WiDS Equity in Healthcare Challenge 1

Loading Libraries

In [2]:
```
pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2 k
B)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-package
s (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packa
ges (from catboost) (3.10.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.11/di
st-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-pac
kages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from catboost) (1.14.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages
(from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (fr
om catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
1/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pac
kages (from pandas>=0.24->catboost) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=0.24->catboost) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pac
kages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib->catboost) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-
packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packag
es (from matplotlib->catboost) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib->catboost) (3.2.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-
packages (from plotly->catboost) (9.0.0)
Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 9.0 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, classification_report
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKF
from sklearn.ensemble import VotingClassifier

import warnings
warnings.filterwarnings('ignore')
```

Loading and exploring the dataset

In [4]:
```
#Loading the data
train = pd.read_csv('/content/training.csv')

#Display the first few rows of the dataframe
display('Train Sample:', train.head())
```

'Train Sample:'

	patient_id	patient_race	payer_type	patient_state	patient_zip3	patient_age	patient
0	475714	NaN	MEDICAID	CA	924	84	
1	349367	White	COMMERCIAL	CA	928	62	
2	138632	White	COMMERCIAL	TX	760	43	
3	617843	White	COMMERCIAL	CA	926	45	
4	817482	NaN	COMMERCIAL	ID	836	55	

5 rows × 83 columns

In [5]:
```
#Checking the dimensions of the dataset
print('The number of rows are ',train.shape[0],'\n''The number of columns (varia
```

```
The number of rows are  12906
The number of columns (variables) are 83
```

In [6]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12906 entries, 0 to 12905
Data columns (total 83 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   patient_id                              12906 non-null  int64
 1   patient_race                            6521 non-null   object
 2   payer_type                              11103 non-null  object
 3   patient_state                           12855 non-null  object
 4   patient_zip3                            12906 non-null  int64
 5   patient_age                             12906 non-null  int64
 6   patient_gender                          12906 non-null  object
 7   bmi                                     3941 non-null   float64
 8   breast_cancer_diagnosis_code            12906 non-null  object
 9   breast_cancer_diagnosis_desc            12906 non-null  object
 10  metastatic_cancer_diagnosis_code        12906 non-null  object
 11  metastatic_first_novel_treatment        24 non-null     object
 12  metastatic_first_novel_treatment_type   24 non-null     object
 13  Region                                  12854 non-null  object
 14  Division                                12854 non-null  object
 15  population                              12905 non-null  float64
 16  density                                 12905 non-null  float64
 17  age_median                              12905 non-null  float64
 18  age_under_10                            12905 non-null  float64
 19  age_10_to_19                            12905 non-null  float64
 20  age_20s                                 12905 non-null  float64
 21  age_30s                                 12905 non-null  float64
 22  age_40s                                 12905 non-null  float64
 23  age_50s                                 12905 non-null  float64
 24  age_60s                                 12905 non-null  float64
 25  age_70s                                 12905 non-null  float64
 26  age_over_80                             12905 non-null  float64
 27  male                                    12905 non-null  float64
 28  female                                  12905 non-null  float64
 29  married                                 12905 non-null  float64
 30  divorced                                12905 non-null  float64
 31  never_married                           12905 non-null  float64
 32  widowed                                 12905 non-null  float64
 33  family_size                             12902 non-null  float64
 34  family_dual_income                      12902 non-null  float64
 35  income_household_median                 12902 non-null  float64
 36  income_household_under_5                12902 non-null  float64
 37  income_household_5_to_10                12902 non-null  float64
 38  income_household_10_to_15               12902 non-null  float64
 39  income_household_15_to_20               12902 non-null  float64
 40  income_household_20_to_25               12902 non-null  float64
 41  income_household_25_to_35               12902 non-null  float64
 42  income_household_35_to_50               12902 non-null  float64
 43  income_household_50_to_75               12902 non-null  float64
 44  income_household_75_to_100              12902 non-null  float64
 45  income_household_100_to_150             12902 non-null  float64
 46  income_household_150_over               12902 non-null  float64
 47  income_household_six_figure             12902 non-null  float64
 48  income_individual_median                12905 non-null  float64
 49  home_ownership                          12902 non-null  float64
 50  housing_units                           12905 non-null  float64
 51  home_value                              12902 non-null  float64
 52  rent_median                             12902 non-null  float64
 53  rent_burden                             12902 non-null  float64
 54  education_less_highschool               12905 non-null  float64
```

