Masters Thesis on

Benchmarking Somatic Variant Callers

By R. S. Chintalapati

29th April 2021

Declaration

Abstract

Variant calling pipelines are of two types namely Germline and Somatic. In terms of benchmarking, many Germline variant calling pipelines were compared and documented by the Genome in a Bottle (GIAB) Consortium standards. For Somatic variants calling pipelines, only a few comparisons were achieved because the comparisons are challenging and less established. In the diploid human genome, a variant can be found either homozygously, heterozygously or not at all and these three levels complicate the comparison process. Tumors, on the other hand, are inhomogeneous cells that might carry variants possibly with rare mutations not found in the others making the Somatic variant calling pipelines benchmarking challenging. The goal of this thesis is to consider an artificial dataset pair from the GIAB project and benchmarking two Somatic variant callers based on a few factors to choose a variant caller for cancer research.

Chapter 1

Introduction

Chapter 2

Related Works

Chapter 3

Background

Chapter 4

Approach

Chapter 5

Experiments

To compare the Strelka, VarScan and Truth VCF files, the first step is calculating the number of positions, SNPs and Indels. This step helps in finding out the common positions, SNPs and Indels amongst the VCF files and even helps in normalising the variants in the second step.

In the second step, to know the bias of the variant callers, each variant type is counted and normalised. Through this step, information about a variant caller showing a specific bias in calling variant combinations quite often can be known in comparison to the truth data.

In the third step, to observe genetic diversity, allele frequencies between the truth data and the Strelka and VarScan VCF files are compared to learn which variant caller is comparably close to the truth data and which variant caller calls the reads with higher allele frequencies.

In the fourth step, to know the number of times a nucleotide has been read, the read depth of each VCF file is compared. This step reveals the coverage of a read in every variant caller while providing an idea as to which variant caller has the better coverage.

In the fifth and final step, ALT variants in each of the VCF files are compared for benchmarking. Through this step, the truth data is compared with the Strelka VCF file and VarScan VCF file individually to know the number of true positives, true negatives, false positives and false negatives.

5.1 Positions, SNPs & Indels

A single-nucleotide polymorphism is a substitution of a single nucleotide at a specific position in the genome that is present in a sufficiently large fraction of the population. Indel is a molecular biology term for an insertion or deletion of bases in the genome of an organism. In this section, Positions, SNPs and INDELs from different somatic variant callers are compared with the artifical truth data obtained from https://ftp-trace.ncbi.nlm.nih.gov/

Tumor Purity	**Positions**	**SNPs**	**Indels**
0.3	13,315	12,897	418
0.5	13,315	12,897	418
0.7	13,315	12,897	418

Table 5.1: Values from Strelka VCF files

Tumor Purity	**Positions**	**SNPs**	**Indels**
0.3	29,316	26,312	3,004
0.5	29,294	26,290	3,004
0.7	29,171	26,185	2,986

Table 5.2: Values from VarScan VCF files

Tumor Purity	**Positions**	**SNPs**	**Indels**
1.0	11,04,786	10,07,793	96,993

Table 5.3: Values from Truth Data VCF files

5.2 Variants

A variant is an alteration in the most common DNA sequence. In this section, variants from different somatic variant callers are compared with the artifical truth data obtained from https://ftp-trace.ncbi.nlm.nih.gov/

Type	Positions
Strelka	13,315
VarScan	29,316
Truth Data	11,04,786
Strelka & VarScan	3,104
VarScan & Truth Data	2,843
Strelka & Truth Data	3,791
Strelka, VarScan & Truth Data	1,052

Table 5.4: Positions comparison with Tumor Purity 0.3

Type	Positions
Strelka	13,315
VarScan	29,294
Truth Data	11,04,786
Strelka & VarScan	3,096
VarScan & Truth Data	2,843
Strelka & Truth Data	3,791
Strelka, VarScan & Truth Data	1,052

Table 5.5: Positions comparison with Tumor Purity 0.5

Type	Positions
Strelka	13,315
VarScan	29,171
Truth Data	11,04,786
Strelka & VarScan	3,051
VarScan & Truth Data	2,843
Strelka & Truth Data	3,791
Strelka, VarScan & Truth Data	1,052

Table 5.6: Positions comparison with Tumor Purity 0.7

5.3 Allele Frequencies

Allele frequency, or gene frequency is defined as the relative frequency of an allele at a particular locus in a population, expressed as a fraction or

Type	SNPs
Strelka	12,897
VarScan	26,312
Truth Data	10,07,793
Strelka & VarScan	2,778
VarScan & Truth Data	2,484
Strelka & Truth Data	3,150
Strelka, VarScan & Truth Data	875

Table 5.7: SNPs comparison with Tumor Purity 0.3

Type	SNPs
Strelka	12,897
VarScan	26,290
Truth Data	10,07,793
Strelka & VarScan	2,770
VarScan & Truth Data	2,484
Strelka & Truth Data	3,150
Strelka, VarScan & Truth Data	875

Table 5.8: SNPs comparison with Tumor Purity 0.5

Type	SNPs
Strelka	12,897
VarScan	26,185
Truth Data	10,07,793
Strelka & VarScan	2,729
VarScan & Truth Data	2,484
Strelka & Truth Data	3,150
Strelka, VarScan & Truth Data	875

Table 5.9: SNPs comparison with Tumor Purity 0.7

percentage. In this section, allele frequencies from different somatic variant callers are compared with the artifical truth data obtained from https://ftp-trace.ncbi.nlm.nih.gov/

Type	Indels
Strelka	418
VarScan	3,004
Truth Data	96,993
Strelka & VarScan	110
VarScan & Truth Data	200
Strelka & Truth Data	127
Strelka, VarScan & Truth Data	29

Table 5.10: Indels comparison with Tumor Purity 0.3 & 0.5

Type	Indels
Strelka	418
VarScan	2,986
Truth Data	96,993
Strelka & VarScan	107
VarScan & Truth Data	200
Strelka & Truth Data	127
Strelka, VarScan & Truth Data	29

Table 5.11: Indels comparison with Tumor Purity 0.7

Figure 5.1 shows Strelka Allele Frequency Counts.
Figure 5.2 shows VarScan Allele Frequency Counts.
Figure 5.3 shows Allele Frequencies with Tumor Purity of 0.3, 0.5 & 0.7.

5.4 Read Depth

Read Depth describes the number of times that a given nucleotide in the genome has been read in an experiment. In this section, read depths from different somatic variant callers are compared with the artifical truth data obtained from https://ftp-trace.ncbi.nlm.nih.gov/

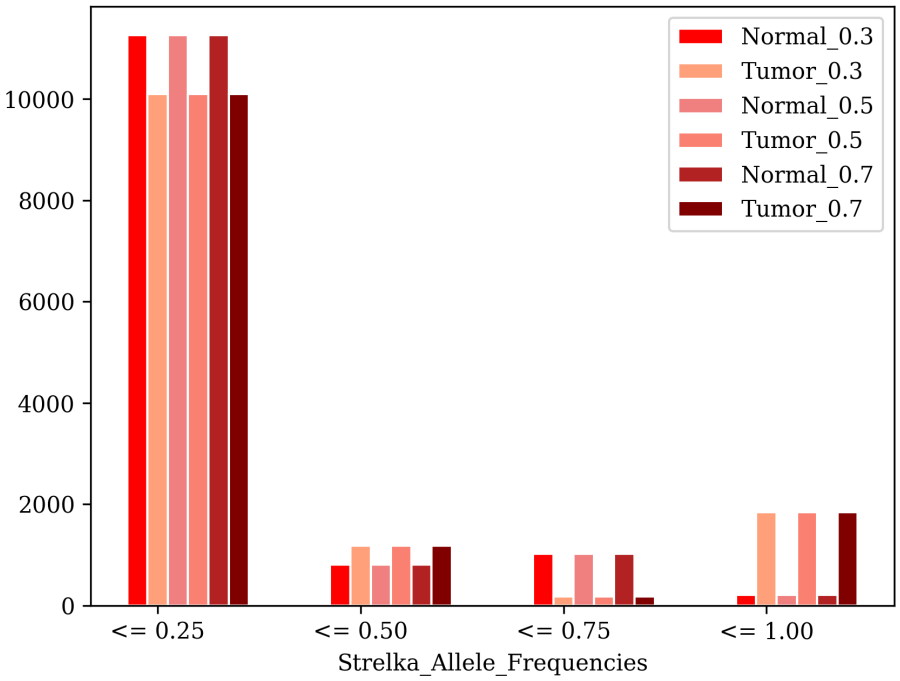

Figure 5.1: Strelka Allele Frequency Counts

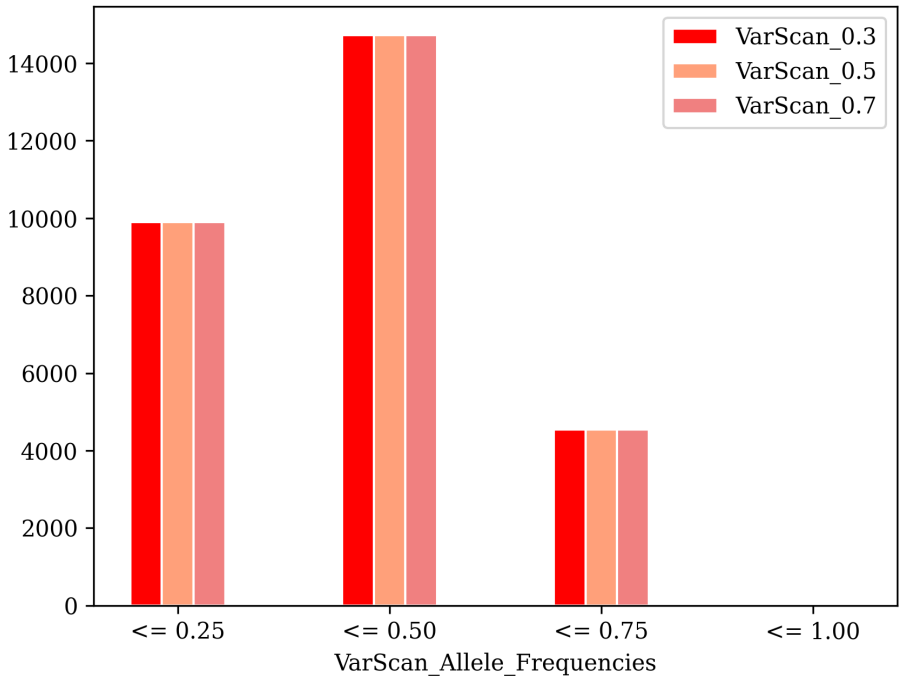

Figure 5.2: VarScan Allele Frequency Counts

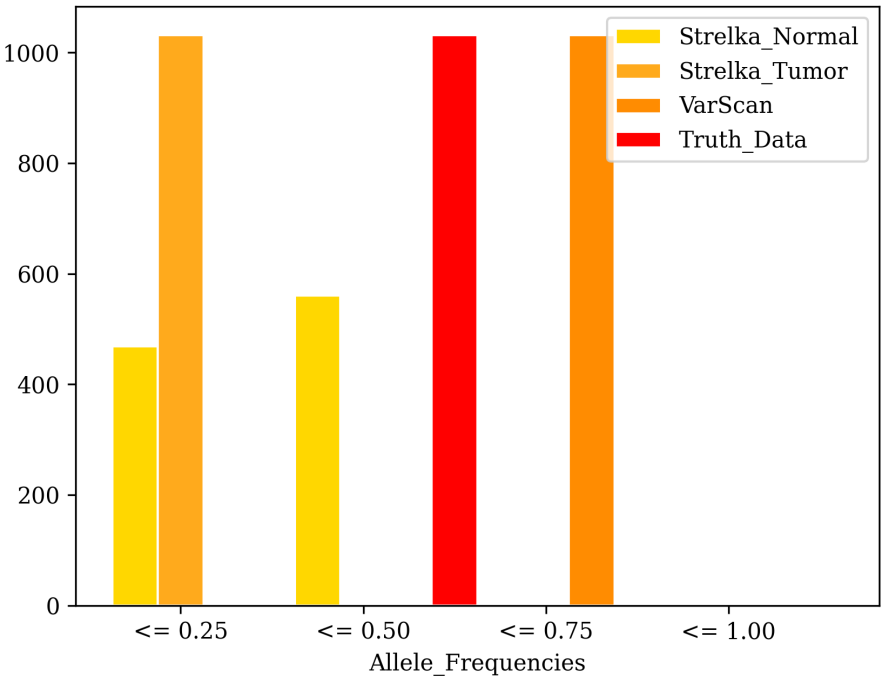

Figure 5.3: Allele Frequencies with Tumor Purity of 0.3, 0.5 & 0.7

Combination	**Tumor Purity 0.3**	**Tumor Purity 0.5**	**Tumor Purity 0.7**
AA	0	0	0
AT	393	393	393
AG	1,196	1,196	1,196
AC	1,942	1,942	1,942
TT	0	0	0
TA	647	647	647
TG	1,491	1,491	1,491
TC	1,245	1,245	1,245
GG	0	0	0
GA	1,700	1,700	1,700
GT	882	882	882
GC	478	478	478
CC	0	0	0
CA	1,072	1,072	1,072
CT	1,441	1,441	1,441
CG	409	409	409

Table 5.12: Variant counts in Strelka variant caller

Combination	**Tumor Purity 0.3**	**Tumor Purity 0.5**	**Tumor Purity 0.7**
AA	0	0	0
AT	662	662	662
AG	4,354	4,354	4,350
AC	962	958	947
TT	0	0	0
TA	722	720	715
TG	990	989	976
TC	4,343	4,342	4,336
GG	0	0	0
GA	4,960	4,959	4,950
GT	969	966	949
GC	1,212	1,212	1,210
CC	0	0	0
CA	1,059	1,054	1,033
CT	4,843	4,840	4,829
CG	1,235	1,233	1,231

Table 5.13: Variant counts in VarScan variant caller

Combination	**Tumor Purity 1.0**
AA	0
AT	32,639
AG	1,52,935
AC	38,384
TT	0
TA	32,817
TG	37,707
TC	1,53,474
GG	0
GA	1,91,603
GT	44,538
GC	43,851
CC	0
CA	44,253
CT	1,91,442
CG	44,049

Table 5.14: Variant counts in Truth Data

Combination	**Strelka**	**VarScan**	**Truth Data**
AA	0	0	0
AT	3.04	2.51	3.23
AG	9.27	16.54	15.17
AC	15.05	3.65	3.80
TT	0	0	0
TA	5.01	2.74	3.25
TG	11.56	3.76	3.74
TC	9.65	16.50	15.22
GG	0	0	0
GA	13.18	18.85	19.01
GT	6.83	3.68	4.41
GC	3.70	4.60	4.35
CC	0	0	0
CA	8.31	4.02	4.39
CT	11.17	18.40	18.99
CG	3.17	4.69	4.37

Table 5.15: Normalised variant counts with Tumor Purity 0.3

Combination	**Strelka**	**VarScan**	**Truth Data**
AA	0	0	0
AT	3.04	2.51	3.23
AG	9.27	16.56	15.17
AC	15.05	3.64	3.80
TT	0	0	0
TA	5.01	2.73	3.25
TG	11.56	3.76	3.74
TC	9.65	16.51	15.22
GG	0	0	0
GA	13.18	18.86	19.01
GT	6.83	3.67	4.41
GC	3.70	4.61	4.35
CC	0	0	0
CA	8.31	4.00	4.39
CT	11.17	18.41	18.99
CG	3.17	4.69	4.37

Table 5.16: Normalised variant counts with Tumor Purity 0.5

Combination	**Strelka**	**VarScan**	**Truth Data**
AA	0	0	0
AT	3.04	2.51	3.23
AG	9.27	16.61	15.17
AC	15.05	3.61	3.80
TT	0	0	0
TA	5.01	2.73	3.25
TG	11.56	3.72	3.74
TC	9.65	16.55	15.22
GG	0	0	0
GA	13.18	18.90	19.01
GT	6.83	3.62	4.41
GC	3.70	4.62	4.35
CC	0	0	0
CA	8.31	3.94	4.39
CT	11.17	18.44	18.99
CG	3.17	4.70	4.37

Table 5.17: Normalised variant counts with Tumor Purity 0.7

Format	**Purity**	$\leq$ **0.25**	**0.25** $>$ **&** $\leq$ **0.50**	**0.50** $>$ **&** $\leq$ **0.75**	$>$ **0.75**
Normal	0.3	11,266	809	1,020	212
Tumor	0.3	10,095	1,185	177	1,845
Normal	0.5	11,266	809	1,020	212
Tumor	0.5	10,095	1,185	177	1,845
Normal	0.7	11,266	809	1,020	212
Tumor	0.7	10,095	1,185	177	1,845

Table 5.18: Allele Frequencies count from Strelka variant caller VCF files

Purity	$\leq$ **0.25**	**0.25** $>$ **&** $\leq$ **0.50**	**0.50** $>$ **&** $\leq$ **0.75**	$>$ **0.75**
0.3	9,893	14,732	4,546	0
0.5	9,893	14,732	4,546	0
0.7	9,893	14,732	4,546	0

Table 5.19: Allele Frequencies count from VarScan variant caller VCF files

$\leq$ **0.25**	**0.25** $>$ & $\leq$ **0.50**	**0.50** $>$ & $\leq$ **0.75**	$>$ **0.75**
11,266	809	1,020	212

Table 5.20: Allele Frequencies count from Truth Data VCF file

Type	$\leq$ **0.25**	**0.25** $>$ & $\leq$ **0.50**	**0.50** $>$ & $\leq$ **0.75**	$>$ **0.75**
Strelka Normal	469	561	2	0
Strelka Tumor	1,032	0	0	0
VarScan	0	0	1,032	0
Truth Data	0	1,032	0	0

Table 5.21: Allele Frequencies count with Tumor Purity of 0.3, 0.5 & 0.7

Format	**Purity**	**Minimum**	**Maximum**	**Mean**	**Median**	**Mode**
Normal	0.3	1	299	58.32	53	0 43
Normal	0.3	0	114	20.71	19	0 17
Normal	0.5	1	299	58.32	53	0 43
Tumor	0.5	0	114	20.71	19	0 17
Normal	0.7	1	299	58.32	53	0 43
Tumor	0.7	0	114	20.71	19	0 17

Table 5.22: Read Depth statistics from Strelka variant caller VCF files

Format	**Purity**	**Minimum**	**Maximum**	**Mean**	**Median**	**Mode**
Normal	0.3	10	99	∞	55	0 38
Normal	0.3	10	98	∞	19	0 17
Normal	0.5	10	99	∞	55	0 38
Tumor	0.5	10	98	∞	19	0 17
Normal	0.7	10	99	∞	55	0 38
Tumor	0.7	10	98	∞	19	0 17

Table 5.23: Read Depth statistics from VarScan variant caller VCF files

Minimum	Maximum	Mean	Median	Mode
100	999	∞	647	0 640

Table 5.24: Read Depth statistics from Truth Data VCF file

Type	Minimum	Maximum	Mean	Median	Mode
Strelka Normal	20	139	70.38	64	0 59
Strelka Tumor	14	61	26.61	25	0 19
VarScan Normal	16	117	55.11	50	0 41
VarScan Tumor	9	52	21.18	19	0 16
Truth Data	274	1,104	662.83	660	0 730

