Exploring Genetic Disparities and Predictive Modeling in Gastric Cancer: A Statistical and Machine Learning Approach Using TCGA Data

Researcher: Joey Park, Junior at Thomas Jefferson High School for Science and Technology

6560 Braddock Rd, Alexandria, VA 22312

Content

Research 1. A Statistical Analysis of Asian and Non-Hispanic White Populations on Gastric Cancer

Research 2. Machine Learning-Based Cancer Type Prediction Using TCGA Data

1. Research Motivation and Data Source Back to Top

When my grandfather was diagnosed with stomach cancer, I felt helpless. I wanted to do something but didn't know where to start. As I began researching, I discovered a striking disparity: gastric cancer disproportionately affects Asians, especially Koreans, while being significantly less common in other racial groups, particularly non-Hispanic whites. (https://pmc.ncbi.nlm.nih.gov/articles/PMC7680373/)

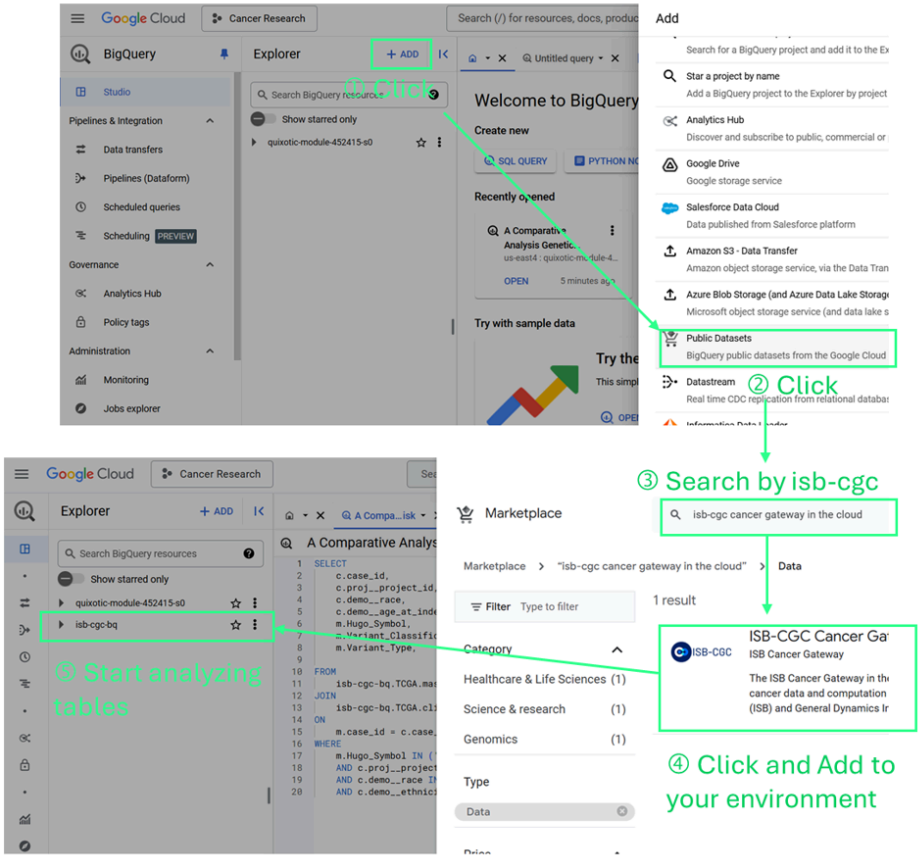
Since I loved my DNA Science class at school and had a strong interest in bioinformatics, I decided to conduct my own research on this topic. Figuring out what kind of data I needed and where to obtain it was challenging, but after extensive searching, I found that The ISB Cancer Gateway in the Cloud (ISB-CGC) provided the datasets I was looking for.

Using Google Cloud Platform's BigQuery, I explored various tables from different databases and identified two key dataset relevant to my research. Finally, I used Google Colab with the BigQuery API to extract the data, analyze it, and build a model for my study.

How to locate the ISB-CGC database and begin querying data on Google Cloud Platform

https://www.isb-cgc.org/

It is more effective to use Google Cloud Platform for reviewing and analyzing data and locating the necessary tables rather than relying on the ISB-CGC homepage for this task.



2. Objective Back to Top

Stomach cancer affects Asian populations, especially Koreans, much more than non-Hispanic whites (NHWs). Research suggests that certain genes, such as LCE1, PRKAA1, PSCA, and TP53, etc. may play a role in this difference. However, scientists haven't fully figured out how much these genetic differences actually impact cancer risk. This study aims to explore this question by comparing genetic variations in stomach cancer patients from Asian and NHW populations and using statistical models to see how important these differences are in predicting cancer risk.

Here are the types of gene mutations I will analyze and the sources (click links) from which I obtained information about them.

- CDH1, CTNNA1, APC, MLH1, MSH2, MSH6, PMS2, EPCAM, STK11, SMAD4, BMPR1A, TP53 Link

- PRKAA1 Link
- PSCA Link

3. TCGA Data Back to Top

After thorough research and analysis of the tables, I was able to determine that TCGA data is suitable for my research because it provides comprehensive genomic and clinical information, allowing for in-depth analysis of gastric cancer patterns and disparities. I also discovered two TCGA tables that are perfectly suited for my project.

1) isb-cgc-bq.TCGA.clinical_gdc_current

This table has the following information useful for my project

- Patient/sample ID (case_id)
- Patient demographics (age, gender, ethnicity, race)
- Diagnosis details (cancer type, stage, tumor location)

2) isb-cgc-bq.TCGA.masked_somatic_mutation_hg38_gdc_current

This table has the following information:

- Patient/sample ID (case_id)
- Gene information (affected gene names, Hugo_Symbol)
- Mutation type (missense, nonsense, frameshift, silent, etc.)

