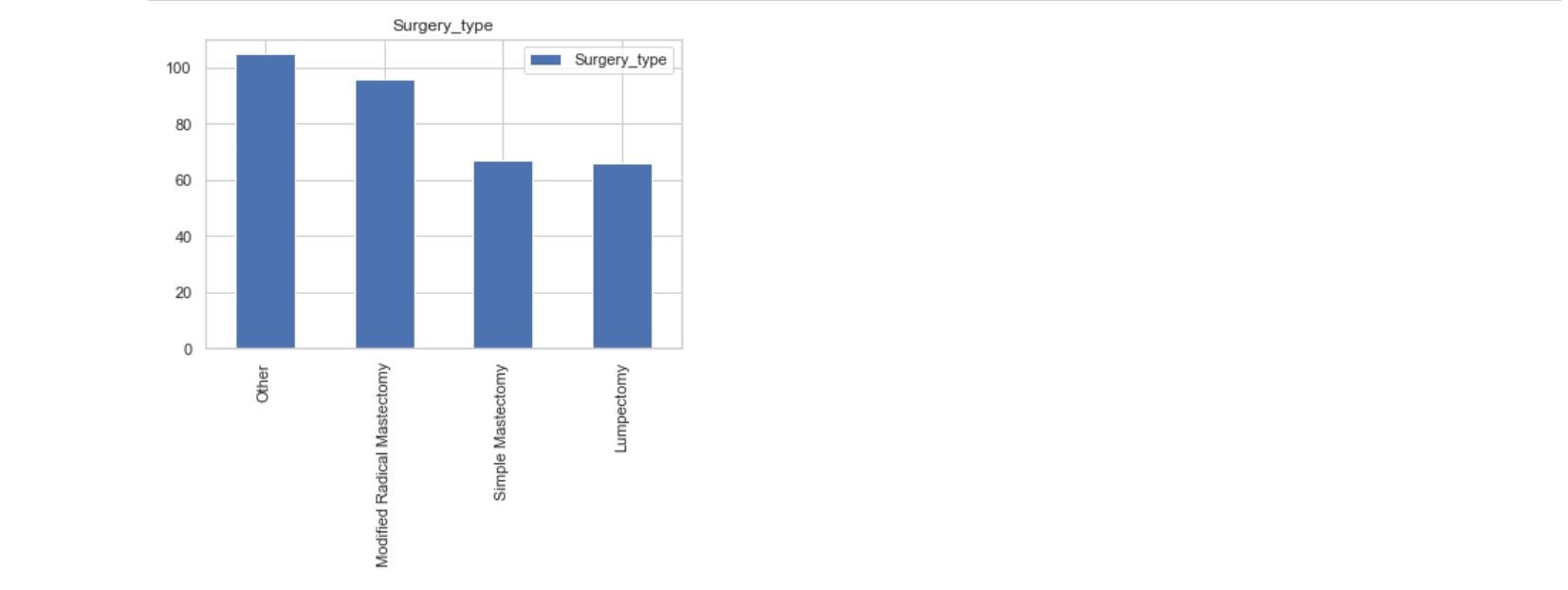
data['HER2 status'].value\_counts()



data['Surgery\_type'].describe()



data.Surgery\_type.value\_counts().plot(kind='bar',title='Surgery\_type',leg end=True,) plt.show()



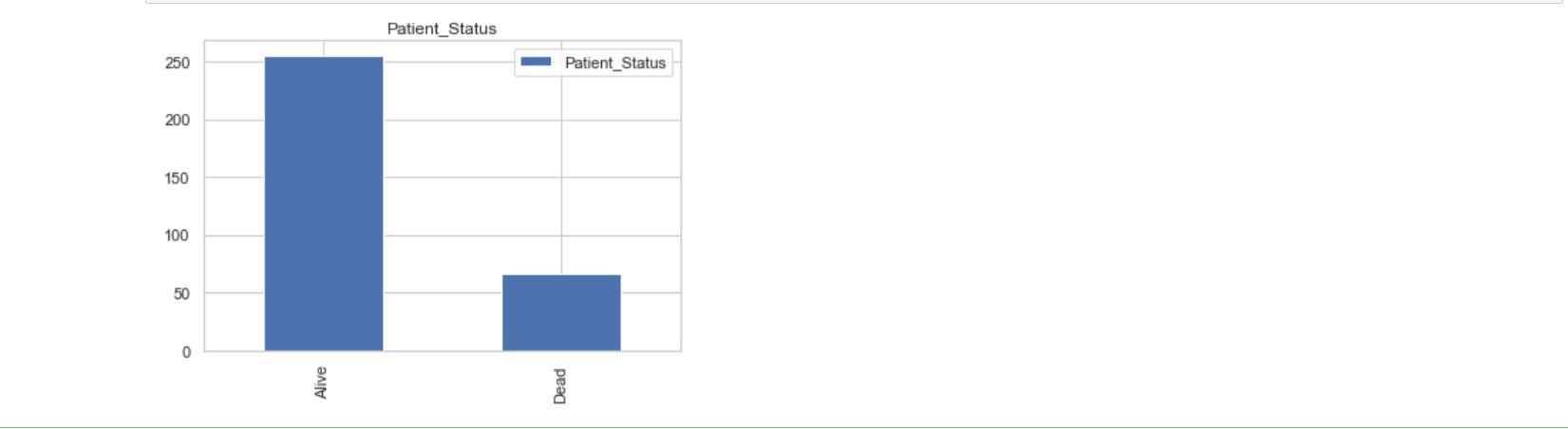
data['Patient\_Status'].describe()



data['Patient\_Status'].value\_counts()



data.Patient\_Status.value\_counts().plot(kind='bar',title='Patient\_Status',le gend=True,) plt.show()