Table 5.25: Read Depth statistics from Tumor Purity 0.3, 0.5, & 0.7

Chapter 6

Conclusions

Chapter 7

Acknowledgments

Chapter 8

Code

8.1 Allele Frequency

8.1.1 Variant Caller Code

8.1.1.1 Strelka Allele Frequency Code

```
# Importing the needed packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import csv

matplotlib.rcParams['font.sans-serif'] = ['Computer Modern Roman',
    'sans-serif']

# The first step is to selected the neccessary columns.
# Step 1 - 'cut -f 1-2,4-5,9-11 Input.vcf > Output.vcf'

# Removing all the rows starting with a #
# Step 2 - 'sed '/^#/d' Output.vcf > Updated_Output.vcf'

# Consider the Updated_Output.vcf as input.
```

```
dff = pd.read_csv("Selected_Strelka_0.3_Indels.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Selected_Strelka_0.5_Indels.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Selected_Strelka_0.7_Indels.vcf", sep = '\t',
    index_col= False)

# Renaming the columns after importing the input.
dff.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'NORMAL',
    'TUMOR']
dff1.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'NORMAL',
    'TUMOR']
dff2.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'NORMAL',
    'TUMOR']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff1['CHROM'].astype(str) + '-' +
    dff1['POS'].astype(str)
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising them.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]

dff1 = dff1.drop(['CHROM', 'POS'], axis=1)
cols = dff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]

dff2 = dff2.drop(['CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
```

```
# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
dff[['Normal_DP', 'Normal_DP2', 'Normal_TAR', 'Normal_TIR',
    'Normal_TOR', 'Normal_DP50', 'Normal_FDP50', 'Normal_SUBDP50',
    'Normal_BCN50']] = dff['NORMAL'].str.split(':',expand=True)
dff[['Tumor_DP', 'Tumor_DP2', 'Tumor_TAR', 'Tumor_TIR',
    'Tumor_TOR', 'Tumor_DP50', 'Tumor_FDP50', 'Tumor_SUBDP50',
    'Tumor_BCN50']] = dff['TUMOR'].str.split(':',expand=True)

dff1[['Normal_DP', 'Normal_DP2', 'Normal_TAR', 'Normal_TIR',
    'Normal_TOR', 'Normal_DP50', 'Normal_FDP50', 'Normal_SUBDP50',
    'Normal_BCN50']] = dff1['NORMAL'].str.split(':',expand=True)
dff1[['Tumor_DP', 'Tumor_DP2', 'Tumor_TAR', 'Tumor_TIR',
    'Tumor_TOR', 'Tumor_DP50', 'Tumor_FDP50', 'Tumor_SUBDP50',
    'Tumor_BCN50']] = dff1['TUMOR'].str.split(':',expand=True)

dff2[['Normal_DP', 'Normal_DP2', 'Normal_TAR', 'Normal_TIR',
    'Normal_TOR', 'Normal_DP50', 'Normal_FDP50', 'Normal_SUBDP50',
    'Normal_BCN50']] = dff2['NORMAL'].str.split(':',expand=True)
dff2[['Tumor_DP', 'Tumor_DP2', 'Tumor_TAR', 'Tumor_TIR',
    'Tumor_TOR', 'Tumor_DP50', 'Tumor_FDP50', 'Tumor_SUBDP50',
    'Tumor_BCN50']] = dff2['TUMOR'].str.split(':',expand=True)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['FORMAT', 'NORMAL', 'TUMOR', 'Normal_DP',
    'Normal_DP2', 'Normal_TOR', 'Normal_DP50', 'Normal_FDP50',
    'Normal_SUBDP50', 'Normal_BCN50', 'Tumor_DP', 'Tumor_DP2',
    'Tumor_DP2', 'Tumor_TOR', 'Tumor_DP50', 'Tumor_FDP50',
    'Tumor_SUBDP50', 'Tumor_BCN50'], axis=1)

dff1 = dff1.drop(['FORMAT', 'NORMAL', 'TUMOR', 'Normal_DP',
    'Normal_DP2', 'Normal_TOR', 'Normal_DP50', 'Normal_FDP50',
    'Normal_SUBDP50', 'Normal_BCN50', 'Tumor_DP', 'Tumor_DP2',
    'Tumor_DP2', 'Tumor_TOR', 'Tumor_DP50', 'Tumor_FDP50',
    'Tumor_SUBDP50', 'Tumor_BCN50'], axis=1)

dff2 = dff2.drop(['FORMAT', 'NORMAL', 'TUMOR', 'Normal_DP',
    'Normal_DP2', 'Normal_TOR', 'Normal_DP50', 'Normal_FDP50',
    'Normal_SUBDP50', 'Normal_BCN50', 'Tumor_DP', 'Tumor_DP2',
```

```
    'Tumor_DP2', 'Tumor_TOR', 'Tumor_DP50', 'Tumor_FDP50',
    'Tumor_SUBDP50', 'Tumor_BCN50'], axis=1)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
dff[['Normal_TAR_First', 'Normal_TAR_Second']] =
    dff['Normal_TAR'].str.split(',',expand=True)
dff[['Normal_TIR_First', 'Normal_TIR_Second']] =
    dff['Normal_TIR'].str.split(',',expand=True)
dff[['Tumor_TAR_First', 'Tumor_TAR_Second']] =
    dff['Tumor_TAR'].str.split(',',expand=True)
dff[['Tumor_TIR_First', 'Tumor_TIR_Second']] =
    dff['Tumor_TIR'].str.split(',',expand=True)

dff1[['Normal_TAR_First', 'Normal_TAR_Second']] =
    dff1['Normal_TAR'].str.split(',',expand=True)
dff1[['Normal_TIR_First', 'Normal_TIR_Second']] =
    dff1['Normal_TIR'].str.split(',',expand=True)
dff1[['Tumor_TAR_First', 'Tumor_TAR_Second']] =
    dff1['Tumor_TAR'].str.split(',',expand=True)
dff1[['Tumor_TIR_First', 'Tumor_TIR_Second']] =
    dff1['Tumor_TIR'].str.split(',',expand=True)

dff2[['Normal_TAR_First', 'Normal_TAR_Second']] =
    dff2['Normal_TAR'].str.split(',',expand=True)
dff2[['Normal_TIR_First', 'Normal_TIR_Second']] =
    dff2['Normal_TIR'].str.split(',',expand=True)
dff2[['Tumor_TAR_First', 'Tumor_TAR_Second']] =
    dff2['Tumor_TAR'].str.split(',',expand=True)
dff2[['Tumor_TIR_First', 'Tumor_TIR_Second']] =
    dff2['Tumor_TIR'].str.split(',',expand=True)

# Renaming the new table with column names.
dff.columns = ['CHROM_POS', 'REF', 'ALT', 'Normal_TAR',
    'Normal_TIR', 'Tumor_TAR', 'Tumor_TAR', 'Normal_TAR_First',
    'Normal_TAR_Second', 'Normal_TIR_First', 'Normal_TIR_Second',
    'Tumor_TAR_First', 'Tumor_TAR_Second', 'Tumor_TIR_First',
    'Tumor_TIR_Second']
```

```
dff1.columns = ['CHROM_POS', 'REF', 'ALT', 'Normal_TAR',
    'Normal_TIR', 'Tumor_TAR', 'Tumor_TAR', 'Normal_TAR_First',
    'Normal_TAR_Second', 'Normal_TIR_First', 'Normal_TIR_Second',
    'Tumor_TAR_First', 'Tumor_TAR_Second', 'Tumor_TIR_First',
    'Tumor_TIR_Second']

dff2.columns = ['CHROM_POS', 'REF', 'ALT', 'Normal_TAR',
    'Normal_TIR', 'Tumor_TAR', 'Tumor_TAR', 'Normal_TAR_First',
    'Normal_TAR_Second', 'Normal_TIR_First', 'Normal_TIR_Second',
    'Tumor_TAR_First', 'Tumor_TAR_Second', 'Tumor_TIR_First',
    'Tumor_TIR_Second']

# Dropping of the unnecessary columns and reorganising them.
dff = dff.drop(['Normal_TAR_Second', 'Normal_TIR_Second',
    'Tumor_TAR_Second', 'Tumor_TIR_Second'], axis=1)
print(dff)

dff1 = dff1.drop(['Normal_TAR_Second', 'Normal_TIR_Second',
    'Tumor_TAR_Second', 'Tumor_TIR_Second'], axis=1)
print(dff1)

dff2 = dff2.drop(['Normal_TAR_Second', 'Normal_TIR_Second',
    'Tumor_TAR_Second', 'Tumor_TIR_Second'], axis=1)
print(dff2)

# Converting string values columns to int for calculations.
dff['Normal_TAR_First'] = dff['Normal_TAR_First'].astype(int)
dff['Normal_TIR_First'] = dff['Normal_TIR_First'].astype(int)
dff['Tumor_TAR_First'] = dff['Tumor_TAR_First'].astype(int)
dff['Tumor_TIR_First'] = dff['Tumor_TIR_First'].astype(int)

dff1['Normal_TAR_First'] = dff1['Normal_TAR_First'].astype(int)
dff1['Normal_TIR_First'] = dff1['Normal_TIR_First'].astype(int)
dff1['Tumor_TAR_First'] = dff1['Tumor_TAR_First'].astype(int)
dff1['Tumor_TIR_First'] = dff1['Tumor_TIR_First'].astype(int)

dff2['Normal_TAR_First'] = dff2['Normal_TAR_First'].astype(int)
dff2['Normal_TIR_First'] = dff2['Normal_TIR_First'].astype(int)
dff2['Tumor_TAR_First'] = dff2['Tumor_TAR_First'].astype(int)
dff2['Tumor_TIR_First'] = dff2['Tumor_TIR_First'].astype(int)
```

```
# Adding the values for the formula.
dff['SUM'] = dff["Normal_TAR_First"] + dff["Normal_TIR_First"]
dff['COMMON'] = dff["Tumor_TAR_First"] + dff["Tumor_TIR_First"]
print(dff)

dff1['SUM'] = dff1["Normal_TAR_First"] + dff1["Normal_TIR_First"]
dff1['COMMON'] = dff1["Tumor_TAR_First"] + dff1["Tumor_TIR_First"]
print(dff1)

dff2['SUM'] = dff2["Normal_TAR_First"] + dff2["Normal_TIR_First"]
dff2['COMMON'] = dff2["Tumor_TAR_First"] + dff2["Tumor_TIR_First"]
print(dff2)

# Getting Allele Frequency
dff['Normal_Allele_Frequency'] = dff['Normal_TIR_First']/dff['SUM']
dff['Tumor_Allele_Frequency'] =
    dff['Tumor_TIR_First']/dff['COMMON']

dff1['Normal_Allele_Frequency'] =
    dff1['Normal_TIR_First']/dff1['SUM']
dff1['Tumor_Allele_Frequency'] =
    dff1['Tumor_TIR_First']/dff1['COMMON']

dff2['Normal_Allele_Frequency'] =
    dff2['Normal_TIR_First']/dff2['SUM']
dff2['Tumor_Allele_Frequency'] =
    dff2['Tumor_TIR_First']/dff2['COMMON']

# Converting string values columans to int.
dff['Normal_Allele_Frequency'] =
    dff['Normal_Allele_Frequency'].astype(float).round(2)
dff['Tumor_Allele_Frequency'] =
    dff['Tumor_Allele_Frequency'].astype(float).round(2)

dff1['Normal_Allele_Frequency'] =
    dff1['Normal_Allele_Frequency'].astype(float).round(2)
dff1['Tumor_Allele_Frequency'] =
    dff1['Tumor_Allele_Frequency'].astype(float).round(2)
```

```
dff2['Normal_Allele_Frequency'] =
    dff2['Normal_Allele_Frequency'].astype(float).round(2)
dff2['Tumor_Allele_Frequency'] =
    dff2['Tumor_Allele_Frequency'].astype(float).round(2)

# Concatinating the "CHROM" and "POS"
dff["Normal_Allele_Frequency"] = dff['REF'].astype(str) + ':' +
    dff['Normal_Allele_Frequency'].astype(str)
dff["Tumor_Allele_Frequency"] = dff['REF'].astype(str) + ':' +
    dff['Tumor_Allele_Frequency'].astype(str)

dff1["Normal_Allele_Frequency"] = dff1['REF'].astype(str) + ':' +
    dff1['Normal_Allele_Frequency'].astype(str)
dff1["Tumor_Allele_Frequency"] = dff1['REF'].astype(str) + ':' +
    dff1['Tumor_Allele_Frequency'].astype(str)

dff2["Normal_Allele_Frequency"] = dff2['REF'].astype(str) + ':' +
    dff2['Normal_Allele_Frequency'].astype(str)
dff2["Tumor_Allele_Frequency"] = dff2['REF'].astype(str) + ':' +
    dff2['Tumor_Allele_Frequency'].astype(str)

# Dropping of the unnecessary columns and reorganising them.
dff = dff.drop(['REF', 'ALT', 'Normal_TAR', 'Normal_TIR',
    'Tumor_TAR', 'Tumor_TAR', 'Normal_TAR_First',
    'Normal_TIR_First', 'Tumor_TAR_First', 'Tumor_TIR_First',
    'SUM', 'COMMON'], axis=1)
print(dff)

dff1 = dff1.drop(['REF', 'ALT', 'Normal_TAR', 'Normal_TIR',
    'Tumor_TAR', 'Tumor_TAR', 'Normal_TAR_First',
    'Normal_TIR_First', 'Tumor_TAR_First', 'Tumor_TIR_First',
    'SUM', 'COMMON'], axis=1)
print(dff1)

dff2 = dff2.drop(['REF', 'ALT', 'Normal_TAR', 'Normal_TIR',
    'Tumor_TAR', 'Tumor_TAR', 'Normal_TAR_First',
    'Normal_TIR_First', 'Tumor_TAR_First', 'Tumor_TIR_First',
    'SUM', 'COMMON'], axis=1)
print(dff2)
```

```
# Merging columns based on "CHROM-POS"
First = pd.merge(dff, dff1, on=['CHROM_POS'])
Second = pd.merge(First, dff2, on=['CHROM_POS'])

# Renaming Columns
Second.columns = ['CHROM_POS', 'Strelka_Normal_0.3',
    'Strelka_Tumor_0.3', 'Strelka_0.5_Normal', 'Strelka_0.5_Tumor',
    'Strelka_0.7_Normal', 'Strelka_0.7_Tumor']
print(Second)

# The first step is to selected the needed columns in the vcf file.
# The second step if to eliminate all lines that start with a '#'
dff = pd.read_csv("Selected_Strelka_0.3_SNP.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Selected_Strelka_0.5_SNP.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Selected_Strelka_0.7_SNP.vcf", sep = '\t',
    index_col= False)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'NORMAL',
    'TUMOR']
dff1.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'NORMAL',
    'TUMOR']
dff2.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'NORMAL',
    'TUMOR']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff1['CHROM'].astype(str) + '-' +
    dff1['POS'].astype(str)
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
```

```
dff1 = dff1.drop(['CHROM', 'POS'], axis=1)
cols = dff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]

dff2 = dff2.drop(['CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]

# Adding columns for single read depth value.
dff['REF_U'] = dff["REF"] + "U"
dff['ALT_U'] = dff["ALT"] + "U"

dff1['REF_U'] = dff1["REF"] + "U"
dff1['ALT_U'] = dff1["ALT"] + "U"

dff2['REF_U'] = dff2["REF"] + "U"
dff2['ALT_U'] = dff2["ALT"] + "U"

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
dff[['Normal_DP', 'Normal_FDP', 'Normal_SDP', 'Normal_SUBDP',
    'Normal_AU','Normal_CU', 'Normal_GU', 'Normal_TU']] =
    dff['NORMAL'].str.split(':',expand=True)
dff[['Tumor_DP', 'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP',
    'Tumor_AU', 'Tumor_CU', 'Tumor_GU', 'Tumor_TU']] =
    dff['TUMOR'].str.split(':',expand=True)

dff1[['Normal_DP', 'Normal_FDP', 'Normal_SDP', 'Normal_SUBDP',
    'Normal_AU','Normal_CU', 'Normal_GU', 'Normal_TU']] =
    dff1['NORMAL'].str.split(':',expand=True)
dff1[['Tumor_DP', 'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP',
    'Tumor_AU', 'Tumor_CU', 'Tumor_GU', 'Tumor_TU']] =
    dff1['TUMOR'].str.split(':',expand=True)

dff2[['Normal_DP', 'Normal_FDP', 'Normal_SDP', 'Normal_SUBDP',
    'Normal_AU','Normal_CU', 'Normal_GU', 'Normal_TU']] =
```

```
    dff2['NORMAL'].str.split(':',expand=True)
dff2[['Tumor_DP', 'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP',
    'Tumor_AU', 'Tumor_CU', 'Tumor_GU', 'Tumor_TU']] =
    dff2['TUMOR'].str.split(':',expand=True)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['FORMAT', 'NORMAL', 'TUMOR', 'Normal_DP',
    'Normal_FDP', 'Normal_SDP', 'Normal_SUBDP', 'Tumor_DP',
    'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP'], axis=1)
dff1 = dff1.drop(['FORMAT', 'NORMAL', 'TUMOR', 'Normal_DP',
    'Normal_FDP', 'Normal_SDP', 'Normal_SUBDP', 'Tumor_DP',
    'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP'], axis=1)
dff2 = dff2.drop(['FORMAT', 'NORMAL', 'TUMOR', 'Normal_DP',
    'Normal_FDP', 'Normal_SDP', 'Normal_SUBDP', 'Tumor_DP',
    'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP'], axis=1)

for i in dff['CHROM_POS']:
    dff.loc[dff['REF_U'] == 'AU', 'REF_Normal'] = dff.Normal_AU
    dff.loc[dff['REF_U'] == 'CU', 'REF_Normal'] = dff.Normal_CU
    dff.loc[dff['REF_U'] == 'GU', 'REF_Normal'] = dff.Normal_GU
    dff.loc[dff['REF_U'] == 'TU', 'REF_Normal'] = dff.Normal_TU
    dff.loc[dff['ALT_U'] == 'AU', 'ALT_Normal'] = dff.Normal_AU
    dff.loc[dff['ALT_U'] == 'CU', 'ALT_Normal'] = dff.Normal_CU
    dff.loc[dff['ALT_U'] == 'GU', 'ALT_Normal'] = dff.Normal_GU
    dff.loc[dff['ALT_U'] == 'TU', 'ALT_Normal'] = dff.Normal_TU
    dff.loc[dff['REF_U'] == 'AU', 'REF_Tumor'] = dff.Tumor_AU
    dff.loc[dff['REF_U'] == 'CU', 'REF_Tumor'] = dff.Tumor_CU
    dff.loc[dff['REF_U'] == 'GU', 'REF_Tumor'] = dff.Tumor_GU
    dff.loc[dff['REF_U'] == 'TU', 'REF_Tumor'] = dff.Tumor_TU
    dff.loc[dff['ALT_U'] == 'AU', 'ALT_Tumor'] = dff.Tumor_AU
    dff.loc[dff['ALT_U'] == 'CU', 'ALT_Tumor'] = dff.Tumor_CU
    dff.loc[dff['ALT_U'] == 'GU', 'ALT_Tumor'] = dff.Tumor_GU
    dff.loc[dff['ALT_U'] == 'TU', 'ALT_Tumor'] = dff.Tumor_TU