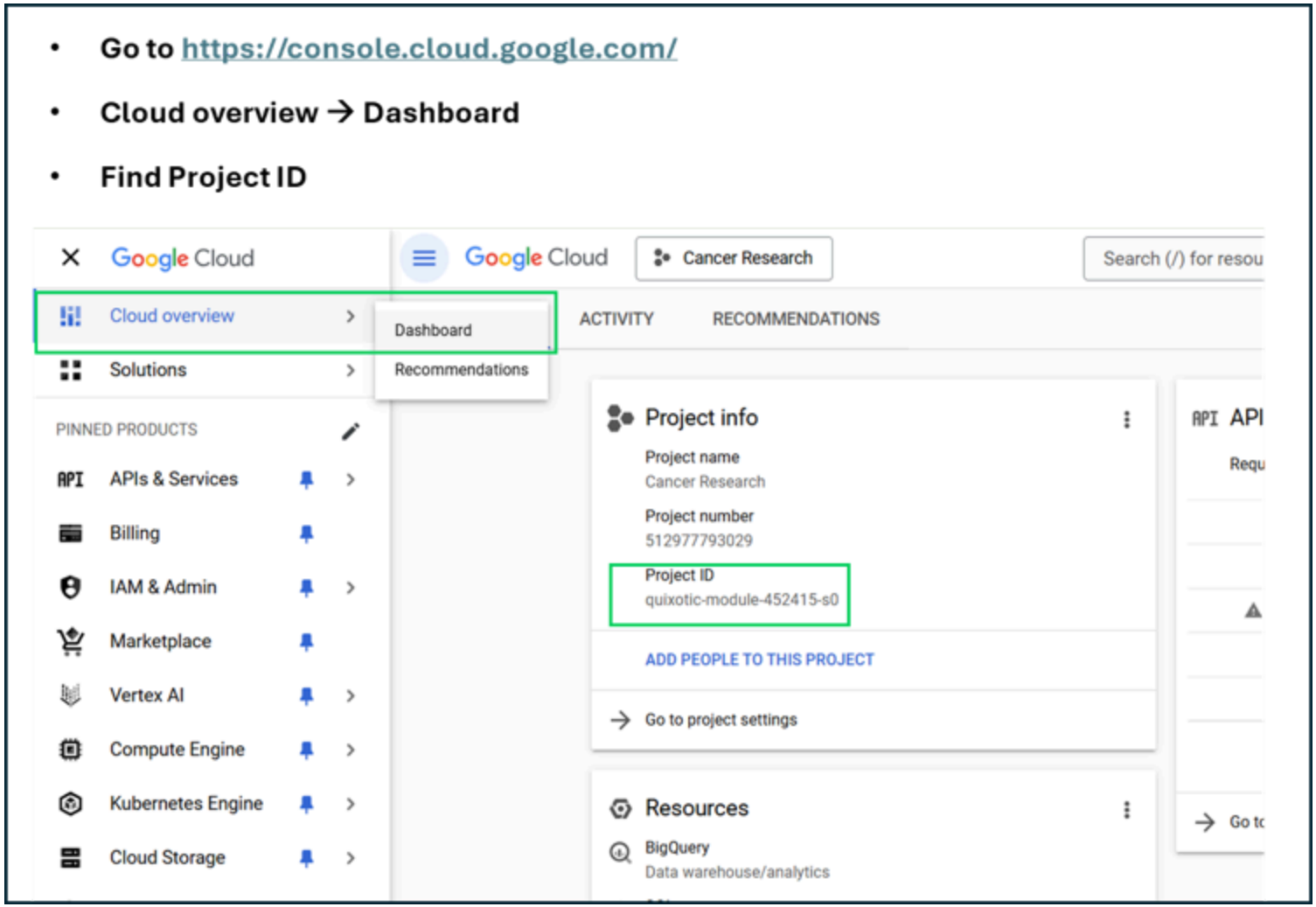
4.Python Libraries and Authentication to Google Cloud Back to Top

I utilized the following libraries for my analysis and model development.

In [ ]:
```
import pandas as pd  # Data manipulation and analysis
import numpy as np  # Numerical computing and array operations
from google.colab import auth  # Authentication for Google Colab
from google.cloud import bigquery  # Querying and retrieving data from Google BigQuery
from scipy.stats import chi2_contingency  # Chi-squared test for statistical analysis
import statsmodels.api as sm  # Statistical modeling and hypothesis testing
from sklearn.model_selection import train_test_split  # Splitting data into training and testing sets
```

```
from sklearn.linear_model import LogisticRegression  # Logistic regression modeling
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve  # Model evaluation metrics
import matplotlib.pyplot as plt  # Data visualization and plotting
import xgboost as xgb  # Extreme Gradient Boosting (XGBoost) for machine learning
from sklearn.ensemble import GradientBoostingClassifier  # Gradient boosting classifier
from sklearn.metrics import confusion_matrix  # Creating and analyzing confusion matrices
import seaborn as sns  # Statistical data visualization
```

You can locate the project ID needed for Google Cloud authentication and BigQuery API usage as shown below.



In [ ]:
```
# Authenticate to Google Cloud
auth.authenticate_user()
project_id = 'quixotic-module-452415-s0' # Replace this project id with your project ID
client = bigquery.Client(project=project_id)
```

In [ ]:
```
# Function to run BigQuery queries and obtain pandas dataframe
def run_query(query):
    query_job = client.query(query)
    df = query_job.to_dataframe()
    return df
```

5. Retrieve data using a query and store it as a pandas dataframe. Back to Top

I combined two tables using case_id. In the clinical_gdc_current table, case_id is a unique ID for each patient. But in the masked_somatic_mutation_hg38_gdc_current table, one case_id can show up multiple times because each patient can have multiple mutations. So, when I joined these tables, the final dataset ended up with multiple rows for each case_id.

In [ ]:
```
# Retrieve data
# I acquired the following information from the meta data in isb-cgc database.
# case_id - GDC unique identifier for this case (corresponds to the case_barcode)
# proj__project_id - GDC-assigned identifier for the project to which a case belongs.
# demo__race - An arbitrary classification of a taxonomic group that is a division of a species
# demo__age_at_index - The patient's age (in years) on the reference or anchor date date used during date obfuscatior
# Hugo_Symbol - HUGO symbol for the gene (HUGO symbols are always in all caps). Unknown is used for regions that do r
# Variant_Classification - Translational effect of variant alle
# Variant_Type - Type of Mutation

clinical_query = """
SELECT
    c.case_id, --
    c.proj__project_id,
    c.demo__race,
    c.demo__age_at_index,
    m.Hugo_Symbol,
    m.Variant_Classification,
    m.Variant_Type
FROM
    isb-cgc-bq.TCGA.masked_somatic_mutation_hg38_gdc_current m
JOIN
```

```
    isb-cgc-bq.TCGA.clinical_gdc_current c
ON
    m.case_id = c.case_id
WHERE
    m.Hugo_Symbol IN ('CDH1','CTNNA1','APC','MLH1','MSH2','MSH6','PMS2','EPCAM','STK11','SMAD4','BMPR1A','TP53','PRKA
    AND c.proj__project_id LIKE '%STAD%'
    AND c.demo__race IN ('white', 'asian')
    AND c.demo__ethnicity = 'not hispanic or latino'
"""
# STAD for identifying the stomac cancer
# demo__ethnicity is set to 'non hispanic or latino'

df = run_query(query=clinical_query) # Read the data
```

