Automatically Annotating Cancer Variants using public databases

A project report submitted to the
Graduate School of Natural and Applied Sciences

by

Kanza Batool Haider

in partial fulfillment for the
degree of MS. Data Sciences.

İSTANBUL
ŞEHİR
ÜNIVERSITY

This is to certify that we have read this report and that in our opinion it is fully adequate, in scope and quality, as a Graduate Project for the degree of Master of Science in Data Science

APPROVED BY:

Asst Prof. Dr. Mehmet Baysan   . . . .
.(Grad project Advisor)

This is to confirm that this report complies with all the standards set by the   Graduate School of Natural and Applied Sciences of İstanbul Şehir University:

DATE OF APPROVAL:

SEAL/SIGNATURE:

Declaration of Authorship

I, Kanza Batool Haider, declare that this graduate project titled, ‘Automatically annotating cancer variants using public databases’ and the work presented in it are my own. I confirm that:

This work was done wholly or mainly while in candidature for a research degree at this University.

Where any part of this graduate project has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

Where I have consulted the published work of others, this is always clearly at- tribute.

Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this graduate project is entirely my own work.

I have acknowledged all main sources of help.

Where the project is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

Automatically annotating cancer variants using public databases

Kanza Batool HAIDER

Abstract

This project has been made in line to aid physicians in identifying the most critical cancer-causing mutations in the patients (Homosapien GRCh37). A user with a sample vcf input file shall be able to annotate the variants by downloading Variant Effect Predictor (VEP) and running a single command based on a perl script. The output annotated file generated shall then be used to extract information for somatic variants only and matched across the data resource of COSMIC. The mutation data shall be compared across one another such that the most critical mutations get on to the top of the list based on their FATHMM score, number of samples reported in COSMIC and number of references for the particular mutation as the priority filter. Finally, top 10 most critical mutations shall be displayed in the output report consisting information with regard to their Gene name, mutation location, type of mutation, cancer-causing potential score, tissue distribution, pathways affected and the number of samples and references present with further integrated details linked with COSMIC.

Keywords: Mutation, Annotation, Homosapien

Acknowledgments

I would like to sincerely thank my professor and project advisor, Dr Mehmet Baysan, for having me accepted for this work in his lab. This project has led me to step in the field of bioinformatics, that for long has been my sight for attraction. Under his mentorship, I learned immensely and it was great honour to work under him.

I would also thank my colleagues from the Baysan lab, Batuhan Kisakol and Sahin Sarihan for their kind and all time ready-to-help attitude. They assisted me with their suggestions during the early research phase when I was struggling with project synopsis. They would also listen and review my work, giving away very helpful feedbacks.

My utmost gratitude for all my respectable professors under whom I got trained during the span of my masters, Dr Mehmet Serkan Apaydin, Dr Ahmet Bulut, Dr Prof Ensar Gul, Dr Prof Selim Zaim, Dr Yasin Ulukus, Dr Bahadir Tunaboylu and Dr Prof. Ömer Cebeci. I am also grateful to Istanbul Sehir University for blessing on me this wonderful opportunity of a lifetime in this beautiful campus of Dragos.

Special thanks for my dearest friends who made my time in Turkey memorable. Zeynep Yikilmaz, Quratulain Ahmed, Sena, Senada, Halit, Noha and my big brothers Abdullah and Sikandar who helped me through my initial time of settlement with all the registration and documental processes.

Finally, I owe huge debt of gratitude for my beloved parents Naseem Haider and Shabnam Haider, my genius siblings, Batool Arhamna and Muhammad Ali and my beloved husband, Noor Ali Jafri. They gave me everything I needed to accomplish this far. Without their support, love and trust this was never ever possible.

Contents

List of Figures

List of Tables

Abbreviations

UML	**U**nified **M**odelling **L**anguage
COSMIC	**C**atalogue **O**f **S**omatic **M**utations **I**n **C**ancer
VCF	**V**ariant **C**all **F**ormat
CNV	**C**opy **N**umber **V**ariation
TCGA	**T**he **C**ancer **G**enomic **A**tlas
SNP	**S**ingle **N**ucleotide **P**olymorphism
HGNC	**H**UGO **G**ene **N**omenclature **C**ommittee
HGVS	**H**uman **G**enome **V**ariation **S**ociety
VEP	**V**ariant **E**ffect **P**redictor
SIFT	**S**orting **I**ntolerant **F**rom **T**olerant
AWS	**A**mazon **W**eb **S**ervices

1. Introduction

1.1 Introduction

Cancer has caused many lives to date. It is the second most common cause of death in the United States with mortality rate spiking up to 50%. According to the International Agency for Research on Cancer, in 2012, there were 14.1 million new cases and 8.2 million cancer-related deaths worldwide. The number of new cancer cases per year is expected to rise to 23.6 million by 2030 [1]. In such an extreme scenario, it is the need of time that the world unites together to "Wage war against cancer" like president Nixon of United States stated in 1971. There is an intense level of research being conducted across the world in the field of Medical Science. Every discovery and attempt to eradicate cancer significantly stand at its place, playing a valuable role for the rest of humanity and pushing it to the ultimate goal. All that has been done, however, has still not been able to find the exact antidote for the disease. Cancer itself is a highly complex disease whereby 200+ different types of it exists. The underlying cause behind tumor occurrence has been identified due to genetic aberrations such as somatic mutations, changed gene expression profiles, copy number variations (CNV), and different epigenetic generations. These causes conform to the fact that a better understanding of the genetics strongly correlates with the diagnosis and treatment of cancer.

In 2005 TCGA and in 2008 International Cancer genome consortium were launched as the two main projects that aimed to catalogue genetic mutations responsible for cancer and discover major cancer causal genome alterations in large cohorts. Through high throughput sequencing technology, genomic characterization and sequence analysis are done. A major goal of the project was to provide publicly available datasets to help improve diagnostic methods, treatment standards, and finally to prevent cancer [2].

The complete sequencing of the human genome denoted a hall-mark achievement in modern science, provided by these publicly available data resources, such as the Cancer Genome Atlas Project (TCGA), 1000 Genomes, GENCODE, UniProt, dbSNP, RefSeq etc yet it additionally delivered an entirely new set of challenges in illustrating the functions and interactions of various parts of the genome. A characteristic initial step in handling these considerable assignments is to develop construct an annotation of the genome. Throughout the years, many experimental and computational methods have been developed in the form of annotation tools that utilizes various techniques (eg.machine learning) that helps predict the impact that an alteration has on the translated protein product of a gene and reveals much more information of interests. [3]

Examples of few annotation tools are ANNOVAR; it is a tool to annotate single nucleotide variants (SNVs) and insertions/deletions, such as examining their functional consequence on genes, inferring cytogenetic bands, reporting functional importance scores, finding variants in conserved regions, or identifying variants reported in the 1000 Genomes Project and dbSNP [4]. The Cancer-Related Analysis of Variants Toolkit (CRAVAT) is another tool which is an evolving suite of informatics tools for mutation interpretation that includes mutation mapping and quality control, impact prediction and extensive annotation, gene- and mutation level interpretation, including joint prioritization of all nonsilent mutation consequence types, and structural and mechanistic visualization. It employs two approaches for predicting mutation impact, namely Cancer-Specific High-Throughput Annotation of Somatic Mutations (CHASM) and Variant Effect Scoring Tool (VEST) [5]. Oncotator is also one of a tool that annotates genomic point mutations and short nucleotide insertions/deletions (indels) with variant- and gene-centric information specific to cancer researchers. This information is drawn from 14 different publicly available resources that have been pooled and indexed. Annotations linked to variants range from basic information, such as gene names and functional classification (e.g. missense), to cancer-specific data from resources such as the Catalogue of Somatic Mutations in Cancer (COSMIC), the Cancer Gene Census, and TCGA. [6]

Variant Effect Predictor (VEP), SNPeff, PhD-SNP, SuSPect, Jannovar etc are some of the other many annotation tools developed. These annotation tools are utilized further by researchers opening doors for more insightful research and for medical practitioners to differentiate among different types of mutations.

1.2 Project background overview

Given that the Genomic Pipeline for analyzing genomic sequence data has been already developed and functional in the Baysan Lab of Bioinformatics which intakes a raw FASTQ file (text file produced by sequencing machines) and ends with a VCF files (detailed files of discovered SNPs/Indels). The next step involves annotation, to process the vcf file consisting of variants and indicate for the endangered mutations specific to cancer.

This project tends to provide the annotation specific for cancer that can be integrated into the genomic pipeline making a complete one-shot solution for doctors and medical researchers to resort for interpretation of the sample genomic sequence to identify the most critical somatic variants.

1.3 Project description

The project provides a comprehensive way that does annotations utilizing Variant Effect Predictor (VEP) and accesses COSMIC's information for particular somatic mutations present in the input file to output details of the most critical somatic mutations. COSMIC is the specialized database that provides cancer specific curated data.

Fig.1 shows the summarizes form of the process flow of the project.

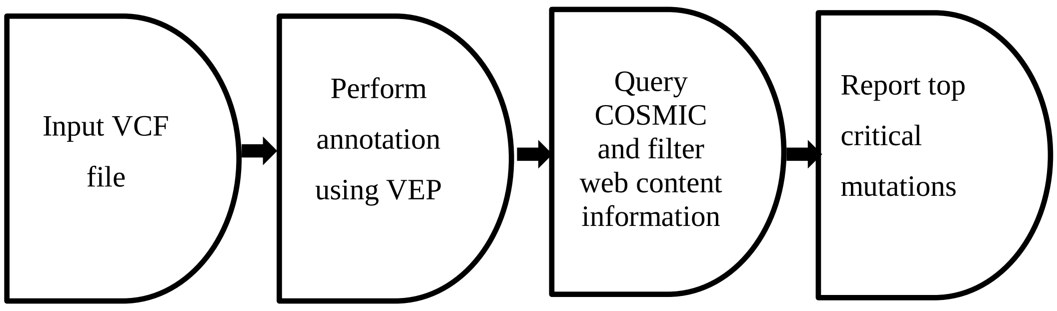

Input VCF file
Perform annotation using VEP
Query COSMIC and filter web content information
Report top critical mutations

Figure 1 Summarized flow diagram of the project steps

1.3.1 Annotation

1.3.1.1 Variant Effect Predictor, VEP

Variant Effect Predictor, is Enseml tool comprising of standalone Perl scripts which is run to perform annotation. It does so by analyzing the variants and predict the functional consequences of known and unknown variants. It can annotate a large data, it is powerful, fast and extendable.

1.3.1.1.1 Annotation sources:

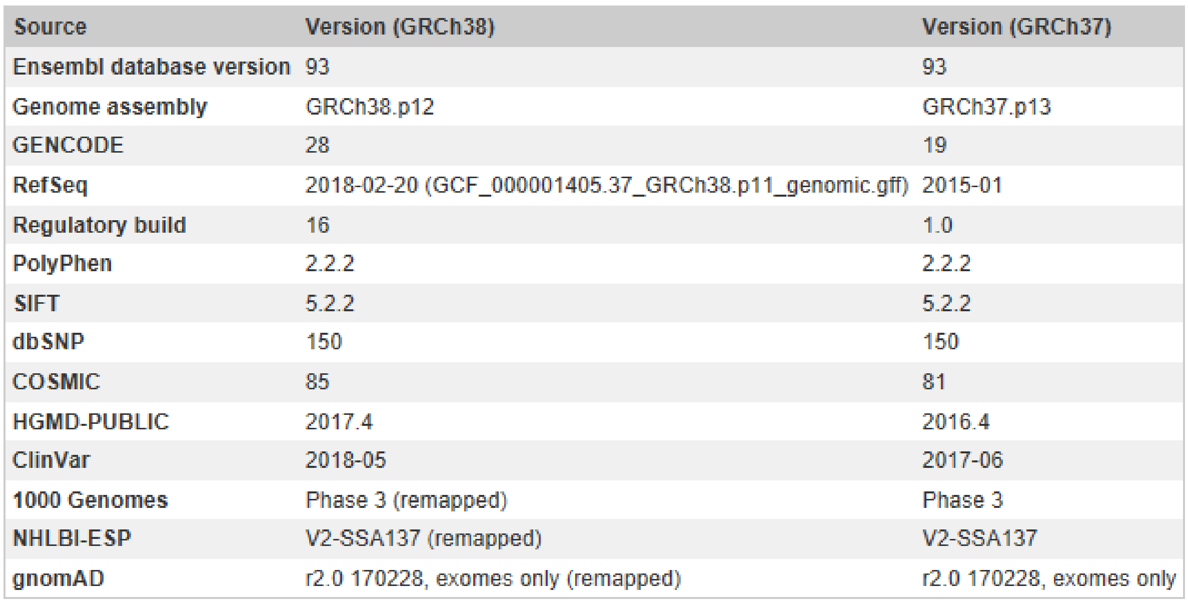
VEP can use a variety of annotation sources to retrieve the transcript models used to predict consequence types.

