Importing Packages

In [1]:
```
import numpy as np
import pandas as pd
import math
from statistics import mean
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

Dataset Exploration

Loading in Dataset

In [2]:
```
clinical_df = pd.read_csv("clinical.csv")
genomic_df = pd.read_csv('genomics.csv')
```

Visualizations (Clinical Dataset)

In [3]:
```
clinical_df.head()
```

Out[3]:

	ID	Outcome	Survival.Months	Age	Grade	Num.Primaries	T	N	M	Radiation	Sta
0	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
1	2	Dead	19.0	73	2	0	UNK	2.0	0.0	5	
2	3	Dead	13.0	72	3	0	2	2.0	0.0	0	
3	4	Dead	15.0	69	9	1	1a	0.0	1.0	0	
4	5	Dead	10.0	76	9	0	UNK	NaN	NaN	0	

In [4]:
```
clinical_df['Survival.Months'].describe()
```

Out[4]:
```
count    190.000000
mean      22.186842
std       12.420140
min        9.000000
25%       11.000000
50%       16.000000
75%       34.000000
max       71.000000
Name: Survival.Months, dtype: float64
```

In [5]:
```
clinical_df['Age'].describe()
```

Out[5]:
```
count    190.000000
mean      70.173684
std        6.146909
min       56.000000
25%       67.000000
50%       71.000000
75%       74.000000
max       84.000000
Name: Age, dtype: float64
```

From first glance, the "Radiation" feature stands out and it would be of interest to know how this affects the survival rate, in part because we can't establish immediately if the correlation between a patient's "Outcome" is causational or correlational. Since the goal of this analysis is for use following diagnosis, whether or not a patient has or will need radiation treatment may not be not at the time of diagnosis.

In [6]:
```
# determining type for each feature
clinical_df.dtypes
```

Out[6]:
```
ID                    int64
Outcome              object
Survival.Months     float64
Age                   int64
Grade                 int64
Num.Primaries         int64
T                    object
N                   float64
M                   float64
Radiation             int64
Stage                object
Primary.Site         object
Histology            object
Tumor.Size          float64
Num.Mutated.Genes     int64
Num.Mutations         int64
dtype: object
```

In [7]:
```
clinical_df.iloc[0,1]
```

Out[7]:
```
'Alive'
```

We can do a quick check to see if there any non-{0,5} values in the "Radiation" feature.

In [8]:
```
rad_vals = list(clinical_df['Radiation'].unique())
print(f"The present data in the radiation feature is in the list {rad_
```

```
The present data in the radiation feature is in the list [0, 5]
```

In [9]:
```
# patients that had radiation treatment
yes_radiation = clinical_df[clinical_df['Radiation'] == 5]
surv_w_radiation = yes_radiation[yes_radiation['Outcome'] == 'Alive']

# patients that didn't have radiation treatment
no_radiation = clinical_df[clinical_df["Radiation"] == 0]
surv_wo_radiation = no_radiation[no_radiation['Outcome'] == 'Alive']


# survival rates with radiation/no-radiation patients
rate_radiation = "{:.2f}".format(len(surv_w_radiation) / len(yes_radia
rate_no_radiation = "{:.2f}".format(len(surv_wo_radiation) / len(no_ra

print(f"The survival rate of patients with radiation treatment: {rate_
print(f"The survival rate of patients without radiation treatment: {ra
```

```
The survival rate of patients with radiation treatment: 0.27
The survival rate of patients without radiation treatment: 0.18
```

We can one-hot encode the radiation feature during preprocessing.

Another question to consider is how the followup time affects a patients outcome. For example, we might expect to see a few potential trends here:

- a patient is more likely to die the longer the followup takes place as a means of time naturally passing
- a patient is more likely to die the longer the followup takes place as other parametrs related to the cancer itself has had time to increase, leading to more serious case
- a patient is more likely to die the shorter the followup takes place as shorter followup times may mean that a patients case is more serious and requires attention sooner

Due to these factors, it's good to establish a basis for what to expect with having a wide range of followup times, especially considering that we might want to control the followup time during an analysis.

Another reason for establishing a better understanding of the effects of different followup times is that we might not be able to simply remove all data not equal or near our desired **control time (12 months)**. The data removed may prove more valuable than not.

In [10]:
```
# limiting our data to just the columns of interest (speed up computat
# but not a necessary thing to do extensively with these datasets)
followup_df = clinical_df[["Outcome", "Survival.Months"]]

num_living = len(followup_df[followup_df['Outcome'] == "Alive"])
num_dead = len(followup_df[followup_df['Outcome'] == "Dead"])

print(f"Number of patients living at followup: {num_living}")
print(f"Number of patients dead at followup: {num_dead}")
print(f"Total number of patients in dataset: {len(followup_df)}")
```