6. Basic Exploration of the Data Back to Top

Before conducting analysis, I first explored the dataset to understand its structure and key characteristics. Here are some basic steps I took:

In [ ]:
```
# Number of rows and columns
print(f'Number of rows: {df.shape[0]}')
print(f'Number of columns: {df.shape[1]}')
```

```
Number of rows: 319
Number of columns: 7
```

In [ ]:
```
# Number of unique case_id in each race.
num_asian = len(df[df.demo__race == 'asian'].case_id.unique())
num_white = len(df[df.demo__race == 'white'].case_id.unique())
print(f'Number of Asian patients (unique case_id): {num_asian}')
print(f'Number of non-hispanic White patients (unique case_id): {num_white}')
```

```
Number of Asian patients (unique case_id): 53
Number of non-hispanic White patients (unique case_id): 145
```

In [ ]:
```
# The first five rows of the dataframe.
df.head(5)
```

Out[ ]:

	case_id	proj__project_id	demo__race	demo__age_at_index	Hugo_Symbol	Variant_Classification	Variant_Type
0	daecea36-b379-46ce-8ae9-a38d22556ce2	TCGA-STAD	white	67	TP53	Missense_Mutation	SNP
1	e17cf9e9-0902-44e0-962b-af5492876f7d	TCGA-STAD	white	59	TP53	Splice_Site	SNP
2	287059fb-4e69-4d6d-99d5-91ec92d35bde	TCGA-STAD	asian	60	APC	Nonsense_Mutation	SNP
3	a28e57e2-9071-4558-856a-162f7a837380	TCGA-STAD	white	67	TP53	Nonsense_Mutation	INS
4	9db0da77-797d-45d2-abdc-a586dea94ed8	TCGA-STAD	white	66	MSH2	Frame_Shift_Del	DEL

In [ ]:
```
# Count the number of genetic mutations in Asian and White
# I didn't consider the multiple records for each patient in this table.
df.groupby(['Hugo_Symbol', 'demo__race']).agg(count=('Hugo_Symbol', 'count')).unstack(level = 0)
```

Out[ ]:

	count													
Hugo_Symbol	**APC**	**BMPR1A**	**CDH1**	**CTNNA1**	**EPCAM**	**MLH1**	**MSH2**	**MSH6**	**PMS2**	**PRKAA1**	**PSCA**	**SMAD4**	**STK11**	**TP53**
demo__race														
asian	19.0	2.0	8.0	5.0	NaN	NaN	1.0	6.0	2.0	3.0	NaN	6.0	2.0	39.0
white	27.0	5.0	29.0	10.0	1.0	4.0	5.0	12.0	7.0	3.0	1.0	20.0	2.0	100.0

7. Two-Proportion Z-Test for Gene Mutation Differences Between Asian and non-hispanic White Gastric Cancer Patients Back to Top

Since the sample sizes for White and Asian patients are different, comparing the proportion of each gene mutation between Asian and White groups rather than just comparing frequency is the correct approach. A two-proportion Z-test will help determine if the difference in mutation proportions between the two groups is statistically significant.

In [ ]:
```
# List of genes to analyze
gene_mutations = ['CDH1', 'CTNNA1', 'APC', 'MLH1', 'MSH2', 'MSH6', 'PMS2', 'EPCAM', 'STK11', 'SMAD4', 'BMPR1A', 'TP5
#  Total # of unique patients in each group
total_white = df[df['demo__race'] == 'white']['case_id'].nunique()
total_asian = df[df['demo__race'] == 'asian']['case_id'].nunique()

# Count occurrences of each gene mutation per race
mutation_counts = df.groupby(['demo__race', 'Hugo_Symbol'])['case_id'].nunique().unstack()
```

In [ ]:
```
# It turns out that the gene mutations EPCAM and MLH1 have null values for the Asian group.
# Since it wouldn't be meaningful to compare these mutations between White and Asian groups in this case, I decided t
mutation_counts = mutation_counts.drop(columns = ['EPCAM','MLH1','PSCA'])
mutation_counts
```

Out[ ]:

Hugo_Symbol	APC	BMPR1A	CDH1	CTNNA1	MSH2	MSH6	PMS2	PRKAA1	SMAD4	STK11	TP53
demo__race											
asian	13.0	2.0	8.0	3.0	1.0	5.0	2.0	3.0	6.0	2.0	38.0
white	24.0	5.0	27.0	8.0	5.0	10.0	5.0	3.0	16.0	2.0	96.0

In [ ]:
```
# Calculate the proportions (mutation count / total patients)
mutation_proportions = mutation_counts.copy()
mutation_proportions.loc['white'] = mutation_proportions.loc['white']/total_white
mutation_proportions.loc['asian'] = mutation_proportions.loc['asian']/total_asian
```