print(dff)

for i in dff1['CHROM_POS']:
    dff1.loc[dff1['REF_U'] == 'AU', 'REF_Normal'] = dff1.Normal_AU
    dff1.loc[dff1['REF_U'] == 'CU', 'REF_Normal'] = dff1.Normal_CU
    dff1.loc[dff1['REF_U'] == 'GU', 'REF_Normal'] = dff1.Normal_GU
    dff1.loc[dff1['REF_U'] == 'TU', 'REF_Normal'] = dff1.Normal_TU
```

```
    dff1.loc[dff1['ALT_U'] == 'AU', 'ALT_Normal'] = dff1.Normal_AU
    dff1.loc[dff1['ALT_U'] == 'CU', 'ALT_Normal'] = dff1.Normal_CU
    dff1.loc[dff1['ALT_U'] == 'GU', 'ALT_Normal'] = dff1.Normal_GU
    dff1.loc[dff1['ALT_U'] == 'TU', 'ALT_Normal'] = dff1.Normal_TU
    dff1.loc[dff1['REF_U'] == 'AU', 'REF_Tumor'] = dff1.Tumor_AU
    dff1.loc[dff1['REF_U'] == 'CU', 'REF_Tumor'] = dff1.Tumor_CU
    dff1.loc[dff1['REF_U'] == 'GU', 'REF_Tumor'] = dff1.Tumor_GU
    dff1.loc[dff1['REF_U'] == 'TU', 'REF_Tumor'] = dff1.Tumor_TU
    dff1.loc[dff1['ALT_U'] == 'AU', 'ALT_Tumor'] = dff1.Tumor_AU
    dff1.loc[dff1['ALT_U'] == 'CU', 'ALT_Tumor'] = dff1.Tumor_CU
    dff1.loc[dff1['ALT_U'] == 'GU', 'ALT_Tumor'] = dff1.Tumor_GU
    dff1.loc[dff1['ALT_U'] == 'TU', 'ALT_Tumor'] = dff1.Tumor_TU
print(dff1)

for i in dff2['CHROM_POS']:
    dff2.loc[dff2['REF_U'] == 'AU', 'REF_Normal'] = dff2.Normal_AU
    dff2.loc[dff2['REF_U'] == 'CU', 'REF_Normal'] = dff2.Normal_CU
    dff2.loc[dff2['REF_U'] == 'GU', 'REF_Normal'] = dff2.Normal_GU
    dff2.loc[dff2['REF_U'] == 'TU', 'REF_Normal'] = dff2.Normal_TU
    dff2.loc[dff2['ALT_U'] == 'AU', 'ALT_Normal'] = dff2.Normal_AU
    dff2.loc[dff2['ALT_U'] == 'CU', 'ALT_Normal'] = dff2.Normal_CU
    dff2.loc[dff2['ALT_U'] == 'GU', 'ALT_Normal'] = dff2.Normal_GU
    dff2.loc[dff2['ALT_U'] == 'TU', 'ALT_Normal'] = dff2.Normal_TU
    dff2.loc[dff2['REF_U'] == 'AU', 'REF_Tumor'] = dff2.Tumor_AU
    dff2.loc[dff2['REF_U'] == 'CU', 'REF_Tumor'] = dff2.Tumor_CU
    dff2.loc[dff2['REF_U'] == 'GU', 'REF_Tumor'] = dff2.Tumor_GU
    dff2.loc[dff2['REF_U'] == 'TU', 'REF_Tumor'] = dff2.Tumor_TU
    dff2.loc[dff2['ALT_U'] == 'AU', 'ALT_Tumor'] = dff2.Tumor_AU
    dff2.loc[dff2['ALT_U'] == 'CU', 'ALT_Tumor'] = dff2.Tumor_CU
    dff2.loc[dff2['ALT_U'] == 'GU', 'ALT_Tumor'] = dff2.Tumor_GU
    dff2.loc[dff2['ALT_U'] == 'TU', 'ALT_Tumor'] = dff2.Tumor_TU
print(dff2)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
dff[['REF_Normal_First', 'REF_Normal_Second']] =
    dff['REF_Normal'].str.split(',',expand=True)
dff[['ALT_Normal_First', 'ALT_Normal_Second']] =
    dff['ALT_Normal'].str.split(',',expand=True)
```

```
dff[['REF_Tumor_First', 'REF_Tumor_Second']] =
    dff['REF_Tumor'].str.split(',',expand=True)
dff[['ALT_Tumor_First', 'ALT_Tumor_Second']] =
    dff['ALT_Tumor'].str.split(',',expand=True)
print(dff)

dff1[['REF_Normal_First', 'REF_Normal_Second']] =
    dff1['REF_Normal'].str.split(',',expand=True)
dff1[['ALT_Normal_First', 'ALT_Normal_Second']] =
    dff1['ALT_Normal'].str.split(',',expand=True)
dff1[['REF_Tumor_First', 'REF_Tumor_Second']] =
    dff1['REF_Tumor'].str.split(',',expand=True)
dff1[['ALT_Tumor_First', 'ALT_Tumor_Second']] =
    dff1['ALT_Tumor'].str.split(',',expand=True)
print(dff1)

dff2[['REF_Normal_First', 'REF_Normal_Second']] =
    dff2['REF_Normal'].str.split(',',expand=True)
dff2[['ALT_Normal_First', 'ALT_Normal_Second']] =
    dff2['ALT_Normal'].str.split(',',expand=True)
dff2[['REF_Tumor_First', 'REF_Tumor_Second']] =
    dff2['REF_Tumor'].str.split(',',expand=True)
dff2[['ALT_Tumor_First', 'ALT_Tumor_Second']] =
    dff2['ALT_Tumor'].str.split(',',expand=True)
print(dff2)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['REF_U', 'ALT_U', 'Normal_AU', 'Normal_CU',
    'Normal_GU', 'Normal_TU', 'Tumor_AU', 'Tumor_CU', 'Tumor_GU',
    'Tumor_TU', 'REF_Normal_Second', 'ALT_Normal_Second',
    'Normal_AU', 'Normal_CU', 'Normal_GU', 'Normal_TU', 'Tumor_AU',
    'Tumor_CU', 'Tumor_GU', 'Tumor_TU', 'REF_Tumor_Second',
    'ALT_Tumor_Second'], axis=1)
print(dff)

dff1 = dff1.drop(['REF_U', 'ALT_U', 'Normal_AU', 'Normal_CU',
    'Normal_GU', 'Normal_TU', 'Tumor_AU', 'Tumor_CU', 'Tumor_GU',
    'Tumor_TU', 'REF_Normal_Second', 'ALT_Normal_Second',
    'Normal_AU', 'Normal_CU', 'Normal_GU', 'Normal_TU', 'Tumor_AU',
    'Tumor_CU', 'Tumor_GU', 'Tumor_TU', 'REF_Tumor_Second',
```

```
    'ALT_Tumor_Second'], axis=1)
print(dff1)

dff2 = dff2.drop(['REF_U', 'ALT_U', 'Normal_AU', 'Normal_CU',
    'Normal_GU', 'Normal_TU', 'Tumor_AU', 'Tumor_CU', 'Tumor_GU',
    'Tumor_TU', 'REF_Normal_Second', 'ALT_Normal_Second',
    'Normal_AU', 'Normal_CU', 'Normal_GU', 'Normal_TU', 'Tumor_AU',
    'Tumor_CU', 'Tumor_GU', 'Tumor_TU', 'REF_Tumor_Second',
    'ALT_Tumor_Second'], axis=1)
print(dff2)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM_POS', 'REF', 'ALT', 'REF_Normal',
    'ALT_Normal', 'REF_Tumor', 'ALT_Tumor', 'REF_Normal_First',
    'ALT_Normal_First', 'REF_Tumor_First', 'ALT_Tumor_First']
print(dff)

dff1.columns = ['CHROM_POS', 'REF', 'ALT', 'REF_Normal',
    'ALT_Normal', 'REF_Tumor', 'ALT_Tumor', 'REF_Normal_First',
    'ALT_Normal_First', 'REF_Tumor_First', 'ALT_Tumor_First']
print(dff1)

dff2.columns = ['CHROM_POS', 'REF', 'ALT', 'REF_Normal',
    'ALT_Normal', 'REF_Tumor', 'ALT_Tumor', 'REF_Normal_First',
    'ALT_Normal_First', 'REF_Tumor_First', 'ALT_Tumor_First']
print(dff2)

# Converting string values columns to int.
dff['REF_Normal_First'] = dff['REF_Normal_First'].astype(int)
dff['ALT_Normal_First'] = dff['ALT_Normal_First'].astype(int)
dff['REF_Tumor_First'] = dff['REF_Tumor_First'].astype(int)
dff['ALT_Tumor_First'] = dff['ALT_Tumor_First'].astype(int)
print(dff)

dff1['REF_Normal_First'] = dff1['REF_Normal_First'].astype(int)
dff1['ALT_Normal_First'] = dff1['ALT_Normal_First'].astype(int)
dff1['REF_Tumor_First'] = dff1['REF_Tumor_First'].astype(int)
dff1['ALT_Tumor_First'] = dff1['ALT_Tumor_First'].astype(int)
print(dff1)
```

```
dff2['REF_Normal_First'] = dff2['REF_Normal_First'].astype(int)
dff2['ALT_Normal_First'] = dff2['ALT_Normal_First'].astype(int)
dff2['REF_Tumor_First'] = dff2['REF_Tumor_First'].astype(int)
dff2['ALT_Tumor_First'] = dff2['ALT_Tumor_First'].astype(int)
print(dff2)

# Adding the values for formula.
dff['SUM'] = dff["REF_Normal_First"] + dff["ALT_Normal_First"]
dff['COMMON'] = dff["REF_Tumor_First"] + dff["ALT_Tumor_First"]
print(dff)

dff1['SUM'] = dff1["REF_Normal_First"] + dff1["ALT_Normal_First"]
dff1['COMMON'] = dff1["REF_Tumor_First"] + dff1["ALT_Tumor_First"]
print(dff1)

dff2['SUM'] = dff2["REF_Normal_First"] + dff2["ALT_Normal_First"]
dff2['COMMON'] = dff2["REF_Tumor_First"] + dff2["ALT_Tumor_First"]
print(dff2)

# Getting Allele Frequency
dff['Normal'] = dff['ALT_Normal_First']/dff['SUM']
dff['Tumor'] = dff['ALT_Tumor_First']/dff['COMMON']
print(dff)

dff1['Normal'] = dff1['ALT_Normal_First']/dff1['SUM']
dff1['Tumor'] = dff1['ALT_Tumor_First']/dff1['COMMON']
print(dff1)

dff2['Normal'] = dff2['ALT_Normal_First']/dff2['SUM']
dff2['Tumor'] = dff2['ALT_Tumor_First']/dff2['COMMON']
print(dff2)

# Converting string values columans to int.
dff['Normal'] = dff['Normal'].astype(float).round(2)
dff['Tumor'] = dff['Tumor'].astype(float).round(2)
print(dff)

dff1['Normal'] = dff1['Normal'].astype(float).round(2)
dff1['Tumor'] = dff1['Tumor'].astype(float).round(2)
print(dff1)
```

```
dff2['Normal'] = dff2['Normal'].astype(float).round(2)
dff2['Tumor'] = dff2['Tumor'].astype(float).round(2)
print(dff2)

# Concatinating the "CHROM" and "POS"
dff["Normal"] = dff['REF'].astype(str) + ':' +
    dff['Normal'].astype(str)
dff["Tumor"] = dff['REF'].astype(str) + ':' +
    dff['Tumor'].astype(str)
print(dff)

dff1["Normal"] = dff1['REF'].astype(str) + ':' +
    dff1['Normal'].astype(str)
dff1["Tumor"] = dff1['REF'].astype(str) + ':' +
    dff1['Tumor'].astype(str)
print(dff1)

dff2["Normal"] = dff2['REF'].astype(str) + ':' +
    dff2['Normal'].astype(str)
dff2["Tumor"] = dff2['REF'].astype(str) + ':' +
    dff2['Tumor'].astype(str)
print(dff2)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['REF', 'ALT', 'REF_Normal', 'ALT_Normal',
    'REF_Tumor', 'ALT_Tumor', 'REF_Normal_First',
    'ALT_Normal_First', 'REF_Tumor_First', 'ALT_Tumor_First',
    'SUM', 'COMMON'], axis=1)
print(dff)

dff1 = dff1.drop(['REF', 'ALT', 'REF_Normal', 'ALT_Normal',
    'REF_Tumor', 'ALT_Tumor', 'REF_Normal_First',
    'ALT_Normal_First', 'REF_Tumor_First', 'ALT_Tumor_First',
    'SUM', 'COMMON'], axis=1)
print(dff1)

dff2 = dff2.drop(['REF', 'ALT', 'REF_Normal', 'ALT_Normal',
    'REF_Tumor', 'ALT_Tumor', 'REF_Normal_First',
    'ALT_Normal_First', 'REF_Tumor_First', 'ALT_Tumor_First',
```

```
    'SUM', 'COMMON'], axis=1)
print(dff2)

# Merging columns based on "CHROM-POS"
First = pd.merge(dff, dff1, on=['CHROM_POS'])
Third = pd.merge(First, dff2, on=['CHROM_POS'])

# Renaming Columns
Third.columns = ['CHROM_POS', 'Strelka_Normal_0.3',
    'Strelka_Tumor_0.3', 'Strelka_0.5_Normal', 'Strelka_0.5_Tumor',
    'Strelka_0.7_Normal', 'Strelka_0.7_Tumor']
print(Third)

# Assigning column names.
Second.columns = ['CHROM_POS', 'Indel_Normal_0.3',
    'Indel_Tumor_0.3', 'Indel_Normal_0.5', 'Indel_Tumor_0.5',
    'Indel_Normal_0.7', 'Indel_Tumor_0.7']
Third.columns = ['CHROM_POS', 'SNP_Normal_0.3', 'SNP_Tumor_0.3',
    'SNP_Normal_0.5', 'SNP_Tumor_0.5', 'SNP_Normal_0.7',
    'SNP_Tumor_0.7']
print(Second)
print(Third)

# Using merge function by setting how='inner'
df = pd.merge(Second, Third, on='CHROM_POS', how='outer')
df['Normal_0.3_AF'] =
    df['Indel_Normal_0.3'].combine_first(df['SNP_Normal_0.3'])
df['Tumor_0.3_AF'] =
    df['Indel_Tumor_0.3'].combine_first(df['SNP_Tumor_0.3'])
df['Normal_0.5_AF'] =
    df['Indel_Normal_0.5'].combine_first(df['SNP_Normal_0.5'])
df['Tumor_0.5_AF'] =
    df['Indel_Tumor_0.5'].combine_first(df['SNP_Tumor_0.5'])
df['Normal_0.7_AF'] =
    df['Indel_Normal_0.7'].combine_first(df['SNP_Normal_0.7'])
df['Tumor_0.7_AF'] =
    df['Indel_Tumor_0.7'].combine_first(df['SNP_Tumor_0.7'])
print(df)

# Dropping unneeded columns.
```

```
df = df.drop(['Indel_Normal_0.3', 'Indel_Tumor_0.3',
    'Indel_Normal_0.5', 'Indel_Tumor_0.5', 'Indel_Normal_0.7',
    'Indel_Tumor_0.7', 'SNP_Normal_0.3', 'SNP_Tumor_0.3',
    'SNP_Normal_0.5', 'SNP_Tumor_0.5', 'SNP_Normal_0.7',
    'SNP_Tumor_0.7'], axis=1)
print(df)

# Saving the result into a csv file for plotting.
df.to_csv('Strelka_Allele_Frequency.csv', sep=',', index = False)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "GT:GQ:DP:AD:ADF:ADR".
df[['Normal_0.3_Allele', 'Normal_0.3_Value']] =
    df['Normal_0.3_AF'].str.split(':',expand=True)
df[['Tumor_0.3_Allele', 'Tumor_0.3_Value']] =
    df['Tumor_0.3_AF'].str.split(':',expand=True)
df[['Normal_0.5_Allele', 'Normal_0.5_Value']] =
    df['Normal_0.5_AF'].str.split(':',expand=True)
df[['Tumor_0.5_Allele', 'Tumor_0.5_Value']] =
    df['Tumor_0.5_AF'].str.split(':',expand=True)
df[['Normal_0.7_Allele', 'Normal_0.7_Value']] =
    df['Normal_0.7_AF'].str.split(':',expand=True)
df[['Tumor_0.7_Allele', 'Tumor_0.7_Value']] =
    df['Tumor_0.7_AF'].str.split(':',expand=True)
print(df)

# Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff = df.drop(['CHROM_POS', 'Normal_0.3_AF', 'Tumor_0.3_AF',
    'Normal_0.5_AF', 'Tumor_0.5_AF', 'Normal_0.7_AF',
    'Tumor_0.7_AF', 'Normal_0.3_Allele', 'Tumor_0.3_Allele',
    'Normal_0.5_Allele', 'Tumor_0.5_Allele', 'Normal_0.7_Allele',
    'Tumor_0.7_Allele'], axis=1)

# Renaming the columns.
dff.columns = ['Normal_0.3', 'Tumor_0.3', 'Normal_0.5',
    'Tumor_0.5', 'Normal_0.7', 'Tumor_0.7']
print(dff)
```

```
# Converting string values columns to float.
dff['Normal_0.3'] = dff['Normal_0.3'].astype(float)
dff['Tumor_0.3'] = dff['Tumor_0.3'].astype(float)
dff['Normal_0.5'] = dff['Normal_0.5'].astype(float)
dff['Tumor_0.5'] = dff['Tumor_0.5'].astype(float)
dff['Normal_0.7'] = dff['Normal_0.7'].astype(float)
dff['Tumor_0.7'] = dff['Tumor_0.7'].astype(float)
print(dff)

# Getting a count based on allele frequency values.
dff1 = dff[dff < 0.26].count()
dff2 = dff[dff < 0.51].count()
dff3 = dff[dff < 0.76].count()
dff4 = dff[dff < 1.01].count()

# Getting the final values
dff5 = (dff1 - dff2).abs()
dff6 = (dff2 - dff3).abs()
dff7 = (dff3 - dff4).abs()
print(dff1)
print(dff5)
print(dff6)
print(dff7)

# Converting into list.
First_Column = dff1.tolist()
Second_Column = dff5.tolist()
Third_Column = dff6.tolist()
Fourth_Column = dff7.tolist()
Fifth_Column =['Normal_0.3', 'Tumor_0.3', 'Normal_0.5',
    'Tumor_0.5', 'Normal_0.7', 'Tumor_0.7']
print(First_Column)
print(Second_Column)
print(Third_Column)
print(Fourth_Column)

# Declaring new columns.
dff8 = pd.DataFrame(Fifth_Column, columns = ['Type'])
dff8['Less than 0.25'] = First_Column
```

```
dff8['Between 0.25 & 0.50'] = Second_Column
dff8['Between 0.50 & 0.75'] = Third_Column
dff8['Between 0.75 & 1.00'] = Fourth_Column
print(dff8)

# Saving the results in csv.
dff8.to_csv('Strelka_Allele_Frequency_Counts.csv', sep=',', index
    = None)

dff9 = dff8.drop(['Type'], axis=1)
print(dff9)

# Converting the values to a list
List = dff9.values.tolist()
a1, a2, a3, a4, a5, a6 = List
print(a1)
print(a2)
print(a3)
print(a4)
print(a5)

# set width of bar
width = 0.10

# Columns from the file
# a1 = First_Column
# a2 = Second_Column
# a3 = Third_Column
# a4 = Fourth_Column

# Set position of bar on X axis
r1 = np.arange(len(a1))
r2 = [x + width for x in r1]
r3 = [x + width for x in r2]
r4 = [x + width for x in r3]
r5 = [x + width for x in r4]
r6 = [x + width for x in r5]

# Make the plot
```

```
plt.bar(r1, a1, color='#ff0000', width=width, edgecolor='white',
    label='Normal_0.3')
plt.bar(r2, a2, color='#ffa07a', width=width, edgecolor='white',
    label='Tumor_0.3')
plt.bar(r3, a3, color='#f08080', width=width, edgecolor='white',
    label='Normal_0.5')
plt.bar(r4, a4, color='#fa8072', width=width, edgecolor='white',
    label='Tumor_0.5')
plt.bar(r5, a5, color='#b22222', width=width, edgecolor='white',
    label='Normal_0.7')
plt.bar(r6, a6, color='#800000', width=width, edgecolor='white',
    label='Tumor_0.7')

csfont = {'fontname':'Comic Sans MS'}
hfont = {'fontname':'Helvetica'}

# Add xticks on the middle of the group bars
plt.xlabel('Strelka_Allele_Frequencies')
plt.xticks([r + width for r in range(len(a1))], ['<= 0.25', '<=
    0.50', '<= 0.75', '<= 1.00'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('Strelka_Allele_Frequency_Plot.pdf')
plt.savefig('Strelka_Allele_Frequency_Plot.png', dpi = 300)
```