```
Number of patients living at followup: 40
Number of patients dead at followup: 150
Total number of patients in dataset: 190
```

In [11]:
```
target_months = clinical_df[(clinical_df['Survival.Months'] >= 10) & (
target_pats = len(target_months)
tot_pats = len(clinical_df)
per_target = "{:.2f}".format((target_pats / tot_pats) * 100)

print(f"Number of patients that followed up within two months of a yea
print(f"Percentage of the dataset made up of patients in target month
```

```
Number of patients that followed up within two months of a year: 71
Percentage of the dataset made up of patients in target month range:
37.37%
```

In [12]:
```
clinical_df['Survival.Months'].value_counts()
```

Out[12]:
```
11.0    27
10.0    23
13.0    21
36.0    18
32.0    11
38.0     9
33.0     8
16.0     8
9.0      7
22.0     7
15.0     7
35.0     6
23.0     6
19.0     6
34.0     4
29.0     3
9.5      3
42.0     3
71.0     2
18.0     2
39.0     2
40.0     1
37.0     1
41.0     1
46.0     1
50.0     1
26.0     1
24.0     1
Name: Survival.Months, dtype: int64
```

Since there aren't actually any patients that followed up at exactly 12 months, this gives me more incentive to predict the outcomes of patients around a target set of months. In order to do so, we can sample half of the target range of months in the dataset for use in testing, which would give us a train/test split for the **entire** dataset that is almost roughly 80/20. Using this method, we can still use the rest of the data for building a model, as well as have some data points from our target pool for use in training.

Visualizations (Genomics Dataset and Gene-related Features)

In [13]:
```
genomic_df.head()
```

Out[13]:

	ID	Gene
0	1	AKT1
1	158	AKT1
2	88	ALK_Col1
3	132	ALK_Col1
4	18	ALK_Col2

In [14]:
```
genomic_df['Gene'].unique()
```

Out[14]:
```
array(['AKT1', 'ALK_Col1', 'ALK_Col2', 'APC', 'ATM_Col1', 'ATM_Col2',
       'BRAF', 'CCND2', 'CDKN2A', 'CTNNB1', 'DNMT3A', 'EGFR', 'ERBB3'
,
       'ERBB4', 'ESR1', 'FBXW7', 'FGFR1', 'FGFR3', 'FLT4', 'FOXL2',
       'GNAS', 'HNF1A', 'KRAS_Col1', 'KRAS_Col2', 'MAP2K2', 'MET',
       'MLH_Col2', 'MSH2', 'MSH6', 'NF_Col1', 'NF_Col2', 'NF_Col3',
       'NF_Col5', 'NOTCH1', 'NTRK1', 'PDGFRB', 'PIK3CA', 'PIK3CB',
       'POLD_Col2', 'PTCH1', 'PTEN', 'RB1', 'SMARCA4', 'SMARCB1', 'SM
O',
       'STK11', 'TERT', 'TP53_Col1', 'TP53_Col2', 'TSC2'], dtype=obje
ct)
```

Given more data, it would be interesting to see the features of the clinical dataset that relate to gene mutatations as separate data, but we might suffer from issues of sparsity here if we **join** the two datasets.

However, if there is still a trend in number of mutated genes and outcome then it might be worth it to keep the mutation columns as independent. If there is no trend, then this means that the type of gene might play a larger role in the outcome than anticipated.

In [15]:
```
num_mutations_lst = sorted(list(clinical_df['Num.Mutated.Genes'].uniqu

num_mutations_lst
```

Out[15]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [16]:
```
for num in num_mutations_lst:
    num_muts = clinical_df[clinical_df['Num.Mutated.Genes'] == num]
    num_living = len(num_muts[num_muts['Outcome'] == 'Alive'])
    num_patients = len(num_muts)
    rate = "{:.2f}".format(num_living / num_patients)

    print(f"The survival rate of patients with {num} mutated genes is:
    print(f"\tThe total number of patients in this pool is: {num_patie
    print()
```

```
The survival rate of patients with 0 mutated genes is: 0.00
        The total number of patients in this pool is: 6