- ❑ Cache - a downloadable file containing all transcript models, regulatory features and variant data for species
- ❑ GFF or GTF - use transcript models defined in a tabix-indexed GFF or GTF file
- ❑ Database - connect to a MySQL database server hosting Ensembl databases

However, in this project, we have used --cache , as the annotation source because --cache is the fastest and most efficient way to use VEP. Only a single initial network connection is made and most data is read from local disk. --offline mode can be used to eliminate all network connections for speed.

1.3.1.1.2 Data in the Cache

Table 1. shows fourteen public databases stored within cache. In this project cache for Version GRCh37 has been used for annotations.



Source	Version (GRCh38)	Version (GRCh37)
Ensembl database version	93	93
Genome assembly	GRCh38.p12	GRCh37.p13
GENCODE	28	19
RefSeq	2018-02-20 (GCF_000001405.37_GRCh38.p11_genomic.gff)	2015-01
Regulatory build	16	1.0
PolyPhen	2.2.2	2.2.2
SIFT	5.2.2	5.2.2
dbSNP	150	150
COSMIC	85	81
HGMD-PUBLIC	2017.4	2016.4
ClinVar	2018-05	2017-06
1000 Genomes	Phase 3 (remapped)	Phase 3
NHLBI-ESP	V2-SSA137 (remapped)	V2-SSA137
gnomAD	r2.0 170228, exomes only (remapped)	r2.0 170228, exomes only

Table 1 VEP Annotation Sources

The cache stores the following information:

- ➢ Transcript location, sequence, exons and other attributes
- ➢ Gene, protein, HGNC and other identifiers for each transcript (where applicable, limitations apply to RefSeq caches)
- ➢ Locations, alleles and frequencies of existing variants
- ➢ Regulatory regions
- ➢ Predictions and scores for SIFT, PolyPhen

1.3.1.2 Database

1.3.1.2.1 COSMIC

COSMIC stands for the Catalogue for Somatic Mutations in Cancer. It is an online database consisting of information of thousands of somatic mutations which are implicated for developing cancer within humans. Somatic mutations are those that occur in non-germline cells which means it does not pass down the generation but are the result of a mutation in the genes due to different environmental factors. COSMIC curates these somatic data from mainly two resources. One is through the papers in the scientific literature and second through the whole genome resequencing studies of cancer samples undertaken by the Cancer Genome Project. The database is freely available to academic researchers and commercially licensed to others. [7]

For every somatic mutation, COSMIC provides an overview, tissue distribution, samples and references that further disintegrates to provide further information. In this project, we have utilized this information from the COSMIC site to display for the mutations that are present in the sample VCF file we annotate.

The Overview section provides information for the source of mutation i.e gene name/sample name/tissue name with a unique ID, and also shows the mutation syntax at the amino acid and nucleotide sequence level. The Tissue distribution section displays the distribution of mutated samples and tissue types. The Sample section displays a table of mutated samples, with tissue, histology and zygosity information. Publication information is also included, where available, with links to PUBMED. And finally, in the References section includes all the references of the particular mutation reported.

1.3.2 Project Scope

The scope of this project is:

- To perform annotation on a given input VCF file
- Identify most critical somatic mutations
- Report details of the critical mutations

1.3.3 Steps involved

The project development shall involve following steps onto which we shall go in detail in chapter 2. These are:

- Perform Annotation on input VCF file
- Extract string parameters from output annotated VCF
- Query Cosmic database and generate COSMIC IDs
- Parse COSMIC mutation pages
- Filter content and display top 10 critical mutations

2 Development Methodology

2.1 Project Architecture

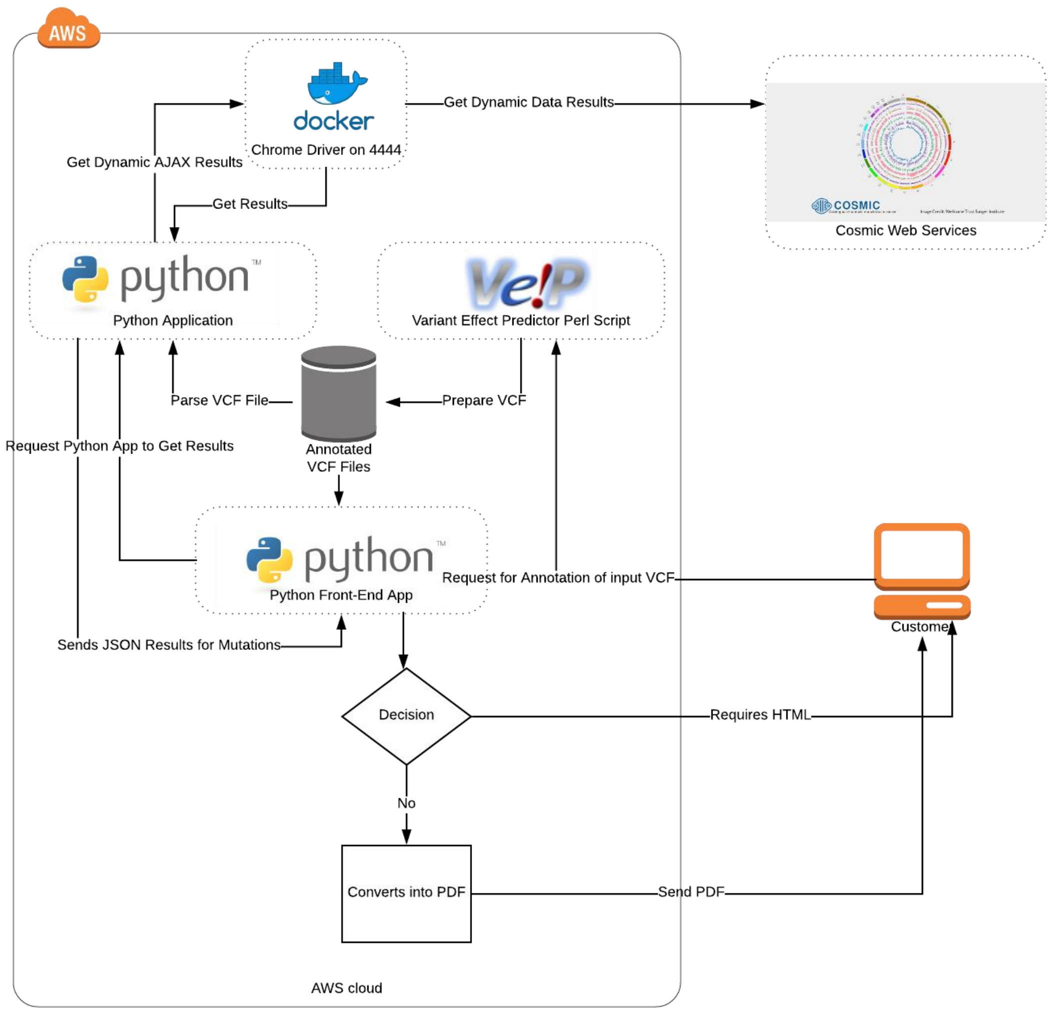


Fig 2. shows the full flow diagram.

Starting off with the request of Customer for annotating VCF file who essentially could be a medical practitioner or researcher. VEP standalone Perl scripts are utilized to annotate the file. This annotated file is then parsed to extract the query string parameters with the help of the back-end python application. Multiple queries are sent using the string parameters through COSMIC API to the COSMIC database. The COSMIC API retrieves COSMIC IDs for each somatic variant present in the input vcf file data. Several URLs are constructed based on the number of IDs generated. These URL are links to the mutation pages on the COSMIC website. Via a Docker, the content from the COSMIC's mutation pages is scrapped out. MiniChrome Driver running on Docker gets this data. Docker makes it runnable on Windows/Mac/Linux whereas Chrome Driver fetches the AJAX request. Docker enables the software to be portable. Upon receiving the data, it is processed via python back-end application. The content is ordered according to the highest mutations on top which are displayed along with details in the form of a report (HTML/PDF). The whole program is run on AWS to provide greater speed.

2.2 Prerequisites

Linux:
```
$ apt-get install wkhtml2pdf
```

Mac:
```
 $ brew install wkhtml2pdf
```

Python3:
```
$ brew install wkhtml2pdf:
$ pip install virtualenv
```

Docker:
Docker needs to be installed separately, which can be downloaded from here and after that these commands needs to run:

```
$ docker pull selenium/standalone-chrome
$ docker run -d -p 4444:4444 -v /dev/shm:/dev/shm selenium/standalone-
chrome:3.14.0-dubnium
```

Git:
```
$ git clone https://github.com/kanzabatool/Grad_project
$ cd Grad_project
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirement.txt
```

To Run Program
```
$ python main.py
```

2.3 Project Steps

The project can be divided into 5 steps. That are:

2.3.1 Perform Annotation on input VCF file using Variant Effect Predictor (VEP)

This step involves to run following commands first:

1. Download

```
git clone https://github.com/Ensembl/ensembl-vep.git
```

2. Install

```
cd ensembl-vep

perl INSTALL.pl
```

3. Test

```
./vep -i _filename_ /homo_sapiens_GRCh37.vcf --o _outputfilename_ --cache --force_overwrite --sift b --symbol --hgvs -- check_existing --tab --fields Uploaded_variation, Location, Allele, Gene, Feature, Consequences, SYMBOL, SIFT,HGVSc,SOMATIC --port 3337
```

Where,
`_filename_` is the file which has been requested for annotation. `_outputfilename_`_is the name of the output file.

`--cache`
Enables use of the cache.

`--force_overwrite`
By default, the script will fail with an error if the output file already exists. We force the overwrite of the existing file by using this flag.

`--sift`
Predicts whether an amino acid substitution affects protein function based on sequence homology and the physical properties of amino acids.

`--symbol`
Adds the gene symbol (e.g. HGNC) (where available) to the output.

`--hgvs`
Adds HGVS nomenclature based on Ensembl stable identifiers to the output. Both coding and protein sequence names are added where appropriate. To generate HGVS identifiers when using --cache or --offline we must use a FASTA file and --fasta. HGVS notations given on Ensembl identifiers are versioned.

`--check_existing`
Checks for the existence of known variants that are co-located with the input. By default the alleles are compared and variants on an allele-specific basis - to compare only coordinates, use --no_check_alleles. It's output field is SOMATIC.