8.1.1.2 Truth Data Allele Frequency Code

```
# Importing packages.
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import csv

# Reading csv files and concatinating "CHROM" and "POS"
dff = pd.read_csv("Somatic_Truth.frq", sep = '\t', index_col=
    False, error_bad_lines=False)
```

```
dff.columns = ['CHROM', 'POS', 'N_ALLELES', 'N_CHR', 'ALLELE:FREQ']
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)

# Reorganising columns.
dff = dff.drop(['N_ALLELES', 'N_CHR', 'CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

# Saving the results in csv.
dff.to_csv('Truth_Data.csv', sep=',', index = False)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "GT:GQ:DP:AD:ADF:ADR".
dff[['Allele', 'Freq']] =
    dff['ALLELE:FREQ'].str.split(':',expand=True)
print(dff)

#  Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff = dff.drop(['CHROM_POS', 'ALLELE:FREQ', 'Allele'], axis=1)
print(dff)

# Renaming the columns.
dff.columns = ['Freq']
print(dff)

# Converting string values columns to float.
dff['Freq'] = dff['Freq'].astype(float)
print(dff)

# Getting a count based on allele frequency values.
dff1 = dff[dff < 0.26].count()
dff2 = dff[dff < 0.51].count()
dff3 = dff[dff < 0.76].count()
dff4 = dff[dff < 1.01].count()
```

```
# Getting the final values
dff5 = (dff1 - dff2).abs()
dff6 = (dff2 - dff3).abs()
dff7 = (dff3 - dff4).abs()
print(dff1)
print(dff5)
print(dff6)
print(dff7)

# Converting into list.
First_Column = dff1.tolist()
Second_Column = dff5.tolist()
Third_Column = dff6.tolist()
Fourth_Column = dff7.tolist()
Fifth_Column =['Truth_Data']
print(First_Column)
print(Second_Column)
print(Third_Column)
print(Fourth_Column)

# Declaring new columns.
dff8 = pd.DataFrame(Fifth_Column, columns = ['Type'])
dff8['Less than 0.25'] = First_Column
dff8['Between 0.25 & 0.50'] = Second_Column
dff8['Between 0.50 & 0.75'] = Third_Column
dff8['Between 0.75 & 1.00'] = Fourth_Column
print(dff8)

# Saving the results in csv.
dff8.to_csv('Truth_Data_Allele_Frequency_Counts.csv', sep=',',
    index = None)
```

8.1.1.3 VarScan Allele Frequency Code

```
# Importing packages.
import numpy as np
import pandas as pd
import matplotlib
```

```
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import csv

# Reading csv files and concatinating "CHROM" and "POS"
dff = pd.read_csv("VarScan_0.3.frq", sep = '\t', index_col= False)
dff.columns = ['CHROM', 'POS', 'N_ALLELES', 'N_CHR', 'ALLELE:FREQ']
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)

dff2 = pd.read_csv("VarScan_0.5.frq", sep = '\t', index_col= False)
dff2.columns = ['CHROM', 'POS', 'N_ALLELES', 'N_CHR',
    'ALLELE:FREQ']
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)

dff3 = pd.read_csv("VarScan_0.7.frq", sep = '\t', index_col= False)
dff3.columns = ['CHROM', 'POS', 'N_ALLELES', 'N_CHR',
    'ALLELE:FREQ']
dff3["CHROM_POS"] = dff3['CHROM'].astype(str) + '-' +
    dff3['POS'].astype(str)

# Reorganising columns.
dff = dff.drop(['N_ALLELES', 'N_CHR', 'CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

dff2 = dff2.drop(['N_ALLELES', 'N_CHR', 'CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
print(dff2)

dff3 = dff3.drop(['N_ALLELES', 'N_CHR', 'CHROM', 'POS'], axis=1)
cols = dff3.columns.tolist()
cols = cols[-1:] + cols[:-1]
```

```
dff3 = dff3[cols]
print(dff3)

# Merging columns based on "CHROM_POS"
Result = pd.merge(dff, dff2, on="CHROM_POS")
Merge = pd.merge(Result, dff3, on="CHROM_POS")
Merge.columns = ['CHROM_POS', 'VarScan_0.3_AF', 'VarScan_0.5_AF',
    'VarScan_0.7_AF']

# Saving the results in csv.
Merge.to_csv('VarScan_Allele_Frequencies.csv', sep=',', index =
    False)

# Creating new columns by splitting the "Allele" and "Value" by
    ':'.
Merge[['VarScan_0.3_Allele', 'VarScan_0.3_Value']] =
    Merge['VarScan_0.3_AF'].str.split(':',expand=True)
Merge[['VarScan_0.5_Allele', 'VarScan_0.5_Value']] =
    Merge['VarScan_0.5_AF'].str.split(':',expand=True)
Merge[['VarScan_0.7_Allele', 'VarScan_0.7_Value']] =
    Merge['VarScan_0.7_AF'].str.split(':',expand=True)
print(Merge)

# Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff4 = Merge.drop(['CHROM_POS', 'VarScan_0.3_AF',
    'VarScan_0.5_AF', 'VarScan_0.7_AF', 'VarScan_0.3_Allele',
    'VarScan_0.5_Allele', 'VarScan_0.7_Allele'], axis=1)

# Renaming the columns.
dff4.columns = ['VarScan_0.3', 'VarScan_0.5', 'VarScan_0.7']
print(dff)

# Converting string values columns to float.
dff4['VarScan_0.3'] = dff4['VarScan_0.3'].astype(float)
dff4['VarScan_0.5'] = dff4['VarScan_0.5'].astype(float)
dff4['VarScan_0.7'] = dff4['VarScan_0.7'].astype(float)
print(dff4)
```

```
# Getting a count based on allele frequency values.
dff5 = dff4[dff4 < 0.26].count()
dff6 = dff4[dff4 < 0.51].count()
dff7 = dff4[dff4 < 0.76].count()
dff8 = dff4[dff4 < 1.01].count()

# Getting the final values
dff9 = (dff5 - dff6).abs()
dff10 = (dff6 - dff7).abs()
dff11 = (dff7 - dff8).abs()
print(dff5)
print(dff9)
print(dff10)
print(dff11)

# Converting into list.
First_Column = dff5.tolist()
Second_Column = dff9.tolist()
Third_Column = dff10.tolist()
Fourth_Column = dff11.tolist()
Fifth_Column =['VarScan_0.3', 'VarScan_0.5', 'VarScan_0.7']
print(First_Column)
print(Second_Column)
print(Third_Column)
print(Fourth_Column)

# Declaring new columns.
dff8 = pd.DataFrame(Fifth_Column, columns = ['Type'])
dff8['Less than 0.25'] = First_Column
dff8['Between 0.25 & 0.50'] = Second_Column
dff8['Between 0.50 & 0.75'] = Third_Column
dff8['Between 0.75 & 1.00'] = Fourth_Column
print(dff8)

# Saving the results in csv.
dff8.to_csv('VarScan_Allele_Frequency_Counts.csv', sep=',', index
    = None)

dff9 = dff8.drop(['Type'], axis=1)
print(dff9)
```

```
# Converting the values to a list
List = dff9.values.tolist()
a1, a2, a3 = List
print(a1)
print(a2)
print(a3)

# set width of bar
width = 0.15

# Set position of bar on X axis
r1 = np.arange(len(a1))
r2 = [x + width for x in r1]
r3 = [x + width for x in r2]

# Make the plot
plt.bar(r1, a1, color='#ff0000', width=width, edgecolor='white',
    label='VarScan_0.3')
plt.bar(r2, a2, color='#ffa07a', width=width, edgecolor='white',
    label='VarScan_0.5')
plt.bar(r3, a3, color='#f08080', width=width, edgecolor='white',
    label='VarScan_0.7')

# Add xticks on the middle of the group bars
plt.xlabel('VarScan_Allele_Frequencies')
plt.xticks([r + width for r in range(len(a1))], ['<= 0.25', '<=
    0.50', '<= 0.75', '<= 1.00'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('VarScan_Allele_Frequency_Plot.pdf')
plt.savefig('VarScan_Allele_Frequency_Plot.png', dpi = 300)
```

8.1.2 Allele Frequency Comparison

```
# Importing packages.
```

```
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import csv

# Reading csv files and concatinating "CHROM" and "POS"
df = pd.read_csv("Strelka_0.3.csv", sep = '\t', index_col= False)
df1 = pd.read_csv("VarScan_0.3.csv", sep = '\t', index_col= False)
df2 = pd.read_csv("Somatic_Truth.csv", sep = '\t', index_col=
    False)

# Merging columns based on "CHROM-POS"
First = pd.merge(df, df1, on=['CHROM_POS'])
Second = pd.merge(First, df2, on=['CHROM_POS'])

# Renaming Columns
Second.columns = ['CHROM_POS', 'Strelka_Normal', 'Strelka_Tumor',
    'VarScan', 'Truth_Data']
print(Second)

# Saving the results in csv.
Second.to_csv('Tumor_Purity_0.3.csv', sep=',', index = None)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "GT:GQ:DP:AD:ADF:ADR".
Second[['Strelka_Normal_Allele', 'Strelka_Normal_Value']] =
    Second['Strelka_Normal'].str.split(':',expand=True)
Second[['Strelka_Tumor_Allele', 'Strelka_Tumor_Value']] =
    Second['Strelka_Tumor'].str.split(':',expand=True)
Second[['VarScan_Normal_Allele', 'VarScan_Normal_Value']] =
    Second['VarScan'].str.split(':',expand=True)
Second[['Truth_Data_Allele', 'Truth_Data_Value']] =
    Second['Truth_Data'].str.split(':',expand=True)
print(Second)
```

```
# Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff = Second.drop(['CHROM_POS', 'Strelka_Normal', 'Strelka_Tumor',
    'VarScan', 'Truth_Data', 'Strelka_Normal_Allele',
    'Strelka_Tumor_Allele', 'VarScan_Normal_Allele',
    'Truth_Data_Allele'], axis=1)
print(dff)

# Renaming the columns.
dff.columns = ['Strelka_Normal', 'Strelka_Tumor', 'VarScan',
    'Truth_Data']
print(dff)

# Converting string values columns to float.
dff['Strelka_Normal'] = dff['Strelka_Normal'].astype(float)
dff['Strelka_Tumor'] = dff['Strelka_Tumor'].astype(float)
dff['VarScan'] = dff['VarScan'].astype(float)
dff['Truth_Data'] = dff['Truth_Data'].astype(float)
print(dff)

# Getting a count based on allele frequency values.
dff1 = dff[dff < 0.26].count()
dff2 = dff[dff < 0.51].count()
dff3 = dff[dff < 0.76].count()
dff4 = dff[dff < 1.01].count()

# Getting the final values
dff5 = (dff1 - dff2).abs()
dff6 = (dff2 - dff3).abs()
dff7 = (dff3 - dff4).abs()
print(dff1)
print(dff5)
print(dff6)
print(dff7)

# Converting into list.
First_Column = dff1.tolist()
Second_Column = dff5.tolist()
Third_Column = dff6.tolist()
```

```
Fourth_Column = dff7.tolist()
Fifth_Column =['Strelka_Normal', 'Strelka_Tumor', 'VarScan',
    'Truth_Data']
print(First_Column)
print(Second_Column)
print(Third_Column)
print(Fourth_Column)

# Declaring new columns.
dff8 = pd.DataFrame(Fifth_Column, columns = ['Type'])
dff8['Less than 0.25'] = First_Column
dff8['Between 0.25 & 0.50'] = Second_Column
dff8['Between 0.50 & 0.75'] = Third_Column
dff8['Between 0.75 & 1.00'] = Fourth_Column
print(dff8)

# Saving the results in csv.
dff8.to_csv('Tumor_Purity_0.3_AF_Counts.csv', sep=',', index =
    None)

# set width of bar
width = 0.25

# Columns from the file
a1 = First_Column
a2 = Second_Column
a3 = Third_Column
a4 = Fourth_Column

# Set position of bar on X axis
r1 = np.arange(len(a1))
r2 = [x + width for x in r1]
r3 = [x + width for x in r2]
r4 = [x + width for x in r3]

# Make the plot
plt.bar(r1, a1, color='#FFD700', width=width, edgecolor='white',
    label='Strelka_Normal')
plt.bar(r2, a2, color='#FFAA1C', width=width, edgecolor='white',
    label='Strelka_Tumor')
```

```
plt.bar(r3, a3, color='#FF8C01', width=width, edgecolor='white',
    label='VarScan')
plt.bar(r4, a4, color='#FF0000', width=width, edgecolor='white',
    label='Truth_Data')

# Add xticks on the middle of the group bars
plt.xlabel('Tumor_0.3_Allele_Frequencies')
plt.xticks([r + width for r in range(len(a1))], ['<= 0.25', '<=
    0.50', '<= 0.75', '<= 1.00'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('Tumor_Purity_0.3_AF_Plot.pdf')
plt.savefig('Tumor_Purity_0.3_AF_Plot.png', dpi = 300)
```

8.2 Benchmarking

8.3 Positions, SNPs, Indels

8.3.1 Strelka

```
# Importing packages.
import numpy as np
import pandas as pd

# Inputing vcf files for positions.
df = pd.read_csv("Updated_Strelka_0.3.vcf", sep = '\t', index_col=
    False)
df1 = pd.read_csv("Updated_Strelka_0.5.vcf", sep = '\t',
    index_col= False)
df2 = pd.read_csv("Updated_Strelka_0.7.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for SNPs.
df3 = pd.read_csv("Updated_Strelka_0.3_SNPs.vcf", sep = '\t',
    index_col= False)
```

```
df4 = pd.read_csv("Updated_Strelka_0.5_SNPs.vcf", sep = '\t',
    index_col= False)
df5 = pd.read_csv("Updated_Strelka_0.7_SNPs.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for Indels.
df6 = pd.read_csv("Updated_Strelka_0.3_Indels.vcf", sep = '\t',
    index_col= False)
df7 = pd.read_csv("Updated_Strelka_0.5_Indels.vcf", sep = '\t',
    index_col= False)
df8 = pd.read_csv("Updated_Strelka_0.7_Indels.vcf", sep = '\t',
    index_col= False)

# Outcome for positions.
Strelka3_Positions = len(df)
Strelka5_Positions = len(df1)
Strelka7_Positions = len(df2)

print("Number of positions in Strelka 0.3:")
print(Strelka3_Positions)
print("Number of positions in Strelka 0.5:")
print(Strelka5_Positions)
print("Number of positions in Strelka 0.7:")
print(Strelka7_Positions)

# Outcome for SNPs.
Strelka3_SNPs = len(df3)
Strelka5_SNPs = len(df4)
Strelka7_SNPs = len(df5)

print("Number of SNPs in Strelka 0.3:")
print(Strelka3_SNPs)
print("Number of SNPs in Strelka 0.5:")
print(Strelka5_SNPs)
print("Number of SNPs in Strelka 0.7:")
print(Strelka7_SNPs)

# Outcome for SNPs.
Strelka3_Indels = len(df6)
Strelka5_Indels = len(df7)
```

```
Strelka7_Indels = len(df8)

print("Number of Indels in Strelka 0.3:")
print(Strelka3_Indels)
print("Number of Indels in Strelka 0.5:")
print(Strelka5_Indels)
print("Number of Indels in Strelka 0.7:")
print(Strelka7_Indels)

# Delcaring a new dataframe.
df = []

# Taking all combinations as a list.
data = {'Type': ['Strelka_Tumor_Purity_0.3',
    'Strelka_Tumor_Purity_0.5', 'Strelka_Tumor_Purity_0.7'],
    'Strelka_Positions': [Strelka3_Positions, Strelka5_Positions,
    Strelka7_Positions], 'Strelka_SNPs': [Strelka3_SNPs,
    Strelka5_SNPs, Strelka7_SNPs], 'Strelka_INDELs':
    [Strelka3_Indels, Strelka5_Indels, Strelka7_Indels]}

# Collecting it into a dataframe.
df = pd.DataFrame(data)
print(df)

# Saving the results in csv.
df.to_csv('Strelka_Counts.csv', sep=',', index = None)
```

8.3.2 Truth Data

```
# Importing packages.
import numpy as np
import pandas as pd

# Inputing vcf files for positions.
df1 = pd.read_csv("Updated_Somatic_Truth.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for SNPs.
```

```
df2 = pd.read_csv("Updated_Somatic_Truth_SNPs.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for Indels.
df3 = pd.read_csv("Updated_Somatic_Truth_Indels.vcf", sep = '\t',
    index_col= False)

# Outcome for positions.
Truth_Positions = len(df1)

print("Number of positions in Somatic Truth:")
print(Truth_Positions)

# Outcome for SNPs.
Truth_SNPs = len(df2)

print("Number of SNPs in Somatic Truth:")
print(Truth_SNPs)

# Outcome for SNPs.
Truth_Indels = len(df3)

print("Number of Indels in Somatic Truth:")
print(Truth_Indels)

# Delcaring a new dataframe.
df = []

# Taking all combinations as a list.
data = {'Type': ['Somatic_Truth'], 'Truth_Positions':
    [Truth_Positions], 'Truth_SNPs': [Truth_SNPs], 'Truth_INDELs':
    [Truth_Indels]}

# Collecting it into a dataframe.
df = pd.DataFrame(data)
print(df)

# Saving the results in csv.
df.to_csv('Truth_Counts.csv', sep=',', index = None)
```

8.3.3 VarScan

```
# Importing packages.
import numpy as np
import pandas as pd

# Inputing vcf files for positions.
df = pd.read_csv("Updated_VarScan_0.3.vcf", sep = '\t', index_col=
    False)
df1 = pd.read_csv("Updated_VarScan_0.5.vcf", sep = '\t',
    index_col= False)
df2 = pd.read_csv("Updated_VarScan_0.7.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for SNPs.
df3 = pd.read_csv("Updated_VarScan_0.3_SNPs.vcf", sep = '\t',
    index_col= False)
df4 = pd.read_csv("Updated_VarScan_0.5_SNPs.vcf", sep = '\t',
    index_col= False)
df5 = pd.read_csv("Updated_VarScan_0.7_SNPs.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for Indels.
df6 = pd.read_csv("Updated_VarScan_0.3_Indels.vcf", sep = '\t',
    index_col= False)
df7 = pd.read_csv("Updated_VarScan_0.5_Indels.vcf", sep = '\t',
    index_col= False)
df8 = pd.read_csv("Updated_VarScan_0.7_Indels.vcf", sep = '\t',
    index_col= False)

# Outcome for positions.
VarScan3_Positions = len(df)
VarScan5_Positions = len(df1)
VarScan7_Positions = len(df2)

print("Number of positions in VarScan 0.3:")
print(VarScan3_Positions)
print("Number of positions in VarScan 0.5:")
print(VarScan5_Positions)
print("Number of positions in VarScan 0.7:")
```

```
print(VarScan7_Positions)

# Outcome for SNPs.
VarScan3_SNPs = len(df3)
VarScan5_SNPs = len(df4)
VarScan7_SNPs = len(df5)

print("Number of SNPs in VarScan 0.3:")
print(VarScan3_SNPs)
print("Number of SNPs in VarScan 0.5:")
print(VarScan5_SNPs)
print("Number of SNPs in VarScan 0.7:")
print(VarScan7_SNPs)

# Outcome for SNPs.
VarScan3_Indels = len(df6)
VarScan5_Indels = len(df7)
VarScan7_Indels = len(df8)

print("Number of Indels in VarScan 0.3:")
print(VarScan3_Indels)
print("Number of Indels in VarScan 0.5:")
print(VarScan5_Indels)
print("Number of Indels in VarScan 0.7:")
print(VarScan7_Indels)

# Delcaring a new dataframe.
df = []

# Taking all combinations as a list.
data = {'Type': ['VarScan_Tumor_Purity_0.3',
    'VarScan_Tumor_Purity_0.5', 'VarScan_Tumor_Purity_0.7'],
    'VarScan_Positions': [VarScan3_Positions, VarScan5_Positions,
    VarScan7_Positions], 'VarScan_SNPs': [VarScan3_SNPs,
    VarScan5_SNPs, VarScan7_SNPs], 'VarScan_INDELs':
    [VarScan3_Indels, VarScan5_Indels, VarScan7_Indels]}

# Collecting it into a dataframe.
df = pd.DataFrame(data)
print(df)
```

```
# Saving the results in csv.
df.to_csv('VarScan_Counts.csv', sep=',', index = None)
```

8.3.4 Comparison

8.3.4.1 Positions Comparison

```
# Importing packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
from matplotlib_venn import venn3_circles, venn3_unweighted
from matplotlib_venn import _common, _venn3
from matplotlib.patches import Circle
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'

# Reading csv files and concatinating "CHROM" and "POS"
df = pd.read_csv("Updated_Strelka_0.3.vcf", sep = '\t', index_col=
    False)
df1 = pd.read_csv("Updated_VarScan_0.3.vcf", sep = '\t',
    index_col= False)
df2 = pd.read_csv("Updated_Somatic_Truth.vcf", sep = '\t',
    index_col= False)

# Merging columns based on "POS"
First = pd.merge(df, df1, on=['POS'])
Second = pd.merge(df1, df2, on=['POS'])
Third = pd.merge(df, df2, on=['POS'])
Fourth = pd.merge(First, df2, on=['POS'])

# Position outcomes.
print("Number of Strelka Positions:")
Strelka_Positions = len(df)
print(Strelka_Positions)
```

```
print("Number of VarScan Positions:")
VarScan_Positions = len(df1)
print(VarScan_Positions)

print("Number of Truth Data Positions:")
Truth_Positions = len(df2)
print(Truth_Positions)

print("Number of Positions in Strelka and VarScan:")
Strelka_VarScan_Positions = len(First)
print(Strelka_VarScan_Positions)

print("Number of Positions in VarScan and Truth Data:")
VarScan_Truth_Positions = len(Second)
print(VarScan_Truth_Positions)

print("Number of Positions in Truth Data and Strelka:")
Strelka_Truth_Positions = len(Third)
print(Strelka_Truth_Positions)

print("Number of Positions in Strelka, VarScan & Truth Data:")
Strelka_VarScan_Truth_Positions = len(Fourth)
print(Strelka_VarScan_Truth_Positions)

# Delcaring a new dataframe.
df3 = []

# Taking all combinations as a list.
data = {'Type': ['Strelka', 'VarScan', 'Truth_Data',
    'Strelka_and_VarScan', 'VarScan_and_Truth_Data',
    'Truth_Data_and_Strelka',
    'Strelka_and_VarScan_and_Truth_Data'], 'Positions':
    [Strelka_Positions, VarScan_Positions, Truth_Positions,
    Strelka_VarScan_Positions, VarScan_Truth_Positions,
    Strelka_Truth_Positions, Strelka_VarScan_Truth_Positions]}

# Collecting it into a dataframe.
df3 = pd.DataFrame(data)
print(df3)
```

```
# Saving the results in csv.
df3.to_csv('Positions_Count.csv', sep=',', index = None)

# Formulas
Strelka_Exclude = Strelka_Positions - (Strelka_VarScan_Positions +
    Strelka_Truth_Positions + Strelka_VarScan_Truth_Positions)
VarScan_Exclude = VarScan_Positions - (Strelka_VarScan_Positions +
    VarScan_Truth_Positions + Strelka_VarScan_Truth_Positions)
Truth_Exclude = Truth_Positions - (VarScan_Truth_Positions +
    Strelka_Truth_Positions + Strelka_VarScan_Truth_Positions)

# Set of values.
subsets = (Strelka_Exclude, VarScan_Exclude,
    Strelka_VarScan_Positions, Truth_Exclude,
    Strelka_Truth_Positions, VarScan_Truth_Positions,
    Strelka_VarScan_Truth_Positions)

# Adding venn diagram.
v = venn3_unweighted(subsets, set_labels = ('Strelka_0.3',
    'VarScan_0.3', 'Truth_Data'), set_colors=('red', 'orange',
    'skyblue'))
areas = (1, 1, 1, 1, 1, 1, 1)
centers, radii = _venn3.solve_venn3_circles(areas)

# Saving the values.
plt.title("Positions [SNPs + Indels]")
plt.show()
plt.savefig('Tumor_Purity_0.3_Positions_Plot.pdf')
plt.savefig('Tumor_Purity_0.3_Positions_Plot.png', dpi = 300)
```

8.3.4.2 SNPs Comparison

```
# Importing packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
```

```
from matplotlib import pyplot as plt
from matplotlib_venn import venn3_circles, venn3_unweighted
from matplotlib_venn import _common, _venn3
from matplotlib.patches import Circle

# Reading csv files and concatinating "CHROM" and "POS"
df = pd.read_csv("Updated_Strelka_0.3_SNPs.vcf", sep = '\t',
    index_col= False)
df1 = pd.read_csv("Updated_VarScan_0.3_SNPs.vcf", sep = '\t',
    index_col= False)
df2 = pd.read_csv("Updated_Somatic_Truth_SNVs.vcf", sep = '\t',
    index_col= False)
print("Length of Strelka SNPs:")
Strelka_SNPs = len(df)
print(Strelka_SNPs)
print("Length of VarScan SNPs:")
VarScan_SNPs = len(df1)
print(VarScan_SNPs)
print("Length of Truth SNPs:")
Truth_SNPs = len(df2)
print(Truth_SNPs)

# Adding REF and ALT
df["REF_ALT"] = df["REF"] + df["ALT"]
df1["REF_ALT"] = df1["REF"] + df1["ALT"]
df2["REF_ALT"] = df2["REF"] + df2["ALT"]

# Merging columns based on "POS"
First = pd.merge(df, df1, on=['POS'])
Second = pd.merge(df1, df2, on=['POS'])
Third = pd.merge(df, df2, on=['POS'])
Fourth = pd.merge(First, df2, on=['POS'])

# Dropping of the unnecessary columns.
First = First.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y'], axis=1)
Second = Second.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y'], axis=1)
Third = Third.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y'], axis=1)
Fourth = Fourth.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y', 'REF',
    'ALT'], axis=1)
print(First)
```

```
print(Second)
print(Third)
print(Fourth)

# Conditional replacement.
First['Result'] = np.where(First["REF_ALT_x"] ==
    First["REF_ALT_y"], 0, 1)
First = First[First["Result"] == 0]
print(First)

Second['Result'] = np.where(Second["REF_ALT_x"] ==
    Second["REF_ALT_y"], 0, 1)
Second = Second[Second["Result"] == 0]
print(Second)

Third['Result'] = np.where(Third["REF_ALT_x"] ==
    Third["REF_ALT_y"], 0, 1)
Third = Third[Third["Result"] == 0]
print(Third)

Fourth['Result'] = np.where(((Fourth["REF_ALT_x"] ==
    Fourth["REF_ALT_y"]) & (Fourth["REF_ALT_x"] ==
    Fourth["REF_ALT"]) & (Fourth["REF_ALT_y"] ==
    Fourth["REF_ALT"])), 0, 1)
Fourth = Fourth[Fourth["Result"] == 0]
print(Fourth)

# Position outcomes.
print("Number of SNPs in Strelka and VarScan:")
Strelka_VarScan_SNPs = len(First)
print(Strelka_VarScan_SNPs)

print("Number of SNPs in VarScan and Truth Data:")
VarScan_Truth_SNPs = len(Second)
print(VarScan_Truth_SNPs)

print("Number of SNPs in Truth Data and Strelka:")
Strelka_Truth_SNPs = len(Third)
print(Strelka_Truth_SNPs)
```

```
print("Number of SNPs in Strelka, VarScan & Truth Data:")
Strelka_VarScan_Truth_SNPs = len(Fourth)
print(Strelka_VarScan_Truth_SNPs)

# Delcaring a new dataframe.
df3 = []

# Taking all combinations as a list.
data = {'Type': ['Strelka', 'VarScan', 'Truth_Data',
    'Strelka_and_VarScan', 'VarScan_and_Truth_Data',
    'Truth_Data_and_Strelka',
    'Strelka_and_VarScan_and_Truth_Data'], 'SNPs': [Strelka_SNPs,
    VarScan_SNPs, Truth_SNPs, Strelka_VarScan_SNPs,
    VarScan_Truth_SNPs, Strelka_Truth_SNPs,
    Strelka_VarScan_Truth_SNPs]}

# Collecting it into a dataframe.
df3 = pd.DataFrame(data)
print(df3)

# Saving the results in csv.
df3.to_csv('SNPs_Count.csv', sep=',', index = None)

# Formulas
Strelka_Exclude = Strelka_SNPs - Strelka_VarScan_SNPs -
    Strelka_Truth_SNPs - Strelka_VarScan_Truth_SNPs
VarScan_Exclude = VarScan_SNPs - Strelka_VarScan_SNPs -
    VarScan_Truth_SNPs - Strelka_VarScan_Truth_SNPs
Truth_Exclude = Truth_SNPs - VarScan_Truth_SNPs -
    Strelka_Truth_SNPs - Strelka_VarScan_Truth_SNPs

# Set of values.
subsets = (Strelka_Exclude, VarScan_Exclude, Strelka_VarScan_SNPs,
    Truth_Exclude, Strelka_Truth_SNPs, VarScan_Truth_SNPs,
    Strelka_VarScan_Truth_SNPs)

# Adding venn diagram.
v = venn3_unweighted(subsets, set_labels = ('Strelka_0.3',
    'VarScan_0.3', 'Truth_Data'), set_colors=('red', 'orange',
    'skyblue'))
```

```
areas = (1, 1, 1, 1, 1, 1, 1)
centers, radii = _venn3.solve_venn3_circles(areas)

# Saving the values.
plt.title("SNPs")
plt.show()
plt.savefig('Tumor_Purity_0.3_SNPs_Plot.pdf')
plt.savefig('Tumor_Purity_0.3_SNPs_Plot.png', dpi = 300)
```

8.3.4.3 Indels Comparison

```
# Importing packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
from matplotlib import pyplot as plt
from matplotlib_venn import venn3_circles, venn3_unweighted
from matplotlib_venn import _common, _venn3
from matplotlib.patches import Circle

# Reading csv files and concatinating "CHROM" and "POS"
df = pd.read_csv("Updated_Strelka_0.3_Indels.vcf", sep = '\t',
    index_col= False)
df1 = pd.read_csv("Updated_VarScan_0.3_Indels.vcf", sep = '\t',
    index_col= False)
df2 = pd.read_csv("Updated_Somatic_Truth_Indels.vcf", sep = '\t',
    index_col= False)
print("Length of Strelka Indels:")
Strelka_Indels = len(df)
print(Strelka_Indels)
print("Length of VarScan Indels:")
VarScan_Indels = len(df1)
print(VarScan_Indels)
print("Length of Truth Indels:")
Truth_Indels = len(df2)
print(Truth_Indels)
```

```
# Adding REF and ALT
df["REF_ALT"] = df["REF"] + df["ALT"]
df1["REF_ALT"] = df1["REF"] + df1["ALT"]
df2["REF_ALT"] = df2["REF"] + df2["ALT"]

# Merging columns based on "POS"
First = pd.merge(df, df1, on=['POS'])
Second = pd.merge(df1, df2, on=['POS'])
Third = pd.merge(df, df2, on=['POS'])
Fourth = pd.merge(First, df2, on=['POS'])

# Dropping of the unnecessary columns.
First = First.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y'], axis=1)
Second = Second.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y'], axis=1)
Third = Third.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y'], axis=1)
Fourth = Fourth.drop(['REF_x', 'ALT_x', 'REF_y', 'ALT_y', 'REF',
    'ALT'], axis=1)
print(First)
print(Second)
print(Third)
print(Fourth)

# Conditional replacement.
First['Result'] = np.where(First["REF_ALT_x"] ==
    First["REF_ALT_y"], 0, 1)
First = First[First["Result"] == 0]
print(First)

Second['Result'] = np.where(Second["REF_ALT_x"] ==
    Second["REF_ALT_y"], 0, 1)
Second = Second[Second["Result"] == 0]
print(Second)

Third['Result'] = np.where(Third["REF_ALT_x"] ==
    Third["REF_ALT_y"], 0, 1)
Third = Third[Third["Result"] == 0]
print(Third)
```

```
Fourth['Result'] = np.where(((Fourth["REF_ALT_x"] ==
    Fourth["REF_ALT_y"]) & (Fourth["REF_ALT_x"] ==
    Fourth["REF_ALT"]) & (Fourth["REF_ALT_y"] ==
    Fourth["REF_ALT"])), 0, 1)
Fourth = Fourth[Fourth["Result"] == 0]
print(Fourth)

# Position outcomes.
print("Number of Indels in Strelka and VarScan:")
Strelka_VarScan_Indels = len(First)
print(Strelka_VarScan_Indels)

print("Number of Indels in VarScan and Truth Data:")
VarScan_Truth_Indels = len(Second)
print(VarScan_Truth_Indels)

print("Number of SNPs in Truth Data and Strelka:")
Strelka_Truth_Indels = len(Third)
print(Strelka_Truth_Indels)

print("Number of SNPs in Strelka, VarScan & Truth Data:")
Strelka_VarScan_Truth_Indels = len(Fourth)
print(Strelka_VarScan_Truth_Indels)

# Delcaring a new dataframe.
df3 = []

# Taking all combinations as a list.
data = {'Type': ['Strelka', 'VarScan', 'Truth_Data',
    'Strelka_and_VarScan', 'VarScan_and_Truth_Data',
    'Truth_Data_and_Strelka',
    'Strelka_and_VarScan_and_Truth_Data'], 'INDELs':
    [Strelka_Indels, VarScan_Indels, Truth_Indels,
    Strelka_VarScan_Indels, VarScan_Truth_Indels,
    Strelka_Truth_Indels, Strelka_VarScan_Truth_Indels]}

# Collecting it into a dataframe.
df3 = pd.DataFrame(data)
print(df3)
```

```
# Saving the results in csv.
df3.to_csv('Indels_Count.csv', sep=',', index = None)

# Formulas
Strelka_Exclude = Strelka_Indels - Strelka_VarScan_Indels -
    Strelka_Truth_Indels - Strelka_VarScan_Truth_Indels
VarScan_Exclude = VarScan_Indels - Strelka_VarScan_Indels -
    VarScan_Truth_Indels - Strelka_VarScan_Truth_Indels
Truth_Exclude = Truth_Indels - VarScan_Truth_Indels -
    Strelka_Truth_Indels - Strelka_VarScan_Truth_Indels

# Set of values.
subsets = (Strelka_Exclude, VarScan_Exclude,
    Strelka_VarScan_Indels, Truth_Exclude, Strelka_Truth_Indels,
    VarScan_Truth_Indels, Strelka_VarScan_Truth_Indels)

# Adding venn diagram.
v = venn3_unweighted(subsets, set_labels = ('Strelka_0.3',
    'VarScan_0.3', 'Truth_Data'), set_colors=('red', 'orange',
    'skyblue'))
areas = (1, 1, 1, 1, 1, 1, 1)
centers, radii = _venn3.solve_venn3_circles(areas)

# Saving the values.
plt.title("Indels")
plt.show()
plt.savefig('Tumor_Purity_0.3_Indels_Plot.pdf')
plt.savefig('Tumor_Purity_0.3_Indels_Plot.png', dpi = 300)
```