The survival rate of patients with 1 mutated genes is: 0.21
        The total number of patients in this pool is: 33

The survival rate of patients with 2 mutated genes is: 0.21
        The total number of patients in this pool is: 53

The survival rate of patients with 3 mutated genes is: 0.16
        The total number of patients in this pool is: 55

The survival rate of patients with 4 mutated genes is: 0.29
        The total number of patients in this pool is: 21

The survival rate of patients with 5 mutated genes is: 0.27
        The total number of patients in this pool is: 15

The survival rate of patients with 6 mutated genes is: 0.25
        The total number of patients in this pool is: 4

The survival rate of patients with 7 mutated genes is: 0.00
        The total number of patients in this pool is: 1

The survival rate of patients with 8 mutated genes is: 1.00
        The total number of patients in this pool is: 2
```

What we can deduce here is that there is a sufficient lack of data at the ends of this range of number of gene mutations. Since there doesn't seem to be any real correlation amongst the number of genes ($1 \leq \text{num\_mutated\_genes} \leq 6$) and the survival rate then we can either drop the column in pre-processing or remove the outliers since the model could very well *learn* that all patients with these specific number of genes will either definitely live or definitely die even if this is too small a sample size to make such a decision.

Instead, what we could do is create a dataframe where each gene is it's own column and we can flag whether a patient has that gene mutation or not.

We can consider a separate dataset where each patient's data is expanded such that we drop the 'Num.Mutated.Genes' and 'Num.Mutations' columns and have a row for each gene present.

In [17]:
```
gene_patient_df = pd.merge(clinical_df, genomic_df)

gene_patient_df.head()
```

Out[17]:

	ID	Outcome	Survival.Months	Age	Grade	Num.Primaries	T	N	M	Radiation	Stag
0	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
1	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
2	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
3	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
4	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	

In [18]:
```
gene_lst = gene_patient_df['Gene'].unique()
num_pats_lst = []
rate_lst = []

for gene in gene_lst:
    gene_df = gene_patient_df[gene_patient_df['Gene'] == gene]
    num_living = len(gene_df[gene_df['Outcome'] == 'Alive'])
    num_patients = len(gene_df)
    rate = "{:.2f}".format(num_living / num_patients)

    num_pats_lst.append(num_patients)
    rate_lst.append(rate)


    print(f"The survival rate of patients with the mutated gene {gene}
    print(f"\tThe total number of patients in this pool is: {num_patie
```

```
The survival rate of patients with the mutated gene AKT1 is: 1.00
        The total number of patients in this pool is: 2

The survival rate of patients with the mutated gene CCND2 is: 1.00
        The total number of patients in this pool is: 2

The survival rate of patients with the mutated gene EGFR is: 1.00
        The total number of patients in this pool is: 6

The survival rate of patients with the mutated gene FGFR3 is: 1.00
        The total number of patients in this pool is: 2

The survival rate of patients with the mutated gene KRAS_Col1 is: 0.2
4
        The total number of patients in this pool is: 55

The survival rate of patients with the mutated gene PDGFRB is: 1.00
        The total number of patients in this pool is: 5

The survival rate of patients with the mutated gene STK11 is: 0.17
```

In [19]:
```
gene_mut_survival_dict = {"Gene":gene_lst, "Number of Patients":num_pa
gene_mut_survival_df = pd.DataFrame(gene_mut_survival_dict)

print(f"Number of genes present in dataframe: {len(gene_mut_survival_d
gene_mut_survival_df.head()
```

```
Number of genes present in dataframe: 50
```

Out[19]:

	Gene	Number of Patients	Survival Rate
0	AKT1	2	1.00
1	CCND2	2	1.00
2	EGFR	6	1.00
3	FGFR3	2	1.00
4	KRAS_Col1	55	0.24

We can clean this up to contain genes where the number of patients exceeds some arbitrary value such as 20.

In [20]:
```
clean_gene_survival_df = gene_mut_survival_df[gene_mut_survival_df["Nu

print(f"Number of significant genes present in dataframe: {len(clean_g
clean_gene_survival_df
```

