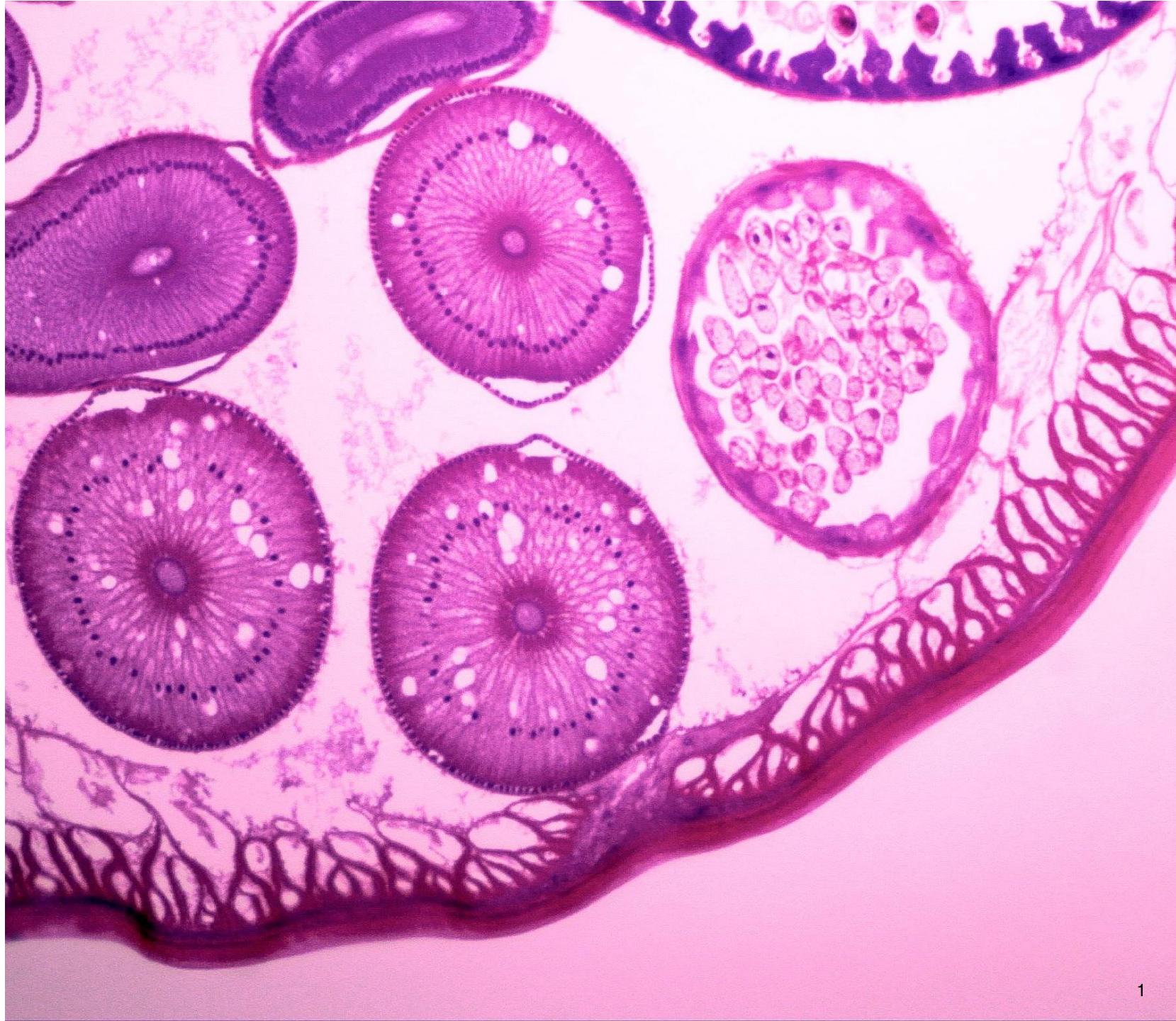


# **Early detection of Gastrointestinal cancers using Machine learning and blood biomarkers**

INFO-I 501 – INTRO TO INFORMATICS

Group 4 - Rashmita Kudamala, Mahitha Gogu, Geethanjali Karuturi, Fredrick Onyango



# Background:

## Early Detection of GI Cancers: Challenges & Innovations



### High Mortality

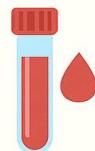
GI cancers (e.g., gastric, colorectal) are leading causes of cancer deaths worldwide. Late-stage diagnosis due to vague symptoms & poor early detection tools.

(Matsuoka & Yashiro, 2018; 2023)



### Current Methods = Limitations

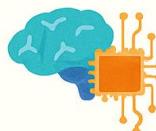
Endoscopy, imaging:  
Invasive, costly, not scalable



### Blood-Based Biomarkers = Game-Changer

Non-invasive, affordable,  
early-stage detection

(Li et al., 2021)



### Machine Learning Power

Analyzes complex biomarker patterns  
Enhances early, accurate  
cancer classification

(Sato et al., 2023; Tabari et al., 2022)

## Global incidence rate of gastric cancer

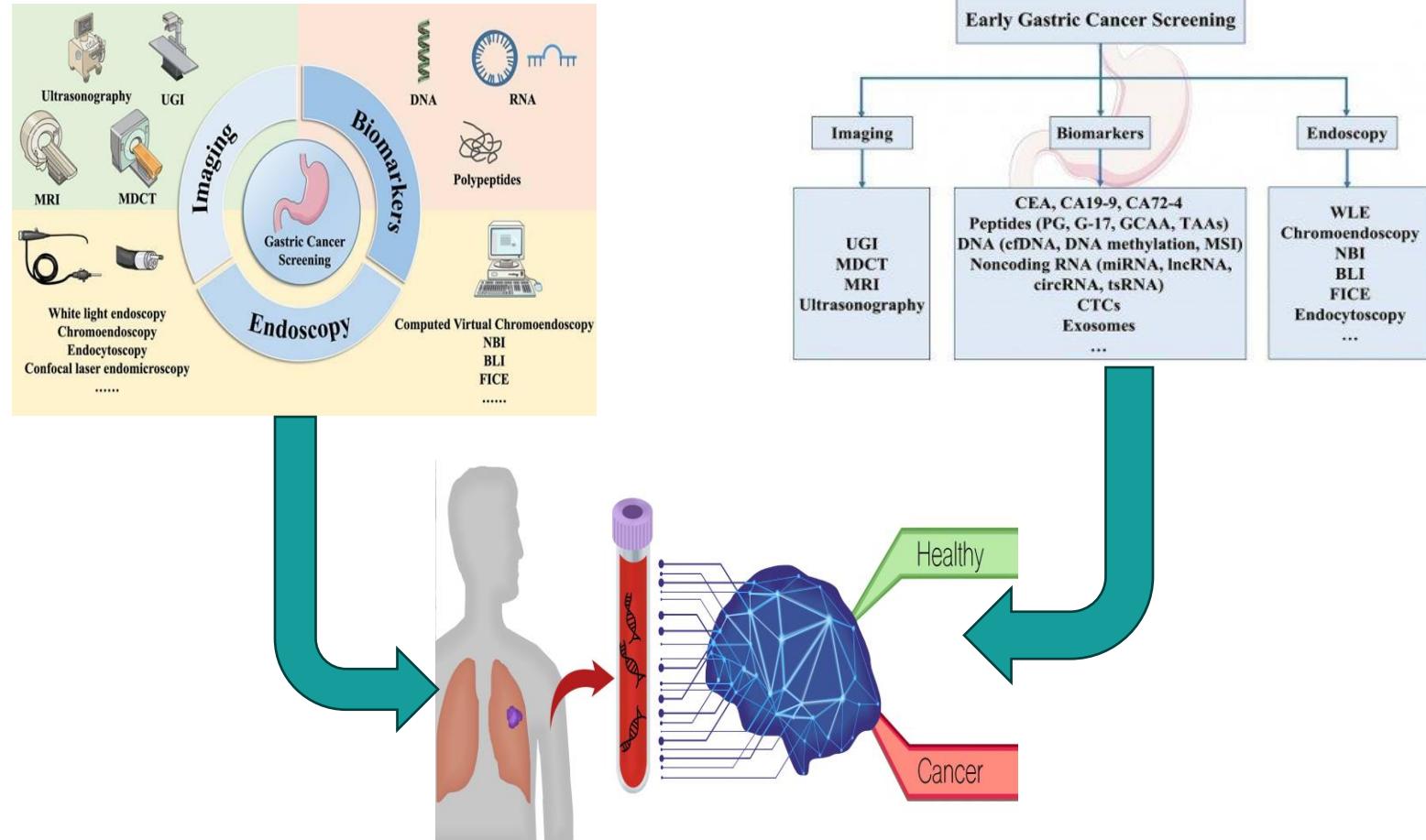


The incidence varies globally, especially high in East Asia and east Europe

# Introduction:

## Gap in literature:

- Blood biomarkers are a non-invasive, scalable alternative.
- Their early detection potential is underused.
- ML is underapplied to multi-biomarker data.
- This project applies ML to CancerSeek for affordable, early GI cancer diagnosis.



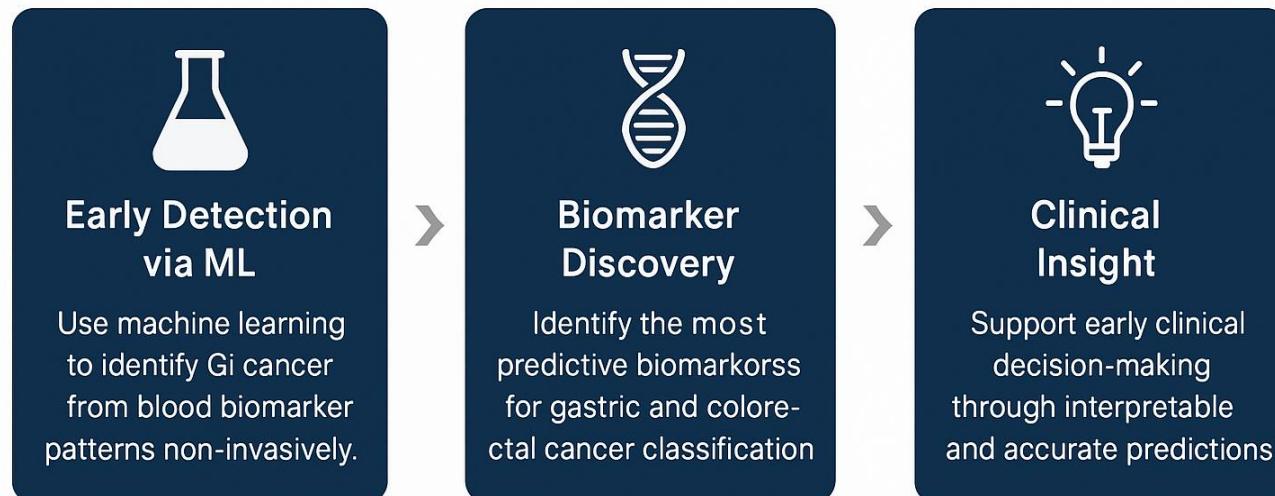
ML for non-invasive GI cancer detection

# Project Overview

## Aim:

To develop a machine learning model for the early detection of gastrointestinal (GI) cancers, specifically gastric and colon cancers, using blood biomarkers and classify GI cancer patients from normal individuals

## Purpose:



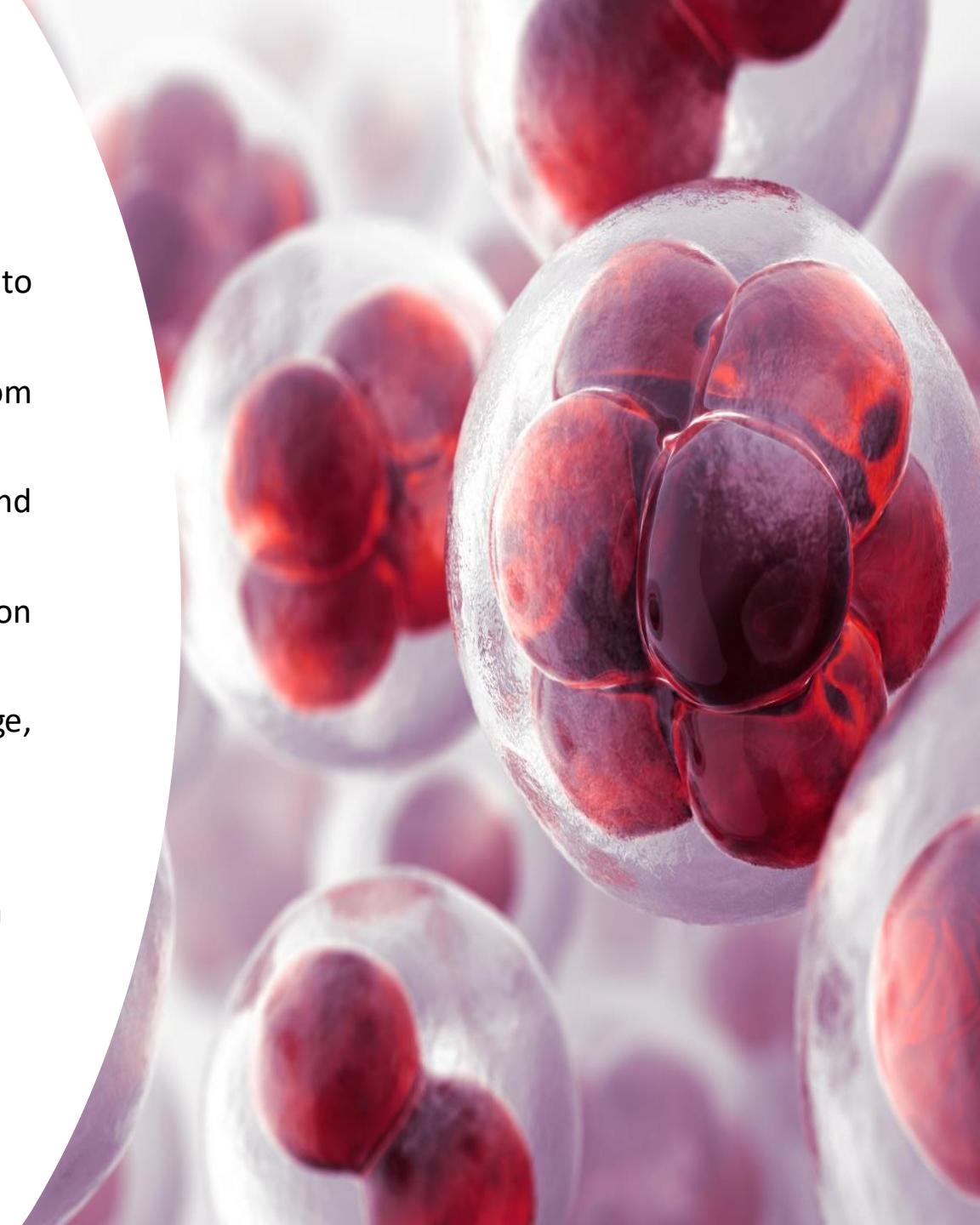
# Project Overview

## Objectives:

- Apply standard clinical thresholds to convert biomarker values into categorical flags indicating abnormality
- Identify statistically significant biomarkers differentiating cancer from normal using appropriate tests
- Address dataset imbalance through oversampling, undersampling, and bootstrapping methods
- Build and evaluate machine learning models to classify GI cancer based on biomarker profiles
- Analyze feature importance and explore demographic associations (age, sex, race) with cancer occurrence

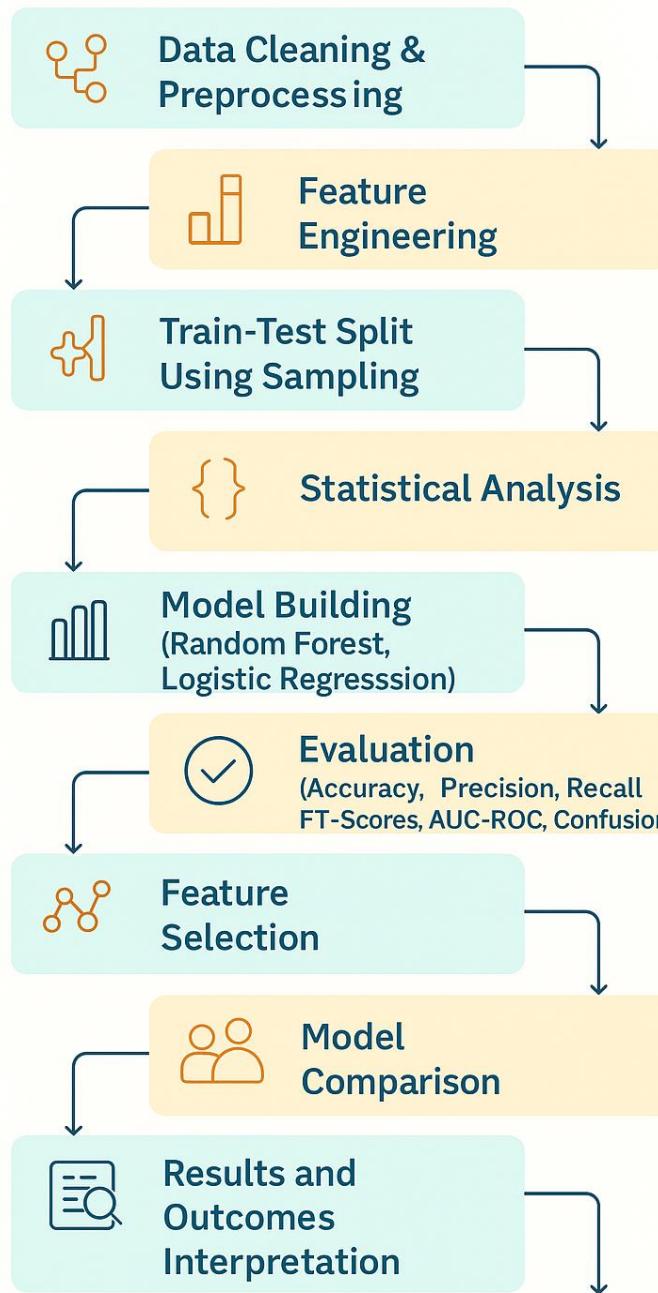
## Hypothesis:

- **Null Hypothesis ( $H_0$ ):** There is no statistically significant difference in biomarker values between GI cancer and normal patients.
- **Alternative Hypothesis ( $H_1$ ):** There is a statistically significant difference in biomarker values between GI cancer and normal patients.



# Step-by-Step Plan

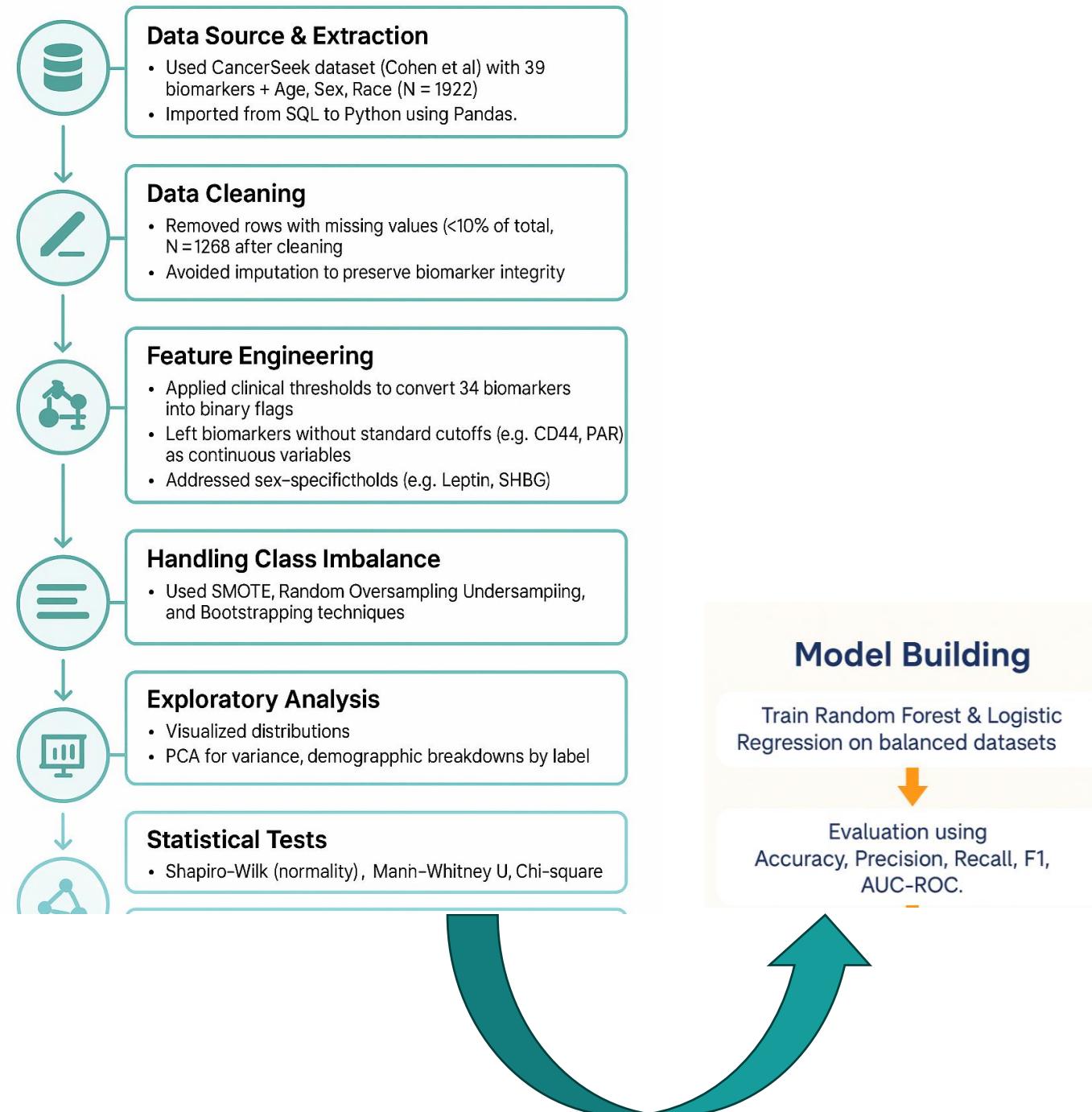
## Project Pipeline



# Methodology

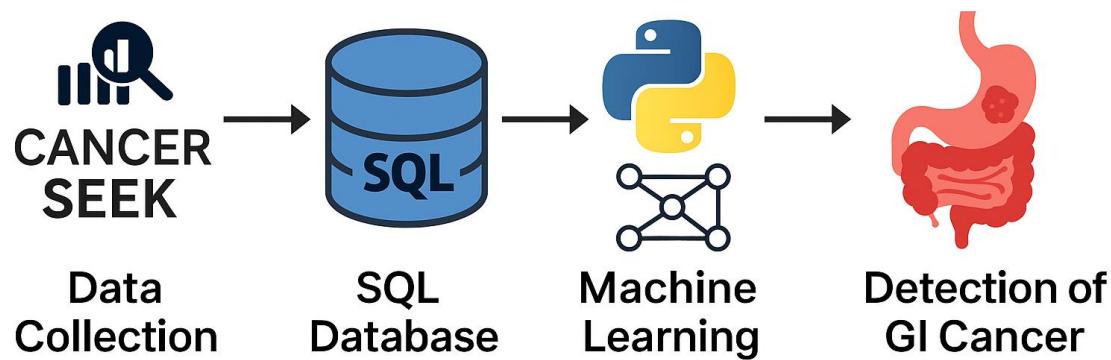
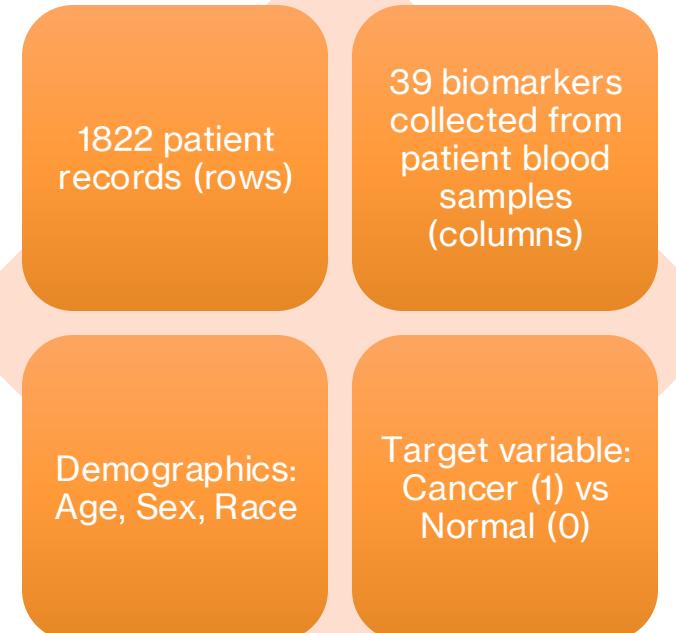
## Study type:

- Quantitative, retrospective ML study using blood biomarkers to detect GI cancers (gastric & colorectal).
- Combines statistical analysis and predictive modeling for binary classification (Normal vs Cancer).



# Data Overview

- Data collected from CancerSeek dataset (Cohen et al., Supplementary Material), along with demographic variables such as sex, age, and race/ethnicity. Link for the dataset - <https://www.science.org/doi/full/10.1126/science.aar3247>





# Data Cleaning & Preparation

# Data Cleaning & Preparation

1. Merged biomarker and clinical data into a single unified dataframe (merged\_df)

	Patient ID #_from_df1	Sample_ID	Primary tumor sample ID #	Age	Sex	Race	Tumor type_from_df1	AJCC Stage_from_df1	Histopathology	Plasma volume (mL)	Plasma DNA concentration (ng/mL)	CancerSEI Logist Regressive Score_from_d
0	CRC 455	CRC 455 PLS 1	Not available	60	Male	Caucasian	Colorectum	I	Adenocarcinoma	5	6.08	0.9
1	CRC 456	CRC 456 PLS 1	CRC 456 PT1	59	Female	Caucasian	Colorectum	I	Adenocarcinoma	4	46.01	0.9
2	CRC 457	CRC 457 PLS 1	CRC 457 PT1	69	Female	Caucasian	Colorectum	II	Adenocarcinoma	4.5	6.94	0.8
3	CRC 458	CRC 458 PLS 1	CRC 458 PT1	70	Female	Caucasian	Colorectum	II	Adenocarcinoma	7.5	7.15	0.6
4	CRC 459	CRC 459 PLS 1	CRC 459 PT1	43	Female	Caucasian	Colorectum	II	Adenocarcinoma	5	9.81	0.3
5	CRC 460	CRC 460 PLS 1	CRC 460 PT1	72	Male	Caucasian	Colorectum	II	Adenocarcinoma	5	16.33	0.9
6	CRC 461	CRC 461 PLS 1	CRC 461 PT1	70	Male	Caucasian	Colorectum	I	Adenocarcinoma	7.5	8.93	0.3

2. Removed unwanted characters such as asterisks (\*), carets (^), white spaces, and commas from the dataset.

```
df = merged_df.applymap(lambda x: str(x).replace('*', '') if isinstance(x, str) else x)
df.head()
```

### 3.Dropped irrelevant columns that were not useful for analysis.

```
keywords = [
    'tumor type_from_df2', 'tumor type_from_df3',
    'patient id #_from_df1', 'patient id #_from_df2',
    'cancerseek logistic regression score_from_df1', 'cancerseek test result_from_df1',
    'cancerseek logistic regression score_from_df2', 'cancerseek test result_from_df2',
    'ajcc stage_from_df2', 'ajcc stage_from_df1', 'histopathology', 'primary tumor sample id #'
]

columns_to_drop = [col for col in merged_df.columns if any(key in col.lower() for key in keywords)]

print("Dropping columns:")
print(columns_to_drop)

df_cleaned = df.drop(columns=columns_to_drop)

df_cleaned.head()

Dropping columns:
['Patient ID #_from_df1', 'Primary tumor sample ID #', 'AJCC Stage_from_df1', 'Histopathology', 'CancerSEEK Logistic Regression Score_from_df1', 'CancerSEEK Test Result_from_df1', 'Patient ID #_from_df2', 'Tumor type_from_df2', 'AJCC Stage_from_df2', 'CancerSEEK Logistic Regression Score_from_df2', 'CancerSEEK Test Result_from_df2']
```

### 4. Filtered the dataset to keep only Gastric, Colorectal, and Normal samples

Sample_ID	Age	Sex	Race	Tumor type_from_df1	Plasma volume (mL)	Plasma DNA concentration (ng/mL)	AFP (pg/mL)	Angiopoietin-2 (pg/mL)	AXL (pg/mL)	CA-125 (U/ml)	CA 15-3 (U/ml)	CA19-9 (U/ml)	CD44 (ng/ml)
0	CRC 455 PLS 1	60	Male	Caucasian	Colorectum	5.00	6.08	1583.450	5598.500	3621.04	5.090	19.080	16.452
1	CRC 456 PLS 1	59	Female	Caucasian	Colorectum	4.00	46.01	715.308	20936.350	2772.96	7.270	10.040	40.910
2	CRC 457 PLS 1	69	Female	Caucasian	Colorectum	4.50	6.94	4365.530	2350.930	4120.77	4.854	16.960	16.452
3	CRC 458 PLS 1	70	Female	Caucasian	Colorectum	7.50	7.15	715.308	1604.340	2029.96	5.390	8.310	16.452

## 5. Removed unnecessary descriptive rows from the end of the dataset to retain only relevant data entries.

```
# Data cleaning 3. Delete unwanted rows
df_cleaned = df_filtered.iloc[:1268]
df_cleaned.tail()
```

## 6. Remove unwanted trailing spaces from column names and string entries.

```
merged_df.columns = merged_df.columns.str.strip()
for col in merged_df.select_dtypes(include='object').columns:
    merged_df[col] = merged_df[col].str.strip()
```

## 7. Converted numeric columns from integer to float for precise calculations during analysis.

```
Skipped 'Sex' - categorical
Skipped 'Race' - categorical
Cleaned and converted 'Plasma volume (mL)' to float.
Cleaned and converted 'Plasma DNA concentration (ng/mL)' to float.
Cleaned and converted 'AFP (pg/ml)' to float.
Cleaned and converted 'Angiopoietin-2 (pg/ml)' to float.
Cleaned and converted 'AXL (pg/ml)' to float.
Cleaned and converted 'CA-125 (U/ml)' to float.
Cleaned and converted 'CA 15-3 (U/ml)' to float.
Cleaned and converted 'CA19-9 (U/ml)' to float.
Cleaned and converted 'CD44 (ne/ml)' to float.
```

## 8. Converted cancer status into binary labels: "Label" - Cancer – 1, Normal – 0

df_cleaned['Label'] = df_cleaned['Tumor type_from_df1'].apply(lambda x: 0 if str(x).strip().lower() == 'normal' else 1)															↑	↓	±	☰	✖
dase /ml	NSE (ng/ml)	OPG (ng/ml)	OPN (pg/ml)	PAR (pg/ml)	Prolactin (pg/ml)	sEGFR (pg/ml)	sFas (pg/ml)	SHBG (nM)	sHER2/sEGFR2/sErbB2 (pg/ml)	sPECAM-1 (pg/ml)	TGF $\alpha$ (pg/ml)	Thrombospondin-2 (pg/ml)	TIMP-1 (pg/ml)	TIMP-2 (pg/ml)	Label				
6.49	25.08	0.28	8913.64	11574.68	9851.43	4677.93	1072.020	32.15	6824.90	9314.25	15.258	4831.59	79549.74	42248.26	0				
9.63	28.46	0.26	20352.33	4964.90	7244.91	5158.71	192.948	22.91	4776.64	3605.28	15.258	3678.90	59497.63	32619.49	0				

## 9. Checked for missing values in the dataset after preprocessing.

```
import pandas as pd
df = pd.read_csv("draft_dataset4.csv")
missing_counts = df.isnull().sum()
missing_columns = missing_counts[missing_counts > 0]
if missing_columns.empty:
    print("No missing values in the dataset.")
else:
    print("Columns with missing values:")
    print(missing_columns.sort_values(ascending=False))
```

Columns with missing values:

AXL (pg/ml)	6
CD44 (ng/ml)	6
G-CSF (pg/ml)	6
Kallikrein-6 (pg/ml)	6
Mesothelin (ng/ml)	6
Midkine (pg/ml)	6
PAR (pg/ml)	6
sEGFR (pg/ml)	6
sHER2/sEGFR2/sErbB2 (pg/ml)	6
sPECAM-1 (pg/ml)	6
Thrombospondin-2 (pg/ml)	6
sFas (pg/ml)	1

## 10. Dropped rows with missing values since they represented less than 10% of the data.

```
df_cleaned = df.dropna()

print("Remaining rows:", df_cleaned.shape[0])
```

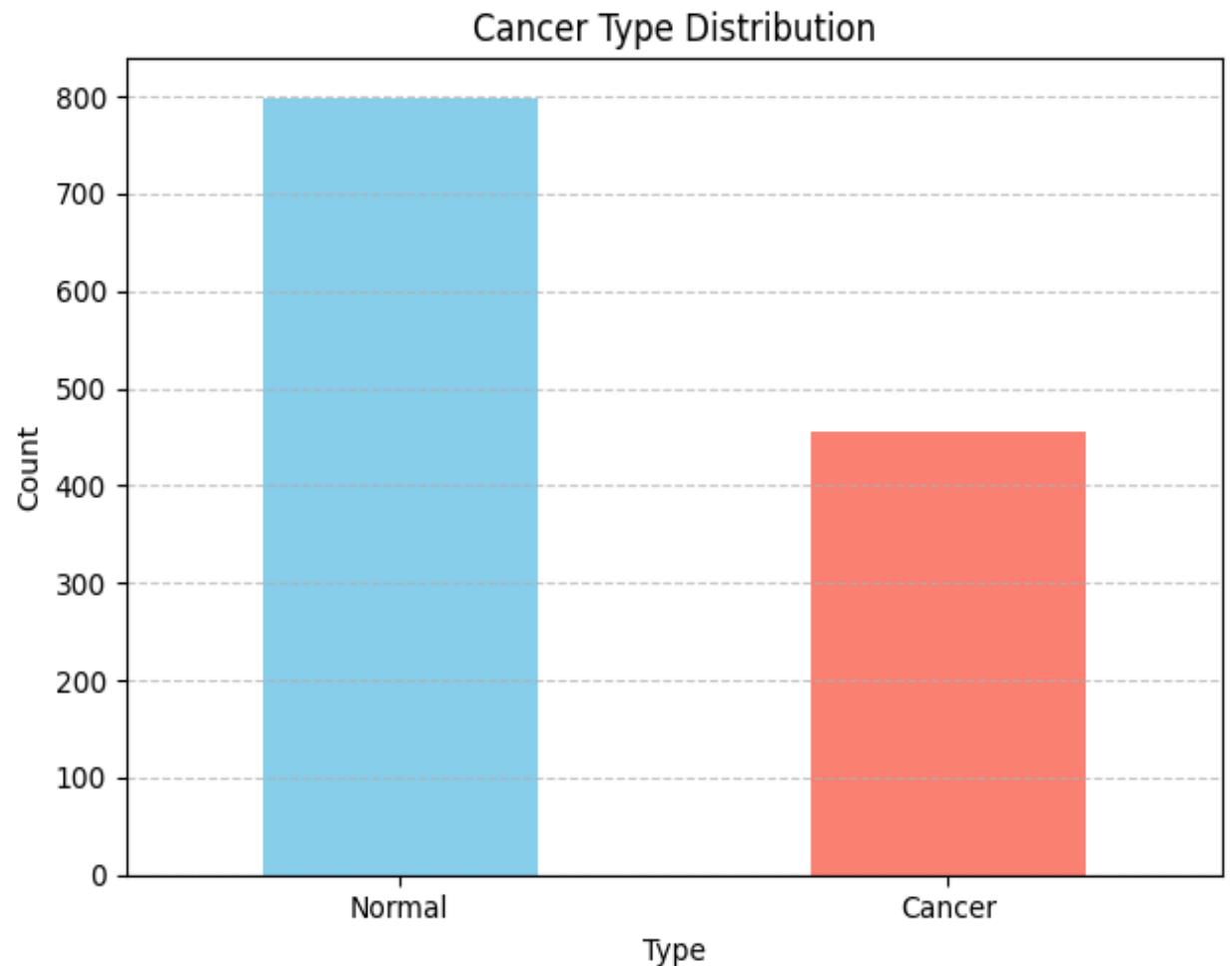
Remaining rows: 1255

# Data visualization and analysis



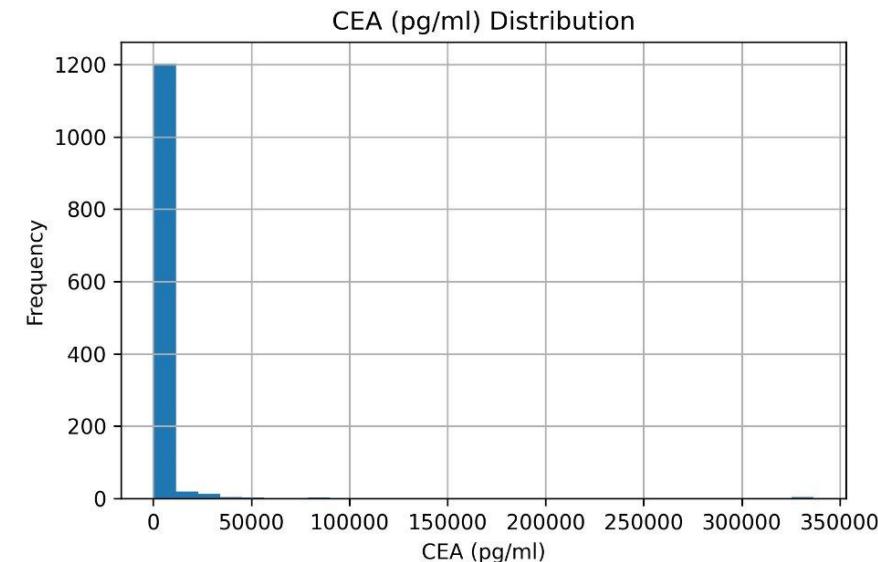
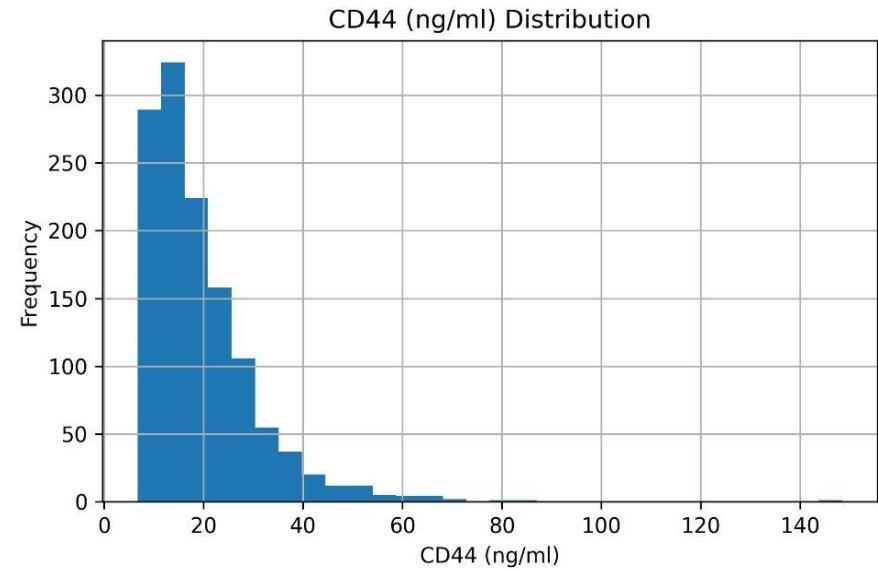
# Class distribution – Normal vs Cancer

The graph shows a clear class imbalance with 812 Normal samples and 453 Cancer samples in the dataset. N=1255



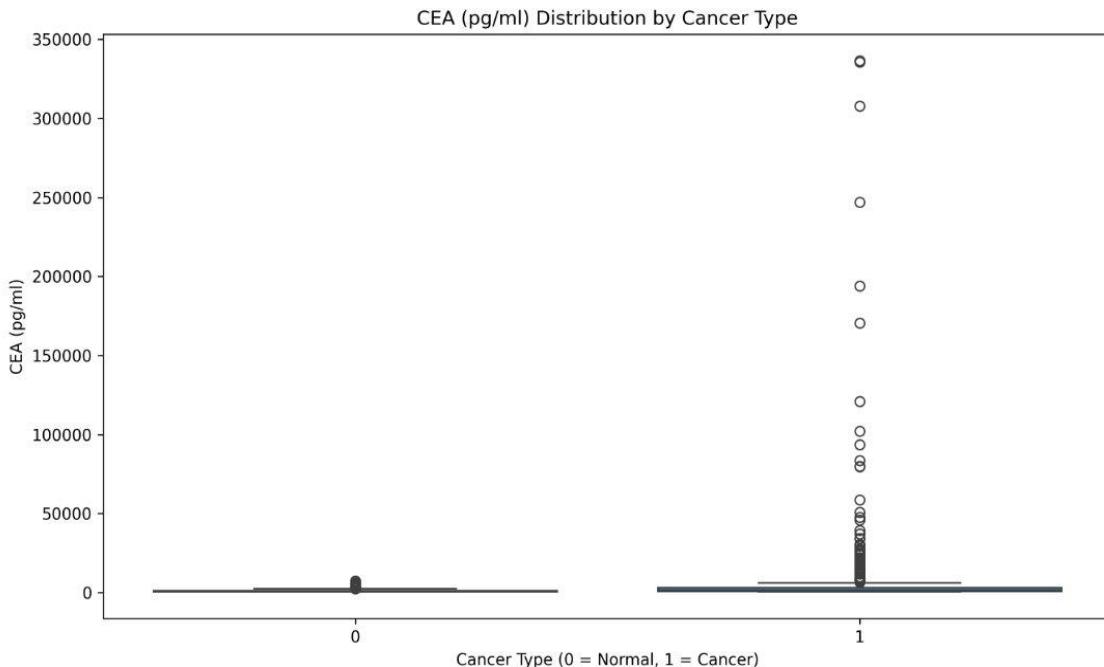
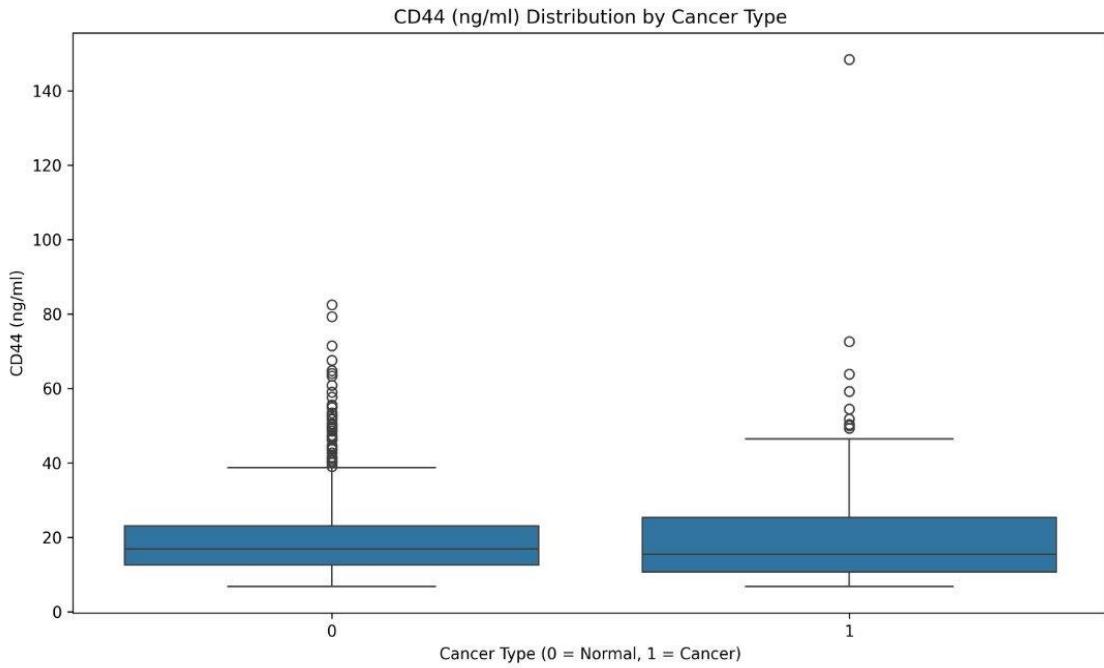
# Biomarkers distribution

The distributions of biomarkers are skewed, with most samples showing lower expression levels and a few showing very high values.