In [ ]:
```
mutation_proportions
```

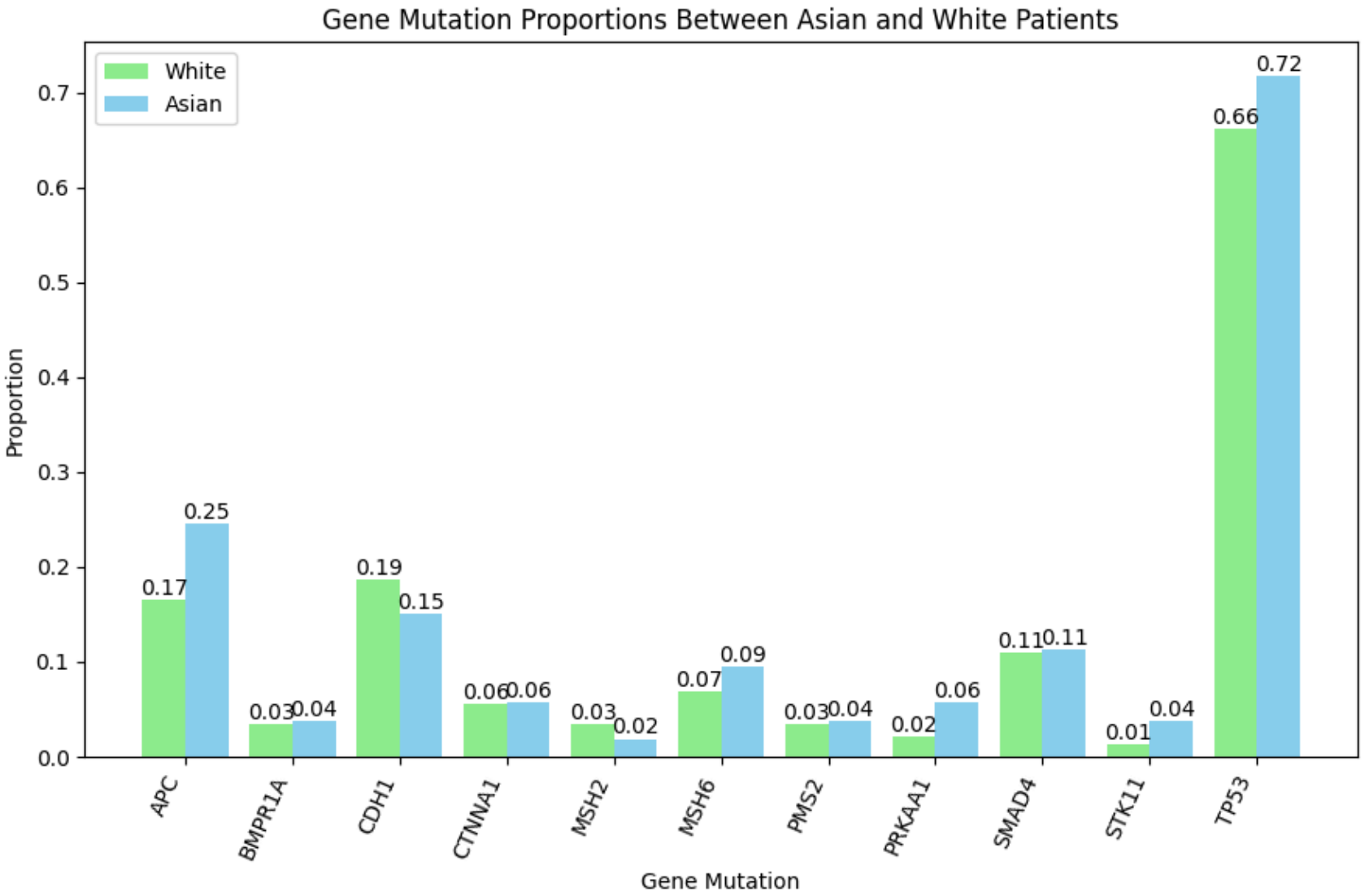
Out[ ]:

Hugo_Symbol	APC	BMPR1A	CDH1	CTNNA1	MSH2	MSH6	PMS2	PRKAA1	SMAD4	STK11	TP53
demo__race											
asian	0.245283	0.037736	0.150943	0.056604	0.018868	0.094340	0.037736	0.056604	0.113208	0.037736	0.716981
white	0.165517	0.034483	0.186207	0.055172	0.034483	0.068966	0.034483	0.020690	0.110345	0.013793	0.662069

In [ ]:
```
# Side-by-side bar charts comparing mutation proportiongs between Asian and White patients for each gene mutation.
genes = mutation_proportions.columns

# Get proportions for White and Asian groups
white_p = mutation_proportions.loc['white'].values
asian_p = mutation_proportions.loc['asian'].values
```

In [ ]:
```
# Set positions for bars
x = np.arange(len(genes))

# Set bar width
bar_width = 0.4

# Create bar chart
plt.figure(figsize=(9, 6))
bars1 = plt.bar(x - bar_width/2, white_p, width=bar_width, label='White', color='lightgreen')
bars2 = plt.bar(x + bar_width/2, asian_p, width=bar_width, label='Asian', color='skyblue')

# Add labels on top of bars with two decimal places
for bar in bars1:
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.005, f"{bar.get_height():.2f}",
             ha='center', fontsize=10, color='black')

for bar in bars2:
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.005, f"{bar.get_height():.2f}",
             ha='center', fontsize=10, color='black')

# Add labels and title
plt.xlabel('Gene Mutation')
plt.ylabel('Proportion')
plt.title('Gene Mutation Proportions Between Asian and White Patients')
plt.xticks(ticks=x, labels=genes, rotation=66, ha="right")
```

```
plt.legend()

# Show plot
plt.tight_layout()
plt.show()
```



It turns out that, except for CDH1 and MSH2, most gene mutation proportions are higher in the Asian group. However, can we conclude that Asians have more mutations in these genes compared to the non-Hispanic White group? To confirm this, I conducted

a two-proportion Z-test.

The formula for the **two-proportion Z-test** is:

$$Z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Where:

- $\hat{p}_1$ = Proportion of success in group 1 (e.g., mutation rate in White group)
- $\hat{p}_2$ = Proportion of success in group 2 (e.g., mutation rate in Asian group)
- n_1 = Sample size of group 1
- n_2 = Sample size of group 2
- $\hat{p}$ = **Pooled proportion**, calculated as:

  $$\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$$

  where x_1 and x_2 are the number of "successes" (e.g., patients with a specific mutation) in each group.

How to Interpret the Z-Test:

- If **p-value < 0.05**, the difference between the two groups is **statistically significant**.
- If **p-value ≥ 0.05**, the difference is **not statistically significant**.

In [ ]:
```
# I used the proportions_ztest function from statsmodels package.
from statsmodels.stats.proportion import proportions_ztest

# List of genes to analyze
genes_list = mutation_proportions.columns.tolist()
```

In [ ]:
```
# Perform two-proportion Z-tests for each gene
proportion_z_test = []
for gene in genes_list:
```

```
    count_white = mutation_counts.loc['white', gene] if gene in mutation_counts.columns else 0
    count_asian = mutation_counts.loc['asian', gene] if gene in mutation_counts.columns else 0

    p_white = count_white / total_white
    p_asian = count_asian / total_asian

    # Perform Z-test
    count = np.array([count_white, count_asian])
    num_observation = np.array([total_white, total_asian])
    stat, p_val = proportions_ztest(count, num_observation)

    proportion_z_test.append({'Gene': gene, 'White_Proportion': p_white, 'Asian_Proportion': p_asian,
                    'Z-Statistic': stat, 'P-Value': p_val})

# Convert results to DataFrame
result = pd.DataFrame(proportion_z_test)

result
```

Out[ ]:

	Gene	White_Proportion	Asian_Proportion	Z-Statistic	P-Value
0	APC	0.165517	0.245283	-1.274845	0.202364
1	BMPR1A	0.034483	0.037736	-0.109745	0.912612
2	CDH1	0.186207	0.150943	0.575907	0.564678
3	CTNNA1	0.055172	0.056604	-0.038930	0.968946
4	MSH2	0.034483	0.018868	0.567500	0.570375
5	MSH6	0.068966	0.094340	-0.597412	0.550232
6	PMS2	0.034483	0.037736	-0.109745	0.912612
7	PRKAA1	0.020690	0.056604	-1.305249	0.191808
8	SMAD4	0.110345	0.113208	-0.056750	0.954744
9	STK11	0.013793	0.037736	-1.060224	0.289043
10	TP53	0.662069	0.716981	-0.731443	0.464508

8. Result of Two-Proportion Z-Test Back to Top

From the two-proportion Z-test, I found that none of the gene mutation comparisons were statistically significant, as all p-values were greater than 0.05. This result was expected from the start since the dataset includes only 53 Asian and 145 non-Hispanic White patients, which is too small of a sample size to produce reliable statistical conclusions.