```
Number of significant genes present in dataframe: 6
```

Out[20]:

	Gene	Number of Patients	Survival Rate
4	KRAS_Col1	55	0.24
6	STK11	23	0.17
7	TSC2	31	0.10
9	TP53_Col1	117	0.26
10	CDKN2A	45	0.07
14	MSH2	30	0.03

In order to see the difference in the 'Num.Mutated.Genes' and 'Num.Mutations' columns, we can look at a dataframe that only contains patients where these two columns aren't unique (which may mean we can drop one of these columns).

In [21]:
```
diff_num_mut = clinical_df[clinical_df['Num.Mutated.Genes'] != clinica

print(f"Number of patients where the number of mutated genes is differ
diff_num_mut.head()
```

```
Number of patients where the number of mutated genes is different fro
m the total number of mutations: 54
```

Out[21]:

	ID	Outcome	Survival.Months	Age	Grade	Num.Primaries	T	N	M	Radiation	Sta
5	6	Dead	11.0	62	9	0	3	2.0	NaN	0	IV
12	13	Dead	22.0	70	3	1	1a	NaN	NaN	5	
14	15	Dead	11.0	62	4	0	3	NaN	NaN	0	II
17	18	Alive	42.0	67	9	0	1b	NaN	0.0	0	
19	20	Dead	10.0	62	9	0	3	NaN	NaN	0	II

Now, since the number of patients that have more than one mutation for at least one mutated gene is a significant number of the total patients, we can check the survival rate of these patients and cross-reference with the survival rate of patients that have only one mutation per mutated gene.

In [22]:
```
# finding the survival rate of patients with the same number of mutati
same_num_mut = clinical_df[clinical_df['Num.Mutated.Genes'] == clinica
num_living_same = len(same_num_mut[same_num_mut['Outcome'] == 'Alive']
num_patients_same = len(same_num_mut)
rate_same = "{:.2f}".format(num_living_same / num_patients_same)

print(f"The survival rate of patients with the same number of mutation
print(f"\tThe total number of patients in this pool is: {num_patients_

# finding the survival rate of patients with different number of mutat
num_living_diff = len(diff_num_mut[diff_num_mut['Outcome'] == 'Alive']
num_patients_diff = len(diff_num_mut)
rate_diff = "{:.2f}".format(num_living_diff / num_patients_diff)

print(f"The survival rate of patients with the more mutations than mut
print(f"\tThe total number of patients in this pool is: {num_patients_

# finding total survival rate in clinical_df
tot_living = len(clinical_df[clinical_df['Outcome'] == 'Alive'])
tot_patients = len(clinical_df)
rate_all = "{:.2f}".format(tot_living / tot_patients)

print(f"The survival rate of all patients: {rate_all}")
```

```
The survival rate of patients with the same number of mutations as mu
tated genes: 0.19
        The total number of patients in this pool is: 136

The survival rate of patients with the more mutations than mutated ge
nes: 0.26
        The total number of patients in this pool is: 54

The survival rate of all patients: 0.21
```

We can speculate why having more mutated genes may lead to a higher survival rate, but this is perhaps best left to an oncologist. Instead it could be beneficial to drop the 'Num.Mutations' and 'Num.Mutated.Genes' columns since we've already established that we will drop the 'Num.Mutated.Genes' column in favor of adding columns for specific genes that seem statistically significant in affecting a patient's survival rate.

To add to this, we can also create a new column that flags whether or not the number of mutations is equal to the number of genes.

Final Explorations of the Features

Before continuing on to preprocessing, we can look at a few of the other features for any interesting insights that might aid us in expanding or removing parts of the data.

In [23]:
```
# calling our df back for ease of use
clinical_df.head()
```

Out[23]:

	ID	Outcome	Survival.Months	Age	Grade	Num.Primaries	T	N	M	Radiation	Sta
0	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
1	2	Dead	19.0	73	2	0	UNK	2.0	0.0	5	
2	3	Dead	13.0	72	3	0	2	2.0	0.0	0	
3	4	Dead	15.0	69	9	1	1a	0.0	1.0	0	
4	5	Dead	10.0	76	9	0	UNK	NaN	NaN	0	

Tumor Grade

In [24]:
```
sorted(list(clinical_df['Grade'].unique()))
```

Out[24]:
```
[2, 3, 4, 9]
```

In [25]:
```
clinical_df['Grade'].value_counts()
```

Out[25]:
```
9    96
4    43
2    29
3    22
Name: Grade, dtype: int64
```

The description of the data states that the tumor grade is either 1-4 or unspecified and a quick check on Cancer Research UK (https://www.cancerresearchuk.org/about-cancer/lung-cancer/stages-types-grades/stages-grades) confirmed this grading system. The value 9 may also be a typo and could actually be a 1, but since we cannot confirm this without seeing the original method of data collection, it might be best to drop this column.

Age

In [26]:
```
min_age = clinical_df['Age'].min()
max_age = clinical_df['Age'].max()
print(f"The minimum age of all patients is {min_age}")
print(f"The maximum age of all patients is {max_age}")
```