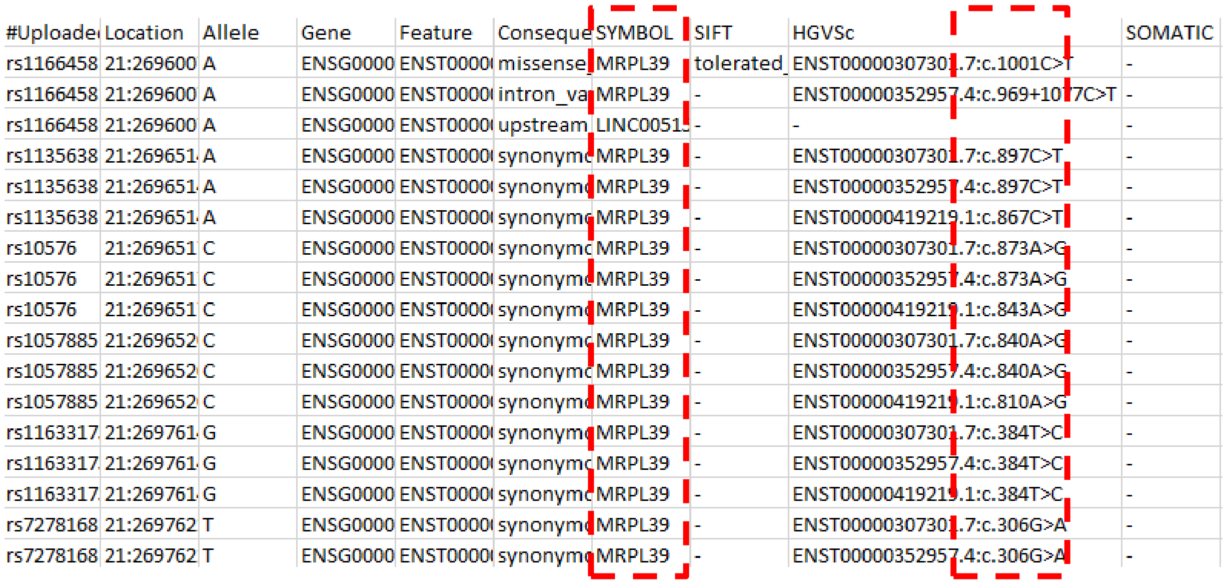
`--tab`
Writes output in tab-delimited format. Not used by default --fields [list]

`--fields [list]`
Configure the output format using a comma separated list of fields.
Can only be used with tab (--tab) or VCF format (--vcf) output.
In our case, we used Uploaded_variation, Location, Allele, Gene, Feature, Consequences, SYMBOL, SIFT, --filter "SOMATIC as our field output.

`--port 3337`
This connects to the GRCh37 version of the VEP for annotation
Additionally, using forking enables VEP to run multiple parallel "threads", with each thread processing a subset of our input. Most present day PCs have more than one processor core, so running VEP with forking can give enormous speed builds (3-4x quicker in most case). PCs with a single core will also get speed increases because of overheads related with utilizing object-oriented code in Perl. [8]

2.3.2 Extract string parameters from annotated VCF file

Once the vcf file has been annotated, the string parameters are extracted from it. The string parameters are basically the Gene names and HGVSc ID which are used in the later step for constructing of the COSMIC APIs to retrieve data of the somatic mutation present in the file.
Following image shows the Gene names and HGVS ID from a sample annotated vcf file that are extracted



#Uploade	Location	Allele	Gene	Feature	Conseque	SYMBOL	SIFT	HGVSc	SOMATIC
rs1166458	21:269600	A	ENSG0000	ENST0000	missense_	MRPL39	tolerated_	ENST00000307301.7:c.1001C>T	-
rs1166458	21:269600	A	ENSG0000	ENST0000	intron_va	MRPL39	-	ENST00000352957.4:c.969+1077C>T	-
rs1166458	21:269600	A	ENSG0000	ENST0000	upstream_	LINC0051	-	-	-
rs1135638	21:269651	A	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000307301.7:c.897C>T	-
rs1135638	21:269651	A	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000352957.4:c.897C>T	-
rs1135638	21:269651	A	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000419219.1:c.867C>T	-
rs10576	21:269651	C	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000307301.7:c.873A>G	-
rs10576	21:269651	C	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000352957.4:c.873A>G	-
rs10576	21:269651	C	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000419219.1:c.843A>G	-
rs1057885	21:269652	C	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000307301.7:c.840A>G	-
rs1057885	21:269652	C	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000352957.4:c.840A>G	-
rs1057885	21:269652	C	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000419219.1:c.810A>G	-
rs1163317	21:269761	G	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000307301.7:c.384T>C	-
rs1163317	21:269761	G	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000352957.4:c.384T>C	-
rs1163317	21:269761	G	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000419219.1:c.384T>C	-
rs7278168	21:269762	T	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000307301.7:c.306G>A	-
rs7278168	21:269762	T	ENSG0000	ENST0000	synonymo	MRPL39	-	ENST00000352957.4:c.306G>A	-

Figure 3 String parameters for querying COSMIC

2.3.3 Query Cosmic database and generate COSMIC IDs

The Sanger Institute has provided an API to retrieve information of COSMIC mutation data. This data has been reduced so now only one data record per Mutation ID is retrieved. [9]

COSMIC API Base URL:

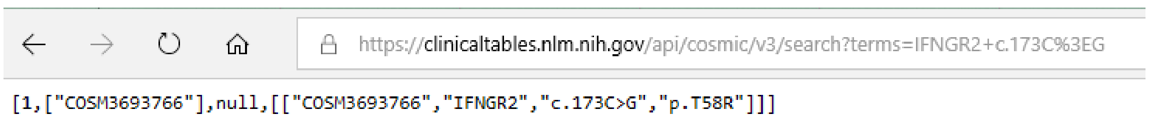
https://clinicaltables.nlm.nih.gov/api/cosmic/v3/search (+ query string parameters)

The query string parameters of COSMIC that we make use of in this project are the following
.

GeneName for eg: IFNGR2	The gene name for which the data has been curated in COSMIC. In most cases this is the accepted HGNC identifier.
HGNC_ID for eg: c.173C>G	If gene is in HGNC, this id helps linking it to HGNC.

Table 2 String parameters for querying COSMIC API

Once we run this COSMIC base API with its query string parameter, we retrieve the COSMIC ID for the particular somatic mutation. For example, Gene name and HGNC ID of “IFNGR2” + “c.173C>G” will provide us with the COSMIC ID - COSM3693766 as shown.


https://clinicaltables.nlm.nih.gov/api/cosmic/v3/search?terms=IFNGR2+c.173C%3EG

```
[1,["COSM3693766"],null,[["COSM3693766","IFNGR2","c.173C>G","p.T58R"]]]
```

The above example is for just one somatic mutation. In real, there could be numerous mutations present in the vcf file, so multiple queries will be sent through the COSMIC API.

2.3.4 Parse COSMIC mutation pages

Once the COSMIC IDs for all the somatic variants present in the VCF file has been generated, it is then constructed for COSMIC mutation page url.

Mutation base URL:

https://cancer.sanger.ac.uk/cosmic/mutation/overview?id=(COSMIC ID)

where, COSMIC ID is the ID generated when queried to COSMIC via API. For eg: 3693766

This mutation base URL is not the API provided by Sanger Institute unlike the COSMIC base URL in the previous step. Rather, this is simply the link that opens the particular mutation page on the COSMIC website. So, for a VCF file, multiple mutation URL is constructed based on the number of COSMIC IDs that were generated. Once the multiple mutation links have been created, the next step involves to scrap the content from the COSMIC page for the mutation.

2.3.5 Display top 10 critical mutations

The methodology used to filter out the top 10 most critical somatic mutations from the COSMIC data are based on three priority factors. These are:

1. **FATHMM prediction score:** a method for predicting pathogenic point mutations in the human genome. This method has been derived from the new FATHMM-MKL algorithm using hidden Markov models. The functional scores are in the form of a single p-value, ranging from 0 to 1. Scores above 0.5 are deleterious, but in order to highlight the most significant data in COSMIC, only scores ≥ 0.7 are classified as 'Pathogenic'. If the mutation score is ≤ 0.5, they are classed as ‘Neutral’.[10]
2. **The number of samples:** The number of samples tell us how many patients have been reported to have been affected by the same mutation. Hence, if there are a greater number of samples, that would mean the particular mutation is high risk mutation.
3. **The number of references:** The number of references determines all the literature where the particular mutation has been reported. Higher number of references for a particular mutation would mean that the mutation is a critical in causing cancer.

Considering the above factors, the mutations found in the vcf file are filtered according to the above priorities from within the COSMIC data. The top 10 mutations are then displayed in the final report.

2.4 UML Diagram

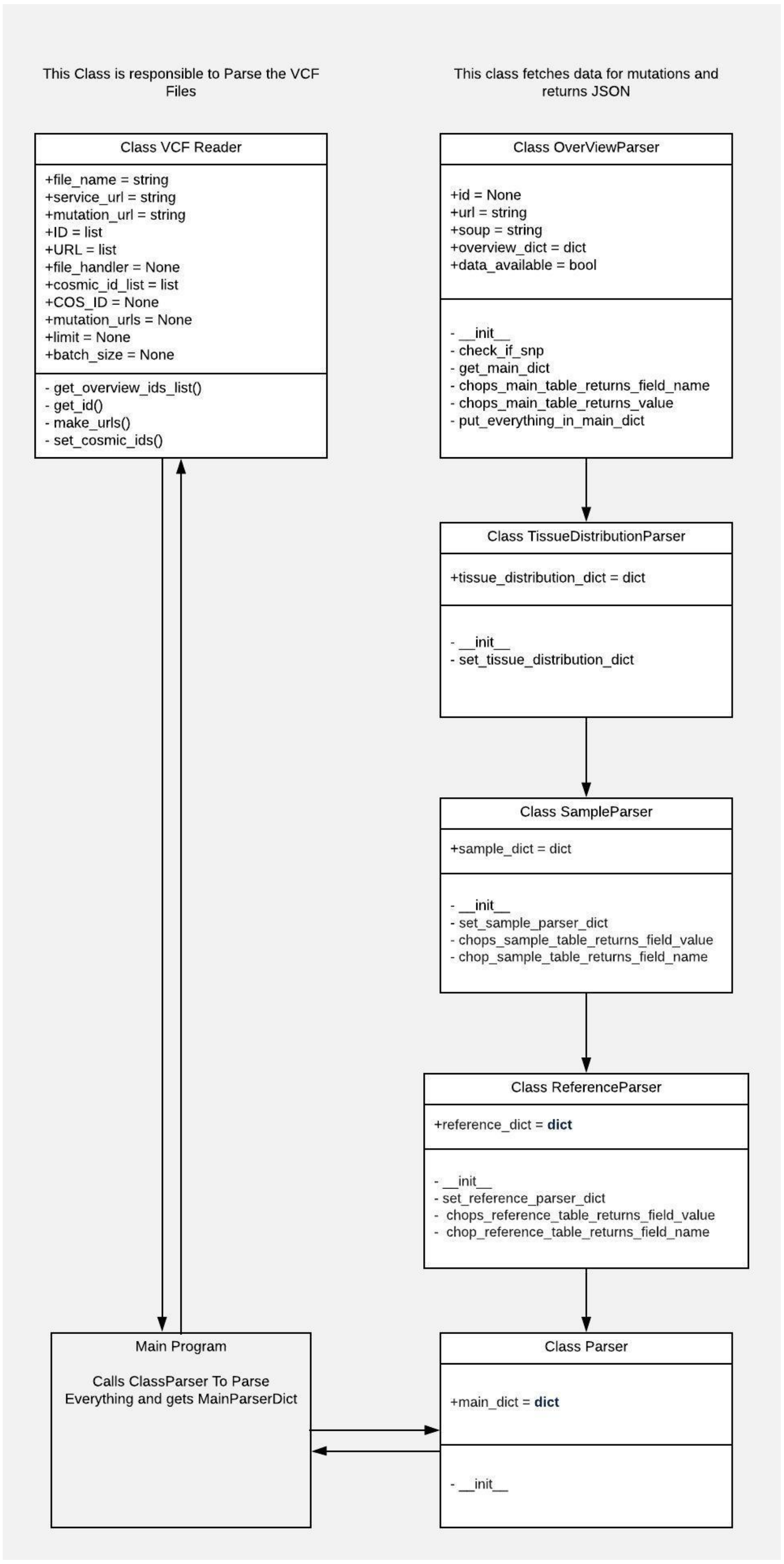
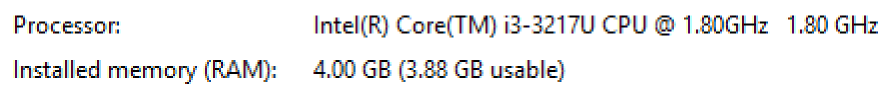