#Bivariate Analysis

#Relationship of Gender and Tumour\_Stage

table=pd.crosstab(data['Tumour\_Stage'],data['Gender']) table

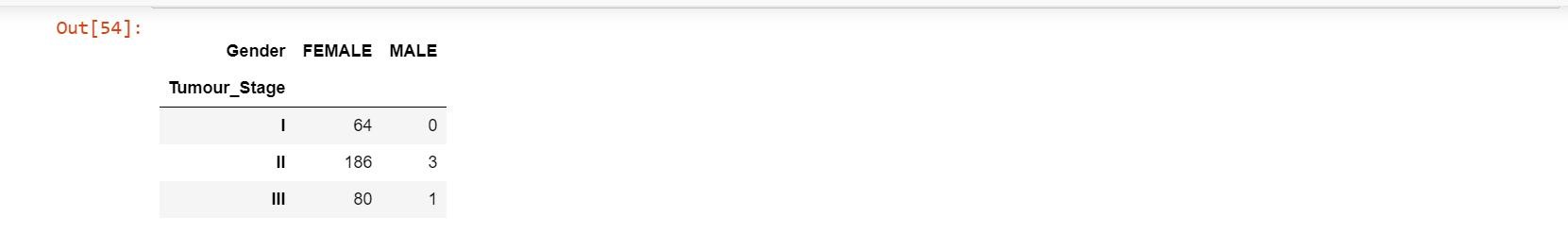
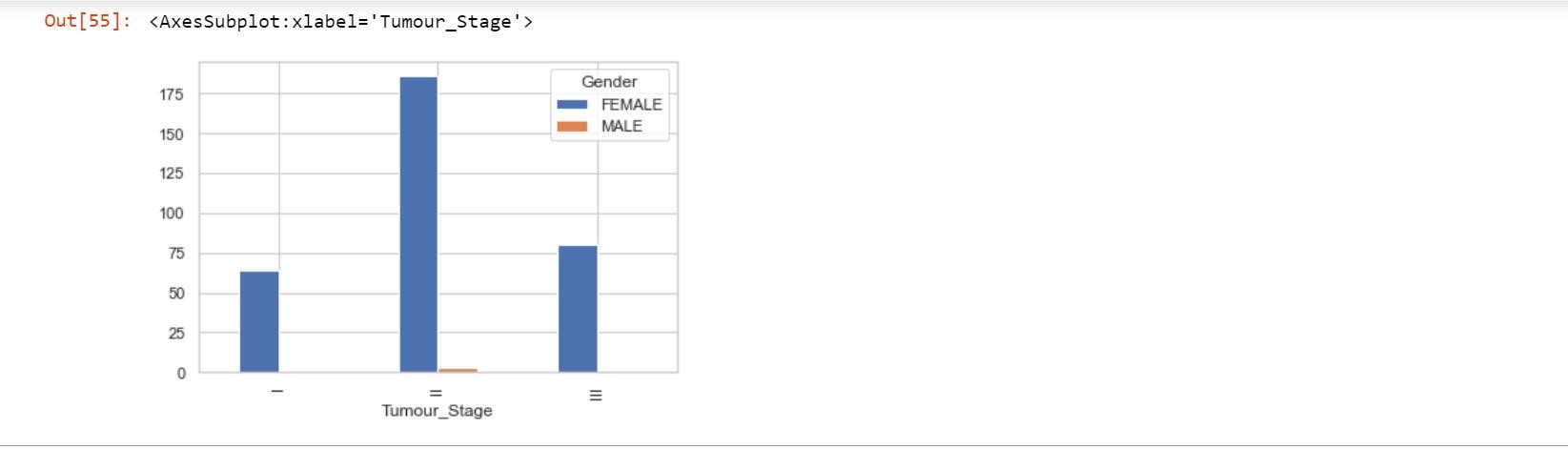


table.plot(kind='bar')



#Relationship of gender and Histology

table1=pd.crosstab(data['Histology'],data['Gender']) table1

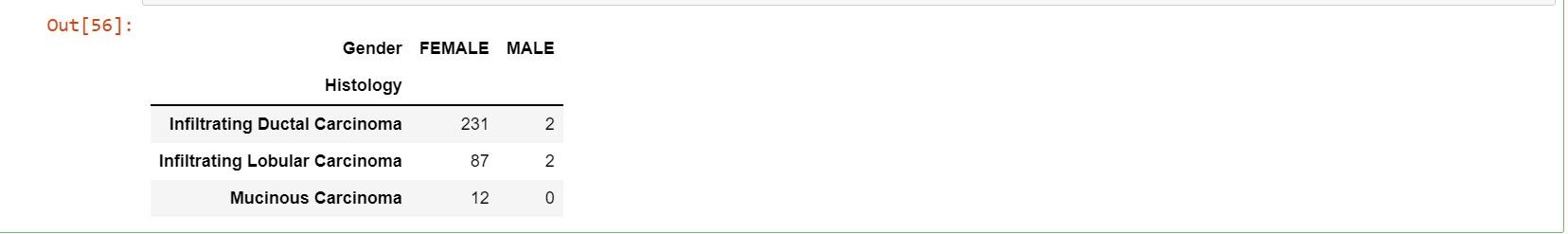
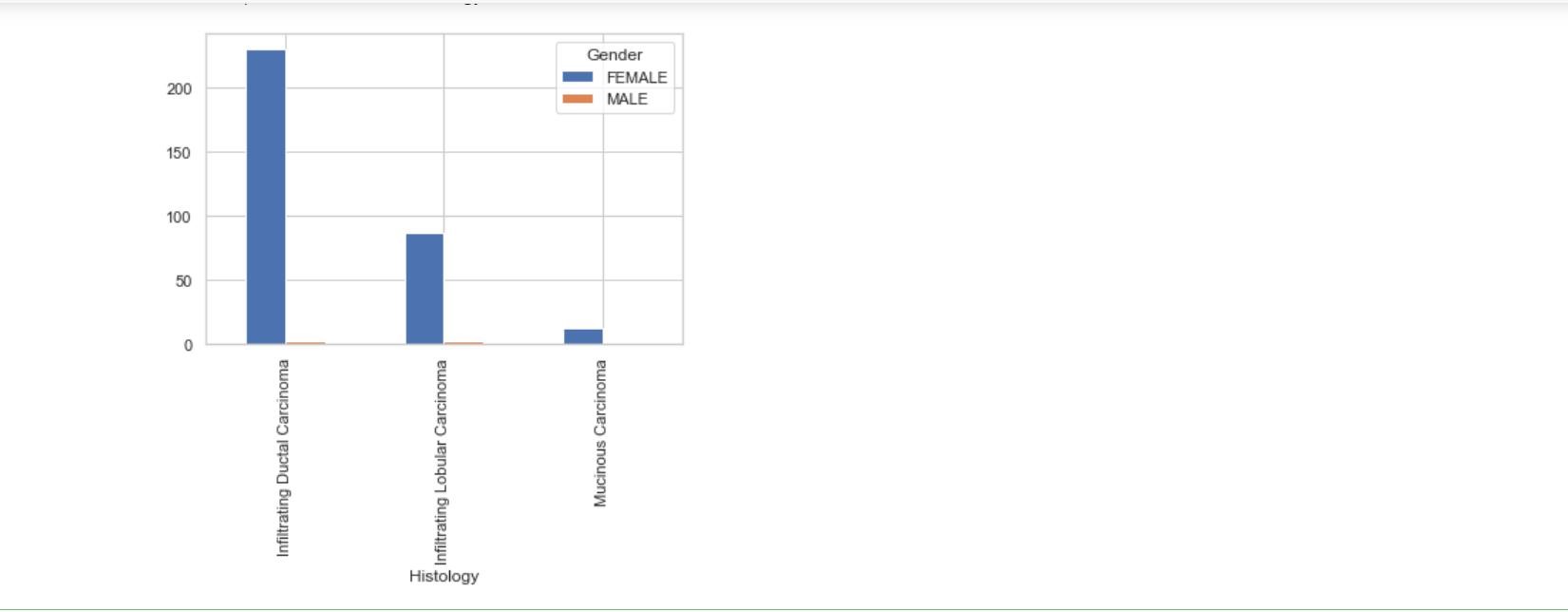


table1.plot(kind='bar')