```
The minimum age of all patients is 56
The maximum age of all patients is 84
```

In [27]:
```
clinical_df['Age'].isnull().sum()
```

Out[27]:
```
0
```

Should I have more time with this analysis, it would be good to see a plot of the survival rate across different age ranges. For now, there are no outstanding outliers or null values here.

Number of Primary Tumors

In [28]:
```
# unique values in the number of primaries column
list(clinical_df['Num.Primaries'].unique())
```

Out[28]:
```
[0, 1]
```

Again, here would be a good chance to see if the number of primary tumors affects survival rate given more time, but since the values aren't too complex, we will assume it has little negative effect on the final model.

(T) Tumor Stage | (N) Num of Metastasis to Lymph Nodes | (M) Num of Distant Metastases

We can explore the TNM system (as explained here (https://www.cancer.org/treatment/understanding-your-diagnosis/staging.html) for someone with no prior experience with this terminology).

In [29]:
```
T_lst = sorted(list(clinical_df['T'].unique()))
T_lst
```

Out[29]:
```
['1', '1a', '1b', '2', '2a', '2b', '3', '4', 'UNK']
```

In [30]:
```
clinical_df['T'].value_counts()
```

Out[30]:
```
UNK    62
3      38
1a     26
4      23
2a     16
2      12
2b     10
1b      2
1       1
Name: T, dtype: int64
```

Since the actual values of the (T) tumor stage can be assumed to not only be relevant but also the size of the value can be assumed to have an impact on whether or not to classify a patient's outcome, we can make these values into integers according to their ordering (as long as they're ordered the value itself does not need to pertain to the value it currently holds since we will most likely scale these values so they're better interpreted by the final ML algorithm.

For the unknown values, we can assign them the mean of the dataset. For the values such as '1a' or '2b' we can cluster them with the integer value already associated with them.

A future analysis might see that we plot these tumor stage classifications against their survival rate and assign the average survival rate for each classification as the value that it takes.

In [31]:
```
N_list = sorted(list(clinical_df['N'].unique()))
N_list
```

Out[31]:
```
[0.0, 2.0, nan, 1.0, 3.0]
```

In [32]:
```
# checking the type for each N classification
for n in N_list:
    print(f"Type: {type(n)}")
```

```
Type: <class 'numpy.float64'>
Type: <class 'numpy.float64'>
Type: <class 'numpy.float64'>
Type: <class 'numpy.float64'>
Type: <class 'numpy.float64'>
```

In [33]:
```
clinical_df['N'].value_counts()
```

Out[33]:
```
2.0    58
0.0    52
1.0     9
3.0     6
Name: N, dtype: int64
```

We can perform the same operations on the 'N' column as the 'T' column by assigning the null values to be the mean of the non-null values.

This would be a great opportunity to consult a domain expert in order to better determine how to classify null values here. For example, should this domain expert inform us that there is a correlation between N and T and/or M, then we could use the classification of the other features to classify this feature. Naturally, the same can be said of the other two features. Lastly, should there be a correlation, we could build a simple model that can predict the value of the missing feature using the other two features rather than simply using methods to fill null values such as the mean or median.

In [34]:
```
M_list = sorted(list(clinical_df['M'].unique()))
M_list
```

Out[34]:
```
[nan, 0.0, 1.0]
```

In [35]:
```
clinical_df['M'].value_counts()
```

Out[35]:
```
0.0    86
1.0     8
Name: M, dtype: int64
```

In [36]:
```
M1_df = clinical_df[clinical_df['M'] == 1.0]
M1_rate = "{:.2f}".format(len(M1_df[M1_df['Outcome'] == 'Alive']) / le
print(f"Rate of survival when distant cancer spread is present: {M1_ra
M0_df = clinical_df[clinical_df['M'] == 0.0]
M0_rate = "{:.2f}".format(len(M0_df[M0_df['Outcome'] == 'Alive']) / le
print(f"Rate of survival when distant cancer spread is present: {M0_ra
```