However, I don't think it's just a coincidence that most gene mutation proportions are higher in the Asian group. In particular, APC and PRKAA1 stand out, with p-values around 0.2, suggesting that these two genes may have the biggest differences in mutation frequency between Asian and White patients. While this is not statistically significant, it could indicate a trend worth investigating further with a larger dataset.

1) APC gene

- To better understand the APC gene and its role in causing gastric cancer, I studied information from the paper at this link.
- According to this paper, the APC gene (Adenomatous Polyposis Coli) plays a important role in controlling cell growth. When this gene is working properly, it helps prevent cells from growing uncontrollably. However, mutations in the APC gene can lead to a condition called Familial Adenomatous Polyposis, where many polyps (small abnormal growths) form in the digestive tract, especially in the stomach and intestines.
- In some cases, these polyps can turn into gastric adenocarcinoma, a type of stomach cancer. A related condition, Gastric Adenocarcinoma and Proximal Polyposis of the Stomach (GAPPS), is also linked to mutations in APC. In both conditions, stomach cancer often develops after the appearance of fundic gland polyps, which are small, benign (non-cancerous) growths in the stomach lining.
- This means that mutations in the APC gene can increase the risk of stomach cancer, especially in people with a family history of these conditions.

2) PRKAA1 gene

- I studied information about this gene at this link
- The PRKAA1 gene helps control how cells use and manage energy. It is part of a system called the AMP-activated protein kinase (AMPK) pathway, which works like a cellular energy sensor. When a cell is running low on energy, this system helps slow down unnecessary activities and focus on survival.
- PRKAA1 helps regulate how fast cells grow and divide. If something goes wrong with this gene, cells might grow too quickly and uncontrollably, which is one of the main causes of cancer.

- Normally, the AMPK pathway acts as a brake to prevent cells from multiplying too fast. If PRKAA1 is mutated, this brake might not work properly, making it easier for cancer to develop.
- Studies have shown that certain genetic changes (mutations) in PRKAA1 may increase the risk of stomach cancer, especially in Asian populations. Scientists believe these mutations could make cells more vulnerable to damage from things like inflammation, poor diet, or environmental factors.

9. Machine Learning-Based Cancer Type Prediction Using TCGA Data Back to Top

Originally, my plan was to build a machine learning model to predict whether a person is White or Asian based on their gene mutation data. However, as I found in the first part of my research, the sample size was too small to create a good model. So, I decided to change my approach and use more TCGA data instead.

I will now use Hugo_Symbol, Variant_Classification, and Variant_Type as the independent variables, and primary_site (cancer type) as the target variable to build a multi-class prediction model. The challenge is that primary_site has 53 cancer types, like skin, colon, brain, pancreas, etc., and some of these cancer types are not well represented in the dataset. To solve this, I decided to focus only on 5 cancer types: 'Bronchus and lung', 'Breast', 'Brain', 'Kidney', and 'Corpus uteri', as these have more data available.

Description of the independent variables:

- Hugo_Symbol: This field represents the gene symbol (identifier) for a specific gene involved in a mutation. Examples: TP53, CDH1, MLH1, etc.
- Variant_Classification: This field describes the type of mutation in the gene. It indicates how the mutation affects the gene function or protein structure. Below are some examples of Variant_Classification
  - Missense_Mutation: A single nucleotide change that results in a different amino acid in the protein.
  - Silent: A change that does not alter the protein.
  - Nonsense_Mutation: A change that creates a premature stop codon, leading to a shortened protein.
- Variant_Type: This field represents the type of genetic variation. It indicates how the mutation affects the DNA sequence. Below are some examples of Variant_Type:
  - SNP (Single Nucleotide Polymorphism): A variation in a single nucleotide.
  - DEL (Deletion): A segment of the DNA is missing.
  - INS (Insertion): Extra nucleotides are inserted into the DNA sequence.

10. Retrieve TCGA data and store it as a pandas dataframe. Back to Top

I included all available data for the target variable, primary_site, as well as for independet variablles, such as race, Hugo Symbol, Variant_Classification, and Variant_Type, without filtering out any data.

In [ ]:
```
clinical_query = """
SELECT
    c.case_id, --
    c.demo__race,
    m.primary_site,
    m.Hugo_Symbol,
    m.Variant_Classification,
    m.Variant_Type
FROM
    isb-cgc-bq.TCGA.masked_somatic_mutation_hg38_gdc_current m
JOIN
    isb-cgc-bq.TCGA.clinical_gdc_current c
ON
    m.case_id = c.case_id
"""
# STAD for identifying the stomac cancer
# demo__ethnicity is set to 'non hispanic or latino'

df = run_query(query=clinical_query) # Read the data
```

In [ ]:
```
# First 5 rows of the data
df.head()
```

Out[ ]:

	case_id	demo__race	primary_site	Hugo_Symbol	Variant_Classification	Variant_Type
0	fc5ad666-d67a-4a5c-8e4e-1c8d099e9f85	white	Brain	ITGAD	Silent	SNP
1	521ea765-1bd1-423d-a75d-091243df37a9	black or african american	Brain	UVRAG	Missense_Mutation	SNP
2	b46263ab-c3ca-4fda-a895-74c7e6e6fe22	white	Ovary	ZNF415	Missense_Mutation	SNP
3	8652ddee-98f4-4584-b450-e6f2f5c9d7ec	white	Ovary	TP53TG5	Missense_Mutation	SNP
4	6c43727d-631a-40fb-b1a1-242a12889eaa	white	Ovary	SYP	Missense_Mutation	SNP

11. Data Cleaning and Structuring Back to Top

The target variable, primary_site, contains 53 cancer types, but some, like Gallbladder, Other and ill-defined sites, and Meninges, have very few occurrences. To ensure a reliable model, I decided to keep only the cancer types that appear more than 500 times in the dataset, meaning more than 500 patients with those cancers. The five cancer types that meet this criterion are **Bronchus and lung, Breast, Brain, Kidney, and Corpus uteri.**

In [ ]:
```
# Find out the five cancer times with more than 500 patients
grouped = df.groupby(['case_id', 'primary_site']).size().reset_index(name = 'count')
grouped.sort_values(by = 'case_id').head()
grouped_count = grouped.primary_site.value_counts().reset_index()
cancer_interest = grouped_count[grouped_count['count'] > 500].primary_site.values.tolist()
print('Five cancer types: ', cancer_interest)

df = df[(df.primary_site.isin(cancer_interest))]
```

```
Five cancer types:  ['Bronchus and lung', 'Breast', 'Brain', 'Kidney', 'Corpus uteri']
```

Since the original table (df) contains multiple gene mutation records for each case_id (patient ID), I reshaped the dataset so that each patient has only one record. Additionally, I transformed all categorical variables (race, Hugo_Symbol, Variant_Classification, and Variant_Type) into separate columns with binary values (1 or 0).