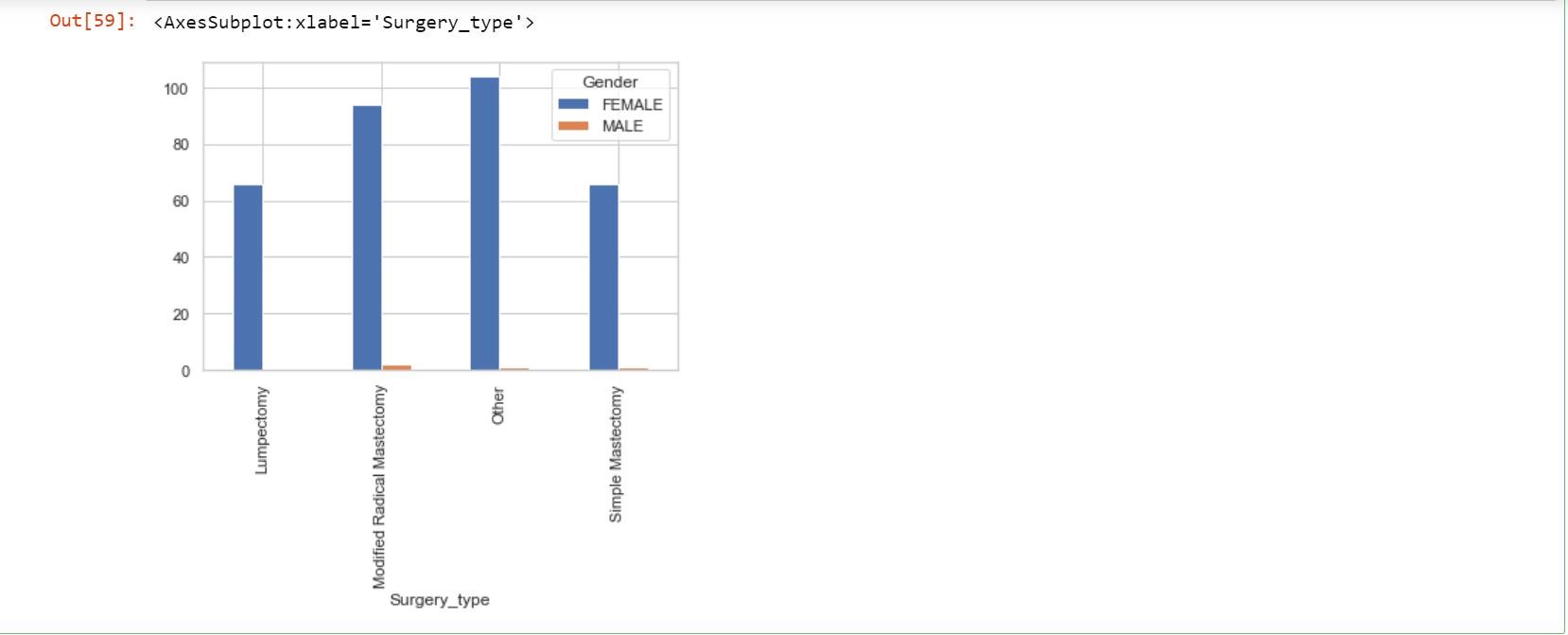


#Relationship of Gender and Surgery\_type

table2=pd.crosstab(data['Surgery\_type'],data['Gender']) table2



table2.plot(kind='bar')



#Relationship of Gender and Patient\_Status

table3=pd.crosstab(data['Patient\_Status'],data['Gender']) table3

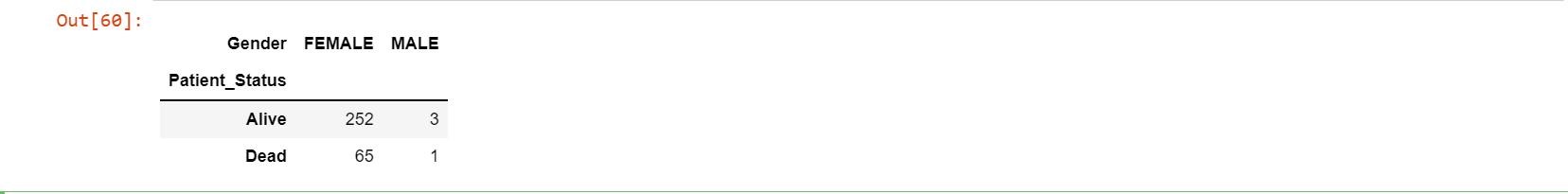
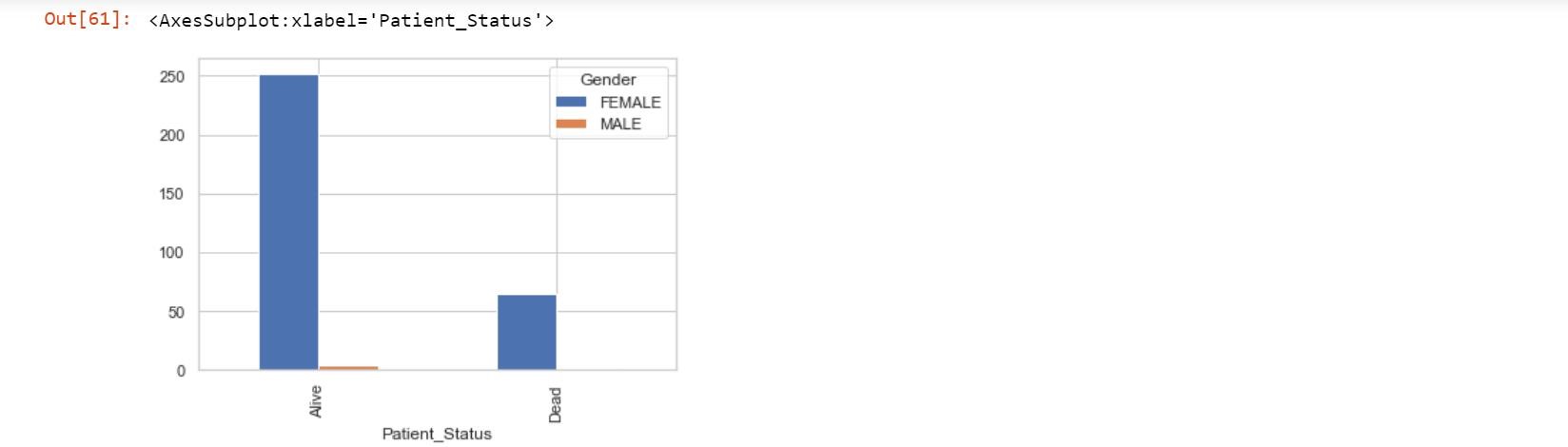