```
Rate of survival when distant cancer spread is present: 0.25
Rate of survival when distant cancer spread is present: 0.23
```

Since the data is already relatively sparse, especially for the classification of the presence of distant cancer, there are more null values than not, and the survival rate difference between the two classes is relatively small we can go ahead and drop this column in preprocessing.

Stage | Primary Site | Histology | Tumor Size

In [37]:
```
cancer_stage_lst = list(clinical_df['Stage'].unique())
cancer_stage_lst
```

Out[37]:
```
['IV', 'IIIA', 'IA', 'IVB', 'IIA', 'IIIB', 'IIB', 'IB', '1B']
```

Similar to the analysis of the tumor stage, we can assign each value here according to the number already associated with it and for now ignore the letters attached to the stage during preprocessing.

In [38]:
```
prim_site_lst = list(clinical_df['Primary.Site'].unique())
prim_site_lst
```

Out[38]:
```
['Left Lower Lobe',
 'Right Upper Lobe',
 'Left Hilar',
 'Right Hilar',
 'Left Upper Lobe',
 'Right Lower Lobe',
 'Both Lung',
 'Right Middle Lobe',
 'Righ Upper Lobe']
```

In [39]:
```
clinical_df['Primary.Site'].value_counts()
```

Out[39]:
```
Right Upper Lobe     53
Right Hilar          33
Left Hilar           31
Right Lower Lobe     25
Left Upper Lobe      21
Left Lower Lobe      17
Both Lung             5
Right Middle Lobe     3
Righ Upper Lobe       2
Name: Primary.Site, dtype: int64
```

In [40]:
```
both_lung_df = clinical_df[clinical_df['Primary.Site'] == 'Both Lung']
both_lung_rate = "{:.2f}".format(len(both_lung_df[both_lung_df['Outcom
print(f"Rate of survival when cancer is present in both lungs: {both_l
print(f"\tNumber of patients with cancer in both lungs: {len(both_lung

one_lung_df = clinical_df[clinical_df['Primary.Site'] != 'Both Lung']
one_lung_rate = "{:.2f}".format(len(one_lung_df[one_lung_df['Outcome']
print(f"Rate of survival when cancer is present in only one lung: {one
print(f"\tNumber of patients with cancer in just one lung: {len(one_lu
```

```
Rate of survival when cancer is present in both lungs: 0.00
        Number of patients with cancer in both lungs: 5
Rate of survival when cancer is present in only one lung: 0.22
        Number of patients with cancer in just one lung: 185
```

Although the sample size is low, we can make an assumption for now that having cancer present in both lungs will not result in a favorable outcome for the patient.

Given more time and potentially more data, I would be curious to see the survival rates of the different sites where the cancer is present. Should there be a significant difference, it could be worth the time to create unique columns for each site and a flag whether this is a primary site or not.

In [41]:
```
hist_lst = list(clinical_df['Histology'].unique())
hist_lst
```

Out[41]:
```
['Squamous cell carcinoma', 'Adenocarcinoma', 'Large-cell carcinoma']
```

In [42]:
```
clinical_df['Histology'].value_counts()
```

Out[42]:
```
Adenocarcinoma              86
Squamous cell carcinoma     77
Large-cell carcinoma        27
Name: Histology, dtype: int64
```

Since this feature is categorical, we can one-hot encode it.

In [43]:
```
tum_size_lst = sorted(list(clinical_df['Tumor.Size'].unique()))
tum_size_lst
```

Out[43]:
```
[1.4,
 nan,
 1.0,
 1.5,
 1.6,
 1.8,
 1.9,
 2.0,
 2.5,
 3.5,
 3.6,
 4.0,
 4.4,
 5.3,
 5.4,
 5.5,
 8.0,
 8.5,
 9.0,
 10.0]
```

In [44]:
```
clinical_df['Tumor.Size'].value_counts().sum()
```

Out[44]:
```
98
```

For the tumor size feature, we can take the mean and assign this to all the null values.

Preprocessing

one-hot encode histology, one or both lungs, and any other categorical feature

clean up grade

Make radiation 0 or 1 or a flag

drop 'M'

MAYBE scale all values

In [45]:
```
data = clinical_df.copy()

data.head()
```

Out[45]:

	ID	Outcome	Survival.Months	Age	Grade	Num.Primaries	T	N	M	Radiation	Sta
0	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
1	2	Dead	19.0	73	2	0	UNK	2.0	0.0	5	
2	3	Dead	13.0	72	3	0	2	2.0	0.0	0	
3	4	Dead	15.0	69	9	1	1a	0.0	1.0	0	
4	5	Dead	10.0	76	9	0	UNK	NaN	NaN	0	

In [46]:
```
data = data.drop(columns=['Grade','M'])
```

In [47]:
```
data['Tumor.Size'].mean()
```

Out[47]: 4.494897959183674

In [48]:
```
data = data.reset_index()
new_T = []
new_N = []
new_Stage = []
new_Radiation = []
new_PrimarySite = []
new_TumorSize = []

N_mean = "{:.2f}".format(data['N'].mean())
TumorSize_mean = "{:.2f}".format(data['Tumor.Size'].mean())

for index, row in data.iterrows():
    # converting non-null values to integers in T
    if (row['T'] != 'UNK'):
        new_T.append(int(row['T'][0]))
    else:
        new_T.append(row['T'])

    # converting non-null values to the mean in N
    if (math.isnan(row['N'])):
        new_N.append(N_mean)
    else:
        new_N.append(row['N'])
```

```
    # creating bool flag for radiation
    if (row['Radiation'] == 5):
        new_Radiation.append(1)
    else:
        new_Radiation.append(0)

    # creating a bool flag for cancer in one or two lungs
    if (row['Primary.Site'] == 'Both Lung'):
        new_PrimarySite.append(1)
    else:
        new_PrimarySite.append(0)

    # setting all null values to mean for tumor size
    if (math.isnan(row['Tumor.Size'])):
        new_TumorSize.append(TumorSize_mean)
    else:
        new_TumorSize.append(row['Tumor.Size'])

    # converting cancer stages to numeric values
    if (row['Stage'] in ['IA','IB','1B']):
        new_Stage.append(1)
    elif (row['Stage'] in ['IIA','IIB']):
        new_Stage.append(2)
    elif (row['Stage'] in ['IIIA','IIIB']):
        new_Stage.append(3)
    elif (row['Stage'] in ['IV','IVB']):
        new_Stage.append(4)
```

In [49]:
```
T_mean = "{:.2f}".format(mean(val for val in new_T if val != 'UNK'))

for i in range(len(new_T)):
    if (new_T[i] == 'UNK'):
        new_T[i] = T_mean
```

In [50]:
```
data['T'] = new_T
data['N'] = new_N
data['Stage'] = new_Stage
data['Radiation'] = new_Radiation
data['Both Lungs'] = new_PrimarySite
data['Tumor Size'] = new_TumorSize
```

In [51]:
```
data = data.drop(columns=['index','Primary.Site','Tumor.Size'])

data.head()
```

Out[51]:

	ID	Outcome	Survival.Months	Age	Num.Primaries	T	N	Radiation	Stage	Hist
0	1	Alive	9.0	67	0	2.43	2.0	0	4	Squamo carc
1	2	Dead	19.0	73	0	2.43	2.0	1	4	Adenocarc
2	3	Dead	13.0	72	0	2	2.0	0	3	Adenocarc
3	4	Dead	15.0	69	1	1	0.0	0	1	Adenocarc
4	5	Dead	10.0	76	0	2.43	1.14	0	3	Larg carc

In [52]:
```
onehot_hist = pd.get_dummies(data['Histology'])
data = data.drop(columns=['Histology'])
data = data.join(onehot_hist)

data.head()
```

Out[52]:

	ID	Outcome	Survival.Months	Age	Num.Primaries	T	N	Radiation	Stage	Num.Muta
0	1	Alive	9.0	67	0	2.43	2.0	0	4	
1	2	Dead	19.0	73	0	2.43	2.0	1	4	
2	3	Dead	13.0	72	0	2	2.0	0	3	
3	4	Dead	15.0	69	1	1	0.0	0	1	
4	5	Dead	10.0	76	0	2.43	1.14	0	3	

In [53]:
```
genomic_df.head()
```

Out[53]:

	ID	Gene
0	1	AKT1
1	158	AKT1
2	88	ALK_Col1
3	132	ALK_Col1
4	18	ALK_Col2

In [54]:
```
clean_gene_survival_df
```

Out[54]:

	Gene	Number of Patients	Survival Rate
4	KRAS_Col1	55	0.24
6	STK11	23	0.17
7	TSC2	31	0.10
9	TP53_Col1	117	0.26
10	CDKN2A	45	0.07
14	MSH2	30	0.03

In [55]:
```
KRAS_Col1 = []
STK11     = []
TSC2      = []
TP53_Col1 = []
CDKN2A    = []
MSH2      = []

gene_lst = [KRAS_Col1, STK11, TSC2, TP53_Col1, CDKN2A, MSH2]
gene_str_lst = ['KRAS_Col1', 'STK11', 'TSC2', 'TP53_Col1', 'CDKN2A', '
```

In [56]:
```
for index_2, row_2 in data.iterrows():
    for gene,gene_str in zip(gene_lst,gene_str_lst):
        gene_df = genomic_df[genomic_df['Gene'] == gene_str]

        if row_2['ID'] in gene_df['ID'].values:
            gene.append(1)
        else:
            gene.append(0)
```

In [57]:
```
for col_name, col in zip(gene_str_lst,gene_lst):
    data[col_name] = col
```

In [58]:
```
# data with gene columns attached
data.head()
```

Out[58]:

	ID	Outcome	Survival.Months	Age	Num.Primaries	T	N	Radiation	Stage	Num.Muta
0	1	Alive	9.0	67	0	2.43	2.0	0	4	
1	2	Dead	19.0	73	0	2.43	2.0	1	4	
2	3	Dead	13.0	72	0	2	2.0	0	3	
3	4	Dead	15.0	69	1	1	0.0	0	1	
4	5	Dead	10.0	76	0	2.43	1.14	0	3	

5 rows × 22 columns

In [59]:
```
# if things go horribly wrong, perhaps reset the indices
```

In [60]:
```
new_Outcome = []
new_MutDiff = []
# final iteration
for index_3, row_3 in data.iterrows():
    # converting our target column to 0/1 flag
    if (row_3['Outcome'] == 'Alive'):
        new_Outcome.append(1)
    elif (row_3['Outcome'] == 'Dead'):
        new_Outcome.append(0)

    # adding bool flag for whenever num mutations is different from nu
    if (row_3['Num.Mutated.Genes'] != row_3['Num.Mutations']):
        new_MutDiff.append(1)
    else:
        new_MutDiff.append(0)

data['Outcome'] = new_Outcome
data['Mutation Difference'] = new_MutDiff
data = data.drop(columns=['Num.Mutated.Genes','Num.Mutations'])

# final version of the dataframe before splitting
data.head()
```

Out[60]:

	ID	Outcome	Survival.Months	Age	Num.Primaries	T	N	Radiation	Stage	Both Lungs	...
0	1	1	9.0	67	0	2.43	2.0	0	4	0	...
1	2	0	19.0	73	0	2.43	2.0	1	4	0	...
2	3	0	13.0	72	0	2	2.0	0	3	0	...
3	4	0	15.0	69	1	1	0.0	0	1	0	...
4	5	0	10.0	76	0	2.43	1.14	0	3	0	...

5 rows × 21 columns

In [61]:
```
# original dataframe
clinical_df.head()
```

Out[61]:

	ID	Outcome	Survival.Months	Age	Grade	Num.Primaries	T	N	M	Radiation	Sta
0	1	Alive	9.0	67	4	0	UNK	2.0	NaN	0	
1	2	Dead	19.0	73	2	0	UNK	2.0	0.0	5	
2	3	Dead	13.0	72	3	0	2	2.0	0.0	0	
3	4	Dead	15.0	69	9	1	1a	0.0	1.0	0	
4	5	Dead	10.0	76	9	0	UNK	NaN	NaN	0	

BUILDING A RANDOM FOREST MODEL

In [62]:
```
X = data.drop(columns=['ID','Outcome'])
X.shape
```

Out[62]: (190, 19)

In [63]:
```
y = data['Outcome'].to_numpy()
y.shape
```

Out[63]: (190,)

In [64]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

In [65]:
```
model = RandomForestClassifier(n_estimators=10, criterion="entropy")

model.fit(X_train, y_train)
preds = model.predict(X_test)
```

In [66]:
```
y_test
```

Out[66]: array([0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0])

In [67]:
```
preds
```

Out[67]: array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0])

In [68]:
```
num_correct = 0
tot_preds = len(preds)

for i in range(tot_preds):
    if (y_test[i] == preds[i]):
        num_correct += 1

acc = "{:.2f}".format((num_correct / tot_preds) * 100)
print(f"The accuracy of the model is: {acc}%")
```

```
The accuracy of the model is: 94.74%
```