In [ ]:
```
# demo__race and primary_site
df_grouped = df.groupby(['case_id']).agg({
    'demo__race': 'first',  # Race remains the same for each patient
    'primary_site': 'first',  # Target variable
})

# Transform categorical variables into binary: demo__race, Hugo_Symbol, Variant_Classification, Variant_Type
race_dummies = pd.get_dummies(df_grouped['demo__race'], prefix='race', dtype=int)
df_grouped = df_grouped.drop('demo__race', axis=1).join(race_dummies)

hugo_dummies = df[['case_id','Hugo_Symbol']].pivot_table(index='case_id', columns='Hugo_Symbol', aggfunc=lambda x: 1,
variant_class_dummies = df[['case_id','Variant_Classification']].pivot_table(index='case_id', columns='Variant_Classi
variant_type_dummies = df[['case_id','Variant_Type']].pivot_table(index='case_id', columns='Variant_Type', aggfunc=la
```

I found out that both Variant_Type and Variant_Classification had the same value, 'INS', in their records. To avoid confusion and errors when combining the two tables, I renamed the 'INS' column in the Variant_Type dummy table to INS_Type.

In [ ]:
```
variant_type_dummies['INS_Type'] = variant_type_dummies['INS']
variant_type_dummies.drop(columns = 'INS', inplace = True)
```

Combining all the transformed tables and create df_final

In [ ]:
```
df_final = df_grouped.join([hugo_dummies, variant_class_dummies, variant_type_dummies])
```

I saved the df_final table in my Google Drive to avoid recreating it while testing different approaches. You can skip this step, but if you plan to run additional tests, you may also want to save the data in your Google Drive.

In [ ]:
```
# Save df_final in my google drive (you can skip this part)
#from google.colab import drive
#drive.mount('/content/drive')
# df_final.to_csv('/content/drive/MyDrive/Colab Notebooks/df_final.csv', index=False)
```

```
Mounted at /content/drive
```

In [ ]:
```
# Read df_final table from my google drive. (you can skp this part)
#df_final = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/df_final.csv')
```

In [ ]:
```
# Size of the dataset
print('Number of rows: ', df_final.shape[0])
```

```
print('Number of columns: ', df_final.shape[1])
```

```
Number of rows:  4127
Number of columns:  19291
```

In [ ]:
```
# As shown below, the table has 19,291 columns from the four categocial variables: race, Hugo_Symbol, Variant_Classif
df_final.head()
```

Out[ ]:

	primary_site	race_american indian or alaska native	race_asian	race_black or african american	race_native hawaiian or other pacific islander	race_white	A1BG	A1CF	A2M	A2ML1	...	RNA	Silent	Sp
0	Brain	0	0	0	0	1	0	0	0	0	...	1	1	
1	Breast	0	0	0	0	1	0	0	0	0	...	0	1	
2	Corpus uteri	0	0	0	0	0	0	0	0	0	...	0	1	
3	Kidney	0	0	0	0	1	0	0	0	0	...	1	1	
4	Corpus uteri	0	0	0	0	1	0	0	0	0	...	0	1	

5 rows × 19291 columns

12. Model Development: XGBoost Back to Top

I know I need to perform Exploratory Data Analysis (EDA) and Feature Transformation before developing and testing different models. However, since this dataset contains 19,291 columns, conducting a thorough EDA and transformation would be quite challenging. I also know that I can perform dimentinality reduction to reduce the number of columns, but, this is a personal research project focused on exploring the potential of TCGA data rather than strictly following every step of a typical machine learning workflow. So, even though some typical machine learning steps might be skipped, this research can still give useful insights.

I'm going to build two types of models — XGBoost and a neural network using TensorFlow — and compare their performances.

- XGBoost Model: XGBoost (Extreme Gradient Boosting) is a machine learning algorithm that uses many decision trees to make better predictions. It is great at handling large datasets and complex data. XGBoost has won multiple Kaggle competitions because it is fast, efficient, and works well even with missing data. This makes it a strong choice for predicting cancer types based on gene mutations.

In [ ]:
```
# Necessary Pagakes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
# Split the dataset into X (independent variables) and y (target variable)
X = df_final.drop(columns=['primary_site'])  # Independent variables
y = df_final['primary_site']  # Target variable
```

In [ ]:
```
# Encode target variable (since XGBoost requires numerical labels)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

In [ ]:
```
y_labels = pd.DataFrame({'target':y, 'encoded':y_encoded})
```

In [ ]:
```
# labels with encoded values
y_labels.groupby(['target', 'encoded']).size()
```

Out[ ]:

target	encoded	0
Brain	**0**	882
Breast	**1**	969
Bronchus and lung	**2**	1043
Corpus uteri	**3**	514
Kidney	**4**	719

dtype: int64

In [ ]:
```
# Split data into training and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)
```

I know I could do the parameter tuning using Grid Search or other techniques, but I'm building an XGBoost model simply to explore the potential of the data.

In [ ]:
```
# Train an XGBoost classifier
clf = XGBClassifier(
    objective='multi:softprob',  # Multi-class classification
    num_class=len(label_encoder.classes_),  # Number of unique cancer types
    eval_metric='mlogloss',  # Multi-class log loss
    n_estimators=100,  # Number of boosting rounds
    learning_rate=0.1,  # Step size shrinkage
    max_depth=6,  # Maximum tree depth
    random_state=42
)
clf.fit(X_train, y_train)
```

Out[ ]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
              max_leaves=None, min_child_weight=None, missing=nan,
```

13. Result of XGBoost Model Back to Top

The overall accuracy of the XGBoost model is 0.79, which means the model correctly predicts the cancer type 79% of the time. For specific cancer types, the model performed like this:

- Brain cancer: 0.85 accuracy (85% correct predictions)
- Breast cancer: 0.71 accuracy (71% correct predictions)
- Bronchus and lung cancer: 0.83 accuracy (83% correct predictions)
- Corpus uteri cancer: 0.82 accuracy (82% correct predictions)
- Kidney cancer: 0.72 accuracy (72% correct predictions)

The highest accuracy was for Brain cancer at 85%, and the lowest accuracy was for Kidney cancer at 72%. Even though I didn't use advanced techniques like parameter tuning to find the best settings, I think this is a good result. If I do things like feature selection, dimensionality reduction, or include more information like age, I believe I can improve the accuracy even further.