8.4 Read Depth

8.4.1 Variant Caller Code

8.4.1.1 Strelka

```
# Importing the needed packages.
import numpy as np
import pandas as pd
```

```
# Reading the csv input file that is obtained after performing the
    following operations on the vcf file.
# Step 1 - 'cut -f 1-2, 9-11 Input-VCF-File > Output-VCF-File'
# Step 2 - 'sed '/^#/d' Output-VCF-File > Updated_Output-VCF-File'

# The first step is to selected the needed columns in the vcf file.
# The second step if to eliminate all lines that start with a '#'
dff = pd.read_csv("Updated_Strelka_0.3.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Updated_Strelka_0.5.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Updated_Strelka_0.7.vcf", sep = '\t',
    index_col= False)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'NORMAL', 'TUMOR', '2:NORMAL',
    '2:TUMOR']
dff1.columns = ['CHROM', 'POS', 'NORMAL', 'TUMOR', '2:NORMAL',
    '2:TUMOR']
dff2.columns = ['CHROM', 'POS', 'NORMAL', 'TUMOR', '2:NORMAL',
    '2:TUMOR']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff2["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

dff1 = dff1.drop(['CHROM', 'POS'], axis=1)
cols = dff1.columns.tolist()
```

```
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]
print(dff1)

dff2 = dff2.drop(['CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
print(dff2)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
dff[['Normal_Read_Depth', 'NORMAL-FDP', 'NORMAL-SDP',
    'NORMAL-SUBDP', 'NORMAL-AU', 'NORMAL-CU', 'NORMAL-GU',
    'NORMAL-TU', 'NORMAL-Last']] =
    dff['NORMAL'].str.split(':',expand=True)
dff[['Tumor_Read_Depth', 'TUMOR-FDP', 'TUMOR-SDP', 'TUMOR-SUBDP',
    'TUMOR-AU', 'TUMOR-CU', 'TUMOR-GU', 'TUMOR-TU', 'TUMOR-Last']]
    = dff['TUMOR'].str.split(':',expand=True)
dff[['2:Normal_Read_Depth', 'NORMAL1-FDP', 'NORMAL1-SDP',
    'NORMAL1-SUBDP', 'NORMAL1-AU', 'NORMAL1-CU', 'NORMAL1-GU',
    'NORMAL1-TU', 'NORMAL1-Last']] =
    dff['2:NORMAL'].str.split(':',expand=True)
dff[['2:Tumor_Read_Depth', 'TUMOR1-FDP', 'TUMOR1-SDP',
    'TUMOR1-SUBDP', 'TUMOR1-AU', 'TUMOR1-CU', 'TUMOR1-GU',
    'TUMOR1-TU', 'TUMOR1-Last']] =
    dff['2:TUMOR'].str.split(':',expand=True)

dff1[['Normal_Read_Depth', 'NORMAL-FDP', 'NORMAL-SDP',
    'NORMAL-SUBDP', 'NORMAL-AU', 'NORMAL-CU', 'NORMAL-GU',
    'NORMAL-TU', 'NORMAL-Last']] =
    dff1['NORMAL'].str.split(':',expand=True)
dff1[['Tumor_Read_Depth', 'TUMOR-FDP', 'TUMOR-SDP', 'TUMOR-SUBDP',
    'TUMOR-AU', 'TUMOR-CU', 'TUMOR-GU', 'TUMOR-TU', 'TUMOR-Last']]
    = dff1['TUMOR'].str.split(':',expand=True)
dff1[['2:Normal_Read_Depth', 'NORMAL1-FDP', 'NORMAL1-SDP',
    'NORMAL1-SUBDP', 'NORMAL1-AU', 'NORMAL1-CU', 'NORMAL1-GU',
    'NORMAL1-TU', 'NORMAL1-Last']] =
    dff1['2:NORMAL'].str.split(':',expand=True)
```

```
dff1[['2:Tumor_Read_Depth', 'TUMOR1-FDP', 'TUMOR1-SDP',
    'TUMOR1-SUBDP', 'TUMOR1-AU', 'TUMOR1-CU', 'TUMOR1-GU',
    'TUMOR1-TU', 'TUMOR1-Last']] =
    dff1['2:TUMOR'].str.split(':',expand=True)

dff2[['Normal_Read_Depth', 'NORMAL-FDP', 'NORMAL-SDP',
    'NORMAL-SUBDP', 'NORMAL-AU', 'NORMAL-CU', 'NORMAL-GU',
    'NORMAL-TU', 'NORMAL-Last']] =
    dff2['NORMAL'].str.split(':',expand=True)
dff2[['Tumor_Read_Depth', 'TUMOR-FDP', 'TUMOR-SDP', 'TUMOR-SUBDP',
    'TUMOR-AU', 'TUMOR-CU', 'TUMOR-GU', 'TUMOR-TU', 'TUMOR-Last']]
    = dff2['TUMOR'].str.split(':',expand=True)
dff2[['2:Normal_Read_Depth', 'NORMAL1-FDP', 'NORMAL1-SDP',
    'NORMAL1-SUBDP', 'NORMAL1-AU', 'NORMAL1-CU', 'NORMAL1-GU',
    'NORMAL1-TU', 'NORMAL1-Last']] =
    dff2['2:NORMAL'].str.split(':',expand=True)
dff2[['2:Tumor_Read_Depth', 'TUMOR1-FDP', 'TUMOR1-SDP',
    'TUMOR1-SUBDP', 'TUMOR1-AU', 'TUMOR1-CU', 'TUMOR1-GU',
    'TUMOR1-TU', 'TUMOR1-Last']] =
    dff2['2:TUMOR'].str.split(':',expand=True)

# Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff = dff.drop(['NORMAL', 'TUMOR', '2:NORMAL', '2:TUMOR',
    'NORMAL-FDP', 'NORMAL-SDP', 'NORMAL-SUBDP', 'NORMAL-AU',
    'NORMAL-CU', 'NORMAL-GU', 'NORMAL-TU', 'NORMAL-Last',
    'TUMOR-FDP', 'TUMOR-SDP', 'TUMOR-SUBDP', 'TUMOR-AU',
    'TUMOR-CU', 'TUMOR-GU', 'TUMOR-TU', 'TUMOR-Last',
    'NORMAL1-FDP', 'NORMAL1-SDP', 'NORMAL1-SUBDP', 'NORMAL1-AU',
    'NORMAL1-CU', 'NORMAL1-GU', 'NORMAL1-TU', 'NORMAL1-Last',
    'TUMOR1-FDP', 'TUMOR1-SDP', 'TUMOR1-SUBDP', 'TUMOR1-AU',
    'TUMOR1-CU', 'TUMOR1-GU', 'TUMOR1-TU', 'TUMOR1-Last'], axis=1)

dff1 = dff1.drop(['NORMAL', 'TUMOR', '2:NORMAL', '2:TUMOR',
    'NORMAL-FDP', 'NORMAL-SDP', 'NORMAL-SUBDP', 'NORMAL-AU',
    'NORMAL-CU', 'NORMAL-GU', 'NORMAL-TU', 'NORMAL-Last',
    'TUMOR-FDP', 'TUMOR-SDP', 'TUMOR-SUBDP', 'TUMOR-AU',
    'TUMOR-CU', 'TUMOR-GU', 'TUMOR-TU', 'TUMOR-Last',
    'NORMAL1-FDP', 'NORMAL1-SDP', 'NORMAL1-SUBDP', 'NORMAL1-AU',
```

```
    'NORMAL1-CU', 'NORMAL1-GU', 'NORMAL1-TU', 'NORMAL1-Last',
    'TUMOR1-FDP', 'TUMOR1-SDP', 'TUMOR1-SUBDP', 'TUMOR1-AU',
    'TUMOR1-CU', 'TUMOR1-GU', 'TUMOR1-TU', 'TUMOR1-Last'], axis=1)

dff2 = dff2.drop(['NORMAL', 'TUMOR', '2:NORMAL', '2:TUMOR',
    'NORMAL-FDP', 'NORMAL-SDP', 'NORMAL-SUBDP', 'NORMAL-AU',
    'NORMAL-CU', 'NORMAL-GU', 'NORMAL-TU', 'NORMAL-Last',
    'TUMOR-FDP', 'TUMOR-SDP', 'TUMOR-SUBDP', 'TUMOR-AU',
    'TUMOR-CU', 'TUMOR-GU', 'TUMOR-TU', 'TUMOR-Last',
    'NORMAL1-FDP', 'NORMAL1-SDP', 'NORMAL1-SUBDP', 'NORMAL1-AU',
    'NORMAL1-CU', 'NORMAL1-GU', 'NORMAL1-TU', 'NORMAL1-Last',
    'TUMOR1-FDP', 'TUMOR1-SDP', 'TUMOR1-SUBDP', 'TUMOR1-AU',
    'TUMOR1-CU', 'TUMOR1-GU', 'TUMOR1-TU', 'TUMOR1-Last'], axis=1)

# Replacing '.' values with '0'
dff.replace('.', '0', inplace=True)
dff1.replace('.', '0', inplace=True)
dff2.replace('.', '0', inplace=True)

# Converting string values columans to int.
dff['Normal_Read_Depth'] = dff['Normal_Read_Depth'].astype(int)
dff['2:Normal_Read_Depth'] = dff['2:Normal_Read_Depth'].astype(int)
dff['Tumor_Read_Depth'] = dff['Tumor_Read_Depth'].astype(int)
dff['2:Tumor_Read_Depth'] = dff['2:Tumor_Read_Depth'].astype(int)

dff1['Normal_Read_Depth'] = dff1['Normal_Read_Depth'].astype(int)
dff1['2:Normal_Read_Depth'] =
    dff1['2:Normal_Read_Depth'].astype(int)
dff1['Tumor_Read_Depth'] = dff1['Tumor_Read_Depth'].astype(int)
dff1['2:Tumor_Read_Depth'] = dff1['2:Tumor_Read_Depth'].astype(int)

dff2['Normal_Read_Depth'] = dff2['Normal_Read_Depth'].astype(int)
dff2['2:Normal_Read_Depth'] =
    dff2['2:Normal_Read_Depth'].astype(int)
dff2['Tumor_Read_Depth'] = dff2['Tumor_Read_Depth'].astype(int)
dff2['2:Tumor_Read_Depth'] = dff2['2:Tumor_Read_Depth'].astype(int)

# Adding columns for single read depth value.
dff['Normal_RD'] = dff["Normal_Read_Depth"] +
    dff["2:Normal_Read_Depth"]
```

```
dff['Tumor_RD'] = dff["Tumor_Read_Depth"] +
    dff["2:Tumor_Read_Depth"]

dff1['Normal_RD'] = dff1["Normal_Read_Depth"] +
    dff1["2:Normal_Read_Depth"]
dff1['Tumor_RD'] = dff1["Tumor_Read_Depth"] +
    dff1["2:Tumor_Read_Depth"]

dff2['Normal_RD'] = dff2["Normal_Read_Depth"] +
    dff2["2:Normal_Read_Depth"]
dff2['Tumor_RD'] = dff2["Tumor_Read_Depth"] +
    dff2["2:Tumor_Read_Depth"]

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['Normal_Read_Depth', 'Tumor_Read_Depth',
    '2:Normal_Read_Depth', '2:Tumor_Read_Depth'], axis=1)
dff.columns = ['CHROM_POS', 'Normal_Read_Depth',
    'Tumor_Read_Depth']
print(dff)

dff1 = dff1.drop(['Normal_Read_Depth', 'Tumor_Read_Depth',
    '2:Normal_Read_Depth', '2:Tumor_Read_Depth'], axis=1)
dff1.columns = ['CHROM_POS', 'Normal_Read_Depth',
    'Tumor_Read_Depth']
print(dff1)

dff2 = dff2.drop(['Normal_Read_Depth', 'Tumor_Read_Depth',
    '2:Normal_Read_Depth', '2:Tumor_Read_Depth'], axis=1)
dff2.columns = ['CHROM_POS', 'Normal_Read_Depth',
    'Tumor_Read_Depth']
print(dff2)

# Merging columns based on "CHROM-POS"
First = pd.merge(dff, dff1, on=['CHROM_POS'])
Second = pd.merge(First, dff2, on=['CHROM_POS'])

# Renaming Columns
Second.columns = ['CHROM_POS', 'Strelka_Normal_0.3',
    'Strelka_Tumor_0.3', 'Strelka_Normal_0.5', 'Strelka_Tumor_0.5',
    'Strelka_Normal_0.7', 'Strelka_Tumor_0.7']
```

```
print(Second)

# Saving the results in csv.
Second.to_csv('Strelka_Read_Depth.csv', sep=',', index = None)

# Finding the minimum values
min1 = Second['Strelka_Normal_0.3'].min()
min2 = Second['Strelka_Tumor_0.3'].min()
min3 = Second['Strelka_Normal_0.5'].min()
min4 = Second['Strelka_Tumor_0.5'].min()
min5 = Second['Strelka_Normal_0.7'].min()
min6 = Second['Strelka_Tumor_0.7'].min()

# Finding the maximum values
max1 = Second['Strelka_Normal_0.3'].max()
max2 = Second['Strelka_Tumor_0.3'].max()
max3 = Second['Strelka_Normal_0.5'].max()
max4 = Second['Strelka_Tumor_0.5'].max()
max5 = Second['Strelka_Normal_0.7'].max()
max6 = Second['Strelka_Tumor_0.7'].max()

# Finding the mean values
mean1 = Second['Strelka_Normal_0.3'].mean()
mean2 = Second['Strelka_Tumor_0.3'].mean()
mean3 = Second['Strelka_Normal_0.5'].mean()
mean4 = Second['Strelka_Tumor_0.5'].mean()
mean5 = Second['Strelka_Normal_0.7'].mean()
mean6 = Second['Strelka_Tumor_0.7'].mean()

# Finding the minimum values
median1 = Second['Strelka_Normal_0.3'].median()
median2 = Second['Strelka_Tumor_0.3'].median()
median3 = Second['Strelka_Normal_0.5'].median()
median4 = Second['Strelka_Tumor_0.5'].median()
median5 = Second['Strelka_Normal_0.7'].median()
median6 = Second['Strelka_Tumor_0.7'].median()
print(median6)

# Finding the minimum values
mode1 = Second['Strelka_Normal_0.3'].mode()
```

```
mode2 = Second['Strelka_Tumor_0.3'].mode()
mode3 = Second['Strelka_Normal_0.5'].mode()
mode4 = Second['Strelka_Tumor_0.5'].mode()
mode5 = Second['Strelka_Normal_0.7'].mode()
mode6 = Second['Strelka_Tumor_0.7'].mode()
print(mode6)

# Delcaring a new dataframe.
df = pd.DataFrame()

# Taking all combinations as a list.
Type = ['Strelka_Normal_0.3', 'Strelka_Tumor_0.3',
    'Strelka_Normal_0.5', 'Strelka_Tumor_0.5',
    'Strelka_Normal_0.7', 'Strelka_Tumor_0.7']
Minimum_Value = [min1, min2, min3, min4, min5, min6]
Maximum_Value = [max1, max2, max3, max4, max5, max6]
Mean_Value = [mean1, mean2, mean3, mean4, mean5, mean6]
Median_Value = [median1, median2, median3, median4, median5,
    median6]
Mode_Value = [mode1, mode2, mode3, mode4, mode5, mode6]

# Adding columns
df['Type'] = Type
df['Minimum_Value'] = Minimum_Value
df['Maximum_Value'] = Maximum_Value
df['Mean_Value'] = Mean_Value
df['Median_Value'] = Median_Value
df['Mode_Value'] = Mode_Value
# Collecting it into a dataframe.
print(df)

# Saving the results in csv.
df.to_csv('Strelka_Read_Depth_Statistics.csv', sep=',', index =
    False)
```

8.4.1.2 Truth Data

```
# Importing the needed packages.
import numpy as np
```

```
import pandas as pd

# Reading the csv input file that is obtained after performing the
    following operations on the vcf file.
# Step 1 - 'cut -f 1-2,9-10 Input.vcf > Output.vcf'
# Step 2 - 'sed '/^#/d' Output.vcf > Updated.vcf'

# The first step is to selected the needed columns in the vcf file.
# The second step if to eliminate all lines that start with a '#'
dff = pd.read_csv("Updated_Somatic_Truth.vcf", sep = '\t',
    index_col= False)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'FORMAT', 'VALUE']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS', 'FORMAT'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "GT:PS:DP:GQ".
dff[['VALUE-GT','VALUE-PS','Read_Depth', 'VALUE-GQ']] =
    dff['VALUE'].str.split(':',expand=True)

# Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff = dff.drop(['VALUE', 'VALUE-GT', 'VALUE-PS', 'VALUE-GQ'],
    axis=1)
print(dff)

# Saving the result into a csv file for plotting.
```

```
dff.to_csv('Truth_Data_Read_Depth.csv', sep=',', index = None)

# Finding the minimum values
min1 = dff['Read_Depth'].min()

# Finding the maximum values
max1 = dff['Read_Depth'].max()

# Finding the mean values
mean1 = dff['Read_Depth'].mean()

# Finding the minimum values
median1 = dff['Read_Depth'].median()

# Finding the minimum values
mode1 = dff['Read_Depth'].mode()

# Delcaring a new dataframe.
df = pd.DataFrame()

# Taking all combinations as a list.
Type = ['Truth_Data']
Minimum_Value = [min1]
Maximum_Value = [max1]
Mean_Value = [mean1]
Median_Value = [median1]
Mode_Value = [mode1]

# Adding columns
df['Type'] = Type
df['Minimum_Value'] = Minimum_Value
df['Maximum_Value'] = Maximum_Value
df['Mean_Value'] = Mean_Value
df['Median_Value'] = Median_Value
df['Mode_Value'] = Mode_Value
# Collecting it into a dataframe.
print(df)

# Saving the results in csv.
```

```
df.to_csv('Truth_Data_Read_Depth_Statistics.csv', sep=',', index =
    False)
```

8.4.1.3 VarScan

```
# Importing the needed packages.
import numpy as np
import pandas as pd

# Reading the csv input file that is obtained after performing the
    following operations on the vcf file.
# Step 1 - 'cut -f 1-2,9-11 Input-VCF-File > Output-VCF-File'
# Step 2 - 'sed '/^#/d' Output-VCF-File > Updated_Output-VCF-File'

# The first step is to selected the needed columns in the vcf file.
# The second step if to eliminate all lines that start with a '#'
dff = pd.read_csv("Updated_VarScan_0.3.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Updated_VarScan_0.5.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Updated_VarScan_0.7.vcf", sep = '\t',
    index_col= False)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'FORMAT', 'NORMAL', 'TUMOR']
dff1.columns = ['CHROM', 'POS', 'FORMAT', 'NORMAL', 'TUMOR']
dff2.columns = ['CHROM', 'POS', 'FORMAT', 'NORMAL', 'TUMOR']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff1['CHROM'].astype(str) + '-' +
    dff1['POS'].astype(str)
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS', 'FORMAT'], axis=1)
cols = dff.columns.tolist()
```

```
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

dff1 = dff1.drop(['CHROM', 'POS', 'FORMAT'], axis=1)
cols = dff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]
print(dff1)

dff2 = dff2.drop(['CHROM', 'POS', 'FORMAT'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
print(dff2)

# Creating new columns by splitting the "NORMAL" and "TUMOR"
    columns by ':' and renaming the new columns based on the format
    "GT:GQ:DP:AD:ADF:ADR".
dff[['NORMAL-GT','NORMAL-GQ','Normal_Read_Depth',
    'NORMAL-AD','NORMAL-ADF', 'NORMAL-ADR']] =
    dff['NORMAL'].str.split(':',expand=True)
dff[['TUMOR-GT','TUMOR-GQ','Tumor_Read_Depth',
    'TUMOR-AD','TUMOR-ADF', 'TUMOR-ADR']] =
    dff['TUMOR'].str.split(':',expand=True)

dff1[['NORMAL-GT','NORMAL-GQ','Normal_Read_Depth',
    'NORMAL-AD','NORMAL-ADF', 'NORMAL-ADR']] =
    dff1['NORMAL'].str.split(':',expand=True)
dff1[['TUMOR-GT','TUMOR-GQ','Tumor_Read_Depth',
    'TUMOR-AD','TUMOR-ADF', 'TUMOR-ADR']] =
    dff1['TUMOR'].str.split(':',expand=True)

dff2[['NORMAL-GT','NORMAL-GQ','Normal_Read_Depth',
    'NORMAL-AD','NORMAL-ADF', 'NORMAL-ADR']] =
    dff2['NORMAL'].str.split(':',expand=True)
dff2[['TUMOR-GT','TUMOR-GQ','Tumor_Read_Depth',
    'TUMOR-AD','TUMOR-ADF', 'TUMOR-ADR']] =
    dff2['TUMOR'].str.split(':',expand=True)
```

```
# Dropping of the unnecessary columns and only choosing the
    "NORMAL Depth" i.e. "NORMAL-DP" and "TUMOR Depth" i.e.
    "TUMOR-DP"
dff = dff.drop(['NORMAL', 'TUMOR', 'NORMAL-GT', 'NORMAL-GQ',
    'NORMAL-AD', 'NORMAL-ADF', 'NORMAL-ADR', 'TUMOR-GT',
    'TUMOR-GQ', 'TUMOR-AD', 'TUMOR-ADF', 'TUMOR-ADR'], axis=1)
print(dff)

dff1 = dff1.drop(['NORMAL', 'TUMOR', 'NORMAL-GT', 'NORMAL-GQ',
    'NORMAL-AD', 'NORMAL-ADF', 'NORMAL-ADR', 'TUMOR-GT',
    'TUMOR-GQ', 'TUMOR-AD', 'TUMOR-ADF', 'TUMOR-ADR'], axis=1)
print(dff1)

dff2 = dff2.drop(['NORMAL', 'TUMOR', 'NORMAL-GT', 'NORMAL-GQ',
    'NORMAL-AD', 'NORMAL-ADF', 'NORMAL-ADR', 'TUMOR-GT',
    'TUMOR-GQ', 'TUMOR-AD', 'TUMOR-ADF', 'TUMOR-ADR'], axis=1)
print(dff2)

# Merging columns based on "CHROM-POS"
First = pd.merge(dff, dff1, on=['CHROM_POS'])
Second = pd.merge(First, dff2, on=['CHROM_POS'])

# Renaming Columns
Second.columns = ['CHROM_POS', 'VarScan_Normal_0.3',
    'VarScan_Tumor_0.3', 'VarScan_Normal_0.5', 'VarScan_Tumor_0.5',
    'VarScan_Normal_0.7', 'VarScan_Tumor_0.7']
print(Second)

# Saving the results in csv.
Second.to_csv('VarScan_Read_Depth_Counts.csv', sep=',', index =
    None)

# Finding the minimum values
min1 = Second['VarScan_Normal_0.3'].min()
min2 = Second['VarScan_Tumor_0.3'].min()
min3 = Second['VarScan_Normal_0.5'].min()
min4 = Second['VarScan_Tumor_0.5'].min()
min5 = Second['VarScan_Normal_0.7'].min()
min6 = Second['VarScan_Tumor_0.7'].min()
```

```
# Finding the maximum values
max1 = Second['VarScan_Normal_0.3'].max()
max2 = Second['VarScan_Tumor_0.3'].max()
max3 = Second['VarScan_Normal_0.5'].max()
max4 = Second['VarScan_Tumor_0.5'].max()
max5 = Second['VarScan_Normal_0.7'].max()
max6 = Second['VarScan_Tumor_0.7'].max()

# Finding the mean values
mean1 = Second['VarScan_Normal_0.3'].mean()
mean2 = Second['VarScan_Tumor_0.3'].mean()
mean3 = Second['VarScan_Normal_0.5'].mean()
mean4 = Second['VarScan_Tumor_0.5'].mean()
mean5 = Second['VarScan_Normal_0.7'].mean()
mean6 = Second['VarScan_Tumor_0.7'].mean()

# Finding the minimum values
median1 = Second['VarScan_Normal_0.3'].median()
median2 = Second['VarScan_Tumor_0.3'].median()
median3 = Second['VarScan_Normal_0.5'].median()
median4 = Second['VarScan_Tumor_0.5'].median()
median5 = Second['VarScan_Normal_0.7'].median()
median6 = Second['VarScan_Tumor_0.7'].median()

# Finding the minimum values
mode1 = Second['VarScan_Normal_0.3'].mode()
mode2 = Second['VarScan_Tumor_0.3'].mode()
mode3 = Second['VarScan_Normal_0.5'].mode()
mode4 = Second['VarScan_Tumor_0.5'].mode()
mode5 = Second['VarScan_Normal_0.7'].mode()
mode6 = Second['VarScan_Tumor_0.7'].mode()

# Delcaring a new dataframe.
df = pd.DataFrame()

# Taking all combinations as a list.
Type = ['VarScan_Normal_0.3', 'VarScan_Tumor_0.3',
    'VarScan_Normal_0.5', 'VarScan_Tumor_0.5',
    'VarScan_Normal_0.7', 'VarScan_Tumor_0.7']
Minimum_Value = [min1, min2, min3, min4, min5, min6]
```

```
Maximum_Value = [max1, max2, max3, max4, max5, max6]
Mean_Value = [mean1, mean2, mean3, mean4, mean5, mean6]
Median_Value = [median1, median2, median3, median4, median5,
    median6]
Mode_Value = [mode1, mode2, mode3, mode4, mode5, mode6]

# Adding columns
df['Type'] = Type
df['Minimum_Value'] = Minimum_Value
df['Maximum_Value'] = Maximum_Value
df['Mean_Value'] = Mean_Value
df['Median_Value'] = Median_Value
df['Mode_Value'] = Mode_Value
# Collecting it into a dataframe.
print(df)

# Saving the results in csv.
df.to_csv('VarScan_Read_Depth_Statistics.csv', sep=',', index =
    False)
```

8.4.2 Comparison

```
# Importing packages.
import numpy as np
import pandas as pd

# Reading csv files and concatinating "CHROM" and "POS"
df = pd.read_csv("Strelka_Read_Depth_Counts.csv", sep = ',',
    index_col= False)
df1 = pd.read_csv("VarScan_Read_Depth_Counts.csv", sep = ',',
    index_col= False)
df2 = pd.read_csv("Truth_Data_Read_Depth_Counts.csv", sep = ',',
    index_col= False)

# Merging columns based on "CHROM-POS"
First = pd.merge(df, df1, on=['CHROM_POS'])
Second = pd.merge(First, df2, on=['CHROM_POS'])
```

```
# Assigning column names.
Second.columns = ['CHROM_POS', 'Strelka_Normal_0.3_Read_Depth',
    'Strelka_Tumor_0.3_Read_Depth',
    'Strelka_Normal_0.5_Read_Depth',
    'Strelka_Tumor_0.5_Read_Depth',
    'Strelka_Normal_0.7_Read_Depth',
    'Strelka_Tumor_0.7_Read_Depth',
    'VarScan_Normal_0.3_Read_Depth',
    'VarScan_Tumor_0.3_Read_Depth',
    'VarScan_Normal_0.5_Read_Depth',
    'VarScan_Tumor_0.5_Read_Depth',
    'VarScan_Normal_0.7_Read_Depth',
    'VarScan_Tumor_0.7_Read_Depth', 'Somatic_Truth_Read_Depth']

# Deleting the unneeded columns.
Second = Second.drop(['Strelka_Normal_0.5_Read_Depth',
    'Strelka_Tumor_0.5_Read_Depth',
    'Strelka_Normal_0.7_Read_Depth',
    'Strelka_Tumor_0.7_Read_Depth',
    'VarScan_Normal_0.5_Read_Depth',
    'VarScan_Tumor_0.5_Read_Depth',
    'VarScan_Normal_0.7_Read_Depth',
    'VarScan_Tumor_0.7_Read_Depth'], axis=1)
print(Second)

# Saving the results in csv.
Second.to_csv('Tumor_Purity_0.3_Read_Depths.csv', sep=',',
    index=False)

Second.columns = ['CHROM_POS', 'Strelka_Normal', 'Strelka_Tumor',
    'VarScan_Normal', 'VarScan_Tumor', 'Truth_Data']
print(Second)

# Finding the minimum values
min1 = Second['Strelka_Normal'].min()
min2 = Second['Strelka_Tumor'].min()
min3 = Second['VarScan_Normal'].min()
min4 = Second['VarScan_Tumor'].min()
min5 = Second['Truth_Data'].min()
```

```
# Finding the maximum values
max1 = Second['Strelka_Normal'].max()
max2 = Second['Strelka_Tumor'].max()
max3 = Second['VarScan_Normal'].max()
max4 = Second['VarScan_Tumor'].max()
max5 = Second['Truth_Data'].max()

# Finding the mean values
mean1 = Second['Strelka_Normal'].mean()
mean2 = Second['Strelka_Tumor'].mean()
mean3 = Second['VarScan_Normal'].mean()
mean4 = Second['VarScan_Tumor'].mean()
mean5 = Second['Truth_Data'].mean()

# Finding the minimum values
median1 = Second['Strelka_Normal'].median()
median2 = Second['Strelka_Tumor'].median()
median3 = Second['VarScan_Normal'].median()
median4 = Second['VarScan_Tumor'].median()
median5 = Second['Truth_Data'].median()

# Finding the minimum values
mode1 = Second['Strelka_Normal'].mode()
mode2 = Second['Strelka_Tumor'].mode()
mode3 = Second['VarScan_Normal'].mode()
mode4 = Second['VarScan_Tumor'].mode()
mode5 = Second['Truth_Data'].mode()

# Delcaring a new dataframe.
df = pd.DataFrame()

# Taking all combinations as a list.
Type = ['Strelka_Normal_0.3', 'Strelka_Tumor_0.3',
    'VarScan_Normal_0.3', 'VarScan_Tumor_0.3', 'Truth_Data']
Minimum_Value = [min1, min2, min3, min4, min5]
Maximum_Value = [max1, max2, max3, max4, max5]
Mean_Value = [mean1, mean2, mean3, mean4, mean5]
Median_Value = [median1, median2, median3, median4, median5]
Mode_Value = [mode1, mode2, mode3, mode4, mode5]
```

```
# Adding columns
df['Type'] = Type
df['Minimum_Value'] = Minimum_Value
df['Maximum_Value'] = Maximum_Value
df['Mean_Value'] = Mean_Value
df['Median_Value'] = Median_Value
df['Mode_Value'] = Mode_Value
# Collecting it into a dataframe.
print(df)

# Saving the results in csv.
df.to_csv('Tumor_Purity_0.3_Read_Depth_Statistics.csv', sep=',',
    index = False)
```

8.5 Variants

8.5.1 Variant Caller Code

8.5.1.1 Strelka

```
# Importing the needed packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import pandas as pd
import csv
import numpy as np

# Reading the csv input file that is obtained after performing the
    following operations on the vcf file.
# Step 1 - 'cut -f 1-2,4-5 Input.vcf > Output.vcf'
# Step 2 - 'sed '/^#/d' Output.vcf > Updated.vcf'

# The first step is to selected the needed columns in the vcf file.
```

```
# The second step if to eliminate all lines that start with a '#'
dff = pd.read_csv("Updated_Strelka_0.3_SNV.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Updated_Strelka_0.5_SNV.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Updated_Strelka_0.7_SNV.vcf", sep = '\t',
    index_col= False)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'REF', 'ALT']
dff1.columns = ['CHROM', 'POS', 'REF', 'ALT']
dff2.columns = ['CHROM', 'POS', 'REF', 'ALT']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff1['CHROM'].astype(str) + '-' +
    dff1['POS'].astype(str)
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

dff1 = dff1.drop(['CHROM', 'POS'], axis=1)
cols = dff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]
print(dff1)

dff2 = dff2.drop(['CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
print(dff2)
```

```
# Mentioning the column names and inputing the csv file.
dff.columns = ['CHROM_POS', 'REF', 'ALT']
dff1.columns = ['CHROM_POS', 'REF', 'ALT']
dff2.columns = ['CHROM_POS', 'REF', 'ALT']

# Printing the list.
dff4 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'A')]
dff5 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'A')]
dff6 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'A')]
AA1 = len(dff4.index)
AA2 = len(dff5.index)
AA3 = len(dff6.index)

dff7 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'T')]
dff8 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'T')]
dff9 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'T')]
AT1 = len(dff7.index)
AT2 = len(dff8.index)
AT3 = len(dff9.index)

dff10 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'G')]
dff11 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'G')]
dff12 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'G')]
AG1 = len(dff10.index)
AG2 = len(dff11.index)
AG3 = len(dff12.index)

dff13 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'C')]
dff14 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'C')]
dff15 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'C')]
AC1 = len(dff13.index)
AC2 = len(dff14.index)
AC3 = len(dff15.index)

dff16 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'T')]
dff17 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'T')]
dff18 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'T')]
TT1 = len(dff16.index)
TT2 = len(dff17.index)
TT3 = len(dff18.index)
```

```
dff19 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'A')]
dff20 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'A')]
dff21 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'A')]
TA1 = len(dff19.index)
TA2 = len(dff20.index)
TA3 = len(dff21.index)

dff22 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'G')]
dff23 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'G')]
dff24 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'G')]
TG1 = len(dff22.index)
TG2 = len(dff23.index)
TG3 = len(dff24.index)

dff25 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'C')]
dff26 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'C')]
dff27 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'C')]
TC1 = len(dff25.index)
TC2 = len(dff26.index)
TC3 = len(dff27.index)

dff28 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'G')]
dff29 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'G')]
dff30 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'G')]
GG1 = len(dff28.index)
GG2 = len(dff29.index)
GG3 = len(dff30.index)

dff31 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'A')]
dff32 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'A')]
dff33 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'A')]
GA1 = len(dff31.index)
GA2 = len(dff32.index)
GA3 = len(dff33.index)

dff34 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'T')]
dff35 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'T')]
dff36 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'T')]
GT1 = len(dff34.index)
```

```
GT2 = len(dff35.index)
GT3 = len(dff36.index)

dff37 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'C')]
dff38 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'C')]
dff39 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'C')]
GC1 = len(dff37.index)
GC2 = len(dff38.index)
GC3 = len(dff39.index)

dff40 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'C')]
dff41 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'C')]
dff42 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'C')]
CC1 = len(dff40.index)
CC2 = len(dff41.index)
CC3 = len(dff42.index)

dff43 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'A')]
dff44 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'A')]
dff45 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'A')]
CA1 = len(dff43.index)
CA2 = len(dff44.index)
CA3 = len(dff45.index)

dff46 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'T')]
dff47 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'T')]
dff48 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'T')]
CT1 = len(dff46.index)
CT2 = len(dff47.index)
CT3 = len(dff48.index)

dff49 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'G')]
dff50 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'G')]
dff51 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'G')]
CG1 = len(dff49.index)
CG2 = len(dff50.index)
CG3 = len(dff51.index)

# Delcaring a new dataframe.
df = []
```

```
df1 = []
df2 = []

# Taking all combinations as a list.
data = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'Strelka_0.3': [AA1, AT1, AG1, AC1, TT1, TA1, TG1, TC1, GG1,
    GA1, GT1, GC1, CC1, CA1, CT1, CG1]}
data1 = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'Strelka_0.5': [AA2, AT2, AG2, AC2, TT2, TA2, TG2, TC2, GG2,
    GA2, GT2, GC2, CC2, CA2, CT2, CG2]}
data2 = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'Strelka_0.7': [AA3, AT3, AG3, AC3, TT3, TA3, TG3, TC3, GG3,
    GA3, GT3, GC3, CC3, CA3, CT3, CG3]}

# Collecting it into a dataframe.
df = pd.DataFrame(data)
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Merging columns based on "CHROM-POS"
First = pd.merge(df, df1, on=['ALT', 'REF'])
Second = pd.merge(First, df2, on=['ALT', 'REF'])

# Mentioning the column names and inputing the csv file.
Second.columns = ['REF', 'ALT', 'Strelka_0.3', 'Strelka_0.5',
    'Strelka_0.7']
print(Second)

# Saving the results in csv.
Second.to_csv('Strelka_Counts.csv', sep=',', index = None)

# Reading csv files and concatinating "CHROM" and "POS"
dff = pd.read_csv("Strelka_Counts.csv", sep = ',', index_col=
    False, error_bad_lines=False)
```

```
dff.columns = ['REF', 'ALT', 'Strelka_One', 'Strelka_Two',
    'Strelka_Three']

# set width of bar
width = 0.25

# Columns from the file
a1 = dff.Strelka_One.to_list()
a2 = dff.Strelka_Two.to_list()
a3 = dff.Strelka_Three.to_list()

# Set position of bar on X axis
r1 = np.arange(len(a1))
r2 = [x + width for x in r1]
r3 = [x + width for x in r2]

# Make the plot
plt.bar(r1, a1, color='#FFD700', width=width, edgecolor='white',
    label='Strelka_0.3')
plt.bar(r2, a2, color='#FFA500', width=width, edgecolor='white',
    label='Strelka_0.5')
plt.bar(r3, a3, color='#DC143C', width=width, edgecolor='white',
    label='Strelka_0.7')

# Add xticks on the middle of the group bars
plt.xlabel('Combinations')
plt.xticks([r + width for r in range(len(a1))], ['AA', 'AT', 'AG',
    'AC', 'TT', 'TA', 'TG', 'TC', 'GG', 'GA', 'GT', 'GC', 'CC',
    'CA', 'CT', 'CG'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('Strelka_Counts_Plot.pdf')
plt.savefig('Strelka_Counts_Plot.png', dpi = 300)
```

8.5.1.2 Truth Data

```
# Importing the needed packages.
```

```
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import pandas as pd
import csv
import numpy as np

# Importing the csv file.
dff = pd.read_csv("Updated_Somatic_Truth_SNP.vcf", sep = '\t',
    index_col= False)

# Mentioning the column names and inputing the csv file.
dff.columns = ['CHROM', 'POS', 'REF', 'ALT']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

# Mentioning the column names and inputing the csv file.
dff.columns = ['CHROM_POS', 'REF', 'ALT']

# Printing the list.
dff1 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'A')]
AA = len(dff1.index)
print('The number of REF as A and ALT as A is')
print(AA)

dff2 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'T')]
AT = len(dff2.index)
```

```
print('The number of REF as A and ALT as T is')
print(AT)

dff3 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'G')]
AG = len(dff3.index)
print('The number of REF as A and ALT as G is')
print(AG)

dff4 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'C')]
AC = len(dff4.index)
print('The number of REF as A and ALT as C is')
print(AC)

dff5 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'T')]
TT = len(dff5.index)
print('The number of REF as T and ALT as T is')
print(TT)

dff6 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'A')]
TA = len(dff6.index)
print('The number of REF as T and ALT as A is')
print(TA)

dff7 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'G')]
TG = len(dff7.index)
print('The number of REF as T and ALT as G is')
print(TG)

dff8 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'C')]
TC = len(dff8.index)
print('The number of REF as T and ALT as C is')
print(TC)

dff9 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'G')]
GG = len(dff9.index)
print('The number of REF as G and ALT as G is')
print(GG)

dff10 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'A')]
GA = len(dff10.index)
```

```
print('The number of REF as G and ALT as A is')
print(GA)

dff11 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'T')]
GT = len(dff11.index)
print('The number of REF as G and ALT as T is')
print(GT)

dff12 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'C')]
GC = len(dff12.index)
print('The number of REF as G and ALT as C is')
print(GC)

dff13 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'C')]
CC = len(dff13.index)
print('The number of REF as C and ALT as C is')
print(CC)

dff14 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'A')]
CA = len(dff14.index)
print('The number of REF as C and ALT as A is')
print(CA)

dff15 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'T')]
CT = len(dff15.index)
print('The number of REF as C and ALT as T is')
print(CT)

dff16 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'G')]
CG = len(dff16.index)
print('The number of REF as C and ALT as G is')
print(CG)

# Delcaring a new dataframe.
df = []

# Taking all combinations as a list.
data = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
```

```
    'Truth_Data': [AA, AT, AG, AC, TT, TA, TG, TC, GG, GA, GT, GC,
    CC, CA, CT, CG]}

# Collecting it into a dataframe.
df = pd.DataFrame(data)
print(df)

# Exporting the outcome into CSV.
df.to_csv('Truth_Data_Counts.csv', sep=',', index = None)

# Reading csv files and concatinating "CHROM" and "POS"
dff = pd.read_csv("Truth_Data_Counts.csv", sep = ',', index_col=
    False, error_bad_lines=False)
dff.columns = ['REF', 'ALT', 'Truth_Data']

# set width of bar
width = 0.35

# Columns from the file
a1 = dff.Truth_Data.to_list()

# Set position of bar on X axis
r1 = np.arange(len(a1))

# Make the plot
plt.bar(r1, a1, color='#FFD700', width=width, edgecolor='white',
    label='Truth_Data')

# Add xticks on the middle of the group bars
plt.xlabel('Combinations')
plt.xticks([r + width for r in range(len(a1))], ['AA', 'AT', 'AG',
    'AC', 'TT', 'TA', 'TG', 'TC', 'GG', 'GA', 'GT', 'GC', 'CC',
    'CA', 'CT', 'CG'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('Truth_Data_Counts_Plot.pdf')
plt.savefig('Truth_Data_Counts_Plot.png', dpi = 300)
```

8.5.1.3 VarScan

```
# Importing the needed packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import pandas as pd
import csv
import numpy as np

# Reading the csv input file that is obtained after performing the
    following operations on the vcf file.
# Step 1 - 'cut -f 1-2,4-5 Input.vcf > Output.vcf'
# Step 2 - 'sed '/^#/d' Output.vcf > Updated.vcf'

# The first step is to selected the needed columns in the vcf file.
# The second step if to eliminate all lines that start with a '#'
dff = pd.read_csv("Updated_VarScan_0.3_SNP.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Updated_VarScan_0.5_SNP.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Updated_VarScan_0.7_SNP.vcf", sep = '\t',
    index_col= False)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'REF', 'ALT']
dff1.columns = ['CHROM', 'POS', 'REF', 'ALT']
dff2.columns = ['CHROM', 'POS', 'REF', 'ALT']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff1['CHROM'].astype(str) + '-' +
    dff1['POS'].astype(str)
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)
```

```
# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

dff1 = dff1.drop(['CHROM', 'POS'], axis=1)
cols = dff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]
print(dff1)

dff2 = dff2.drop(['CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
print(dff2)

# Mentioning the column names and inputing the csv file.
dff.columns = ['CHROM_POS', 'REF', 'ALT']
dff1.columns = ['CHROM_POS', 'REF', 'ALT']
dff2.columns = ['CHROM_POS', 'REF', 'ALT']

# Printing the list.
dff4 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'A')]
dff5 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'A')]
dff6 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'A')]
AA1 = len(dff4.index)
AA2 = len(dff5.index)
AA3 = len(dff6.index)

dff7 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'T')]
dff8 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'T')]
dff9 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'T')]
AT1 = len(dff7.index)
AT2 = len(dff8.index)
AT3 = len(dff9.index)
```

```
dff10 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'G')]
dff11 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'G')]
dff12 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'G')]
AG1 = len(dff10.index)
AG2 = len(dff11.index)
AG3 = len(dff12.index)

dff13 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'C')]
dff14 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'C')]
dff15 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'C')]
AC1 = len(dff13.index)
AC2 = len(dff14.index)
AC3 = len(dff15.index)

dff16 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'T')]
dff17 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'T')]
dff18 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'T')]
TT1 = len(dff16.index)
TT2 = len(dff17.index)
TT3 = len(dff18.index)

dff19 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'A')]
dff20 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'A')]
dff21 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'A')]
TA1 = len(dff19.index)
TA2 = len(dff20.index)
TA3 = len(dff21.index)

dff22 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'G')]
dff23 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'G')]
dff24 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'G')]
TG1 = len(dff22.index)
TG2 = len(dff23.index)
TG3 = len(dff24.index)

dff25 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'C')]
dff26 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'C')]
dff27 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'C')]
TC1 = len(dff25.index)
TC2 = len(dff26.index)
```

```
TC3 = len(dff27.index)

dff28 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'G')]
dff29 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'G')]
dff30 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'G')]
GG1 = len(dff28.index)
GG2 = len(dff29.index)
GG3 = len(dff30.index)

dff31 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'A')]
dff32 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'A')]
dff33 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'A')]
GA1 = len(dff31.index)
GA2 = len(dff32.index)
GA3 = len(dff33.index)

dff34 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'T')]
dff35 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'T')]
dff36 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'T')]
GT1 = len(dff34.index)
GT2 = len(dff35.index)
GT3 = len(dff36.index)

dff37 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'C')]
dff38 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'C')]
dff39 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'C')]
GC1 = len(dff37.index)
GC2 = len(dff38.index)
GC3 = len(dff39.index)

dff40 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'C')]
dff41 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'C')]
dff42 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'C')]
CC1 = len(dff40.index)
CC2 = len(dff41.index)
CC3 = len(dff42.index)

dff43 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'A')]
dff44 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'A')]
dff45 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'A')]
```

```
CA1 = len(dff43.index)
CA2 = len(dff44.index)
CA3 = len(dff45.index)

dff46 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'T')]
dff47 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'T')]
dff48 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'T')]
CT1 = len(dff46.index)
CT2 = len(dff47.index)
CT3 = len(dff48.index)

dff49 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'G')]
dff50 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'G')]
dff51 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'G')]
CG1 = len(dff49.index)
CG2 = len(dff50.index)
CG3 = len(dff51.index)

# Delcaring a new dataframe.
df = []
df1 = []
df2 = []

# Taking all combinations as a list.
data = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'VarScan_0.3': [AA1, AT1, AG1, AC1, TT1, TA1, TG1, TC1, GG1,
    GA1, GT1, GC1, CC1, CA1, CT1, CG1]}
data1 = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'VarScan_0.5': [AA2, AT2, AG2, AC2, TT2, TA2, TG2, TC2, GG2,
    GA2, GT2, GC2, CC2, CA2, CT2, CG2]}
data2 = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'VarScan_0.7': [AA3, AT3, AG3, AC3, TT3, TA3, TG3, TC3, GG3,
    GA3, GT3, GC3, CC3, CA3, CT3, CG3]}
```

```
# Collecting it into a dataframe.
df = pd.DataFrame(data)
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Merging columns based on "CHROM-POS"
First = pd.merge(df, df1, on=['ALT', 'REF'])
Second = pd.merge(First, df2, on=['ALT', 'REF'])

# Mentioning the column names and inputing the csv file.
Second.columns = ['REF', 'ALT', 'VarScan_0.3', 'VarScan_0.5',
    'VarScan_0.7']
print(Second)

# Saving the results in csv.
Second.to_csv('VarScan_Counts.csv', sep=',', index = None)

# Reading csv files and concatinating "CHROM" and "POS"
dff = pd.read_csv("VarScan_Counts.csv", sep = ',', index_col=
    False, error_bad_lines=False)
dff.columns = ['REF', 'ALT', 'VarScan_One', 'VarScan_Two',
    'VarScan_Three']

# set width of bar
width = 0.25

# Columns from the file
a1 = dff.VarScan_One.to_list()
a2 = dff.VarScan_Two.to_list()
a3 = dff.VarScan_Three.to_list()

# Set position of bar on X axis
r1 = np.arange(len(a1))
r2 = [x + width for x in r1]
r3 = [x + width for x in r2]

# Make the plot
plt.bar(r1, a1, color='#FFD700', width=width, edgecolor='white',
    label='VarScan_0.3')
```

```
plt.bar(r2, a2, color='#FFA500', width=width, edgecolor='white',
    label='VarScan_0.5')
plt.bar(r3, a3, color='#DC143C', width=width, edgecolor='white',
    label='VarScan_0.7')

# Add xticks on the middle of the group bars
plt.xlabel('Combinations')
plt.xticks([r + width for r in range(len(a1))], ['AA', 'AT', 'AG',
    'AC', 'TT', 'TA', 'TG', 'TC', 'GG', 'GA', 'GT', 'GC', 'CC',
    'CA', 'CT', 'CG'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('VarScan_Counts_Plot.pdf')
plt.savefig('VarScan_Counts_Plot.png', dpi = 300)
```

8.5.2 Comparison

```
# Importing the needed packages.
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import rc
matplotlib.rcParams['mathtext.fontset'] = 'cm'
matplotlib.rcParams['font.family'] = 'serif'
import matplotlib.pyplot as plt
import pandas as pd
import csv
import numpy as np

# Reading the csv input file that is obtained after performing the
    following operations on the vcf file.
# Step 1 - 'cut -f 1-2,4-5 Input.vcf > Output.vcf'
# Step 2 - 'sed '/^#/d' Output.vcf > Updated.vcf'

# The first step is to selected the needed columns in the vcf file.
# The second step if to eliminate all lines that start with a '#'
```

```
dff = pd.read_csv("Updated_Strelka_0.3_SNV.vcf", sep = '\t',
    index_col= False)
dff1 = pd.read_csv("Updated_VarScan_0.3_SNP.vcf", sep = '\t',
    index_col= False)
dff2 = pd.read_csv("Updated_Somatic_Truth_SNP.vcf", sep = '\t',
    index_col= False)

# SNP Counts
SSC = len(dff)
VSC = len(dff1)
TSC = len(dff2)

# Naming the columns after importing the csv file.
dff.columns = ['CHROM', 'POS', 'REF', 'ALT']
dff1.columns = ['CHROM', 'POS', 'REF', 'ALT']
dff2.columns = ['CHROM', 'POS', 'REF', 'ALT']

# Concatinating the "CHROM" and "POS"
dff["CHROM_POS"] = dff['CHROM'].astype(str) + '-' +
    dff['POS'].astype(str)
dff1["CHROM_POS"] = dff1['CHROM'].astype(str) + '-' +
    dff1['POS'].astype(str)
dff2["CHROM_POS"] = dff2['CHROM'].astype(str) + '-' +
    dff2['POS'].astype(str)

# Dropping of the unnecessary columns and reorganising the columns.
dff = dff.drop(['CHROM', 'POS'], axis=1)
cols = dff.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff = dff[cols]
print(dff)

dff1 = dff1.drop(['CHROM', 'POS'], axis=1)
cols = dff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dff1 = dff1[cols]
print(dff1)

dff2 = dff2.drop(['CHROM', 'POS'], axis=1)
cols = dff2.columns.tolist()
```

```
cols = cols[-1:] + cols[:-1]
dff2 = dff2[cols]
print(dff2)

# Mentioning the column names and inputing the csv file.
dff.columns = ['CHROM_POS', 'REF', 'ALT']
dff1.columns = ['CHROM_POS', 'REF', 'ALT']
dff2.columns = ['CHROM_POS', 'REF', 'ALT']

# Printing the list.
dff4 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'A')]
dff5 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'A')]
dff6 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'A')]
AA1 = len(dff4.index)
AA2 = len(dff5.index)
AA3 = len(dff6.index)

dff7 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'T')]
dff8 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'T')]
dff9 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'T')]
AT1 = (len(dff7.index)/SSC) * 100
AT2 = (len(dff8.index)/VSC) * 100
AT3 = (len(dff9.index)/TSC) * 100

dff10 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'G')]
dff11 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'G')]
dff12 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'G')]
AG1 = (len(dff10.index)/SSC) * 100
AG2 = (len(dff11.index)/VSC) * 100
AG3 = (len(dff12.index)/TSC) * 100

dff13 = dff[(dff['REF'] == 'A') & (dff['ALT'] == 'C')]
dff14 = dff1[(dff1['REF'] == 'A') & (dff1['ALT'] == 'C')]
dff15 = dff2[(dff2['REF'] == 'A') & (dff2['ALT'] == 'C')]
AC1 = (len(dff13.index)/SSC) * 100
AC2 = (len(dff14.index)/VSC) * 100
AC3 = (len(dff15.index)/TSC) * 100

dff16 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'T')]
dff17 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'T')]
```

```
dff18 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'T')]
TT1 = len(dff16.index)
TT2 = len(dff17.index)
TT3 = len(dff18.index)

dff19 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'A')]
dff20 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'A')]
dff21 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'A')]
TA1 = (len(dff19.index)/SSC) * 100
TA2 = (len(dff20.index)/VSC) * 100
TA3 = (len(dff21.index)/TSC) * 100

dff22 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'G')]
dff23 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'G')]
dff24 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'G')]
TG1 = (len(dff22.index)/SSC) * 100
TG2 = (len(dff23.index)/VSC) * 100
TG3 = (len(dff24.index)/TSC) * 100

dff25 = dff[(dff['REF'] == 'T') & (dff['ALT'] == 'C')]
dff26 = dff1[(dff1['REF'] == 'T') & (dff1['ALT'] == 'C')]
dff27 = dff2[(dff2['REF'] == 'T') & (dff2['ALT'] == 'C')]
TC1 = (len(dff25.index)/SSC) * 100
TC2 = (len(dff26.index)/VSC) * 100
TC3 = (len(dff27.index)/TSC) * 100

dff28 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'G')]
dff29 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'G')]
dff30 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'G')]
GG1 = len(dff28.index)
GG2 = len(dff29.index)
GG3 = len(dff30.index)

dff31 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'A')]
dff32 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'A')]
dff33 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'A')]
GA1 = (len(dff31.index)/SSC) * 100
GA2 = (len(dff32.index)/VSC) * 100
GA3 = (len(dff33.index)/TSC) * 100
```

```
dff34 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'T')]
dff35 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'T')]
dff36 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'T')]
GT1 = (len(dff34.index)/SSC) * 100
GT2 = (len(dff35.index)/VSC) * 100
GT3 = (len(dff36.index)/TSC) * 100

dff37 = dff[(dff['REF'] == 'G') & (dff['ALT'] == 'C')]
dff38 = dff1[(dff1['REF'] == 'G') & (dff1['ALT'] == 'C')]
dff39 = dff2[(dff2['REF'] == 'G') & (dff2['ALT'] == 'C')]
GC1 = (len(dff37.index)/SSC) * 100
GC2 = (len(dff38.index)/VSC) * 100
GC3 = (len(dff39.index)/TSC) * 100

dff40 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'C')]
dff41 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'C')]
dff42 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'C')]
CC1 = len(dff40.index)
CC2 = len(dff41.index)
CC3 = len(dff42.index)

dff43 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'A')]
dff44 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'A')]
dff45 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'A')]
CA1 = (len(dff43.index)/SSC) * 100
CA2 = (len(dff44.index)/VSC) * 100
CA3 = (len(dff45.index)/TSC) * 100

dff46 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'T')]
dff47 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'T')]
dff48 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'T')]
CT1 = (len(dff46.index)/SSC) * 100
CT2 = (len(dff47.index)/VSC) * 100
CT3 = (len(dff48.index)/TSC) * 100

dff49 = dff[(dff['REF'] == 'C') & (dff['ALT'] == 'G')]
dff50 = dff1[(dff1['REF'] == 'C') & (dff1['ALT'] == 'G')]
dff51 = dff2[(dff2['REF'] == 'C') & (dff2['ALT'] == 'G')]
CG1 = (len(dff49.index)/SSC) * 100
CG2 = (len(dff50.index)/VSC) * 100
```

```
CG3 = (len(dff51.index)/TSC) * 100

# Delcaring a new dataframe.
df = []
df1 = []
df2 = []

# Taking all combinations as a list.
data = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'Strelka_0.3': [AA1, AT1, AG1, AC1, TT1, TA1, TG1, TC1, GG1,
    GA1, GT1, GC1, CC1, CA1, CT1, CG1]}
data1 = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'VarScan_0.3': [AA2, AT2, AG2, AC2, TT2, TA2, TG2, TC2, GG2,
    GA2, GT2, GC2, CC2, CA2, CT2, CG2]}
data2 = {'REF': ['A', 'A', 'A', 'A', 'T', 'T', 'T', 'T', 'G', 'G',
    'G', 'G', 'C', 'C', 'C', 'C'], 'ALT': ['A', 'T', 'G', 'C', 'T',
    'A', 'G', 'C', 'G', 'A', 'T', 'C', 'C', 'A', 'T', 'G'],
    'Truth_Data': [AA3, AT3, AG3, AC3, TT3, TA3, TG3, TC3, GG3,
    GA3, GT3, GC3, CC3, CA3, CT3, CG3]}

# Collecting it into a dataframe.
df = pd.DataFrame(data)
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Merging columns based on "CHROM-POS"
First = pd.merge(df, df1, on=['ALT', 'REF'])
Second = pd.merge(First, df2, on=['ALT', 'REF'])

# Mentioning the column names and inputing the csv file.
Second.columns = ['REF', 'ALT', 'Strelka_0.3', 'VarScan_0.3',
    'Truth_Data']
print(Second)

# Saving the results in csv.
Second.to_csv('Tumor_Purity_0.3_Counts.csv', sep=',', index = None)
```

```
# Reading csv files and concatinating "CHROM" and "POS"
dff = pd.read_csv("Tumor_Purity_0.3_Counts.csv", sep = ',',
    index_col= False, error_bad_lines=False)
dff.columns = ['REF', 'ALT', 'Strelka', 'VarScan', 'Truth']

# set width of bar
width = 0.25

# Columns from the file
a1 = dff.Strelka.to_list()
a2 = dff.VarScan.to_list()
a3 = dff.Truth.to_list()

# Set position of bar on X axis
r1 = np.arange(len(a1))
r2 = [x + width for x in r1]
r3 = [x + width for x in r2]

# Make the plot
plt.bar(r1, a1, color='#FFD700', width=width, edgecolor='white',
    label='Strelka_0.3')
plt.bar(r2, a2, color='#FFA500', width=width, edgecolor='white',
    label='VarScan_0.3')
plt.bar(r3, a3, color='#DC143C', width=width, edgecolor='white',
    label='Truth_Data')

# Add xticks on the middle of the group bars
plt.xlabel('SNP Combinations')
plt.xticks([r + width for r in range(len(a1))], ['AA', 'AT', 'AG',
    'AC', 'TT', 'TA', 'TG', 'TC', 'GG', 'GA', 'GT', 'GC', 'CC',
    'CA', 'CT', 'CG'])

# Create legend & Show graphic
plt.legend()
plt.show()
plt.savefig('Tumor_Purity_0.3_Plot.pdf')
plt.savefig('Tumor_Purity_0.3_Plot.png', dpi = 300)
```