Figure 4 UML Diagram

3 Test Results and Discussion

In this chapter we shall perform a test on a sample input vcf file and generate the final output report with top 10 critical mutations present in it. Then discuss about some of the hurdles faced during this project and finally come up with some future improvement prospective.
The processor power used for the test is the following:



Processor:	Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz 1.80 GHz
Installed memory (RAM):	4.00 GB (3.88 GB usable)

3.1 VCF input specifications

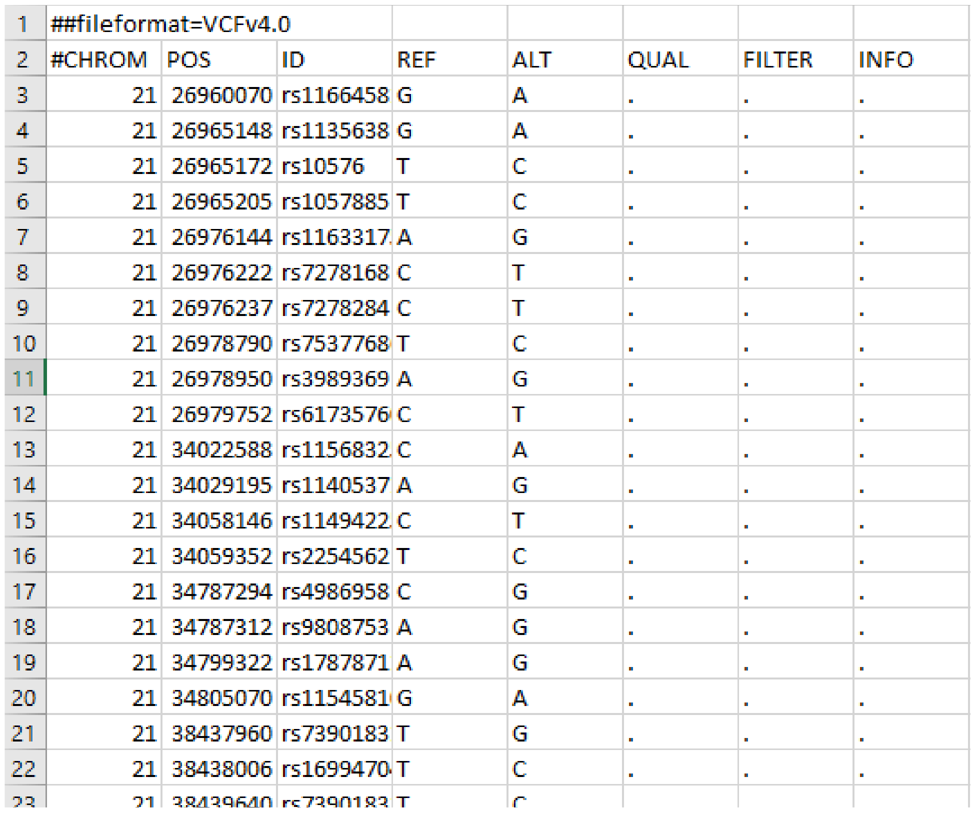
File version VCF4.0:
The sample input file should be specific with the VCF (Variant Call Format) version 4.0. This is a common format used by the 1000 genomes project, and can be produced as an output format by many variant calling tools.

Default format
chromosome - just the number, with no 'chr' prefix
The identifier must be in the 3rd column of VCF:
The VCF specification requires that the base comes immediately before the variant should be included in both the reference and variant alleles. In order to parse this correctly, VEP needs to convert such variants into Ensembl-type coordinates, and it does this by removing the additional base and adjusting the coordinates accordingly. This means that if an identifier is not supplied for a variant (in the 3rd column of the VCF), then the identifier constructed and the position reported in VEP's output file will differ from the input. [8]

3.2 Test input and output

3.2.1 Sample vcf input file

The input is a VCF file for a sample "homosapienGRCh37" of format VCF4.0. It consists of 173 variants in total. (*See Appendix B*)
Fig 5 shows the sample input file with Chromosome, Position, ID, reference and alternate allele as the first five columns and first 20 variants.



1	##fileformat=VCFv4.0							
2	#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
3	21	26960070	rs1166458	G	A	.	.	.
4	21	26965148	rs1135638	G	A	.	.	.
5	21	26965172	rs10576	T	C	.	.	.
6	21	26965205	rs1057885	T	C	.	.	.
7	21	26976144	rs1163317	A	G	.	.	.
8	21	26976222	rs7278168	C	T	.	.	.
9	21	26976237	rs7278284	C	T	.	.	.
10	21	26978790	rs7537768	T	C	.	.	.
11	21	26978950	rs3989369	A	G	.	.	.
12	21	26979752	rs6173576	C	T	.	.	.
13	21	34022588	rs1156832	C	A	.	.	.
14	21	34029195	rs1140537	A	G	.	.	.
15	21	34058146	rs1149422	C	T	.	.	.
16	21	34059352	rs2254562	T	C	.	.	.
17	21	34787294	rs4986958	C	G	.	.	.
18	21	34787312	rs9808753	A	G	.	.	.
19	21	34799322	rs1787871	A	G	.	.	.
20	21	34805070	rs1154581	G	A	.	.	.
21	21	38437960	rs7390183	T	G	.	.	.
22	21	38438006	rs1699470	T	C	.	.	.
23	21	38439640	rs7390183	T	C			

Figure 5 Input vcf file showing initial 20 variants

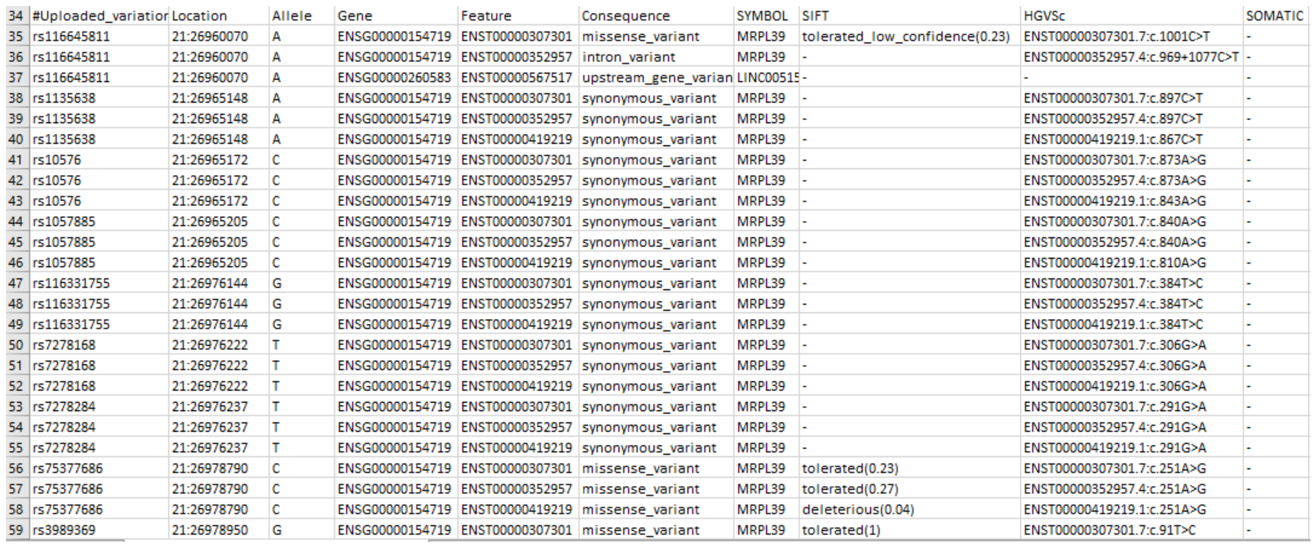
3.2.2 VEP annotation output

3.2.2.1 Annotated VCF file

Once the sample VCF file has been run through the PERL scripts provided by the VEP, we get an output file with the following output fields: *(See Appendix C)*

SOMATIC - Somatic status of existing variant(s); multiple values correspond to multiple values in the Existing_variation field
Location - in standard coordinate format (chr:start or chr:start-end)
Allele - the variant allele used to calculate the consequence
Gene - Ensembl stable ID of affected gene
Feature - Ensembl stable ID of feature
SYMBOL - the gene symbol
SIFT - the SIFT prediction and/or score, with both given as prediction(score). SIFT is based on a program that predicts whether an amino acid substitution affects protein function so that users can prioritize substitutions for further study.[11]
HGVSc - the HGVS coding sequence name

Following image shows the first 25 annotated variants from the input file.



34	#Uploaded_variatior	Location	Allele	Gene	Feature	Consequence	SYMBOL	SIFT	HGVSc	SOMATIC
35	rs116645811	21:26960070	A	ENSG00000154719	ENST00000307301	missense_variant	MRPL39	tolerated_low_confidence(0.23)	ENST00000307301.7:c.1001C>T	-
36	rs116645811	21:26960070	A	ENSG00000154719	ENST00000352957	intron_variant	MRPL39	-	ENST00000352957.4:c.969+1077C>T	-
37	rs116645811	21:26960070	A	ENSG00000260583	ENST00000567517	upstream_gene_varian	LINC00515	-	-	-
38	rs1135638	21:26965148	A	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.897C>T	-
39	rs1135638	21:26965148	A	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.897C>T	-
40	rs1135638	21:26965148	A	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.867C>T	-
41	rs10576	21:26965172	C	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.873A>G	-
42	rs10576	21:26965172	C	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.873A>G	-
43	rs10576	21:26965172	C	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.843A>G	-
44	rs1057885	21:26965205	C	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.840A>G	-
45	rs1057885	21:26965205	C	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.840A>G	-
46	rs1057885	21:26965205	C	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.810A>G	-
47	rs116331755	21:26976144	G	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.384T>C	-
48	rs116331755	21:26976144	G	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.384T>C	-
49	rs116331755	21:26976144	G	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.384T>C	-
50	rs7278168	21:26976222	T	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.306G>A	-
51	rs7278168	21:26976222	T	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.306G>A	-
52	rs7278168	21:26976222	T	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.306G>A	-
53	rs7278284	21:26976237	T	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.291G>A	-
54	rs7278284	21:26976237	T	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.291G>A	-
55	rs7278284	21:26976237	T	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.291G>A	-
56	rs75377686	21:26978790	C	ENSG00000154719	ENST00000307301	missense_variant	MRPL39	tolerated(0.23)	ENST00000307301.7:c.251A>G	-
57	rs75377686	21:26978790	C	ENSG00000154719	ENST00000352957	missense_variant	MRPL39	tolerated(0.27)	ENST00000352957.4:c.251A>G	-
58	rs75377686	21:26978790	C	ENSG00000154719	ENST00000419219	missense_variant	MRPL39	deleterious(0.04)	ENST00000419219.1:c.251A>G	-
59	rs3989369	21:26978950	G	ENSG00000154719	ENST00000307301	missense_variant	MRPL39	tolerated(1)	ENST00000307301.7:c.91T>C	-

Figure 6 Showing first 25 Annotated variants from the output vcf

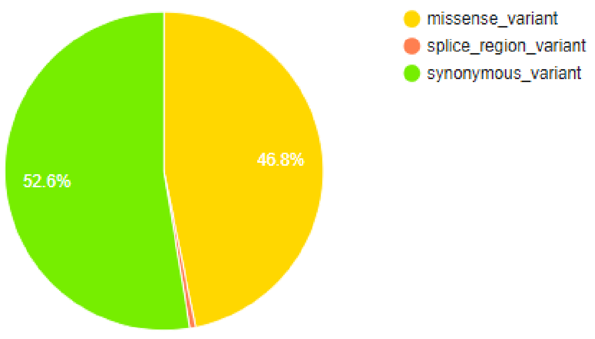
3.2.2.2 Annotation Statistics

The VEP also outputs a summary of annotation and basic statistics about the result. Following is the summary of annotated result.