Feature Importance

From the feature importance, I found that the top 20 features are all related to Hugo_Symbol, which contains information about gene mutations. Among these, the top 3 most important features for the model are IDH1, VHL, and CSMD3. These genes have the biggest impact on predicting the cancer types in the model. Below are the some description of these genes I found from Google:

- IDH1: Mutations (changes) in the IDH1 gene have been found in several genetic conditions and in some types of cancer, including acute myelogenous leukemia, myelodysplastic syndromes, and brain cancer.

- VHL: Mutations in the VHL gene have been identified in a type of tumor called a hemangioblastoma. These tumors are made of newly formed blood vessels and tend to develop in the brain and spinal cord
- CSMD3: CSMD3 is thought to be a tumor suppressor gene. Its mutations may help cancer cells avoid immune detection and influence tumor growth. It's associated with cancers like gliomas and colorectal cancer.

In [ ]:
```python
# Evaluate the model
y_pred = clf.predict(X_test)

# Convert predictions back to original labels
y_pred_labels = label_encoder.inverse_transform(y_pred)
y_test_labels = label_encoder.inverse_transform(y_test)

# Print accuracy and classification report
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test_labels, y_pred_labels))
```

```
Accuracy: 0.7820823244552058
                   precision    recall  f1-score   support

            Brain       0.85      0.74      0.79       176
           Breast       0.71      0.75      0.73       194
Bronchus and lung       0.83      0.86      0.85       209
     Corpus uteri       0.82      0.75      0.78       103
           Kidney       0.72      0.78      0.75       144

         accuracy                           0.78       826
        macro avg       0.79      0.78      0.78       826
     weighted avg       0.79      0.78      0.78       826
```

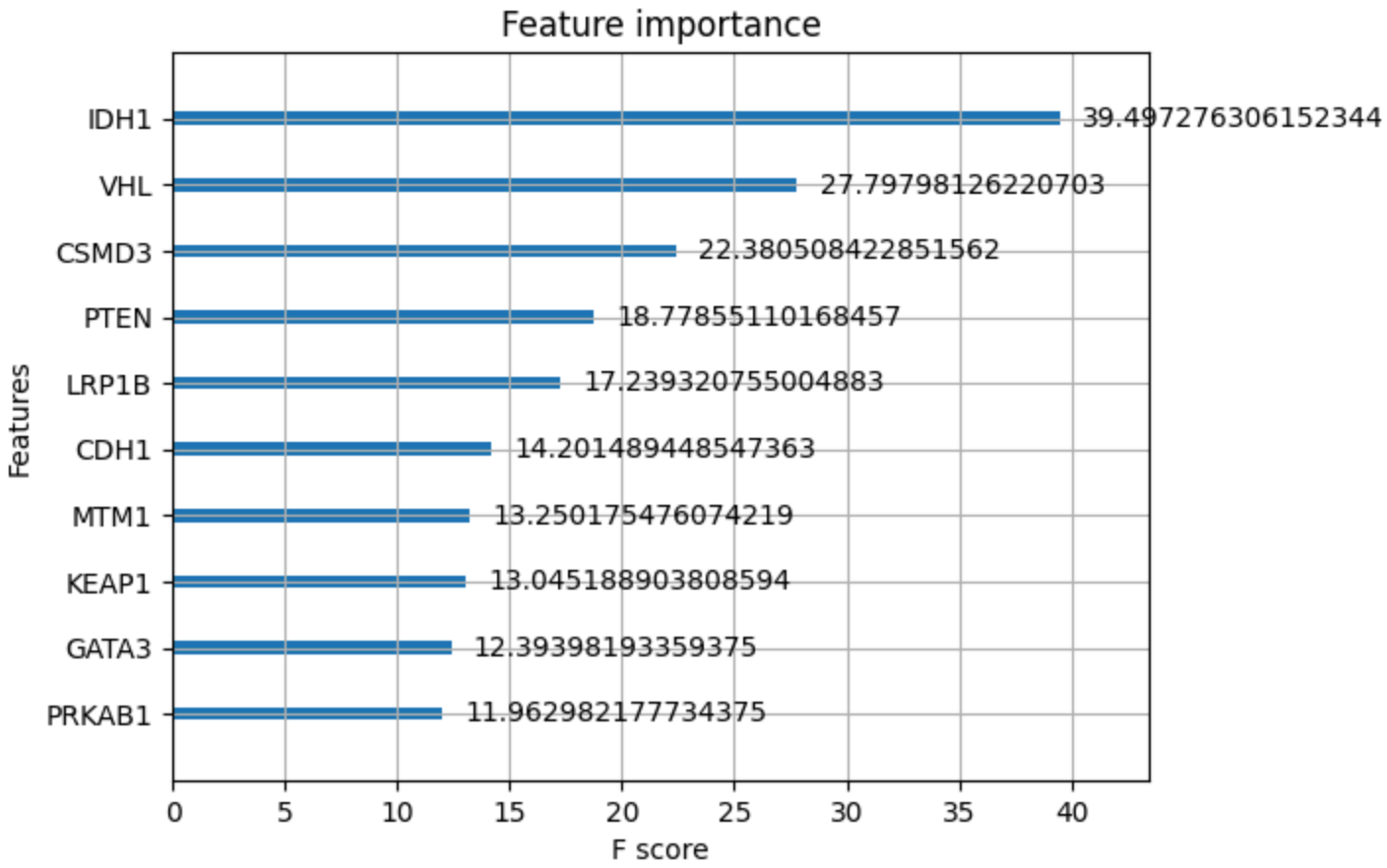
In [ ]:
```python
# Feature Importance: Top 10 Features
feature_importance = clf.get_booster().get_score(importance_type='gain')
importance_df = pd.DataFrame({'Feature': feature_importance.keys(), 'Importance': feature_importance.values()})
importance_df = importance_df.sort_values(by="Importance", ascending=False)
importance_df.head(10) # Show top 10 features
```

Out[ ]:

	Feature	Importance
649	IDH1	39.497276
1529	VHL	27.797981
349	CSMD3	22.380508
1157	PTEN	18.778551
780	LRP1B	17.239321
250	CDH1	14.201489
865	MTM1	13.250175
726	KEAP1	13.045189
563	GATA3	12.393982
1132	PRKAB1	11.962982

In [ ]:

```
# Get top 10 feature importance chart
xgb.plot_importance(clf, max_num_features=10, importance_type="gain")  # Show top 10 features
plt.show()
```



14. Neural Network Model with TensorFlow Package Back to Top

I built a neural network model using Keras with several layers to learn patterns in the data and predict cancer types. The model starts with an input layer that matches the number of features in the dataset. Then, I added three hidden layers with 128, 64, and 32 neurons to help the model learn from the data. This design isn't that deep or complex, but I think it's enough for the research purpose. I used ReLU activation in these layers, which helps the model learn complex relationships in the data. To avoid the model getting too focused on the training data (called overfitting), I added dropout layers, which randomly turn off some neurons during training. In the final output layer, I used the softmax activation function. I learned that softmax helps the model give a probability for each cancer type, which allows it to predict the most likely cancer for each patient. This design helps the model learn from the data, make accurate predictions, and avoid memorizing the training data.