# Box plots of biomarkers to visualize outliers

Box plot of biomarkers exhibit a high number of outliers, especially in cancer cases, reflecting biological variability and heterogeneity among patients.



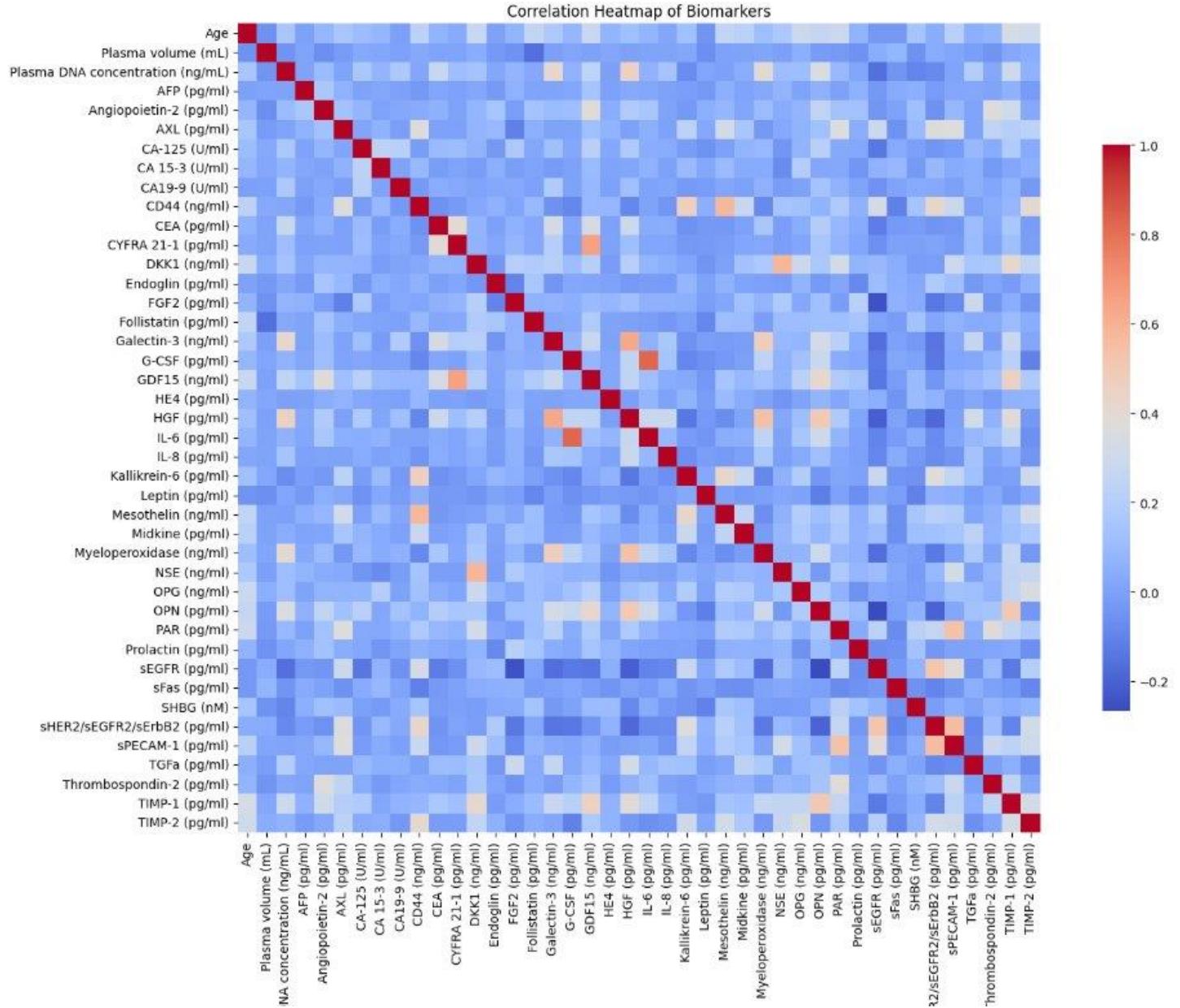
## Outliers' summary:

- Several biomarkers, including CEA, AFP, and Thrombospondin-2, exhibited extreme outliers in both normal and GI cancer groups, which can distort statistical analyses and model performance.
- To address this, we applied standard clinical thresholds to convert continuous biomarker values into categorical flags indicating abnormality.

Biomarker	Label	Num Outliers	Outlier Min	Outlier Max
CEA (pg/ml)	Normal	39	2575.2	7377.64
CEA (pg/ml)	GI Cancer	69	6637.5	336427.99
AFP (pg/ml)	Normal	85	2811.58	171255.75
AFP (pg/ml)	GI Cancer	66	3066.13	592312.72
CA-125 (U/ml)	Normal	167	5.24	33.07
CA-125 (U/ml)	GI Cancer	70	9.88	111.49
CD44 (ng/ml)	Normal	45	38.62	82.53
CD44 (ng/ml)	GI Cancer	9	49.31	148.44
GDF15 (ng/ml)	Normal	56	1.08	8.56
GDF15 (ng/ml)	GI Cancer	39	2.27	24.29
IL-6 (pg/ml)	Normal	156	6.9	356.64
IL-6 (pg/ml)	GI Cancer	61	92.08	2818.46
CA 15-3 (U/ml)	Normal	32	36.17	102.98
CA 15-3 (U/ml)	GI Cancer	21	29.75	336.25
CYFRA 21-1 (pg/ml)	Normal	108	1816.46	13499.0
CYFRA 21-1 (pg/ml)	GI Cancer	66	4541.87	722034.0
Thrombospondin-2 (pg/ml)	Normal	76	10756.08	131014.81
Thrombospondin-2 (pg/ml)	GI Cancer	52	12973.51	157461.07

# Correlation heatmap of biomarkers

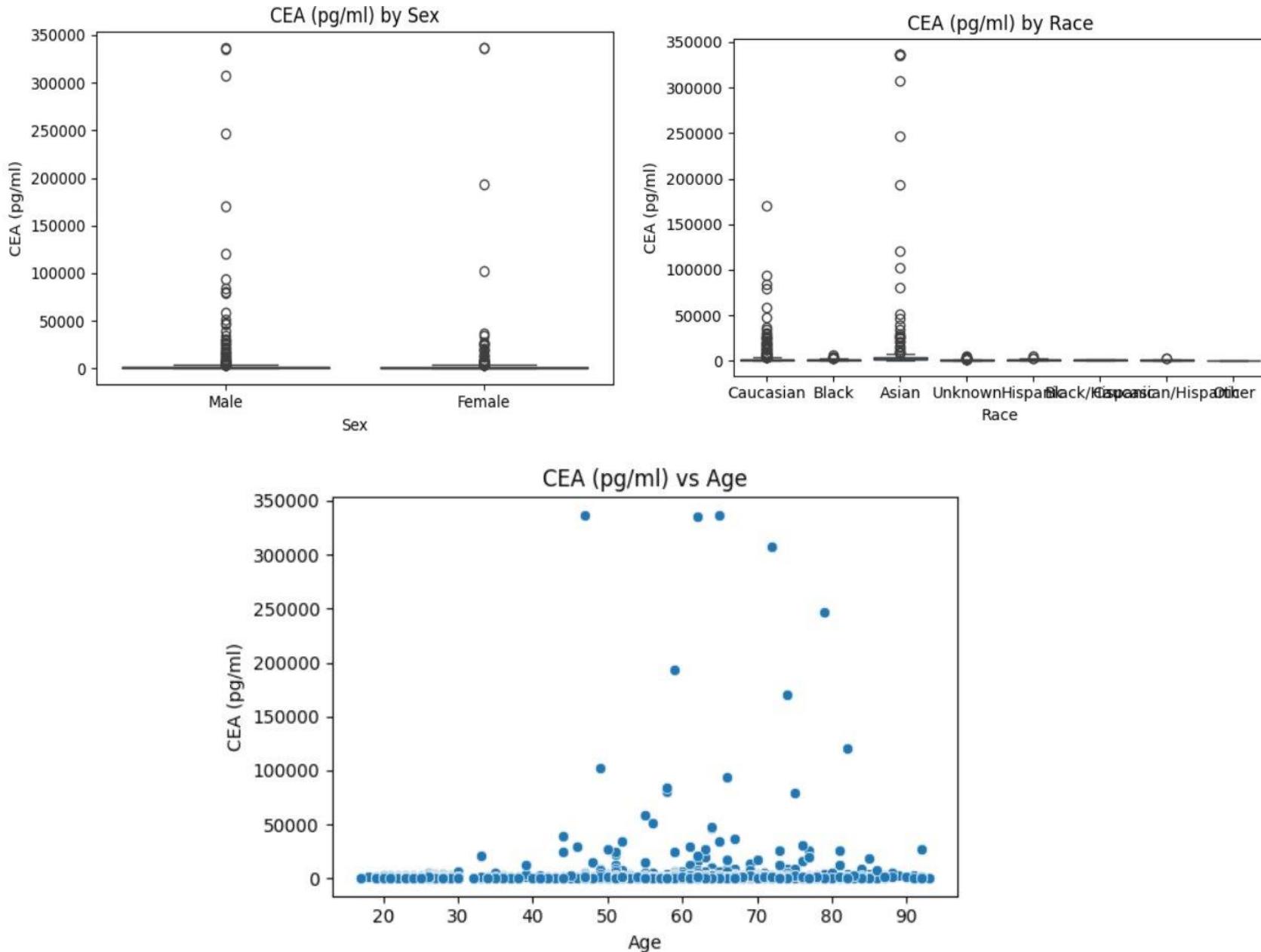
Most biomarkers show weak correlations with each other, indicating they likely provide independent information. This low multicollinearity supports using a diverse set of features in multivariate models for GI cancer detection.



# Demographics trend analysis

Most biomarkers, exhibit wide variability across sex, race, and age groups, with no strong or consistent demographic trend.

This suggests that while demographic factors may influence biomarker levels in specific cases but in this dataset, no clear trends were observed which is attributed due to the limited sample size



# Normality test summary

None of the key biomarkers in this dataset follow a normal distribution, as indicated by the Shapiro-Wilk test.

As a result, the next step involves applying clinical thresholds to convert continuous biomarker values into categorical flags indicating abnormality.

```
from scipy.stats import shapiro
import pandas as pd

normality_results = []

for col in key_biomarkers:
    stat, p_value = shapiro(df_cleaned[col].dropna())
    normality_results.append({
        'Biomarker': col,
        'Shapiro-Wilk p-value': round(p_value, 5),
        'Normality': 'Not Normal' if p_value < 0.05 else 'Normal'
    })

normality_df = pd.DataFrame(normality_results)
print(normality_df)
normality_df.to_csv("Normality_test_results.csv", index=False)
```

	Biomarker	Shapiro-Wilk p-value	Normality
0	CEA (pg/ml)	0.0	Not Normal
1	AFP (pg/ml)	0.0	Not Normal
2	CA-125 (U/ml)	0.0	Not Normal
3	CD44 (ng/ml)	0.0	Not Normal
4	GDF15 (ng/ml)	0.0	Not Normal
5	IL-6 (pg/ml)	0.0	Not Normal
6	CA 15-3 (U/ml)	0.0	Not Normal
7	CYFRA 21-1 (pg/ml)	0.0	Not Normal
8	Thrombospondin-2 (pg/ml)	0.0	Not Normal

# Feature engineering and Threshold Flagging

Clinical thresholds gathered from literature for 34 biomarkers and applied known thresholds to flag values (0 – Normal (low), 1 – Abnormal (elevated))

Handled sex-specific biomarkers by applying variable thresholds : Leptin, Prolactin and SHBG

Few biomarkers with no universal thresholds were kept as numerical : CD44, PAR, TIMP-2, sEGFR and sPECAM-1

```
import pandas as pd

df_cleaned = pd.read_csv("Dropped_rows_dataset.csv")
clinical_thresholds = {
    'AFP (pg/ml)': 20000,
    'Angiopoietin-2 (pg/ml)': 4141,
    'AXL (pg/ml)': 1243,
    'CA-125 (U/ml)': 35,
    'CA 15-3 (U/ml)': 30,
    'CA19-9 (U/ml)': 37,
    'CYFRA 21-1 (pg/ml)': 3500,
    'CEA (pg/ml)': 5000,
    'DKK1 (ng/ml)': 2.03 * 1000, # ng/ml to pg/ml
    'Endoglin (pg/ml)': 2000,
    'FGF2 (pg/ml)': 10,
    'Follistatin (pg/ml)': 4700,
    'Galectin-3 (ng/ml)': 22.1 * 1000,
    'G-CSF (pg/ml)': 15,
    'GDF15 (ng/ml)': 1200 * 1000,
    'HE4 (pg/ml)': 3500,
    'HGF (pg/ml)': 1000,
    'IL-6 (pg/ml)': 5,
    'IL-8 (pg/ml)': 57.7,
    'Kallikrein-6 (pg/ml)': 1000,
    'Mesothelin (ng/ml)': 2 * 1000,
    'Midkine (pg/ml)': 400,
    'Myeloperoxidase (ng/ml)': 220 * 1000,
    'NSE (ng/ml)': 16.3 * 1000,
    'OPG (ng/ml)': 0.2 * 1000,
    'OPN (pg/ml)': 200000,
    'sFas (pg/ml)': 150,
    'sHER2/sEGFR2/srbB2 (pg/ml)': 15000,
    'TGFa (pg/ml)': 100,
    'Thrombospondin-2 (pg/ml)': 45000,
    'TIMP-1 (pg/ml)': 126000,
}

#Sex-specific thresholds
sex_specific_thresholds = {
    'Leptin (pg/ml)': {'Male': 15200, 'Female': 12500},
    'Prolactin (pg/ml)': {'Male': 18000, 'Female': 25000},
    'SHBG (nM)': {'Male': 54, 'Female': 145},
}

#Convert biomarker columns to numeric
for col in df_cleaned.columns:
    df[col] = pd.to_numeric(df_cleaned[col], errors='coerce')

#Clinical threshold flags
for biomarker, threshold in clinical_thresholds.items():
    if biomarker in df_cleaned.columns:
        df_cleaned[f'{biomarker}_flag'] = (df_cleaned[biomarker] > threshold).astype(int)

#sex-specific thresholds
for biomarker, thresholds in sex_specific_thresholds.items():
    if biomarker in df_cleaned.columns and 'Sex' in df_cleaned.columns:
        df_cleaned[f'{biomarker}_flag'] = df_cleaned.apply(
            lambda row: int(row[biomarker] > thresholds.get(row['Sex'], float('inf'))), axis=1
        )

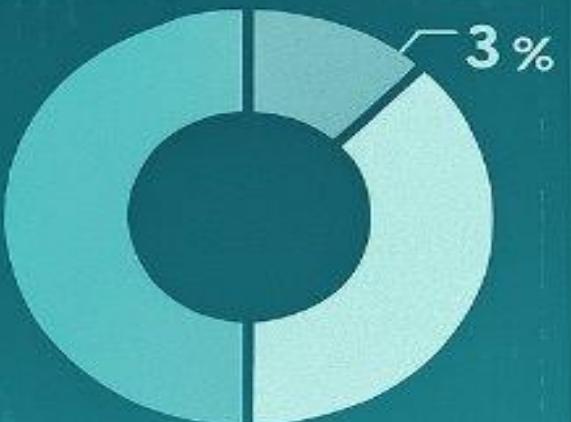
#Remove any mistakenly flagged non-biomarker
non_biomarker_flags = ['Sex_flag', 'Sample_ID_flag']
df_cleaned.drop(columns=[col for col in non_biomarker_flags if col in df_cleaned.columns], inplace=True)

df_cleaned.to_csv("threshold_flagged_dataset_final.csv", index=False)
df_cleaned.head()
```

# Threshold flagging:

# STATISTICAL TESTING

COTAGENCY  
TABLE



CONTAGENCY

CHI-SQUARE  
CONTIGENCY TABLE



CHI-SQUARE  
P-VALUE



STATISTICAL  
TESTING

# Statistical comparison by cancer status

We performed a statistical comparison of biomarker levels between normal and cancer groups, which revealed that several biomarkers show significant mean differences. This suggests their potential relevance in distinguishing cancer status.

The t-test helps in prioritizing features for further modeling based on their statistical relevance.

```
# 5. Statistical comparison between Normal and Cancer
from scipy.stats import ttest_ind

stats_summary = []

for col in biomarkers:
    group0 = df_cleaned[df_cleaned['Label'] == 0][col].dropna()
    group1 = df_cleaned[df_cleaned['Label'] == 1][col].dropna()

    stat, p_value = ttest_ind(group0, group1, equal_var=False)
    stats_summary.append({
        'Biomarker': col,
        'Mean (Normal)': round(group0.mean(), 2),
        'Mean (Cancer)': round(group1.mean(), 2),
        'p-value': round(p_value, 5)
    })

# Display the summary table
import pandas as pd
stats_df = pd.DataFrame(stats_summary)
print(stats_df)
```

	Biomarker	Mean (Normal)	Mean (Cancer)	p-value
0	CEA (pg/ml)	1094.01	10010.17	0.00000
1	AFP (pg/ml)	1676.21	4616.65	0.07017
2	CA-125 (U/ml)	5.52	8.58	0.00000
3	CD44 (ng/ml)	19.46	19.08	0.57499
4	GDF15 (ng/ml)	0.44	1.02	0.00000
5	IL-6 (pg/ml)	8.05	47.93	0.00000
6	CA 15-3 (U/ml)	13.98	13.07	0.38371
7	CYFRA 21-1 (pg/ml)	2069.59	5120.42	0.06132
8	Thrombospondin-2 (pg/ml)	4179.51	5143.08	0.06015

# Statistical Analysis

- **Categorical data:** Flagged biomarkers
- **Method:** Chi-Square Test was applied to categorical biomarker variables
- **Result:** A majority of biomarker flags show statistically significant differences ( $p < 0.05$ ) between cancer and normal groups. This highlights their potential utility in classification and feature selection for predictive modeling. Non-significant ones retained for biological relevance.

Biomarker	p_value	Significant
IL-6 (pg/ml)_flag	0.0	Yes
Leptin (pg/ml)_flag	0.0	Yes
TIMP-1 (pg/ml)_flag	0.0	Yes
OPN (pg/ml)_flag	0.0	Yes
CA19-9 (U/ml)_flag	0.0	Yes
CYFRA 21-1 (pg/ml)_flag	0.0	Yes
CEA (pg/ml)_flag	0.0	Yes
Midkine (pg/ml)_flag	0.0	Yes
IL-8 (pg/ml)_flag	0.0	Yes
HGF (pg/ml)_flag	0.0	Yes
Prolactin (pg/ml)_flag	0.0	Yes
Gnghiopoietin-2 (pg/ml)_flag	1e-05	Yes
SHBG (nM)_flag	2e-05	Yes
CA-125 (U/ml)_flag	2e-05	Yes
AFP (pg/ml)_flag	0.00886	Yes
Kallikrein-6 (pg/ml)_flag	0.02145	Yes
CA 15-3 (U/ml)_flag	0.05517	No
TGF $\alpha$ (pg/ml)_flag	0.09993	No
AXL (pg/ml)_flag	0.14044	No
Endoglin (pg/ml)_flag	0.8987	No
SFR2/sErbB2 (pg/ml)_flag	1.0	No
Abbospondin-2 (pg/ml)_flag	1.0	No
Follistatin (pg/ml)_flag	1.0	No

# Statistical Analysis – Continuous Biomarkers

- **Method:** Mann-Whitney U Test
- **Biomarkers Tested:** CD44, PAR, sEGFR, TIMP-2, sPECAM-1

Most numerical features showed statistically significant differences between cancer and normal groups, indicating strong discriminatory potential.

	Biomarker	Mann-Whitney U	p-value	Significant
2	sEGFR (pg/ml)	248907.0	0.00000	Yes
1	PAR (pg/ml)	159440.5	0.00149	Yes
4	TIMP-2 (pg/ml)	197661.0	0.00187	Yes
0	CD44 (ng/ml)	191261.5	0.03966	Yes
3	sPECAM-1 (pg/ml)	187833.5	0.13531	No

# Discussions and conclusion

**NUL HYPOTHESIS ( $H_0$ ):** There is no statistically significant difference in biomarker values between GI cancer and normal patients.

- Statistical hypothesis testing was conducted to determine whether there are significant differences in biomarker levels between GI cancer and normal patients. Based on p-values from multiple tests (all  $p < 0.05$ ), the **null hypothesis** – stating that there is no difference – was **rejected** for all listed features.
- Angiopoietin-2 p-value= 0.00001, reject null hypothesis ( $H_0$ )
- SHBG p-value= 0.00002, reject null hypothesis ( $H_0$ )
- CA-125 p-value= 0.00002, reject null hypothesis ( $H_0$ )
- AFP p-value= 0.00886 reject null hypothesis ( $H_0$ )
- Kalikrein-6 p-value= 0.02145 reject null hypothesis ( $H_0$ )
- PAR p-value= 0.00149, reject null hypothesis ( $H_0$ )
- TIMP2 p-value=0.00187 reject null hypothesis ( $H_0$ )
- CD4 p-value =0.03966, reject null hypothesis ( $H_0$ )

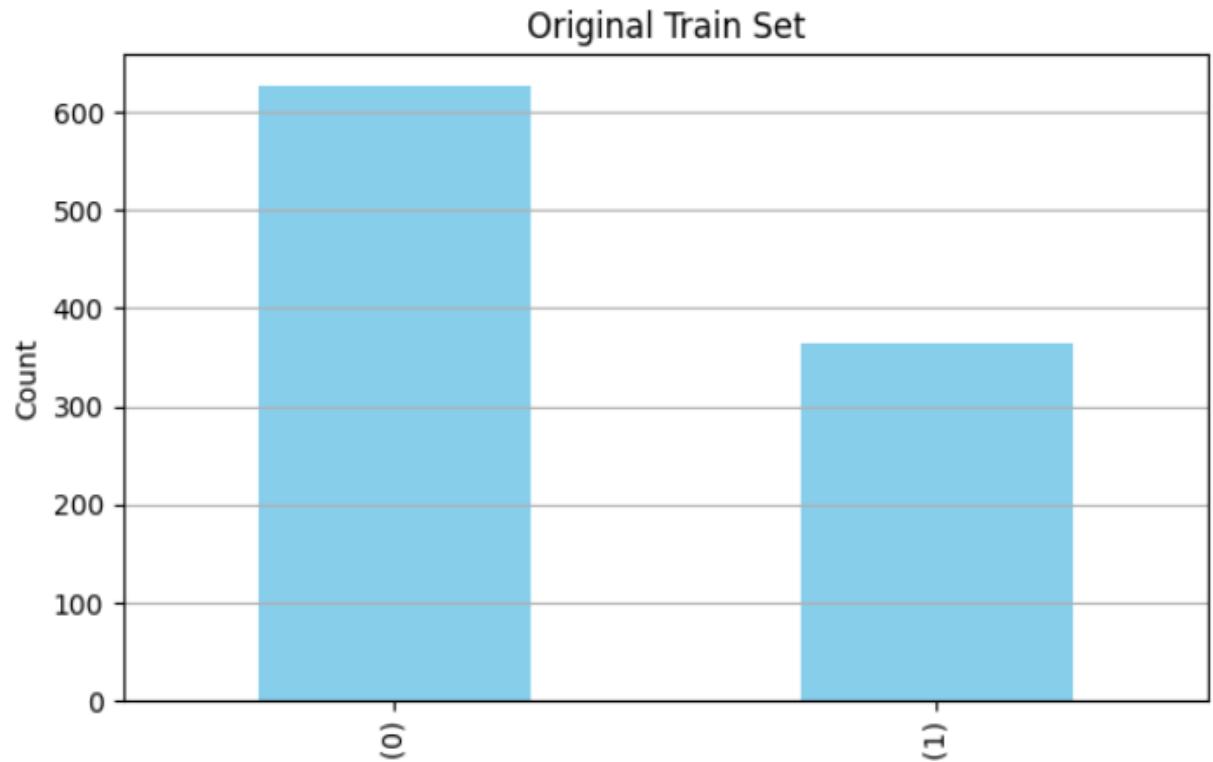
**Conclusion:** These features exhibit statistically significant differences between groups serve as important variables for cancer classification or prediction models.

# Machine learning

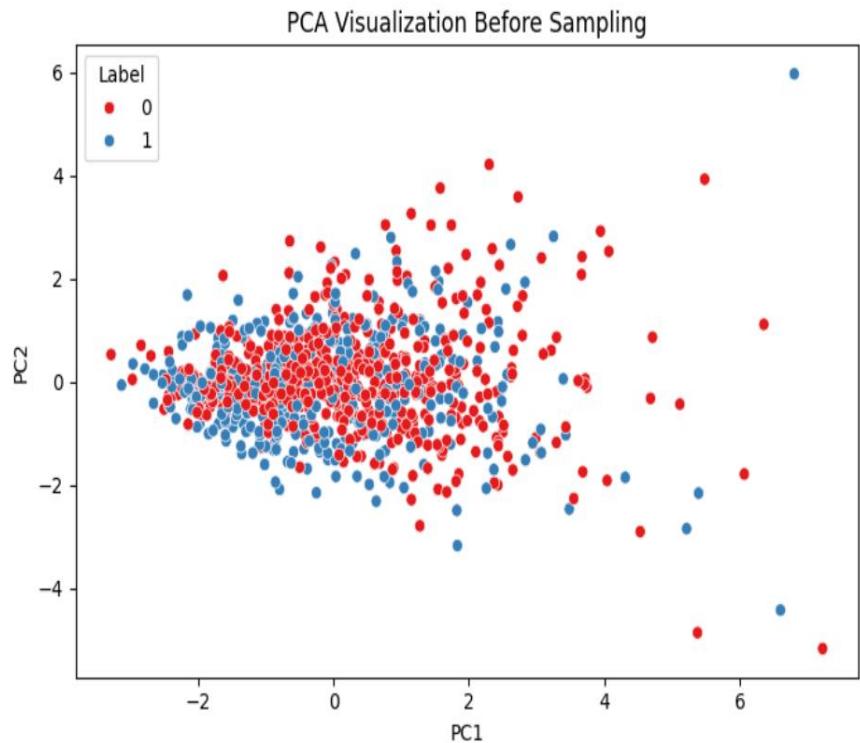


# Class Imbalance & Sampling

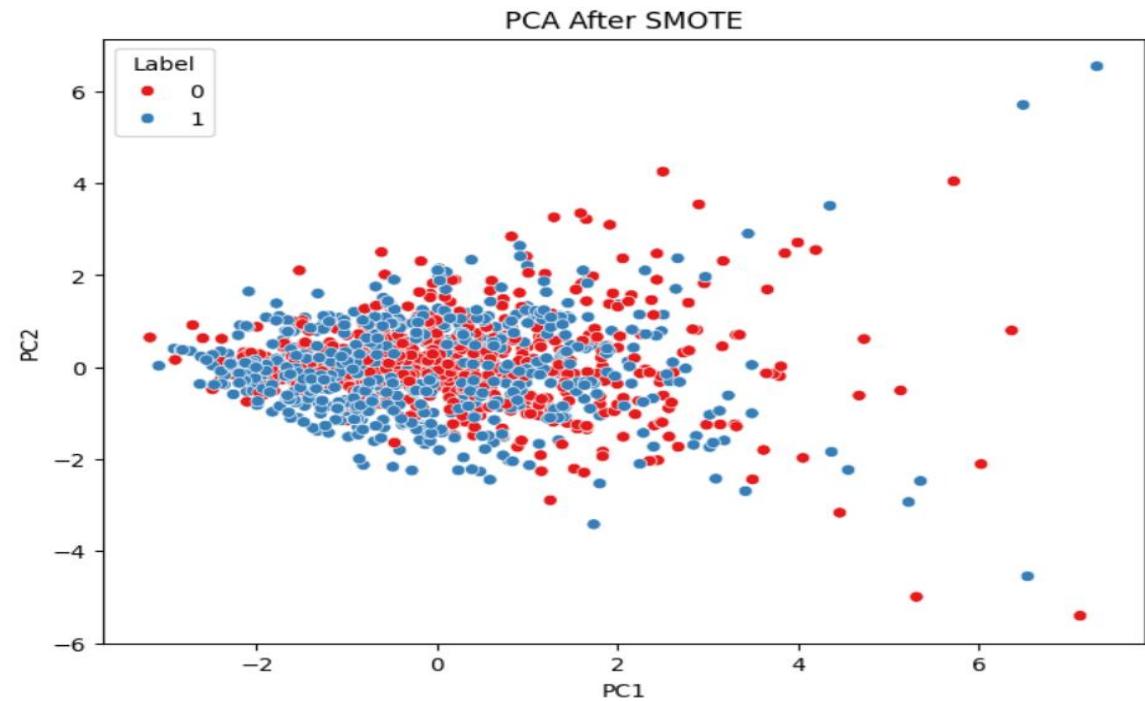
- Initial imbalance: 455 cancer vs. 800 non-cancer (Normal) cases
- Applied techniques:
  - SMOTE to generate synthetic cancer samples
  - Random Oversampling /Undersampling for balance
  - Bootstrapping to improve training diversity



# Sampling balance visualization - PCA scatter plot:

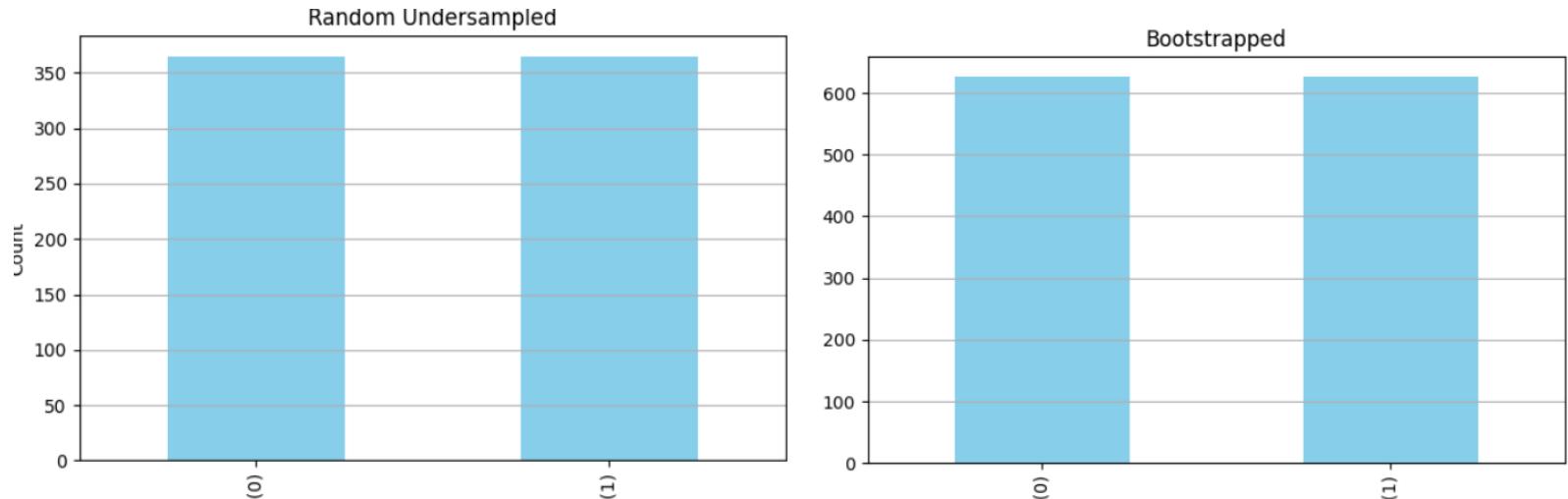
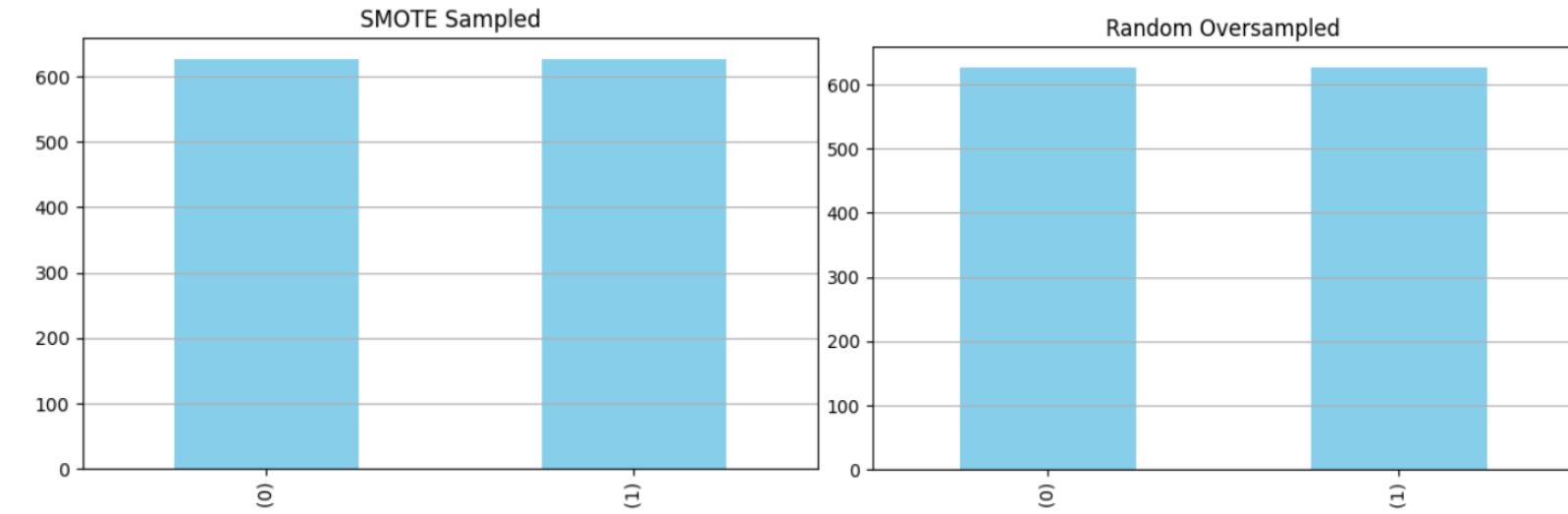
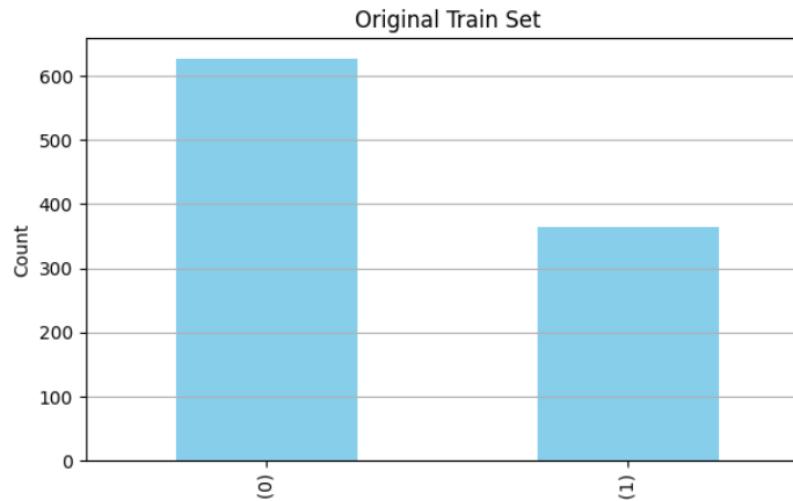


Before SMOTE: Data is imbalanced; the model favors the majority class (red)



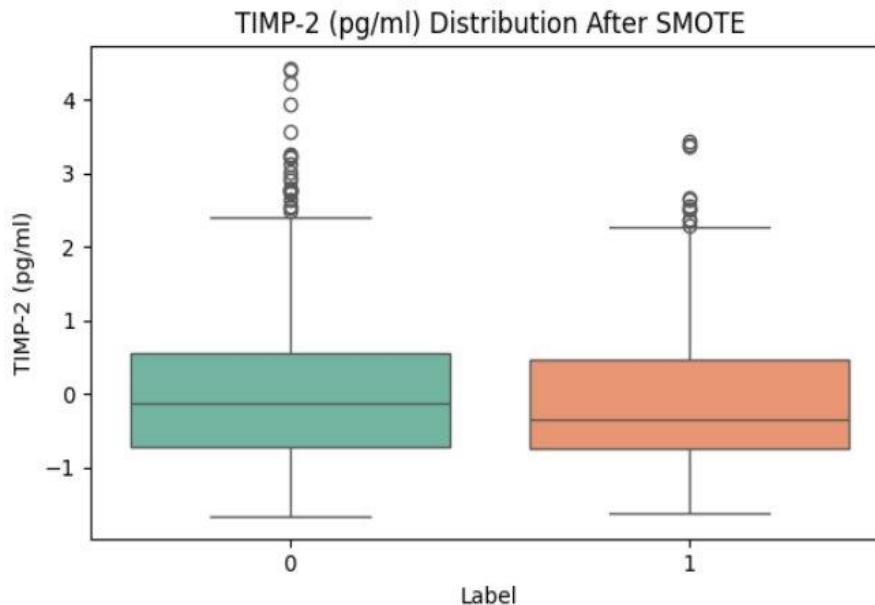
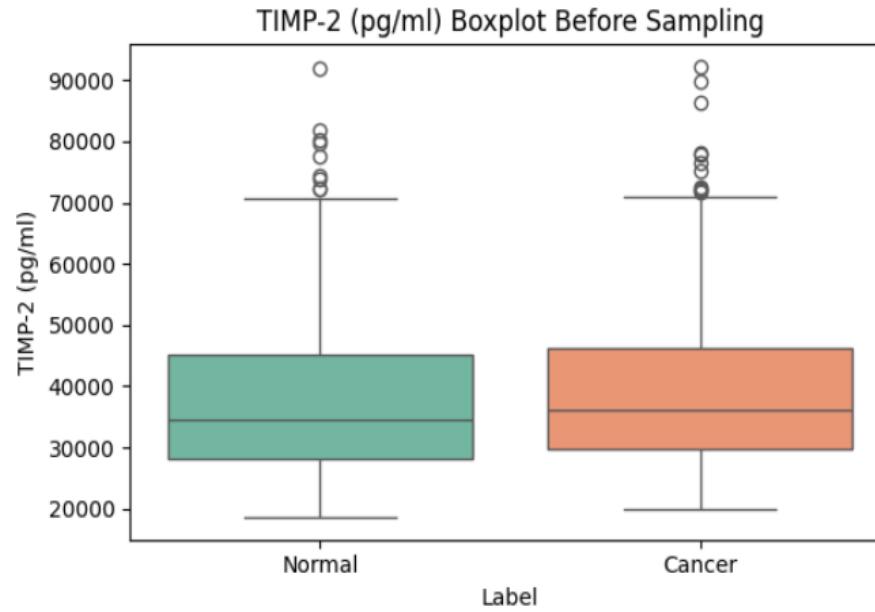
After SMOTE: Classes are balanced; model learns both classes more effectively, improving fairness and recall (balance between red and blue).

# Sampling balance visualization – Data distribution



# Sampling balance visualization – Box plots:

- Before sampling, the distribution of TIMP-2 values was imbalanced between the normal and cancer groups, with visible differences in spread and medians.
- After applying SMOTE (Synthetic Minority Over-sampling Technique), the sample size was balanced, leading to more symmetrical box plots for both classes.
- The SMOTE-adjusted distribution helps ensure that the model is trained on balanced data, improving fairness and reducing bias during classification.



# Model building - Overview



## Random Forest (RF):

An ensemble learning method that combines multiple decision trees to improve prediction accuracy and control overfitting.

Why? Classification problem

Models Used:

Random Forest and logistic regression



## Logistic Regression (LR):

A linear model used for binary classification, useful for interpretability and identifying relationships between biomarkers and cancer outcome.



## Why These Models?

Both models perform well on imbalanced biomedical data and allow for feature importance interpretation.

# Random forest

- Suitable for binary classification, such as predicting GI cancer presence.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

#SMOTE
rf_model.fit(X_train_smote, y_train_smote)
y_pred_smote = rf_model.predict(X_test_scaled)
print("SMOTE")
print(classification_report(y_test, y_pred_smote))

#Random Oversampling
rf_model.fit(X_train_ros, y_train_ros)
y_pred_ros = rf_model.predict(X_test_scaled)
print("Random Oversampling")
print(classification_report(y_test, y_pred_ros))

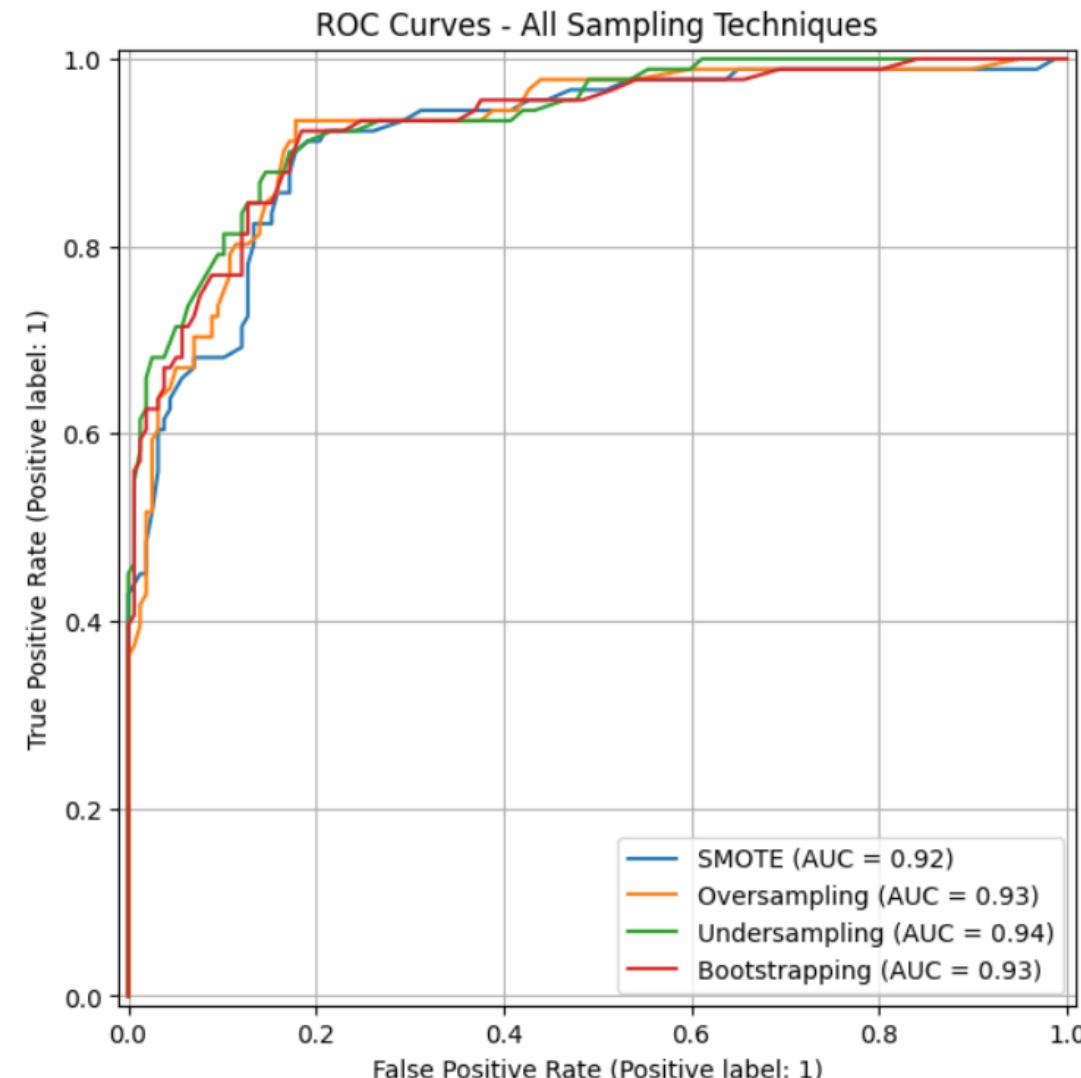
#Random Undersampling
rf_model.fit(X_train_rus, y_train_rus)
y_pred_rus = rf_model.predict(X_test_scaled)
print("Random Undersampling")
print(classification_report(y_test, y_pred_rus))

#Bootstrapping
rf_model.fit(X_train_bootstrap, y_train_bootstrap)
y_pred_boot = rf_model.predict(X_test_scaled)
print("Bootstrapping")
print(classification_report(y_test, y_pred_boot))
```

# Random Forest overview and results:

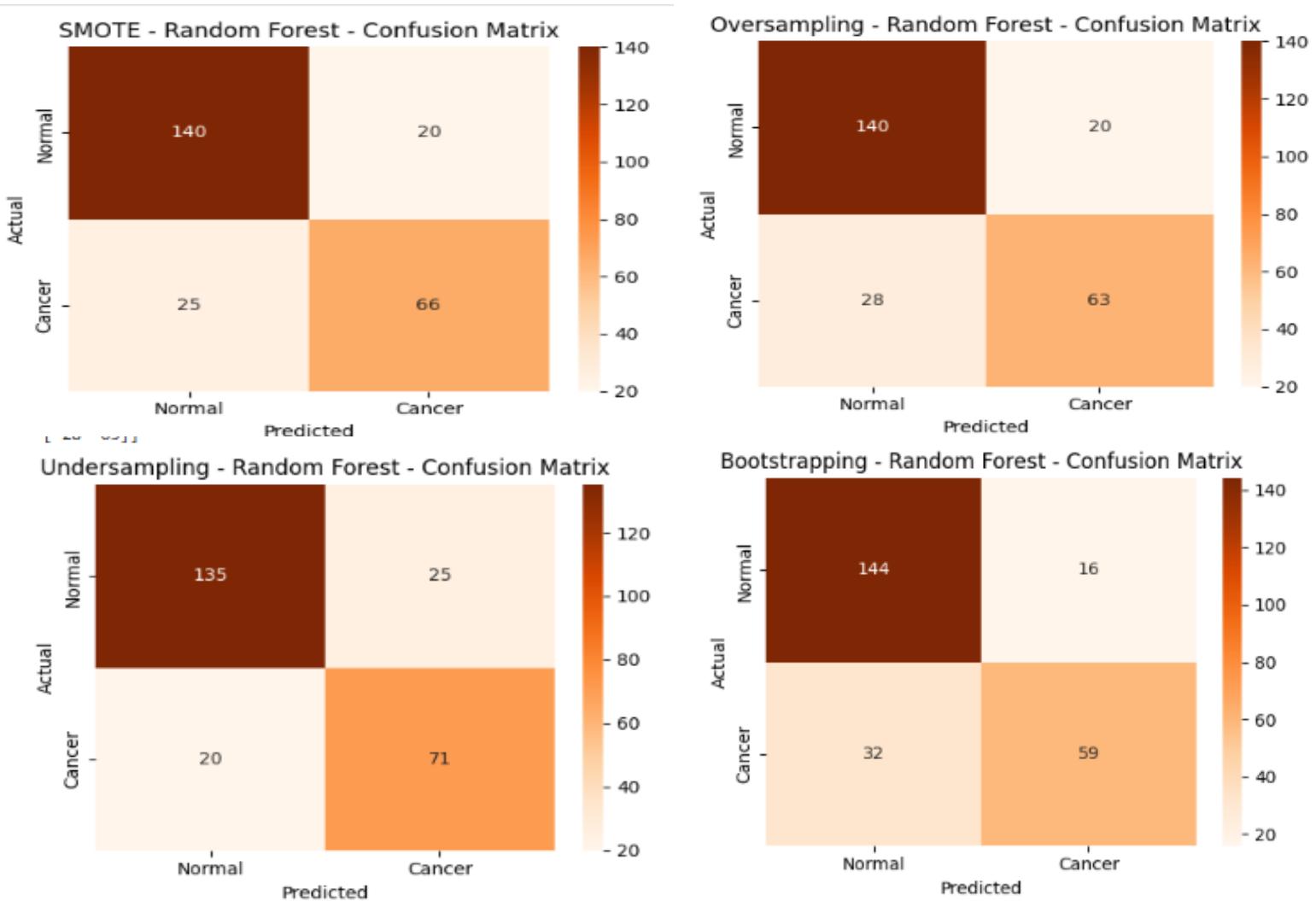
- SMOTE AUC (0.92): 92% chance of correctly ranking positive cases higher.
- Undersampling AUC (0.94): 94% chance, slightly better performance.
- Oversampling AUC (0.93) and Bootstrapping AUC (0.93): Strong but slightly lower than undersampling.

SMOTE					
	precision	recall	f1-score	support	
0	0.90	0.85	0.87	157	
1	0.76	0.84	0.80	91	
accuracy			0.84	248	
macro avg	0.83	0.84	0.83	248	
weighted avg	0.85	0.84	0.84	248	
Random Oversampling					
	precision	recall	f1-score	support	
0	0.88	0.87	0.88	157	
1	0.78	0.80	0.79	91	
accuracy			0.85	248	
macro avg	0.83	0.84	0.84	248	
weighted avg	0.85	0.85	0.85	248	
Random Undersampling					
	precision	recall	f1-score	support	
0	0.92	0.86	0.89	157	
1	0.78	0.87	0.82	91	
accuracy			0.86	248	
macro avg	0.85	0.86	0.86	248	
weighted avg	0.87	0.86	0.86	248	
Bootstrapping					
	precision	recall	f1-score	support	
0	0.86	0.92	0.89	157	
1	0.85	0.75	0.80	91	
accuracy			0.86	248	
macro avg	0.86	0.84	0.84	248	
weighted avg	0.86	0.86	0.86	248	



# Random Forest overview and results:

- SMOTE and Undersampling gave the best cancer detection.
- Bootstrapping had the lowest sensitivity for cancer cases.
- SMOTE offered a balanced trade-off between precision and recall.



# Logistic regression

- Ideal for binary classification tasks like predicting GI cancer.
- Offers clear interpretability of how biomarkers affect cancer risk.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

lr_model = LogisticRegression(max_iter=1000, random_state=42)

#SMOTE
lr_model.fit(X_train_smote, y_train_smote)
y_pred_smote_lr = lr_model.predict(X_test_scaled)
print("SMOTE - Logistic Regression")
print(classification_report(y_test, y_pred_smote_lr))

#Random Oversampling
lr_model.fit(X_train_ros, y_train_ros)
y_pred_ros_lr = lr_model.predict(X_test_scaled)
print("Oversampling - Logistic Regression")
print(classification_report(y_test, y_pred_ros_lr))

#Random Undersampling
lr_model.fit(X_train_rus, y_train_rus)
y_pred_rus_lr = lr_model.predict(X_test_scaled)
print("Undersampling - Logistic Regression")
print(classification_report(y_test, y_pred_rus_lr))

#Bootstrapping
lr_model.fit(X_train_bootstrap, y_train_bootstrap)
y_pred_boot_lr = lr_model.predict(X_test_scaled)
print("Bootstrapping - Logistic Regression")
print(classification_report(y_test, y_pred_boot_lr))
```

# Logistic Regression overview and results:

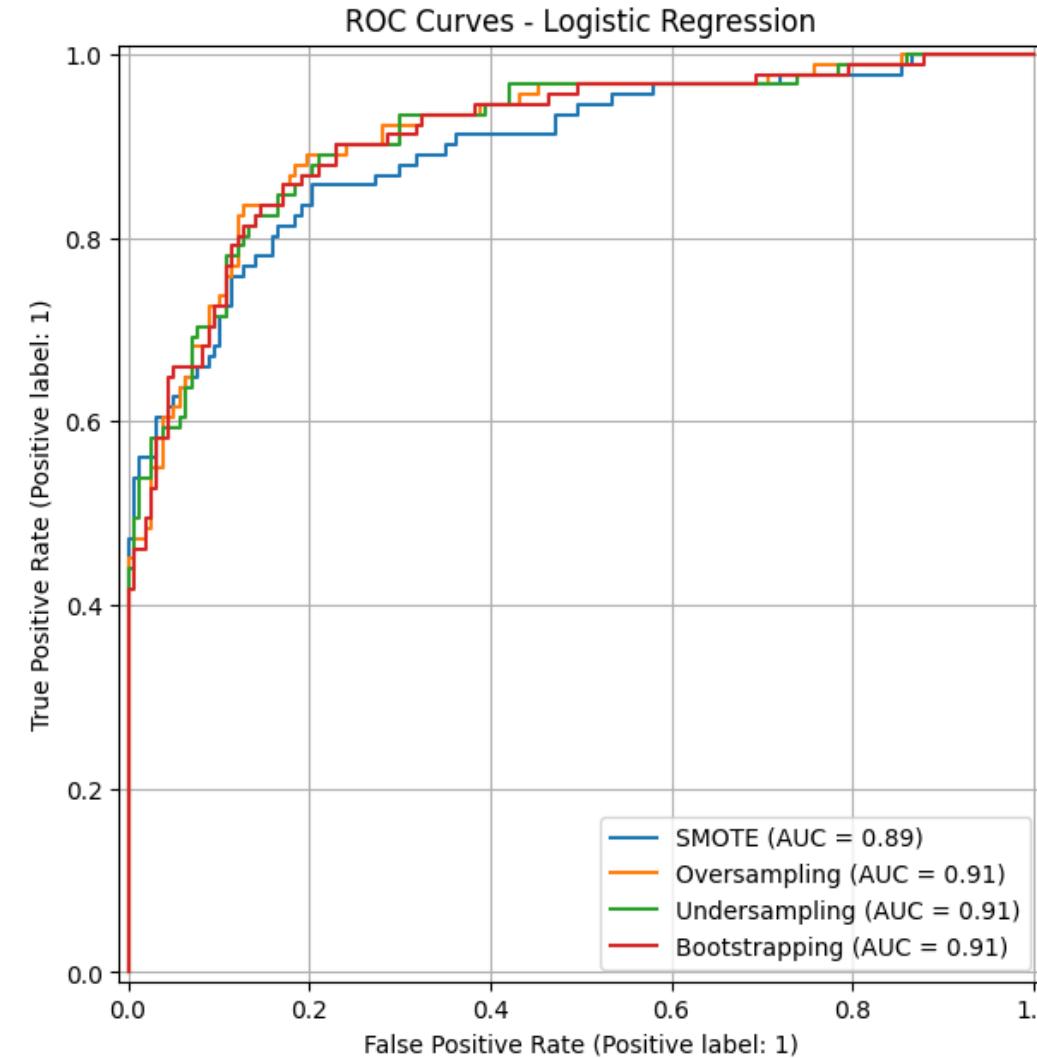
- **SMOTE AUC (0.89):** 89% chance of correctly ranking positive cases higher.
- **Oversampling AUC (0.91):** 91% chance, good model performance.
- **Undersampling AUC (0.91) and Bootstrapping AUC (0.91):** All show similar, strong results.

SMOTE - Logistic Regression				
	precision	recall	f1-score	support
0	0.87	0.84	0.86	157
1	0.74	0.79	0.77	91

Oversampling - Logistic Regression				
	precision	recall	f1-score	support
0	0.88	0.88	0.88	157
1	0.79	0.80	0.80	91
accuracy			0.82	248
macro avg	0.81	0.82	0.81	248
weighted avg	0.83	0.82	0.82	248

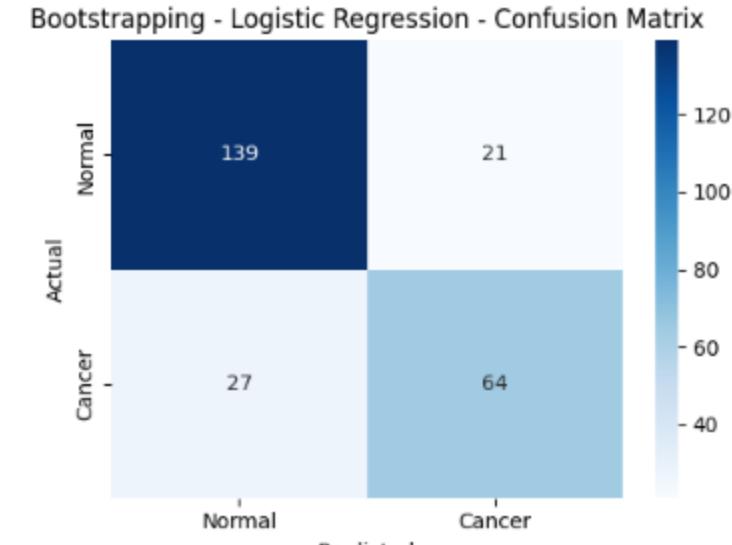
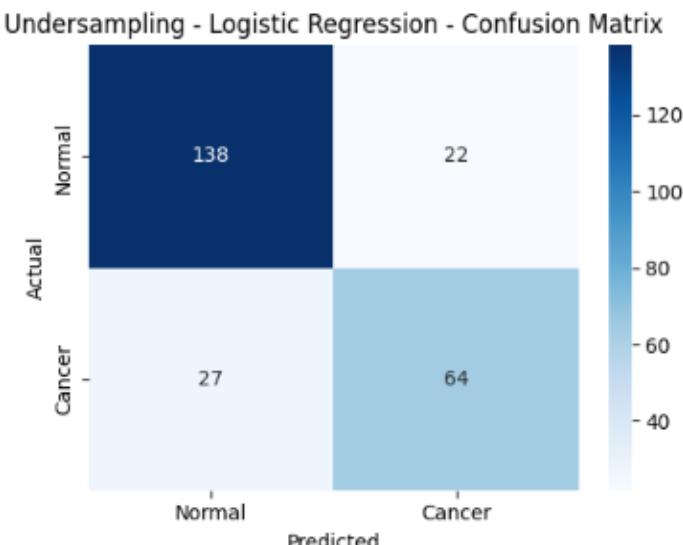
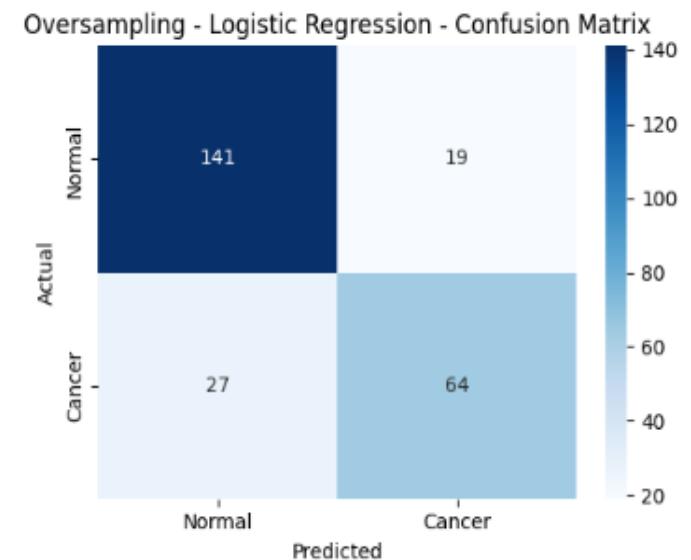
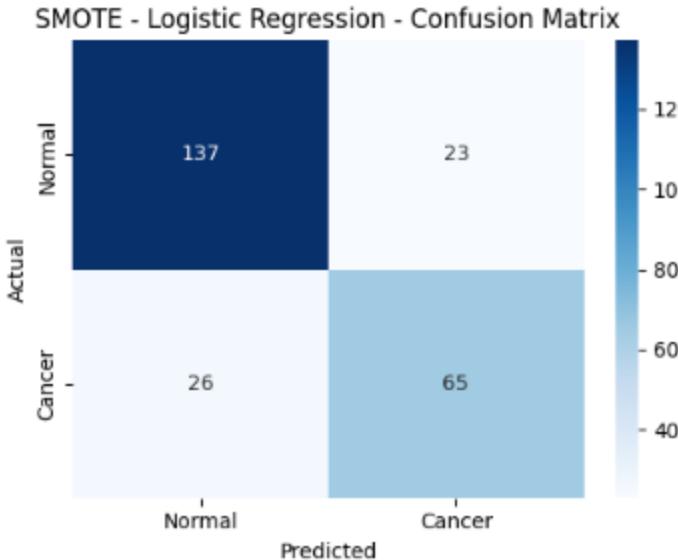
Undersampling - Logistic Regression				
	precision	recall	f1-score	support
0	0.89	0.85	0.87	157
1	0.77	0.82	0.79	91
accuracy			0.85	248
macro avg	0.83	0.84	0.83	248
weighted avg	0.85	0.84	0.85	248

Bootstrapping - Logistic Regression				
	precision	recall	f1-score	support
0	0.88	0.88	0.88	157
1	0.79	0.79	0.79	91
accuracy			0.85	248
macro avg	0.84	0.84	0.84	248
weighted avg	0.85	0.85	0.85	248



# Logistic Regression overview and results:

- All sampling methods produced similar results with minor variations.
- **SMOTE** slightly outperformed others in detecting cancer cases (65 true positives).
- **Oversampling** and **Bootstrapping** showed consistent performance but had slightly more false negatives.
- Logistic Regression remained stable across all sampling strategies.



# Evaluation Metrics Summary



ACCURACY



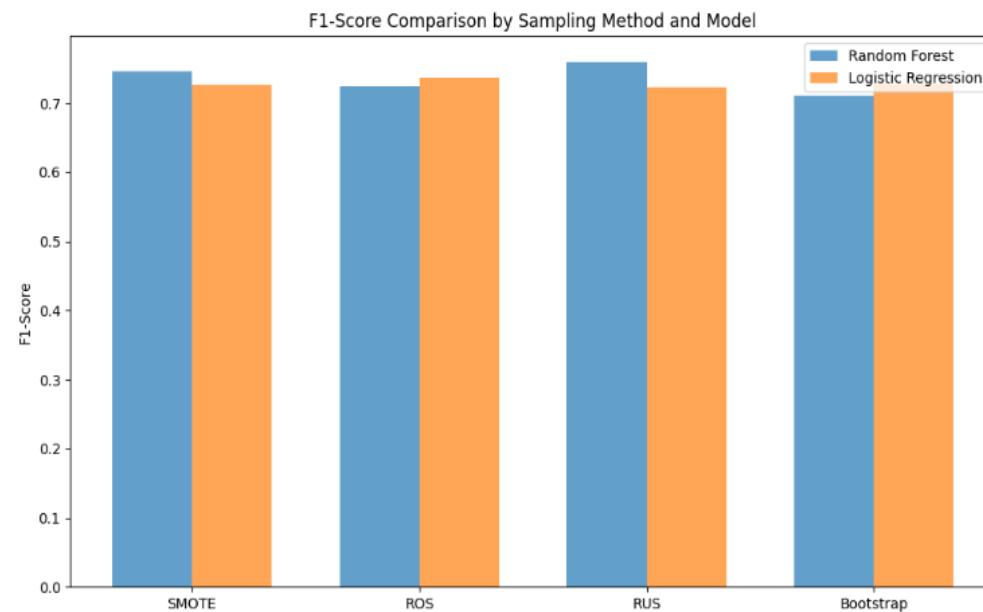
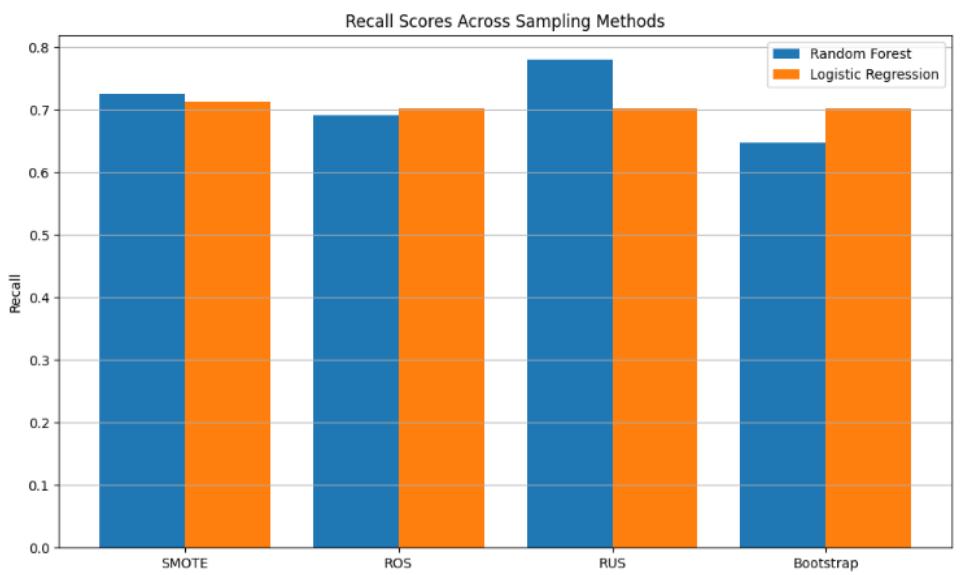
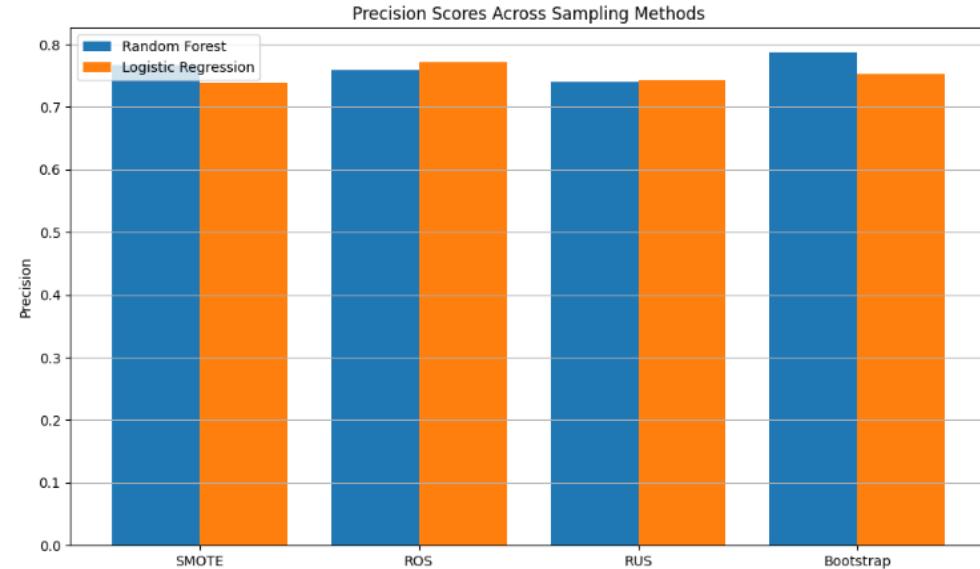
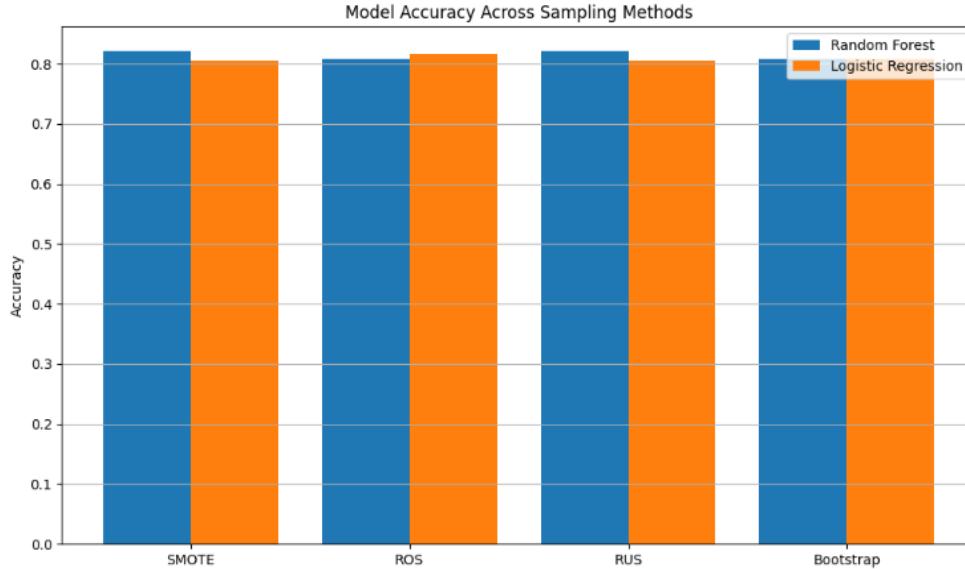
PRECISION



RECALL



F1-SCORE



## Performance summary table

# Model Comparison:

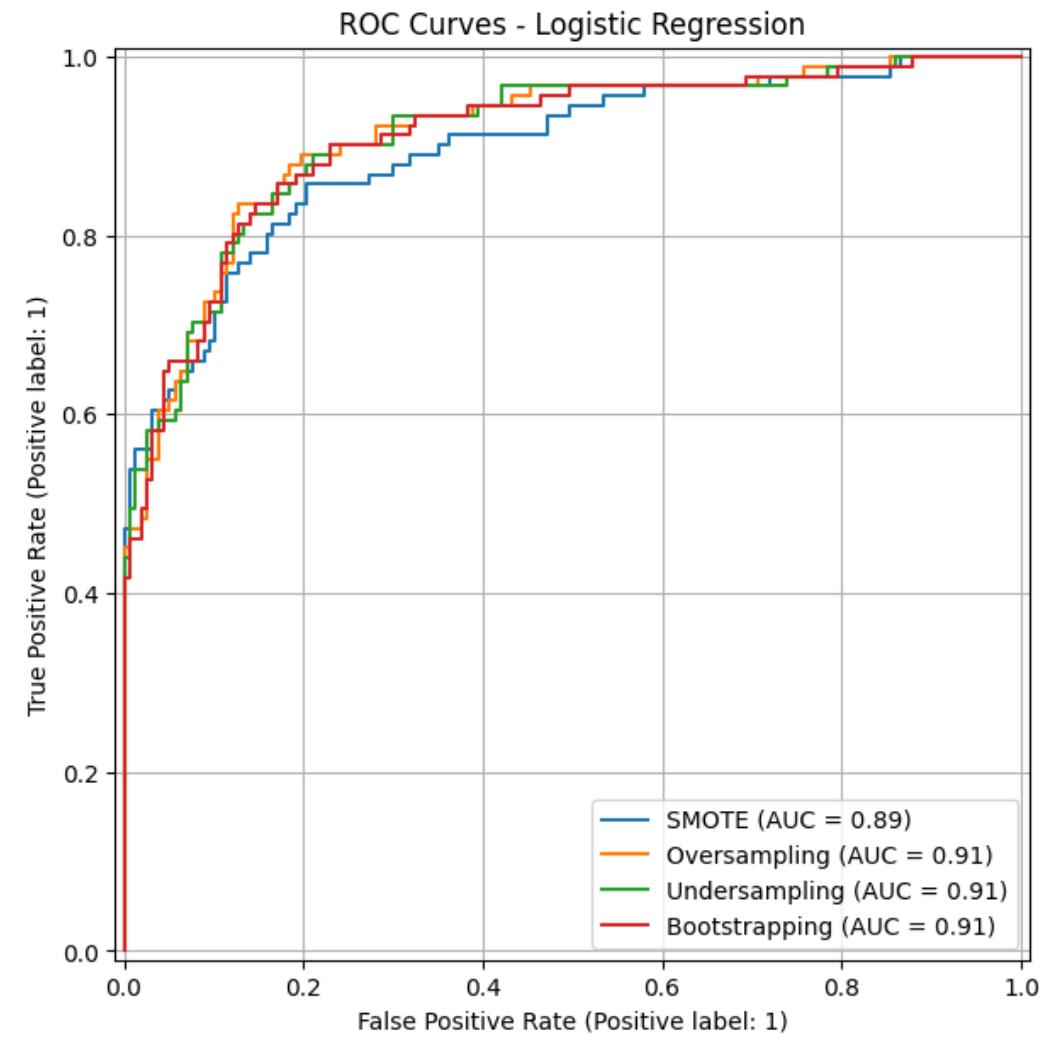
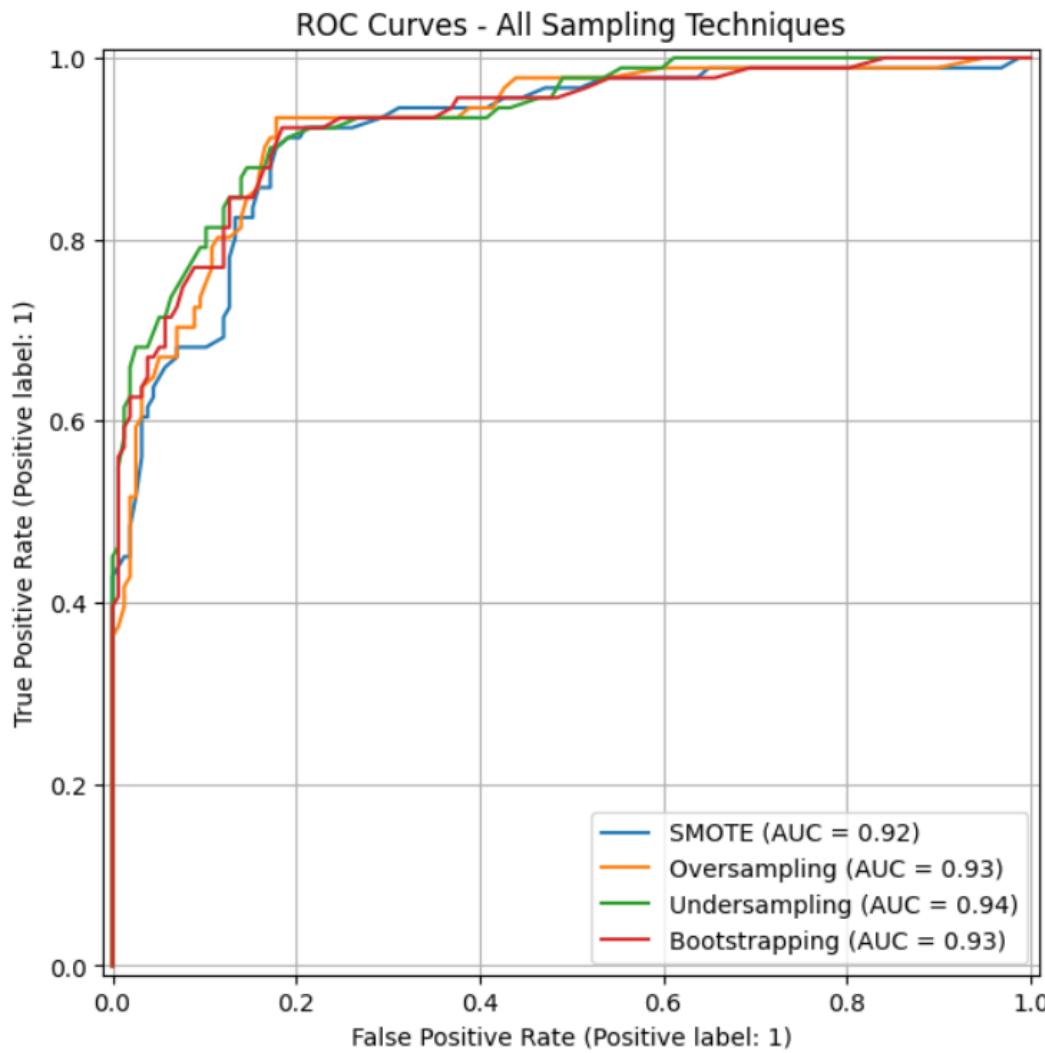
- The SMOTE AUC of 0.92 for Random forest classifier
- The SMOTE AUC of 0.89 for logistic regression
- The best model for predicting GI cancer is Random forest as it is evident that Random Forest consistently outperforms Logistic Regression across all sampling techniques.
- Undersampling + RF achieved the highest F1-score (0.82), showing strong class balance.
- SMOTE + RF provides balanced precision (0.76) and recall (0.83), making it a strong default choice.
- Logistic Regression performs reasonably but slightly lags behind Random Forest in all metrics.
- All sampling strategies helped mitigate class imbalance, with undersampling slightly outperforming others in final model performance.

	Model	Accuracy	Precision	Recall	F1-Score
0	RF - SMOTE	0.843	0.760	0.835	0.796
1	RF - ROS	0.847	0.785	0.802	0.793
2	RF - RUS	0.863	0.782	0.868	0.823
3	RF - Bootstrapping	0.859	0.850	0.747	0.795
4	LR - SMOTE	0.823	0.742	0.791	0.766
5	LR - ROS	0.851	0.793	0.802	0.798
6	LR - RUS	0.843	0.765	0.824	0.794
7	LR - Bootstrapping	0.847	0.791	0.791	0.791

Model Comparison Summary

	Model	Sampling	Accuracy	Precision	Recall	F1-Score
0	Random Forest	SMOTE	0.84	0.76	0.83	0.79
1	Logistic Regression	SMOTE	0.82	0.74	0.79	0.76
2	Random Forest	Oversampling	0.84	0.78	0.80	0.79
3	Random Forest	Undersampling	0.84	0.78	0.86	0.82

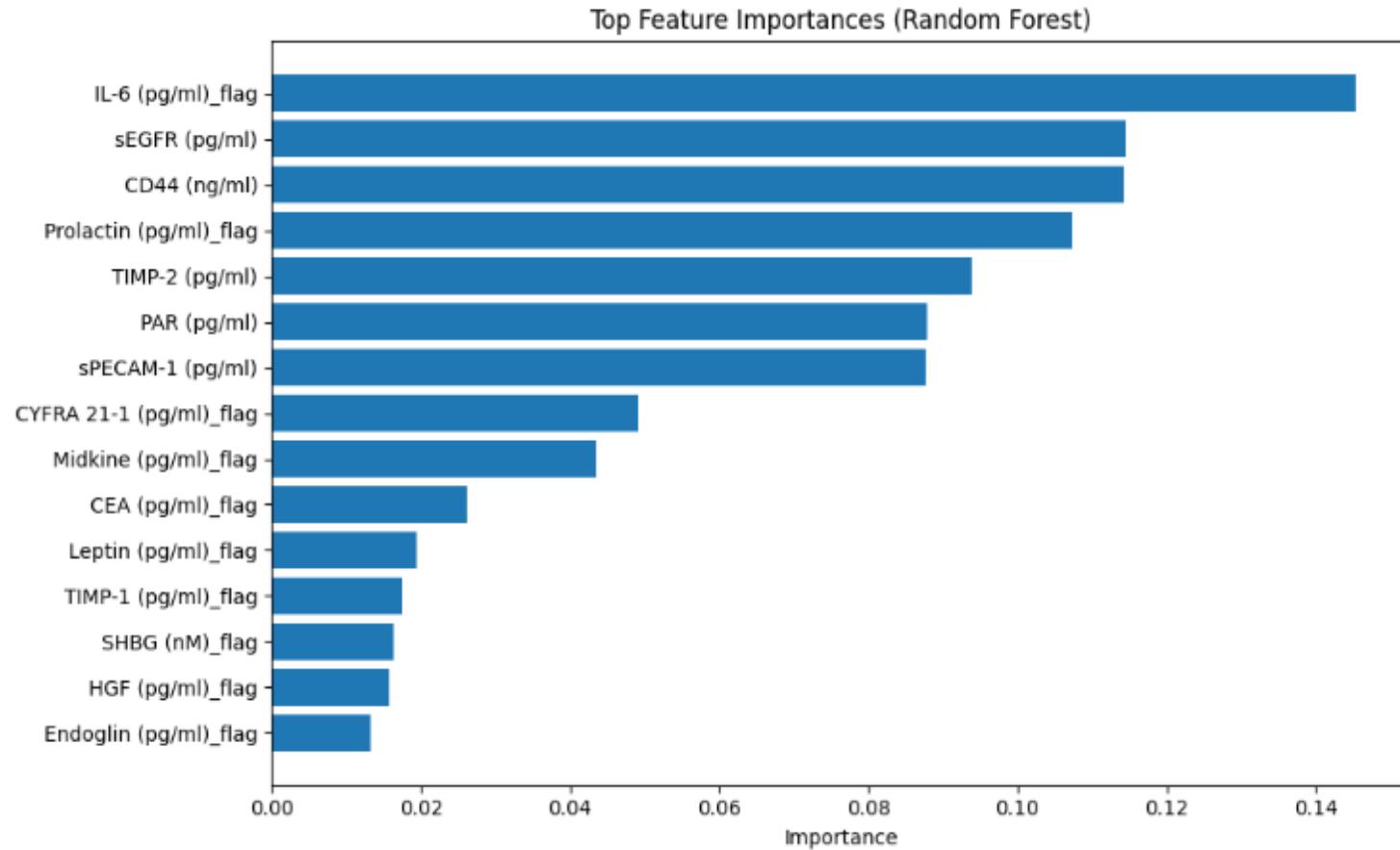
# ROC Curves



# Feature Importance

The Random Forest model revealed that a subset of biomarkers plays a major role in distinguishing GI cancer from normal cases.

Features with higher importance scores contribute more to the model's decision-making, highlighting their potential relevance in diagnostic applications.

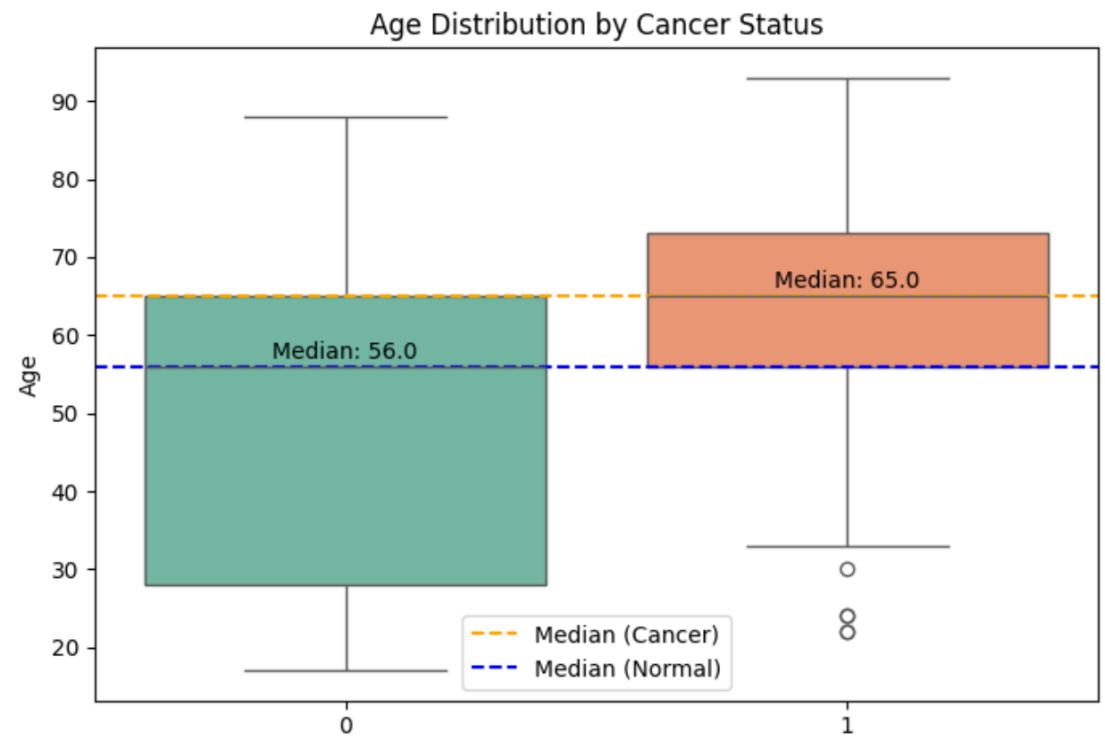


# Demographic Analysis



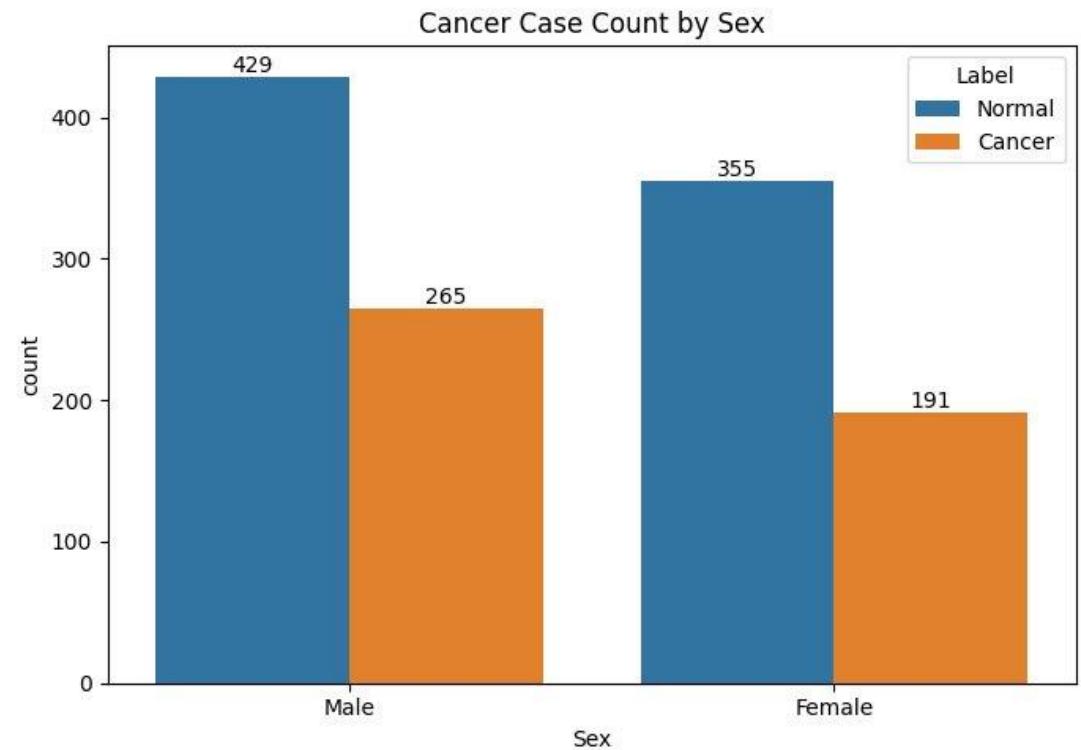
# Age distribution:

- Interpretation:
- Median age for cancer patients from the results is 56. However, patients aged 20, tested positive for GI cancer, indicating a sociodemographic shift



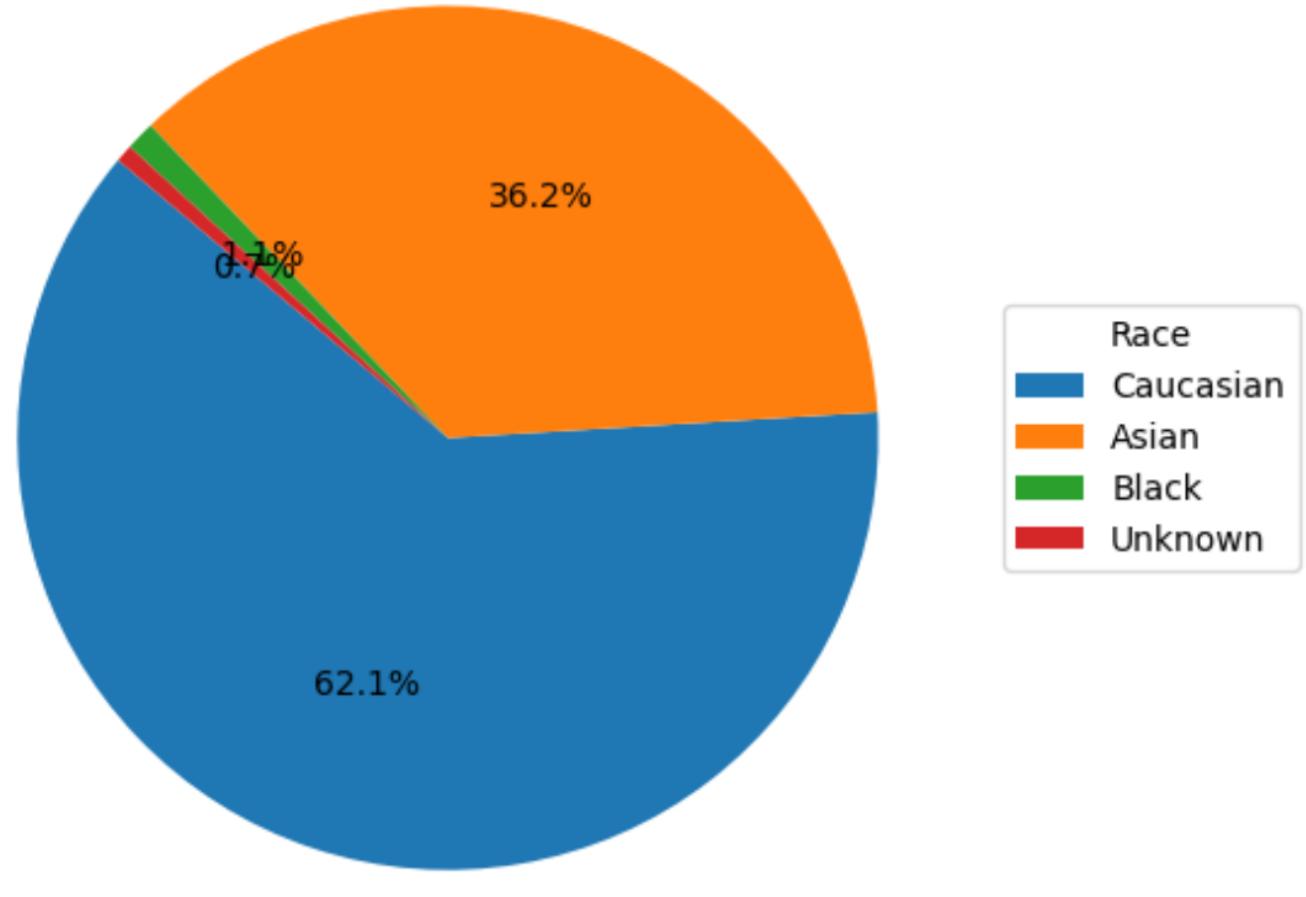
# Sex: Cancer rate by gender

- Interpretation: The results indicate that most of the cancer patients in the data set were males n=265



# Race: Pie chart for class split

Race Breakdown (Cancer)



Interpretation: the Caucasians formed a majority of the GI cancer patients (62.1%) followed by Asians 36.2 %

## DISCUSSION:

- Demographic variables supported clinical correlations
- SMOTE improved model performance by addressing class imbalance, leading to higher AUC scores.
- Random Forest Classifier is the most effective model for predicting GI cancer in this dataset.
- Random Forest + SMOTE outperformed Logistic Regression with an AUC of 0.92, indicating better discrimination ability.
- Significant biomarkers ( $p < 0.05$ ): IL-6, SeGFR, CD-44, Prolactin, TIMP-2, PAR, CEA , Leptin etc.. show strong potential for distinguishing GI cancer from normal cases, consistent with the findings and aided in rejection of null hypothesis (Sato et al., 2023; Tabari et al., 2022).
- **H<sub>0</sub> Rejected:**  
Significant differences found in multiple biomarkers and demographics between cancer and normal patient records

# **Future scope:**

- Future work: Include external validation or multi-class classification
- Evaluate model robustness using independent datasets from external sources to confirm generalizability.
- Extend to subtype classification (e.g., CRC vs gastric)
- Develop a web-based clinical tool for real-time GI cancer risk prediction
- Incorporate time-series or follow-up data to assess biomarker trends over disease progression or treatment response.

# Challenges & Limitations

- Not all biomarkers have universal clinical cutoffs
- Black-box behavior of ensemble models
- **Class Imbalance:** The original dataset had a 64:36 imbalance (800 negative, 455 positive), which could bias the model toward the majority class
- **Synthetic Oversampling (SMOTE):** While it helped balance the data, it may introduce artificial patterns not present in real-world data, risking overfitting
- **Small Sample Size:** Limits generalizability and may reduce model robustness across larger or more diverse populations.

## **Team Contributions:**

---

Mahitha Gogu – Data cleaning and preprocessing

---

Geetanjali Karuturi – Data visualization and exploratory data analysis

---

Frederick Onyango – Statistical analysis and demographic insights

---

Rashmita Kudamala – Machine learning model training and evaluation

# References

- Cohen, J. D., Li, L., Wang, Y., Thoburn, C., Afsari, B., Danilova, L., Douville, C., Javed, A. A., Wong, F., Mattox, A., Hruban, R. H., Wolfgang, C. L., Goggins, M. G., Dal Molin, M., Wang, T.-L., Roden, R., Klein, A. P., Ptak, J., Dobbyn, L., & Schaefer, J. (2018). Detection and localization of surgically resectable cancers with a multi-analyte blood test. *Science*, 359(6378), 926–930. <https://doi.org/10.1126/science.aar3247>
- Du, H., Yang, Q., Ge, A., Zhao, C., Ma, Y., & Wang, S. (2024). Explainable machine learning models for early gastric cancer diagnosis. *Scientific reports*, 14(1), 17457. <https://doi.org/10.1038/s41598-024-67892-z>
- Huang, Y., Shao, Y., Yu, X., Chen, C., Guo, J., & Ye, G. (2024). Global progress and future prospects of early gastric cancer screening. *Journal of Cancer*, 15(10), 3045–3064. <https://doi.org/10.7150/jca.95311>
- Li, L. S., Guo, X. Y., & Sun, K. (2021). Recent advances in blood-based and artificial intelligence-enhanced approaches for gastrointestinal cancer diagnosis. *World journal of gastroenterology*, 27(34), 5666–5681. <https://doi.org/10.3748/wjg.v27.i34.5666>
- Matsuoka, T., & Yashiro, M. (2023). Novel biomarkers for early detection of gastric cancer. *World journal of gastroenterology*, 29(17), 2515–2533. <https://doi.org/10.3748/wjg.v29.i17.2515>
- Sato, Y., Okamoto, K., Kawano, Y., Kasai, A., Kawaguchi, T., Sagawa, T., Sogabe, M., Miyamoto, H., & Takayama, T. (2023). Novel biomarkers of gastric cancer: Current research and future perspectives. *Journal of Clinical Medicine*, 12(14), 4646. <https://doi.org/10.3390/jcm12144646>
- Tabari, A., Chan, S. M., Omar, O. M. F., Iqbal, S. I., Gee, M. S., & Daye, D. (2023). Role of machine learning in precision oncology: Applications in gastrointestinal cancers. *Cancers*, 15(1), 63. <https://doi.org/10.3390/cancers15010063>
- Tong, W., Ye, F., He, L., Cui, L., Cui, M., Hu, Y., Li, W., Jiang, J., Zhang, D. Y., & Suo, J. (2016). Serum biomarker panels for diagnosis of gastric cancer. *Oncotargets and Therapy*, 9, 2455–2463. <https://doi.org/10.2147/OTT.S86139>

- Lee, T., Teng, T. Z. J., & Shelat, V. G. (2020). Carbohydrate antigen 19-9 - tumor marker: Past, present, and future. *World journal of gastrointestinal surgery*, 12(12), 468–490. <https://doi.org/10.4240/wjgs.v12.i12.468>
- Hing, J. X., Mok, C. W., Tan, P. T., Sudhakar, S. S., Seah, C. M., Lee, W. P., & Tan, S. M. (2020). Clinical utility of tumour marker velocity of cancer antigen 15-3 (CA 15-3) and carcinoembryonic antigen (CEA) in breast cancer surveillance. *Breast (Edinburgh, Scotland)*, 52, 95–101. <https://doi.org/10.1016/j.breast.2020.05.005>
- Martínez-Morillo, E., Diamandis, A., & Diamandis, E. P. (2012). Reference intervals and biological variation for kallikrein 6: influence of age and renal failure. *Clinical chemistry and laboratory medicine*, 50(5), 931–934. <https://doi.org/10.1515/cclm-2012-0075>
- Gendall, A. R., Levy, Y. Y., Wilson, A., & Dean, C. (2001). The VERNALIZATION 2 gene mediates the epigenetic regulation of vernalization in *Arabidopsis*. *Cell*, 107(4), 525–535. [https://doi.org/10.1016/s0092-8674\(01\)00573-6](https://doi.org/10.1016/s0092-8674(01)00573-6)
- Wakatsuki, M., Shintani, Y., Abe, M., Liu, Z. H., Shitsukawa, K., & Saito, S. (1996). Immunoradiometric assay for follistatin: serum immunoreactive follistatin levels in normal adults and pregnant women. *The Journal of clinical endocrinology and metabolism*, 81(2), 630–634. <https://doi.org/10.1210/jcem.81.2.8636280>
- Weng, M., Yue, Y., Wu, D., Zhou, C., Guo, M., Sun, C., Liao, Q., Sun, M., Zhou, D., & Miao, C. (2022). Increased MPO in Colorectal Cancer Is Associated With High Peripheral Neutrophil Counts and a Poor Prognosis: A TCGA With Propensity Score-Matched Analysis. *Frontiers in Oncology*, 12. <https://doi.org/10.3389/fonc.2022.940706>
-

- Kemper, M., Hentschel, W., Graß, J., Stüben, B., Konczalla, L., Rawnaq, T., Ghadban, T., Izwicki, J. R., & Reeh, M. (2020). Serum Midkine is a clinical significant biomarker for colorectal cancer and associated with poor survival. *Cancer Medicine*, 9(6), 2010–2018.  
<https://doi.org/10.1002/cam4.2884>
- Li, S., Xie, L., He, L., Fan, Z., Xu, J., Xu, K., Zhu, L., Ma, G., Du, M., Chu, H., Zhang, Z., Ni, M., & Wang, M. (2017). Plasma Mesothelin as a Novel Diagnostic and Prognostic Biomarker in Colorectal Cancer. *Journal of Cancer*, 8(8), 1355–1361. <https://doi.org/10.7150/jca.18014>
- Capelle, L. G., De Vries, A. C., Haringsma, J., Steyerberg, E. W., Loosman, C. W. N., Nagtzaam, N. M. A., Van Dekken, H., Ter Borg, F., De Vries, R. A., & Kuipers, E. J. (2009). Serum Levels of Leptin As Marker For Patients At High Risk of Gastric Cancer. *Helicobacter*, 14(6), 596–604.  
<https://doi.org/10.1111/j.1523-5378.2009.00728.x>
- Lin, D., Alborn, W. E., Slebos, R. J. C., & Liebler, D. C. (2013). Comparison of Protein Immunoprecipitation-Multiple Reaction Monitoring with ELISA for Assay of Biomarker Candidates in Plasma. *Journal of Proteome Research*, 12(12), 5996–6003. <https://doi.org/10.1021/pr400877e>
- Gordon-Weeks, A. N., Lim, S. Y., Yuzhalin, A. E., Jones, K., Markelc, B., Kim, K. J., Buzzelli, J. N., Fokas, E., Cao, Y., Smart, S., & Muschel, R. (2017). Neutrophils promote hepatic metastasis growth through fibroblast growth factor 2-dependent angiogenesis in mice. *Hepatology (Baltimore, Md.)*, 65(6), 1920–1935. <https://doi.org/10.1002/hep.29088>
- Kushlinskii, N. E., Gershtein, E. S., Ivannikov, A. A., Davydov, M. M., Chang, V. L., Ognerubov, N. A., & Stilidi, I. S. (2019). Clinical Significance of Matrix Metalloproteinases in Blood Plasma of Patients with Gastric Cancer. *Bulletin of Experimental Biology and Medicine*, 166(3), 373–376.  
<https://doi.org/10.1007/s10517-019-04353-y>

- Cosentino-Boehm, A. L., Lafky, J. M., Greenwood, T. M., Kimbler, K. D., Buenafe, M. C., Wang, Y., Branscum, A. J., Yang, P., Maihle, N. J., & Baron, A. T. (2013). Soluble Human Epidermal Growth Factor Receptor 2 (sHER2) as a Potential Risk Assessment, Screening, and Diagnostic Biomarker of Lung Adenocarcinoma. *Diagnostics (Basel, Switzerland)*, 3(1), 13–32. <https://doi.org/10.3390/diagnostics3010013>
- Moskal, T. L., Huang, S., Ellis, L. M., Fritsche, H. A., & Chakrabarty, S. (1995). Serum levels of transforming growth factor alpha in gastrointestinal cancer patients. *Cancer Epidemiology, Biomarkers & Prevention : A Publication of the American Association for Cancer Research, Cosponsored by the American Society of Preventive Oncology*, 4(2), 127–131. <https://pubmed.ncbi.nlm.nih.gov/7742720/>
- Qin, L., Wang, Y., Yang, N., Zhang, Y., Zhao, T., Wu, Y., & Jiang, J. (2021). Tissue inhibitor of metalloproteinase-1 (TIMP-1) as a prognostic biomarker in gastrointestinal cancer: a meta-analysis. *PeerJ*, 9, e10859. <https://doi.org/10.7717/peerj.10859>
- Sun, H., Yang, L., Li, N., Hu, Y., Hu, Q., Zhou, Z., & Cong, X. (2025). The effect of 1,25(OH)2D3 on Dickkopf-1 methylation in colorectal cancer. *Clinical Epigenetics*, 17(1). <https://doi.org/10.1186/s13148-025-01857-5>
- Lipton, A., Ali, S. M., Leitzel, K., Chinchilli, V., Witters, L., Engle, L., Holloway, D., Bekker, P., & Dunstan, C. R. (2002). Serum osteoprotegerin levels in healthy controls and cancer patients. *Clinical cancer research : an official journal of the American Association for Cancer Research*, 8(7), 2306–2310.
- Byrling, J., Hilmersson, K. S., Ansari, D., Andersson, R., & Andersson, B. (2022). Thrombospondin-2 as a diagnostic biomarker for distal cholangiocarcinoma and pancreatic ductal adenocarcinoma. *Clinical & translational oncology : official publication of the Federation of Spanish Oncology Societies and of the National Cancer Institute of Mexico*, 24(2), 297–304. <https://doi.org/10.1007/s12094-021-02685-8>

- Boroumand-Noughabi, S., Sima, H. R., Ghaffarzadehgan, K., Jafarzadeh, M., Raziee, H. R., Hosseinnezhad, H., Moaven, O., Rajabi-Mashhad, M. T., Azarian, A. A., Mashhadinejad, M., & Tavakkol-Afshari, J. (2010). Soluble Fas might serve as a diagnostic tool for gastric adenocarcinoma. *BMC cancer*, 10, 275. <https://doi.org/10.1186/1471-2407-10-275>
- Karabulut, M., Afsar, C. U., Alis, H., Oran, E., Karabulut, S., Akarsu, C., Sahbaz, N. A., Gümüsoglu, A. Y., Bilgin, E., & Aykan, N. F. (2017). Evaluation of epidermal growth factor receptor serum levels and their association with clinicopathological characteristics in patients with colorectal cancer. *Molecular and clinical oncology*, 7(5), 787–797. <https://doi.org/10.3892/mco.2017.1405>
- Kankanala, V. L., & Mukkamalla, S. K. R. (2023, January 23). *Carcinoembryonic Antigen*. PubMed; StatPearls Publishing. <https://www.ncbi.nlm.nih.gov/books/NBK578172/>
- Adigun, O. O., & Shailesh Khetarpal. (2019, February 22). *Alpha Fetoprotein (AFP, Maternal Serum Alpha Fetoprotein, MSAFP)*. Nih.gov; StatPearls Publishing. <https://www.ncbi.nlm.nih.gov/books/NBK430750/>
- Karagiannidis, I., Salataj, E., Said Abu Egal, E., & Beswick, E. J. (2021). G-CSF in tumors: Aggressiveness, tumor microenvironment and immune cell regulation. *Cytokine*, 142, 155479. <https://doi.org/10.1016/j.cyto.2021.155479>
- Anas. (2020, September 9). *Prolactin: Reference Range, Interpretation, Collection and Panels*. Medscape.com; Medscape. [https://emedicine.medscape.com/article/2089400-overview?utm\\_source=chatgpt.com](https://emedicine.medscape.com/article/2089400-overview?utm_source=chatgpt.com)

# Appendix: Code Snippets

```
: myvars = {}
with open("rakuda-mysql-password") as myfile:
    for line in myfile:
        name, var = line.partition(":")[:2]
        myvars[name.strip()] = var.strip()

: myvars.keys()

: dict_keys(['DB username', 'DB databasename', 'DB password'])

: import MySQLdb
conn = MySQLdb.connect(host="localhost", user=myvars['DB username'], passwd=myvars['DB password'], db=myvars['DB databasename'])
cursor = conn.cursor()

: cursor.execute("USE I501_Spring2025_Sec27856_group04_db")
cursor.execute("SELECT DATABASE()")
print("Now using:", cursor.fetchone())

Now using: ('I501_Spring2025_Sec27856_group04_db',)

: cursor.execute('select * FROM intro_project_dataset');

rows_table1 = cursor.fetchall()

cursor.execute('select * FROM intro_project_dataset_2');

rows_table2 = cursor.fetchall()

: print("Rows in intro_project_dataset:", len(rows_table1))
print("Rows in intro_project_dataset_2:", len(rows_table2))

Rows in intro_project_dataset: 1822
Rows in intro_project_dataset_2: 1822
```

```

import pandas as pd

pd.set_option('display.max_rows', None)

cursor.execute('SELECT * FROM intro_project_dataset')
rows_table1 = cursor.fetchall()
columns_table1 = [desc[0] for desc in cursor.description]
df1 = pd.DataFrame(rows_table1, columns=columns_table1)

print("Full Dataset 1:")
display(df1.head())

```

Full Dataset 1:

	COL 1	COL 2	COL 3	COL 4	COL 5	COL 6	COL 7	COL 8	COL 9	COL 10	COL 11	COL 12	COL 13
0	Patient ID #	Plasma sample ID #	Primary tumor sample ID #	Age	Sex	Race	Tumor type	AJCC Stage	Histopathology	Plasma volume (mL)	Plasma DNA concentration (ng/mL)	CancerSEEK Logistic Regression Score	CancerSEEK Test Result
1	CRC 455	CRC 455 PLS 1	Not available	60	Male	Caucasian	Colorectum	I	Adenocarcinoma	5	6.08	0.938	Positive
2	CRC 456	CRC 456 PLS 1	CRC 456 PT1	59	Female	Caucasian	Colorectum	I	Adenocarcinoma	4	46.01	0.925	Positive
3	CRC 457	CRC 457 PLS 1	CRC 457 PT1	69	Female	Caucasian	Colorectum	II	Adenocarcinoma	4.5	6.94	0.852	Negative
4	CRC 458	CRC 458 PLS 1	CRC 458 PT1	70	Female	Caucasian	Colorectum	II	Adenocarcinoma	7.5	7.15	0.618	Negative

```

cursor.execute('SELECT * FROM intro_project_dataset_2')
rows_table2 = cursor.fetchall()
columns_table2 = [desc[0] for desc in cursor.description]
df2 = pd.DataFrame(rows_table2, columns=columns_table2)

print("Full Dataset 2:")
display(df2.head())

```

Full Dataset 2:

	COL 1	COL 2	COL 3	COL 4	COL 5	COL 6	COL 7	COL 8	COL 9	COL 10	COL 11	COL 12	COL 13	COL 14	COL 15	COL 16	COL 17	COL 18
0	Patient ID #	Sample ID #	Tumor type	AJCC Stage	AFP (pg/ml)	Angiopoietin-2 (pg/ml)	AXL (pg/ml)	CA-125 (U/ml)	CA 15-3 (U/ml)	CA19-9 (U/ml)	CD44 (ng/ml)	CEA (pg/ml)	CYFRA 21-1 (pg/ml)	DKK1 (ng/ml)	Endoglin (pg/ml)	FGF2 (pg/ml)	Follistatin (pg/ml)	Galec (ng/ml)
1	CRC 455	CRC 455 PLS 1	Colorectum	I	1583.45	5598.5	3621.04	5.09	19.08	*16.452	9.81	540.1	*1938.654	0.78	2882.65	92.02	2144.33	17
2	CRC 456	CRC 456 PLS 1	Colorectum	I	*715.308	20936.35	2772.96	7.27	10.04	40.91	27.57	5,902.40	*1938.654	0.77	3921.77	164.06	1646.26	
CRC																		

```
cursor.execute("SELECT * FROM intro_project_dataset")
rows_table1 = cursor.fetchall()

cursor.execute("SELECT * FROM intro_project_dataset_2")
rows_table2 = cursor.fetchall()

# Using first row as column names
columns_table1 = rows_table1[0]
data_table1 = rows_table1[1:]

columns_table2 = rows_table2[0]
data_table2 = rows_table2[1:]

df1 = pd.DataFrame(data_table1, columns=columns_table1)
df2 = pd.DataFrame(data_table2, columns=columns_table2)

print("df1 columns:", df1.columns.tolist())
print("df2 columns:", df2.columns.tolist())
display(df1.head())
display(df2.head())
```

```
df1 columns: ['Patient ID #', 'Plasma sample ID #', 'Primary tumor sample ID #', 'Age', 'Sex', 'Race', 'Tumor type', 'AJCC Stage', 'Histopathology', 'Plasma volume (mL)', 'Plasma DNA concentration (ng/mL)', 'CancerSEEK Logistic Regression Score', 'CancerSEEK Test Result']
df2 columns: ['Patient ID #', 'Sample ID #', 'Tumor type', 'AJCC Stage', 'AFP (pg/ml)', 'Angiopoietin-2 (pg/ml)', 'AXL (pg/ml)', 'CA-125 (U/ml)', 'CA 5-3 (U/ml)', 'CA19-9 (U/ml)', 'CD44 (ng/ml)', 'CEA (pg/ml)', 'CYFRA 21-1 (pg/ml)', 'DKK1 (ng/ml)', 'Endoglin (pg/ml)', 'FGF2 (pg/ml)', 'Follistatin (pg/ml)', 'Galectin-3 (ng/ml)', 'G-CSF (pg/ml)', 'GDF15 (ng/ml)', 'HE4 (pg/ml)', 'HGF (pg/ml)', 'IL-6 (pg/ml)', 'IL-8 (pg/ml)', 'Kallikrein-6 (pg/ml)', 'Leptin (pg/ml)', 'Mesothelin (ng/ml)', 'Midkine (pg/ml)', 'Myeloperoxidase (ng/ml)', 'NSE (ng/ml)', 'OPG (ng/ml)', 'OPN (pg/ml)', 'PAR (pg/ml)', 'Pro actin (pg/ml)', 'sEGFR (pg/ml)', 'sFas (pg/ml)', 'SHBG (nM)', 'sHER2/sEGFR2/sErbB2 (pg/ml)', 'sPECAM-1 (pg/ml)', 'TGFa (pg/ml)', 'Thrombospondin-2 (pg ml)', 'TIMP-1 (pg/ml)', 'TIMP-2 (pg/ml)', 'CancerSEEK Logistic Regression Score', 'CancerSEEK Test Result']
```

```

df1 = df1.rename(columns={"Plasma sample ID #": "Sample_ID"})
df2 = df2.rename(columns={"Sample ID #": "Sample_ID"})

# full outer join on Sample_ID
merged_df = pd.merge(df1, df2, on="Sample_ID", how="outer", suffixes=('_from_df1', '_from_df2'))

print("Merged DataFrame (with unmatched rows showing NaN):")
display(merged_df.head())

```

merged\_df

	Patient ID #_from_df1	Sample_ID	Primary tumor sample ID #	Age	Sex	Race	Tumor type_from_df1	AJCC Stage_from_df1	Histopathology	Plasma volume (mL)	Plasma DNA concentration (ng/mL)	CancerSEI Logis Regres
0	CRC 455	CRC 455 PLS 1	Not available	60	Male	Caucasian	Colorectum	I	Adenocarcinoma	5	6.08	0.9
1	CRC 456	CRC 456 PLS 1	CRC 456 PT1	59	Female	Caucasian	Colorectum	I	Adenocarcinoma	4	46.01	0.9
2	CRC 457	CRC 457 PLS 1	CRC 457 PT1	69	Female	Caucasian	Colorectum	II	Adenocarcinoma	4.5	6.94	0.8
3	CRC 458	CRC 458 PLS 1	CRC 458 PT1	70	Female	Caucasian	Colorectum	II	Adenocarcinoma	7.5	7.15	0.6
4	CRC 459	CRC 459 PLS 1	CRC 459 PT1	43	Female	Caucasian	Colorectum	II	Adenocarcinoma	5	9.81	0.3
5	CRC 460	CRC 460 PLS 1	CRC 460 PT1	72	Male	Caucasian	Colorectum	II	Adenocarcinoma	5	16.33	0.9
6	CRC 461	CRC 461 PLS 1	CRC 461 PT1	70	Male	Caucasian	Colorectum	I	Adenocarcinoma	7.5	8.93	0.3

```
print("\nMissing values per column:")
print(merged_df.isna().sum())
```

```
Missing values per column:
Patient ID #_from_df1          0
Sample_ID                         0
Primary tumor sample ID #        0
Age                               0
Sex                               0
Race                             0
Tumor type_from_df1              0
AJCC Stage_from_df1              0
Histopathology                     0
Plasma volume (mL)               0
Plasma DNA concentration (ng/mL) 0
CancerSEEK Logistic Regression Score_from_df1 0
CancerSEEK Test Result_from_df1   0
Patient ID #_from_df2              0
Tumor type_from_df2              0
AJCC Stage_from_df2              0
AFP (pg/ml)                       0
Angiopoietin-2 (pg/ml)            0
AXL (pg/ml)                      0
CA-125 (U/ml)                    0
CA 15-3 (U/ml)                   0
CA19-9 (U/ml)                    0
CD44 (ng/ml)                     0
CEA (pg/ml)                      0
CYFRA 21-1 (pg/ml)               0
DKK1 (ng/ml)                     0
Endoglin (pg/ml)                 0
FGF2 (pg/ml)                     0
Follistatin (pg/ml)              0
```

```
has_null = merged_df.isnull().values.any()

if has_null:
    print("The dataset contains null values.")
    null_counts = merged_df.isnull().sum()
    print("\nNull value counts by column:")
    print(null_counts[null_counts > 0])

else:
    print("The dataset does not contain any null values.")

print("\nDataset info:")
merged_df.info()
```

The dataset does not contain any null values.

```

df = merged_df.applymap(lambda x: str(x).replace('*', '') if isinstance(x, str) else x)
df.head()

```

AJCC m_df1	Histopathology	Plasma	... (pg/ml)	sFas	SHBG	sHER2/sEGFR2/sErbB2	sPECAM-1 (pg/ml)	TGFα	Thrombospondin-2 (pg/ml)	TIMP-1 (pg/ml)	TIMP-2 (pg/ml)	CancerSEEK
		volume (mL)		(nM)	(pg/ml)	(pg/ml)	(pg/ml)	(pg/ml)	Score_from_df2	Logistic Regression	CancerSEEK Test Result_from_df2	
I	Adenocarcinoma	5	...	204.792	55.06		6832.07	9368.53	16.086	21863.74	56428.71	39498.82
I	Adenocarcinoma	4	...	204.792	72.92		5549.47	6224.55	16.086	29669.66	73940.49	41277.09
II	Adenocarcinoma	4.5	...	204.792	173.78		3698.16	4046.48	179.03	6020.47	22797.28	28440.6
II	Adenocarcinoma	7.5	...	204.792	29.47		5856	6121.93	16.086	4331.02	20441.19	25896.73
II	Adenocarcinoma	5	...	204.792	78.07		5447.93	6982.32	16.086	2311.91	56288.51	49425.2

```
keywords = [
    'tumor type_from_df2', 'tumor type_from_df3',
    'patient id #_from_df1', 'patient id #_from_df2',
    'cancerseek logistic regression score_from_df1', 'cancerseek test result_from_df1',
    'cancerseek logistic regression score_from_df2', 'cancerseek test result_from_df2',
    'ajcc stage_from_df2', 'ajcc stage_from_df1', 'histopathology', 'primary tumor sample id #'
]

columns_to_drop = [col for col in merged_df.columns if any(key in col.lower() for key in keywords)]

print("Dropping columns:")
print(columns_to_drop)

df_cleaned = df.drop(columns=columns_to_drop)

df_cleaned.head()

Dropping columns:
['Patient ID #_from_df1', 'Primary tumor sample ID #', 'AJCC Stage_from_df1', 'Histopathology', 'CancerSEEK Logistic Regression Score_from_df1', 'CancerSEEK Test Result_from_df1', 'Patient ID #_from_df2', 'Tumor type_from_df2', 'AJCC Stage_from_df2', 'CancerSEEK Logistic Regression Score_from_df2', 'CancerSEEK Test Result_from_df2']

exclude_tumors = ['breast', 'esophagus', 'liver', 'lung', 'ovary', 'pancreas']

df_filtered = df_cleaned[~df_cleaned['Tumor type_from_df1'].str.lower().isin(exclude_tumors)]

print("Remaining tumor types:", df_filtered['Tumor type_from_df1'].unique())
print("Filtered dataset shape:", df_filtered.shape)
df_filtered
```

Remaining tumor types: ['Colorectum' 'Stomach' 'Normal']  
Filtered dataset shape: (1255, 47)

```
# Data cleaning 3. Delete unwanted rows
```

```
df_cleaned = df_filtered.iloc[:1268]
```

```
df_cleaned.tail()
```

	Sample_ID	Age	Sex	Race	Tumor type_from_df1	Plasma volume (mL)	Plasma DNA concentration (ng/mL)	AFP (pg/ml)	Angiopoietin-2 (pg/ml)	AXL (pg/ml)	...	Prolactin (pg/ml)	sEGFR (pg/ml)	sFas (pg/ml)	SHBG (nM)
1698	NL PLSA 1850	58	Male	Unknown	Normal	7.5	1.36	740.016	1197.88	3780.3	...	9851.43	4677.93	1072.02	32.15
1699	NL PLSA 1851	51	Male	Unknown	Normal	7.5	2.2	740.016	1519.08	2893.79	...	7244.91	5158.71	192.948	22.91
1700	NL PLSA 1852	28	Male	Unknown	Normal	7.5	4.27	740.016	3916.14	3334.98	...	7783.77	2112.99	1016.67	147.48
1701	NL PLSA 1853	62	Male	Unknown	Normal	7.5	0.66	740.016	1142.66	1649.39	...	5271.87	2219.83	844.47	37.79
1702	NL PLSA 1854	64	Male	Unknown	Normal	7.5	0.59	740.016	2184.08	1492.69	...	5222.35	1732.68	1137.23	118.99

```
# Count missing values per column
```

```
missing_counts = df_cleaned.isna().sum()
```

```
missing_counts = missing_counts[missing_counts > 0]
```

```
if not missing_counts.empty:
```

```
    print("Columns with missing values:\n")
```

```
    for col in missing_counts.index:
```

```
        missing_rows = df_cleaned[df_cleaned[col].isna()].index.tolist()
```

```
        print(f"- {col}: {missing_counts[col]} missing (Rows: {missing_rows})")
```

```
else:
```

```
    print("No missing values found in the dataset!")
```

No missing values found in the dataset!

```

import pandas as pd

df_cleaned = pd.read_csv("cleaned_dataset3.csv")

df_cleaned["Age"] = pd.to_numeric(df_cleaned["Age"], errors="coerce").fillna(0).astype(int)

for col in df_cleaned.columns:
    if col in ["Age", "Sample_ID", "Tumor type_from_df1"]:
        continue

    col_cleaned = df_cleaned[col].astype(str).str.replace(","," ")
    col_cleaned = col_cleaned.str.replace(r"[^0-9.\-]", "", regex=True)

    converted = pd.to_numeric(col_cleaned, errors="coerce")
    success_rate = converted.notna().mean()

    if success_rate > 0.5:
        df_cleaned[col] = converted
        print(f"Cleaned and converted '{col}' to float.")
    else:
        print(f"Skipped '{col}' - categorical")

```

Skipped 'Sex' - categorical  
 Skipped 'Race' - categorical  
 Cleaned and converted 'Plasma volume (mL)' to float.  
 Cleaned and converted 'Plasma DNA concentration (ng/mL)' to float.  
 Cleaned and converted 'AFP (pg/ml)' to float.  
 Cleaned and converted 'Angiopoietin-2 (pg/ml)' to float.  
 Cleaned and converted 'AXL (pg/ml)' to float.  
 Cleaned and converted 'CA-125 (U/ml)' to float.  
 Cleaned and converted 'CA 15-3 (U/ml)' to float.  
 Cleaned and converted 'CA19-9 (U/ml)' to float.  
 Cleaned and converted 'CD44 (ng/ml)' to float.  
 Cleaned and converted 'CEA (pg/ml)' to float.  
 Cleaned and converted 'CYFRA 21-1 (pg/ml)' to float.  
 Cleaned and converted 'DKK1 (ng/ml)' to float.  
 Cleaned and converted 'Endoglin (pg/ml)' to float.  
 Cleaned and converted 'EGFR (ng/ml)' to float

```
df_cleaned['Label'] = df_cleaned['Tumor type_from_df1'].apply(lambda x: 0 if str(x).strip().lower() == 'normal' else 1)
df_cleaned.tail()
```

dase /ml)	NSE (ng/ml)	OPG (ng/ml)	OPN (pg/ml)	PAR (pg/ml)	Prolactin (pg/ml)	sEGFR (pg/ml)	sFas (pg/ml)	SHBG (nM)	sHER2/sEGFR2/sErbB2 (pg/ml)	sPECAM- 1 (pg/ml)	TGF $\alpha$ (pg/ml)	Thrombospondin- 2 (pg/ml)	TIMP-1 (pg/ml)	TIMP-2 (pg/ml)	Label
6.49	25.08	0.28	8913.64	11574.68	9851.43	4677.93	1072.020	32.15	6824.90	9314.25	15.258	4831.59	79549.74	42248.26	0
9.63	28.46	0.26	20352.33	4964.90	7244.91	5158.71	192.948	22.91	4776.64	3605.28	15.258	3678.90	59497.63	32619.49	0
7.56	22.30	0.29	31948.50	8118.53	7783.77	2112.99	1016.670	147.48	5327.25	4151.03	15.258	5595.27	59549.00	32305.11	0
6.83	12.74	0.25	21467.85	10695.84	5271.87	2219.83	844.470	37.79	5396.73	5784.36	15.258	835.00	51314.32	24498.68	0
9.69	22.90	0.28	15122.44	3341.92	5222.35	1732.68	1137.230	118.99	3535.00	3506.51	15.258	835.00	49214.66	31252.03	0

```
import pandas as pd
df = pd.read_csv("draft_dataset4.csv")
missing_counts = df.isnull().sum()
missing_columns = missing_counts[missing_counts > 0]
if missing_columns.empty:
    print("No missing values in the dataset.")
else:
    print("Columns with missing values:")
    print(missing_columns.sort_values(ascending=False))
```

```
Columns with missing values:
AXL (pg/ml)           6
CD44 (ng/ml)          6
G-CSF (pg/ml)          6
Kallikrein-6 (pg/ml)   6
Mesothelin (ng/ml)     6
Midkine (pg/ml)        6
PAR (pg/ml)            6
sEGFR (pg/ml)          6
sHER2/sEGFR2/sErbB2 (pg/ml) 6
sPECAM-1 (pg/ml)       6
Thrombospondin-2 (pg/ml) 6
sFas (pg/ml)           1
dtype: int64
```

```
# Drop missing value rows
df_cleaned = df.dropna()

print("Remaining rows:", df_cleaned.shape[0])
```

```
Remaining rows: 1255
```

```
# Checking skewness for handling missing values
numeric_cols = df_cleaned.select_dtypes(include=['float64', 'int64']).columns
if 'Label' in numeric_cols:
    numeric_cols = numeric_cols.drop('Label')

skew_values = df_cleaned[numeric_cols].skew().sort_values(ascending=False)

print("Skewness of numeric columns:")
print(skew_values)

Skewness of numeric columns:
CA19-9 (U/ml)           34.734604
CYFRA 21-1 (pg/ml)       32.489266
IL-8 (pg/ml)             24.049594
AFP (pg/ml)              22.573246
IL-6 (pg/ml)              19.607046
HE4 (pg/ml)              18.528075
G-CSF (pg/ml)             18.114551
CA 15-3 (U/ml)            12.075983
GDF15 (ng/ml)             11.752472
CEA (pg/ml)                11.449617
TGFa (pg/ml)               11.370911
sFas (pg/ml)                 9.721895
Thrombospondin-2 (pg/ml)   9.677284
CA-125 (U/ml)                9.000275
Midkine (pg/ml)              8.451463
Angiopoietin-2 (pg/ml)      7.928636
Myeloperoxidase (ng/ml)      6.870869
HGF (pg/ml)                  6.777124
Galectin-3 (ng/ml)            6.141039
~ ~ ~ ~ ~
```

```

rows_with_na = df[df.isnull().any(axis=1)]
df_cleaned = df.dropna()

pd.set_option('display.max_columns', None)
rows_with_na.to_csv("Dropped_rows_table.csv", index=False)

```

`df_cleaned.tail()`

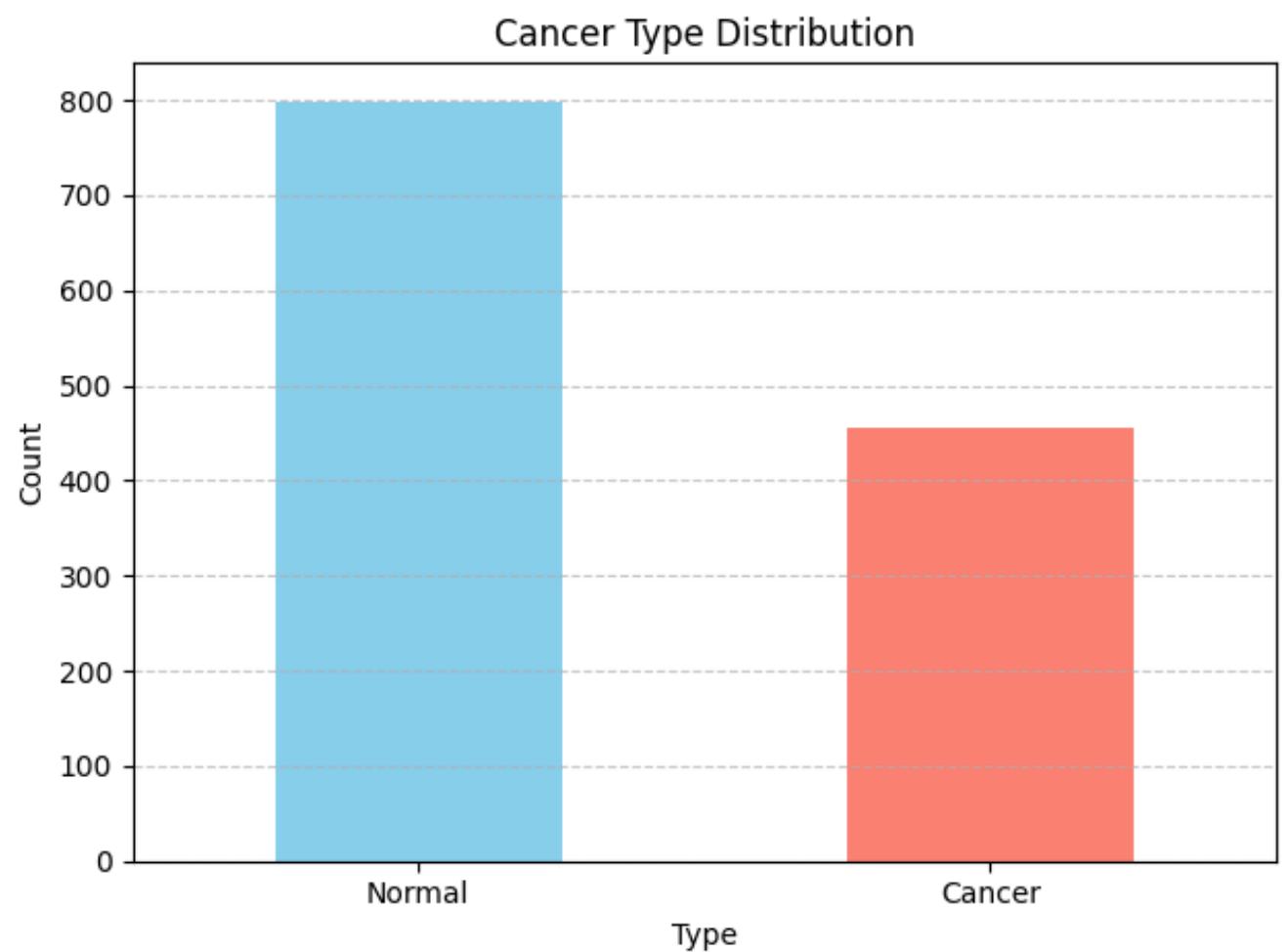
	Sample_ID	Age	Sex	Race	Tumor type_from_df1	Plasma volume (mL)	Plasma DNA concentration (ng/mL)	AFP (pg/ml)	Angiopoietin-2 (pg/ml)	AXL (pg/ml)	...	sEGFR (pg/ml)	sFas (pg/ml)	SHBG (nM)
1263	NL PLSA 1866	55	Female	Unknown	Normal	7.5	2.17	740.016	1683.29	2151.58	...	2989.40	1237.140	60.85
1264	NL PLSA 1867	73	Female	Unknown	Normal	7.5	0.89	740.016	802.16	1069.51	...	1779.41	1237.140	24.30
1265	NL PLSA 1868	38	Male	Unknown	Normal	7.5	1.29	740.016	1112.37	1606.96	...	1823.73	623.750	14.60
1266	NL PLSA 1869	52	Female	Unknown	Normal	7.5	0.49	740.016	583.62	1708.47	...	2334.46	1142.230	28.83
1267	NL PLSA 1870	34	Female	Unknown	Normal	7.5	0.77	740.016	312.88	685.72	...	3263.91	192.948	109.81
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	count	mean	std	min	25%	50%	75%	max
<b>Age</b>	1255.0	54.697211	18.827110	17.000	39.000	59.000	68.000	93.000
<b>Plasma volume (mL)</b>	1255.0	7.439060	0.398912	2.000	7.500	7.500	7.500	7.500
<b>Plasma DNA concentration (ng/mL)</b>	1255.0	7.226183	12.575776	0.000	1.990	3.950	7.190	157.480
<b>AFP (pg/ml)</b>	1255.0	2744.607906	21275.864680	706.158	807.054	895.128	1603.800	592312.722
<b>Angiopoietin-2 (pg/ml)</b>	1255.0	1825.922755	1918.302916	38.391	925.125	1424.210	2125.260	30001.791
<b>AXL (pg/ml)</b>	1255.0	2378.595267	1254.514174	109.440	1567.680	2169.040	2960.785	12247.310
<b>CA-125 (U/ml)</b>	1255.0	6.628457	7.322471	4.608	4.854	4.980	5.355	111.490
<b>CA 15-3 (U/ml)</b>	1255.0	13.647175	14.988504	1.320	6.570	10.930	16.940	336.250
<b>CA19-9 (U/ml)</b>	1255.0	29.803707	272.517374	14.214	16.320	16.470	17.075	9615.690
<b>CD44 (ng/ml)</b>	1255.0	19.321235	11.187244	6.750	11.735	16.440	23.805	148.440
<b>CEA (pg/ml)</b>	1255.0	4333.668324	24266.226984	426.438	580.980	984.800	1752.435	336427.986
<b>CYFRA 21-1 (pg/ml)</b>	1255.0	3178.097793	20975.988332	1816.458	1946.196	1994.874	2035.440	722034.000
<b>DKK1 (ng/ml)</b>	1255.0	1.063578	0.447797	0.350	0.730	0.960	1.295	3.870

```
#2. checking class distribution
import matplotlib.pyplot as plt

label_mapping = {0: 'Normal', 1: 'Cancer'}
label_counts = df_cleaned['Label'].map(label_mapping).value_counts()

label_counts.plot(kind='bar', title='Cancer Type Distribution', color=['skyblue', 'salmon'])
plt.xlabel('Type')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



```

import matplotlib.pyplot as plt
import numpy as np

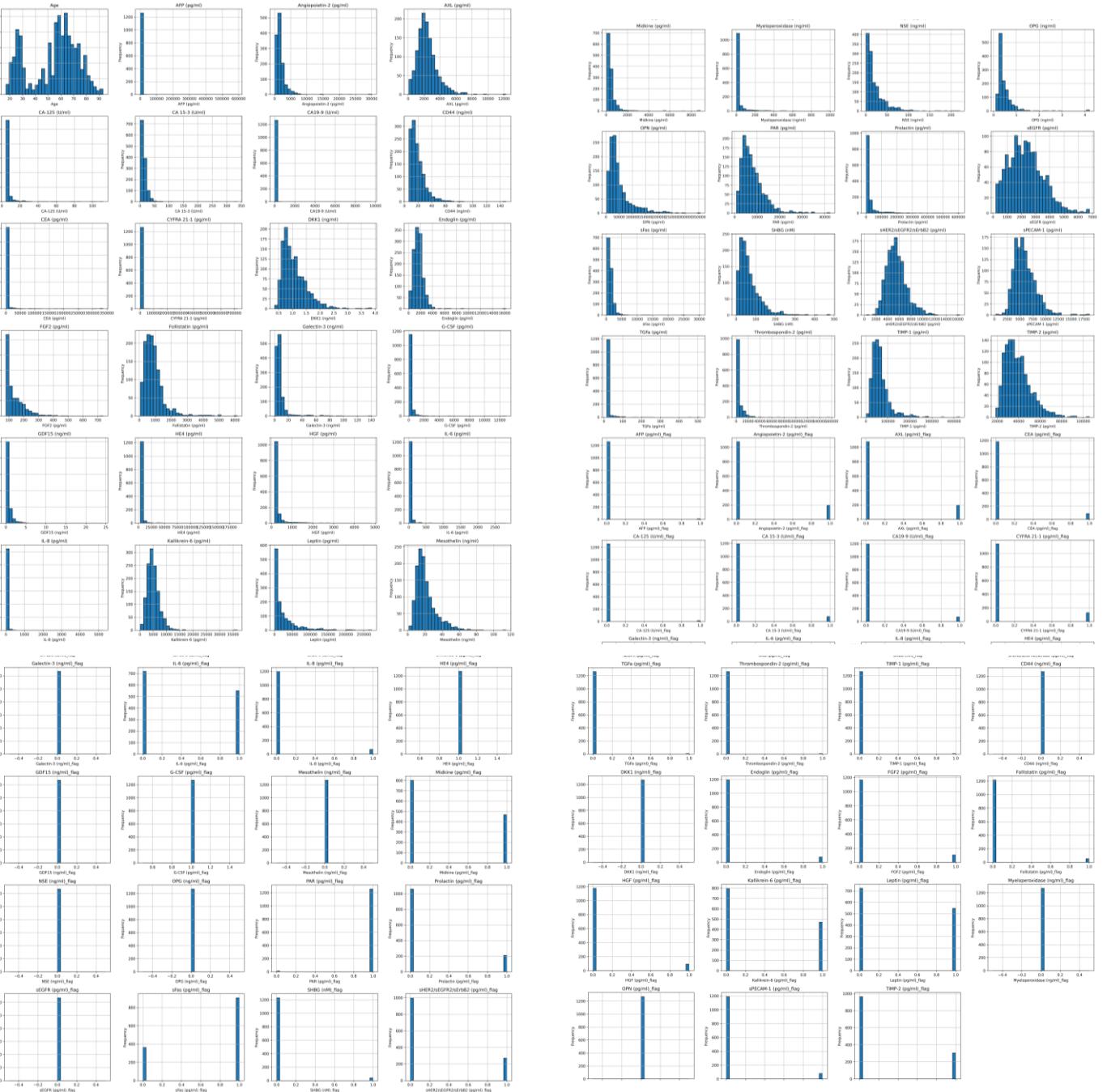
numeric_cols = df_cleaned.select_dtypes(include=['float64', 'int64']).columns.drop('Label')
n_cols = 4
n_rows = int(np.ceil(len(numeric_cols) / n_cols))
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, n_rows * 4))
axes = axes.flatten()

for i, col in enumerate(numeric_cols):
    axes[i].hist(df_cleaned[col].dropna(), bins=30, edgecolor='black')
    axes[i].set_title(f'{col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')
    axes[i].grid(True)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

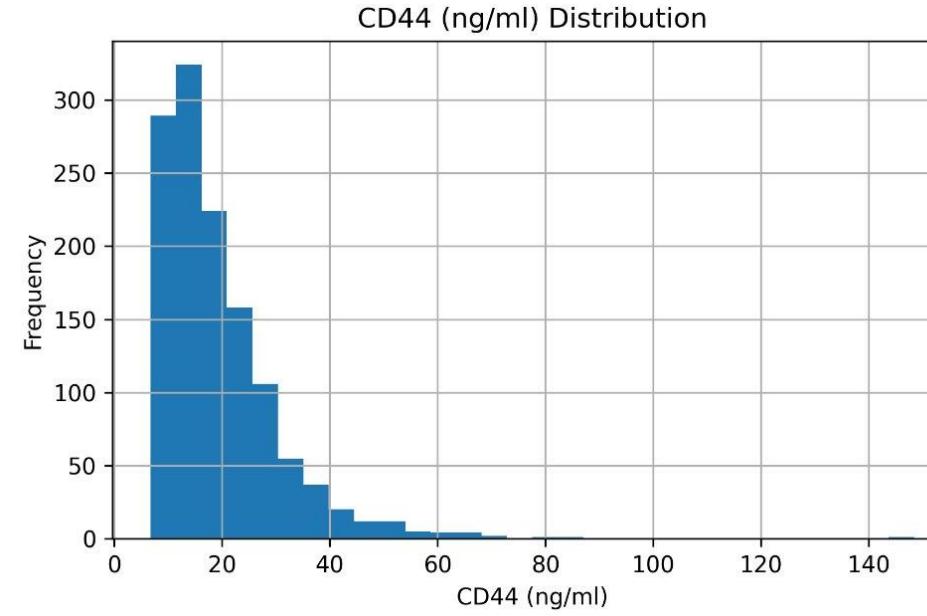
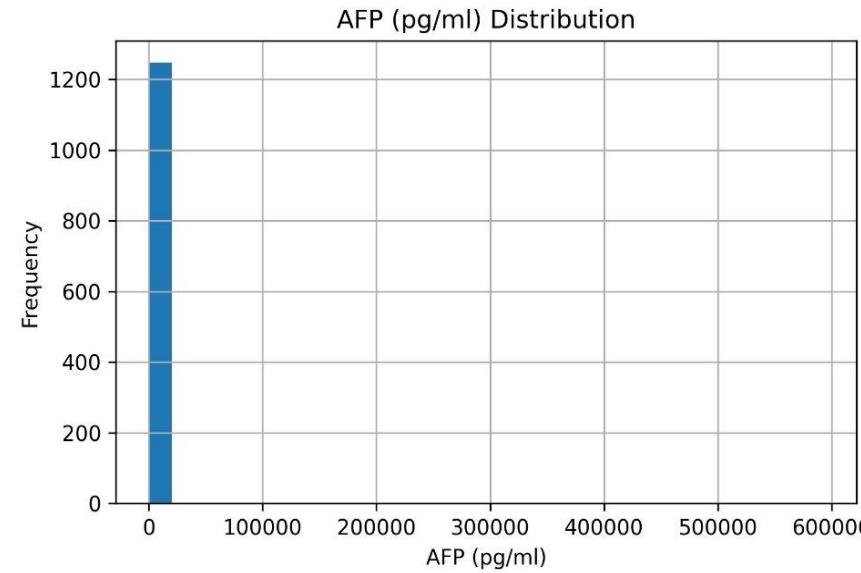
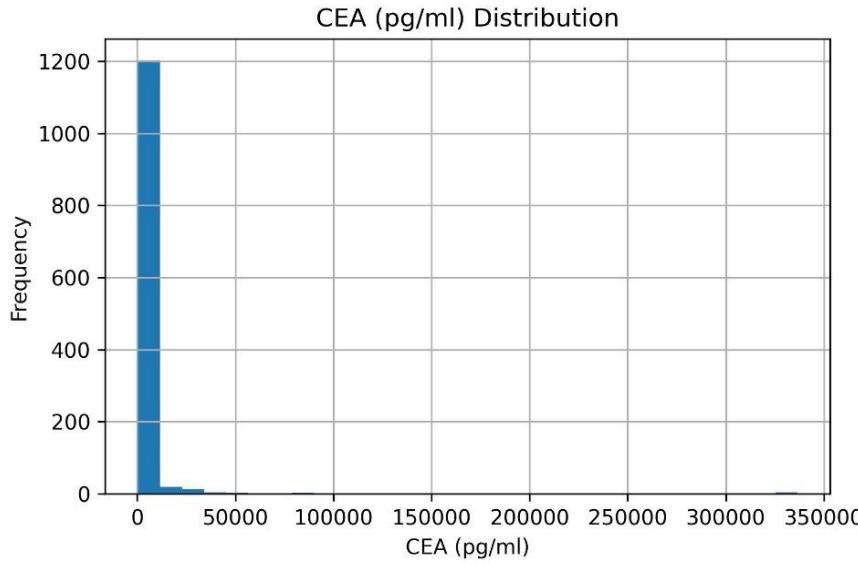
```



```
# 3. Histogram for key biomarkers to visualize the distribution
import matplotlib.pyplot as plt

#key biomarkers
key_biomarkers = [
    'CEA (pg/ml)', 'AFP (pg/ml)', 'CA-125 (U/ml)', 'CD44 (ng/ml)',
    'GDF15 (ng/ml)', 'IL-6 (pg/ml)', 'CA 15-3 (U/ml)',
    'CYFRA 21-1 (pg/ml)', 'Thrombospondin-2 (pg/ml)'
]

for col in key_biomarkers:
    plt.figure(figsize=(6, 4))
    plt.hist(df_cleaned[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.tight_layout()
    filename = f"{col.replace('/', '_').replace(' ', '_)}_histogram.png"
    plt.savefig(filename, dpi=300)
    plt.show()
```

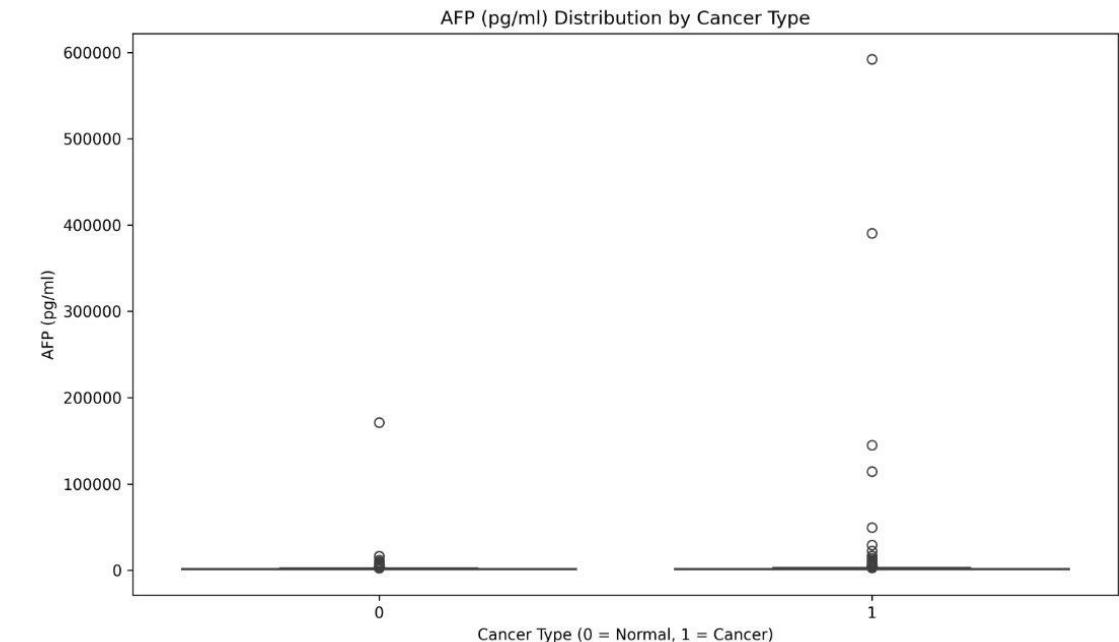
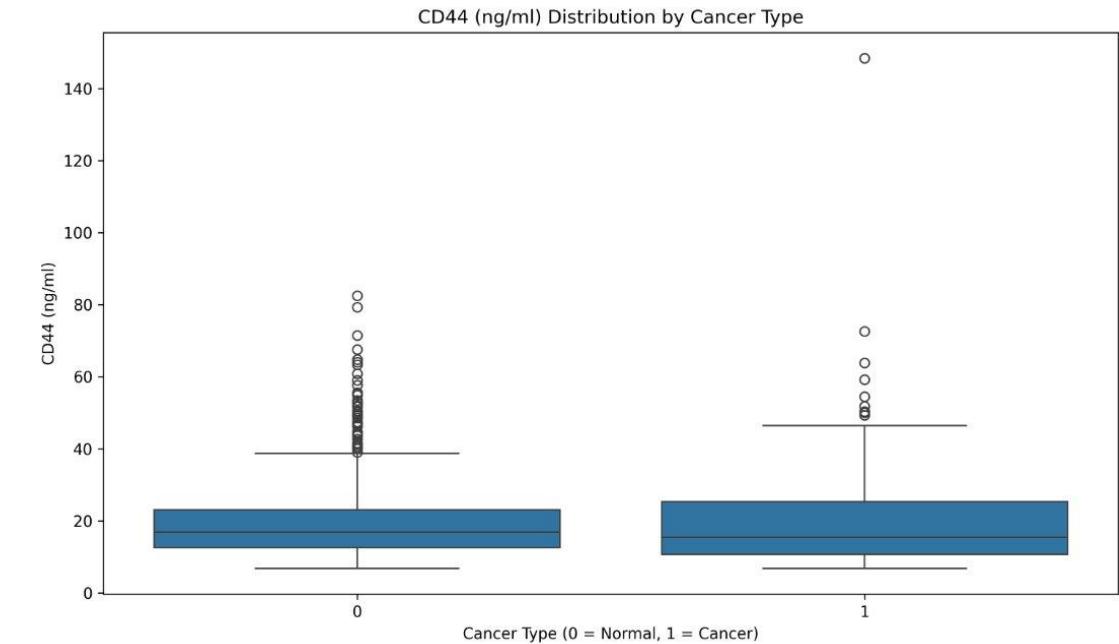
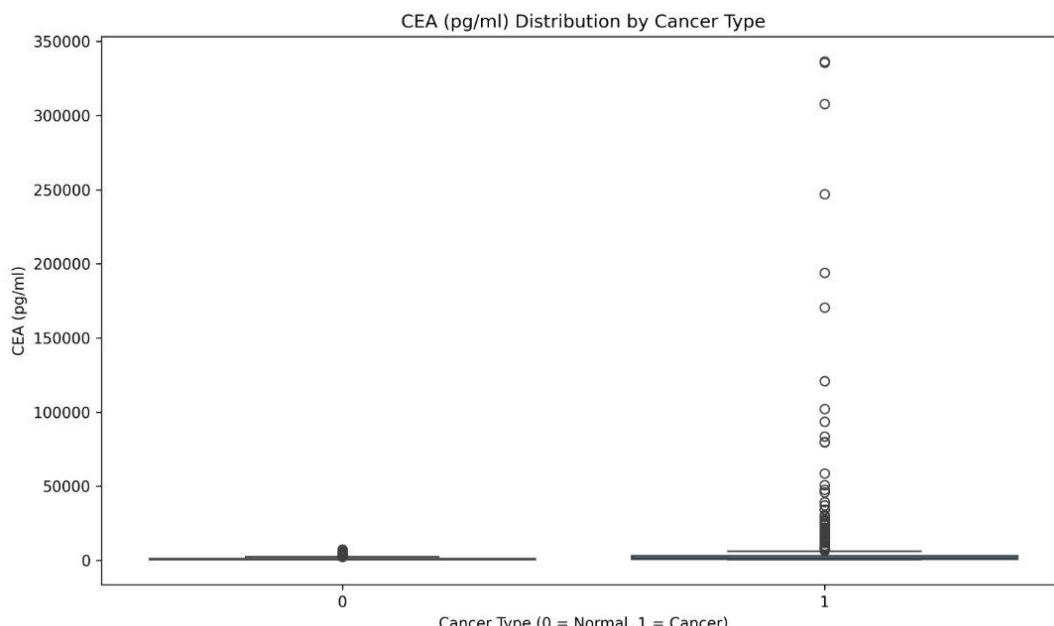


#### # 4. Boxplots of biomarkers by cancer status

```
import matplotlib.pyplot as plt
import seaborn as sns

# key biomarkers
key_biomarkers = [
    'CEA (pg/ml)', 'AFP (pg/ml)', 'CA-125 (U/ml)', 'CD44 (ng/ml)',
    'GDF15 (ng/ml)', 'IL-6 (pg/ml)', 'CA 15-3 (U/ml)',
    'CYFRA 21-1 (pg/ml)', 'Thrombospondin-2 (pg/ml)'
]

# boxplots grouped by label
for col in key_biomarkers:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='Label', y=col, data=df_cleaned, hue='Label', palette='pastel', legend=False)
    plt.title(f"(col) by Cancer Status (0 = Normal, 1 = GI Cancer)")
    plt.xlabel("Label")
    plt.ylabel(col)
    plt.tight_layout()
    filename = f"{col.replace('/', '_').replace(' ', '_)}_boxplot.png"
    plt.savefig(filename, dpi=300)
    plt.show()
```



```

# 5. Statistical comparison between Normal and Cancer
from scipy.stats import ttest_ind

stats_summary = []

for col in biomarkers:
    group0 = df_cleaned[df_cleaned['Label'] == 0][col].dropna()
    group1 = df_cleaned[df_cleaned['Label'] == 1][col].dropna()

    stat, p_value = ttest_ind(group0, group1, equal_var=False)
    stats_summary.append({
        'Biomarker': col,
        'Mean (Normal)': round(group0.mean(), 2),
        'Mean (Cancer)': round(group1.mean(), 2),
        'p-value': round(p_value, 5)
    })

# Display the summary table
import pandas as pd
stats_df = pd.DataFrame(stats_summary)
print(stats_df)

```

	Biomarker	Mean (Normal)	Mean (Cancer)	p-value
0	CEA (pg/ml)	1094.01	10010.17	0.00000
1	AFP (pg/ml)	1676.21	4616.65	0.07017
2	CA-125 (U/ml)	5.52	8.58	0.00000
3	CD44 (ng/ml)	19.46	19.08	0.57499
4	GDF15 (ng/ml)	0.44	1.02	0.00000
5	IL-6 (pg/ml)	8.05	47.93	0.00000
6	CA 15-3 (U/ml)	13.98	13.07	0.38371
7	CYFRA 21-1 (pg/ml)	2069.59	5120.42	0.06132
8	Thrombospondin-2 (pg/ml)	4179.51	5143.08	0.06015

#### # 6. Detect outliers and summarize for key biomarkers

```

import numpy as np
import pandas as pd

#key biomarkers
key_biomarkers = [
    'CEA (pg/ml)', 'AFP (pg/ml)', 'CA-125 (U/ml)', 'CD44 (ng/ml)',
    'GDF15 (ng/ml)', 'IL-6 (pg/ml)', 'CA 15-3 (U/ml)',
    'CYFRA 21-1 (pg/ml)', 'Thrombospondin-2 (pg/ml)'
]

outlier_summary = []

for col in key_biomarkers:
    for label in [0, 1]:
        group = df_cleaned[df_cleaned['Label'] == label][col].dropna()
        Q1 = np.percentile(group, 25)
        Q3 = np.percentile(group, 75)
        IQR = Q3 - Q1
        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR
        outliers = group[(group < lower) | (group > upper)]

        outlier_summary.append({
            'Biomarker': col,
            'Label': 'Normal' if label == 0 else 'GI Cancer',
            'Num Outliers': len(outliers),
            'Outlier Min': round(outliers.min(), 2) if not outliers.empty else None,
            'Outlier Max': round(outliers.max(), 2) if not outliers.empty else None
        })

outlier_df = pd.DataFrame(outlier_summary)
print(outlier_df)
outlier_df.to_csv("outlier_summary.csv", index=False)

```

Biomarker	Label	Num Outliers	Outlier Min	Outlier Max
CEA (pg/ml)	Normal	39	2575.2	7377.64
CEA (pg/ml)	GI Cancer	69	6637.5	336427.99
AFP (pg/ml)	Normal	85	2811.58	171255.75
AFP (pg/ml)	GI Cancer	66	3066.13	592312.72
CA-125 (U/ml)	Normal	167	5.24	33.07
CA-125 (U/ml)	GI Cancer	70	9.88	111.49
CD44 (ng/ml)	Normal	45	38.62	82.53
CD44 (ng/ml)	GI Cancer	9	49.31	148.44
GDF15 (ng/ml)	Normal	56	1.08	8.56
GDF15 (ng/ml)	GI Cancer	39	2.27	24.29
IL-6 (pg/ml)	Normal	156	6.9	356.64
IL-6 (pg/ml)	GI Cancer	61	92.08	2818.46
CA 15-3 (U/ml)	Normal	32	36.17	102.98
CA 15-3 (U/ml)	GI Cancer	21	29.75	336.25
CYFRA 21-1 (pg/ml)	Normal	108	1816.46	13499.0
CYFRA 21-1 (pg/ml)	GI Cancer	66	4541.87	722034.0
Thrombospondin-2 (pg/ml)	Normal	76	10756.08	131014.81
Thrombospondin-2 (pg/ml)	GI Cancer	52	12973.51	157461.07

```

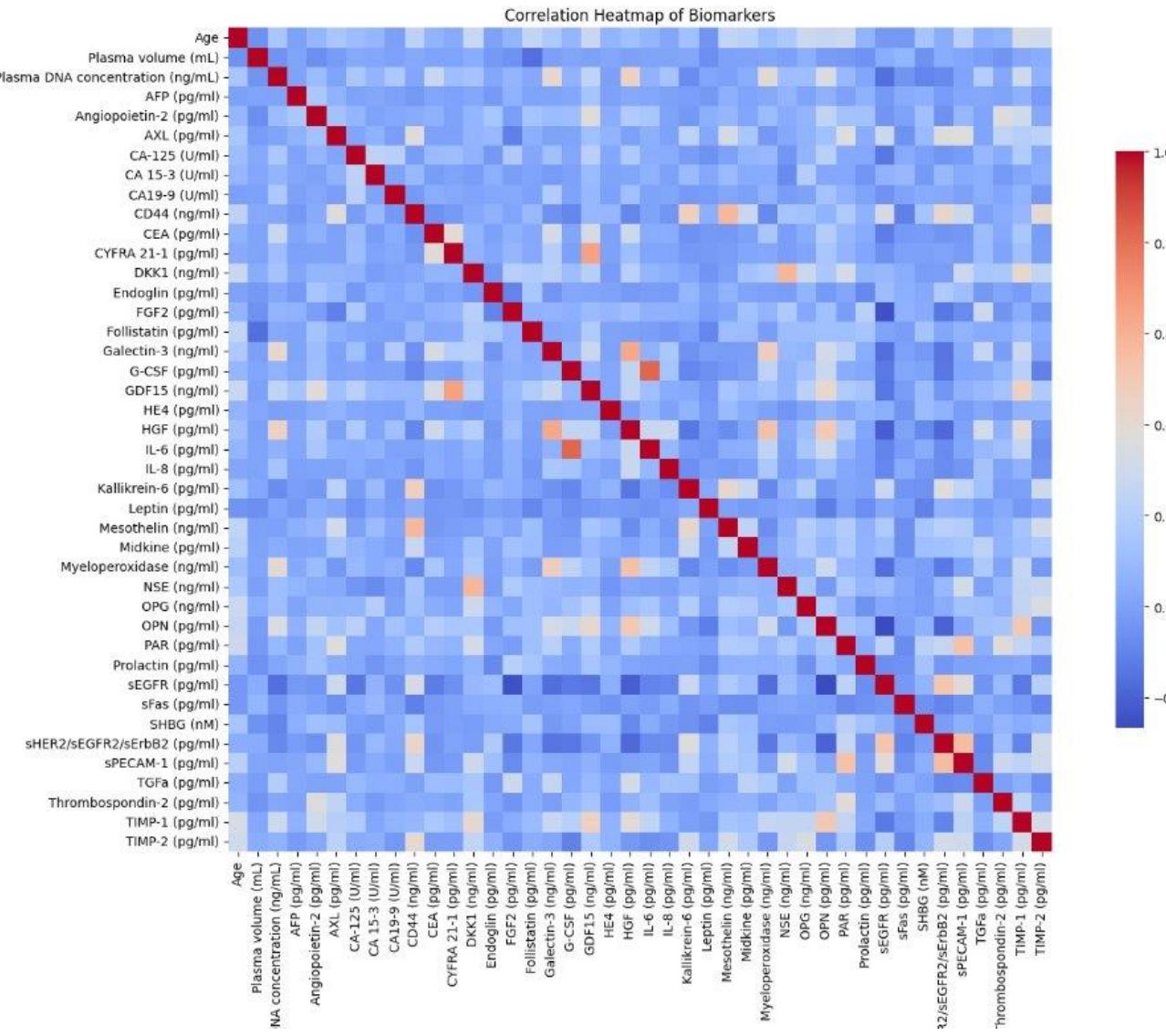
# 7. Correlation heatmap of biomarkers
import matplotlib.pyplot as plt
import seaborn as sns

numeric_features = df_cleaned.select_dtypes(include=['float64', 'int64']).drop(columns=['Label'])

corr_matrix = numeric_features.corr()

#heatmap
plt.figure(figsize=(16, 12))
sns.heatmap(corr_matrix, cmap='coolwarm', annot=False, fmt=".2f", square=True, cbar_kws={"shrink": 0.7})
plt.title("Correlation Heatmap of Biomarkers")
plt.tight_layout()
plt.savefig("correlation_heatmap.png", dpi=300)
plt.show()

```



```
# 8. Grouped key feature averages with demographic insights
```

```
# A. Mean of Key Biomarkers by Sex
```

```
key_biomarkers = [  
    'CEA (pg/ml)', 'AFP (pg/ml)', 'CA-125 (U/ml)', 'CD44 (ng/ml)',  
    'GDF15 (ng/ml)', 'IL-6 (pg/ml)', 'CA 15-3 (U/ml)',  
    'CYFRA 21-1 (pg/ml)', 'Thrombospondin-2 (pg/ml)'  
]
```

```
mean_by_sex = df_cleaned.groupby("Sex")[key_biomarkers].mean().round(2)
```

```
print("Mean biomarker levels by Sex:")
```

```
print(mean_by_sex)
```

```
Mean biomarker levels by Sex:
```

```
    CEA (pg/ml)  AFP (pg/ml)  CA-125 (U/ml)  CD44 (ng/ml)  GDF15 (ng/ml)  \
```

```
Sex
```

```
Female    3708.55      2330.50       7.03     19.12      0.67
```

```
Male      4905.25      3113.07      6.36     19.41      0.64
```

```
    IL-6 (pg/ml)  CA 15-3 (U/ml)  CYFRA 21-1 (pg/ml)  \
```

```
Sex
```

```
Female     15.84        12.11      3944.94
```

```
Male      28.21        14.90      2603.49
```

```
    Thrombospondin-2 (pg/ml)
```

```
Sex
```

```
Female    4668.85
```

```
Male      4407.40
```

```
# C. Correlation of Age with Biomarkers
```

```
age_corr = df_cleaned[["Age"] + key_biomarkers].corr().round(2)[['Age']].drop('Age')
```

```
print("Correlation of Age with biomarkers:")
```

```
print(age_corr)
```

```
Correlation of Age with biomarkers:
```

```
CEA (pg/ml)          0.07
```

```
AFP (pg/ml)          0.00
```

```
CA-125 (U/ml)        0.11
```

```
CD44 (ng/ml)         0.23
```

```
GDF15 (ng/ml)        0.28
```

```
IL-6 (pg/ml)          0.08
```

```
CA 15-3 (U/ml)        0.08
```

```
CYFRA 21-1 (pg/ml)    0.03
```

```
Thrombospondin-2 (pg/ml) 0.07
```

```
Name: Age, dtype: float64
```

```
#B. Mean of Key Biomarkers by Race
```

```
mean_by_race = df_cleaned.groupby("Race")[key_biomarkers].mean().round(2)
```

```
print("Mean biomarker levels by Race:")
```

```
print(mean_by_race)
```

```
Mean biomarker levels by Race:
```

Race	CEA (pg/ml)	AFP (pg/ml)	CA-125 (U/ml)	CD44 (ng/ml)
Asian	16812.84	5958.07	8.99	17.14
Black	1232.25	1522.12	5.26	15.15
Black/Hispanic	1062.46	1250.15	5.20	14.84
Caucasian	2939.40	2561.94	6.90	21.50
Caucasian/Hispanic	1013.29	1180.23	5.44	14.42
Hispanic	1186.77	1000.16	5.48	18.08
Other	504.11	1928.40	4.98	25.24
Unknown	931.86	2310.46	5.26	19.11

```
GDF15 (ng/ml)  IL-6 (pg/ml)  CA 15-3 (U/ml)  \
```

```
Race
```

Race	GDF15 (ng/ml)	IL-6 (pg/ml)	CA 15-3 (U/ml)
Asian	1.20	72.64	12.29
Black	0.24	6.44	15.33
Black/Hispanic	0.18	3.99	9.02
Caucasian	0.76	18.45	14.47
Caucasian/Hispanic	0.14	4.73	10.01
Hispanic	0.17	11.83	12.19
Other	0.35	3.84	9.40
Unknown	0.41	7.43	12.46

```
CYFRA 21-1 (pg/ml)  Thrombospondin-2 (pg/ml)
```

```
Race
```

Race	CYFRA 21-1 (pg/ml)	Thrombospondin-2 (pg/ml)
Asian	7849.40	4333.06
Black	2056.87	3599.83
Black/Hispanic	2008.97	2406.38
Caucasian	2624.14	4546.46
Caucasian/Hispanic	2007.82	3920.96
Hispanic	1973.03	2490.55
Other	2035.44	2245.65
Unknown	1975.28	6749.04

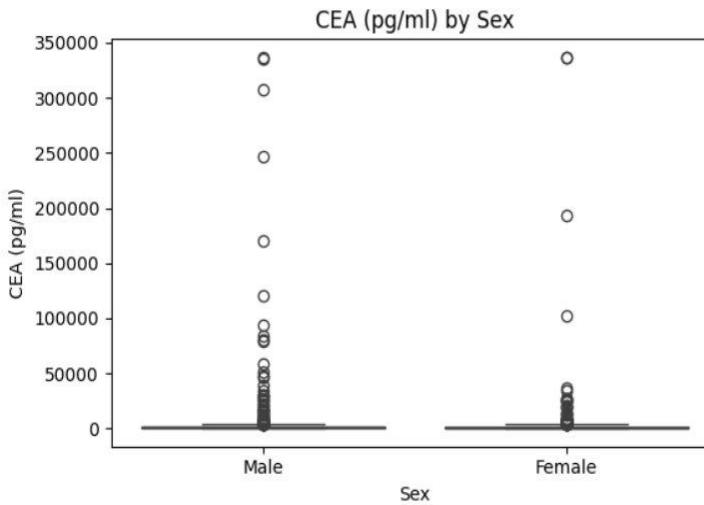
```

# Visualizing demographic insights
# D. Box plot of key biomarkers by Sex and Race
import matplotlib.pyplot as plt
import seaborn as sns

key_biomarkers = [
    'CEA (pg/ml)', 'AFP (pg/ml)', 'CA-125 (U/ml)', 'CD44 (ng/ml)',
    'GDF15 (ng/ml)', 'IL-6 (pg/ml)', 'CA 15-3 (U/ml)',
    'CYFRA 21-1 (pg/ml)', 'Thrombospondin-2 (pg/ml)'
]

# Boxplots by Sex
for col in key_biomarkers:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x="Sex", y=col, data=df_cleaned)
    plt.title(f"{col} by Sex")
    plt.xlabel("Sex")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()

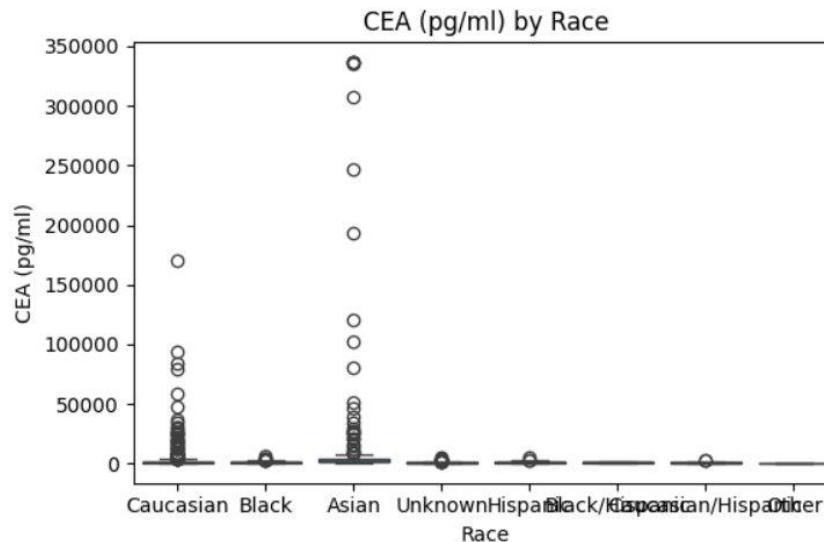
```



```

# Boxplots by Race
for col in key_biomarkers:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x="Race", y=col, data=df_cleaned)
    plt.title(f"{col} by Race")
    plt.xlabel("Race")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()

```



```
# E. Scatterplots with Age
for col in key_biomarkers:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(x="Age", y=col, data=df_cleaned)
    plt.title(f"{col} vs Age")
    plt.xlabel("Age")
    plt.ylabel(col)
    plt.tight_layout()
```

---

```
from scipy.stats import shapiro
import pandas as pd

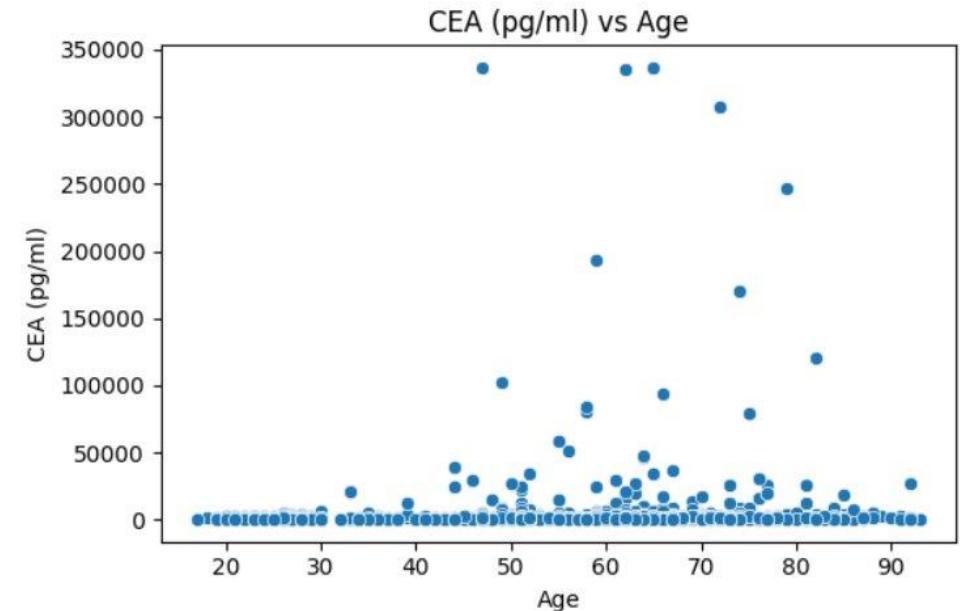
normality_results = []

for col in key_biomarkers:
    stat, p_value = shapiro(df_cleaned[col].dropna())
    normality_results.append({
        'Biomarker': col,
        'Shapiro-Wilk p-value': round(p_value, 5),
        'Normality': 'Not Normal' if p_value < 0.05 else 'Normal'
    })

normality_df = pd.DataFrame(normality_results)
print(normality_df)
normality_df.to_csv("Normality_test_results.csv", index=False)
```

---

	Biomarker	Shapiro-Wilk p-value	Normality
0	CEA (pg/ml)	0.0	Not Normal
1	AFP (pg/ml)	0.0	Not Normal
2	CA-125 (U/ml)	0.0	Not Normal
3	CD44 (ng/ml)	0.0	Not Normal
4	GDF15 (ng/ml)	0.0	Not Normal
5	IL-6 (pg/ml)	0.0	Not Normal
6	CA 15-3 (U/ml)	0.0	Not Normal
7	CYFRA 21-1 (pg/ml)	0.0	Not Normal
8	Thrombospondin-2 (pg/ml)	0.0	Not Normal



```

import pandas as pd

df_cleaned = pd.read_csv("Dropped_rows_dataset.csv")
clinical_thresholds = {
    'AFP (pg/ml)': 20000,
    'Angiopoietin-2 (pg/ml)': 4141,
    'AXL (pg/ml)': 1243,
    'CA-125 (U/ml)': 35,
    'CA 15-3 (U/ml)': 30,
    'CA19-9 (U/ml)': 37,
    'CYFRA 21-1 (pg/ml)': 3500,
    'CEA (pg/ml)': 5000,
    'DKK1 (ng/ml)': 2.03 * 1000, # ng/ml to pg/ml
    'Endoglin (pg/ml)': 2000,
    'FGF2 (pg/ml)': 10,
    'Follistatin (pg/ml)': 4700,
    'Galectin-3 (ng/ml)': 22.1 * 1000,
    'G-CSF (pg/ml)': 15,
    'GDF15 (ng/ml)': 1200 * 1000,
    'HE4 (pg/ml)': 3500,
    'HGF (pg/ml)': 1000,
    'IL-6 (pg/ml)': 5,
    'IL-8 (pg/ml)': 57.7,
    'Kallikrein-6 (pg/ml)': 1000,
    'MesotheLIN (ng/ml)': 2 * 1000,
    'Midkine (pg/ml)': 400,
    'Myeloperoxidase (ng/ml)': 220 * 1000,
    'NSE (ng/ml)': 16.3 * 1000,
    'OPG (ng/ml)': 0.2 * 1000,
    'OPN (pg/ml)': 200000,
    'sFas (pg/ml)': 150,
    'sHER2/sEGFR2/sErbB2 (pg/ml)': 15000,
    'TGf $\alpha$  (pg/ml)': 100,
    'Thrombospondin-2 (pg/ml)': 45000,
    'TIMP-1 (pg/ml)': 126000,
}

```

```

#Sex-specific thresholds
sex_specific_thresholds = {
    'Leptin (pg/ml)': {'Male': 15200, 'Female': 12500},
    'Prolactin (pg/ml)': {'Male': 18000, 'Female': 25000},
    'SHBG (nM)': {'Male': 54, 'Female': 145},
}

#Convert biomarker columns to numeric
for col in df_cleaned.columns:
    df[col] = pd.to_numeric(df_cleaned[col], errors='coerce')

#clinical threshold flags
for biomarker, threshold in clinical_thresholds.items():
    if biomarker in df_cleaned.columns:
        df_cleaned[f'{biomarker}_flag'] = (df_cleaned[biomarker] > threshold).astype(int)

#sex-specific thresholds
for biomarker, thresholds in sex_specific_thresholds.items():
    if biomarker in df_cleaned.columns and 'Sex' in df_cleaned.columns:
        df_cleaned[f'{biomarker}_flag'] = df_cleaned.apply(
            lambda row: int(row[biomarker] > thresholds.get(row['Sex'], float('inf'))), axis=1
        )

#Remove any mistakenly flagged non-biomarker
non_biomarker_flags = ['Sex_flag', 'Sample_ID_flag']
df_cleaned.drop(columns=[col for col in non_biomarker_flags if col in df_cleaned.columns], inplace=True)

df_cleaned.to_csv("threshold_flagged_dataset_final.csv", index=False)
df_cleaned.head()

```

Label	AFP (pg/ml)_flag	angiopoietin-2 (pg/ml)_flag	AXL (pg/ml)_flag	CA-125 (U/ml)_flag	CA 15-3 (U/ml)_flag	CA19-9 (U/ml)_flag	CYFRA 21-1 (pg/ml)_flag	CEA (pg/ml)_flag	DKK1 (ng/ml)_flag	Endoglin (pg/ml)_flag
1	0	1	1	0	0	0	0	0	0	1
1	0	1	1	0	0	1	0	1	0	1
1	0	0	1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	1
1	0	0	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0
1	0	0	1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1

```

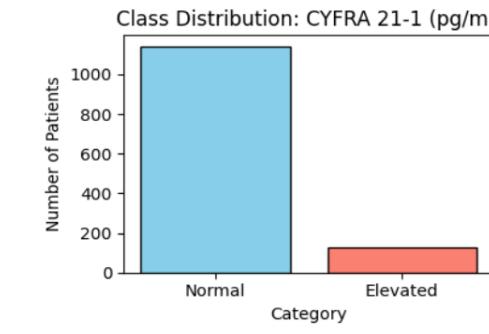
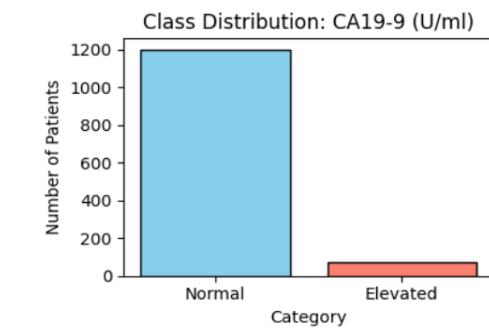
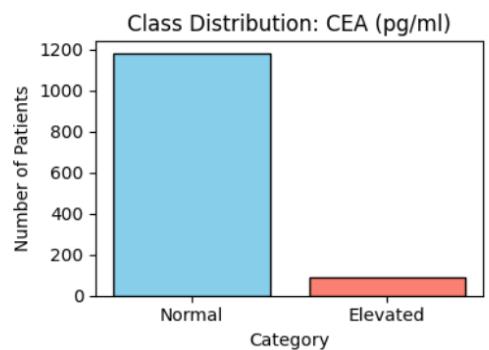
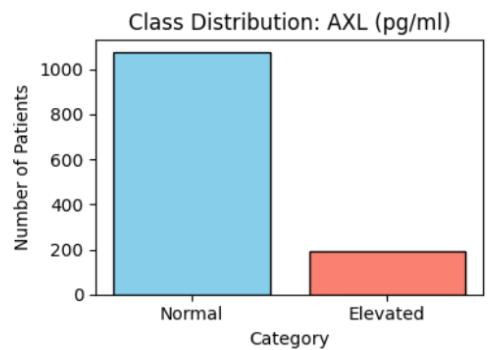
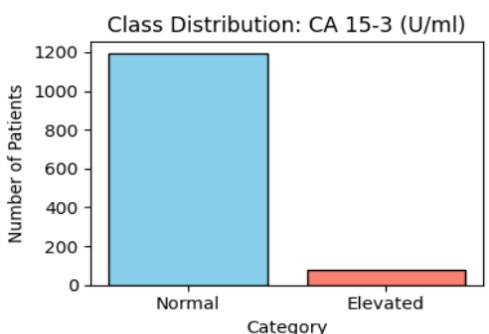
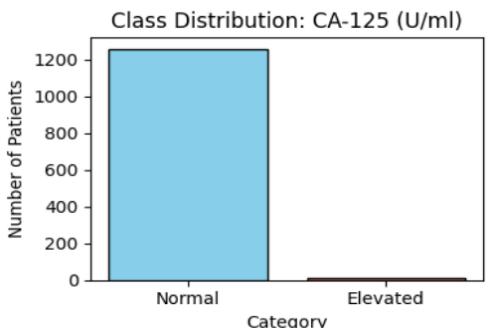
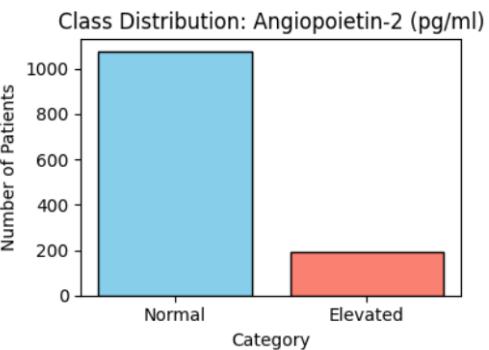
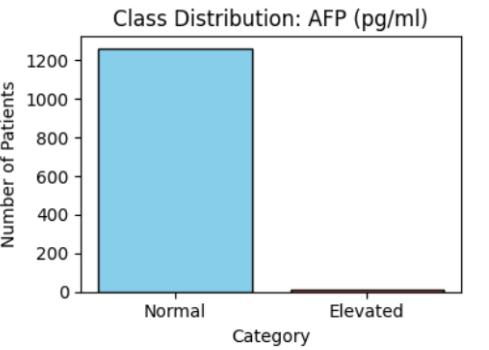
import matplotlib.pyplot as plt

flag_columns = [col for col in df_cleaned.columns if col.endswith('_flag')]

for biomarker_flag in flag_columns:
    plt.figure(figsize=(4, 3))
    counts = df_cleaned[biomarker_flag].value_counts().sort_index()

    plt.bar(counts.index, counts.values, color=['skyblue', 'salmon'], edgecolor='black')
    plt.xticks([0, 1], labels=["Normal", "Elevated"])
    plt.xlabel("Category")
    plt.ylabel("Number of Patients")
    plt.title(f"Class Distribution: {biomarker_flag.replace('_flag', '')}")
    plt.tight_layout()
    plt.show()

```



```

from scipy.stats import chi2_contingency
import pandas as pd

flag_columns = [col for col in df_cleaned.columns if col.endswith('_flag') and col != 'Sex_flag']
chi2_results = []

for col in flag_columns:
    contingency_table = pd.crosstab(df_cleaned[col], df_cleaned['Label'])

    if contingency_table.shape == (2, 2): # Ensure it's a valid 2x2 table
        chi2, p, dof, expected = chi2_contingency(contingency_table)
        chi2_results.append({
            'Biomarker': col,
            'p_value': round(p, 5),
            'Significant': 'Yes' if p < 0.05 else 'No'
        })

chi2_df = pd.DataFrame(chi2_results).sort_values(by='p_value')
chi2_df.to_csv("chi_square_results.csv", index=False)
print(chi2_df.head())

```

	Biomarker	p_value	Significant
11	IL-6 (pg/ml)_flag	0.0	Yes
20	Leptin (pg/ml)_flag	0.0	Yes
19	TIMP-1 (pg/ml)_flag	0.0	Yes
15	OPN (pg/ml)_flag	0.0	Yes
5	CA19-9 (U/ml)_flag	0.0	Yes

Biomarker	p_value	Significant
IL-6 (pg/ml)_flag	0.0	Yes
Leptin (pg/ml)_flag	0.0	Yes
TIMP-1 (pg/ml)_flag	0.0	Yes
OPN (pg/ml)_flag	0.0	Yes
CA19-9 (U/ml)_flag	0.0	Yes
CYFRA 21-1 (pg/ml)_flag	0.0	Yes
CEA (pg/ml)_flag	0.0	Yes
Midkine (pg/ml)_flag	0.0	Yes
IL-8 (pg/ml)_flag	0.0	Yes
HGF (pg/ml)_flag	0.0	Yes
Prolactin (pg/ml)_flag	0.0	Yes
angiopoietin-2 (pg/ml)_flag	1e-05	Yes
SHBG (nM)_flag	2e-05	Yes
CA-125 (U/ml)_flag	2e-05	Yes
AFP (pg/ml)_flag	0.00886	Yes
Kallikrein-6 (pg/ml)_flag	0.02145	Yes
CA 15-3 (U/ml)_flag	0.05517	No
TGFa (pg/ml)_flag	0.09993	No
AXL (pg/ml)_flag	0.14044	No
Endoglin (pg/ml)_flag	0.8987	No
SFR2/sErbB2 (pg/ml)_flag	1.0	No
fibronectin-2 (pg/ml)_flag	1.0	No
Follistatin (pg/ml)_flag	1.0	No

```

import pandas as pd
from scipy.stats import mannwhitneyu

df = pd.read_csv("threshold_flagged_dataset_final.csv")
continuous_biomarkers = [
    'CD44 (ng/ml)',
    'PAR (pg/ml)',
    'sEGFR (pg/ml)',
    'sPECAM-1 (pg/ml)',
    'TIMP-2 (pg/ml)'
]

mannwhitney_results = []

for biomarker in continuous_biomarkers:
    if biomarker in df.columns:
        group_0 = df[df['Label'] == 0][biomarker].dropna()
        group_1 = df[df['Label'] == 1][biomarker].dropna()

        stat, p = mannwhitneyu(group_0, group_1, alternative='two-sided')
        mannwhitney_results.append({
            'Biomarker': biomarker,
            'Mann-Whitney U': round(stat, 2),
            'p-value': round(p, 5),
            'Significant': 'Yes' if p < 0.05 else 'No'
        })

mannwhitney_df = pd.DataFrame(mannwhitney_results)
mannwhitney_df = mannwhitney_df.sort_values(by='p-value')
mannwhitney_df.to_csv("mannwhitney_results.csv", index=False)

print(mannwhitney_df)

```

	Biomarker	Mann-Whitney U	p-value	Significant
2	sEGFR (pg/ml)	248907.0	0.00000	Yes
1	PAR (pg/ml)	159440.5	0.00149	Yes
4	TIMP-2 (pg/ml)	197661.0	0.00187	Yes
0	CD44 (ng/ml)	191261.5	0.03966	Yes
3	sPECAM-1 (pg/ml)	187833.5	0.13531	No

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("threshold_flagged_dataset_final.csv")

y = df["Label"]

flagged_features = [col for col in df.columns if col.endswith('_flag')]
continuous_features = [
    "CD44 (ng/ml)", "PAR (pg/ml)", "sEGFR (pg/ml)",
    "sPECAM-1 (pg/ml)", "TIMP-2 (pg/ml)"
]

X_categorical = df[flagged_features]
X_continuous = df[continuous_features]
X_full = pd.concat([X_categorical, X_continuous], axis=1)

#Splitting before scaling to avoid data leakage
X_train, X_test, y_train, y_test = train_test_split(
    X_full, y, test_size=0.2, stratify=y, random_state=42
)

scaler = StandardScaler()
X_train_cont = scaler.fit_transform(X_train[continuous_features])
X_test_cont = scaler.transform(X_test[continuous_features])

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[continuous_features] = X_train_cont
X_test_scaled[continuous_features] = X_test_cont

X_train_scaled.to_csv("X_train_prepared.csv", index=False)
X_test_scaled.to_csv("X_test_prepared.csv", index=False)
y_train.to_csv("y_train.csv", index=False)
y_test.to_csv("y_test.csv", index=False)
```

```

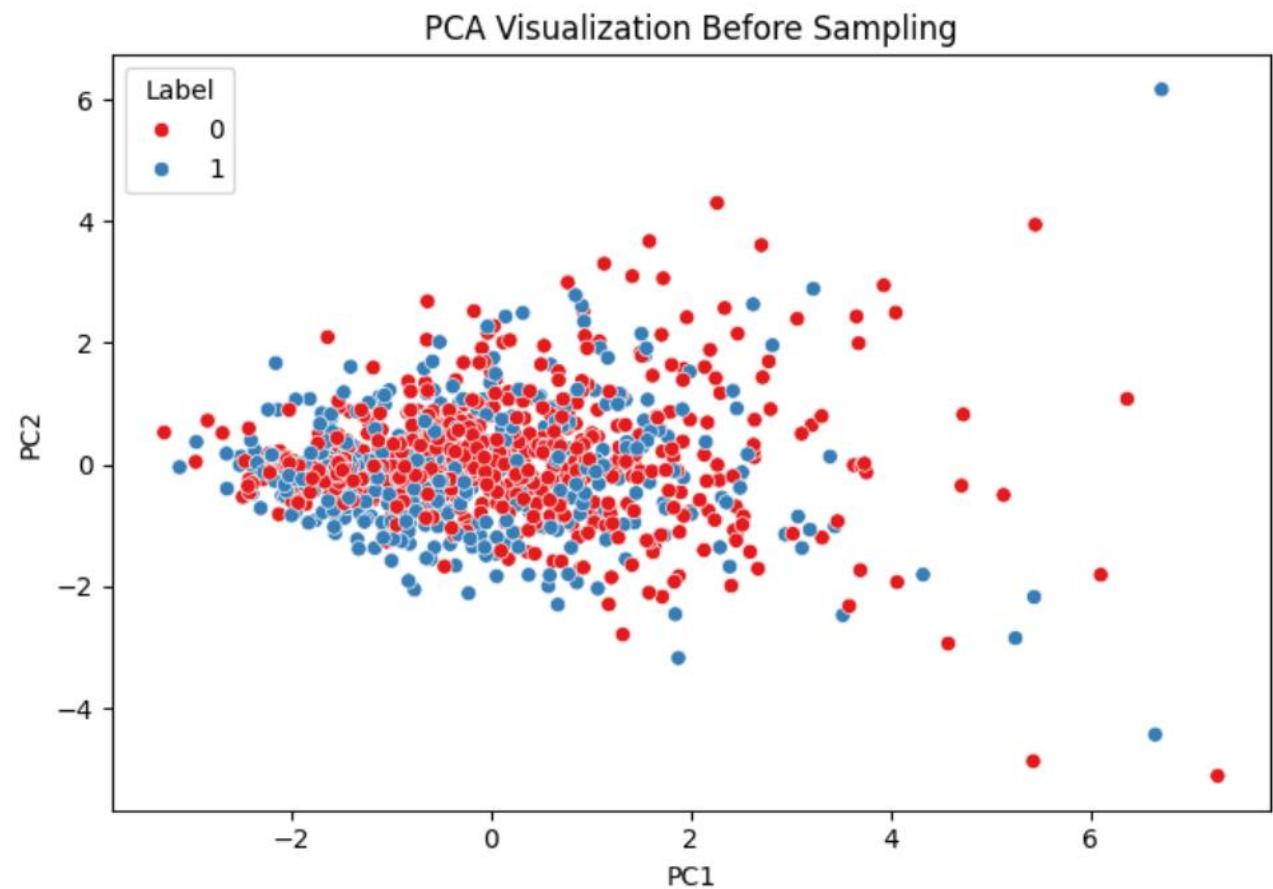
import seaborn as sns
from sklearn.decomposition import PCA

X_pca_pre = PCA(n_components=2).fit_transform(X_train_scaled)

pca_pre_df = pd.DataFrame(X_pca_pre, columns=["PC1", "PC2"])
pca_pre_df["Label"] = y_train.reset_index(drop=True)

plt.figure(figsize=(7, 5))
sns.scatterplot(x="PC1", y="PC2", hue="Label", data=pca_pre_df, palette="Set1")
plt.title("PCA Visualization Before Sampling")
plt.tight_layout()
plt.show()

```

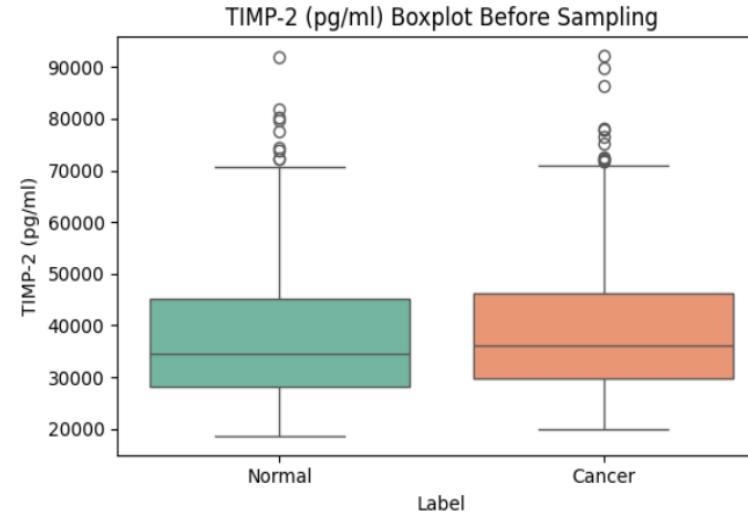
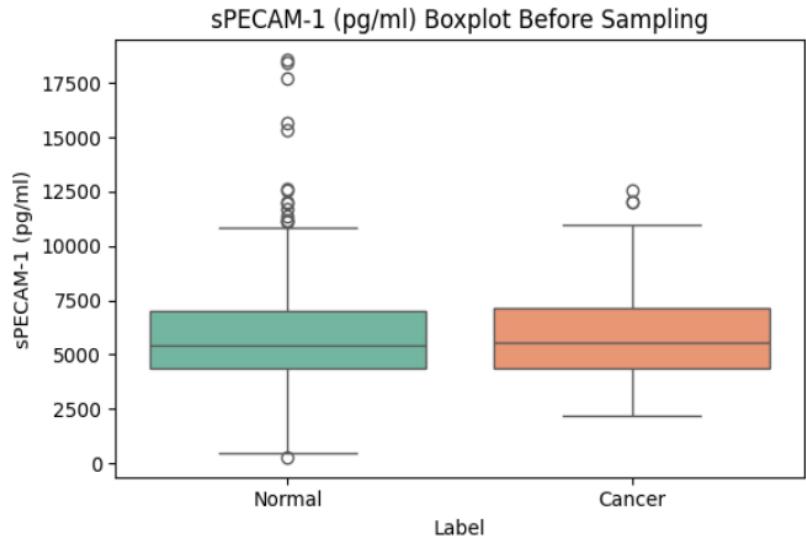
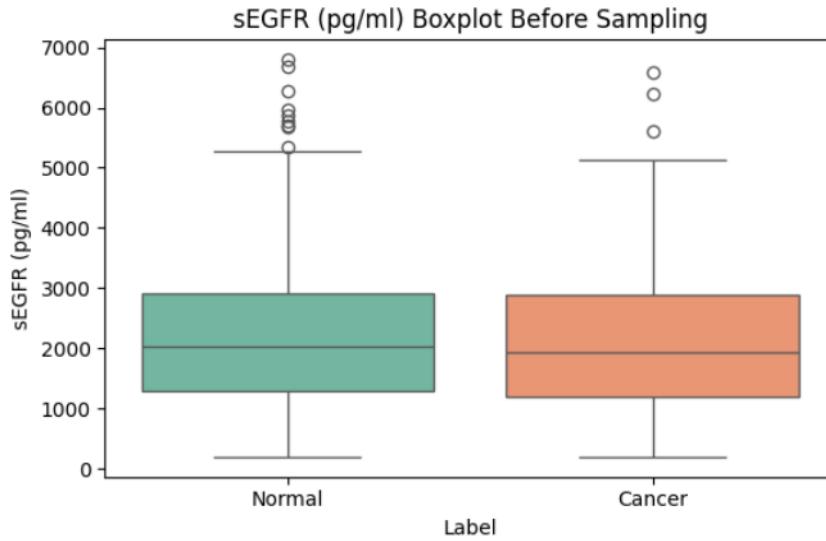
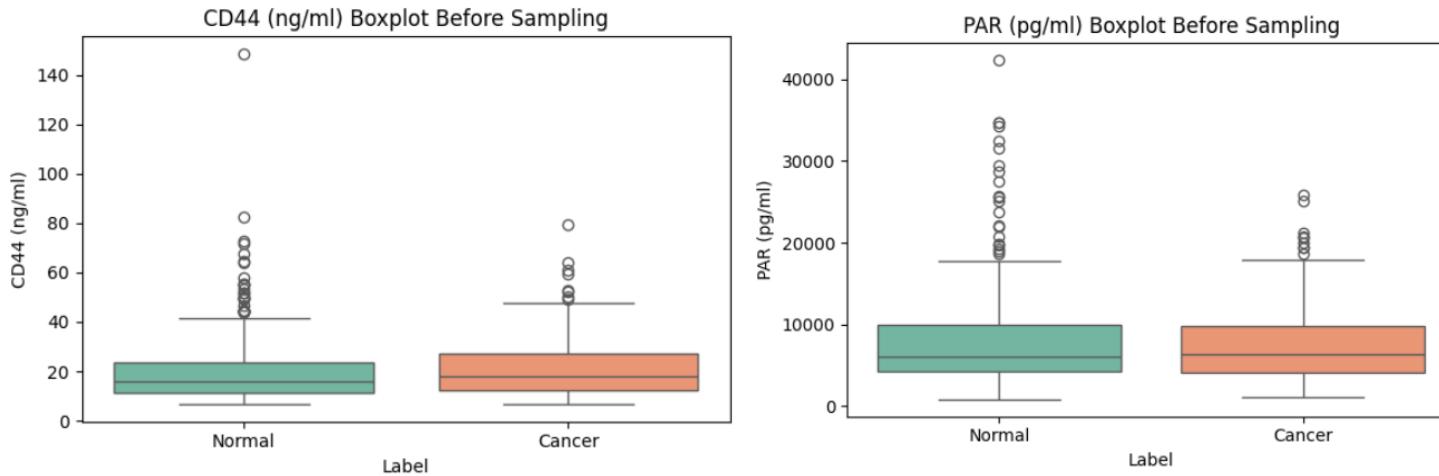


```

boxplot_df = X_train[["CD44 (ng/ml)", "PAR (pg/ml)", "sEGFR (pg/ml)",
                     "sPECAM-1 (pg/ml)", "TIMP-2 (pg/ml)"]].copy()
boxplot_df["Label"] = y_train.reset_index(drop=True)

for col in boxplot_df.columns[:-1]:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='Label', y=col, data=boxplot_df, palette='Set2')
    plt.title(f"{col} Boxplot Before Sampling")
    plt.xlabel("Label")
    plt.xticks([0, 1], ['Normal', 'Cancer'])
    plt.tight_layout()
    plt.show()