VEP version (API)	93 (93)
Annotation sources	Cache: /home/kanzabatool/.vep/homo_sapiens/93_GRCh37; homo_sapiens_core_93_37 on ensembldb.ensembl.org
Species	homo_sapiens
Start time	2018-08-25 10:47:04
End time	2018-08-25 11:12:01
Run time	1497 seconds
Input file	examples/homo_sapiens_GRCh37.vcf
Output file	outputSiftSymtabLAGFC_HGVSc_SOMATIC.vcf
Lines of input read	173
Variants processed	173
Variants filtered out	0

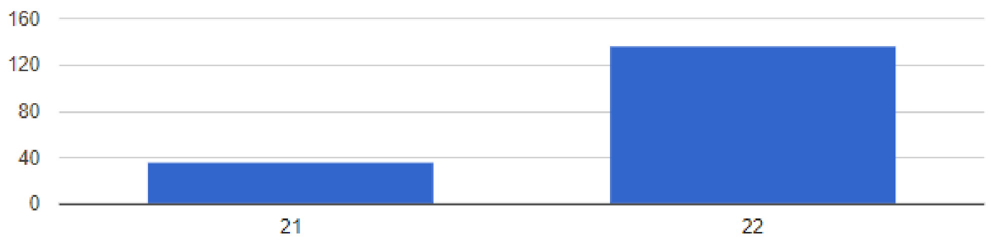
Novel / existing variants	0 (0.0) / 173 (100.0)
Overlapped genes	45
Overlapped transcripts	300
Overlapped regulatory features	-

Table 3 Annotation Statistics



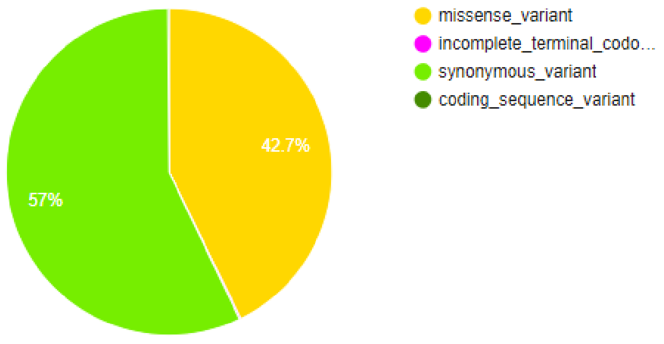
Consequence type	Count
missense_variant	81
splice_region_variant	1
synonymous_variant	91

Figure 7 Consequences (most severe)



Chromosome	Count
21	37
22	136

Figure 8 Variants by Chromosome



Consequence type	Count
missense_variant	306
incomplete_terminal_codon_variant	1
synonymous_variant	408
coding_sequence_variant	1

Figure 9 Coding consequences

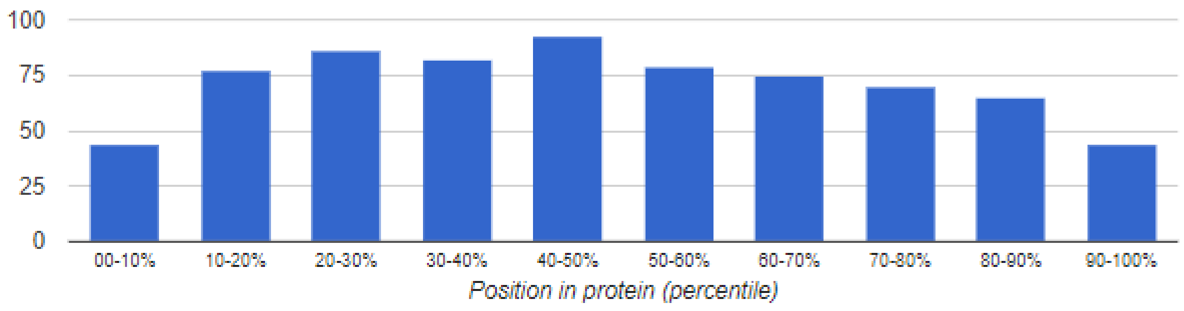
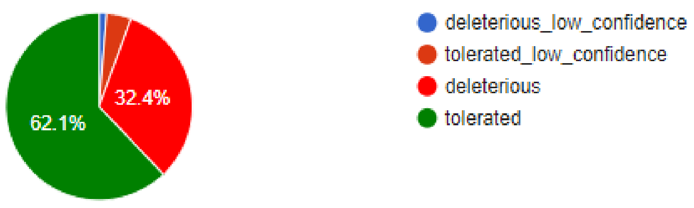


Figure 10 Position in Protien



Prediction	Count
deleterious_low_confidence	4
tolerated_low_confidence	13
deleterious	99

Figure 11 Sift Summary

3.2.3 Output report with top 10 critical mutations

The final output reports top 10 most critical mutations present within the input file. (For full report follow link in appendix D). Following is the summary of the top 10 most critical mutations found in the input file. These are:

1. Mutation ID COSM4670691 Gene name CDC42BPA_ENST00000366769 is critical mutation with its somatic nature confirmed in the literature. It has 2 samples reported from the database where tumor location was found in the large intestine. 2 references for the mutations is also reported. A FATHMM prediction score is high- 0.99 pathogenic.

2. Mutation ID COSM1988848 - Gene name R3HDM2_ENST00000347140 is critical mutation reported as pathogenic (0.99 FATHMM score). 1 sample found in the database with tumor in large intestine. 1 reference reported.

3. Mutation ID COSM6699587 Gene name LARGE_ENST00000452586 – is critical mutation suspected with tissue distribution found in the large intestine. 1 sample and 1 reference reported in the COSMIC database.

4. Mutation ID COSM5513679 Gene name TOP3B_ENST00000413067 – is critical mutation with tissue effected in the billary tract region in 1 sample report found in the database. 1 reference reported. Overall pathogenic.

5. Mutation ID COSM4188887 Gene name HIRA – tissue distribution found in the kidney in 1 sample reported. Is a confirmed somatic.

6. Mutation ID COSM6674668 Gene name GPS1_ENST00000306823 – has found cancerous traces in large intestine in a sample and is also a confirmed somatic mutation.

7. Mutation ID COSM1308066 Gene name SEC14L3 – has found traces of tumor in urinary track reported in one sample. Is pathogenic and confirmed somatic.

8. Mutation ID COSM4512219 Gene name SEC14L3 – is a pathogenic mutation associated to Skin and confirmed somatic according to the cancer database.

9. Mutation ID COSM6223153 Gene name DDX3Y_ENST00000537441 – has found tumor effects in the tissue of the large intestine and is confirmed to be a somatic mutation.

10. Mutation ID COSM6756630 Gene name PRPF4B_ENST00000337659 – tissue distribution found in the large intestine of a sample reported. Overall pathogenic and somatic nature is confirmed.

3.3 Challenges Overcome

No API for mutation data:
Unlike the COSMIC base API for COSMIC mutation ID, COSMIC does not provide an API for accessing mutation information from its database. Hence, in order to access mutation information for somatic mutations present in our input file, the data has been scrapped out from the COSMIC mutation pages using beautiful soup-a python library. Next hurdle that appeared alongside it was the fact that the COSMIC mutation pages consisted of data that were both in static and dynamic form. Beautiful soup can easily get the static data, but to access the dynamic data, it does not work the same way. Hence in order to deal with this issue, we made use of the Docker with chrome driver on port 4444 that helped us to get the dynamic AJAX content from the web page.

Some mutation pages in COSMIC flagged as SNP:
Whereas a number of COSMIC mutation page links were generated for mutations present in the input file, some of the pages in COSMIC provided no information and were labelled as 'flagged as SNP'. For eg: https://cancer.sanger.ac.uk/cosmic/mutation/overview?id=4418232 is the mutation page in COSMIC flagged as SNP, unlike this: https://cancer.sanger.ac.uk/cosmic/mutation/overview?id=6261085 which actually provides full information. Hence, to cater this issue, the python back-end program filtered out the links that were flagged and retained information from mutation pages that actually provided the details.

Program Speed
Program speed is a problem when the input file consists of a large number of variants. The total number of variants present in the input file means that as many queries will also be sent to the COSMIC database using the base API in the first phase. In the second phase, as many variants that have been matched from the database will then be scrapped out for their content from COSMIC mutation page. Hence, these processes consume a good amount of processing power especially when the number of variants in the input file is larger. In order to make the processes a bit more efficient in terms of its speed, the program was run on the AWS. No matter, there can be much more room for improvement here.

Following table shows the level of optimization achieved using AWS for 173 total variants in the input vcf file:

	Without optimisation	Using AWS
Sending COSMIC multiple queries (First phase)	70 seconds	1.5 seconds
Filtering through mutation pages (Second phase)	32 minutes	16 minutes

Table 4 Program Optimization

3.4 Future recommendation

There is quite a room for improvement in this project for those who wish to extend the work in future. Following are some recommendations:

Acceptance for diverse versions of VCF file as input:
Currently, the program accepts the input file with a limitation for only vcf version 4.0. along with other specifications as provided in the section 3.1. For a more pliable solution, the program can be improved to be able to process input file with different versions of VCF input file.

Parallel Processing:
The program involves couple of processes which can be run in parallel to achieve higher speed. For example, extraction of string parameters from input annotated file and sending query to COSMIC can be some of the processed that can be run in parallel to make the program faster.

Development of an interface:
For a better accessibility of the software, a user-friendly interface will be the necessity, so a user can directly upload their vcf files who even do not have any relevant programming experience and get an interactive result at the same time.

Getting License from COSMIC:
Since, the data in the mutation pages is being retrieved through web scrapping, a better approach would be to access license from COSMIC to be able to get a direct access of the data stored in the mutation pages for increased reliability of the program.

APPENDIX A Code

Please click here for full repository: GithubRepository

Requirements.py

```
beautifulsoup4==4.6.3
certifi==2018.8.24
chardet==3.0.4
chromedriver-installer==0.0.6
get==1.0.3
idna==2.7
lxml==4.2.4
numpy==1.15.1
panda==0.3.1
pandas==0.23.4
pdfkit==0.6.1
post==1.0.2
public==1.0.3
python-dateutil==2.7.3
pytz==2018.5
query-string==1.0.2
request==1.0.2
requests==2.19.1
selenium==3.14.0
six==1.11.0
urllib3==1.23
Jinja2==2.10
```

Vcf_reader.py

```
import re
from urllib.request import urlopen
import os
here = os.path.dirname(os.path.abspath(__file__))
class VCFReader:
    file_name = ""
    service_url = ""
    mutation_url = ""
    ID = []
    URL = []
    file_handler = None
    cosmic_id_list = []
    COS_ID = None
    mutation_urls = None
    limit = None

    def __init__(self, file=None, limit=900):
        if file:
            self.file_name = os.path.join(here, file)
            self.limit = limit
            self.service_url = 'https://clinicaltables.nlm.nih.gov/api/cosmic/v3/search?terms='
            self.mutation_url = 'https://cancer.sanger.ac.uk/cosmic/mutation/overview?id='
            self.file_handler = open(self.file_name)

            self.get_id()
            self.make_urls()
            self.set_cosmic_ids()
            self.get_overview_ids_list()

    def get_overview_ids_list(self):
        return self.COS_ID

    def get_id(self):
        chunks = []
        for line in self.file_handler:
            line = line.strip()
            data = re.findall('[a-z]\s([A-Z].*)\s[a-z-].*:([a-z].+>[A-Z])', line)
            if data:
                chunks.append(data)
        self.file_handler.close()
        for c in chunks:
            for iterator in c:
                self.ID.append(iterator)

    def make_urls(self):
        for i in range(len(self.ID) - self.limit):
            a = self.service_url + self.ID[i][0] + '+' + self.ID[i][1]
            self.URL.append(a)

            self.URL = [s.replace('>', '%3E') for s in self.URL]

    def set_cosmic_ids(self):
        for url in self.URL:
            content = urlopen(url)
            cosmic_id = content.read()
            if cosmic_id != b'[0,[],null,[]]':
                self.cosmic_id_list.append(cosmic_id.decode())
        # Regex all the COSMIC IDs
        y = re.findall('COSM([0-9]{7})', str(self.cosmic_id_list))

        # Retain only unique IDs
        self.COS_ID = list(set(y))
```