- Neural Network (TensorFlow) Model: A neural network is a type of deep learning model that works like a simplified version of the human brain. It has layers of connected "neurons" that help it learn patterns in data. Using the TensorFlow package, this model can recognize relationships between gene mutations and cancer types. Neural networks are powerful because they can find hidden patterns that other models might miss, making them useful for multi-class classification problems like this one.

In [ ]:
```
# Encode target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

In [ ]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)

# Convert target labels to categorical (one-hot encoding for multi-class classification)
num_classes = len(label_encoder.classes_)
y_train_categorical = keras.utils.to_categorical(y_train, num_classes)
y_test_categorical = keras.utils.to_categorical(y_test, num_classes)
```

In [ ]:
```
# Build the Deep Learning Model
model = keras.Sequential([
    keras.layers.Input(shape=(X_train.shape[1],)),  # Input layer with correct feature count
    keras.layers.Dense(128, activation='relu'),  # First hidden layer
    keras.layers.Dropout(0.5),  # Dropout to prevent overfitting
    keras.layers.Dense(64, activation='relu'),  # Second hidden layer
    keras.layers.Dropout(0.5),  # Dropout to prevent overfitting
    keras.layers.Dense(32, activation='relu'),  # Third hidden layer
    keras.layers.Dense(num_classes, activation='softmax')  # Output layer (multi-class classification)
])
```

In [ ]:
```
# Compile the Model
from tensorflow.keras.callbacks import EarlyStopping
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# I added early_stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

In [ ]:
```
history = model.fit(X_train, y_train_categorical, epochs=50, batch_size=32, validation_data=(X_test, y_test_categori
```

```
Epoch 1/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 4s 29ms/step - accuracy: 0.2773 - loss: 1.5800 - val_accuracy: 0.6247 - val_loss: 1.2089
Epoch 2/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.5540 - loss: 1.2507 - val_accuracy: 0.7518 - val_loss: 0.8119
Epoch 3/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.7787 - loss: 0.6699 - val_accuracy: 0.7724 - val_loss: 0.6980
Epoch 4/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.8918 - loss: 0.4526 - val_accuracy: 0.7906 - val_loss: 0.6088
Epoch 5/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9424 - loss: 0.2863 - val_accuracy: 0.8027 - val_loss: 0.6259
Epoch 6/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - accuracy: 0.9579 - loss: 0.2295 - val_accuracy: 0.8027 - val_loss: 0.6177
Epoch 7/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9730 - loss: 0.1235 - val_accuracy: 0.8087 - val_loss: 0.5670
Epoch 8/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.9784 - loss: 0.1160 - val_accuracy: 0.7906 - val_loss: 0.6234
Epoch 9/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9826 - loss: 0.0959 - val_accuracy: 0.8039 - val_loss: 0.6481
Epoch 10/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.9853 - loss: 0.0895 - val_accuracy: 0.8063 - val_loss: 0.6674
Epoch 11/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9828 - loss: 0.0687 - val_accuracy: 0.7845 - val_loss: 0.7766
Epoch 12/50
104/104 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.9844 - loss: 0.0713 - val_accuracy: 0.7978 - val_loss: 0.6880
```

15. Result of Neural Network Model Back to Top

I initially built the neural network model without testing different dropout rates or adding EarlyStopping, and I got an accuracy of 0.77, which wasn't great. However, after experimenting with various dropout rates, I found that using 50% dropout between both layers gave me a better result, with an accuracy of 0.81. I also added EarlyStopping, which stops the model from training once it starts to overfit.

I think an accuracy of 0.81 is a good result, especially since it's better than the XGBoost model's accuracy. However, since I didn't do things like parameter tuning for XGBoost, I can't definitively say the neural network model is better. That said, the neural network only has 3 layers with a relatively small number of neurons (128, 64, and 32). If I try a deeper neural network with more layers, I believe it could give me even better results.

Another issue for neural network model is the size of the dataset. Neural networks usually work best with large amounts of data, but in this case, there are only 4,127 patient records in the dataset. This is a pretty small amount for a neural network, which might

make it harder for the model to learn properly and make accurate predictions.

In [ ]:
```
# Evaluate the Model
test_loss, test_acc = model.evaluate(X_test, y_test_categorical)
print(f"\nTest Accuracy: {test_acc:.4f}")
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8179 - loss: 0.5646

Test Accuracy: 0.8087
```

In [ ]:
```
# Generate Classification Report
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)  # Convert probabilities to class labels
y_test_labels = label_encoder.inverse_transform(y_test)
from sklearn.metrics import classification_report
print(classification_report(y_test_labels, label_encoder.inverse_transform(y_pred)))
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
                   precision    recall  f1-score   support

            Brain       0.85      0.83      0.84       176
           Breast       0.70      0.75      0.72       194
Bronchus and lung       0.87      0.90      0.89       209
     Corpus uteri       0.83      0.70      0.76       103
           Kidney       0.82      0.80      0.81       144

         accuracy                           0.81       826
        macro avg       0.81      0.80      0.80       826
     weighted avg       0.81      0.81      0.81       826
```

16. Final Thoughts Back to Top

After completing this project, I can say that it was both challenging and rewarding! I learned a lot about cancer data, especially TCGA, and how to work with bioinformatics data. At first, finding and understanding the datasets, including cancer types and gene mutations, was much more time-consuming than I expected. But as I explored the data and built machine learning models, I started to appreciate how powerful bioinformatics can be in cancer research.

I also enjoyed experimenting with different models like XGBoost and neural networks, and it was interesting to see how different techniques affected accuracy. One of the biggest takeaways for me was realizing that working with biological data isn't just about

coding—it requires a lot of background knowledge in genetics and cancer biology, which I found really fascinating.

Through this project, I became even more interested in bioinformatics, and I hope to study it further in college. I want to keep learning how to use machine learning and data science to help with medical research, and maybe one day, contribute to real-world discoveries in genetics and cancer treatment!

In [ ]: