Masters Thesis on
Benchmarking Somatic Variant Callers

By Ravi Shankar Chintalapati

Examiners: Prof. Dr. Rolf Backofen, Prof. Dr. Dr. Melanie Börries
Advisers: Dr. Wolfgang Maier, Dr. Mehmet Tekman

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Bioinformatics

29th April 2021

Writing period
29.10.2020 – 29.04.2021

Examiners
Prof. Dr. Rolf Backofen
Department of Bioinformatics
University of Freiburg

Prof. Dr. Dr. Melanie Börries
Institute of Medical Bioinformatics and Systems Medicine
Faculty of Medicine, Freiburg University

Advisers
Dr. Wolfgang Maier, Dr. Mehmet Tekman

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date

Signature

Abstract

Variant calling pipelines consists of two main types, aimed at quantifying different types of variants namely Germline (inheritable variants) and Somatic (uninheritable variants). In terms of benchmarking, many Germline variant calling pipelines were compared and documented by the standards of the Genome in a Bottle (GIAB) Consortium [1]. For Somatic variant calling pipelines, only a few comparisons were achieved because the comparisons are more complex and less well-established. In the diploid human genome, a variant can be found to be either homozygous or heterozygous or not at all. Tumors, on the other hand, are inhomogeneous cells that possibly carry variants with rare mutations thereby making the Somatic variant calling pipelines benchmarking challenging. Due to the diverse natures of these two types of analysis, the goal of this thesis is to benchmark Somatic variant callers by selecting few datasets, in order to establish a reliable variant caller for cancer research.

Zusammenfassung

Varianten-Anzeiger-Pipelines bestehen aus zwei Haupttypen, die darauf abzielen, verschiedene Arten von Varianten zu quantifizieren, namentlich Keimbahnen (vererbbare Varianten) und somatische Varianten (nicht vererbbare Varianten). In Bezug auf das Benchmarking wurden viele Keimbahn-Varianten-Anzeiger-Pipelines nach den Standards des Genome in a Bottle (GIAB)-Konsortiums verglichen und dokumentiert [1]. Für somatische Varianten-Anzeiger-Pipelines wurden nur wenige Vergleiche durchgeführt, da die Vergleiche komplexer und weniger gut erforscht sind. Im diploiden menschlichen Genom kann eine Variante entweder homozygot oder heterozygot sein oder überhaupt nicht existieren. Tumore hingegen sind inhomogene Zellen, die möglicherweise Varianten mit seltenen Mutationen tragen, was das Benchmarking der somatischen Varianten-Anzeige-Pipelines zu einer Herausforderung macht. Aufgrund der unterschiedlichen Natur dieser beiden Analysetypen ist es das Ziel dieser Arbeit, somatische Varianten-Anzeiger durch die Auswahl weniger Datensätze zu benchmarken, um einen zuverlässigen Varianten-Anzeiger für die Krebsforschung zu finden.

Contents

List of Tables

List of Figures

9. Figure 6.6 shows (a) Percentage of Strelka allele frequency counts with Contamination tolerance 0.3 (b) Percentage of Strelka allele frequency counts with Contamination tolerance of 0.5 (c) Percentage of Strelka allele frequency counts with Contamination tolerance of 0.7

10. Figure 6.7 shows (a) Percentage of VarScan allele frequency with Tumor purity of 0.3 (b) Percentage of VarScan allele frequency with Tumor purity of 0.5 (c) Percentage of VarScan allele frequency with Tumor purity of 0.7

11. Figure 6.8 shows (a) Percentage of filtered reads in Strelka with Contamination tolerance of 0.3 (b) Percentage of filtered reads in Strelka with Contamination tolerance of 0.5 (c) Percentage of filtered reads in Strelka with Contamination tolerance of 0.7

12. Figure 6.9 shows (a) Percentage of filtered reads in VarScan with Tumor purity of 0.3 (b) Percentage of filtered reads in VarScan with Tumor purity of 0.5 (c) Percentage of filtered reads in VarScan with Tumor purity of 0.7.

List of Abbreviations

1. GIAB - Genome in a Bottle
2. INDEL - INsertion/DELetion polymorphism
3. MIRACUM - Medical Informatics in Research and Care in University Medicine
4. SNP - Single Nucleotide Polymorphisms
5. TCGA - The Cancer Genome Atlas
6. VCF - Variant Calling Format
7. WES - Whole Exome Sequencing
8. WGS - Whole Genome Sequencing

Chapter 1

Introduction

A variant is a type of mutation that differs in some respect from a reference genome/transcriptome standard, where a variant in bioinformatics is an alteration in the most common DNA/RNA nucleotide sequence. It is defined based on the type of DNA/RNA error and can be used to describe an alternation that may be benign, pathogenic or of unknown significance.

Based on the way they occur, variants are of two types i.e. Germline variants and Somatic variants. Germline variants are a gene change in the egg or sperm that are incorporated into the DNA of every cell in the offspring from the parent. These hereditary variants are even referred to as Germline mutations. Somatic variants are alterations in the DNA that occur after conception and can occur in any cell of the body except the germ cells i.e. egg and sperm cells. Therefore, Somatic variants cannot be passed on to children.

The process to identify the variants is called Variant Calling and the variants are identified from the sequence data obtained through sequencing [2]. Sequencing is the operation of determining the precise order of a chain of four bases Adenine (A), Guanine (G), Cytosine (C) and Thymine (T) in a DNA strand.

In terms of sequencing, there is Whole Genome Sequencing (WGS) [3] or Whole Exome Sequencing (WES) [4]. In WGS, also known as full genome sequencing, the DNA sequence of an entire organism's genome is determined at a single time. WES, also known as Exome Sequencing, is a genomic technique for sequencing all of the protein-coding regions of genes in a genome also known as exomes.

Through one of the many different sequencing technologies, a biological sequence and its corresponding quality scores are obtained as FASTQ files, a text-based format for storing both the values. These FASTQ files [5] are then aligned based on a reference genome thereby creating a BAM file [6] which is a compressed binary version of a SAM file or the human-readable text file with biological sequences aligned to a reference sequence.

Using the BAM files, differences between the aligned reads and the reference genome can be written into a Variant Call Format (VCF) file [7]. This process of identifying variants from the sequence data is called Variant Calling and this process is different for both Germline and Somatic variant calling.

In the Germline variant calling, the reference genome is a golden standard sequence for the species of interest and the genomes are diploid. So at any given locus, either all reads have the same base, indicating homozygosity or approximately half the reads have one base and the other half have another indicating heterozygosity with the only exception being the sex chromosomes in the male mammals. In Somatic variant calling, the reference is tissue from the same individual with mosaicism between the cells, where mosaicism is when a person has two or more genetically different sets of cells in their body.

The goal of this thesis is to benchmark Somatic variant callers based on their ability to identify cancer-causing variants. To perform the comparison, we require the Truth Data to which the variant calling outcomes i.e. the VCF files compared, are know. We, therefore, consider the artificial dataset pair for Somatic variant calling from the GIAB project[1] as the Truth Data and based on the comparisons, the effectiveness of the Somatic variant callers is determined.

There are several variant callers for both Germline and Somatic variant calling. Some commonly-used Germline variant callers are FreeBayes, Strelka2, VarScan, and Beagle. Similarly, some common Somatic variant callers are LoFreq, MuSE, MuTect2, SomaticSniper, Strelka, VarScan and

[1] GIAB project is available at ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/use_cases/mixtures/UMCUTRECHT_NA12878_NA24385_mixture_10052016/

VarDict. The choice of the variant callers that are benchmarked in this thesis is dependent on work done by Dr. Wolfgang Maier and Dr. Björn Grüning, with data from Use Case 3 of the German MIRACUM initiative [8].

The Medical Informatics in Research and Care in University Medicine (MIRACUM) initiative has three use cases and the referred third use case is "From Knowledge to Action - Support for Molecular Tumor Boards" [8]. A tumor board is a group of doctors and health care providers with different specialities meeting regularly to discuss cancer cases and share knowledge [9]. In contrast to traditional cancer tumor boards, molecular tumor boards are made up of cancer experts across specialities as well as researchers with expertise on a variety of cancer types, gene sequencing technologies and genomic data, who work together to make informed treatment decisions [10].

The MIRACUM consortium aims to support Molecular Tumor Boards with innovative IT solutions by improving the complex processes of quality assurance, data preparation, data analysis, data integration and information retrieval between genetic high-throughput analysis and medical therapy decisions. Additionally, clinicians will be offered decision support through efficient data visualization [8].

Using Galaxy, an open web-based platform that provides accessibility, reproducibility, and transparency, Dr. Wolfgang Maier built two workflows that are supported by the MIRACUM partners [11]. A workflow is a series of tools and dataset actions that run in a sequence as a batch operation. The first workflow is the WES Tumor/Normal sample pair analysis and the second is gene panel data with only the Tumor sample.

The major difference between the two workflows is the variant calling software used in them. For the first workflow or the WES Tumor/Normal sample pair analysis workflow, VarScan Somatic is used and for the second workflow or the gene panel data workflow, LoFreq Call is used.

For this thesis, the WES Tumor/Normal sample pair analysis workflow is used to obtain VCF files through VarScan Somatic variant caller. Aside from this, the Strelka Somatic variant caller is also added to the workflow using Galaxy and the outcomes from both the variant callers are compared to the artificial Truth Data considered from the GIAB FTP site to benchmark.

This report comprises a total of six sections and is outlined as follows. The first section is ‘Introduction’ dealing with the basics needed to approach the thesis. The second section is ‘Literature Study’ encompassing articles, research papers, tools and websites that helped the thesis. The third section is ‘Background’ dealing with different workflows, information about the input data and the sequence of workflow executions. The fourth section is ‘Approach’ dealing each step starting from writing YMAL files, using Planemo, understanding the tools, reading VCF files etc.

The fifth section is ‘Experiments’ dealing with different comparisons, bias, observations and benchmarking methods. The sixth section is ‘Results’ dealing with the outcomes of the experiments in terms of positions, SNPs, allele frequencies, read depths, variant combinations, and INDELs. Lastly, the seventh section is ‘Conclusion’ dealing with the choice of the variant caller based on the results and scope for future work.

Chapter 2

Literature Study

The first section 'Background Study' deals with comprehending the scope of the topic. In the second section, the platform needed to execute the goal i.e. Galaxy [12] and its concepts are dealt with. In the third section, tasks and tools used based on the information obtained from the Galaxy are mentioned and in the last section, research papers related to the thesis are presented.

2.1 Background Study

To begin with, an understanding of the variant calling was developed using the material "Galaxy Training: Introduction to Variant Calling" [13] and the difference between the mutations in Germline and Somatic were comprehended using the resource from BioNinja[1]. In-order to upload and update the progress of the thesis, GitHub was used and as an introduction to the operations as to how to use GitHub, the "Version Control with Git and GitHub" [14] paper was quite useful.

To comprehend information about the third use case of which the thesis is a part, as well as to gain access to the workflows that are used for the variant calling based on MIRACUM [8], the MIRACUM Pipe Galaxy repository [11] was used. Of the two workflows in the MIRACUM Pipe Galaxy repository, the first workflow i.e. WES Tumor/Normal sample pair analysis workflow [11] is selected for the Somatic variant calling using VarScan Somatic and

[1] https://ib.bioninja.com.au/standard-level/topic-3-genetics/33-meiosis/somatic-vs-germline-mutatio.html

Strelka Somatic variant callers.

To obtain the input data i.e. the forward reads, reverse reads, bed file, and Truth Data VCF file for the selected WES Tumor/Normal sample pair analysis workflow, access the GIAB FTP[2] resource. For benchmarking the variant calling outcomes, use the Genome In A Bottle (GIAB) consortium standards material [15] as a reference along with the Germline Benchmarking Tools and Standards [16].

2.2 Galaxy

In Galaxy, numerous training materials and tutorials including the variant calling are a part of the "Galaxy Training" material [17]. So to understand the tools used in exome sequencing, the "Exome sequencing data analysis for diagnosing a genetic disease" material [18][19] was used. Similarly, for an introduction to the Somatic Variant Calling, the "Identification of Somatic and Germline Variants from Tumor and Normal Sample Pairs" material [19][20] was used.

To execute the workflows from the command line, a command-line utility resource of the Galaxy named 'Planemo version 0.73.0.dev0' [21] was used. This resource which helps in developing tools, workflows and training materials for Galaxy is also helpful to execute workflows through the command line. To execute a Galaxy workflow through Planemo, a YAML file [22] is to be created and then using Planemo and a few other details explained in the fourth chapter "Approach", the workflows are executed remotely.

2.3 Tools

After executing the WES Tumor/Normal sample pair analysis workflow and obtaining the Strelka and VarScan VCF files, there are three tools to analyze, summarize and compare the VCF files. The first tool named 'vcftools version0.1.15' [23] can be used to extract SNPs or INDELs, calculate allele

[2] ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/use_cases/mixtures/UMCUTRECHT_NA12878_NA24385_mixture_10052016/

frequencies, and summarize data in a VCF file. The second tool named 'vcf-stats version 0.0.6' [24] can be used to plot VCF data with specific metrics and focus on variants with certain filters. The third tool named 'vcftoolz version 1.2.0' [25] can be used to compare the number of SNPs and positions in two or three VCF files.

An exception in calculating allele frequency using vcftools is the Strelka VCF file. To calculate the allele frequencies in the Strelka VCF file, resources from 'Strelka User Guide - Last updated on Nov 1st, 2018' [26] were used and for operations on VCF files such as filtering rows or columns or conditional filtering etc, 'Pandas version 1.2.4' [27] was used.

2.4 Research Papers

An understanding about the VarScan Somatic and Strelka Somatic variant callers was developed using the resources "VarScan 2: Somatic mutation and copy number alterations discovery in cancer by exome sequencing" [28] and "Strelka: accurate somatic small-variant calling from sequenced tumor-normal sample pairs" [29]. Apart from the introduction to the variant callers, through the Strelka2 paper, an introduction to the concept of normal sample contamination model and the terms to parameterize the Strelka Somatic variant caller in Galaxy were learnt.

As an introduction to the comparison of Somatic variant callers, the Lancet Somatic variant caller for short-read data is compared with the other Somatic variant callers in the paper "Genome-wide somatic variant calling using localized colored de Bruijn graphs" by Giuseppe Narzisi et all [30]. Through this paper, an introduction to the concepts of the variant scoring system, low-frequency mutation detection and the relation between sensitivity and benchmarking were acquired.

To further understand the contamination of the Normal samples, the "DeTiN: Overcoming Tumor in Normal Contamination" by Amaro Taylor-Weiner et all [31] was referred. Through this paper, a use case as to how to improve the sensitivity by detecting the Normal samples contaminated with Tumor cells was acquired.

To understand the different factors involved to compare the variant callers, the "Systematic comparison of germline variant calling pipelines cross multiple next-generation sequencers" by Jiayun Chen, et al [32] was referred. In this paper, three variant calling pipelines HaplotypeCaller, Strelka2 and Samtools-Varscan2 were analysed and through this paper, an introduction to the concepts of F-scores, performance in INDEL and SNP calling, variant calling period in different coverages were learnt.

Finally, as a reference for Somatic variant calling benchmarking, the "Comparing somatic mutation-callers: beyond Venn diagrams" by Su Yeon Kim & Terence P Speed [33] was referred. Through this paper, an introduction to the studies revealed by The Cancer Genome Atlas (TCGA) and guidelines to compare outputs from multiple callers was learnt.

Chapter 3

Background

The VCF files that are to be benchmarked in comparison to the Truth Data are an outcome of the WES Analysis workflow. However, there are two more Galaxy workflows involved to calculate the subsampled data that is provided as an input to the WES Analysis workflow. These two workflows are the Map and Filter workflows. Of all the three workflows, the WES Analysis workflow is supported by the MIRACUM partners and was built and parameterised by Dr. Wolfgang Maier in the MIRACUM Pipe Galaxy repository [11].

The Map workflow is a sub-workflow of the WES Analysis workflow used to calculate the mapped reads for both Normal and Tumor samples and the Filter workflow built by Dr. Mehmet Tekman uses these mapped reads and capture regions to subsample the input data. The reason for these two additional workflows instead of the WES Analysis workflow is due to the input size. Using the Map and Filter workflows, the input size is decreased by only selecting the mapped reads in capture regions. This operation decreases the input size extensively thereby decreasing the time for executing the WES Analysis workflow and obtaining the Strelka and VarScan VCF files.

3.1 Map Workflow

In the Map workflow, there are three sections. The first section is about the inputs i.e. the forward and reverse reads, the second section is about trimming the unwanted reads and the third section is about mapping the trimmed reads to the reference reads.

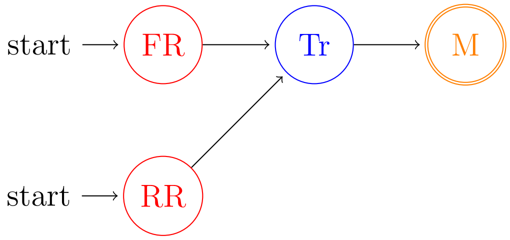


Figure 3.1: Sequence of operations in Map workflow.

In the input section, for the Somatic variant calling, there are two types of samples i.e Normal and Tumor samples. Therefore, there are Normal forward reads, Normal reverse reads, Tumor forward reads and Tumor reverse reads. In the flowchart, the Forward Reads are represented by FR and the Reverse Reads are represented by RR for both Normal and Tumor samples.

In the second section, forward and reverse reads of both Normal and Tumor samples are trimmed based on the minimum quality per base, minimum length of the reads and a few other conditions. Using the Galaxy tool, ‘Trimmomatic 0.36.5’, either the paired-end or the single-ended reads are trimmed. In the flowchart, Tr represents all the trimming operations.

In the third section, the trimmed forward and reverse reads of Normal and Tumor samples are mapped with the reference genome i.e. Human Feb. 2009 (GRCh37/hg19)(hg19). Using the Galaxy tool, ‘Map with BWA-MEM 0.7.17.1’, Normal forward and reverse reads are mapped into ‘Mapped Normal forward reads’ and ‘Mapped Normal reverse reads’. Similarly, the Tumor forward and reverse reads are mapped into ‘Mapped Tumor forward reads’ and ‘Mapped Tumor reverse reads’. In the flowchart, M represents the mapping operation.

3.2 Filter Workflow

In the Filter workflow, there are three sections. The first section is about the inputs i.e. capture regions, forward, mapped forward, reverse and mapped reverse reads for both Normal and Tumor samples. The second section is

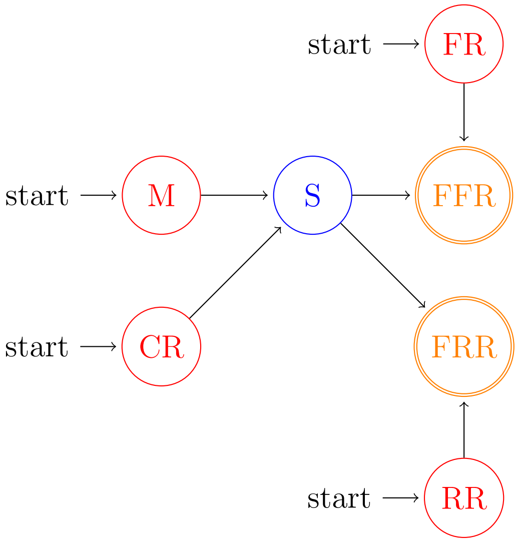


Figure 3.2: Sequence of operations in Filter workflow.

about filtering Mapped forward and Mapped reverse reads for both Normal and Tumor samples based on the capture regions and the third section, the forward and reverse reads for both Normal and Tumor samples are filtered based on their mapping with filtered Mapped forward and filtered Mapped reverse reads for both Normal and Tumor samples.

In the first section, the considered inputs are the outcome of the Map workflow or the mapped Normal and Tumor samples for forward and reverse reads, capture regions represented in a BED file, Normal forward reads, Normal reverse reads, Tumor forward reads and Tumor reverse reads. In the flowchart, Mapped forward and reverse reads for both Normal and Tumor samples are represented by M, the Capture Regions are represented by CR, the Forward Reads are represented by FR and the Reverse Reads are represented by RR for both Normal and Tumor samples.

In the second section, based on the capture regions, the mapped Normal and Tumor samples for forward and reverse reads are filtered using the Galaxy tool, ‘Samtools view 1.9+galaxy2’. This tool filters and subsamples alignments as per the user requirement and generates a BAM file.

In the third section, based on the BAM file from the Galaxy tool, 'Samtools view 1.9+galaxy2', the forward reads and reverse reads for both Normal and Tumor samples represented by FR and RR in the flowchart are mapped to create the Mapped Normal and Tumor samples for forward and reverse reads. Using the Galaxy tool, 'Filter Sequences by Mapping 0.0.6', the selected reads in the BAM file are mapped with forward and reverse reads for both Normal and Tumor samples. In the flowchart, these outcomes are represented by FFR for Filtered Forward Reads and FRR for Filtered Reverse Reads.

3.3 WES Analysis Workflow

In the WES Analysis workflow, there are five sections. The first section is about the inputs i.e. Normal filtered forward, Normal filtered reverse, Tumor filtered forward, and Tumor filtered reverse reads. The second section is about trimming the filtered reads, the third section is about mapping the trimmed filtered reads, the fourth section is about post-processing the mapped trimmed reads and the fifth section is about variant calling.

In the first section, the considered inputs are the outcome of the 'Filter workflow' or the Filtered Normal and Tumor samples for forward and reverse reads, sample names, and purity estimates. In the flowchart, Filtered Normal and Tumor samples for forward and reverse reads are represented by FFR and FRR, sample names are represented by NM, and purity estimates are represented by PR.

In the second section, Filtered forward and reverse reads for both normal and tumor samples are trimmed based on the minimum quality per base, minimum length of the reads and a few other conditions using the Galaxy tool, 'Trimmomatic 0.36.5'. During this operation, either the paired-end or the single-ended reads are trimmed. In the flowchart, Tr represents all the trimming operations.

In the third section, the trimmed Filtered forward and reverse reads for both Normal and Tumor samples are mapped with the reference genome i.e. Human Feb. 2009 (GRCh37/hg19)(hg19). Using the Galaxy tool, 'Map with BWA-MEM 0.7.17.1', trimmed filtered normal forward and reverse reads are

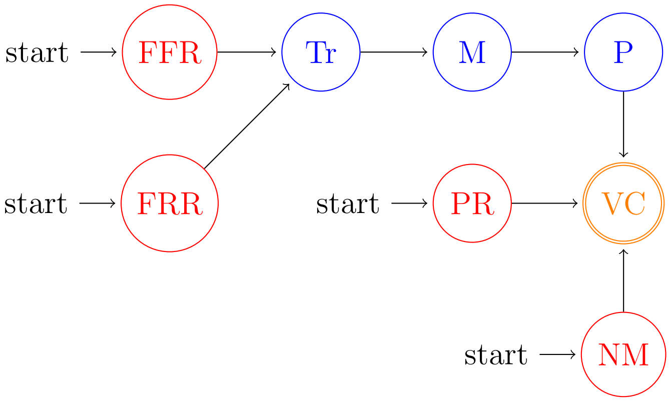


Figure 3.3: Sequence of operations in WES Analysis workflow.

mapped into ‘Filtered Mapped Normal reads’ and Tumor forward and reverse reads are mapped into ‘Filtered Mapped Tumor reads’. In the flowchart, M represents the mapping operation.

In the fourth section, the filtered mapped Normal reads and filtered mapped Tumor reads are processed to filter the paired-end reads of all samples to retain only those read pairs, for which both the forward and the reverse reads have been mapped to the reference successfully. In this process, data is deduplicated [18][19] and the outcome of this process is a BAM file represented by P for the Post-mapping operation in the flow chart.

In the fifth section, with the BAM input from the Post-mapping operation, sample names and purity estimates for both the Normal and Tumor samples, the variant calling is performed. The inherent choice of the variant caller in the WES workflow is a parameterised ‘VarScan Somatic 2.4.3.6’ and second selected choice is ‘Strelka Somatic 2.9.10+galaxy0’. With details of parameterisation completely mentioned in the fourth chapter “Approach” and “Appendix G”, the outcome of this section are the VCF files. In the flowchart, VC represents Variant Callers.

3.4 Truth Dataset

Apart from the workflows, the input data used for benchmarking is created from two ‘Genome in a Bottle 3.2.2’ [15] samples NA12878 and NA24385. Generated using a script from ‘HBC Project: GIAB Somatic’ repository [34], the input data is a mixture that simulates a Tumor and Normal cancer dataset for validation of low frequency calling by Somatic variant callers.

Out of the two samples, the NA12878 are Somatic variations and the shared NA12878/NA24385 are Germline variants in the background. The Normal forward and reverse fastq paired are from the NA24385 sample and the Tumor forward and reverse fastqs are generated by mixing 30% of NA12878 with 70% of NA24385. With data at this combination, Somatic variants are expected at 15% unique heterozygotes in NA12878 sample and 30% unique homozygotes in NA12878 sample [34].

Chapter 4

Approach

Starting from the input data until benchmarking the VCF files, there are five steps involved. The first step is obtaining the input data and the capture regions, the second step is executing the Map, Filter and WES Analysis workflows, the third step is parameterising the variant callers, the fourth step is ensuring consistency between the VCF files and selecting the necessary columns for operations and the fifth step is about performing operations like count, sum, comparisons, and variant combinations on the VCF files.

4.1 Inputs

The list of inputs needed to obtain the VCF files is based on the Map, Filter and WES Analysis workflows. For the Map workflow, the needed files are Normal forward reads, Normal reverse reads, Tumor forward reads, and Tumor reverse reads. For the Filter workflow, the needed files are Normal forward reads, Normal reverse reads, Tumor forward reads, Tumor reverse reads, Capture Regions, mapped Normal reads, and mapped Tumor reads.

For the WES Analysis workflow, the needed files are Normal sample name, Tumor sample name, Normal purity estimate, Tumor purity estimate, filtered mapped Normal forward reads, filtered mapped Normal reverse reads, filtered mapped Tumor forward reads, filtered mapped Tumor reverse reads and Truth Data VCF file.

The input for the Map workflow and the Truth Data can be downloaded from GIAB FTP[1] and the Capture Regions needed for the Filter workflow can be downloaded from MIRACUM Annotation Data[2]. Except for these inputs, the rest of the inputs are created while executing the workflows.

From the user end, Normal sample name, Tumor sample name, Normal purity estimate, Tumor purity estimate, Contamination tolerance in SNVs, Contamination tolerance in INDELs, Prior probability of Somatic SNVs, and Prior probability of Somatic INDELs are needed.

In this thesis, the Normal purity estimate is always 1.0 and the Tumor purity estimate which represents the percentage of cancer cells in a solid Tumor sample are 0.3, 0.5 and 0.7 in every experiment. The Contamination tolerance in SNVs and Contamination tolerance in INDELs are like the Tumor Purity in terms of the values assigned to them i.e. 0.3, 0.5 and 0.7 in every experiment. The Prior probability of Somatic SNV and Prior probability of Somatic INDEL are an arbitary choice.

4.2 Execution

The Map, Filter and WES Analysis workflows can be executed either through the command line or through Galaxy user interface. When executing through the command line, choose the Galaxy domain, set up the workflow, create the YAML file and then use the Planemo command given below to execute the workflow. Example YAML files for the three workflows can be accessed at Appendix A.

```
planemo run workflow.ga
params.yml
--galaxy_url https://usegalaxy.eu/
--galaxy_user_key APIKEY
--engine external_galaxy
--no_shed_install
```

[1] ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/use_cases/mixtures/UMCUTRECHT_NA12878_NA24385_mixture_10052016/

[2] https://usegalaxy.eu/u/wolfgang-maier/h/miracum-annotation-data

If using Galaxy user interface, import the workflow at usegalaxy.eu and run the workflow. In the three workflows, Map workflow should be executed first for the mapped forward and reverse reads for both Normal and Tumor samples. Using the Map workflow outcomes, Filter workflow should be executed to obtain mapped filtered forward and mapped filtered reverse reads for Normal and Tumor samples. Using the Filter workflow outcomes, the WES Analysis workflow should be executed. By using Galaxy, the outcomes of one workflow can be dragged and dropped into the new workflow without any processing time.

4.3 Variant Caller Parameters

In executing the WES Analysis workflow either through command-line or Galaxy, the parameterised VarScan Somatic variant caller in the WES Analysis workflow with the specifications mentioned in Appendix G is executed. As the second variant caller, Strelka Somatic replaces the VarScan Somatic in the WES Analysis workflow and Tumor Purity input is no longer applicable. Instead, different parameters are altered.

The main parameter that changes is the Contamination Tolerance value specified for SNVs in `Set ssnvContamTolerance` and for INDELs in `Set indelContamTolerance`. When executing the modified WES Analysis workflow with the Strelka Somatic specifications mentioned in Appendix G, alter the Contamination Tolerance to 0.3, 0.5 and 0.7 for SNVs and INDELs through YAML file or user-data.

The Contamination Tolerance terms were introduced to allow for contamination in the Normal sample by some fraction of Tumor cells. This is particularly useful for analyses of liquid Tumors, where the Normal sample may be contaminated by Tumor cells [29].

4.4 Columns in VCFs

After executing the workflows, three VCF outcomes from the Strelka Somatic and VarScan Somatic variant callers are obtained respectively and the columns in all of the VCFs are CHROM, POS, ID, REF, ALT, QUAL, FIL-

TER, INFO, FORMAT, NORMAL, and TUMOR.

CHROM is typically a chromosome, POS is the position of a variation in a sequence, ID is the identifier of the variation, REF is the reference base or bases, ALT is the list of alternative alleles, QUAL is the quality score of the alleles, FILTER indicates if a given set of filters passed or not, INFO and FORMAT columns have sub-fields that vary as per the variant caller and SAMPLEs column has NORMAL and TUMOR samples.

Of these columns, ID, QUAL, FILTER and INFO aren't used either in comparison or benchmarking. Of the other columns, CHROM column value is sometimes just a number i.e. 12 and sometimes a string concatenated with a number i.e. chr12. To compare the VCF files, the CHROM values in both of the files should be consistent. To add chr, use the command

```
awk ’{
if($0 !~ /^#/)
print "chr"$0;
else if(match($0,/(##contig=<ID=)(.*)/,m))
print m[1]"chr"m[2];
else print $0
}’ no_chr.vcf > with_chr.vcf
```

With the chr, the VCF files are consistent and can be compared. Using this CHROM column, reads can be filtered based on chromosomes using the command given below. Replace the chrno to the specific chromosome i.e. chr12 or chr14 or etc to execute the command

```
grep -w ’^#\|^chrno’ old_file.vcf > new_file.vcf
```

4.5 Operations

With consistent VCF files, three operations are to be performed to determine the choice of a variant caller for detecting cancer-causing variants. These three operations are comparisons, bias detection and benchmarking. However, before performing the operations, apart from the consistent VCF files, specific columns from the VCF files should be selected and header information in the VCF file should be removed.

The command to select a few columns is given below. To execute the command, replace the column_numbers with integers based on the column occurrence in the VCF files starting from 1 and separate the list with commas if there are couple of columns to be selected i.e 1-3,5.

```
cut -f column_numbers old_file.vcf > new_file.vcf
```

After selecting the needed columns, to perform operations on the selected VCF files, the header information before the data explaining the columns aren’t needed. Therefore, use the command given below to delete the header information

```
sed ’/^#/d’ old_file.vcf > new_file.vcf
```

With the VCF files with selected columns or all columns and no header information, the experiments on the VCF files can be performed.

Chapter 5

Experiments

For comparison, bias detection and benchmarking of VCF files, five experiments are performed. The first experiment is comparing the positions, SNPs and Indels between the VCF files. The second experiment is comparing the allele frequencies between the VCF files. The third experiment is comparing the read depths between the VCF files. The fourth experiment is searching for the bias in the sixteen variant combinations between the VCF files. The fifth experiment is benchmarking the VCF files in comparison to the Truth Data. Based on the conclusions drawn from the results of these experiments, the choice of the variant caller between Strelka Somatic and VarScan Somatic is determined for detecting cancer-causing variants.

5.1 Positions, SNPs, & INDELs

A Single-Nucleotide Polymorphism (SNP) 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 and an INDEL is a molecular biology term for an INsertion or DELetion of bases in the genome of an organism. In the first experiment, positions, SNPs and Indels are compared between the variant caller outcomes and the Truth Data.

This comparison between two or three VCF files can be achieved through the VCFToolz command

```
vcftoolz compare file1.vcf file2.vcf file3.vcf > output.txt
```

This tool notes the differences in a text file alongside plotting Venn diagrams showing the count of the common and mutually exclusive positions and SNPs. However, in the outcome, the number of SNPs in comparison to the number of positions aren't consistent. At least for the input files considered for the thesis and the INDELs aren't even compared by this tool.

So by using the code in Appendix D, positions, SNPs and INDELs are compared. To execute the code in Appendix D, files with only SNPs and files with only INDELs are needed. To obtain the file with only SNPs, consider the VCF file as an input and use the command

```
vcftools --vcf input_file.vcf --remove-indels --recode
    --recode-INFO-all --out SNPs_only
```

To obtain the file with only INDELs, consider the VCF file as an input and use the command

```
vcftools --vcf input_file.vcf --keep-only-indels --recode
    --recode-INFO-all --out INDELs_only
```

Using these the VCF, SNP, INDEL files and the code in Appendix D which uses the `len` and `merge` operations along with `matplotlib` [35] package, the comparison of the positions, SNPs and INDELs in terms of their `count` are listed in tables. The outcomes are even represented graphically as Venn diagrams and are classified based on the Tumor purity or Contamination Tolerance of 0.3, 0.5, and 0.7.

5.2 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 percentage. In the second experiment, the number of positions within specific allele frequencies limits is tabulated and plotted individually per variant caller using the code in Appendix B.

The calculation of the allele frequency varies for VarScan Somatic and Strelka Somatic variant callers. For VarScan Somatic and even the Truth Data VCF file, the allele frequency values are a part of the obtained VCF files. So by using the vcftools, allele frequency for REF and ALT values can be obtained by using the command

```
vcftools --vcf input.vcf --freq --out output
```

For Strelka Somatic, the allele frequency values are not a part of the obtained VCF file. So vcftools cannot be used. Instead, the code in the Strelka Allele Frequency section of Appendix B should be used to calculate the allele frequencies for REF and ALT values. In Strelka Somatic, the allele frequency calculation varies between SNPs and INDELs. The formulas and further documentation can be accessed at Strelka User Guide [26].

To calculate the Strelka Somatic SNPs allele frequency, the formula is

```
refCounts =
Value of FORMAT column $REF + "U" (e.g. if REF="A" then use the
    value in FOMRAT/AU)

altCounts =
Value of FORMAT column $ALT + "U" (e.g. if ALT="T" then use the
    value in FOMRAT/TU)

tier1RefCounts = First comma-delimited value from $refCounts
tier1AltCounts = First comma-delimited value from $altCounts

Somatic allele frequency =
$tier1AltCounts / ($tier1AltCounts + $tier1RefCounts)
```

To calculate the Strelka Somatic INDELs allele frequency, the formula is

```
tier1RefCounts = First comma-delimited value from FORMAT/TAR
tier1AltCounts = First comma-delimited value from FORMAT/TIR

Somatic allele freqeuncy =
$tier1AltCounts / ($tier1AltCounts + $tier1RefCounts)
```

With the allele frequencies of VarScan Somatic, Strelka Somatic, and Truth Data obtained, the values can be classified into four categories.

The first category is the number of positions with less than 0.25 allele frequency value. The second category is the number of positions with allele frequency values between 0.26 and 0.50. The third category is the number of positions with allele frequency values between 0.51 and 0.75 and the fourth category is the number of positions with allele frequency values between 0.76 to 1.00. Based on these four categories, a comparison can be made.

5.3 Read Depths

Read Depth describes the number of times that a given nucleotide in the genome has been read in an experiment and it is also referred to as coverage. A base with 30 read depth or 30x coverage means on average each base has been read by 30 sequences. The third experiment would be comparing the count of a range of read depths between the variant caller outcomes and the Truth Data using the code in Appendix E.

To calculate the summary statistics and site depth of VCF files, vcftools could be used. The command to obtain the summary statistic is

```
vcftools --vcf input_data.vcf --depth -c > depth_summary.txt
```

The command to calculate the site depth per base is

```
vcftools --vcf input_data.vcf --site-depth --max-missing 1.0 --out
    site_depth_summary
```

However, using neither of the commands, read depth per base or position cannot be obtained. To obtain these values, use the code in Appendix E. In VCF files, read depth is often represented by DP and it is a part of the FORMAT column. For Somatic variant calling, there are both Normal Read Depth and Tumor Read Depth.

For Strelka Somatic, the FORMAT column is represented as DP:FDP:SDP: SUBDP:AU:CU:GU:TU of which DP is the read depth. For VarScan Somatic, the FORMAT column is represented as GT:GQ:DP:AD:ADF:ADR of which DP is the read depth. For Truth Data, the FORMAT column is represented as GT:PS:DP:GQ of which DP is the read depth.

After filtering out the DP values, calculate and compare the minimum value, maximum value, standard deviation, mean of the read depths for Normal and Tumor samples for Strelka, VarScan and Truth Data. This comparison reveals the coverage of the variant callers and the results are documented in a tabular form.

Then by using the mean and standard deviation, a range for read depth that has good coverage for reads is determined and the positions which are between this range are selected and filtered. These filtered positions in comparison to the total number of positions for each variant caller help determine the choice of the variant caller.

5.4 Variant Combination Bias

The REF and ALT columns in VCF files represent the allele in the reference genome and any other allele found at that locus respectively. When only SNPs are considered, then REF and ALT only have four possible bases to occur i.e. A, T, G, C. If the REF and ALT are concatenated, there are in total sixteen combinations that could occur i.e. AA, AT, AG, AC, TA, TT, TG, TC, GA, GT, GG, GC, CA, CT, CG, CC.

The fourth experiment is to find the bias of variant callers in calling a few variant combinations more often than the others in comparison to the Truth Data. Using the code in Appendix F, the count for each of these sixteen combinations from the VCF files are calculated and normalised by division with the total count. The normalization is to calculate the percentage of a variant occurrence in all of the combinations.

Out of the sixteen combinations, AA, TT, GG and CC occur zero times as the variant callers do not add those positions which have the same REF and ALT bases. So out of the twelve normalised counts, bias if any can be identified when a combination's percentage is compared to the Truth Data.

5.5 Benchmarking

Benchmarking is defined as a standard or point of reference against which things may be compared. In this experiment, ALT variants between the different Somatic variant callers are compared with the Truth Data using the code in Appendix C. Based on the comparisons, the number of True Positives, True Negatives, False Positives and False Negatives are obtained.

In True Positives, the variant which appears in True Data also appears in variant caller data. In True Negative, the variant is missing in True Data and is also missing in the variant caller data. In False Positive, the variant is missing from the True Data but appears in the variant caller data. In False Negative, the variant appears in True Data but does not appear in variant caller data.

For True Positives, based on the ALTs, there is either True-True Positive or True-False Positive. In True-True Positive, the ALTs match and in True-False Positive, the ALTs don't match but broadly using True Positives, True Negatives, False Positives and False Negatives, Sensitivity and Specificity could be calculated.

Sensitivity is the proportion of patients with the disease who test positive and it can be calculated using the formula given below where TP is considered to be the number of True Positive ALTs and FN is considered to be the number of False Negative ALTs.

$$P\left[\frac{T^{+}}{D^{+}}\right] = \frac{TP}{TP+FN} \tag{5.1}$$

Specificity is the proportion of patients without disease who test negative and it can be calculated using the formula given below where TN is considered to be the number of True Negative ALTs and FP is considered to be the number of False Positive ALTs.

$$P\left[\frac{T^-}{D^-}\right] = \frac{TN}{TN + FP} \tag{5.2}$$

Based on the Sensitivity and Specificity values of the variant callers, they can be chosen for an application. For tests with high sensitivity, there are a fewer false negative results and for tests with high specificity, there are almost no individuals who aren't affected not ruled out.

Chapter 6

Results

Before analysing the results, it must be noted that the VCF files from the Strelka and VarScan variant callers are subsampled based on their mapping and capture regions through the Map and Filter workflows while the Truth Data isn't. Therefore the size of the Strelka and VarScan VCF files are significantly less in comparison to the Truth Data. Given below are the number of positions, SNPs, INDELs in the Truth Data

Positions	**SNPs**	**INDELs**
1,104,786	1,007,793	96,993

Table 6.1: Positions, SNPs and INDELs in Truth Data VCF file

As an experiment for Benchmarking, even the Truth Data was subsampled with respect to the capture regions downloaded from MIRACUM Annotation Data[1] using the Galaxy tool 'VCF-BEDintersect 1.0.0_rc3+galaxy0'. The outcome is a subsampled Truth VCF file with reads only from the selected capture regions. Given below are the number of positions, SNPs, INDELs in the Subsampled Truth Data

[1] https://usegalaxy.eu/u/wolfgang-maier/h/miracum-annotation-data

Positions	**SNPs**	**Indels**
891	817	74

Table 6.2: Positions, SNPs and INDELs in Subsampled Truth Data VCF file

6.1 Positions Comparison

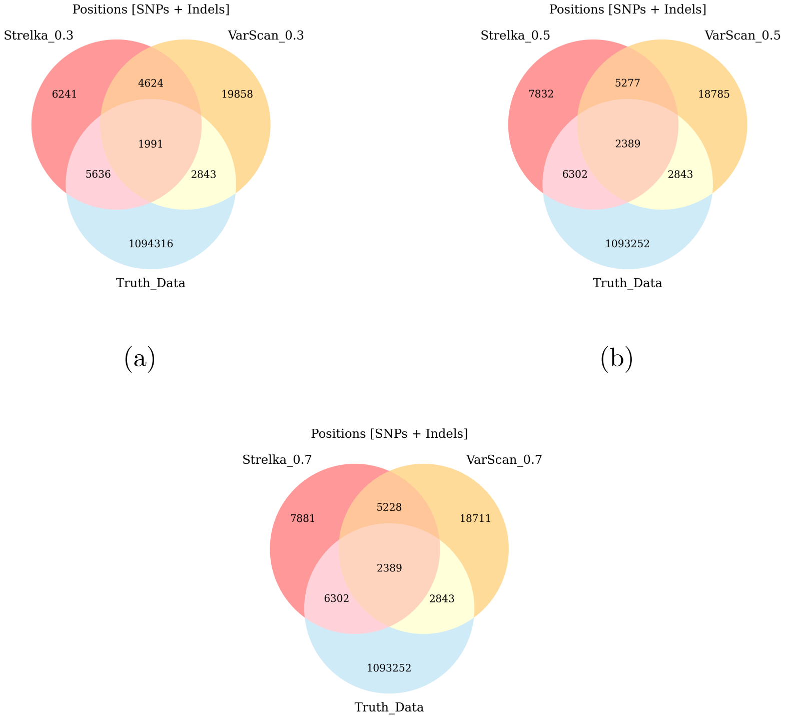
When comparing positions between the VCF files, the Strelka Somatic VCF files are categorized based on their Contamination tolerance and VarScan Somatic VCF files are categorized based on their Tumor purity.

For Strelka Somatic, with the increase in the Contamination tolerance the number of positions increase while for VarScan Somatic with the increase in the Tumor purity, the number of positions decreases slightly.

Type	**0.3**	**0.5**	**0.7**
Strelka	18,492	21,800	21,800
VarScan	29,316	29,294	29,171
Truth Data	1,104,786	1,104,786	1,104,786
Strelka & VarScan	4,624	5,277	5,228
VarScan & Truth Data	2,843	2,843	2,843
Strelka & Truth Data	5,636	6,302	6,302
Strelka, VarScan & Truth Data	1,991	2,389	2,389

Table 6.3: Positions count in different Strelka Contamination tolerance values, VarScan Tumor purity values & Truth Data VCF files

In comparison to the Truth Data, the number of common positions between Strelka and Truth Data VCF files are more than the number of common positions between VarScan and Truth Data VCF files.



(a)

(b)

(c)

Figure 6.1: (a) Positions with Contamination tolerance and Tumor purity of 0.3 (b) Positions with Contamination tolerance and Tumor purity of 0.5 (c) Positions with Contamination tolerance and Tumor purity of 0.7

Another observation is that with the increase in the Contamination tolerance and Tumor purity, the number of common positions between the three VCF files increases until a value i.e. 0.5.

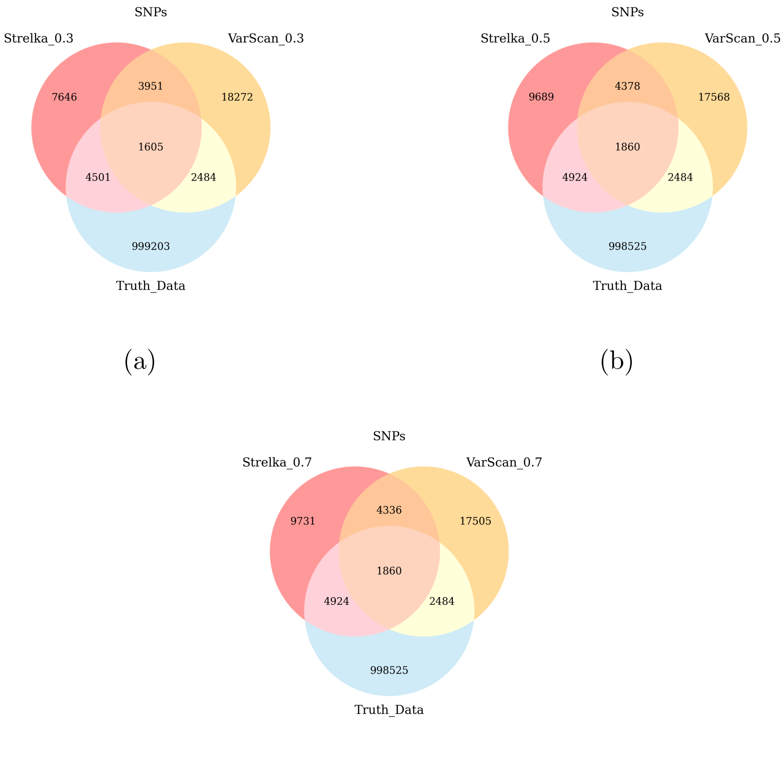
6.2 SNPs Comparison

When comparing the number of SNPs between the VCF files, there is a decrease in the count of the SNPs in the VarScan VCF files when the Tumor purity is increased and there is an increase in the number of SNPs in Strelka VCF files with the increase of the Contamination tolerance value. Given below are the comparison results between the SNPs in VCF files.

Type	**0.3**	**0.5**	**0.7**
Strelka	17,703	20,851	20,851
VarScan	26,312	26,290	26,185
Truth Data	1,007,793	1,007,793	1,007,793
Strelka & VarScan	3,951	4,378	4,336
VarScan & Truth Data	2,484	2,484	2,484
Strelka & Truth Data	4,501	4,924	4,924
Strelka, VarScan & Truth Data	1,605	1,860	1,860

Table 6.4: SNPs count in different Strelka Tumor contaminations, VarScan Tumor purities & Truth Data VCF files

In comparison to the Truth Data, the number of common SNPs between Strelka and Truth Data VCF files is higher than the number of common SNPs between the VarScan and Truth Data VCF files.



(a)

(b)

(c)

Figure 6.2: (a) SNPs with Contamination tolerance and Tumor purity of 0.3 (b) SNPs with Contamination tolerance and Tumor purity of 0.5 (c) SNPs with Contamination tolerance and Tumor purity of 0.7

Another observation is that with the increase in the Contamination tolerance and Tumor purity value, the number of common SNPs between the three VCF files increases until a value i.e. 0.5.

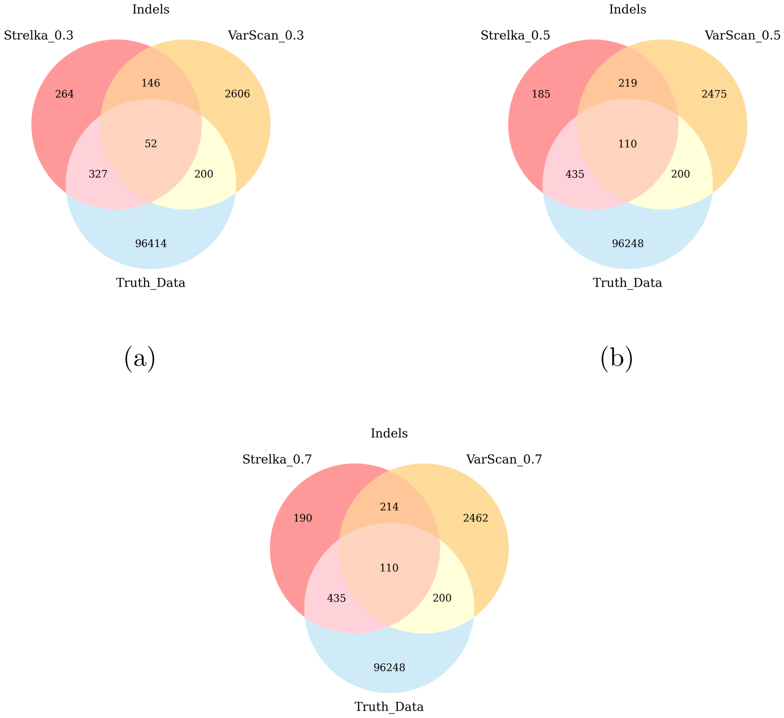
6.3 INDELs Comparison

When comparing the number of INDELs between the VCF files, there is a slight decrease in the count of the INDELs in the VarScan VCF files when the Tumor purity is increased and there is an increase in the number of INDELs in Strelka VCF files with the increase of the Contamination tolerance value. Given below are the comparison results between the INDELs in VCF files.

Type	**0.3**	**0.5**	**0.7**
Strelka	789	949	949
VarScan	3,004	3,004	2,986
Truth Data	96,993	96,993	96,993
Strelka & VarScan	146	219	214
VarScan & Truth Data	200	200	200
Strelka & Truth Data	327	435	435
Strelka, VarScan & Truth Data	52	110	110

Table 6.5: INDELs count in different Strelka Tumor contaminations, VarScan Tumor purities & Truth Data VCF files

In comparison to the Truth Data, the number of common INDELs between Strelka and Truth Data VCF files is higher than the number of common INDELs between the VarScan and Truth Data VCF files.



(a)

(b)

(c)

Figure 6.3: (a) INDELs with Contamination tolerance and Tumor purity of 0.3 (b) INDELs with Contamination tolerance and Tumor purity of 0.5 (c) INDELs with Contamination tolerance and Tumor purity of 0.7

Another observation is that with the increase in the Contamination tolerance and Tumor purity value, the number of common INDELs between the three VCF files increases until a value i.e. 0.5.

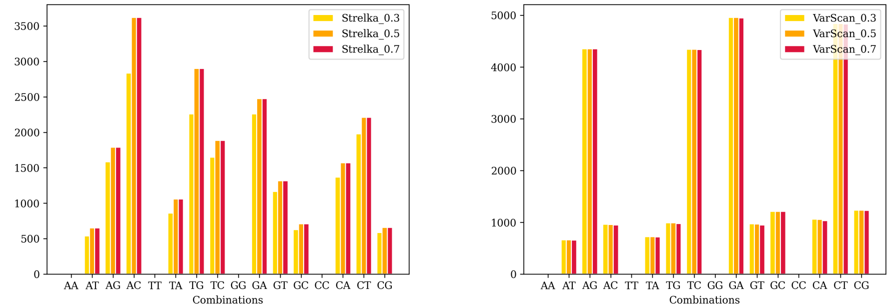
6.4 Variants Comparison

Given below are the counts for the SNP combinations in the three VCF files.

Comb	S3	S5	S7	V3	V5	V7	TD
AA	0	0	0	0	0	0	0
AT	537	652	652	662	662	658	32,639
AG	1,583	1,790	1,790	4,354	4,354	4,350	152,935
AC	2,836	3,623	3,623	962	958	947	38,384
TT	0	0	0	0	0	0	0
TA	859	1,058	1,058	722	720	715	32,817
TG	2,260	2,902	2,902	990	989	976	37,707
TC	1,648	1,885	1,885	4,343	4,342	4,336	153,474
GG	0	0	0	0	0	0	0
GA	2,258	2,477	2,477	4,960	4,959	4,950	191,603
GT	1,165	1,317	1,317	969	966	949	44,538
GC	625	708	708	1,212	1,212	1,210	43,851
CC	0	0	0	0	0	0	0
CA	1,366	1,570	1,570	1,059	1,054	1,033	44,253
CT	1,978	2,211	2,211	4,843	4,840	4,829	191,442
CG	587	657	657	1,235	1,233	1,231	44,049

Table 6.6: SNPs counts in Strelka Tumor contaminations, VarScan Tumor purities, & Truth Data SNP files where Comb means Combinations, S3 means Strelka SNP file with Tumor contamination of 0.3, S5 means Strelka SNP file with Tumor contamination of 0.5, S7 means Strelka SNP file with Tumor contamination of 0.7, V3 means VarScan SNP file with Tumor Purity of 0.3, V5 means VarScan SNP file with Tumor Purity of 0.5, V7 means VarScan SNP file with Tumor Purity of 0.7, & TD means Truth Data SNP file.

In terms of their count, for the combinations of AT, AG, TC, GA, GC, CT, & CG the count in the VarScan SNP combinations is higher compared to the Strelka SNP combinations in all Contamination tolerances and Tumor purities. For the combinations of AC, TA, TG, GT, & CA, the count in the Strelka SNP combinations is higher compared to the VarScan SNP combinations in all Contamination tolerances and Tumor purities.



(a) (b)

Figure 6.4: (a) Strelka SNPs Counts with Tumor contamination of 0.3, 0.5 and 0.7 (b) VarScan SNPs Counts with Tumor Purity of 0.3, 0.5 and 0.7

Instead of absolute value comparisons, if the normalised values are compared i.e. dividing individual counts by the total numbers of counts, it can be noted that the combinations of AT, AC, TA, TG, GT, & CA are higher in the count for the Strelka SNP combinations compared to the VarScan SNP combinations.

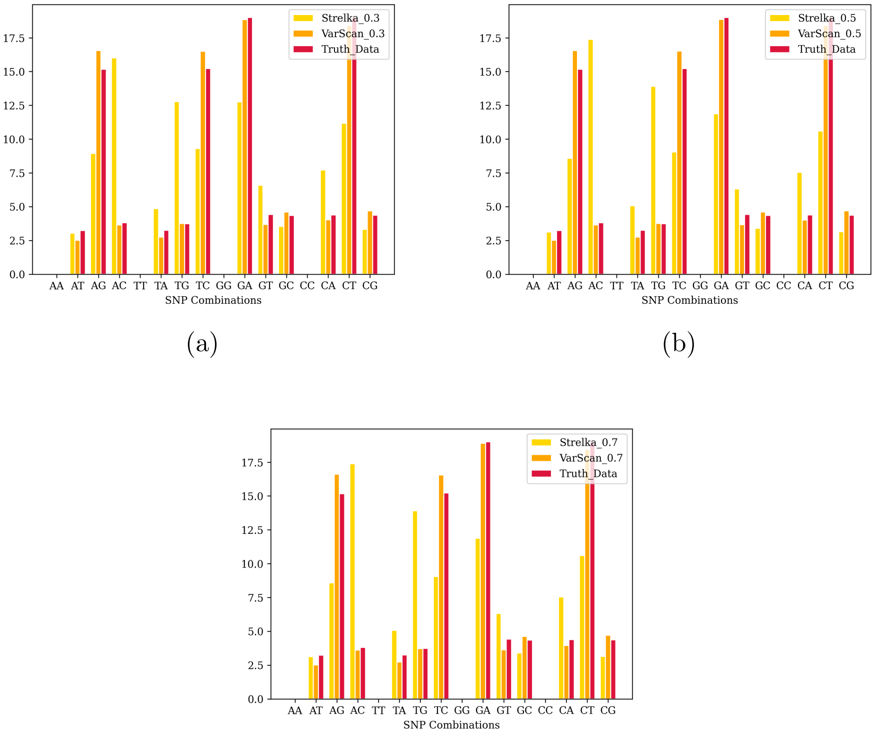
For the combinations of AG, TC, GA, GC, CT, & CG the normalised SNP counts are higher in VarScan SNP combinations compared to the Strelka SNP combinations in all Contamination tolerances and Tumor purities.

Comb	**S3**	**V3**	**S5**	**V5**	**S7**	**V7**	**TD**
AA	0	0	0	0	0	0	0
AT	3.04	2.51	3.12	2.51	3.12	2.51	3.23
AG	8.94	16.54	8.58	16.56	8.58	16.61	15.17
AC	16.02	3.65	17.37	3.64	17.37	3.61	3.80
TT	0	0	0	0	0	0	0
TA	4.85	2.74	5.07	2.73	5.07	2.73	3.25
TG	12.76	3.76	13.91	3.76	13.91	3.72	3.74
TC	9.30	16.50	9.04	16.51	9.04	16.55	15.22
GG	0	0	0	0	0	0	0
GA	12.75	18.85	11.88	18.86	11.88	18.90	19.01
GT	6.58	3.68	6.31	3.67	6.31	3.62	4.41
GC	3.53	4.60	3.39	4.61	3.39	4.62	4.35
CC	0	0	0	0	0	0	0
CA	7.71	4.02	7.52	4.00	7.52	3.94	4.39
CT	11.17	18.40	10.60	18.41	10.60	18.44	18.99
CG	3.31	4.69	3.15	4.69	3.15	4.70	4.37

Table 6.7: Normalised SNPs counts in Strelka Tumor contaminations, VarScan Tumor purities, & Truth Data SNP files where Comb means Combinations, S3 means Strelka SNP file with Tumor contamination of 0.3, S5 means Strelka SNP file with Tumor contamination of 0.5, S7 means Strelka SNP file with Tumor contamination of 0.7, V3 means VarScan SNP file with Tumor Purity of 0.3, V5 means VarScan SNP file with Tumor Purity of 0.5, V7 means VarScan SNP file with Tumor Purity of 0.7, & TD means Truth Data SNP file.

In comparison to the Truth Data, the Strelka SNP file has AT combination while the VarScan SNP file has AG, AC, TG, TC, GA, GC, CA, CT, and CG near to the normalised Truth Data SNP combinations.

In terms of the bias, Strelka variant caller calls AC and TG combinations far more times and TC combination far fewer times in comparison to both Truth Data and VarScan Somatic variant caller.



(a)

(b)

(c)

Figure 6.5: (a) Normalised SNPs Counts with Strelka Tumor Contamination and VarScan Tumor Purity of 0.3 (b) Normalised SNPs Counts with Strelka Tumor Contamination and VarScan Tumor Purity of 0.5 (c) Normalised SNPs Counts with Strelka Tumor Contamination and VarScan Tumor Purity of 0.7

6.5 Allele Frequencies

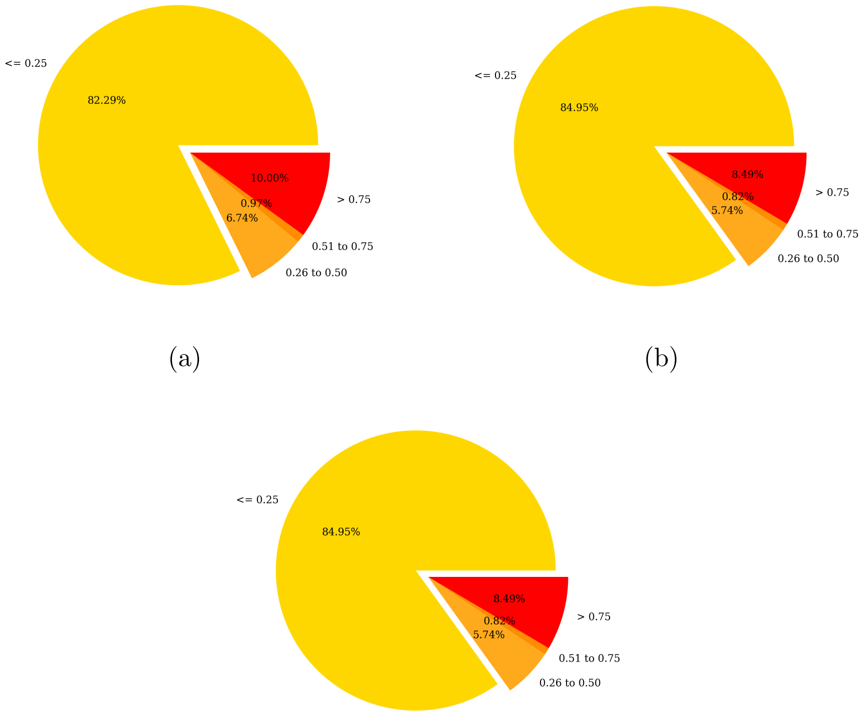
To compare the allele frequencies in the VCF files, the values are divided into four categories based on the value of allele frequency for a position. The first category is the value being ≤ 0.25, the second category is the value being > 0.25 and less than ≤ 0.50, the third category is the value being > 0.50 and less than ≤ 0.75 and the fourth category is the value being > 0.75.

Given below are the Strelka Somatic allele frequency counts with different Contamination tolerance values divided into the four categories.

Con	**≤ 0.25**	**0.25 > & ≤ 0.50**	**0.50 > & ≤ 0.75**	**> 0.75**
0.3	15,211	1,245	179	1,849
0.5	18,513	1,250	179	1,850
0.7	18,513	1,250	179	1,850

Table 6.8: Allele Frequencies count from Strelka Somatic variant caller VCF files with different Contamination tolerance values

In Strelka Somatic allele frequency distribution, the third category i.e. the values being > 0.50 and less than ≤ 0.75 is the least in count and the first category i.e. the values being ≤ 0.25 is the highest category in the count.



(a)

(b)

(c)

Figure 6.6: (a) Percentage of Strelka allele frequency counts with Contamination tolerance 0.3 (b) Percentage of Strelka allele frequency counts with Contamination tolerance of 0.5 (c) Percentage of Strelka allele frequency counts with Contamination tolerance of 0.7

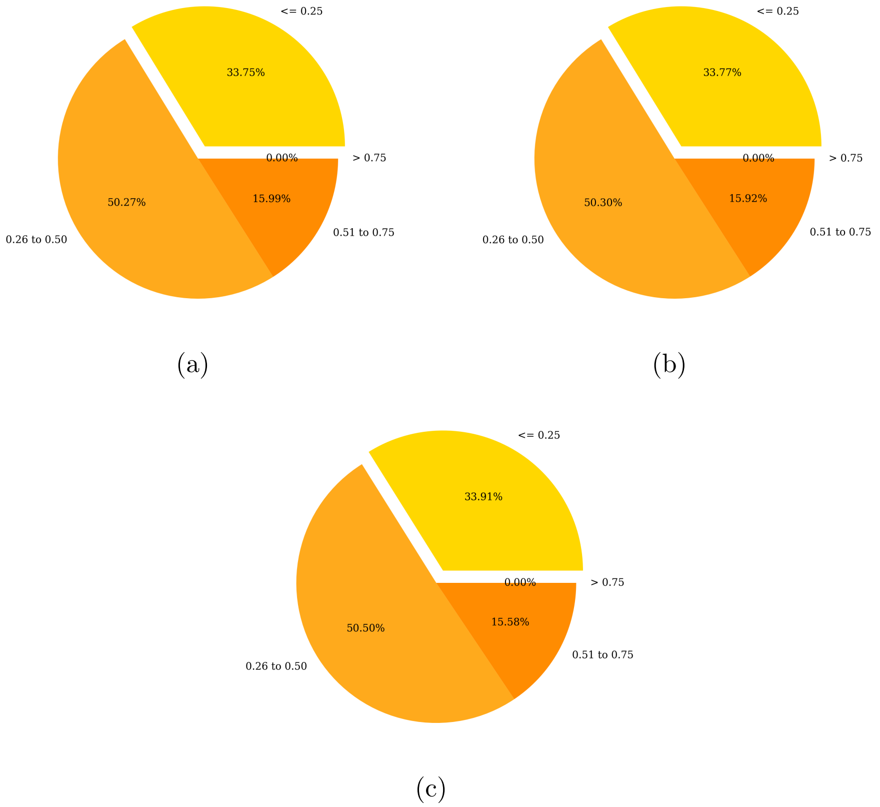
On the otherhand, for the VarScan Somatic allele frequencies, the least count is the fourth category i.e. the values being > 0.75 and the highest count is in the second category i.e. the values being > 0.25 and less than ≤ 0.50.

Given below are the VarScan Somatic allele frequency counts divided into the four categories.

Purity	**$\leq$ 0.25**	**0.25 $>$ & $\leq$ 0.50**	**0.50 $>$ & $\leq$ 0.75**	**$>$ 0.75**
0.3	9,893	14,736	4,687	0
0.5	9,893	14,736	4,665	0
0.7	9,893	14,732	4,546	0

Table 6.9: Allele Frequencies count from VarScan Somatic variant caller VCF files with different Tumor purities

There are two differences between the distributions in the allele frequencies between the variant callers. The first one is that unlike Strelka Somatic allele frequencies, there are zero counts in the fourth category i.e. $>$ 0.75 for VarScan Somatic allele frequencies. The second one is that for Strelka Somatic, the maximum number of counts are in the first category and for VarScan Somatic, the maximum number of counts are in the second category.



(a)

(b)

(c)

Figure 6.7: (a) Percentage of VarScan allele frequency with Tumor Purity of 0.3 (b) Percentage of VarScan allele frequency with Tumor Purity of 0.5 (c) Percentage of VarScan allele frequency with Tumor Purity of 0.7

Given below are the Truth Data allele frequency counts divided into the four categories

≤ 0.25	**0.25 > & ≤ 0.50**	**0.50 > & ≤ 0.75**	**> 0.75**
11,266	809	1,020	212

Table 6.10: Allele Frequencies count from Truth Data VCF file in the four categories

With all of the allele frequencies counts in the VCF files known, comparing the allele frequencies in the common positions could provide an understanding about the variant callers with respect to the Truth Data.

Given below are the allele frequency counts of the three VCF files divided into the four categories.

Type	**≤ 0.25**	**Btw 0.25 & 0.50**	**Btw 0.50 & 0.75**	**> 0.75**
Strelka	1,963	0	0	0
VarScan	0	0	1,963	0
Truth Data	0	1,963	0	0

Table 6.11: Allele Frequencies count in common positions with Tumor Purity and Contamination tolerance of 0.3

Type	**≤ 0.25**	**Btw 0.25 & 0.50**	**Btw 0.50 & 0.75**	**> 0.75**
Strelka	2,356	0	0	0
VarScan	0	1	2,355	0
Truth Data	0	2,356	0	0

Table 6.12: Allele Frequencies count in common positions with Tumor Purity and Contamination tolerance of 0.5 & 0.7

Based on the outcome, the conclusion is

6.6 Read Depth

To choose a variant caller based on Read Depth, a test would be to compare the number positions with maximum coverage. To find out these positions, filter the read depth values from the VCF files and calculate minimum, maximum, mean, median, mode and standard deviation for each of the VCF file using the Pandas library in Python.

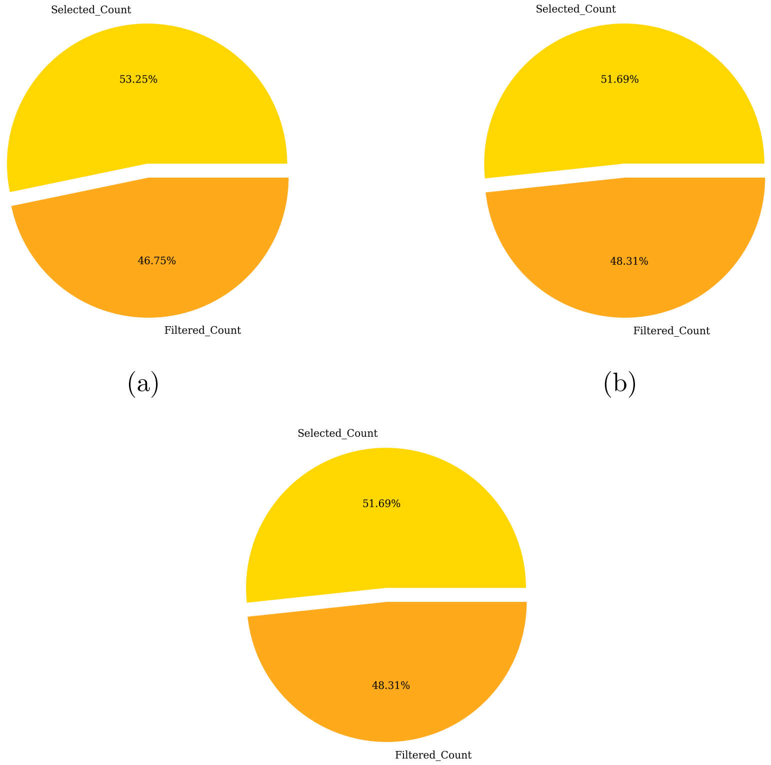
Given below are the outcome for Normal and Tumor samples for Strelka variant caller with Contamination tolerance of 0.3, 0.5 and 0.7.

Format	**Con**	**Min**	**Max**	**Mean**	**Median**	**Mode**	**SD**
Normal	0.3	1	299	64.20	61	0 43	33.91
Tumor	0.3	0	114	22.86	22	0 17	12.51
Normal	0.5	1	299	66.87	65	0 49	34.29
Tumor	0.5	0	114	23.83	23	0 17	12.70
Normal	0.7	1	299	66.87	65	0 49	32.29
Tumor	0.7	0	114	23.83	23	0 17	12.70

Table 6.13: Read Depth statistics from Strelka variant caller VCF files with different Contamination tolerance represented by Con column.

Based on the above statistics, calculate the range to select the positions with maximum coverage. To calculate the lower limit of this range, subtract the Standard Deviation (SD) from Mean and to calculate the upper limit of this range, add the Standard Deviation (SD) to the Mean. With this range, filter the reads that do not have the read depth values within this range.

Given below is a pie chart showing the filtered vs the selected reads in Strelka Somatic with different Contamination tolerances.



(a) (b)

(c)

Figure 6.8: (a) Percentage of filtered reads in Strelka with Contamination tolerance of 0.3 (b) Percentage of filtered reads in Strelka with Contamination tolerance of 0.5 (c) Percentage of filtered reads in Strelka with Contamination tolerance of 0.7

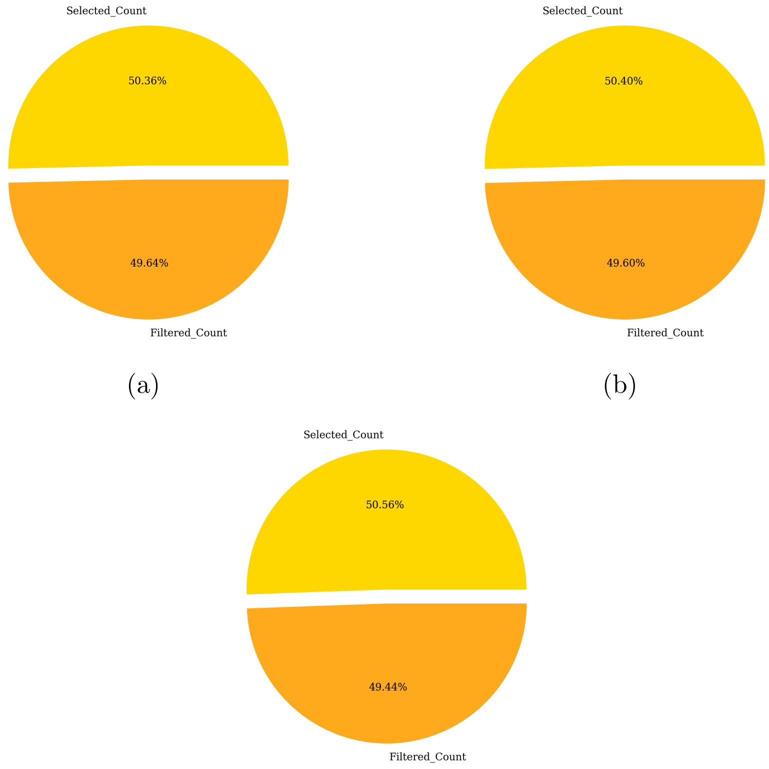
Similarly, given below are the outcome for Normal and Tumor samples for VarScan variant caller with Tumor purity of 0.3, 0.5 and 0.7.

Format	**Purity**	**Min**	**Max**	**Mean**	**Median**	**Mode**	**SD**
Normal	0.3	8	250	57.63	55	0 38	25.85
Tumor	0.3	8	98	20.82	20	0 11	9.32
Normal	0.5	8	250	57.60	55	0 38	25.83
Tumor	0.5	8	98	20.80	20	0 11	9.30
Normal	0.7	8	250	57.48	55	0 38	25.78
Tumor	0.7	8	98	20.74	19	0 11	9.27

Table 6.14: Read Depth statistics from VarScan variant caller VCF files with different Tumor purities

After calculating the range based on the Standard Deviation (SD) and Mean of the VarScan Somatic statistics, filter the reads that do not have the read depth values within this range.

Given below is a pie chart showing the filtered vs the selected reads in VarScan Somatic with different Tumor purity of 0.3, 0.5 & 0.7.



(a)

(b)

(c)

Figure 6.9: (a) Percentage of filtered reads in VarScan with Tumor purity of 0.3 (b) Percentage of filtered reads in VarScan with Tumor purity of 0.5 (c) Percentage of filtered reads in VarScan with Tumor purity of 0.7

With the percentage of selected reads in both Strelka and VarScan Somatic, it can be observed that more reads have been selected in Strelka Somatic. However, if the minimum and maximum values for both the variant callers are noted, there is a distinction.

This is due to the fact that Strelka Somatic unlike the VarScan Somatic has not been paramterised in-terms of the Read Depth filters. This difference ensures that more reads are selected for Strelka Somatic in comparison to VarScan Somatic which has the minimum Read Depth value of 8.

Considering this factor and slight difference in the percentage of selected reads, based on the Read Depth experiment, the choice of the variant caller between VarScan Somatic and Strelke Somatic cannot be determined.

6.7 Benchmarking

Before observing the benchmarking results, it must noted that there is a significant difference between the number of positions between the Strelka somatic, VarScan somatic VCF files and the Truth Data VCF file. This is because both the variant callers input data has been subsampled using the Map and Filter workflows. Due to this, there is a disproportionate difference between the number of True Positive ALTs, True Negative ALTs, False Positive ALTs and False Negative ALTs.

Given below is the benchmarking outcome for Strelka Somatic variant caller with respect to the Truth Data.

Type	Con	Total	TP	TTP	TFP	TN	FP	FN
Strelka	0.3	1,117,682	5,636	4,845	791	0	12,895	1,099,151
Strelka	0.5	1,120,330	6,302	5,379	923	0	15,541	1,098,487
Strelka	0.7	1,120,330	6,302	5,379	923	0	15,541	1,098,487

Table 6.15: Benchmarking Strelka ALTs with respect to the Truth Data ALTs where Con represents Contamination Tolerance, TP represents True Positive ALTs, TTP represents True True Positive ALTs, TFP represents True False Positive ALTs, TNA represents True Negative ALTs, FP represents False Positive ALTs, and FN represents False Negative ALTs.

From the outcome, two observations can be made. The first one being the True Negative ALTs are zero and the second one being the number of False Negative ALTs are huge irrespective of Contamination tolerance values.

With respect to the number of True Negative ALTs being zero for both Strelka Somatic and VarScan Somatic benchmarking because the VCF files only call variants that have different REF and ALT bases. Therefore when benchmarking the variant callers with respect to the Truth Data, the number of True Negatives will be zero.

The number of False Positives being high can be regulated if the Truth Data is subsampled. One way to subsample the Truth Data is by selecting only those reads that belong to the capture regions. This way, the Truth Data can be subsampled and the outcome of the benchmarking of Strelka Somatic with respect to the subsampled Truth Data is as follows

Type	**Con**	**Total**	**TP**	**TTP**	**TFP**	**TN**	**FP**	**FN**
Strelka	0.3	18,771	612	480	132	0	17,880	279
Strelka	0.5	21,891	802	631	171	0	20,998	91
Strelka	0.7	21,891	802	631	171	0	20,998	91

Table 6.16: Benchmarking Strelka ALTs with respect to the Subsampled Truth Data ALTs where Con represents Contamination Tolerance, TP represents True Positive ALTs, TTP represents True True Positive ALTs, TFP represents True False Positive ALTs, TNA represents True Negative ALTs, FP represents False Positive ALTs, and FN represents False Negative ALTs.

Similarly the outcome of benchmarking VarScan Somatic VCF file with respect to the Truth Data VCF file is as follows

Type	**Pur**	**Total**	**TP**	**TTP**	**TFP**	**TN**	**FP**	**FN**
VarScan	0.3	1,131,279	2,843	2,699	144	0	26,493	1,101,943
VarScan	0.5	1,131,257	2,843	2,699	144	0	26,471	1,101,943
VarScan	0.7	1,131,134	2,843	2,699	144	0	26,348	1,101,943

Table 6.17: Benchmarking VarScan ALTs with respect to the Truth Data ALTs where Pur represents Tumor Purity, TP represents True Positive ALTs, TTP represents True True Positive ALTs, TFP represents True False Positive ALTs, TNA represents True Negative ALTs, FP represents False Positive ALTs, and FN represents False Negative ALTs.

The outcome of benchmarking VarScan Somatic VCF file with respect to subsampled Truth Data VCF file is as follows

Type	**Pur**	**Total**	**TP**	**TTP**	**TFP**	**TN**	**FP**	**FN**
VarScan	0.3	29,499	708	708	0	0	28,608	183
VarScan	0.5	29,477	708	708	0	0	28,586	183
VarScan	0.7	29,354	708	708	0	0	28,463	183

Table 6.18: Benchmarking VarScan ALTs with respect to the Subsampled Truth Data ALTs where Pur represents Tumor Purity, TP represents True Positive ALTs, TTP represents True True Positive ALTs, TFP represents True False Positive ALTs, TNA represents True Negative ALTs, FP represents False Positive ALTs, and FN represents False Negative ALTs.

Based on the benchmarking results, Sensitivity and Specificity for the variant callers can be calculated. However since the True Negatives are zero in the benchmarking results, Specificity for both the variant callers is zero. However the Sensitivity values are as follows

Type	**Pur/Con**	**Truth**	**Sensitivity**
Strelka	0.3	Original	0.0051
Strelka	0.3	Subsampled	0.68
Strelka	0.5	Original	0.0057
Strelka	0.5	Subsampled	0.89
Strelka	0.7	Original	0.0057
Strelka	0.7	Subsampled	0.89
VarScan	0.3	Original	0.0025
VarScan	0.3	Subsampled	0.79
VarScan	0.5	Original	0.0025
VarScan	0.5	Subsampled	0.79
VarScan	0.7	Original	0.0025
VarScan	0.7	Subsampled	0.79

Table 6.19: Sensitivity values for variant callers.

Based on the Sensitivity scores, Strelka Somatic variant caller can be chosen as the Sensitivity values for both the Truth Data and subsampled Truth Data are more compared to the VarScan Somatic sensitivity values.

Chapter 7

Conclusion

Based on the positions, SNPs, INDELs and Benchmarking experiments, the choice of the variant caller between Strelka Somatic and VarScan Somatic is clearly Strelka Somatic to find cancer-causing variants. Even the unparameterised Strelka Somatic bechmarking results mentioned in Appendix H identifies cancer-causing variants qualitatively better than VarScan Somatic.

However, Strelka Somatic variant calling is biased in calling a few SNP combinations quite often more than the Truth Data. This isn't an issue with VarScan Somatic variant caller. Second issue with Strelka Somatic variant calling is that changing parameters in the variant caller isn't easy especially dealing with Tumor purity.

Acknowledgments

First and foremost, I would like to thank Dr. Mehmet Tekman and Dr. Wolfgang Maier for their unending support and incredible patience. Without their motivation, trust and opportunity, this thesis wouldn't have happened.

Secondly, I would like to thank Prof. Dr. Rolf Backofen for supervising the thesis and giving me an opportunity to work in the Department for Bioinformatics and I would also like to thank Prof. Dr. Dr. Melanie Börries for being the external supervisor.

Finally, I would like to thank Dr. Manjusha Chintalapati and Dr. Sreeraj Kolora from University of California, Berkeley for their incredible insights in the time of need.

Bibliography

[1] Cornish, A., & Guda, C. (2015). A Comparison of Variant Calling Pipelines Using Genome in a Bottle as a Reference. BioMed Research International, 2015, 1–11. https://doi.org/10.1155/2015/456479

[2] Saraswathy, N., & Ramalingam, P. (2011) Genome sequencing method. Concepts and Techniques in Genomics and Proteomics (pp. 95–107). Elsevier.

[3] Ng, P. C., & Kirkness, E. F. (2010). Whole Genome Sequencing. In Methods in Molecular Biology (pp. 215–226). Humana Press. https://doi.org/10.1007/978-1-60327-367-1_12

[4] Bick, D., & Dimmock, D. (2011). Whole exome and whole genome sequencing. Current Opinion in Pediatrics, 23(6), 594–600. https://doi.org/10.1097/mop.0b013e32834b20ec

[5] Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. (2009). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Research, 38(6), 1767–1771. https://doi.org/10.1093/nar/gkp1137

[6] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., & Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics, 25(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

[7] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, 1000 Genomes Project Analysis Group, The variant call format and VCFtools,

Bioinformatics, Volume 27, Issue 15, 1 August 2011, Pages 2156–2158, https://doi.org/10.1093/bioinformatics/btr330

[8] Prokosch, H.-U., Acker, T., Bernarding, J., Binder, H., Boeker, M., Boerries, M., Daumke, P., Ganslandt, T., Hesser, J., Höning, G., Neumaier, M., Marquardt, K., Renz, H., Rothkötter, H.-J., Schade-Brittinger, C., Schmücker, P., Schüttler, J., Sedlmayr, M., Serve, H., … Storf, H. (2018). MIRACUM: Medical Informatics in Research and Care in University Medicine. Methods of Information in Medicine, 57(S 01), e82–e91. https://doi.org/10.3414/me17-02-0025

[9] Petty, J. K., & Vetto, J. T. (2002). Beyond doughnuts: tumor board recommendations influence patient care. Journal of cancer education : the official journal of the American Association for Cancer Education, 17(2), 97–100. https://doi.org/10.1080/08858190209528807

[10] Schwaederle, M., Parker, B. A., Schwab, R. B., Fanta, P. T., Boles, S. G., Daniels, G. A., Bazhenova, L. A., Subramanian, R., Coutinho, A. C., Ojeda-Fournier, H., Datnow, B., Webster, N. J., Lippman, S. M., & Kurzrock, R. (2014). Molecular Tumor Board: The University of California San Diego Moores Cancer Center Experience. The Oncologist, 19(6), 631–636. https://doi.org/10.1634/theoncologist.2013-0405

[11] Wolfgang M., et al. (2021, Apr 21). Institute of Medical Bioinformatics and Systems Medicine (IBSM) - MIRACUM Pipe Galaxy. https://github.com/AG-Boerries/MIRACUM-Pipe-Galaxy

[12] Afgan, E., Baker, D., Batut, B., van den Beek, M., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Grüning, B. A., Guerler, A., Hillman-Jackson, J., Hiltemann, S., Jalili, V., Rasche, H., Soranzo, N., Goecks, J., Taylor, J., Nekrutenko, A., & Blankenberg, D. (2018). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Research, 46(W1), W537–W544. https://doi.org/10.1093/nar/gky379

[13] Bérénice B., & Yvan L. B. (2021, Apr 21). Introduction to Variant Analysis. Galaxy Training Material. https://training.galaxyproject.org/training-material/topics/variant-analysis/slides/introduction.html#1

[14] Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. PLOS Computational Biology, 12(1), e1004668. https://doi.org/10.1371/journal.pcbi.1004668

[15] Xiao, C., Zook, J., Trask, S., & Sherry, S. (2014, September 30). Abstract 5328: GIAB: Genome reference material development resources for clinical sequencing. Molecular and Cellular Biology. Proceedings: AACR Annual Meeting 2014; April 5-9, 2014; San Diego, CA. https://doi.org/10.1158/1538-7445.am2014-5328

[16] Krusche, P., Trigg, L., Boutros, P. C., Mason, C. E., De La Vega, F. M., Moore, B. L., Gonzalez-Porta, M., Eberle, M. A., Tezak, Z., Lababidi, S., Truty, R., Asimenos, G., Funke, B., Fleharty, M., Salit, M., & Zook, J. M. (2018). Best Practices for Benchmarking Germline Small Variant Calls in Human Genomes. Cold Spring Harbor Laboratory. https://doi.org/10.1101/270157

[17] Clements, D., Hiltemann, S., Batut, B., Rasche, H., Heydarian, M., & Training Network, T. G. (2020). Using the Galaxy Training Network tutorial library for bioinformatics training programs. Journal of Biomolecular Techniques : JBT, 31(Suppl), S2.

[18] Wolfgang M., Bérénice B., Torsten H., Anika E., Björn (2021, Apr 10). Exome sequencing data analysis for diagnosing a genetic disease. Galaxy Training Materials.
https://training.galaxyproject.org/archive/2019-12-01/topics/variant-analysis/tutorials/exome-seq/tutorial.html

[19] Batut, B., Hiltemann, S., Bagnacani, A., Baker, D., Bhardwaj, V., Blank, C., Bretaudeau, A., Brillet-Guéguen, L., Čech, M., Chilton, J., Clements, D., Doppelt-Azeroual, O., Erxleben, A., Freeberg, M. A., Gladman, S., Hoogstrate, Y., Hotz, H.-R., Houwaart, T., Jagtap, P., … Grüning, B. (2018). Community-Driven Data Analysis Training for Biology. Cell Systems, 6(6), 752-758.e1. https://doi.org/10.1016/j.cels.2018.05.012

[20] Wolfgang M. (2021, Apr 10). Identification of somatic and germline variants from tumor and normal sample pairs. Galaxy Training Materials. https://galaxyproject.github.io/training-material/topics/variant-analysis/tutorials/somatic-variants/tutorial.html

[21] Galaxy Project and Community. (2021, Apr 21). Planemo Documentation Release 0.74.4.
https://planemo.readthedocs.io/en/latest/readme.html

[22] Ben-Kiki, O., Evans, C., & Ingerson, B., 2009. (2021, Apr 21). YAML ain't markup language (YAMLTM) version 1.2.
http://www.yaml.org/spec/1.2/spec.html

[23] Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., Durbin, R., & 1000 Genomes Project Analysis Group (2011). The variant call format and VCFtools. Bioinformatics (Oxford, England), 27(15), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

[24] Pwwang 2009. (2021, Apr 21). VCFStats version 0.0.6.
https://vcfstats.readthedocs.io/en/latest/

[25] Davis, S. (2019). vcftoolz: a Python package for comparing and evaluating Variant Call Format files. Journal of Open Source Software, 4(35), 1144. https://doi.org/10.21105/joss.01144

[26] Illumina. (2021, Apr 21). Strelka User Guide.
https://github.com/Illumina/strelka/blob/v2.9.x/docs/userGuide/README.md#somatic

[27] Python. (2021, Apr 27). Pandas User Guide.
https://pandas.pydata.org/docs/user_guide/

[28] Koboldt, D. C., Zhang, Q., Larson, D. E., Shen, D., McLellan, M. D., Lin, L., Miller, C. A., Mardis, E. R., Ding, L., & Wilson, R. K. (2012). VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome research, 22(3), 568–576. https://doi.org/10.1101/gr.129684.111

[29] Saunders, C. T., Wong, W. S. W., Swamy, S., Becq, J., Murray, L. J., & Cheetham, R. K. (2012). Strelka: accurate somatic small-variant calling from sequenced tumor–normal sample pairs. Bioinformatics, 28(14), 1811–1817. https://doi.org/10.1093/bioinformatics/bts271

[30] Narzisi, G., Corvelo, A., Arora, K., Bergmann, E. A., Shah, M., Musunuri, R., Emde, A.-K., Robine, N., Vacic, V., & Zody, M. C. (2018).

Genome-wide somatic variant calling using localized colored de Bruijn graphs. Communications Biology, 1(1). https://doi.org/10.1038/s42003-018-0023-9

[31] Taylor-Weiner, A., Stewart, C., Giordano, T., Miller, M., Rosenberg, M., Macbeth, A., Lennon, N., Rheinbay, E., Landau, D.-A., Wu, C. J., & Getz, G. (2018). DeTiN: overcoming tumor-in-normal contamination. Nature Methods, 15(7), 531–534. https://doi.org/10.1038/s41592-018-0036-9

[32] Chen, J., Li, X., Zhong, H., Meng, Y., & Du, H. (2019). Systematic comparison of germline variant calling pipelines cross multiple next-generation sequencers. Scientific Reports, 9(1). https://doi.org/10.1038/s41598-019-45835-3

[33] Kim, S. Y., & Speed, T. P. (2013). Comparing somatic mutation-callers: beyond Venn diagrams. BMC Bioinformatics, 14(1). https://doi.org/10.1186/1471-2105-14-189

[34] Harvard Chan School: Bioinformatics Core. (2021, Apr 27). Projects: GIAB Somatic.
https://github.com/hbc/projects/tree/master/giab_somatic

[35] Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/mcse.2007.55

Appendix A

Map Workflow YAML

```
NORMAL forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

NORMAL reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

TUMOR forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

TUMOR reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz
```

Filter Workflow YAML

```
NORMAL forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

NORMAL reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz
```

```
NORMAL mapped reads:
     class: File
     location: https://usegalaxy.eu/...display?to_ext=bam

Capture regions:
     class: File
     location: https://usegalaxy.eu/...display?to_ext=bed
```

WES Analysis Workflow VarScan YAML

```
Normal sample name: "Normal sample name"
Normal purity estimate: 1.0
Tumor sample name: "Tumor sample name"
Tumor purity estimate: 0.7

NORMAL forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

NORMAL reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

TUMOR forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

TUMOR reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

Capture regions:
     class: File
     location: https://usegalaxy.eu/...display?to_ext=bed
```

WES Analysis Workflow YAML

```
Normal sample name: "Normal sample name"
SNV Contamination Tolerance: 0.3
Tumor sample name: "Tumor sample name"
Indels Contamination Tolerance: 0.7

NORMAL forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

NORMAL reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

TUMOR forward reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

TUMOR reverse reads:
     class: File
     location: https://zenodo.org/record/....fastq.gz

Capture regions:
     class: File
     location: https://usegalaxy.eu/...display?to_ext=bed
```

Appendix B

Strelka Allele Frequency

```
# 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']

# For Strelka, Allele Frequency values are not given in the VCF
    file.
# Calculations of AF values is different for INDEL and SNV files.

# The first step is to selected the necessary columns in INDEL
    files from command-line.
# cut -f 1-2,4-5,9,11 Input.vcf > Output.vcf

# Removing the header information of VCF files in command-line
# Step 2 - 'sed '/^#/d' Output.vcf > Updated_Output.vcf'

# Consider the three INDEL inputs based on Contamination
    tolerances of 0.3, 0.5 & 0.7.
```

```
dff = pd.read_csv("Input_0.3_INDEL.vcf", sep = '\t', index_col=
    False)
dff1 = pd.read_csv("Input_0.5_INDEL.vcf", sep = '\t', index_col=
    False)
dff2 = pd.read_csv("Input_0.7_INDEL.vcf", sep = '\t', index_col=
    False)

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

# Splitting the "TUMOR" columns by ':' and renaming the new
    columns based on the format "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
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[[’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[[’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 reorganizing the columns.
dff = dff.drop([’FORMAT’, ’TUMOR’, ’Tumor_DP’, ’Tumor_DP2’,
    ’Tumor_DP2’, ’Tumor_TOR’, ’Tumor_DP50’, ’Tumor_FDP50’,
    ’Tumor_SUBDP50’, ’Tumor_BCN50’], axis=1)
dff1 = dff1.drop([’FORMAT’, ’TUMOR’, ’Tumor_DP’, ’Tumor_DP2’,
    ’Tumor_DP2’, ’Tumor_TOR’, ’Tumor_DP50’, ’Tumor_FDP50’,
    ’Tumor_SUBDP50’, ’Tumor_BCN50’], axis=1)
dff2 = dff2.drop([’FORMAT’, ’TUMOR’, ’Tumor_DP’, ’Tumor_DP2’,
    ’Tumor_DP2’, ’Tumor_TOR’, ’Tumor_DP50’, ’Tumor_FDP50’,
    ’Tumor_SUBDP50’, ’Tumor_BCN50’], axis=1)

# Creating new columns by splitting the "TUMOR" columns by ’,’ for
    TIR and TAR values"
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[[’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[[’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’, ’Tumor_TAR’,
    ’Tumor_TAR’, ’Tumor_TAR_First’, ’Tumor_TAR_Second’,
    ’Tumor_TIR_First’, ’Tumor_TIR_Second’]
```

```
dff1.columns = ['CHROM_POS', 'REF', 'ALT', 'Tumor_TAR',
    'Tumor_TAR', 'Tumor_TAR_First', 'Tumor_TAR_Second',
    'Tumor_TIR_First', 'Tumor_TIR_Second']
dff2.columns = ['CHROM_POS', 'REF', 'ALT', 'Tumor_TAR',
    'Tumor_TAR', 'Tumor_TAR_First', 'Tumor_TAR_Second',
    'Tumor_TIR_First', 'Tumor_TIR_Second']

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

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

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

# Converting string values columns to int for calculations.
dff['Tumor_TAR_First'] = dff['Tumor_TAR_First'].astype(int)
dff['Tumor_TIR_First'] = dff['Tumor_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['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['COMMON'] = dff["Tumor_TAR_First"] + dff["Tumor_TIR_First"]
print(dff)

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

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

# Getting the Tumor Allele Frequency
dff['Tumor_Allele_Frequency'] =
    dff['Tumor_TIR_First']/dff['COMMON']
```

```
dff1[’Tumor_Allele_Frequency’] =
    dff1[’Tumor_TIR_First’]/dff1[’COMMON’]
dff2[’Tumor_Allele_Frequency’] =
    dff2[’Tumor_TIR_First’]/dff2[’COMMON’]

# Converting string values columns to int.
dff[’Tumor_Allele_Frequency’] =
    dff[’Tumor_Allele_Frequency’].astype(float).round(2)
dff1[’Tumor_Allele_Frequency’] =
    dff1[’Tumor_Allele_Frequency’].astype(float).round(2)
dff2[’Tumor_Allele_Frequency’] =
    dff2[’Tumor_Allele_Frequency’].astype(float).round(2)

# Concatenating the "CHROM" and "POS"
dff["Tumor_Allele_Frequency"] = dff[’REF’].astype(str) + ’:’ +
    dff[’Tumor_Allele_Frequency’].astype(str)
dff1["Tumor_Allele_Frequency"] = dff1[’REF’].astype(str) + ’:’ +
    dff1[’Tumor_Allele_Frequency’].astype(str)
dff2["Tumor_Allele_Frequency"] = dff2[’REF’].astype(str) + ’:’ +
    dff2[’Tumor_Allele_Frequency’].astype(str)

# Dropping of the unnecessary columns and reorganizing them.
dff = dff.drop([’REF’, ’ALT’, ’Tumor_TAR’, ’Tumor_TAR’,
    ’Tumor_TAR_First’, ’Tumor_TIR_First’, ’COMMON’], axis=1)
print(dff)

dff1 = dff1.drop([’REF’, ’ALT’, ’Tumor_TAR’, ’Tumor_TAR’,
    ’Tumor_TAR_First’, ’Tumor_TIR_First’, ’COMMON’], axis=1)
print(dff1)

dff2 = dff2.drop([’REF’, ’ALT’, ’Tumor_TAR’, ’Tumor_TAR’,
    ’Tumor_TAR_First’, ’Tumor_TIR_First’, ’COMMON’], axis=1)
print(dff2)

# Saving the results in csv.
dff.to_csv(’Strelka3_INDEL_AF.csv’, sep=’,’, index = False)
dff1.to_csv(’Strelka5_INDEL_AF.csv’, sep=’,’, index = False)
dff2.to_csv(’Strelka7_INDEL_AF.csv’, sep=’,’, index = False)
```

```
# Now complete selecting the columns and removing header
    information for the SNV files and add them as inputs.
dfff = pd.read_csv("Input_0.3_SNV.vcf", sep = '\t', index_col=
    False)
dfff1 = pd.read_csv("Input_0.5_SNV.vcf", sep = '\t', index_col=
    False)
dfff2 = pd.read_csv("Input_0.7_SNV.vcf", sep = '\t', index_col=
    False)

# Naming the columns after importing the csv file.
dfff.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'TUMOR']
dfff1.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'TUMOR']
dfff2.columns = ['CHROM', 'POS', 'REF', 'ALT', 'FORMAT', 'TUMOR']

# Concatenating the "CHROM" and "POS"
dfff["CHROM_POS"] = dfff['CHROM'].astype(str) + '-' +
    dfff['POS'].astype(str)
dfff1["CHROM_POS"] = dfff1['CHROM'].astype(str) + '-' +
    dfff1['POS'].astype(str)
dfff2["CHROM_POS"] = dfff2['CHROM'].astype(str) + '-' +
    dfff2['POS'].astype(str)

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

dfff1 = dfff1.drop(['CHROM', 'POS'], axis=1)
cols = dfff1.columns.tolist()
cols = cols[-1:] + cols[:-1]
dfff1 = dfff1[cols]

dfff2 = dfff2.drop(['CHROM', 'POS'], axis=1)
cols = dfff2.columns.tolist()
cols = cols[-1:] + cols[:-1]
dfff2 = dfff2[cols]

# Adding columns for single read depth value.
dfff['REF_U'] = dfff["REF"] + "U"
```

```
dfff['ALT_U'] = dfff["ALT"] + "U"

dfff1['REF_U'] = dfff1["REF"] + "U"
dfff1['ALT_U'] = dfff1["ALT"] + "U"

dfff2['REF_U'] = dfff2["REF"] + "U"
dfff2['ALT_U'] = dfff2["ALT"] + "U"

# Creating new columns by splitting the "TUMOR" columns by ':' and
   renaming the new columns based on the format
   "DP:FDP:SDP:SUBDP:AU:CU:GU:TU"
dfff[['Tumor_DP', 'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP',
   'Tumor_AU', 'Tumor_CU', 'Tumor_GU', 'Tumor_TU']] =
   dfff['TUMOR'].str.split(':',expand=True)
dfff1[['Tumor_DP', 'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP',
   'Tumor_AU', 'Tumor_CU', 'Tumor_GU', 'Tumor_TU']] =
   dfff1['TUMOR'].str.split(':',expand=True)
dfff2[['Tumor_DP', 'Tumor_FDP', 'Tumor_SDP', 'Tumor_SUBDP',
   'Tumor_AU', 'Tumor_CU', 'Tumor_GU', 'Tumor_TU']] =
   dfff2['TUMOR'].str.split(':',expand=True)

# Dropping of the unnecessary columns and reorganizing the columns.
dfff = dfff.drop(['FORMAT', 'TUMOR', 'Tumor_DP', 'Tumor_FDP',
   'Tumor_SDP', 'Tumor_SUBDP'], axis=1)
dfff1 = dfff1.drop(['FORMAT', 'TUMOR', 'Tumor_DP', 'Tumor_FDP',
   'Tumor_SDP', 'Tumor_SUBDP'], axis=1)
dfff2 = dfff2.drop(['FORMAT', 'TUMOR', 'Tumor_DP', 'Tumor_FDP',
   'Tumor_SDP', 'Tumor_SUBDP'], axis=1)

for i in dfff['CHROM_POS']:
   dfff.loc[dfff['REF_U'] == 'AU', 'REF_Tumor'] = dfff.Tumor_AU
   dfff.loc[dfff['REF_U'] == 'CU', 'REF_Tumor'] = dfff.Tumor_CU
   dfff.loc[dfff['REF_U'] == 'GU', 'REF_Tumor'] = dfff.Tumor_GU
   dfff.loc[dfff['REF_U'] == 'TU', 'REF_Tumor'] = dfff.Tumor_TU
   dfff.loc[dfff['ALT_U'] == 'AU', 'ALT_Tumor'] = dfff.Tumor_AU
   dfff.loc[dfff['ALT_U'] == 'CU', 'ALT_Tumor'] = dfff.Tumor_CU
   dfff.loc[dfff['ALT_U'] == 'GU', 'ALT_Tumor'] = dfff.Tumor_GU
   dfff.loc[dfff['ALT_U'] == 'TU', 'ALT_Tumor'] = dfff.Tumor_TU
print(dfff)
```

```
for i in dff1[’CHROM_POS’]:
    dfff1.loc[dfff1[’REF_U’] == ’AU’, ’REF_Tumor’] = dfff1.Tumor_AU
    dfff1.loc[dfff1[’REF_U’] == ’CU’, ’REF_Tumor’] = dfff1.Tumor_CU
    dfff1.loc[dfff1[’REF_U’] == ’GU’, ’REF_Tumor’] = dfff1.Tumor_GU
    dfff1.loc[dfff1[’REF_U’] == ’TU’, ’REF_Tumor’] = dfff1.Tumor_TU
    dfff1.loc[dfff1[’ALT_U’] == ’AU’, ’ALT_Tumor’] = dfff1.Tumor_AU
    dfff1.loc[dfff1[’ALT_U’] == ’CU’, ’ALT_Tumor’] = dfff1.Tumor_CU
    dfff1.loc[dfff1[’ALT_U’] == ’GU’, ’ALT_Tumor’] = dfff1.Tumor_GU
    dfff1.loc[dfff1[’ALT_U’] == ’TU’, ’ALT_Tumor’] = dfff1.Tumor_TU
print(dfff1)

for i in dff2[’CHROM_POS’]:
    dfff2.loc[dfff2[’REF_U’] == ’AU’, ’REF_Tumor’] = dfff2.Tumor_AU
    dfff2.loc[dfff2[’REF_U’] == ’CU’, ’REF_Tumor’] = dfff2.Tumor_CU
    dfff2.loc[dfff2[’REF_U’] == ’GU’, ’REF_Tumor’] = dfff2.Tumor_GU
    dfff2.loc[dfff2[’REF_U’] == ’TU’, ’REF_Tumor’] = dfff2.Tumor_TU
    dfff2.loc[dfff2[’ALT_U’] == ’AU’, ’ALT_Tumor’] = dfff2.Tumor_AU
    dfff2.loc[dfff2[’ALT_U’] == ’CU’, ’ALT_Tumor’] = dfff2.Tumor_CU
    dfff2.loc[dfff2[’ALT_U’] == ’GU’, ’ALT_Tumor’] = dfff2.Tumor_GU
    dfff2.loc[dfff2[’ALT_U’] == ’TU’, ’ALT_Tumor’] = dfff2.Tumor_TU
print(dfff2)

# Creating new columns by splitting the "TUMOR" columns by ’,’
dfff[[’REF_Tumor_First’, ’REF_Tumor_Second’]] =
    dfff[’REF_Tumor’].str.split(’,’,expand=True)
dfff[[’ALT_Tumor_First’, ’ALT_Tumor_Second’]] =
    dfff[’ALT_Tumor’].str.split(’,’,expand=True)
print(dfff)

dfff1[[’REF_Tumor_First’, ’REF_Tumor_Second’]] =
    dfff1[’REF_Tumor’].str.split(’,’,expand=True)
dfff1[[’ALT_Tumor_First’, ’ALT_Tumor_Second’]] =
    dfff1[’ALT_Tumor’].str.split(’,’,expand=True)
print(dfff1)

dfff2[[’REF_Tumor_First’, ’REF_Tumor_Second’]] =
    dfff2[’REF_Tumor’].str.split(’,’,expand=True)
dfff2[[’ALT_Tumor_First’, ’ALT_Tumor_Second’]] =
    dfff2[’ALT_Tumor’].str.split(’,’,expand=True)
print(dfff2)
```

```
# Dropping of the unnecessary columns and reorganizing the columns.
dfff = dfff.drop([’REF_U’, ’ALT_U’, ’Tumor_AU’, ’Tumor_CU’,
    ’Tumor_GU’, ’Tumor_TU’, ’Tumor_AU’, ’Tumor_CU’, ’Tumor_GU’,
    ’Tumor_TU’, ’REF_Tumor_Second’, ’ALT_Tumor_Second’], axis=1)
print(dfff)

dfff1 = dfff1.drop([’REF_U’, ’ALT_U’, ’Tumor_AU’, ’Tumor_CU’,
    ’Tumor_GU’, ’Tumor_TU’, ’Tumor_AU’, ’Tumor_CU’, ’Tumor_GU’,
    ’Tumor_TU’, ’REF_Tumor_Second’, ’ALT_Tumor_Second’], axis=1)
print(dfff1)

dfff2 = dfff2.drop([’REF_U’, ’ALT_U’, ’Tumor_AU’, ’Tumor_CU’,
    ’Tumor_GU’, ’Tumor_TU’, ’Tumor_AU’, ’Tumor_CU’, ’Tumor_GU’,
    ’Tumor_TU’, ’REF_Tumor_Second’, ’ALT_Tumor_Second’], axis=1)
print(dfff2)

# Naming the columns after importing the csv file.
dfff.columns = [’CHROM_POS’, ’REF’, ’ALT’, ’REF_Tumor’,
    ’ALT_Tumor’, ’REF_Tumor_First’, ’ALT_Tumor_First’]
print(dfff)

dfff1.columns = [’CHROM_POS’, ’REF’, ’ALT’, ’REF_Tumor’,
    ’ALT_Tumor’, ’REF_Tumor_First’, ’ALT_Tumor_First’]
print(dfff1)

dfff2.columns = [’CHROM_POS’, ’REF’, ’ALT’, ’REF_Tumor’,
    ’ALT_Tumor’, ’REF_Tumor_First’, ’ALT_Tumor_First’]
print(dfff2)

# Converting string values columns to int.
dfff[’REF_Tumor_First’] = dfff[’REF_Tumor_First’].astype(int)
dfff[’ALT_Tumor_First’] = dfff[’ALT_Tumor_First’].astype(int)
print(dfff)

dfff1[’REF_Tumor_First’] = dfff1[’REF_Tumor_First’].astype(int)
dfff1[’ALT_Tumor_First’] = dfff1[’ALT_Tumor_First’].astype(int)
print(dfff1)

dfff2[’REF_Tumor_First’] = dfff2[’REF_Tumor_First’].astype(int)
```

```
dfff2['ALT_Tumor_First'] = dfff2['ALT_Tumor_First'].astype(int)
print(dfff2)

# Adding the values for formula.
dfff['COMMON'] = dfff["REF_Tumor_First"] + dfff["ALT_Tumor_First"]
dfff1['COMMON'] = dfff1["REF_Tumor_First"] +
    dfff1["ALT_Tumor_First"]
dfff2['COMMON'] = dfff2["REF_Tumor_First"] +
    dfff2["ALT_Tumor_First"]

# Getting Allele Frequency
dfff['Tumor'] = dfff['ALT_Tumor_First']/dfff['COMMON']
dfff1['Tumor'] = dfff1['ALT_Tumor_First']/dfff1['COMMON']
dfff2['Tumor'] = dfff2['ALT_Tumor_First']/dfff2['COMMON']

# Converting string values columns to int.
dfff['Tumor'] = dfff['Tumor'].astype(float).round(2)
dfff1['Tumor'] = dfff1['Tumor'].astype(float).round(2)
dfff2['Tumor'] = dfff2['Tumor'].astype(float).round(2)

# Concatenating the "CHROM" and "POS"
dfff["Tumor"] = dfff['REF'].astype(str) + ':' +
    dfff['Tumor'].astype(str)
dfff1["Tumor"] = dfff1['REF'].astype(str) + ':' +
    dfff1['Tumor'].astype(str)
dfff2["Tumor"] = dfff2['REF'].astype(str) + ':' +
    dfff2['Tumor'].astype(str)

# Dropping of the unnecessary columns and reorganizing the columns.
dfff = dfff.drop(['REF', 'ALT', 'REF_Tumor', 'ALT_Tumor',
    'REF_Tumor_First', 'ALT_Tumor_First', 'COMMON'], axis=1)
dfff1 = dfff1.drop(['REF', 'ALT', 'REF_Tumor', 'ALT_Tumor',
    'REF_Tumor_First', 'ALT_Tumor_First', 'COMMON'], axis=1)
dfff2 = dfff2.drop(['REF', 'ALT', 'REF_Tumor', 'ALT_Tumor',
    'REF_Tumor_First', 'ALT_Tumor_First', 'COMMON'], axis=1)

# Saving the results in csv.
dfff.to_csv('Strelka3_SNV_AF.csv', sep=',', index = False)
dfff1.to_csv('Strelka5_SNV_AF.csv', sep=',', index = False)
dfff2.to_csv('Strelka7_SNV_AF.csv', sep=',', index = False)
```