```



```

from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.utils import resample
from collections import Counter
import pandas as pd

# 1. SMOTE
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

# 2. Random Oversampling
ros = RandomOverSampler(random_state=42)
X_train_ros, y_train_ros = ros.fit_resample(X_train_scaled, y_train)

# 3. Random Undersampling
rus = RandomUnderSampler(random_state=42)
X_train_rus, y_train_rus = rus.fit_resample(X_train_scaled, y_train)

# 4. Bootstrapping the minority class
df_train = X_train_scaled.copy()
df_train['Label'] = y_train.values

df_majority = df_train[df_train.Label == 0]
df_minority = df_train[df_train.Label == 1]

df_minority_boot = resample(
    df_minority,
    replace=True,
    n_samples=len(df_majority),
    random_state=42
)

df_bootstrap = pd.concat([df_majority, df_minority_boot])
X_train_bootstrap = df_bootstrap.drop(columns=['Label'])
y_train_bootstrap = df_bootstrap['Label']

```

```

# Summary of class balance
print("Original:", Counter(y_train))
print("SMOTE:", Counter(y_train_smote))
print("Random Oversampling:", Counter(y_train_ros))
print("Random Undersampling:", Counter(y_train_rus))
print("Bootstrapped:", Counter(y_train_bootstrap))

```

```

Original: Counter({0: 627, 1: 365})
SMOTE: Counter({0: 627, 1: 627})
Random Oversampling: Counter({0: 627, 1: 627})
Random Undersampling: Counter({0: 365, 1: 365})
Bootstrapped: Counter({0: 627, 1: 627})

```

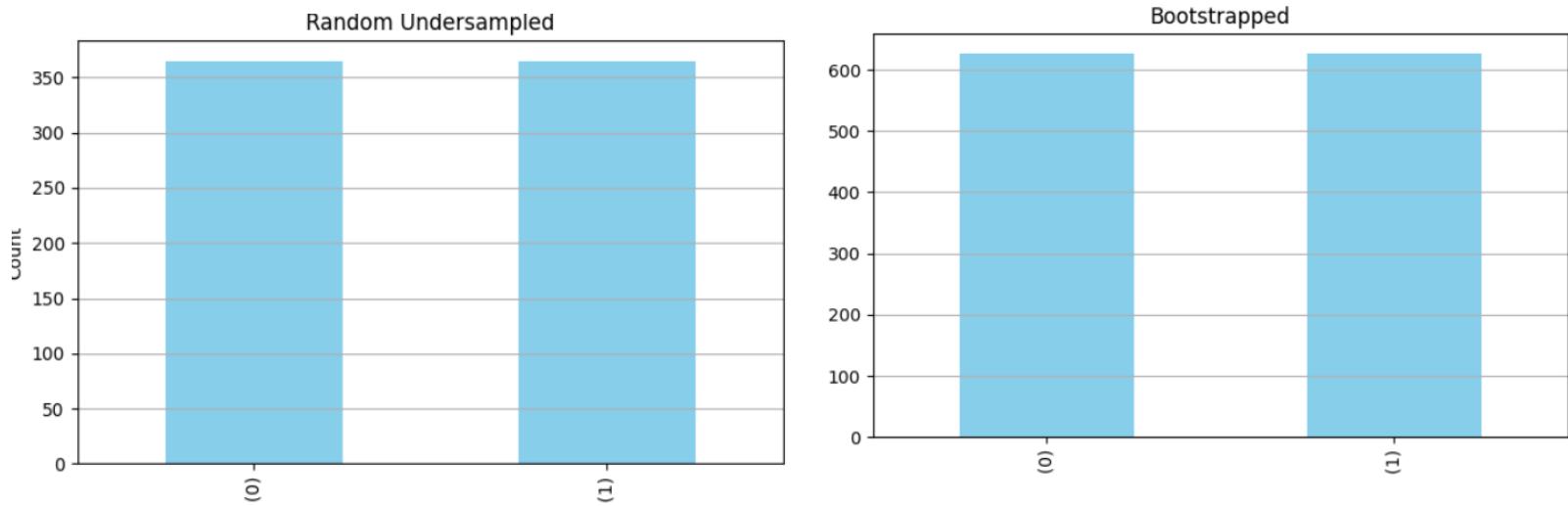
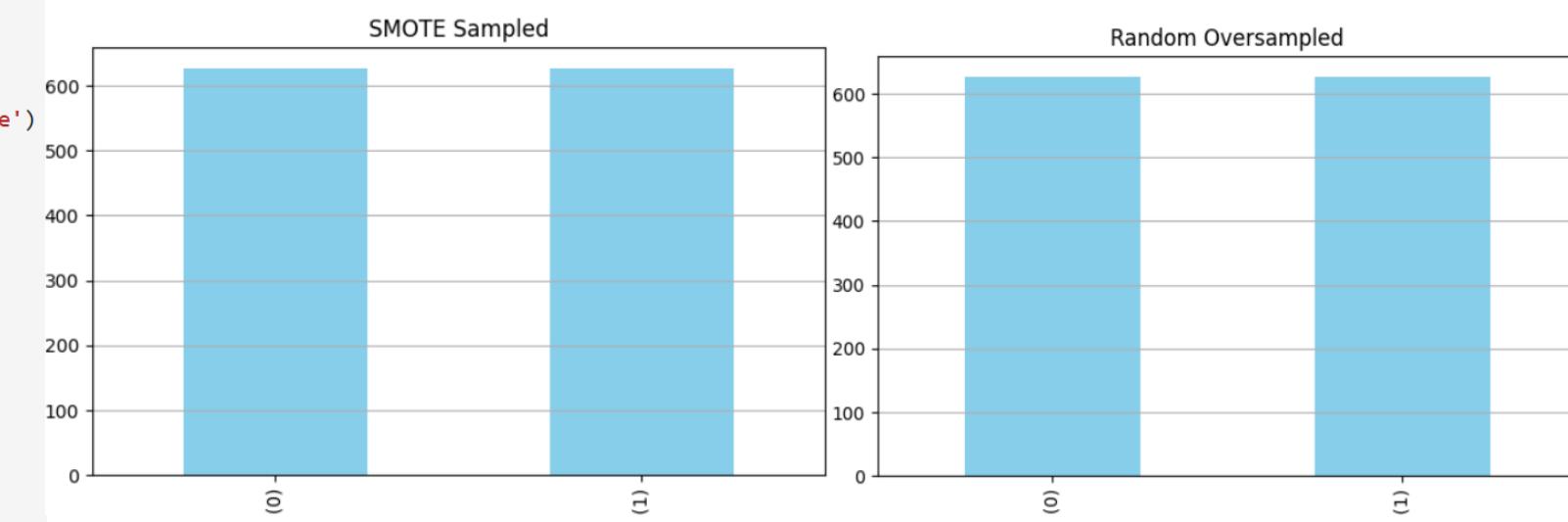
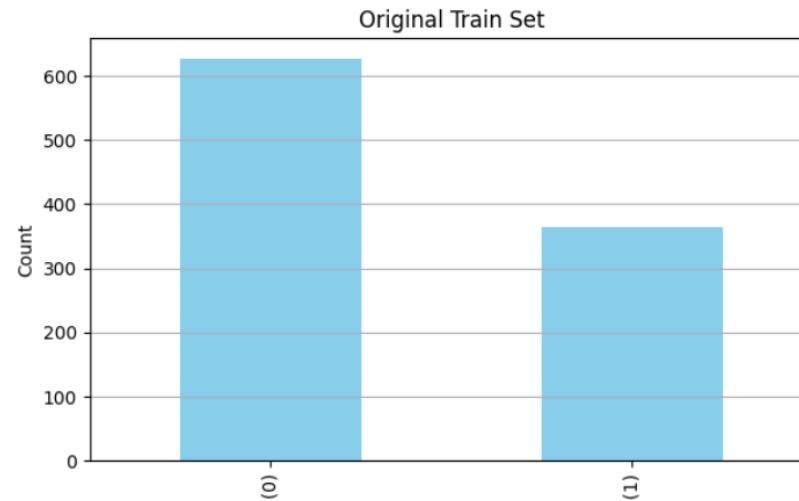
```

import matplotlib.pyplot as plt

def plot_class_distribution(labels, title):
    labels.value_counts().sort_index().plot(kind='bar', color='skyblue')
    plt.title(title)
    plt.xlabel('Class')
    plt.ylabel('Count')
    plt.xticks([0, 1], ['Normal (0)', 'Cancer (1)'])
    plt.grid(axis='y')
    plt.tight_layout()
    plt.show()

# Plots
plot_class_distribution(y_train, "Original Train Set")
plot_class_distribution(y_train_smote, "SMOTE Sampled")
plot_class_distribution(y_train_ros, "Random Oversampled")
plot_class_distribution(y_train_rus, "Random Undersampled")
plot_class_distribution(y_train_bootstrap, "Bootstrapped")

```



```

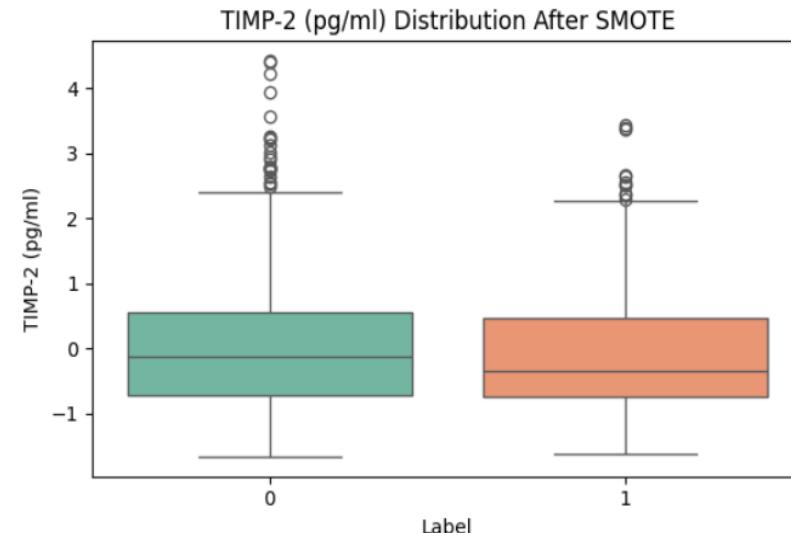
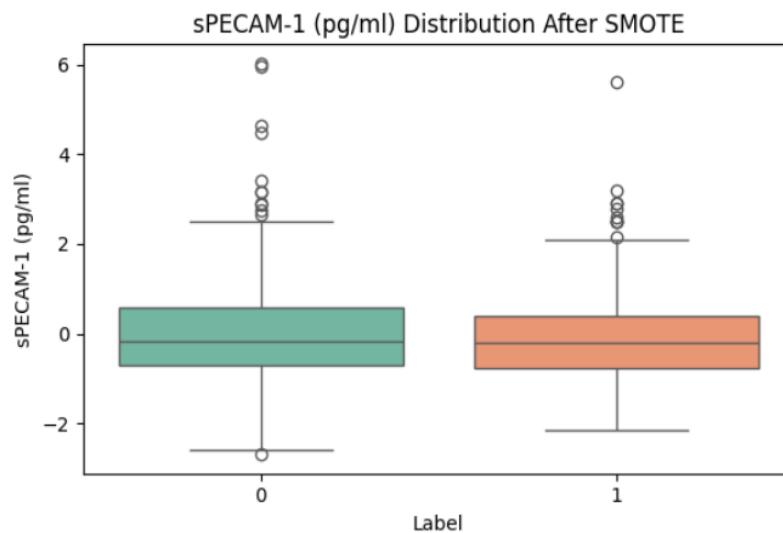
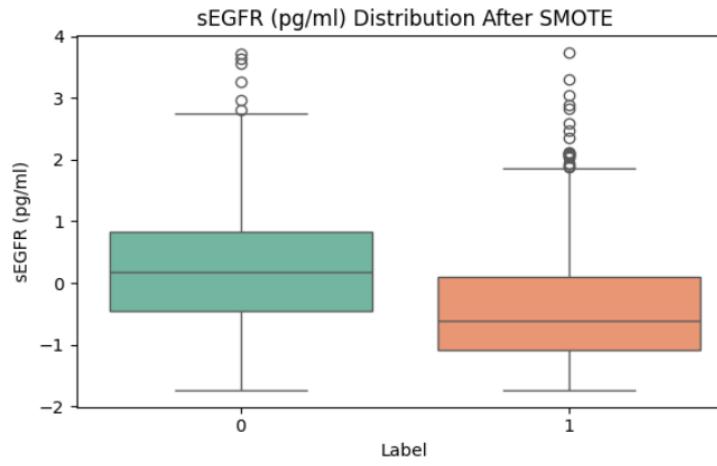
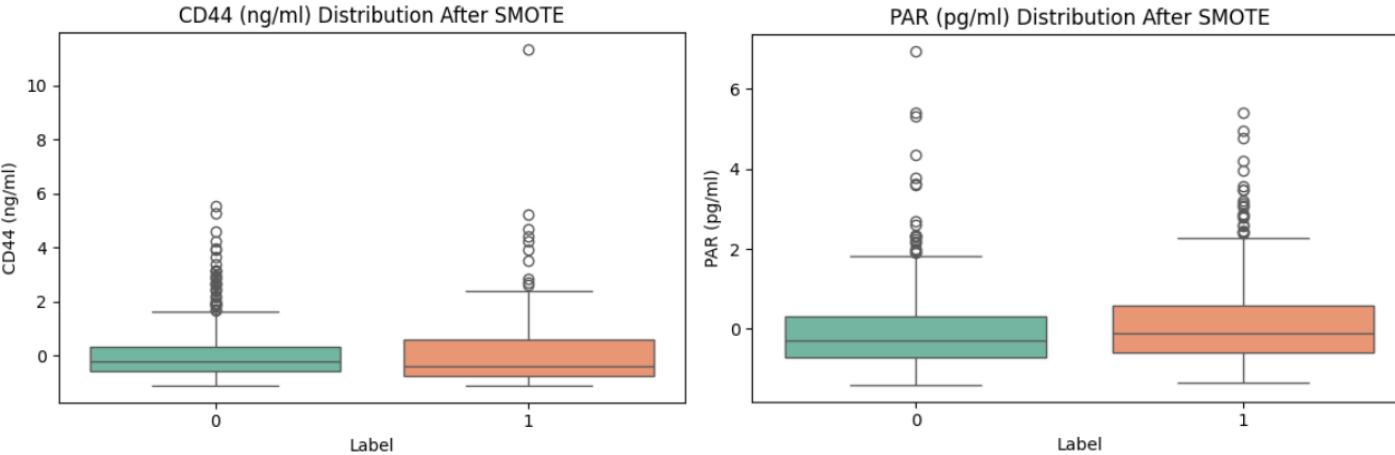
smote_cont_df = pd.DataFrame(X_train_smote, columns=X_train_scaled.columns)
smote_cont_df['Label'] = y_train_smote.reset_index(drop=True)

continuous_features = ["CD44 (ng/ml)", "PAR (pg/ml)", "sEGFR (pg/ml)",
                       "sPECAM-1 (pg/ml)", "TIMP-2 (pg/ml)"]

import seaborn as sns
import matplotlib.pyplot as plt

for feature in continuous_features:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='Label', y=feature, data=smote_cont_df, palette="Set2")
    plt.title(f"{feature} Distribution After SMOTE")
    plt.tight_layout()
    plt.show()

```

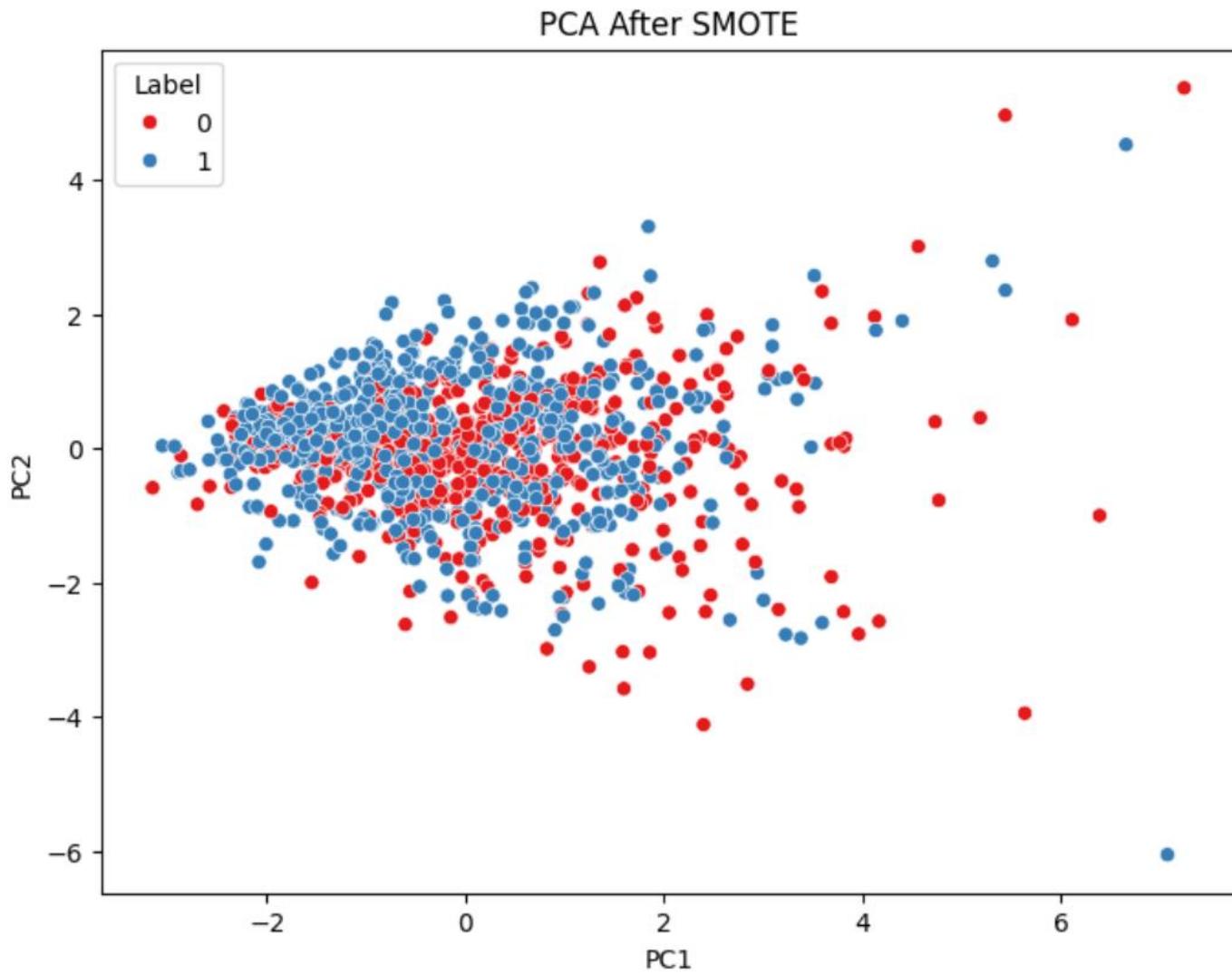


```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_smote)

pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['Label'] = y_train_smote.reset_index(drop=True)

plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue='Label', data=pca_df, palette='Set1')
plt.title("PCA After SMOTE")
plt.show()
```



```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

#SMOTE
rf_model.fit(X_train_smote, y_train_smote)
y_pred_smote = rf_model.predict(X_test_scaled)
print("SMOTE")
print(classification_report(y_test, y_pred_smote))

#Random Oversampling
rf_model.fit(X_train_ros, y_train_ros)
y_pred_ros = rf_model.predict(X_test_scaled)
print("Random Oversampling")
print(classification_report(y_test, y_pred_ros))

#Random Undersampling
rf_model.fit(X_train_rus, y_train_rus)
y_pred_rus = rf_model.predict(X_test_scaled)
print("Random Undersampling")
print(classification_report(y_test, y_pred_rus))

#Bootstrapping
rf_model.fit(X_train_bootstrap, y_train_bootstrap)
y_pred_boot = rf_model.predict(X_test_scaled)
print("Bootstrapping")
print(classification_report(y_test, y_pred_boot))

```

SMOTE		precision	recall	f1-score	support
	0	0.90	0.85	0.87	157
	1	0.76	0.84	0.80	91
		accuracy		0.84	248
		macro avg	0.83	0.84	0.83
		weighted avg	0.85	0.84	0.84
Random Oversampling		precision	recall	f1-score	support
	0	0.88	0.87	0.88	157
	1	0.78	0.80	0.79	91
		accuracy		0.85	248
		macro avg	0.83	0.84	0.84
		weighted avg	0.85	0.85	0.85
Random Undersampling		precision	recall	f1-score	support
	0	0.92	0.86	0.89	157
	1	0.78	0.87	0.82	91
		accuracy		0.86	248
		macro avg	0.85	0.86	0.86
		weighted avg	0.87	0.86	0.86
Bootstrapping		precision	recall	f1-score	support
	0	0.86	0.92	0.89	157
	1	0.85	0.75	0.80	91
		accuracy		0.86	248
		macro avg	0.86	0.84	0.84
		weighted avg	0.86	0.86	0.86

```

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
def plot_conf_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges', xticklabels=['Normal', 'Cancer'], yticklabels=['Normal', 'Cancer'])
    plt.title(f'{title} - Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()
    plt.show()

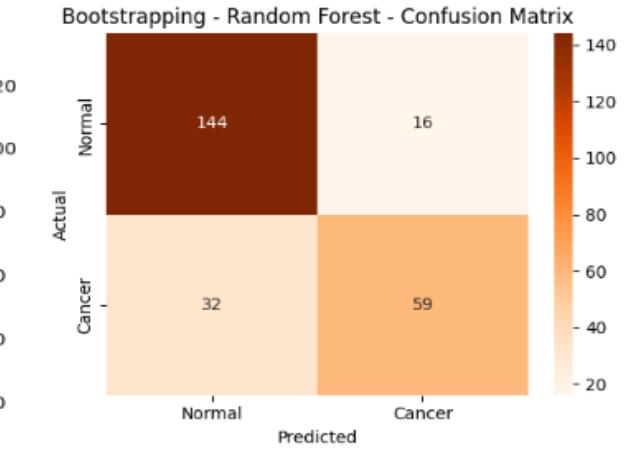
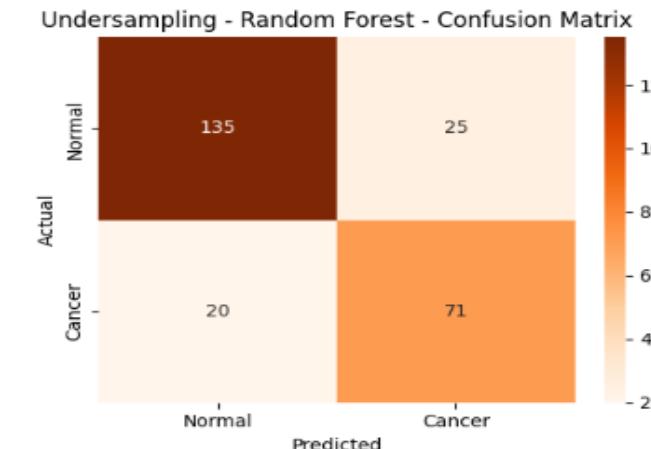
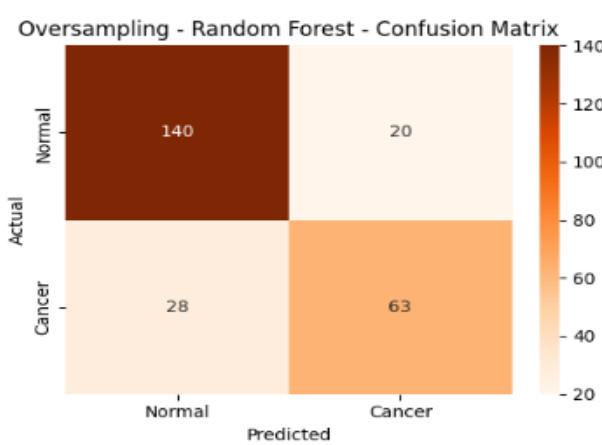
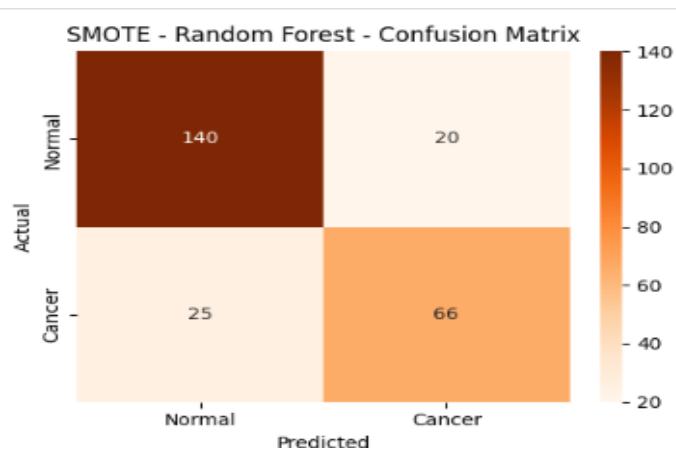
#SMOTE
plot_conf_matrix(y_test, y_pred_smote, "SMOTE - Random Forest")
print("\nSMOTE Confusion Matrix")
print(confusion_matrix(y_test, y_pred_smote))

# Oversampling
plot_conf_matrix(y_test, y_pred_ros, "Oversampling - Random Forest")
print("\nRandom Oversampling Confusion Matrix")
print(confusion_matrix(y_test, y_pred_ros))

#Undersampling
plot_conf_matrix(y_test, y_pred_rus, "Undersampling - Random Forest")
print("\nRandom Undersampling Confusion Matrix")
print(confusion_matrix(y_test, y_pred_rus))

#Bootstrapping
plot_conf_matrix(y_test, y_pred_boot, "Bootstrapping - Random Forest")
print("\nBootstrapping Confusion Matrix")
print(confusion_matrix(y_test, y_pred_boot))

```



```

from sklearn.metrics import roc_auc_score, RocCurveDisplay
import matplotlib.pyplot as plt

# Probabilities for ROC
y_proba_smote = rf_model.predict_proba(X_test_scaled)[:, 1]
rf_model.fit(X_train_smote, y_train_smote)
y_proba_smote = rf_model.predict_proba(X_test_scaled)[:, 1]

rf_model.fit(X_train_ros, y_train_ros)
y_proba_ros = rf_model.predict_proba(X_test_scaled)[:, 1]

rf_model.fit(X_train_rus, y_train_rus)
y_proba_rus = rf_model.predict_proba(X_test_scaled)[:, 1]

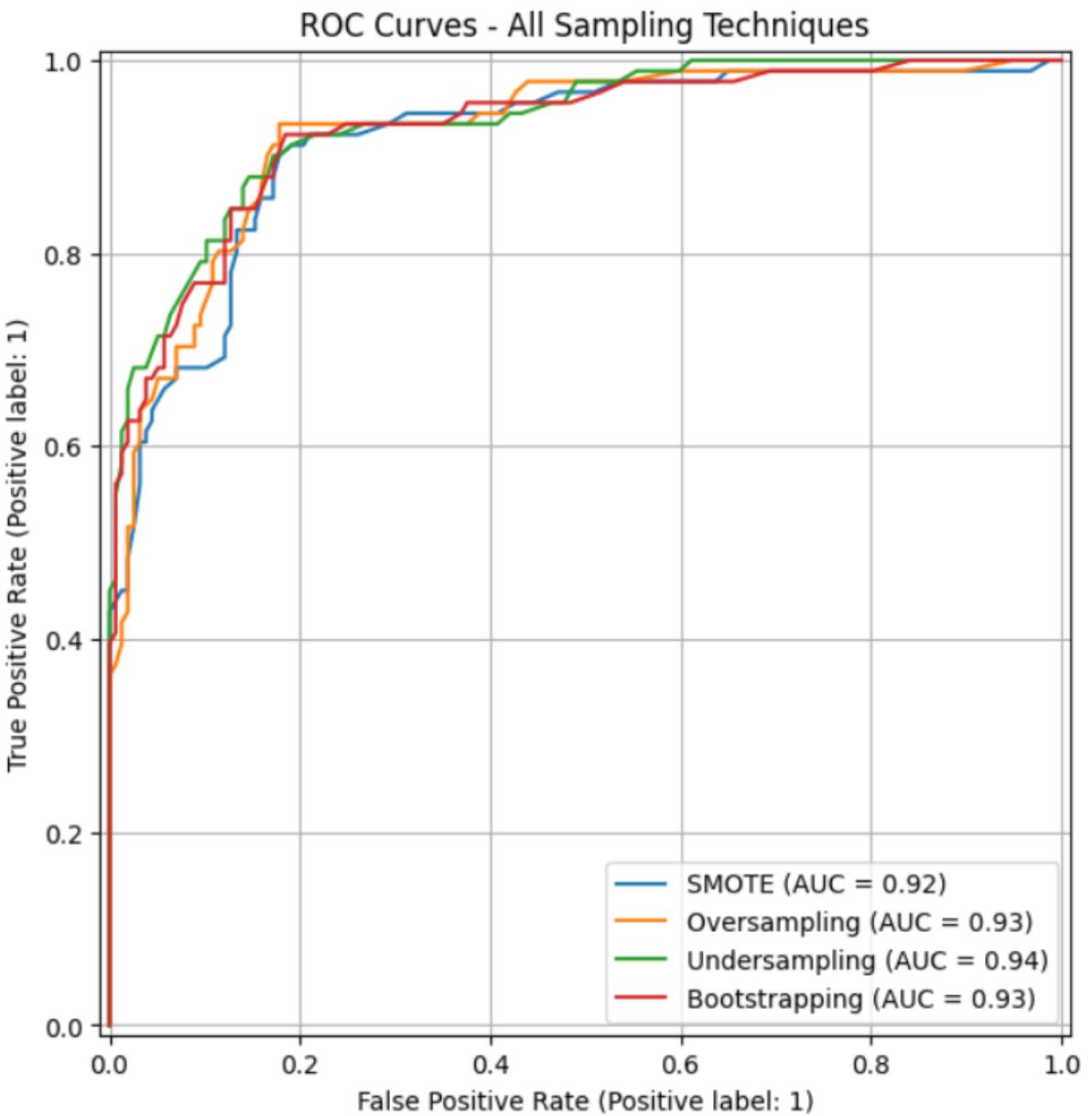
rf_model.fit(X_train_bootstrap, y_train_bootstrap)
y_proba_boot = rf_model.predict_proba(X_test_scaled)[:, 1]

# AUC Scores
print("SMOTE AUC:", roc_auc_score(y_test, y_proba_smote))
print("Random Oversampling AUC:", roc_auc_score(y_test, y_proba_ros))
print("Random Undersampling AUC:", roc_auc_score(y_test, y_proba_rus))
print("Bootstrapping AUC:", roc_auc_score(y_test, y_proba_boot))

# Plot ROC Curves
plt.figure(figsize=(10, 7))
RocCurveDisplay.from_predictions(y_test, y_proba_smote, name="SMOTE", ax=plt.gca())
RocCurveDisplay.from_predictions(y_test, y_proba_ros, name="Oversampling", ax=plt.gca())
RocCurveDisplay.from_predictions(y_test, y_proba_rus, name="Undersampling", ax=plt.gca())
RocCurveDisplay.from_predictions(y_test, y_proba_boot, name="Bootstrapping", ax=plt.gca())
plt.title("ROC Curves - All Sampling Techniques")
plt.grid(True)
plt.show()

```

SMOTE AUC: 0.9172324490795828  
 Random Oversampling AUC: 0.925211730944215  
 Random Undersampling AUC: 0.9357457828795408  
 Bootstrapping AUC: 0.9288864002239798



```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

lr_model = LogisticRegression(max_iter=1000, random_state=42)

#SMOTE
lr_model.fit(X_train_smote, y_train_smote)
y_pred_smote_lr = lr_model.predict(X_test_scaled)
print("SMOTE - Logistic Regression")
print(classification_report(y_test, y_pred_smote_lr))

#Random Oversampling
lr_model.fit(X_train_ros, y_train_ros)
y_pred_ros_lr = lr_model.predict(X_test_scaled)
print("Oversampling - Logistic Regression")
print(classification_report(y_test, y_pred_ros_lr))

#Random Undersampling
lr_model.fit(X_train_rus, y_train_rus)
y_pred_rus_lr = lr_model.predict(X_test_scaled)
print("Undersampling - Logistic Regression")
print(classification_report(y_test, y_pred_rus_lr))

#Bootstrapping
lr_model.fit(X_train_bootstrap, y_train_bootstrap)
y_pred_boot_lr = lr_model.predict(X_test_scaled)
print("Bootstrapping - Logistic Regression")
print(classification_report(y_test, y_pred_boot_lr))

```

SMOTE - Logistic Regression				
	precision	recall	f1-score	support
0	0.87	0.84	0.86	157
1	0.74	0.79	0.77	91
accuracy			0.82	248
macro avg			0.81	0.81
weighted avg			0.83	0.82
Oversampling - Logistic Regression				
	precision	recall	f1-score	support
0	0.88	0.88	0.88	157
1	0.79	0.80	0.80	91
accuracy			0.85	248
macro avg			0.84	0.84
weighted avg			0.85	0.85
Undersampling - Logistic Regression				
	precision	recall	f1-score	support
0	0.89	0.85	0.87	157
1	0.77	0.82	0.79	91
accuracy			0.84	248
macro avg			0.83	0.83
weighted avg			0.85	0.84
Bootstrapping - Logistic Regression				
	precision	recall	f1-score	support
0	0.88	0.88	0.88	157
1	0.79	0.79	0.79	91
accuracy			0.85	248
macro avg			0.84	0.84
weighted avg			0.85	0.85

```

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

def plot_conf_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal', 'Cancer'], yticklabels=['Normal', 'Cancer'])
    plt.title(f"{title} - Confusion Matrix")
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()
    plt.show()

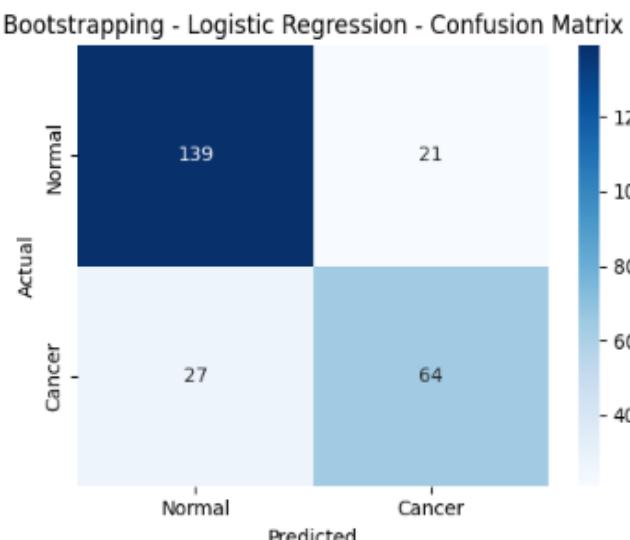
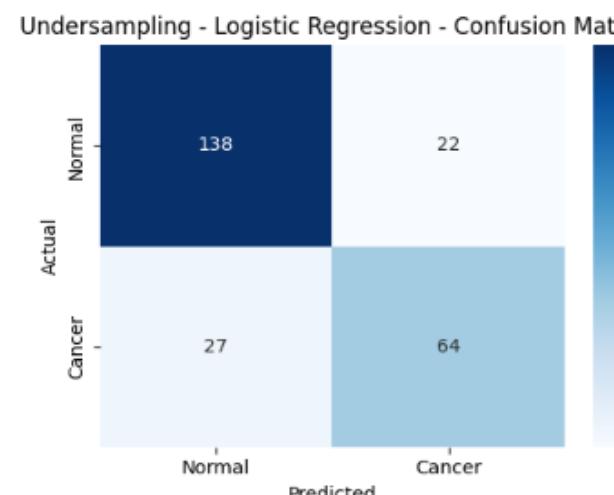
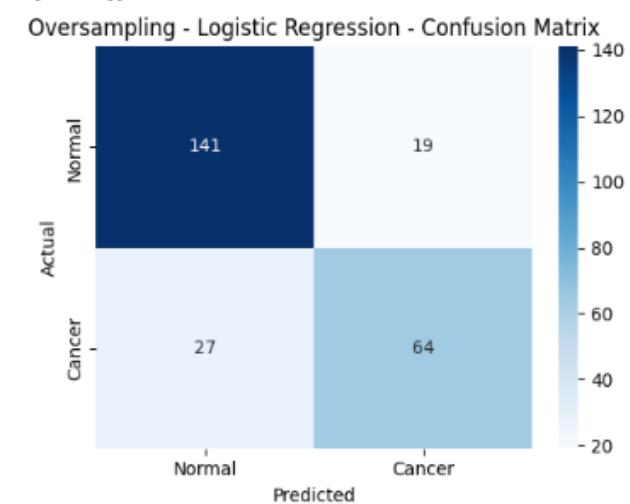
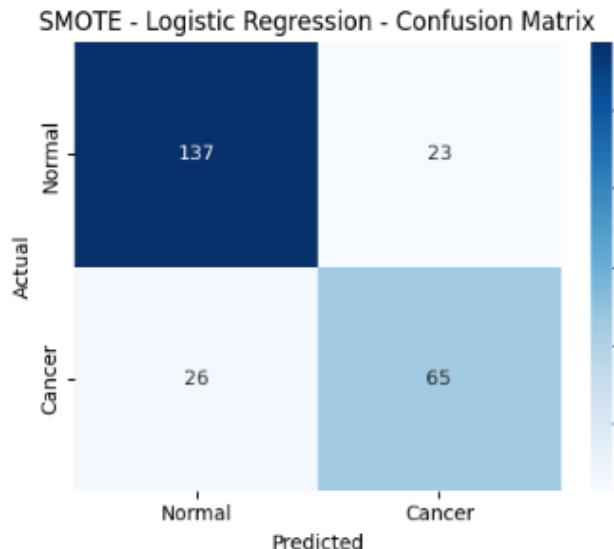
#SMOTE
plot_conf_matrix(y_test, y_pred_smote_lr, "SMOTE - Logistic Regression")
print("SMOTE Confusion Matrix")
print(confusion_matrix(y_test, y_pred_smote_lr))

#Oversampling
plot_conf_matrix(y_test, y_pred_ros_lr, "Oversampling - Logistic Regression")
print("\nOversampling Confusion Matrix")
print(confusion_matrix(y_test, y_pred_ros_lr))

#Undersampling
plot_conf_matrix(y_test, y_pred_rus_lr, "Undersampling - Logistic Regression")
print("\nUndersampling Confusion Matrix")
print(confusion_matrix(y_test, y_pred_rus_lr))

#Bootstrapping
plot_conf_matrix(y_test, y_pred_boot_lr, "Bootstrapping - Logistic Regression")
print("\nBootstrapping Confusion Matrix")
print(confusion_matrix(y_test, y_pred_boot_lr))