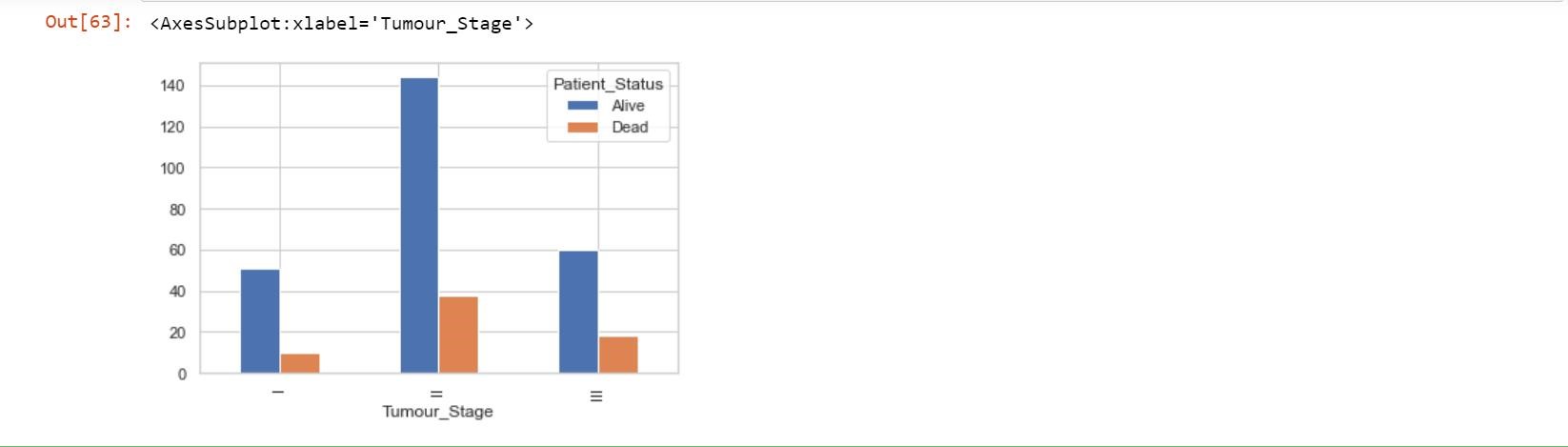
table3.plot(kind='bar')



#Relationship of Tumour\_Stage and Patient\_Status

table4=pd.crosstab(data['Tumour\_Stage'],data['Patient\_Status']) table4



table4.plot(kind='bar')

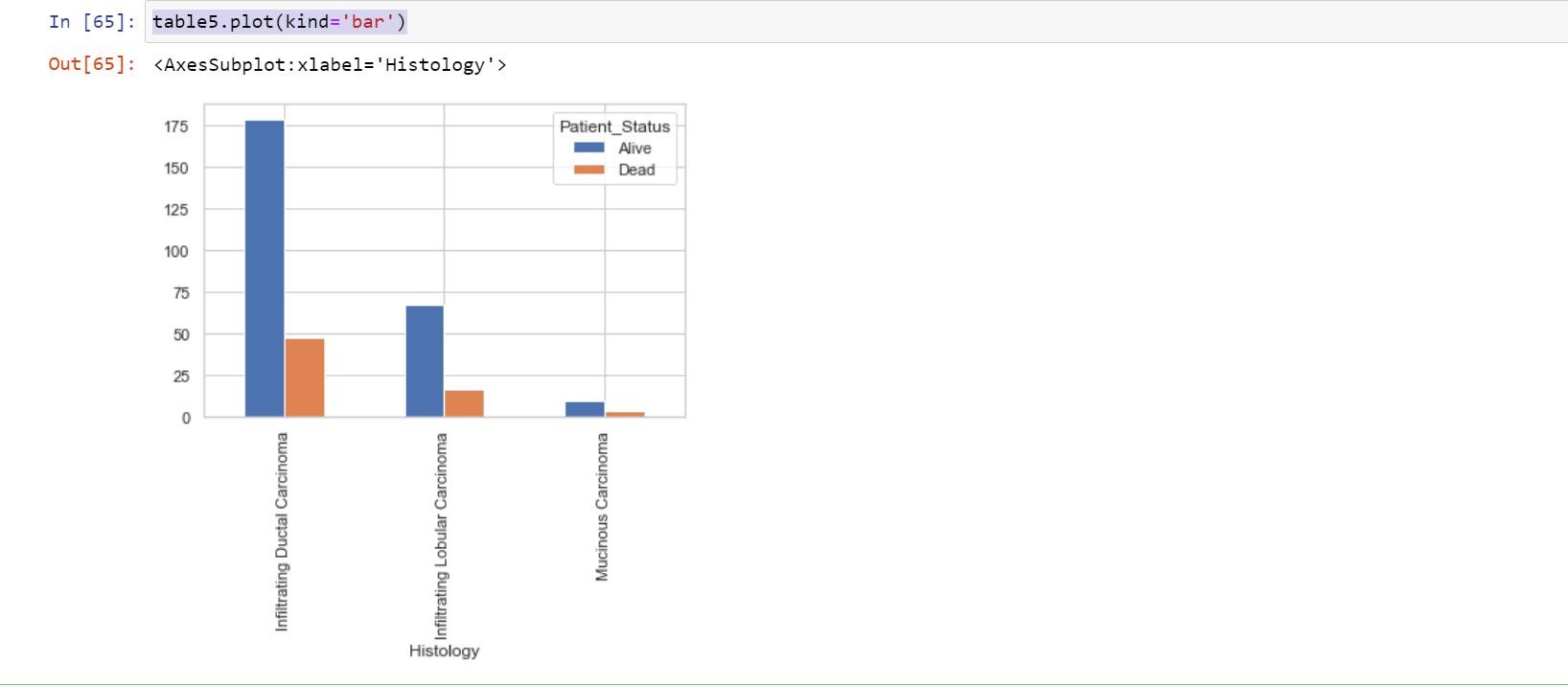
#Relationship of Histology and Patient\_Status

table5=pd.crosstab(data['Histology'],data['Patient\_Status'])

table5



table5.plot(kind='bar')