Strelka Allele Frequency Plot

```
# 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’]

# Consider the Allele Frequency of SNV and INDEL with a specific
    Contamination tolerance value as input.
df1 = pd.read_csv("Input_INDEL_AF.csv", sep = ’,’, index_col=
    False)
df2 = pd.read_csv("Input_SNV_AF.csv", sep = ’,’, index_col= False)

# Renaming the columns.
df1.columns = [’CHROM_POS’, ’Tumor’]
df2.columns = [’CHROM_POS’, ’Tumor’]

# Creating new columns by splitting the "TUMOR" columns by ’:’ and
    renaming the new columns based on allele and value.
df1[[’INDEL_Allele’, ’INDEL_Value’]] =
    df1[’Tumor’].str.split(’:’,expand=True)
df2[[’SNV_Allele’, ’SNV_Value’]] =
    df2[’Tumor’].str.split(’:’,expand=True)
print(df1)
print(df2)

# Renaming Columns
# dff.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’]
```

```
# Dropping of the unnecessary columns and only choosing the "TUMOR
   Depth" i.e. "TUMOR-DP"
df1 = df1.drop(['CHROM_POS', 'Tumor', 'INDEL_Allele'], axis=1)
df2 = df2.drop(['CHROM_POS', 'Tumor', 'SNV_Allele'], axis=1)
print(df1)
print(df2)

# Converting string values columns to float.
df1['INDEL_Value'] = df1['INDEL_Value'].astype(float)
df2['SNV_Value'] = df2['SNV_Value'].astype(float)
print(df1)
print(df2)

# Getting a count based on allele frequency values.
IN1 = df1[df1 < 0.26].count()
IN2 = df1[df1 < 0.51].count()
IN3 = df1[df1 < 0.76].count()
IN4 = df1[df1 < 1.01].count()
print('Indel count')
print(IN1)

SN1 = df2[df2 < 0.26].count()
SN2 = df2[df2 < 0.51].count()
SN3 = df2[df2 < 0.76].count()
SN4 = df2[df2 < 1.01].count()

# Getting the final values
IN5 = (IN1 - IN2).abs()
IN6 = (IN2 - IN3).abs()
IN7 = (IN3 - IN4).abs()
print('Second round Indels')
print(IN5)

SN5 = (SN1 - SN2).abs()
SN6 = (SN2 - SN3).abs()
SN7 = (SN3 - SN4).abs()
print('Second round')
print(SN5)

# Converting into list.
```

```
First_Indels = IN1.tolist()
Second_Indels = IN5.tolist()
Third_Indels = IN6.tolist()
Fourth_Indels = IN7.tolist()
First_SNV = SN1.tolist()
Second_SNV = SN5.tolist()
Third_SNV = SN6.tolist()
Fourth_SNV = SN7.tolist()

# Final count.
First = np.add(First_Indels, First_SNV)
Second = np.add(Second_Indels, Second_SNV)
Third = np.add(Third_Indels, Third_SNV)
Fourth = np.add(Fourth_Indels, Fourth_SNV)
print('Testing the count')
print(First)

# Declaring new columns.
a1 = np.append(First, Second)
a1 = np.append(a1, Third)
a1 = np.append(a1, Fourth)
print(a1)

# set width of bar
width = 0.10

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['<= 0.25', '0.26 to 0.50', '0.51 to 0.75', '> 0.75']
explode = (0.1, 0, 0, 0)
colors = ['#FFD700','#FFAA1C','#FF8C01','#FF0000']
ax.pie(a1, explode=explode, labels = langs, colors=colors,
   autopct='%1.2f%%')
ax.set_title('Allele Frequency Counts Percentage')
plt.savefig('Output_Plot.png', dpi = 300)
```

VarScan Allele Frequency

```
# 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 concatenating "CHROM" and "POS" for all
    Tumor purity values.
dff = pd.read_csv("Input_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("Input_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("Input_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)

# Reorganizing 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)

# Saving the results in csv.
dff.to_csv(’Output_0.3_AF.csv’, sep=’,’, index = False)
dff2.to_csv(’Output_0.5_AF.csv’, sep=’,’, index = False)
dff3.to_csv(’Output_0.7_AF.csv’, sep=’,’, index = False)
```

VarScan Allele Frequency Plot

```
# 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’]

# Consider the allele frequency values of a specific Tumor purity
   as input.
df = pd.read_csv("Input_AF.csv", sep = ’,’, index_col= False)

# Creating new columns by splitting the "Allele" and "Value" by
    ’:’.
df[[’VarScan_Allele’, ’VarScan_Value’]] =
   df[’ALLELE:FREQ’].str.split(’:’,expand=True)
```

```
# Dropping of the unnecessary columns and only choosing the "TUMOR
    Depth" i.e. "TUMOR-DP"
df = df.drop([’CHROM_POS’, ’ALLELE:FREQ’, ’VarScan_Allele’],
    axis=1)

# Renaming the columns.
df.columns = [’AF’]
print(df)

# Converting string values columns to float.
df[’AF’] = df[’AF’].astype(float)

# Getting a count based on allele frequency values.
df1 = df[df < 0.26].count()
df2 = df[df < 0.51].count()
df3 = df[df < 0.76].count()
df4 = df[df < 1.01].count()
print(’First quater’)
print(df1)

# Getting the final values
df5 = (df1 - df2).abs()
df6 = (df2 - df3).abs()
df7 = (df3 - df4).abs()
print(’Second quater’)
print(df5)
print(df6)
print(df7)

# Converting into list.
First = df1.tolist()
Second = df5.tolist()
Third = df6.tolist()
Fourth = df7.tolist()

# Declaring new columns.
a1 = np.append(First, Second)
a1 = np.append(a1, Third)
a1 = np.append(a1, Fourth)
```

```
print(a1)

# set width of bar
width = 0.10

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis(’equal’)
langs = [’<= 0.25’, ’0.26 to 0.50’, ’0.51 to 0.75’, ’> 0.75’]
explode = (0.1, 0, 0, 0)
colors = [’#FFD700’,’#FFAA1C’,’#FF8C01’,’#FF0000’]
ax.pie(a1, explode=explode, labels = langs, colors=colors,
   autopct=’%1.2f%%’)
ax.set_title(’Allele Frequency Counts Percentage’)
plt.savefig(’Output_Plot.png’, dpi = 300)
```

Truth Data Allele Frequency

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

# Reading csv files and concatenating "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)

# Reorganizing 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 "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 "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 = dff1.tolist()
Second = dff5.tolist()
Third = dff6.tolist()
Fourth = dff7.tolist()

# Declaring new columns.
a1 = np.append(First, Second)
a1 = np.append(a1, Third)
a1 = np.append(a1, Fourth)
print(a1)

# set width of bar
width = 0.10

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis(’equal’)
langs = [’<= 0.25’, ’0.26 to 0.50’, ’0.51 to 0.75’, ’> 0.75’]
explode = (0.1, 0, 0, 0)
colors = [’#FFD700’,’#FFAA1C’,’#FF8C01’,’#FF0000’]
ax.pie(a1, explode=explode, labels = langs, colors=colors,
    autopct=’%1.2f%%’)
ax.set_title(’Allele Frequency Counts Percentage’)
plt.savefig(’Truth_Data_Allele_Frequency.png’, dpi = 300)
```

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 concatenating "CHROM" and "POS" based on
    common Contamination tolerance and Tumor purity value.
df = pd.read_csv("Strelka_0.3.csv", sep = ’,’, 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)
print(df)
print(df1)
print(df2)

# Renaming Columns
df.columns = [’CHROM_POS’, ’AF’]
df1.columns = [’CHROM_POS’, ’AF’]
df2.columns = [’CHROM_POS’, ’AF’]
print(df)
print(df1)
print(df2)

# 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’, ’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 "TUMOR" columns by allele
    and value.
Second[[’Strelka_Allele’, ’Strelka_Value’]] =
    Second[’Strelka’].str.split(’:’,expand=True)
Second[[’VarScan_Allele’, ’VarScan_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 "TUMOR
   Depth" i.e. "TUMOR-DP"
dff = Second.drop([’CHROM_POS’, ’Strelka’, ’VarScan’,
   ’Truth_Data’, ’Strelka_Allele’, ’VarScan_Allele’,
   ’Truth_Data_Allele’], axis=1)
print(dff)

# Renaming the columns.
dff.columns = [’Strelka’, ’VarScan’, ’Truth_Data’]
print(dff)

# Converting string values columns to float.
dff[’Strelka’] = dff[’Strelka’].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’, ’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)

# This code must be executed for 0.5 and 0.7 values for Strelka
    Somatic and VarScan Somatic VCF files.
```

Appendix C

Strelka Bechmarking

```
# 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

# For benchmarking only the ALTs in the two VCF files are compared.
# The first step is to selected the necessary columns in the
   command-line.
# cut -f 2,5 Input.vcf > Output.vcf

# Removing the header information of VCF files in command-line
# sed ’/ˆ#/d’ Input.vcf > Output.vcf

# Reading csv files and concatenating "CHROM" and "POS" for Truth
   Data and Strelka VCF files with different Contamination
   tolerance values.
df1 = pd.read_csv("Truth.vcf", sep = ’\t’, index_col= False)
df2 = pd.read_csv("Input_0.3.vcf", sep = ’\t’, index_col= False)
df3 = pd.read_csv("Input_0.5.vcf", sep = ’\t’, index_col= False)
df4 = pd.read_csv("Input_0.7.vcf", sep = ’\t’, index_col= False)
```

```
# Merging columns based on "Positions" and to get all positions,
   use the "Outer" merge.
dff1 = pd.merge(df1, df2, how="outer", on=['POS'])
dff2 = pd.merge(df1, df3, how="outer", on=['POS'])
dff3 = pd.merge(df1, df4, how="outer", on=['POS'])

# Renaming the columns after importing the input.
dff1.columns = ['POS', 'Truth_Data', 'Strelka_Three']
dff2.columns = ['POS', 'Truth_Data', 'Strelka_Five']
dff3.columns = ['POS', 'Truth_Data', 'Strelka_Seven']
print(dff1)
print(dff2)
print(dff3)

# Total number of reads
dff4 = len(dff1)
print('Total lenght is')
print(dff4)

dff47 = len(dff2)
print('Total lenght is')
print(dff47)

dff48 = len(dff3)
print('Total lenght is')
print(dff48)

# Comparison column
dff1["TP"] = np.where(dff1["Truth_Data"].notnull() &
   dff1["Strelka_Three"].notnull(), 0, 1)
dff1["TN"] = np.where(dff1["Truth_Data"].isnull() &
   dff1["Strelka_Three"].isnull(), 0, 1)
dff1["FP"] = np.where(dff1["Truth_Data"].isnull() &
   dff1["Strelka_Three"].notnull(), 0, 1)
dff1["FN"] = np.where(dff1["Truth_Data"].notnull() &
   dff1["Strelka_Three"].isnull(), 0, 1)

# Selecting the columns
dff5 = dff1.loc[dff1['TP'] == 0]
dff6 = dff1.loc[dff1['TN'] == 0]
```

```
dff7 = dff1.loc[dff1['FP'] == 0]
dff8 = dff1.loc[dff1['FN'] == 0]

# Total numbers of Comparisons
dff9 = len(dff5)
dff10 = len(dff6)
dff11 = len(dff7)
dff12 = len(dff8)

print("Total number of True Positives:")
print(dff9)
print("Total number of True Negatives:")
print(dff10)
print("Total number of False Postivies:")
print(dff11)
print("Total number of False Negatives:")
print(dff12)

# Comparison column
dff2["TP"] = np.where(dff2["Truth_Data"].notnull() &
   dff2["Strelka_Five"].notnull(), 0, 1)
dff2["TN"] = np.where(dff2["Truth_Data"].isnull() &
   dff2["Strelka_Five"].isnull(), 0, 1)
dff2["FP"] = np.where(dff2["Truth_Data"].isnull() &
   dff2["Strelka_Five"].notnull(), 0, 1)
dff2["FN"] = np.where(dff2["Truth_Data"].notnull() &
   dff2["Strelka_Five"].isnull(), 0, 1)

# Selecting the columns
dff13 = dff2.loc[dff2['TP'] == 0]
dff14 = dff2.loc[dff2['TN'] == 0]
dff15 = dff2.loc[dff2['FP'] == 0]
dff16 = dff2.loc[dff2['FN'] == 0]

# Total numbers of Comparisons
dff17 = len(dff13)
dff18 = len(dff14)
dff19 = len(dff15)
dff20 = len(dff16)
```

```
print("Total number of True Positives:")
print(dff17)
print("Total number of True Negatives:")
print(dff18)
print("Total number of False Postivies:")
print(dff19)
print("Total number of False Negatives:")
print(dff20)

# Comparison column
dff3["TP"] = np.where(dff3["Truth_Data"].notnull() &
   dff3["Strelka_Seven"].notnull(), 0, 1)
dff3["TN"] = np.where(dff3["Truth_Data"].isnull() &
   dff3["Strelka_Seven"].isnull(), 0, 1)
dff3["FP"] = np.where(dff3["Truth_Data"].isnull() &
   dff3["Strelka_Seven"].notnull(), 0, 1)
dff3["FN"] = np.where(dff3["Truth_Data"].notnull() &
   dff3["Strelka_Seven"].isnull(), 0, 1)

# Selecting the columns
dff21 = dff3.loc[dff3['TP'] == 0]
dff22 = dff3.loc[dff3['TN'] == 0]
dff23 = dff3.loc[dff3['FP'] == 0]
dff24 = dff3.loc[dff3['FN'] == 0]

# Total numbers of Comparisons
dff25 = len(dff21)
dff26 = len(dff22)
dff27 = len(dff23)
dff28 = len(dff24)

print("Total number of True Positives:")
print(dff25)
print("Total number of True Negatives:")
print(dff26)
print("Total number of False Postivies:")
print(dff27)
print("Total number of False Negatives:")
print(dff28)
```

```
# Merging columns based on "Positions"
dff29 = pd.merge(df1, df2, on=['POS'])
dff30 = pd.merge(df1, df3, on=['POS'])
dff31 = pd.merge(df1, df4, on=['POS'])
dff29.columns = ['POS', 'Truth_Data', 'Strelka_Three']
dff30.columns = ['POS', 'Truth_Data', 'Strelka_Five']
dff31.columns = ['POS', 'Truth_Data', 'Strelka_Seven']
print(dff29)
print(dff30)
print(dff31)

# Comparison column
dff29["Comparison"] = np.where((dff29["Truth_Data"] ==
    dff29["Strelka_Three"]), 0, 1)
dff30["Comparison"] = np.where((dff30["Truth_Data"] ==
    dff30["Strelka_Five"]), 0, 1)
dff31["Comparison"] = np.where((dff31["Truth_Data"] ==
    dff31["Strelka_Seven"]), 0, 1)

# Selecting Positive Comparison
dff35 = dff29.loc[dff29['Comparison'] == 0]
dff36 = dff30.loc[dff30['Comparison'] == 0]
dff37 = dff31.loc[dff31['Comparison'] == 0]

# Total number of Positive Comparison
dff38 = len(dff35)
dff39 = len(dff36)
dff40 = len(dff37)

# Selecting Negative Comparison
dff41 = dff29.loc[dff29['Comparison'] == 1]
dff42 = dff30.loc[dff30['Comparison'] == 1]
dff43 = dff31.loc[dff31['Comparison'] == 1]

# Total number of Negative Comparison
dff44 = len(dff41)
dff45 = len(dff42)
dff46 = len(dff43)

# Delcaring a new dataframe.
```

```
df = pd.DataFrame()

# Taking all combinations as a list.
Type = ['Strelka_0.3', 'Strelka_0.5', 'Strelka_0.7']
Total = [dff4, dff47, dff48]
True_Positive = [dff9, dff17, dff25]
True_Negative = [dff10, dff18, dff26]
False_Positives = [dff11, dff19, dff27]
False_Negatives = [dff12, dff20, dff28]
True_True_Postive = [dff38, dff39, dff40]
True_False_Postive = [dff44, dff45, dff46]

# Adding columns
df['Type'] = Type
df['Total'] = Total
df['True_Positives_ALTs'] = True_Positive
df['True_True_Positives_ALTs'] = True_True_Postive
df['True_False_Positives_ALTs'] = True_False_Postive
df['True_Negatives_ALTs'] = True_Negative
df['False_Positives_ALTs'] = False_Positives
df['False_Negatives_ALTs'] = False_Negatives

# Collecting it into a dataframe.
print(df)

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

VarScan Benchmarking

```
# 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

# Completing the command-line operations on the Truth Data and the
   three VarScan VCF files with different Tumor purity values.
# Reading the csv files and concatinating "CHROM" and "POS"
df1 = pd.read_csv("Truth.vcf", sep = '\t', index_col= False)
df2 = pd.read_csv("Input_0.3.vcf", sep = '\t', index_col= False)
df3 = pd.read_csv("Input_0.5.vcf", sep = '\t', index_col= False)
df4 = pd.read_csv("Input_0.7.vcf", sep = '\t', index_col= False)

# Merging columns based on "Positions". To get all positions, use
   the "Outer" merge.
dff1 = pd.merge(df1, df2, how="outer", on=['POS'])
dff2 = pd.merge(df1, df3, how="outer", on=['POS'])
dff3 = pd.merge(df1, df4, how="outer", on=['POS'])

# Renaming the columns after importing the input.
dff1.columns = ['POS', 'Truth_Data', 'VarScan_Three']
dff2.columns = ['POS', 'Truth_Data', 'VarScan_Five']
dff3.columns = ['POS', 'Truth_Data', 'VarScan_Seven']
print(dff1)
print(dff2)
print(dff3)

# Total number of reads
dff4 = len(dff1)
print('Total lenght is')
print(dff4)

dff47 = len(dff2)
print('Total lenght is')
print(dff47)

dff48 = len(dff3)
print('Total lenght is')
print(dff48)

# Comparison column
dff1["TP"] = np.where(dff1["Truth_Data"].notnull() &
   dff1["VarScan_Three"].notnull(), 0, 1)
```

```
dff1["TN"] = np.where(dff1["Truth_Data"].isnull() &
    dff1["VarScan_Three"].isnull(), 0, 1)
dff1["FP"] = np.where(dff1["Truth_Data"].isnull() &
    dff1["VarScan_Three"].notnull(), 0, 1)
dff1["FN"] = np.where(dff1["Truth_Data"].notnull() &
    dff1["VarScan_Three"].isnull(), 0, 1)

# Selecting the columns
dff5 = dff1.loc[dff1['TP'] == 0]
dff6 = dff1.loc[dff1['TN'] == 0]
dff7 = dff1.loc[dff1['FP'] == 0]
dff8 = dff1.loc[dff1['FN'] == 0]

# Total numbers of Comparisons
dff9 = len(dff5)
dff10 = len(dff6)
dff11 = len(dff7)
dff12 = len(dff8)

print("Total number of True Positives:")
print(dff9)
print("Total number of True Negatives:")
print(dff10)
print("Total number of False Postivies:")
print(dff11)
print("Total number of False Negatives:")
print(dff12)

# Comparison column
dff2["TP"] = np.where(dff2["Truth_Data"].notnull() &
    dff2["VarScan_Five"].notnull(), 0, 1)
dff2["TN"] = np.where(dff2["Truth_Data"].isnull() &
    dff2["VarScan_Five"].isnull(), 0, 1)
dff2["FP"] = np.where(dff2["Truth_Data"].isnull() &
    dff2["VarScan_Five"].notnull(), 0, 1)
dff2["FN"] = np.where(dff2["Truth_Data"].notnull() &
    dff2["VarScan_Five"].isnull(), 0, 1)

# Selecting the columns
dff13 = dff2.loc[dff2['TP'] == 0]
```

```
dff14 = dff2.loc[dff2[’TN’] == 0]
dff15 = dff2.loc[dff2[’FP’] == 0]
dff16 = dff2.loc[dff2[’FN’] == 0]

# Total numbers of Comparisons
dff17 = len(dff13)
dff18 = len(dff14)
dff19 = len(dff15)
dff20 = len(dff16)

print("Total number of True Positives:")
print(dff17)
print("Total number of True Negatives:")
print(dff18)
print("Total number of False Postivies:")
print(dff19)
print("Total number of False Negatives:")
print(dff20)

# Comparison column
dff3["TP"] = np.where(dff3["Truth_Data"].notnull() &
    dff3["VarScan_Seven"].notnull(), 0, 1)
dff3["TN"] = np.where(dff3["Truth_Data"].isnull() &
    dff3["VarScan_Seven"].isnull(), 0, 1)
dff3["FP"] = np.where(dff3["Truth_Data"].isnull() &
    dff3["VarScan_Seven"].notnull(), 0, 1)
dff3["FN"] = np.where(dff3["Truth_Data"].notnull() &
    dff3["VarScan_Seven"].isnull(), 0, 1)

# Selecting the columns
dff21 = dff3.loc[dff3[’TP’] == 0]
dff22 = dff3.loc[dff3[’TN’] == 0]
dff23 = dff3.loc[dff3[’FP’] == 0]
dff24 = dff3.loc[dff3[’FN’] == 0]

# Total numbers of Comparisons
dff25 = len(dff21)
dff26 = len(dff22)
dff27 = len(dff23)
dff28 = len(dff24)
```

```
print("Total number of True Positives:")
print(dff25)
print("Total number of True Negatives:")
print(dff26)
print("Total number of False Postivies:")
print(dff27)
print("Total number of False Negatives:")
print(dff28)

# Merging columns based on "Positions"
dff29 = pd.merge(df1, df2, on=['POS'])
dff30 = pd.merge(df1, df3, on=['POS'])
dff31 = pd.merge(df1, df4, on=['POS'])
dff29.columns = ['POS', 'Truth_Data', 'VarScan_Three']
dff30.columns = ['POS', 'Truth_Data', 'VarScan_Five']
dff31.columns = ['POS', 'Truth_Data', 'VarScan_Seven']
print(dff29)
print(dff30)
print(dff31)

# Comparison column
dff29["Comparison"] = np.where((dff29["Truth_Data"] ==
    dff29["VarScan_Three"]), 0, 1)
dff30["Comparison"] = np.where((dff30["Truth_Data"] ==
    dff30["VarScan_Five"]), 0, 1)
dff31["Comparison"] = np.where((dff31["Truth_Data"] ==
    dff31["VarScan_Seven"]), 0, 1)

# Selecting Positive Comparison
dff35 = dff29.loc[dff29['Comparison'] == 0]
dff36 = dff30.loc[dff30['Comparison'] == 0]
dff37 = dff31.loc[dff31['Comparison'] == 0]

# Total number of Positive Comparison
dff38 = len(dff35)
dff39 = len(dff36)
dff40 = len(dff37)

# Selecting Negative Comparison
```

```
dff41 = dff29.loc[dff29['Comparison'] == 1]
dff42 = dff30.loc[dff30['Comparison'] == 1]
dff43 = dff31.loc[dff31['Comparison'] == 1]

# Total number of Negative Comparison
dff44 = len(dff41)
dff45 = len(dff42)
dff46 = len(dff43)

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

# Taking all combinations as a list.
Type = ['VarScan_0.3', 'VarScan_0.5', 'VarScan_0.7']
Total = [dff4, dff47, dff48]
True_Positive = [dff9, dff17, dff25]
True_Negative = [dff10, dff18, dff26]
False_Positives = [dff11, dff19, dff27]
False_Negatives = [dff12, dff20, dff28]
True_True_Postive = [dff38, dff39, dff40]
True_False_Postive = [dff44, dff45, dff46]

# Adding columns
df['Type'] = Type
df['Total'] = Total
df['True_Positives_ALTs'] = True_Positive
df['True_True_Positives_ALTs'] = True_True_Postive
df['True_False_Positives_ALTs'] = True_False_Postive
df['True_Negatives_ALTs'] = True_Negative
df['False_Positives_ALTs'] = False_Positives
df['False_Negatives_ALTs'] = False_Negatives

# Collecting it into a dataframe.
print(df)

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

Appendix D

Strelka Positions, SNPs, & INDELs

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

# The first step is to selected the necessary columns in
    command-line. In this case, there is no need for this step.
# Removing the header information of VCF files in command-line
# sed ’/^#/d’ Input.vcf > Output.vcf

# Input the vcf files for positions with different Contamination
    tolerance values.
df = pd.read_csv("Input_0.3.vcf", sep = ’\t’, index_col= False)
df1 = pd.read_csv("Input_0.5.vcf", sep = ’\t’, index_col= False)
df2 = pd.read_csv("Input_0.7.vcf", sep = ’\t’, index_col= False)

# Obtain the SNPs from a VCF file using the command: vcftools
    --vcf input_file.vcf --remove-indels --recode --recode-INFO-all
    --out SNPs_only and then update the inputs for the SNP count.
df3 = pd.read_csv("Input_0.3_SNV.vcf", sep = ’\t’, index_col=
    False)
df4 = pd.read_csv("Input_0.5_SNV.vcf", sep = ’\t’, index_col=
    False)
df5 = pd.read_csv("Input_0.7_SNV.vcf", sep = ’\t’, index_col=
    False)

# Obtain the SNPs from a VCF file using the command: vcftools
    --vcf input_file.vcf --keep-only-indels --recode
    --recode-INFO-all --out INDELs_only and then update the inputs
    for the INDEL count.
df6 = pd.read_csv("Input_0.3_INDEL.vcf", sep = '\t', index_col=
    False)
df7 = pd.read_csv("Input_0.5_INDEL.vcf", sep = '\t', index_col=
    False)
df8 = pd.read_csv("Input_0.7_INDEL.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('Positions_SNPs_Indels_Strelka_Counts.csv', sep=',',
    index = None)
```

VarScan Positions, SNPs, & INDELs

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

# Inputing vcf files for positions.
df = pd.read_csv("Input_0.3.vcf", sep = '\t', index_col= False)
df1 = pd.read_csv("Input_0.5.vcf", sep = '\t', index_col= False)
df2 = pd.read_csv("Input_0.7.vcf", sep = '\t', index_col= False)

# Inputing vcf files for SNPs after the command-line operation.
df3 = pd.read_csv("Input_0.3_SNP.vcf", sep = '\t', index_col=
    False)
```

```
df4 = pd.read_csv("Input_0.5_SNP.vcf", sep = '\t', index_col=
    False)
df5 = pd.read_csv("Input_0.7_SNP.vcf", sep = '\t', index_col=
    False)

# Inputing vcf files for INDELs after the command-line operation.
df6 = pd.read_csv("Updated_VarScan_0.3_INDEL.vcf", sep = '\t',
    index_col= False)
df7 = pd.read_csv("Updated_VarScan_0.5_INDEL.vcf", sep = '\t',
    index_col= False)
df8 = pd.read_csv("Updated_VarScan_0.7_INDEL.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('Positions_SNPs_Indels_VarScan_Counts.csv', sep=',',
    index = None)
```

Truth Data Positions, SNPs, & INDELs

```
# 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 after the command-line operation.
df2 = pd.read_csv("Updated_Somatic_Truth_SNPs.vcf", sep = '\t',
    index_col= False)

# Inputing vcf files for INDELs after the command-line operation.
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)
```

Count Positions and Plots

```
# 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 concatenating "CHROM" and "POS" based on
    Contamination tolerance and Tumor purity values.
df = pd.read_csv("Strelka_0.3.vcf", sep = ’\t’, index_col= False)
df1 = pd.read_csv("VarScan_0.3.vcf", sep = ’\t’, index_col= False)
df2 = pd.read_csv("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('Positions_Plot.pdf')
plt.savefig('Positions_Plot.png', dpi = 300)

# This code must be executed for 0.5 and 0.7 values for Strelka
   Somatic and VarScan Somatic VCF files.
```

Count INDELs & Plots

```
# 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 concatenating "CHROM" and "POS" based on
    Contamination tolerance and Tumor purity values.
df = pd.read_csv("Strelka_0.3_INDEL.vcf", sep = '\t', index_col=
    False)
df1 = pd.read_csv("VarScan_0.3_INDEL.vcf", sep = '\t', index_col=
    False)
df2 = pd.read_csv("ruth_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('INDELs_Plot.pdf')
plt.savefig('INDELs_Plot.png', dpi = 300)

# This code must be executed for 0.5 and 0.7 values for Strelka
    Somatic and VarScan Somatic VCF files.
```

Count SNPs & Plots

```
# 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" based on
    Contamination tolerance and Tumor purity values.
df = pd.read_csv("Updated_Strelka_0.3_SNV.vcf", sep = '\t',
    index_col= False)
df1 = pd.read_csv("Updated_VarScan_0.3_SNP.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('SNPs_Plot.pdf')
plt.savefig('SNPs_Plot.png', dpi = 300)

# This code must be executed for 0.5 and 0.7 values for Strelka
    Somatic and VarScan Somatic VCF files.
```

Appendix E

Strelka Read Depth

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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,9-11 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed '/^#/d' Input.vcf > Output.vcf

# Considering the filtered VCF files with different Contamination
    tolerance as input
dff = pd.read_csv("Input_0.3.vcf", sep = '\t', index_col= False)
dff1 = pd.read_csv("Input_0.5.vcf", sep = '\t', index_col= False)
dff2 = pd.read_csv("Input_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']
```

```
# Concatenating 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 reorganizing 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)
```

```
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)

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)

# 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-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’], axis=1)

dff1 = dff1.drop([’NORMAL’, ’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’], axis=1)

dff2 = dff2.drop([’NORMAL’, ’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’], axis=1)

# Replacing ’.’ values with ’0’
dff.replace(’.’, ’0’, inplace=True)
dff1.replace(’.’, ’0’, inplace=True)
```

```
dff2.replace('.', '0', inplace=True)

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

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

dff2['Normal_Read_Depth'] = dff2['Normal_Read_Depth'].astype(int)
dff2['Tumor_Read_Depth'] = dff2['Tumor_Read_Depth'].astype(int)
print(dff)
print(dff1)
print(dff2)

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

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

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

# Saving the results in csv files.
dff.to_csv('Strelka3_Read_Depth.csv', sep=',', index = None)
dff1.to_csv('Strelka5_Read_Depth.csv', sep=',', index = None)
dff2.to_csv('Strelka7_Read_Depth.csv', sep=',', index = None)

# Converting the data to int.
dff = dff.drop(['CHROM_POS'], axis=1)
dff = dff.astype('int')
print(dff)
```

```
dff1 = dff1.drop(['CHROM_POS'], axis=1)
dff1 = dff1.astype('int')
print(dff1)

dff2 = dff2.drop(['CHROM_POS'], axis=1)
dff2 = dff2.astype('int')
print(dff2)

# Normality Test
# dk = Second['Third_Strelka_Normal']
# dkk = dk.hist()
# dkk.figure.savefig('Strelka_Normal_Histogram.png', dpi = 300)

# Finding the minimum values
min1 = dff['Normal_Read_Depth'].min()
min2 = dff['Tumor_Read_Depth'].min()
min3 = dff1['Normal_Read_Depth'].min()
min4 = dff1['Tumor_Read_Depth'].min()
min5 = dff2['Normal_Read_Depth'].min()
min6 = dff2['Tumor_Read_Depth'].min()

# Finding the maximum values
max1 = dff['Normal_Read_Depth'].max()
max2 = dff['Tumor_Read_Depth'].max()
max3 = dff1['Normal_Read_Depth'].max()
max4 = dff1['Tumor_Read_Depth'].max()
max5 = dff2['Normal_Read_Depth'].max()
max6 = dff2['Tumor_Read_Depth'].max()

# Finding the mean values
mean1 = dff['Normal_Read_Depth'].mean()
mean2 = dff['Tumor_Read_Depth'].mean()
mean3 = dff1['Normal_Read_Depth'].mean()
mean4 = dff1['Tumor_Read_Depth'].mean()
mean5 = dff2['Normal_Read_Depth'].mean()
mean6 = dff2['Tumor_Read_Depth'].mean()

# Finding the median values
median1 = dff['Normal_Read_Depth'].median()
```

```
median2 = dff[’Tumor_Read_Depth’].median()
median3 = dff1[’Normal_Read_Depth’].median()
median4 = dff1[’Tumor_Read_Depth’].median()
median5 = dff2[’Normal_Read_Depth’].median()
median6 = dff2[’Tumor_Read_Depth’].median()

# Finding the mode values
mode1 = dff[’Normal_Read_Depth’].mode()
mode2 = dff[’Tumor_Read_Depth’].mode()
mode3 = dff1[’Normal_Read_Depth’].mode()
mode4 = dff1[’Tumor_Read_Depth’].mode()
mode5 = dff2[’Normal_Read_Depth’].mode()
mode6 = dff2[’Tumor_Read_Depth’].mode()

# Finding the standard deviation values
std1 = dff[’Normal_Read_Depth’].std()
std2 = dff[’Tumor_Read_Depth’].std()
std3 = dff1[’Normal_Read_Depth’].std()
std4 = dff1[’Tumor_Read_Depth’].std()
std5 = dff2[’Normal_Read_Depth’].std()
std6 = dff2[’Tumor_Read_Depth’].std()

# 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]
Std_Value = [std1, std2, std3, std4, std5, std6]

# 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
df[’SD_Value’] = Std_Value

# Collecting it into a dataframe.
print(df)

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

# Total count
dff_Count = len(dff[’Normal_Read_Depth’].index)
dff1_Count = len(dff1[’Normal_Read_Depth’].index)
dff2_Count = len(dff2[’Normal_Read_Depth’].index)

# Selecting the range of values
dff_Normal_Lower = df[’Mean_Value’].iloc[0] -
    df[’SD_Value’].iloc[0]
dff_Normal_Higher = df[’Mean_Value’].iloc[0] +
    df[’SD_Value’].iloc[0]
dff_Tumor_Lower = df[’Mean_Value’].iloc[1] - df[’SD_Value’].iloc[1]
dff_Tumor_Higher = df[’Mean_Value’].iloc[1] +
    df[’SD_Value’].iloc[1]

dff1_Normal_Lower = df[’Mean_Value’].iloc[2] -
    df[’SD_Value’].iloc[2]
dff1_Normal_Higher = df[’Mean_Value’].iloc[2] +
    df[’SD_Value’].iloc[2]
dff1_Tumor_Lower = df[’Mean_Value’].iloc[3] -
    df[’SD_Value’].iloc[3]
dff1_Tumor_Higher = df[’Mean_Value’].iloc[3] +
    df[’SD_Value’].iloc[3]

dff2_Normal_Lower = df[’Mean_Value’].iloc[4] -
    df[’SD_Value’].iloc[4]
dff2_Normal_Higher = df[’Mean_Value’].iloc[4] +
    df[’SD_Value’].iloc[4]
```

```
dff2_Tumor_Lower = df['Mean_Value'].iloc[5] -
   df['SD_Value'].iloc[5]
dff2_Tumor_Higher = df['Mean_Value'].iloc[5] +
   df['SD_Value'].iloc[5]

# Filtering the data
dff = dff.loc[(dff['Normal_Read_Depth'] >= dff_Normal_Lower) &
    (dff['Normal_Read_Depth'] <= dff_Normal_Higher) &
    (dff['Tumor_Read_Depth'] >= dff_Tumor_Lower) &
    (dff['Tumor_Read_Depth'] <= dff_Tumor_Higher)]
dff1 = dff1.loc[(dff1['Normal_Read_Depth'] >= dff1_Normal_Lower) &
    (dff1['Normal_Read_Depth'] <= dff1_Normal_Higher) &
    (dff1['Tumor_Read_Depth'] >= dff1_Tumor_Lower) &
    (dff1['Tumor_Read_Depth'] <= dff1_Tumor_Higher)]
dff2 = dff2.loc[(dff2['Normal_Read_Depth'] >= dff2_Normal_Lower) &
    (dff2['Normal_Read_Depth'] <= dff2_Normal_Higher) &
    (dff2['Tumor_Read_Depth'] >= dff2_Tumor_Lower) &
    (dff2['Tumor_Read_Depth'] <= dff2_Tumor_Higher)]
print(dff)
print(dff1)
print(dff2)

# Final counts
dff_Counts = len(dff['Normal_Read_Depth'].index)
dff1_Counts = len(dff1['Normal_Read_Depth'].index)
dff2_Counts = len(dff2['Normal_Read_Depth'].index)

# Filtered reads
dff_Toll = dff_Count - dff_Counts
dff1_Toll = dff1_Count - dff1_Counts
dff2_Toll = dff2_Count - dff2_Counts

# Converting the values to a list
a1 = [dff_Counts, dff_Toll]
a2 = [dff1_Counts, dff1_Toll]
a3 = [dff2_Counts, dff2_Toll]

# Getting plots for each Contamination tolerance value
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
```

```
ax.axis('equal')
langs = ['Selected_Count', 'Filtered_Count']
explode = (0.1, 0)
colors = ['#FFD700','#FFAA1C']
ax.pie(a1, explode=explode, labels = langs, colors=colors,
   autopct='%1.2f%%')
ax.set_title('Read Depth Counts Percentage')
plt.savefig('Strelka3_Read_Depth.png', dpi = 300)

fig1 = plt.figure()
ax1 = fig1.add_axes([0,0,1,1])
ax1.axis('equal')
langs1 = ['Selected_Count', 'Filtered_Count']
explode1 = (0.1, 0)
colors1 = ['#FFD700','#FFAA1C']
ax1.pie(a2, explode=explode1, labels = langs1, colors=colors1,
   autopct='%1.2f%%')
ax1.set_title('Read Depth Counts Percentage')
plt.savefig('Strelka5_Read_Depth.png', dpi = 300)

fig2 = plt.figure()
ax2 = fig2.add_axes([0,0,1,1])
ax2.axis('equal')
langs2 = ['Selected_Count', 'Filtered_Count']
explode2 = (0.1, 0)
colors2 = ['#FFD700','#FFAA1C']
ax2.pie(a3, explode=explode2, labels = langs2, colors=colors2,
   autopct='%1.2f%%')
ax2.set_title('Read Depth Counts Percentage')
plt.savefig('Strelka7_Read_Depth.png', dpi = 300)
```

Truth Data Read Depth

```
# 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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,9-11 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed ’/^#/d’ Input.vcf > Output.vcf

# Considering the filtered VCF file as input
dff = pd.read_csv("Truth.vcf", sep = ’\t’, index_col= False)

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

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

# Dropping of the unnecessary columns and reorganizing 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)

# Converting the data to Integers.
dff = dff.drop(['CHROM_POS'], axis=1)
dff = dff.astype('int')
print(dff)

# Normality Test
dk = dff['Read_Depth']
dkk = dk.hist()
dkk.figure.savefig('Truth_Data_Histogram.png', dpi = 300)

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

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

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

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

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

# Finding the standard deviation value
std1 = dff['Read_Depth'].std()

# Declaring 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]
SD_Value = [std1]

# 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
df['SD_Value'] = SD_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)

# Total count
Counts = len(dff['Read_Depth'].index)
print('Total number of reads')
print(Counts)

# Selecting the range of values
SD_Lower = df['Mean_Value'].iloc[0] - df['SD_Value'].iloc[0]
SD_Higher = df['Mean_Value'].iloc[0] + df['SD_Value'].iloc[0]

# Filtering the data
dff = dff.loc[(dff['Read_Depth'] >= SD_Lower) & (dff['Read_Depth']
    <= SD_Higher)]
print(dff)

# Final count
Count = len(dff['Read_Depth'].index)
print('Selected number of selected reads')
print(Count)

# Converting the values to a list
```

```
a1 = [Counts, Count]

# Getting plot
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis(’equal’)
langs = [’Selected_Count’, ’Filtered_Count’]
explode = (0.1, 0)
colors = [’#FFD700’,’#FFAA1C’]
ax.pie(a1, explode=explode, labels = langs, colors=colors,
   autopct=’%1.2f%%’)
ax.set_title(’Read Depth Counts Percentage’)
plt.savefig(’Truth_Data_Read_Depth.png’, dpi = 300)
```

VarScan Read Depth

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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,9-11 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed ’/^#/d’ Input.vcf > Output.vcf

# Updating filtered VCF files with different Tumor purity values
dff = pd.read_csv("Input_0.3.vcf", sep = ’\t’, index_col= False)
dff1 = pd.read_csv("Input_0.5.vcf", sep = ’\t’, index_col= False)
dff2 = pd.read_csv("Input_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’]

# Concatenating 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 reorganizing 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)

# Saving the results in csv.
dff.to_csv(’VarScan3_Read_Depth.csv’, sep=’,’, index = None)
dff1.to_csv(’VarScan5_Read_Depth.csv’, sep=’,’, index = None)
dff2.to_csv(’VarScan7_Read_Depth.csv’, sep=’,’, index = None)

# 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)

# Second.columns = ['CHROM_POS', 'Third_VarScan_Normal',
    'Third_VarScan_Tumor', 'Fifth_VarScan_Normal',
    'Fifth_VarScan_Tumor', 'Seventh_VarScan_Normal',
    'Seventh_VarScan_Tumor']

# Converting the data to Integers.
# Second = Second.drop(['CHROM_POS'], axis=1)
# Second = Second.astype('int')
# print(Second)

dff = dff.drop(['CHROM_POS'], axis=1)
dff = dff.astype('int')
print(dff)

dff1 = dff1.drop(['CHROM_POS'], axis=1)
dff1 = dff1.astype('int')
print(dff1)

dff2 = dff2.drop(['CHROM_POS'], axis=1)
dff2 = dff2.astype('int')
print(dff2)

# Normality Test
# dk = Second['Third_VarScan_Normal']
# dkk = dk.hist()
# dkk.figure.savefig('VarScan_Normal_Histogram.png', dpi = 300)
# Finding the minimum values
min1 = dff['Normal_Read_Depth'].min()
min2 = dff['Tumor_Read_Depth'].min()
min3 = dff1['Normal_Read_Depth'].min()
min4 = dff1['Tumor_Read_Depth'].min()
min5 = dff2['Normal_Read_Depth'].min()
min6 = dff2['Tumor_Read_Depth'].min()

# Finding the maximum values
max1 = dff['Normal_Read_Depth'].max()
max2 = dff['Tumor_Read_Depth'].max()
max3 = dff1['Normal_Read_Depth'].max()
max4 = dff1['Tumor_Read_Depth'].max()
max5 = dff2['Normal_Read_Depth'].max()
max6 = dff2['Tumor_Read_Depth'].max()

# Finding the mean values
mean1 = dff['Normal_Read_Depth'].mean()
mean2 = dff['Tumor_Read_Depth'].mean()
mean3 = dff1['Normal_Read_Depth'].mean()
mean4 = dff1['Tumor_Read_Depth'].mean()
mean5 = dff2['Normal_Read_Depth'].mean()
mean6 = dff2['Tumor_Read_Depth'].mean()

# Finding the median values
median1 = dff['Normal_Read_Depth'].median()
median2 = dff['Tumor_Read_Depth'].median()
median3 = dff1['Normal_Read_Depth'].median()
median4 = dff1['Tumor_Read_Depth'].median()
median5 = dff2['Normal_Read_Depth'].median()
median6 = dff2['Tumor_Read_Depth'].median()

# Finding the mode values
mode1 = dff['Normal_Read_Depth'].mode()
mode2 = dff['Tumor_Read_Depth'].mode()
mode3 = dff1['Normal_Read_Depth'].mode()
mode4 = dff1['Tumor_Read_Depth'].mode()
mode5 = dff2['Normal_Read_Depth'].mode()
mode6 = dff2['Tumor_Read_Depth'].mode()
```

```
# Finding the standard deviation values
std1 = dff['Normal_Read_Depth'].std()
std2 = dff['Tumor_Read_Depth'].std()
std3 = dff1['Normal_Read_Depth'].std()
std4 = dff1['Tumor_Read_Depth'].std()
std5 = dff2['Normal_Read_Depth'].std()
std6 = dff2['Tumor_Read_Depth'].std()

# 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]
Std_Value = [std1, std2, std3, std4, std5, std6]

# 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
df['SD_Value'] = Std_Value

# Collecting it into a dataframe.
print(df)

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

# Total count
```

```
dff_Count = len(dff[’Normal_Read_Depth’].index)
dff1_Count = len(dff1[’Normal_Read_Depth’].index)
dff2_Count = len(dff2[’Normal_Read_Depth’].index)

# Selecting the range of values
dff_Normal_Lower = df[’Mean_Value’].iloc[0] -
    df[’SD_Value’].iloc[0]
dff_Normal_Higher = df[’Mean_Value’].iloc[0] +
    df[’SD_Value’].iloc[0]
dff_Tumor_Lower = df[’Mean_Value’].iloc[1] - df[’SD_Value’].iloc[1]
dff_Tumor_Higher = df[’Mean_Value’].iloc[1] +
    df[’SD_Value’].iloc[1]

dff1_Normal_Lower = df[’Mean_Value’].iloc[2] -
    df[’SD_Value’].iloc[2]
dff1_Normal_Higher = df[’Mean_Value’].iloc[2] +
    df[’SD_Value’].iloc[2]
dff1_Tumor_Lower = df[’Mean_Value’].iloc[3] -
    df[’SD_Value’].iloc[3]
dff1_Tumor_Higher = df[’Mean_Value’].iloc[3] +
    df[’SD_Value’].iloc[3]

dff2_Normal_Lower = df[’Mean_Value’].iloc[4] -
    df[’SD_Value’].iloc[4]
dff2_Normal_Higher = df[’Mean_Value’].iloc[4] +
    df[’SD_Value’].iloc[4]
dff2_Tumor_Lower = df[’Mean_Value’].iloc[5] -
    df[’SD_Value’].iloc[5]
dff2_Tumor_Higher = df[’Mean_Value’].iloc[5] +
    df[’SD_Value’].iloc[5]

# Filtering the data
dff = dff.loc[(dff[’Normal_Read_Depth’] >= dff_Normal_Lower) &
    (dff[’Normal_Read_Depth’] <= dff_Normal_Higher) &
    (dff[’Tumor_Read_Depth’] >= dff_Tumor_Lower) &
    (dff[’Tumor_Read_Depth’] <= dff_Tumor_Higher)]
dff1 = dff1.loc[(dff1[’Normal_Read_Depth’] >= dff1_Normal_Lower) &
    (dff1[’Normal_Read_Depth’] <= dff1_Normal_Higher) &
    (dff1[’Tumor_Read_Depth’] >= dff1_Tumor_Lower) &
    (dff1[’Tumor_Read_Depth’] <= dff1_Tumor_Higher)]
```

```
dff2 = dff2.loc[(dff2['Normal_Read_Depth'] >= dff2_Normal_Lower) &
    (dff2['Normal_Read_Depth'] <= dff2_Normal_Higher) &
    (dff2['Tumor_Read_Depth'] >= dff2_Tumor_Lower) &
    (dff2['Tumor_Read_Depth'] <= dff2_Tumor_Higher)]
print(dff)
print(dff1)
print(dff2)

# Final counts
dff_Counts = len(dff['Normal_Read_Depth'].index)
dff1_Counts = len(dff1['Normal_Read_Depth'].index)
dff2_Counts = len(dff2['Normal_Read_Depth'].index)

# Filtered reads
dff_Toll = dff_Count - dff_Counts
dff1_Toll = dff1_Count - dff1_Counts
dff2_Toll = dff2_Count - dff2_Counts

# Converting the values to a list
a1 = [dff_Counts, dff_Toll]
a2 = [dff1_Counts, dff1_Toll]
a3 = [dff2_Counts, dff2_Toll]

# Getting plots for each Tumor purity value
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['Selected_Count', 'Filtered_Count']
explode = (0.1, 0)
colors = ['#FFD700','#FFAA1C']
ax.pie(a1, explode=explode, labels = langs, colors=colors,
    autopct='%1.2f%%')
ax.set_title('Read Depth Counts Percentage')
plt.savefig('VarScan3_Read_Depth.png', dpi = 300)

fig1 = plt.figure()
ax1 = fig1.add_axes([0,0,1,1])
ax1.axis('equal')
langs1 = ['Selected_Count', 'Filtered_Count']
explode1 = (0.1, 0)
```

```
colors1 = ['#FFD700','#FFAA1C']
ax1.pie(a2, explode=explode1, labels = langs1, colors=colors1,
   autopct='%1.2f%%')
ax1.set_title('Read Depth Counts Percentage')
plt.savefig('VarScan5_Read_Depth.png', dpi = 300)

fig2 = plt.figure()
ax2 = fig2.add_axes([0,0,1,1])
ax2.axis('equal')
langs2 = ['Selected_Count', 'Filtered_Count']
explode2 = (0.1, 0)
colors2 = ['#FFD700','#FFAA1C']
ax2.pie(a3, explode=explode2, labels = langs2, colors=colors2,
   autopct='%1.2f%%')
ax2.set_title('Read Depth Counts Percentage')
plt.savefig('VarScan7_Read_Depth.png', dpi = 300)
```

Appendix F

Strelka Variants

```
# 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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,4-5 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed ’/^#/d’ Input.vcf > Output.vcf

# Updating the filtered VCF files
dff = pd.read_csv("Input_0.3_SNV.vcf", sep = ’\t’, index_col=
    False)
dff1 = pd.read_csv("Input_0.5_SNV.vcf", sep = ’\t’, index_col=
    False)
dff2 = pd.read_csv("Input_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’]

# Concatenating 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 reorganizing 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)
```

Truth Data Variants

```
# 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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,4-5 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed ’/^#/d’ Input.vcf > Output.vcf

# Updating the filtered VCF files
dff = pd.read_csv("Truth_SNP.vcf", sep = ’\t’, index_col= False)

# Renaming the column names.
dff.columns = [’CHROM’, ’POS’, ’REF’, ’ALT’]

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

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

# Renaming the column names.
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)

# Declaring 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 the csv files and concatenating "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)
```

VarScan Variants

```
# 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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,4-5 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed '/^#/d' Input.vcf > Output.vcf

# Updating the filtered VCF files
dff = pd.read_csv("Input_0.3_SNP.vcf", sep = '\t', index_col=
    False)
dff1 = pd.read_csv("Input_0.5_SNP.vcf", sep = '\t', index_col=
    False)
dff2 = pd.read_csv("Input_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 reorganizing 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)

# Renaming the column names.
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)

# Declaring 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’])

# Renaming the column names.
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 the csv files and concatenating "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)
```

Variants 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

# Selecting the necessary columns in the VCF file
# cut -f 1-2,4-5 Input.vcf > Output.vcf

# Removing the header information in the VCF file
# sed ’/^#/d’ Input.vcf > Output.vcf
# Updating the filtered VCF files with same Contamination
    tolerance and Tumor purity values
dff = pd.read_csv("Strelka_0.3_SNV.vcf", sep = ’\t’, index_col=
    False)
dff1 = pd.read_csv("VarScan_0.3_SNP.vcf", sep = ’\t’, index_col=
    False)
dff2 = pd.read_csv("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

# Declaring 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 concatenating "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)
```

Appendix G

VarScan Somatic Customization

```
- Reference Genome: Human (Homo sapiens): hg19
- Aligned reads from Normal sample: Data input ’normal_bam’ (bam)
- Aligned reads from Tumor sample: Data input ’tumor_bam’ (bam)
- Estimated purity (non-tumor content) of normal sample
- Estimated purity (tumor content) of tumor sample
- Generate separate output datasets for SNP and indel calls? No
- Minimum base quality: 28
- Minimum mapping quality: 1
- Use reads from anomalously mapped reads: No
- Try to correct for read-pair overlaps: Yes
- Maximum number of reads per site: 8000
- Minimum coverage: 8
- Minimum supporting reads: 2
- Minimum variant allele frequency: 0.1.
- Minimum homozygous variant allele frequency: 0.75
- p-value threshold for calling variants: 0.99
- P-value threshold for calling somatic variants and LOH events:
   0.05
- Minimum number of variant-supporting reads: 4
- Low coverage minimum number of variant-supporting reads: 2
- Minimum variant allele count threshold: 10
- Minimum variant allele frequency: 0.1
- Minimum relative variant position in ref-supporting reads: 0.1
- Minimum relative variant position in variant-supporting reads:
   0.1
- Minimum distance of variant site from 3’-end of ref-supporting
   reads: 0.1
```

```
- Minimum distance of variant site from 3'-end of
   variant-supporting reads: 0.1
- Minimum length of ref-supporting reads: 90
- Minimum length of variant-supporting reads: 90
- Maximum relative read length difference: 0.25
- Minimum fraction of variant reads from each strand: 0.01
- Minimum variant allele depth required to apply the
   --min-strandedness filter: 5
- Minimum average base quality for the ref allele: 15
- Minimum average base quality for the variant allele: 15
- Maximum base quality difference between ref- and variant
   supporting reads: 50
- Minimum average mapping quality of ref-supporting reads: 28
- Minimum average mapping quality of variant-supporting reads: 28
- Maximum mapping quality difference between ref- and
   variant-supporting reads: 50
- Maximum mismatch base quality sum of ref-supporting reads: 100
- Maximum mismatch base quality sum of var-supporting reads: 100
- Minimum difference between mismatch base quality sums of
   variant- and ref- supporting reads: 0
- Maximum difference between mismatch base quality sums of
   variant- and ref- supporting reads: 50
```

Strelka Somatic Customization

```
- Reference Genome: Human (Homo sapiens): hg19
- Aligned reads from Normal sample: Data input 'normal_bam' (bam)
- Aligned reads from Tumor sample: Data input 'tumor_bam' (bam)
- Call variants across: Whole reference
- Generate compressed variants output (vcf.gz): No
- Generate bed file describing somatic callable regions of the
   genome: No
- Set maximum reported indel size: 49
- Set depth Filter Multiple: 3.0
- Set snv Max Filtered Basecall Frac: 0.4
- Set snv Max Spannin Deletion Frac: 0.75
- Set indel Max Window Filtered Basecall Frac: 0.3
- Set ssnv Prior: 0.0001
- Set sindel Prior: 0.000001
- Set ssnv Noise: 5e-10
- Set sindel Noise Factor: 2.2
- Set ssnv Noise Strand Bias Frac: 0
- Set minTier1 Mapq: 20
- Set minTier2 Mapq: 0
- Set ssnvQuality_LowerBound: 15
- Set sindelQuality_LowerBound: 40
```

Appendix H

Non Parameterized Strelka Benchmarking

Given below are the result of Non Parameterized Strelka Somatic variant caller when benchmarked with respect to Truth Data VCF.

Type	**Total**	**TP**	**TTP**	**TFP**	**TN**	**FP**	**FN**
Strelka	11,14,341	3,791	3,292	499	0	9,555	11,00,995

Table 7.1: Benchmarking Strelka ALTs with respect to the Truth Data ALTs where TP represents True Positive ALTs, TTP represents True True Positive ALTs, TFP represents True False Positive ALTs, TNA represents True Negative ALTs, FP represents False Positive ALTs, and FN represents False Negative ALTs.

Prior Probability and Contamination Tolerance Strelka Benchmarking

Given below are the result of Strelka Somatic variant caller when benchmarked with respect to Truth Data VCF with `Set ssnvPrior`, `Set INDELPrior`, `Set ssnvContamTolerance` & `Set INDELContamTolerance` values as 0.3, 0.5 and 0.7.

Value	**Total**	**TP**	**TTP**	**TFP**	**TN**	**FP**	**FN**
0.3	2,263,239	15,755	9,079	6,676	0	1,158,361	1,089,123
0.5	3,096,311	20,959	10,343	10,616	0	1,991,351	1,084,001
0.7	4,607,708	30,525	12,627	17,898	0	3,502,586	1,074,597

Table 7.2: Benchmarking Strelka ALTs with respect to the Truth Data ALTs where Value represents Contamination Tolerances and Prior probability of a somatic SNV and INDEL, TP represents True Positive ALTs, TTP represents True True Positive ALTs, TFP represents True False Positive ALTs, TNA represents True Negative ALTs, FP represents False Positive ALTs, and FN represents False Negative ALTs.