```



```

from sklearn.metrics import roc_auc_score, RocCurveDisplay
import matplotlib.pyplot as plt

lr_model.fit(X_train_smote, y_train_smote)
y_proba_smote_lr = lr_model.predict_proba(X_test_scaled)[:, 1]

lr_model.fit(X_train_ros, y_train_ros)
y_proba_ros_lr = lr_model.predict_proba(X_test_scaled)[:, 1]

lr_model.fit(X_train_rus, y_train_rus)
y_proba_rus_lr = lr_model.predict_proba(X_test_scaled)[:, 1]

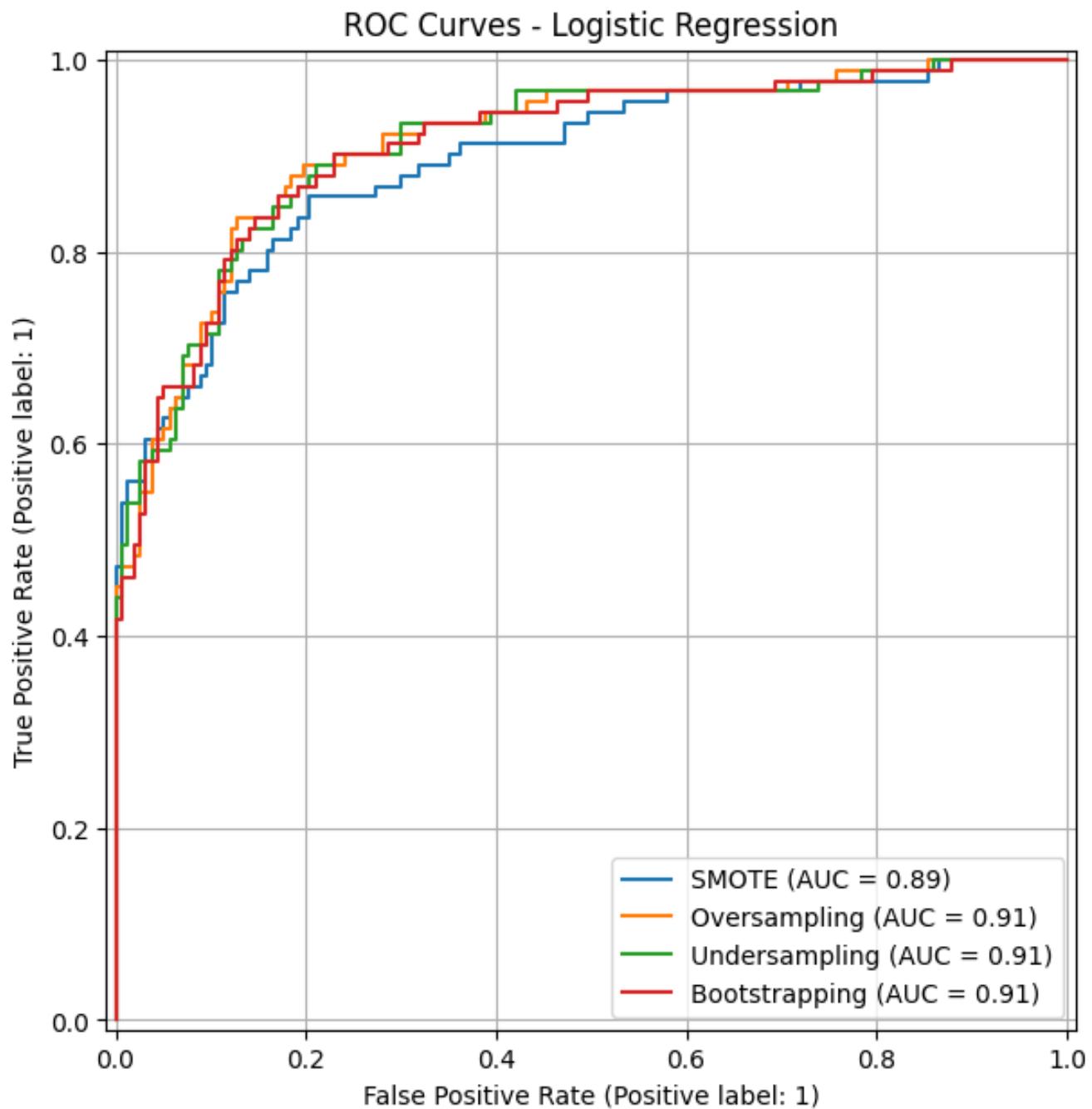
lr_model.fit(X_train_bootstrap, y_train_bootstrap)
y_proba_boot_lr = lr_model.predict_proba(X_test_scaled)[:, 1]

# AUC
print("SMOTE AUC (LogReg):", roc_auc_score(y_test, y_proba_smote_lr))
print("Oversampling AUC (LogReg):", roc_auc_score(y_test, y_proba_ros_lr))
print("Undersampling AUC (LogReg):", roc_auc_score(y_test, y_proba_rus_lr))
print("Bootstrapping AUC (LogReg):", roc_auc_score(y_test, y_proba_boot_lr))

# Plot
plt.figure(figsize=(10, 7))
RocCurveDisplay.from_predictions(y_test, y_proba_smote_lr, name="SMOTE", ax=plt.gca())
RocCurveDisplay.from_predictions(y_test, y_proba_ros_lr, name="Oversampling", ax=plt.gca())
RocCurveDisplay.from_predictions(y_test, y_proba_rus_lr, name="Undersampling", ax=plt.gca())
RocCurveDisplay.from_predictions(y_test, y_proba_boot_lr, name="Bootstrapping", ax=plt.gca())
plt.title("ROC Curves - Logistic Regression")
plt.grid(True)
plt.show()

```

SMOTE AUC (LogReg): 0.8949394554490097  
 Oversampling AUC (LogReg): 0.9125078742913137  
 Undersampling AUC (LogReg): 0.9117379435850773  
 Bootstrapping AUC (LogReg): 0.9097081262686358



```

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import pandas as pd
def summarize_model_performance(y_true, y_pred, model_name):
    return {
        "Model": model_name,
        "Accuracy": accuracy_score(y_true, y_pred),
        "Precision": precision_score(y_true, y_pred),
        "Recall": recall_score(y_true, y_pred),
        "F1-Score": f1_score(y_true, y_pred)
    }

# Random Forest
rf_results = [
    summarize_model_performance(y_test, y_pred_smote, "RF - SMOTE"),
    summarize_model_performance(y_test, y_pred_ros, "RF - ROS"),
    summarize_model_performance(y_test, y_pred_rus, "RF - RUS"),
    summarize_model_performance(y_test, y_pred_boot, "RF - Bootstrapping")
]

# Logistic Regression
lr_results = [
    summarize_model_performance(y_test, y_pred_smote_lr, "LR - SMOTE"),
    summarize_model_performance(y_test, y_pred_ros_lr, "LR - ROS"),
    summarize_model_performance(y_test, y_pred_rus_lr, "LR - RUS"),
    summarize_model_performance(y_test, y_pred_boot_lr, "LR - Bootstrapping")
]
performance_table = pd.DataFrame(rf_results + lr_results)
performance_table = performance_table.round(3)
performance_table.to_csv("model_performance_summary.csv", index=False)

print(performance_table)

```

	Model	Accuracy	Precision	Recall	F1-Score
0	RF - SMOTE	0.843	0.760	0.835	0.796
1	RF - ROS	0.847	0.785	0.802	0.793
2	RF - RUS	0.863	0.782	0.868	0.823
3	RF - Bootstrapping	0.859	0.850	0.747	0.795
4	LR - SMOTE	0.823	0.742	0.791	0.766
5	LR - ROS	0.851	0.793	0.802	0.798
6	LR - RUS	0.843	0.765	0.824	0.794
7	LR - Bootstrapping	0.847	0.791	0.791	0.791

```

import pandas as pd

# Model performance summary
performance_data = [
    ['Random Forest', 'SMOTE', 0.84, 0.76, 0.83, 0.79],
    ['Logistic Regression', 'SMOTE', 0.82, 0.74, 0.79, 0.76],
    ['Random Forest', 'Oversampling', 0.84, 0.78, 0.80, 0.79],
    ['Random Forest', 'Undersampling', 0.84, 0.78, 0.86, 0.82]
]

columns = ['Model', 'Sampling', 'Accuracy', 'Precision', 'Recall', 'F1-Score']
summary_df = pd.DataFrame(performance_data, columns=columns)
styled_table = summary_df.style \
    .highlight_max(subset=['F1-Score'], color='lightgreen') \
    .set_caption("Model Comparison Summary") \
    .format({'Accuracy': '{:.2f}', 'Precision': '{:.2f}', 'Recall': '{:.2f}', 'F1-Score': '{:.2f}'})

```

styled\_table

Model Comparison Summary

	<b>Model</b>	<b>Sampling</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>
<b>0</b>	Random Forest	SMOTE	0.84	0.76	0.83	0.79
<b>1</b>	Logistic Regression	SMOTE	0.82	0.74	0.79	0.76
<b>2</b>	Random Forest	Oversampling	0.84	0.78	0.80	0.79
<b>3</b>	Random Forest	Undersampling	0.84	0.78	0.86	0.82

```

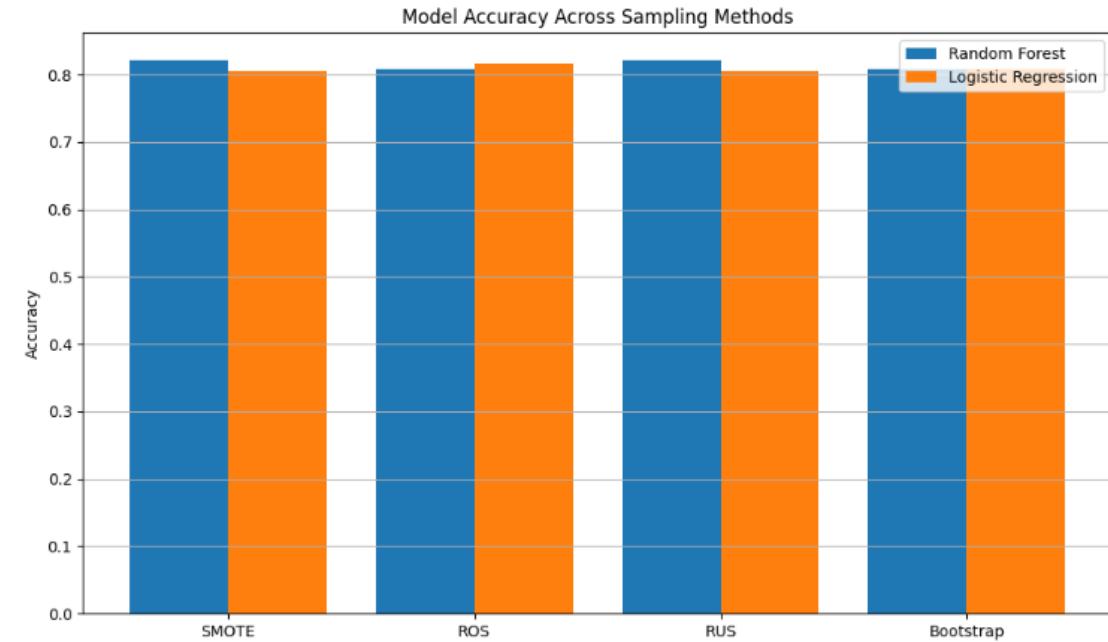
import matplotlib.pyplot as plt
accuracy_rf = {
    'SMOTE': performance_table[performance_table['Model'] == 'RF - SMOTE']['Accuracy'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'RF - ROS']['Accuracy'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'RF - RUS']['Accuracy'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'RF - Bootstrapping']['Accuracy'].values[0]
}

accuracy_lr = {
    'SMOTE': performance_table[performance_table['Model'] == 'LR - SMOTE']['Accuracy'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'LR - ROS']['Accuracy'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'LR - RUS']['Accuracy'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'LR - Bootstrapping']['Accuracy'].values[0]
}

sampling_methods = list(accuracy_rf.keys())
x = range(len(sampling_methods))
rf_scores = list(accuracy_rf.values())
lr_scores = list(accuracy_lr.values())

plt.figure(figsize=(10, 6))
plt.bar([i - 0.2 for i in x], rf_scores, width=0.4, label='Random Forest')
plt.bar([i + 0.2 for i in x], lr_scores, width=0.4, label='Logistic Regression')
plt.xticks(ticks=x, labels=sampling_methods)
plt.ylabel("Accuracy")
plt.title("Model Accuracy Across Sampling Methods")
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()

```



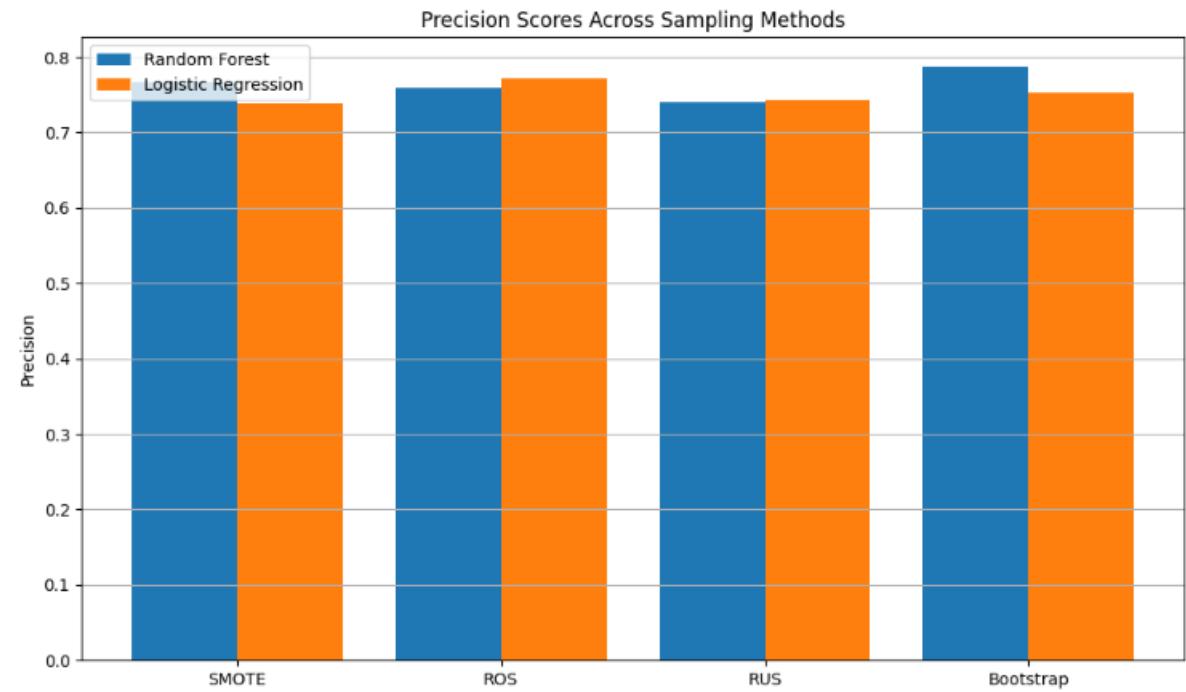
```

precision_rf = {
    'SMOTE': performance_table[performance_table['Model'] == 'RF - SMOTE']['Precision'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'RF - ROS']['Precision'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'RF - RUS']['Precision'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'RF - Bootstrapping']['Precision'].values[0]
}

precision_lr = {
    'SMOTE': performance_table[performance_table['Model'] == 'LR - SMOTE']['Precision'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'LR - ROS']['Precision'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'LR - RUS']['Precision'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'LR - Bootstrapping']['Precision'].values[0]
}

plt.figure(figsize=(10, 6))
plt.bar([i - 0.2 for i in x], list(precision_rf.values()), width=0.4, label='Random Forest')
plt.bar([i + 0.2 for i in x], list(precision_lr.values()), width=0.4, label='Logistic Regression')
plt.xticks(ticks=x, labels=sampling_methods)
plt.ylabel("Precision")
plt.title("Precision Scores Across Sampling Methods")
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()

```



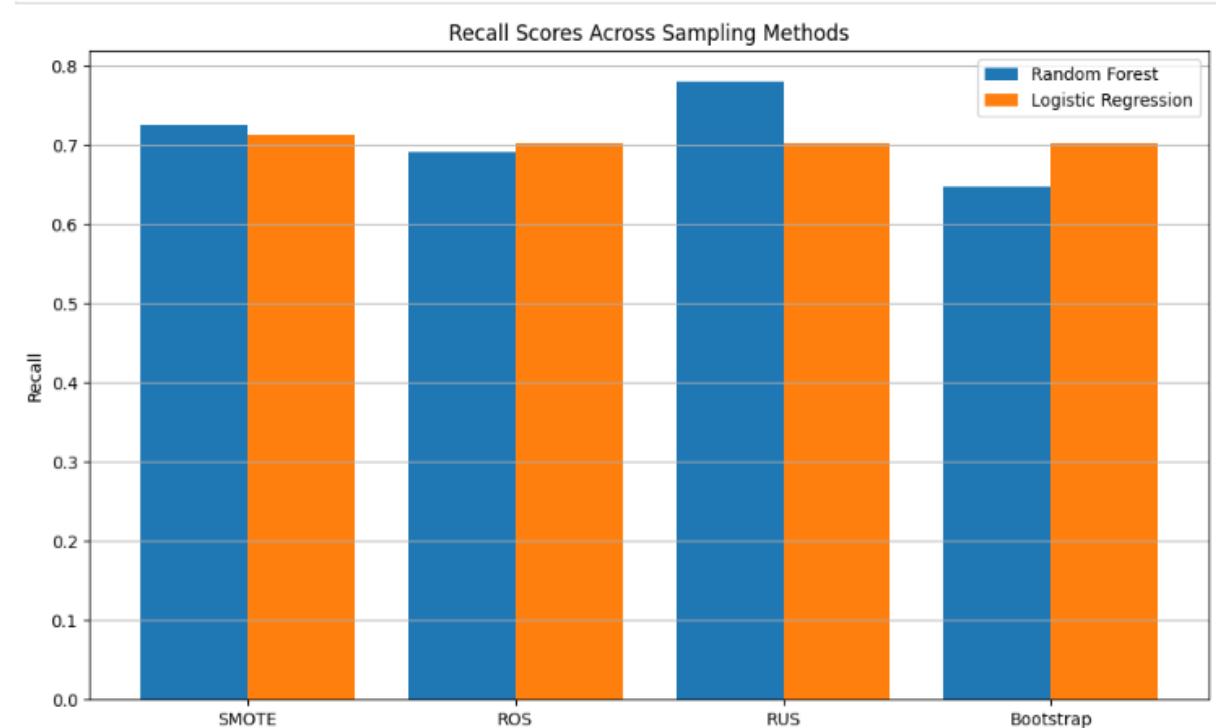
```

# Recall values
recall_rf = {
    'SMOTE': performance_table[performance_table['Model'] == 'RF - SMOTE']['Recall'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'RF - ROS']['Recall'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'RF - RUS']['Recall'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'RF - Bootstrapping']['Recall'].values[0]
}

recall_lr = {
    'SMOTE': performance_table[performance_table['Model'] == 'LR - SMOTE']['Recall'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'LR - ROS']['Recall'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'LR - RUS']['Recall'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'LR - Bootstrapping']['Recall'].values[0]
}

plt.figure(figsize=(10, 6))
plt.bar([i - 0.2 for i in x], list(recall_rf.values()), width=0.4, label='Random Forest')
plt.bar([i + 0.2 for i in x], list(recall_lr.values()), width=0.4, label='Logistic Regression')
plt.xticks(ticks=x, labels=sampling_methods)
plt.ylabel("Recall")
plt.title("Recall Scores Across Sampling Methods")
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()

```



```

import matplotlib.pyplot as plt

f1_rf = {
    'SMOTE': performance_table[performance_table['Model'] == 'RF - SMOTE']['F1-Score'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'RF - ROS']['F1-Score'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'RF - RUS']['F1-Score'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'RF - Bootstrapping']['F1-Score'].values[0]
}

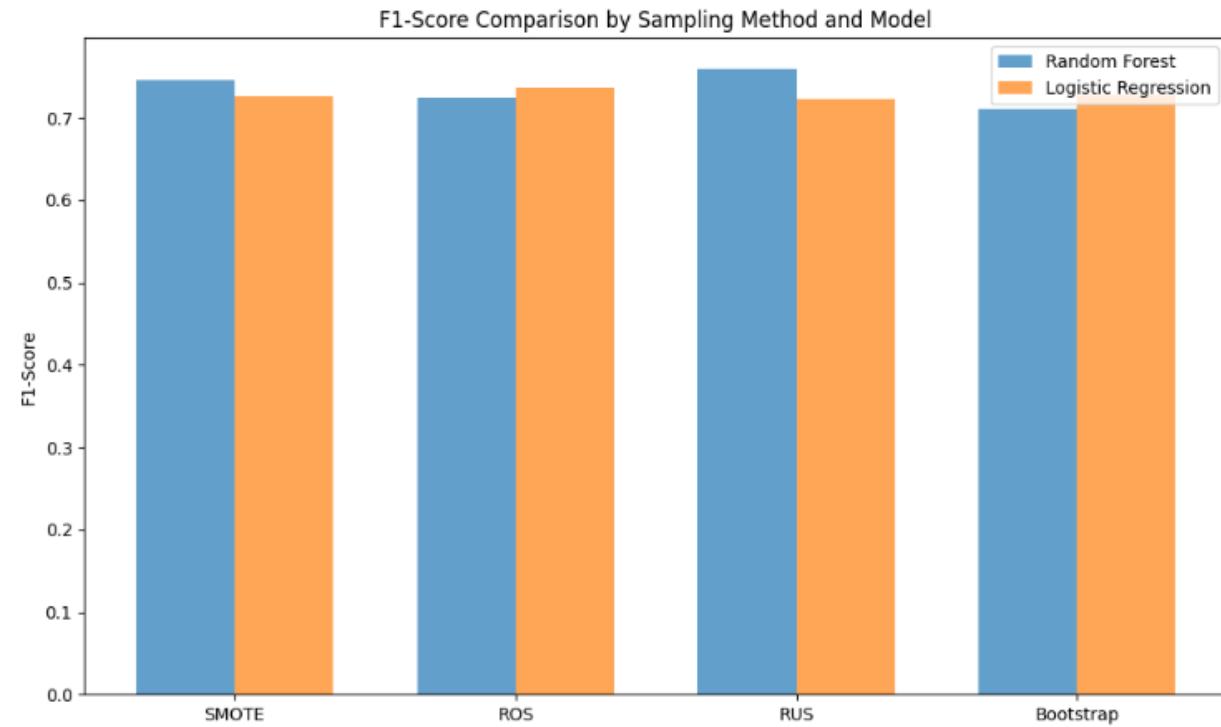
f1_lr = {
    'SMOTE': performance_table[performance_table['Model'] == 'LR - SMOTE']['F1-Score'].values[0],
    'ROS': performance_table[performance_table['Model'] == 'LR - ROS']['F1-Score'].values[0],
    'RUS': performance_table[performance_table['Model'] == 'LR - RUS']['F1-Score'].values[0],
    'Bootstrap': performance_table[performance_table['Model'] == 'LR - Bootstrapping']['F1-Score'].values[0]
}

bar_width = 0.35
methods = list(f1_rf.keys())
x = range(len(methods))

plt.figure(figsize=(10, 6))
plt.bar(x, f1_rf.values(), width=bar_width, label='Random Forest', alpha=0.7)
plt.bar([i + bar_width for i in x], f1_lr.values(), width=bar_width, label='Logistic Regression', alpha=0.7)

plt.xticks([i + bar_width / 2 for i in x], methods)
plt.ylabel('F1-Score')
plt.title('F1-Score Comparison by Sampling Method and Model')
plt.legend()
plt.tight_layout()
plt.show()

```



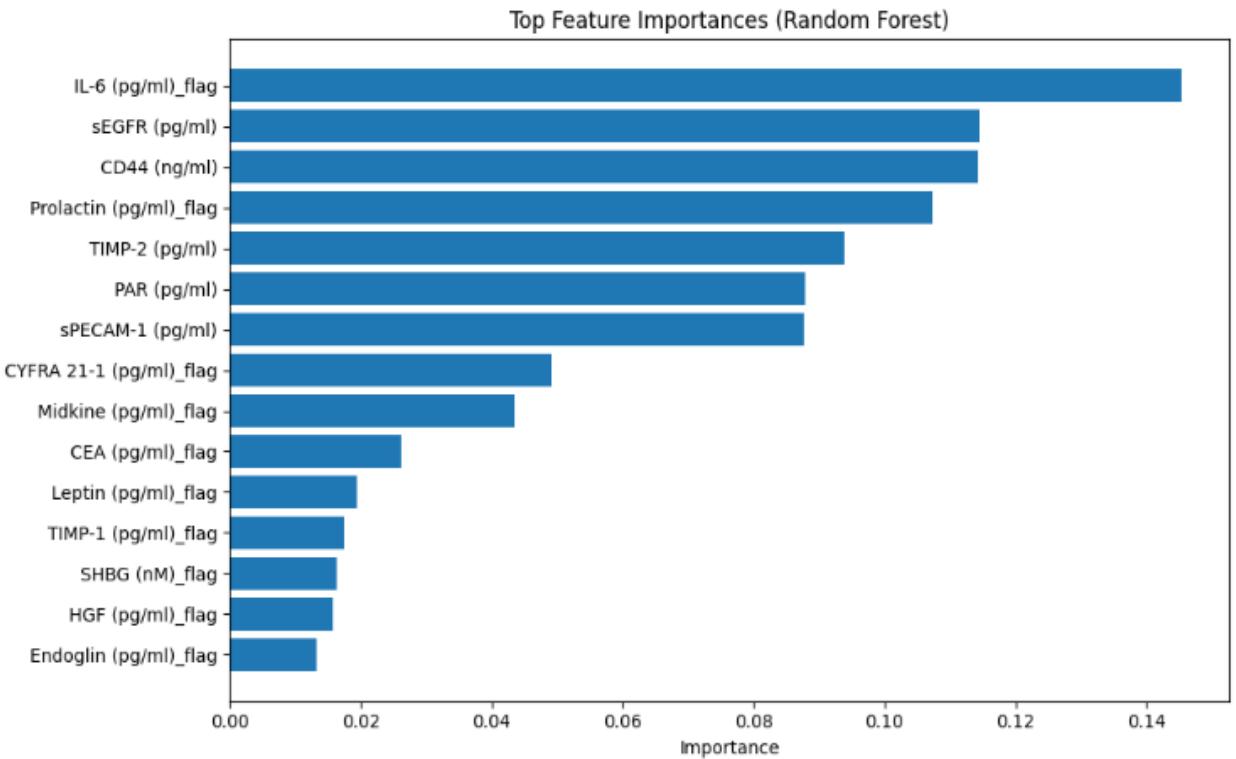
```

importances = rf_model.feature_importances_
features = X_train_scaled.columns

importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

#Top features
top_n = 15
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'][::top_n][::-1], importance_df['Importance'][::top_n][::-1])
plt.xlabel('Importance')
plt.title('Top Feature Importances (Random Forest)')
plt.tight_layout()
plt.show()

```



```
from scipy.stats import chi2_contingency

sex = df['Sex']
label = df['Label']
contingency = pd.crosstab(sex, label)
chi2, p, _, _ = chi2_contingency(contingency)
print(f"Chi2: {chi2:.2f}, p-value: {p:.4f}")
```

Chi2: 1.21, p-value: 0.2706

```
from scipy.stats import mannwhitneyu

group0 = df[df['Label'] == 0]['Age']
group1 = df[df['Label'] == 1]['Age']
stat, p_age = mannwhitneyu(group0, group1)
print(f"Mann-Whitney U test for Age: p-value = {p_age:.4f}")
```

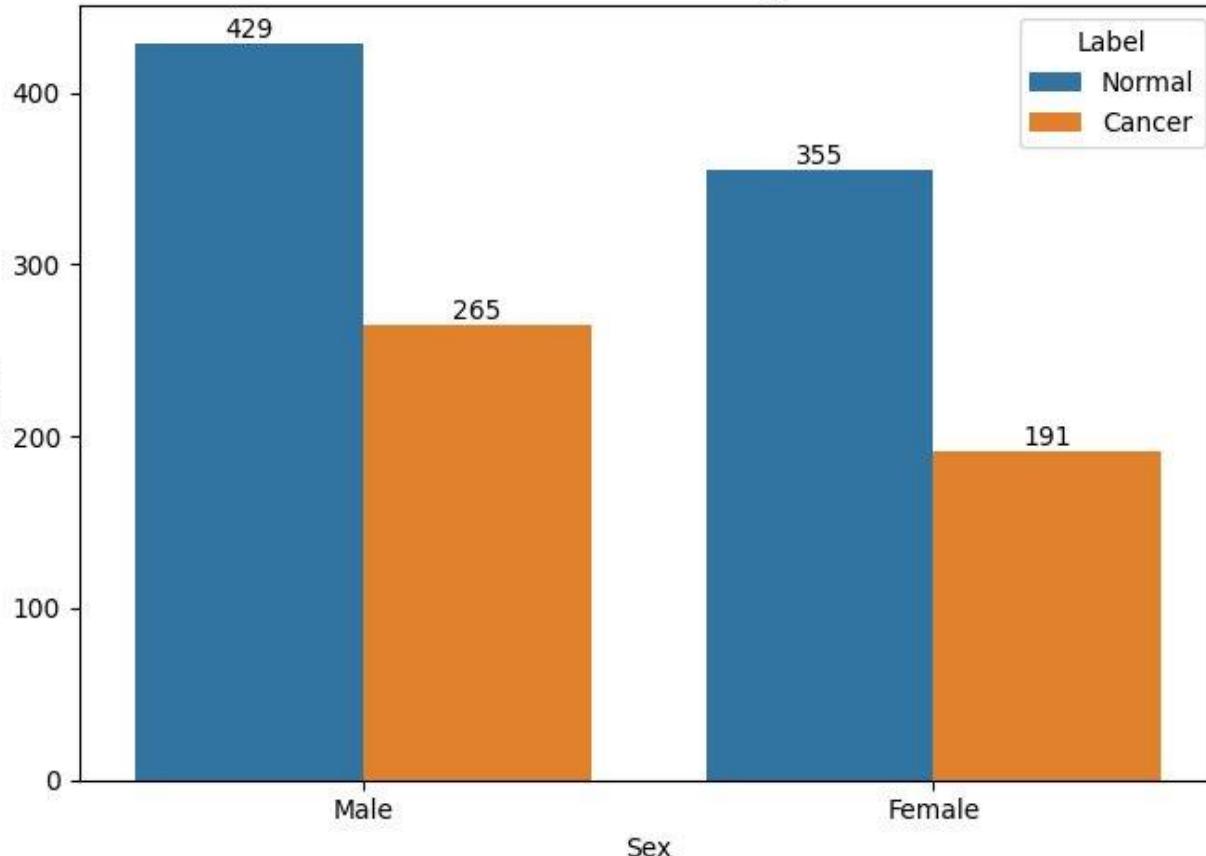
Mann-Whitney U test for Age: p-value = 0.0000

```

plt.figure(figsize=(7, 5))
sns.countplot(data=df, x='Sex', hue='Label')
plt.title("Cancer Case Count by Sex")
plt.legend(title='Label', labels=['Normal', 'Cancer'])
for container in plt.gca().containers:
    plt.bar_label(container, label_type='edge', fontsize=10)
plt.tight_layout()
plt.show()

```

Cancer Case Count by Sex



```

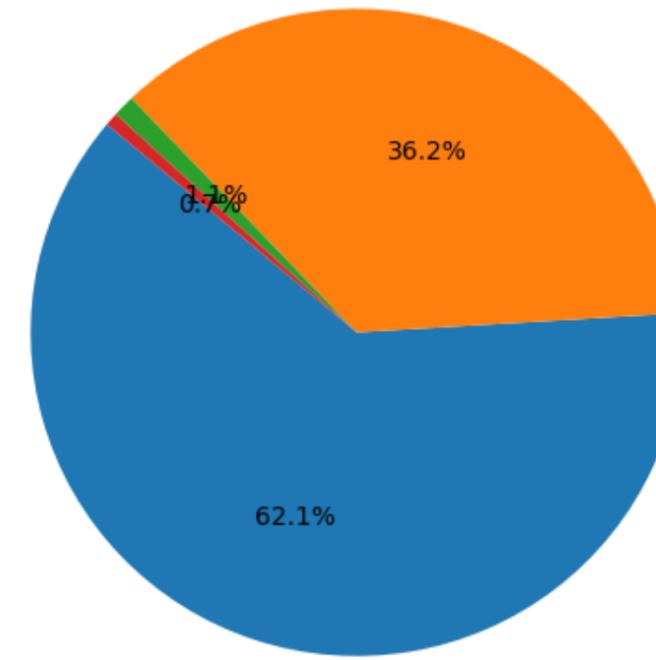
race_counts = df[df['Label'] == 1]['Race'].value_counts()

plt.figure(figsize=(6, 6))
wedges, texts, autotexts = plt.pie(
    race_counts,
    labels=None,
    autopct='%1.1f%%',
    startangle=140
)

plt.title("Race Breakdown (Cancer)")
plt.legend(wedges, race_counts.index, title="Race", loc="center left", bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()

```

Race Breakdown (Cancer)



```

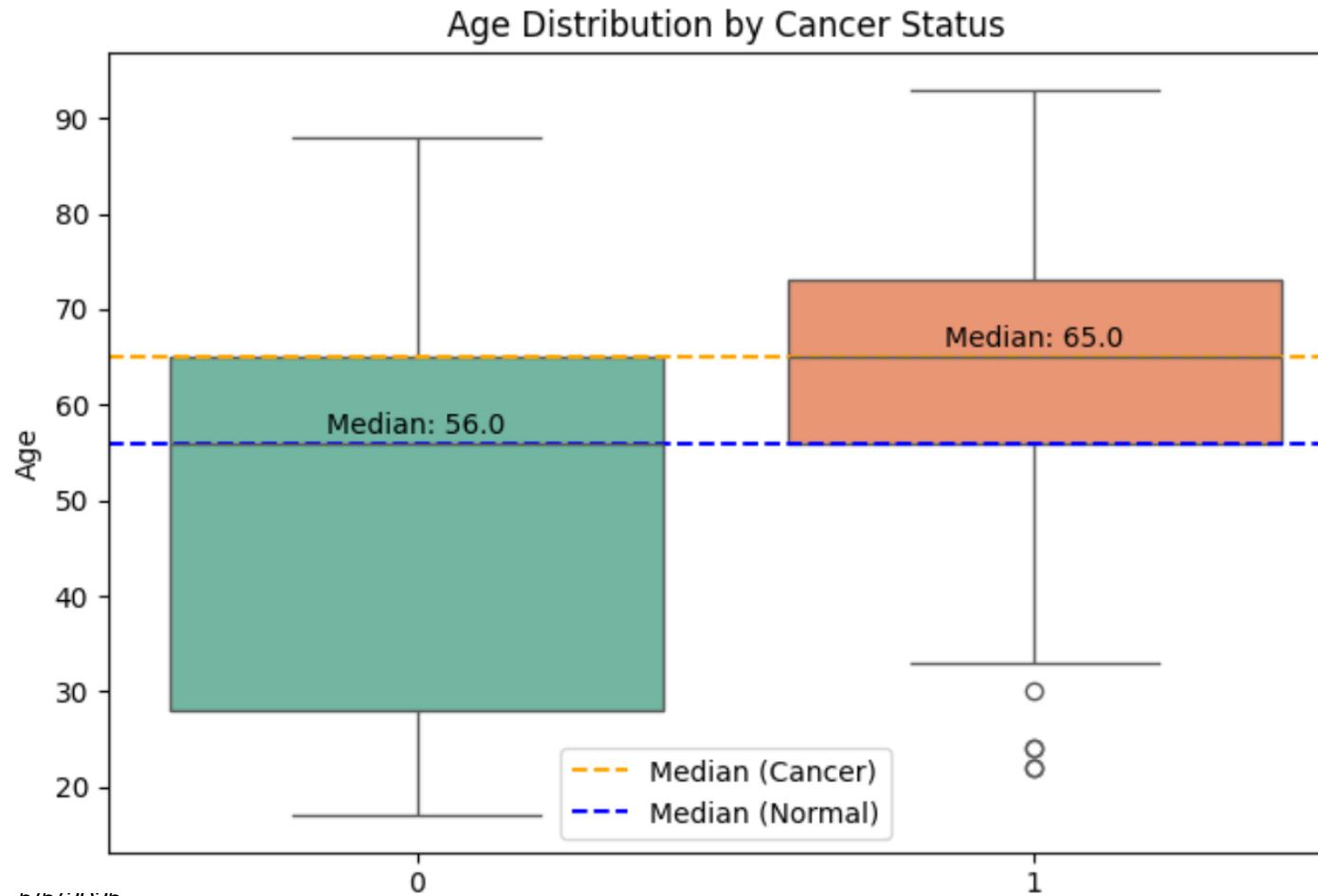
Race
Caucasian
Asian
Black
Unknown

```

```

plt.figure(figsize=(7, 5))
sns.boxplot(x='Label', y='Age', data=df, palette='Set2')
plt.title("Age Distribution by Cancer Status")
medians = df.groupby('Label')['Age'].median()
for i, median in enumerate(medians):
    plt.text(i, median + 1, f"Median: {median:.1f}", ha='center', color='black')
plt.axhline(df[df['Label'] == 1]['Age'].median(), linestyle='--', color='orange', label='Median (Cancer)')
plt.axhline(df[df['Label'] == 0]['Age'].median(), linestyle='--', color='blue', label='Median (Normal)')
plt.legend()
plt.tight_layout()
plt.show()

```





CA-125

Endoglin

# THANK YOU

Early Detection of Gastrointestinal Cancers  
Using Machine Learning and Blood Biomarkers