Gene.parser.py

```
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities

from time import sleep
from bs4 import BeautifulSoup
import requests

chrome = webdriver.Remote(
        command_executor='http://localhost:4444/wd/hub',
        desired_capabilities=DesiredCapabilities.CHROME)


class OverViewParser:
    id = None
    url = "https://cancer.sanger.ac.uk/cosmic/mutation/overview?id="
    soup = ""
    overview_dict = dict()
    data_available = False


    def __init__(self, id="6261085"):
        self.id = id
        self.url = self.url+self.id
        end_point = self.url
        chrome.get(end_point)
        sleep(2)
        main_soup = BeautifulSoup(chrome.page_source, 'lxml')
        self.soup = main_soup
        self.check_if_snp()
        self.get_main_dict()

    def check_if_snp(self):
        if self.soup:
            try:
                if "has been flagged as a SNP" in self.soup.find('p', class_='quote').text:
                    self.data_available = False
            except:
                self.data_available = True

    def get_main_dict(self):
        if self.data_available:
            field_name = self.chops_main_table_returns_field_name()
            field_value = self.chops_main_table_returns_value()

            # This is our main dict
            dict_main_fields = dict(zip(field_name, field_value))

            self.overview_dict = self.put_everything_in_main_dict(value=dict_main_fields)

    def chops_main_table_returns_field_name(self):
        if self.data_available:
            # chops the main table that occurs on the top of the page. Very Necessary
            first_entry_table = self.soup.find('dl', class_='inline')
            try:
                # chops up the individual fields like Mutation ID, Gene name, AA mutation etc etc.
                field_name = [link.string for link in first_entry_table.find_all('dt')]
            except Exception as ex:
                raise Exception("Internet Connection is Down: Therefore: {}".format(ex))

            return field_name

    def chops_main_table_returns_value(self):
        if self.data_available:
            # chops up the individual values like COSM6261085, etc etc
            field_value = []
            first_entry_table = self.soup.find('dl', class_='inline')
            f_value = first_entry_table.find_all('dd')

            for fv in f_value:
                tmp_dict = dict()
                if fv.find_all('a'):
                    for fv_hrefs in fv.find_all('a'):
                        try:
                            tester_tag = fv_hrefs.contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                            tester_link = fv_hrefs['href']
                            tmp_dict[tester_tag] = tester_link
                        except:
                            pass
                    field_value.append(tmp_dict)
                else:
                    try:
                        data = fv.find_all('p')[0].text.replace('\n', ' ').strip(' ')
                        field_value.append(data)
                    except:
                        data = fv.text.replace('\n', ' ').strip(' ')
                        field_value.append(data)

            return field_value

    def put_everything_in_main_dict(self, value=None):
        if self.data_available and value:
            main_dict = dict()
            main_dict[self.soup.find('h2').text.replace('COSM', '')] = value

            return main_dict


class TissueDistributionParser(OverViewParser):
    tissue_distribution_dict = dict()

    def __init__(self, id="6261085"):
        super().__init__(id=id)
        self.set_tissue_distribution_dict()

    def set_tissue_distribution_dict(self):
        if self.data_available:
            all_tissues_even_odd = self.soup.find_all('div', class_='section-content')[2].find_all('tr', {"class": ["even",
"odd"]})
            tmp_dict = dict()
            for all in all_tissues_even_odd:
                all_tissues = all.find_all('a')
                for tissues in all_tissues:
                    link = tissues['href']
                    if 'tissue' in link:
                        link_name = tissues.contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                        tmp_dict[link_name] = link
        self.tissue_distribution_dict = tmp_dict


class SampleParser(TissueDistributionParser):
    sample_dict = dict()

    def __init__(self, id="6261085"):
        super().__init__(id=id)
        self.set_sample_parser_dict()

    def set_sample_parser_dict(self):
        if self.data_available:
            field_values = self.chops_sample_table_returns_field_value()
            field_name = self.chop_sample_table_returns_field_name()

            dict_sample_fields = {k: field_values[i::len(field_name)] for i, k in enumerate(field_name)}

            self.sample_dict = dict_sample_fields

    def chops_sample_table_returns_field_value(self):
        if self.data_available:
            # chops up the individual values like CHG-13-09220T, etc etc
            field_value = []

            for f_even_odd in self.soup.find_all('div', class_='section-content')[2].find_all('tr', {"class": ["even", "odd"]}):
                f_value = f_even_odd.find_all('td')
                for fv in f_value:
                    tmp_dict = dict()
                    if fv.find_all('a'):
                        for fv_hrefs in fv.find_all('a'):
                            try:
                                tester_tag = fv_hrefs.contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                                tester_link = fv_hrefs['href']
                                tmp_dict[tester_tag] = tester_link
                            except:
                                tester_tag = fv_hrefs.contents[0].contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                                tester_link = fv_hrefs['href']
                                tmp_dict[tester_tag] = tester_link
                        field_value.append(tmp_dict)
                    else:
                        try:
                            data = fv.find_all('p')[0].text.replace('\n', ' ').strip(' ')
                            field_value.append(data)
                        except:
                            data = fv.text.replace('\n', ' ').strip(' ')
                            field_value.append(data)


            return field_value

    def chop_sample_table_returns_field_name(self):
        if self.data_available:
            # chops up the individual values like SAMPLE NAME, GENE NAME, etc etc
            field_name = []
            f_name = self.soup.find_all('div', class_='section-content')[2].find('tr').find_all('th')

            for fn in f_name:
                name = fn.contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                field_name.append(name)
        return field_name


class ReferenceParser(SampleParser):
    reference_dict = dict()

    def __init__(self, id="6261085"):
        super().__init__(id=id)
        self.set_reference_parser_dict()

    def set_reference_parser_dict(self):
        if self.data_available:
            field_values = self.chops_reference_table_returns_field_value()
            field_name = self.chop_reference_table_returns_field_name()

            dict_reference_fields = {k: field_values[i::len(field_name)] for i, k in enumerate(field_name)}

            self.reference_dict = dict_reference_fields

    def chops_reference_table_returns_field_value(self):
        if self.data_available:
            # chops up the individual values like CHG-13-09220T, etc etc
            field_value = []

            for f_even_odd in self.soup.find_all('div', class_='section-content')[2].find_all('tr', {"class": ["even", "odd"]}):
                f_value = f_even_odd.find_all('td')
                for fv in f_value:
                    tmp_dict = dict()
                    if fv.find_all('a'):
                        for fv_hrefs in fv.find_all('a'):
                            try:
                                tester_tag = fv_hrefs.contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                                tester_link = fv_hrefs['href']
                                tmp_dict[tester_tag] = tester_link
                            except:
                                tester_tag = fv_hrefs.contents[0].contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                                tester_link = fv_hrefs['href']
                                tmp_dict[tester_tag] = tester_link
                        field_value.append(tmp_dict)
                    else:
                        try:
                            data = fv.find_all('p')[0].text.replace('\n', ' ').strip(' ')
                            field_value.append(data)
                        except:
                            data = fv.text.replace('\n', ' ').strip(' ')
                            field_value.append(data)

            return field_value

    def chop_reference_table_returns_field_name(self):
        if self.data_available:
            # chops up the individual values like SAMPLE NAME, GENE NAME, etc etc
            field_name = []
            f_name = self.soup.find_all('div', class_='section-content')[3].find('tr').find_all('th')

            for fn in f_name:
                name = fn.contents[0].strip(' ').replace('\n', " ").replace(" ", "")
                field_name.append(name)

            return field_name
```

```
class Parser(ReferenceParser):
    main_dict = dict()

    def __init__(self, id="6261085"):
        super().__init__(id=id)
        self.main_dict['OverView'] = self.overview_dict
        self.main_dict['Tissue'] = self.tissue_distribution_dict
        self.main_dict['Sample'] = self.sample_dict
        self.main_dict['Reference'] = self.reference_dict


# p = Chopper(id=str(4745787))
# if p:
#     pass
```

Main.py

```
from gene_parser import Parser
from vcf_reader import VCFReader
import time
import json
from jinja2 import Environment, FileSystemLoader

import os
dirname, filename = os.path.split(os.path.abspath(__file__))




class Handler:
    sorted_list_reference = []

    def __init__(self, caching=False):
        # Total Time Starts
        total_start_time = time.time()

        # Call for loading VCF
        vcf_reader_start_time = time.time()
        vcf_reader_cos_id = self.load_vcf()
        vcf_reader_end_time = time.time()

        if vcf_reader_cos_id:
            pass

        vcf_reader_cos_id = self.omit_ids_already_checked(vcf_reader_cos_id)

        # Call for Parsing
        parsing_start_time = time.time()
        list_of_dicts = self.start_parsing(vcf_reader_cos_id, caching)
        parsing__stop_time = time.time()

        # Total Time Stops
        total_stop_time = time.time()

        if caching:
            print("--- CACHED RESULT = YES ---")
        print("--- VCF TIME %s seconds ---" % (vcf_reader_end_time - vcf_reader_start_time))
        print("--- PARSER TIME %s seconds ---" % (parsing__stop_time - parsing_start_time))
        print("--- TOTAL TIME %s seconds ---" % (total_stop_time - total_start_time))

        sorted_list_fathmm = self.sort_with_fathmm(list_of_dicts)
        sorted_list_reference = self.sort_with_reference(sorted_list_fathmm[:20])

        if sorted_list_reference:
            self.sorted_list_reference = sorted_list_reference

        self.save_sorted()

    def save_sorted(self):
        with open(dirname + '/cached/sorted_output.json', 'w') as outfile:
            json.dump(self.sorted_list_reference, outfile)

    @staticmethod
```

```
# Check if cached result is there? if Yes then use it else create it.
def load_vcf():
    try:
        with open(dirname+'/cached/vcf.json', 'r') as input_file:
            vcf_reader_cos_id = json.loads(input_file.read())

    except:
        vcf_reader = VCFReader(file='outputSiftSymtabLAGFC_HGVSc_SOMATIC.vcf', limit=0)
        vcf_reader_cos_id = vcf_reader.COS_ID

        with open(dirname+'/cached/vcf.json', 'w') as outfile:
            json.dump(vcf_reader_cos_id, outfile)

    return vcf_reader_cos_id

@staticmethod
def cached_results():
    with open(dirname + '/cached/parsed_output.json', 'r') as input_file:
        parsed_data = json.loads(input_file.read())

    return parsed_data

def start_parsing(self, vcf_reader_cos_id, caching=False):
    # Start parsing
    if caching:
        return self.cached_results()

    list_of_dicts = []
    for id in vcf_reader_cos_id:
        try:
            parsed_data = Parser(id=str(id))
            if parsed_data.data_available:
                self.save_delta_output(parsed_data.main_dict)
        except:
            print('error occured for {}'.format(id))
            pass

    return list_of_dicts

@staticmethod
def save_delta_output(parsed_data_dict):
    try:
        with open(dirname+'/cached/parsed_output.json', 'r') as input_file:
            parsed_data = json.loads(input_file.read())

        parsed_data.append(parsed_data_dict)

        with open(dirname+'/cached/parsed_output.json', 'w') as outfile:
            json.dump(parsed_data, outfile)
    except:
        tmp_list = []
        tmp_list.append(parsed_data_dict)
        with open(dirname+'/cached/parsed_output.json', 'w') as outfile:
            json.dump(tmp_list, outfile)

@staticmethod
def save_complete(list_of_dicts):
    try:
        with open(dirname+'/cached/parsed_output.json', 'w') as outfile:
            json.dump(list_of_dicts, outfile)
        except:
            pass

    @staticmethod
    def omit_ids_already_checked(vcf_reader_cos_id):
        try:
            with open(dirname+'/cached/parsed_output.json', 'r') as input_file:
                parsed_output = json.loads(input_file.read())

            parsed_output_ids = [list(i['OverView'].keys()) for i in parsed_output]
            parsed_output_ids = [ item for sub in parsed_output_ids for item in sub ]

            vcf_reader_cos_id = list(set(vcf_reader_cos_id) - set(parsed_output_ids))
            return vcf_reader_cos_id
        except:
            return vcf_reader_cos_id

    @staticmethod
    def sort_with_fathmm(list_of_dicts):
        # float(list_of_dicts[0]['OverView']['5484386']['FATHMM prediction'].split('score')[1][:-1].strip(' '))
        tmp_list = []
        for dicts in list_of_dicts:
            for key, value in dicts['OverView'].items():
                try:
                    dicts['prediction_score'] = float(value['FATHMM prediction'].split('score')[1][:-1].strip(' '))
                except:
                    dicts['prediction_score'] = float(0)

                tmp_list.append(dicts)

        new_list = sorted(tmp_list, key=lambda k:k['prediction_score'], reverse=True)

        if new_list:
            return new_list

    @staticmethod
    def sort_with_reference(list_of_dicts):
        # float(list_of_dicts[0]['OverView']['5484386']['FATHMM prediction'].split('score')[1][:-1].strip(' '))
        tmp_list = []
        for dicts in list_of_dicts:
            try:
                dicts['reference_score'] = len(dicts['Reference']['ReferenceTitle'])
            except:
                dicts['reference_score'] = float(0)

            tmp_list.append(dicts)

        new_list = sorted(tmp_list, key=lambda k: k['reference_score'], reverse=True)

        if new_list:
            return new_list


def main():
    from selenium import webdriver
    from selenium.webdriver.common.desired_capabilities import DesiredCapabilities

    from time import sleep
    from bs4 import BeautifulSoup
    import requests
    chrome = webdriver.Remote(
        command_executor='http://localhost:4444/wd/hub',
        desired_capabilities=DesiredCapabilities.CHROME)

    handler = Handler(caching=True)
    sorted_list = handler.sorted_list_reference
    sorted_ids = [list(x['OverView'].keys())[0] for x in sorted_list]
    sorted_ids = sorted_ids[:10]

    i = 0
    soups = []
    for id in sorted_ids:
        end_point = "https://cancer.sanger.ac.uk/cosmic/mutation/overview?id="+id
        chrome.get(end_point)
        sleep(2)
        main_soup = BeautifulSoup(chrome.page_source, 'lxml')

        #Decompose
        if i>0:
            for script in main_soup.find_all('script'):
                script.decompose()

        try:
            main_soup.find('section', {'id': 'ccc'}).decompose()
        except:
            pass

        main_soup.find('h1', {'class': 'subhead'}).decompose()
        main_soup.find('footer').decompose()
        main_soup.find('header').decompose()
        main_soup.find('ul', {'class': 'slimmenu'}).decompose()
        for div in main_soup.find_all('div', {'class': 'dataTables_length'}):
            div.decompose()

        for div in main_soup.find_all('div', {'class': 'dataTables_filter'}):
            div.decompose()

        for img in main_soup.find_all('img'):
            img.decompose()

        soups.append(main_soup)
        i = i + 1

    file_loader = FileSystemLoader(dirname + '/output/templates')
    env = Environment(loader=file_loader)

    template = env.get_template('index.html')
    output = template.render(mutations=soups)

    if output:
        pass

if __name__ == "__main__":
    main()
```

APPENDIX B Input VCF file

For full Input sample VCF file, please click here: homo_sapiens_GRCh37.vcf

```
##fileformat=VCFv4.0
#CHROM   POS        ID          REF        ALT        QUAL       FILTER      INFO
         21  26960070  rs1166458  G          A          .          .          .
         21  26965148  rs1135638  G          A          .          .          .
         21  26965172  rs10576    T          C          .          .          .
         21  26965205  rs1057885  T          C          .          .          .
         21  26976144  rs1163317  A          G          .          .          .
         21  26976222  rs7278168  C          T          .          .          .
         21  26976237  rs7278284  C          T          .          .          .
         21  26978790  rs7537768  T          C          .          .          .
         21  26978950  rs3989369  A          G          .          .          .
         21  26979752  rs6173576  C          T          .          .          .
         21  34022588  rs1156832  C          A          .          .          .
         21  34029195  rs1140537  A          G          .          .          .
         21  34058146  rs1149422  C          T          .          .          .
         21  34059352  rs2254562  T          C          .          .          .
         21  34787294  rs4986958  C          G          .          .          .
         21  34787312  rs9808753  A          G          .          .          .
         21  34799322  rs1787871  A          G          .          .          .
         21  34805070  rs1154581  G          A          .          .          .
         21  38437960  rs7390183  T          G          .          .          .
         21  38438006  rs1699470  T          C          .          .          .
         21  38439640  rs7390183  T          C          .          .          .
         21  38444863  rs2507733  C          T          .          .          .
         21  38444881  rs7320024  G          A          .          .          .
         21  40186202  rs3437335  G          A          .          .          .
         21  40190405  rs1159082  G          A          .          .          .
         21  40190504  rs1166989  G          A          .          .          .
         21  40191431  rs457705   T          G          .          .          .
         21  40191548  rs1134178  C          T          .          .          .
         21  40191638  rs461155   A          G          .          .          .
         21  40193613  rs1164760  C          T          .          .          .
         21  44514601  rs6173706  C          T          .          .          .
         21  44514809  rs6173706  G          A          .          .          .
         21  45387915  rs1153977  C          T          .          .          .
         21  45389031  rs7930925  C          G          .          .          .
         21  45389040  rs7553287  C          T          .          .          .
         21  45389127  rs1158750  G          A          .          .          .
         21  45402270  rs1145732  A          G          .          .          .
         22  17662793  rs7289170  A          G          .          .          .
         22  17669306  rs2231495  T          C          .          .          .
         22  19340928  rs3400036  G          A          .          .          .
         22  19365526  rs1158778  G          C          .          .          .
         22  19371166  rs6173592  G          A          .          .          .
         22  19373100  rs1151579  A          T          .          .          .
         22  19384355  rs9618556  C          T          .          .          .
         22  19384439  rs1164583  G          A          .          .          .
         22  19958829  rs5993890  G          A          .          .          .
```

APPENDIX C Output Annotated VCF

For full output annotated VCF, please click here: outputSiftSymtabLAGFC_HGVSc_SOMATIC.vcf

```
## ENSEMBL VARIANT EFFECT PREDICTOR v93.3
## Output produced at 2018-08-25 10:47:08
## Connected to homo_sapiens_core_93_37 on ensembldb.ensembl.org
## Using cache in /home/kanzabatool/.vep/homo_sapiens/93_GRCh37
## Using API version 93, DB version 93
## ensembl version 93.b4d45ee
## ensembl-io version 93.cc29a66
## ensembl-funcgen version 93.0c98373
## ensembl-variation version 93.c9c43a7
## HGMD-PUBLIC version 20164
## regbuild version 1.0
## gencode version GENCODE 19
## gnomAD version 170228
## 1000genomes version phase3
## COSMIC version 81
## sift version sift5.2.2
## polyphen version 2.2.2
## ClinVar version 201706
## genebuild version 2011-04
## assembly version GRCh37.p13
## ESP version 20141103
## dbSNP version 150
## Column descriptions:
## Uploaded_variation : Identifier of uploaded variant
## Location : Location of variant in standard coordinate format (chr:start or chr:start-end)
## Allele : The variant allele used to calculate the consequence
## Gene : Stable ID of affected gene
## Feature : Stable ID of feature
## Consequence : Consequence type
## SYMBOL : Gene symbol (e.g. HGNC)
## SIFT : SIFT prediction and/or score
## HGVSc : HGVS coding sequence name
## SOMATIC : Somatic status of existing variant
#Uploaded_variation	Location	Allele	Gene	Feature	Consequence	SYMBOL	SIFT	HGVSc	SOMATIC
rs116645811	21:26960070	A	ENSG00000154719	ENST00000307301	missense_variant	MRPL39	tolerated_low_confidence(0.23)	ENST00000307301.7:c.1001C>T	-
rs116645811	21:26960070	A	ENSG00000154719	ENST00000352957	intron_variant	MRPL39	-	ENST00000352957.4:c.969+1077C>T	-
rs116645811	21:26960070	A	ENSG00000260583	ENST00000567517	upstream_gene_variant	LINC00515	-	-	-

rs1135638	21:26965148	A	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.897C>T	-
rs1135638	21:26965148	A	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.897C>T	-
rs1135638	21:26965148	A	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.867C>T	-
rs10576	21:26965172	C	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.873A>G	-
rs10576	21:26965172	C	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.873A>G	- rs10576
21:26965172	C	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.843A>G	- rs1057885
21:26965205	C	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.840A>G	- rs1057885
21:26965205	C	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.840A>G	- rs1057885
21:26965205	C	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.810A>G	- rs116331755
21:26976144	G	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.384T>C	- rs116331755
21:26976144	G	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.384T>C	- rs116331755
21:26976144	G	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.384T>C	- rs7278168
21:26976222	T	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.306G>A	- rs7278168
21:26976222	T	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.306G>A	- rs7278168
21:26976222	T	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.306G>A	- rs7278284
21:26976237	T	ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.291G>A	- rs7278284
21:26976237	T	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.291G>A	- rs7278284
21:26976237	T	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.291G>A	-
rs75377686	21:26978790	C	ENSG00000154719	ENST00000307301	missense_variant	MRPL39	tolerated(0.23)	ENST00000307301.7:c.251A>G	- rs75377686
21:26978790	C	ENSG00000154719	ENST00000352957	missense_variant	MRPL39	tolerated(0.27)	ENST00000352957.4:c.251A>G	- rs75377686	21:26978790
C	ENSG00000154719	ENST00000419219	missense_variant	MRPL39	deleterious(0.04)	ENST00000419219.1:c.251A>G	- rs3989369	21:26978950	G
ENSG00000154719	ENST00000307301	missense_variant	MRPL39	tolerated(1)	ENST00000307301.7:c.91T>C	- rs3989369	21:26978950	G
ENSG00000154719	ENST00000352957	missense_variant	MRPL39	tolerated(1)	ENST00000352957.4:c.91T>C	- rs3989369	21:26978950	G
ENSG00000154719	ENST00000419219	missense_variant	MRPL39	tolerated(0.99)	ENST00000419219.1:c.91T>C	- rs61735760	21:26979752	T
ENSG00000154719	ENST00000307301	synonymous_variant	MRPL39	-	ENST00000307301.7:c.36G>A	-
rs61735760	21:26979752	T	ENSG00000154719	ENST00000352957	synonymous_variant	MRPL39	-	ENST00000352957.4:c.36G>A
- rs61735760	21:26979752	T	ENSG00000154719	ENST00000419219	synonymous_variant	MRPL39	-	ENST00000419219.1:c.36G>A
-
rs115683257	21:34022588	A	ENSG00000159082	ENST00000322229	missense_variant	SYNJ1	tolerated(0.83)	ENST00000322229.7:c.2943G>T	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000357345	missense_variant	SYNJ1	tolerated(1)	ENST00000357345.3:c.2943G>T	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000382491	missense_variant	SYNJ1	tolerated(0.85)	ENST00000382491.3:c.2928G>T	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000382499	missense_variant	SYNJ1	tolerated(1)	ENST00000382499.2:c.3060G>T	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000416083	upstream_gene_variant	SYNJ1	-	-	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000433931	missense_variant	SYNJ1	tolerated(0.73)	ENST00000433931.2:c.3060G>T	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000438952	upstream_gene_variant	SYNJ1	-	-	0,1,1
rs115683257	21:34022588	A	ENSG00000159082	ENST00000467445	non_coding_transcript_exon_variant	SYNJ1	-	ENST00000467445.1:n.883G>T	0,1,1
rs114053718	21:34029195	G	ENSG00000159082	ENST00000322229	missense_variant	SYNJ1	deleterious(0)	ENST00000322229.7:c.2597T>C	-
rs114053718	21:34029195	G	ENSG00000159082	ENST00000357345	missense_variant	SYNJ1	deleterious(0)	ENST00000357345.3:c.2597T>C	- rs114053718
21:34029195	G	ENSG00000159082	ENST00000382491	missense_variant	SYNJ1	deleterious(0)	ENST00000382491.3:c.2582T>C	- rs114053718	21:34029195
G	ENSG00000159082	ENST00000382499	missense_variant	SYNJ1	deleterious(0)	ENST00000382499.2:c.2714T>C	- rs114053718	21:34029195	G
ENSG00000159082	ENST00000433931	missense_variant	SYNJ1	deleterious(0)	ENST00000433931.2:c.2714T>C	- rs114053718	21:34029195	G
ENSG00000159082	ENST00000464778	non_coding_transcript_exon_variant	SYNJ1	-	ENST00000464778.1:n.384T>C	- rs114053718	21:34029195	G
ENSG00000159082	ENST00000467445	upstream_gene_variant	SYNJ1	-	-	-
rs114942253	21:34058146	T	ENSG00000159082	ENST00000322229	missense_variant	SYNJ1	tolerated(0.67)	ENST00000322229.7:c.1030G>A	- rs114942253
```

APPENDIX D Final Output Report

With top 10 critical somatic mutations found in the sample input vcf file.