#Relationship of Surgery\_type and Patient\_Status

table6=pd.crosstab(data['Surgery\_type'],data['Patient\_Status']) table6

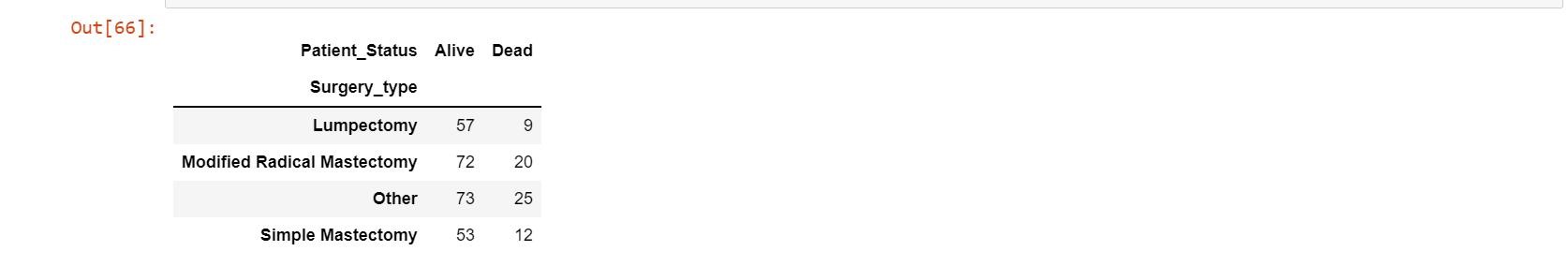
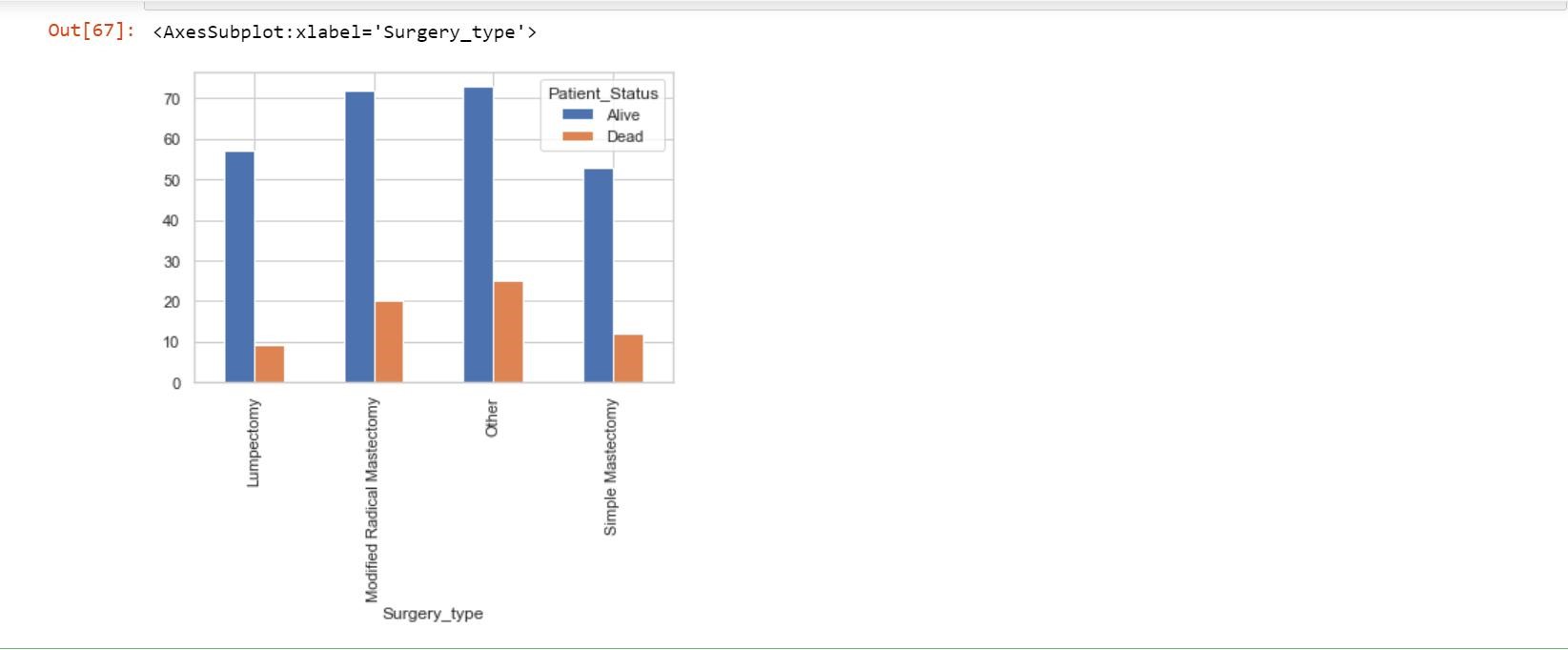


table6.plot(kind='bar')

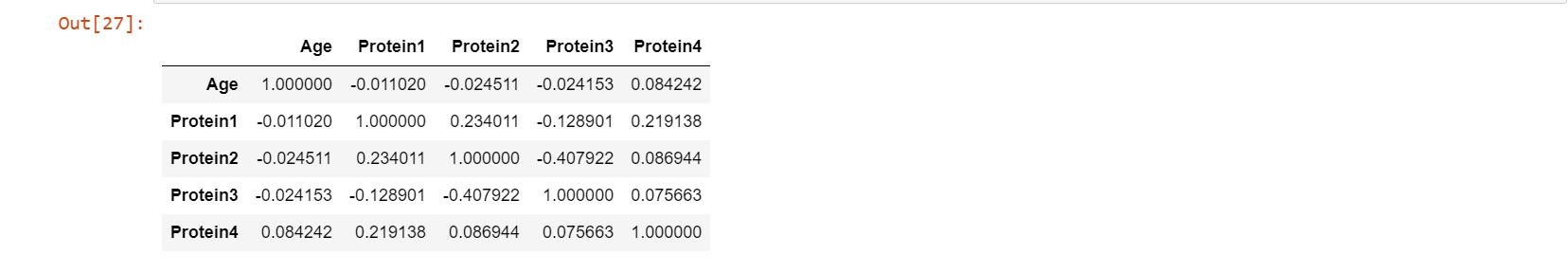


#Multivariate Analysis

# Finding out the correlation between the features

corr = data.corr()

Corr



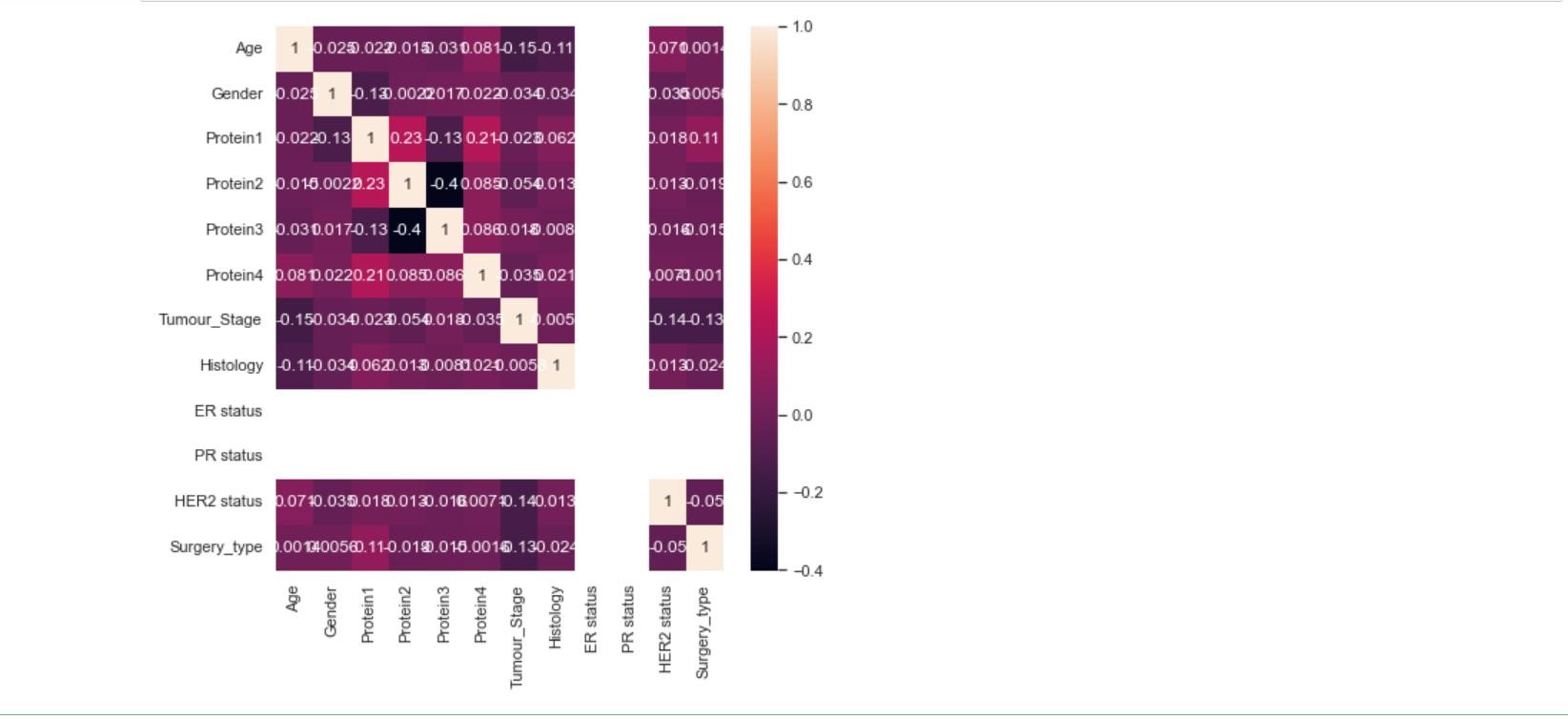
corr.shape



# Plotting the heatmap of correlation between features

plt.figure(figsize=(7,7))

dataplot=sns.heatmap(data.corr(),annot=True) plt.show()

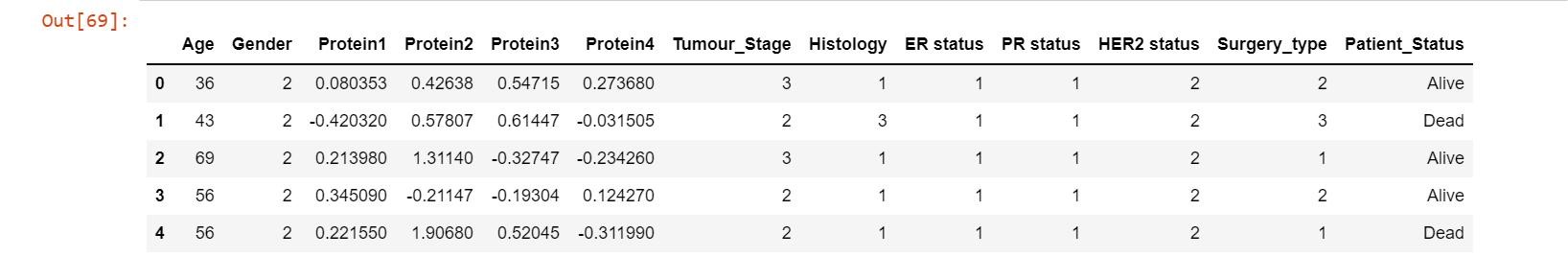


#### Data Preparation

data['Tumour\_Stage']=data['Tumour\_Stage'].map({'I':1,'II':2,'III':3}) data['Histology']=data['Histology'].map({'Infiltrating Ductal

Carcinoma':1,'Infiltrating Lobular Carcinoma':2,'Mucinous Carcinoma':3,}) data['ER status']=data['ER status'].map({'Positive':1}) data['PR status']=data['PR status'].map({'Positive':1})

data['HER2 status']=data['HER2 status'].map({'Positive':1,'Negative':2}) data['Gender']=data['Gender'].map({'MALE':1,'FEMALE':2}) data['Surgery\_type']=data['Surgery\_type'].map({'Other':1,'Modified Radical Mastectomy':2,'Lumpectomy':3,'Simple Mastectomy':4}) data.head()



**Algorithms**

**#Before applying algorithms in the dataset first we have to split the dataset into two parts i.e train\_set and test\_set.**

from sklearn.model\_selection import train\_test\_split

# Import train\_test\_split function

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size = 0.3, random\_state = 0)

print("Size of training set:", X\_train.shape) print("Size of training set:", X\_test.shape)



#### Classification Model

#### # Logistic Regression

# Import library for LogisticRegression

from sklearn.linear\_model import LogisticRegression

# Create a Logistic regression classifier logreg = LogisticRegression()

# Train the model using the training sets logreg.fit(X\_train, y\_train)



# Prediction on test data

y\_pred1 = logreg.predict(X\_test)

#Calculating Confusion Matrix

c\_matrix= confusion\_matrix(y\_test, y\_pred1)

print('Confusion matrix:\n',c\_matrix)

# Calculating the accuracy

acc\_logreg = round(metrics.accuracy\_score(y\_test, y\_pred1) \* 100, 2 )

print( 'Accuracy of Logistic Regression model : ', acc\_logreg )

#Calculating Precision, Recall and F1 Score

print(classification\_report(y\_test,y\_pred1))

**Output :-**

Confusion matrix:

[[53 0]

[11 0]]

Accuracy of Logistic Regression model : 82.81

precision recall f1-score support

Alive 0.84 0.98 0.90 53

Dead 0.50 0.09 0.15 11

accuracy 0.83 64

macro avg 0.67 0.54 0.53 64

weighted avg 0.78 0.83 0.78 64

#### # Gaussian Naive Bayes

# Import library of Gaussian Naive Bayes model from sklearn.naive\_bayes import GaussianNB

# Create a Gaussian Classifier model = GaussianNB()

# Train the model using the training sets model.fit(X\_train,y\_train)



# Prediction on test set

y\_pred2 = model.predict(X\_test)

#Calculating Confusion Matrix

c\_matrix= confusion\_matrix(y\_test, y\_pred2)

print('Confusion matrix:\n',c\_matrix)

# Calculating the accuracy

acc\_nb = round( metrics.accuracy\_score(y\_test, y\_pred2) \* 100, 2 )

print( 'Accuracy of Gaussian Naive Bayes model : ', acc\_nb )

#Calculating Precision, Recall and F1 Score

print(classification\_report(y\_test,y\_pred2))

**Output :-**

Confusion matrix:

[[53 0]

[11 0]]

Accuracy of Gaussian Naive Bayes model : 82.81

precision recall f1-score support

Alive 0.83 1.00 0.91 53

Dead 0.00 0.00 0.00 11

accuracy 0.83 64

macro avg 0.41 0.50 0.45 64

weighted avg 0.69 0.83 0.75 64

**# Decision Tree Classifier**

# Import Decision tree classifier

from sklearn.tree import DecisionTreeClassifier # Create a Decision tree classifier model

clf = DecisionTreeClassifier()

# Hyperparameter Optimization

parameters = {'max\_features': ['log2', 'sqrt','auto'],

'criterion': ['entropy', 'gini'],

'max\_depth': [2, 3, 5, 10, 50],

'min\_samples\_split': [2, 3, 50, 100],

'min\_samples\_leaf': [1, 5, 8, 10]

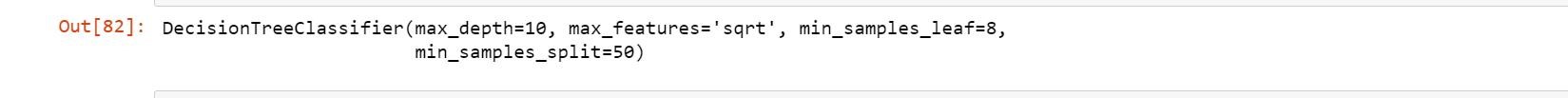
} # Run the grid search grid\_obj = GridSearchCV(clf, parameters)

grid\_obj = grid\_obj.fit(X\_train, y\_train)

# Set the clf to the best combination of parameters

clf = grid\_obj.best\_estimator\_

# Train the model using the training sets clf.fit(X\_train, y\_train)



# Prediction on test set y\_pred3 = clf.predict(X\_test)

#Calculating Confusion Matrix

c\_matrix= confusion\_matrix(y\_test, y\_pred3)

print('Confusion matrix:\n',c\_matrix)

# Calculating the accuracy

acc\_dt = round( metrics.accuracy\_score(y\_test, y\_pred3) \* 100, 2 ) print( 'Accuracy of Decision Tree model : ', acc\_dt )

#Calculating Precision, Recall and F1 Score

print(classification\_report(y\_test,y\_pred3))

**Output :-**

Confusion matrix:

[[52 1]

[11 0]]

Accuracy of Decision Tree model : 81.25

precision recall f1-score support

Alive 0.83 0.98 0.90 53

Dead 0.00 0.00 0.00 11

accuracy 0.81 64

macro avg 0.41 0.49 0.45 64

weighted avg 0.68 0.81 0.74 64

#### # Random Forest Classifier

# Import library of RandomForestClassifier model from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest Classifier

rf = RandomForestClassifier()

# Hyperparameter Optimization

parameters = {'n\_estimators': [4, 6, 9, 10, 15],

'max\_features': ['log2', 'sqrt','auto'],

'criterion': ['entropy', 'gini'],

'max\_depth': [2, 3, 5, 10],

'min\_samples\_split': [2, 3, 5],

'min\_samples\_leaf': [1, 5, 8]

}

# Run the grid search

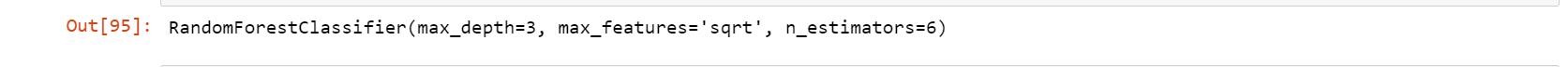
grid\_obj = GridSearchCV(rf, parameters) grid\_obj = grid\_obj.fit(X\_train, y\_train)

# Set the rf to the best combination of parameters

rf = grid\_obj.best\_estimator\_

# Train the model using the training sets

rf.fit(X\_train,y\_train)



# Prediction on test data y\_pred4 = rf.predict(X\_test)

#Calculating Confusion Matrix

c\_matrix= confusion\_matrix(y\_test, y\_pred4)

print('Confusion matrix:\n',c\_matrix)

# Calculating the accuracy

acc\_rf = round( metrics.accuracy\_score(y\_test, y\_pred4) \* 100 , 2 ) print( 'Accuracy of Random Forest model : ', acc\_rf )

#Calculating Precision, Recall, F1 Score

print(classification\_report(y\_test,y\_pred4))

**Output :-**

Confusion matrix:

[[53 0]

[11 0]]

Accuracy of Random Forest model : 82.81

precision recall f1-score support

Alive 0.83 1.00 0.91 53

Dead 0.00 0.00 0.00 11

accuracy 0.83 64

macro avg 0.41 0.50 0.45 64

weighted avg 0.69 0.83 0.75 64

#### # SVM Classifier

# Creating scaled set to be used in model to improve the results from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Import Library of Support Vector Machine model from sklearn import svm

# Create a Support Vector Classifier

svc = svm.SVC()

# Hyperparameter Optimization parameters = [

{'C': [1, 10, 100, 1000], 'kernel': ['linear']},

{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},

]

# Run the grid search

grid\_obj = GridSearchCV(svc, parameters) grid\_obj = grid\_obj.fit(X\_train, y\_train)

# Set the svc to the best combination of parameters

svc = grid\_obj.best\_estimator\_

# Train the model using the training sets svc.fit(X\_train,y\_train)



# Prediction on test data y\_pred5 = svc.predict(X\_test)

#Calculating Confusion Matrix

c\_matrix= confusion\_matrix(y\_test,y\_pred5)

print('Confusion matrix:\n',c\_matrix)

# Calculating the accuracy

acc\_svm = round( metrics.accuracy\_score(y\_test, y\_pred5) \* 100, 2 ) print( 'Accuracy of SVM model : ', acc\_svm )

#Calculating Precision, Recall, F1 Score

print(classification\_report(y\_test,y\_pred5))

**Output :-**

Confusion matrix:

[[53 0]

[11 0]]

Accuracy of Support Vector Machine model : 82.81

precision recall f1-score support

Alive 0.83 1.00 0.91 53

Dead 0.00 0.00 0.00 11

accuracy 0.83 64

macro avg 0.41 0.50 0.45 64

weighted avg 0.69 0.83 0.75 64

#### # K - Nearest Neighbors

# Import library of KNeighborsClassifier model from sklearn.neighbors import KNeighborsClassifier

# Create a KNN Classifier

knn = KNeighborsClassifier()

# Hyperparameter Optimization parameters = {'n\_neighbors': [3, 4, 5, 10],

'weights': ['uniform', 'distance'],

'algorithm' : ['auto', 'ball\_tree', 'kd\_tree', 'brute'],

'leaf\_size' : [10, 20, 30, 50]

}

# Run the grid search

grid\_obj = GridSearchCV(knn, parameters) grid\_obj = grid\_obj.fit(X\_train, y\_train)

# Set the knn to the best combination of parameters

knn = grid\_obj.best\_estimator\_

# Train the model using the training sets

knn.fit(X\_train,y\_train)



# Prediction on test data y\_pred6 = knn.predict(X\_test)

#Calculating Confusion Matrix

c\_matrix= confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix:\n',c\_matrix)

# Calculating the accuracy

acc\_knn = round( metrics.accuracy\_score(y\_test, y\_pred6) \* 100, 2 ) print( 'Accuracy of KNN model : ', acc\_knn )

#Calculating Precision, Recall, F1 Score

print(classification\_report(y\_test,y\_p))

**Output :-**

Confusion matrix:

[[53 0]

[11 0]]

Accuracy of KNN model : 82.81

precision recall f1-score support

Alive 0.83 1.00 0.91 53

Dead 0.00 0.00 0.00 11

accuracy 0.83 64

macro avg 0.41 0.50 0.45 64

weighted avg 0.69 0.83 0.75 64

#### Model Evalution

model = pd.DataFrame({'Model':['Logistic Regression', 'Naive Bayes', 'Decision Tree', 'Random Forest', 'Support Vector Machines',

'K - Nearest Neighbors'],

'Accuracy ':[acc\_logreg,acc\_nb,acc\_rf,acc\_knn,acc\_dt,acc\_svm]}) model\_ev.sort\_values(by='Accuracy ',ignore\_index=True,inplace=True) model\_ev

Model Accuracy

0 Logistic Regression 82.81

1 Naive Bayes 82.81

3 Random Forest 82.81

4 Support Vector Machines 82.81

5 K - Nearest Neighbors 82.81

2 Decision Tree 81.25

From this we can infer that Logistic Regression, Naïve Bayes, Random Forest, Support Vector Machine and K – Nearest Neighbour have the same accuracy.

# Barplot for Accuracy of different models

m = model.sort\_values(by=Accuracy, ascending = True)

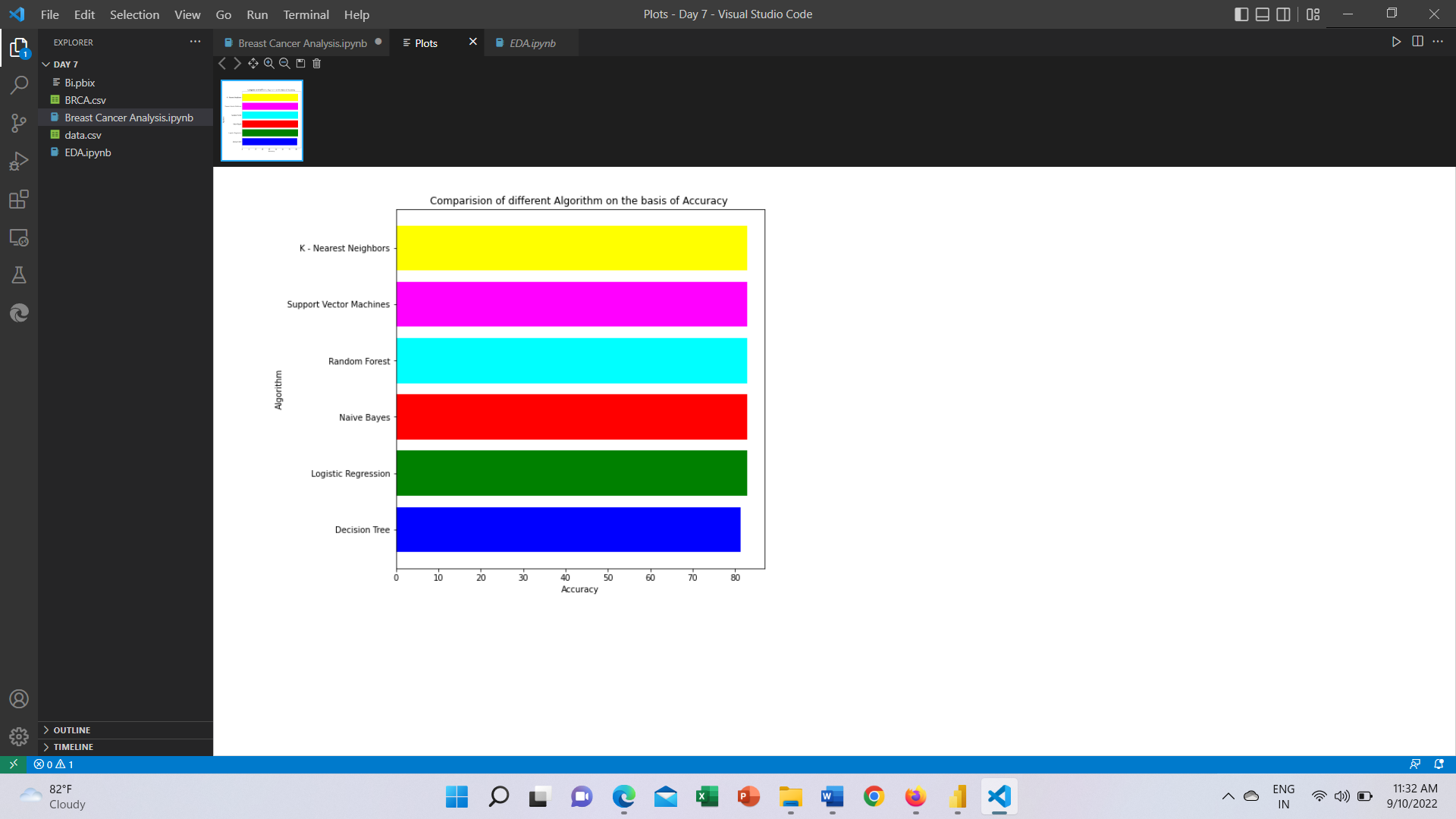
plt.figure(figsize=(8,8))

plt.title("Comparision Of different Algorithm on the basis of Accuracy") plt.xlabel("Accuracy ")

plt.ylabel("Algorithms")

plt.barh(m['Model'],m['Accuracy'],color=['blue','green','red','cyan’,’magenta’,’yellow’]

plt.show()



**Conclusion :-** Out of six machine learning algorithms we applied, we conclude that all algorithms except Decision Tree Classifier gives same accuracy i.e 82.81%.

#### Dashboard