For FULL REPORT please click here : output.pdf

Mutation

COSM4670691

- × Overview
- ×Tissue distribution
- × Samples
- ×Pathways affected
- ×References

Reset page

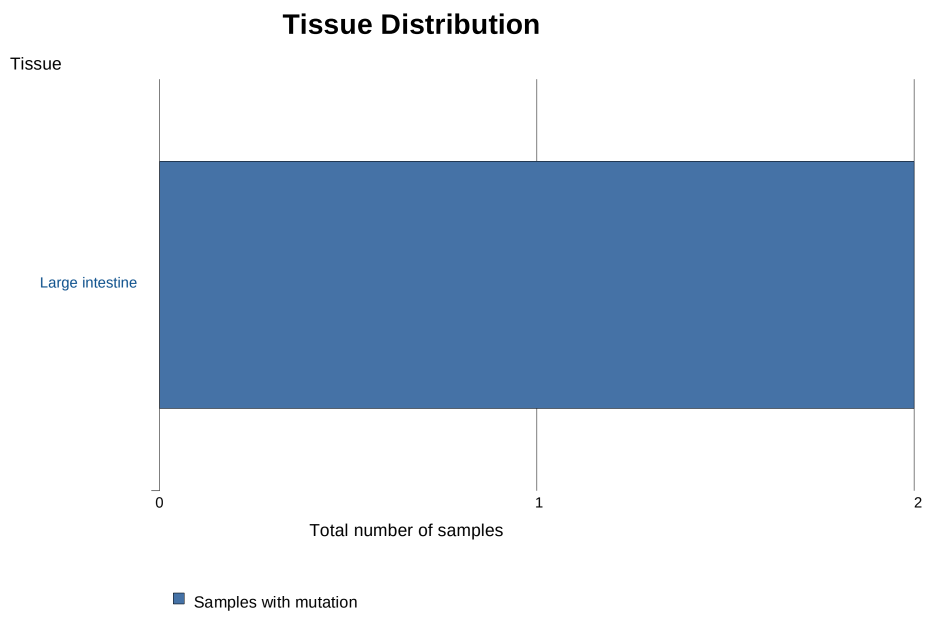
Overview

This section shows a general overview of the selected mutation. It describes the source of the mutation i.e gene name/sample name/tissue name with unique ID, and also shows the mutation syntax at the amino acid and nucleotide sequence level. You can see more information on our help pages.

Mutation ID	COSM4670691
Gene name	CDC42BPA_ENST00000366769
AA mutation	p.R598H (Substitution - Missense, position 598, R➞H)
CDS mutation	c.1793G>A (Substitution, position 1793, G➞A)
Nucleotides inserted	n/a
Genomic coordinates	GRCh38, 1:227112768..227112768, view Ensembl contig
CDD	n/a
HomoloGene	n/a
Ever confirmed somatic?	Yes
FATHMM prediction	Pathogenic (score 0.99)
Remark	n/a
Recurrent	n/a
Drug resistance	n/a

Tissue distribution

This section displays the distribution of mutated samples and tissue types (top 5). You can see more information on our help pages.



Samples

This section displays a table of mutated samples, with tissue, histology and zygosity information. Publication information is also included, where available, with links to PUBMED.

Sample name	Gene name	Transcript	Primary Tissue	Tissue Subtype 1	Primary Histology	Histology Subtype 1	Pubmed ID	Zygosity	Somatic Status	Sample Type	LOH	Resistant Mutation	Drugs
T464	CDC42BPA_ENST00000366769	ENST00000366769	Large intestine	NS	Carcinoma	Adenocarcinoma	27149842	Unknown	Confirmed Somatic		Unknown	-	
T464	CDC42BPA_ENST00000366769	ENST00000366769	Large intestine	Colon	Carcinoma	Adenocarcinoma	25344691	Unknown	Confirmed Somatic		Unknown	-	

References

This section displays a table of references for the mutation. You can see more information on the help pages.

Reference Title	Author	Year	Journal	Status	COSMIC	Pubmed
RNF43 is frequently mutated in colorectal and endometrial cancers	Giannakis M et al	2014	Nature genetics;46(12):1264-6	Curated	COSP37678	25344691
Genomic Correlates of Immune-Cell Infiltrates in Colorectal Carcinoma	Giannakis M et al	2016	Cell reports	Curated	COSP44021	27149842

****

Mutation

COSM1988848

- × Overview
- ×Tissue distribution
- × Samples
- ×Pathways affected
- ×References

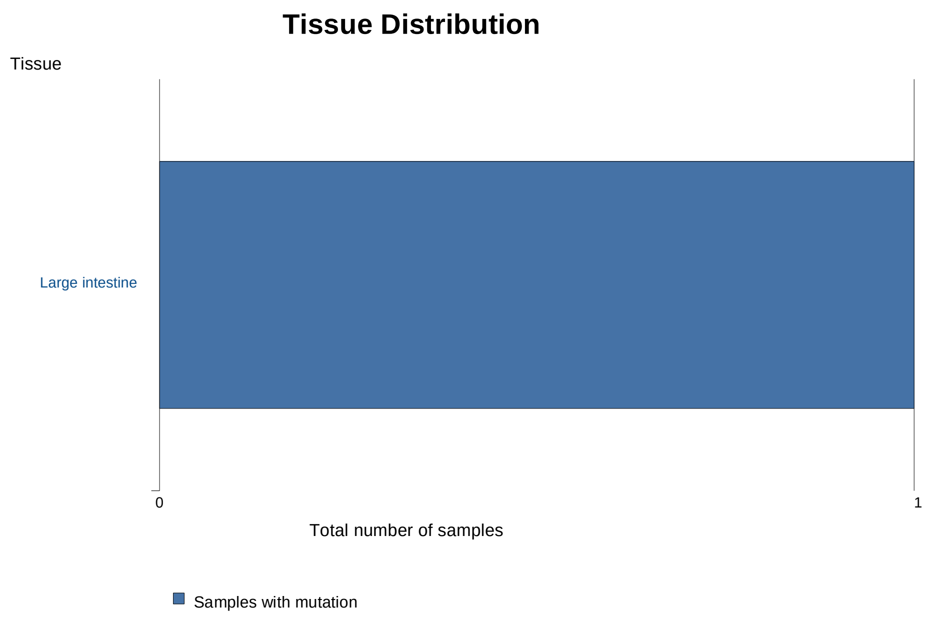
Overview

This section shows a general overview of the selected mutation. It describes the source of the mutation i.e gene name/sample name/tissue name with unique ID, and also shows the mutation syntax at the amino acid and nucleotide sequence level. You can see more information on our help pages.

Mutation ID	COSM1988848
Gene name	R3HDM2_ENST00000347140
AA mutation	p.R449H (Substitution - Missense, position 449, R➞H)
CDS mutation	c.1346G>A (Substitution, position 1346, G➞A)
Nucleotides inserted	n/a
Genomic coordinates	GRCh38, 12:57269951..57269951, view Ensembl contig
CDD	n/a
HomoloGene	n/a
Ever confirmed somatic?	No
FATHMM prediction	Pathogenic (score 0.99)
Remark	n/a
Recurrent	n/a
Drug resistance	n/a

Tissue distribution

This section displays the distribution of mutated samples and tissue types (top 5). You can see more information on our help pages.



Samples

This section displays a table of mutated samples, with tissue, histology and zygosity information. Publication information is also included, where available, with links to PUBMED.

Sample name	Gene name	Transcript	Primary Tissue	Tissue Subtype 1	Primary Histology	Histology Subtype 1	Pubmed ID	Zygosity	Somatic Status	Sample Type	LOH	Resistant Mutation	Drugs
RKO	R3HDM2_ENST00000347140	ENST00000347140	Large intestine	NS	Carcinoma	Adenocarcinoma	24755471	Heterozygous	Unknown	Cultured	Unknown	-	

Showing 1 to 1 of 1 entries

FirstPrevious1NextLast

References

This section displays a table of references for the mutation. You can see more information on the help pages.

Reference Title	Author	Year	Journal	Status	COSMIC	Pubmed
Colorectal cancer cell lines are representative models of the main molecular subtypes of primary cancer	Mouradov D et al	2014	Cancer research;74(12):3238-47	Curated	COSP35484	24755471

Bibliography

[1] J. Ferlay *et al.*, "Cancer incidence and mortality worldwide: sources, methods and major patterns in GLOBOCAN 2012.," *Int. J. cancer*, vol. 136, no. 5, pp. E359-86, Mar. 2015.

[2] K. Tomczak, P. Czerwińska, and M. Wiznerowicz, "The Cancer Genome Atlas (TCGA): An immeasurable source of knowledge," *Wspolczesna Onkol.*, vol. 1A, pp. A68–A77, 2015.

[3] K. Y. Yip, C. Cheng, and M. Gerstein, "Machine learning and genome annotation: A match meant to be?," *Genome Biol.*, vol. 14, no. 5, 2013.

[4] K. Wang, M. Li, and H. Hakonarson, "ANNOVAR: Functional annotation of genetic variants from high-throughput sequencing data," *Nucleic Acids Res.*, vol. 38, no. 16, pp. 1–7, 2010.

[5] D. L. Masica *et al.*, "CRAVAT 4: Cancer-related analysis of variants toolkit," *Cancer Res.*, vol. 77, no. 21, pp. e35–e38, 2017.

[6] A. H. Ramos *et al.*, "Oncotator: Cancer variant annotation tool," *Hum. Mutat.*, vol. 36, no. 4, pp. E2423–E2429, 2015.

[7] S. A. Forbes *et al.*, "COSMIC: Mining complete cancer genomes in the catalogue of somatic mutations in cancer," *Nucleic Acids Res.*, vol. 39, no. SUPPL. 1, pp. 945–950, 2011.

[8] W. McLaren *et al.*, "The Ensembl Variant Effect Predictor," *Genome Biology*, 06-Dec-2016. [Online]. Available: http://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-0974-4. [Accessed: 11-Jan-2019].

[9] "Clinical Table Search Service." [Online]. Available: https://clinicaltables.nlm.nih.gov/apidoc/cosmic/v3/doc.html. [Accessed: 11-Jan-2019].

[10] H. A. Shihab *et al.*, "An integrative approach to predicting the functional effects of non-coding and coding sequence variation," *Bioinformatics*, vol. 31, no. 10, pp. 1536–1543, 2015.

[11] P. C. Ng and S. Henikoff, "SIFT: Predicting amino acid changes that affect protein function," *Nucleic Acids Res.*, vol. 31, no. 13, pp. 3812–3814, 2003.