```
 55  education_highschool                12905 non-null  float64
 56  education_some_college              12905 non-null  float64
 57  education_bachelors                 12905 non-null  float64
 58  education_graduate                  12905 non-null  float64
 59  education_college_or_above          12905 non-null  float64
 60  education_stem_degree               12905 non-null  float64
 61  labor_force_participation           12905 non-null  float64
 62  unemployment_rate                   12905 non-null  float64
 63  self_employed                       12902 non-null  float64
 64  farmer                              12902 non-null  float64
 65  race_white                          12905 non-null  float64
 66  race_black                          12905 non-null  float64
 67  race_asian                          12905 non-null  float64
 68  race_native                         12905 non-null  float64
 69  race_pacific                        12905 non-null  float64
 70  race_other                          12905 non-null  float64
 71  race_multiple                       12905 non-null  float64
 72  hispanic                            12905 non-null  float64
 73  disabled                            12905 non-null  float64
 74  poverty                             12902 non-null  float64
 75  limited_english                     12902 non-null  float64
 76  commute_time                        12905 non-null  float64
 77  health_uninsured                    12905 non-null  float64
 78  veteran                             12905 non-null  float64
 79  Ozone                               12877 non-null  float64
 80  PM25                                12877 non-null  float64
 81  N02                                 12877 non-null  float64
 82  DiagPeriodL90D                      12906 non-null  int64
dtypes: float64(68), int64(4), object(11)
memory usage: 8.2+ MB
```

Dataset Overview

Here are key insights derived from the initial exploration of the dataset:

Size and Structure

- **Entries**: 12,906 patients.
- **Attributes**: 83 attributes per patient, encompassing a mix of numerical and categorical data.

Missing Values

- **Significant missing data** in columns such as `patient_race`, `payer_type`, `bmi`, `metastatic_first_novel_treatment`, and `metastatic_first_novel_treatment_type`.
- Environmental indicators like `Ozone`, `PM25`, and `N02` also show missing entries, which need to be addressed for comprehensive analyses.

Data Types

- **Numeric Columns**: 68 (including both integers and floating-point numbers).

- **Categorical Columns**: 11, which includes textual or nominal data, requiring encoding for many analytical methods.

Key Features

- **Demographics**: Includes detailed demographic data like age, gender, and race.
- **Geographic Data**: State, region, and zip code information for location-based analysis.
- **Health and Treatment Data**: Detailed codes and descriptions for breast cancer diagnoses, metastatic conditions, and treatment specifics.
- **Socioeconomic Indicators**: Includes a variety of indicators such as income, education, employment, housing, and insurance status.

Environmental Indicators

- Includes measurements like ozone (O3), fine particulate matter (PM2.5), and nitrogen dioxide (NO2) levels, which may correlate with health outcomes.

Target Variable

- **DiagPeriodL90D**: Indicates whether the diagnosis period was less than 90 days, serving as the target variable for predictive modeling tasks.

In [7]:
```
train.describe().T.style.background_gradient().format("{:.2f}")
```

Out[7]:

	count	mean	std	min	25%	
patient_id	12906.00	547381.20	260404.96	100063.00	321517.00	543
patient_zip3	12906.00	573.75	275.45	101.00	331.00	
patient_age	12906.00	59.18	13.34	18.00	50.00	
bmi	3941.00	28.98	5.70	14.00	24.66	
population	12905.00	20744.44	13886.90	635.55	9463.90	19
density	12905.00	1581.95	2966.31	0.92	171.86	
age_median	12905.00	40.50	4.04	20.60	37.13	
age_under_10	12905.00	11.12	1.51	0.00	10.16	
age_10_to_19	12905.00	12.95	1.92	6.31	11.74	
age_20s	12905.00	13.29	3.35	5.92	11.01	
age_30s	12905.00	12.86	2.32	1.50	11.29	
age_40s	12905.00	12.07	1.25	0.80	11.34	
age_50s	12905.00	13.44	1.64	0.00	12.30	
age_60s	12905.00	12.62	2.57	0.20	10.62	
age_70s	12905.00	7.65	2.15	0.00	6.01	
age_over_80	12905.00	4.00	1.24	0.00	3.28	
male	12905.00	50.10	1.66	39.73	49.16	
female	12905.00	49.90	1.66	38.40	49.06	
married	12905.00	47.68	7.49	0.90	42.93	
divorced	12905.00	12.67	2.05	0.20	11.16	
never_married	12905.00	33.83	8.02	13.44	27.53	
widowed	12905.00	5.81	1.54	0.00	4.76	
family_size	12902.00	3.20	0.22	2.55	3.04	
family_dual_income	12902.00	51.85	6.81	19.31	47.64	
income_household_median	12902.00	74374.37	20712.59	29222.00	61284.45	69
income_household_under_5	12902.00	3.27	1.45	0.75	2.26	
income_household_5_to_10	12902.00	2.51	1.34	0.36	1.54	
income_household_10_to_15	12902.00	4.14	1.77	1.02	2.90	
income_household_15_to_20	12902.00	3.93	1.45	1.03	2.92	
income_household_20_to_25	12902.00	4.07	1.35	1.10	3.17	
income_household_25_to_35	12902.00	8.40	2.21	2.65	6.78	
income_household_35_to_50	12902.00	11.56	2.58	1.70	9.90	
income_household_50_to_75	12902.00	16.89	2.72	4.95	15.34	

	count	mean	std	min	25%	
income_household_75_to_100	12902.00	12.66	1.83	4.73	11.35	
income_household_100_to_150	12902.00	15.83	3.18	4.29	13.62	
income_household_150_over	12902.00	16.72	8.94	0.84	10.07	
income_household_six_figure	12902.00	32.55	11.13	5.69	24.57	
income_individual_median	12905.00	36520.52	8195.16	4316.00	31575.26	3
home_ownership	12902.00	65.99	14.12	15.85	56.53	
housing_units	12905.00	7575.71	4916.91	0.00	3446.64	
home_value	12902.00	339817.16	251697.60	60629.00	167760.45	24
rent_median	12902.00	1237.30	427.84	448.40	895.79	
rent_burden	12902.00	31.34	4.78	17.42	28.20	
education_less_highschool	12905.00	11.99	5.14	0.00	8.25	
education_highschool	12905.00	27.56	7.99	0.00	21.64	
education_some_college	12905.00	28.92	4.89	7.20	26.03	
education_bachelors	12905.00	19.27	6.17	2.47	14.02	
education_graduate	12905.00	12.26	5.94	2.09	7.65	
education_college_or_above	12905.00	31.53	11.63	7.05	22.09	
education_stem_degree	12905.00	43.39	4.64	23.91	40.35	
labor_force_participation	12905.00	61.62	5.95	30.70	57.94	
unemployment_rate	12905.00	5.95	1.95	0.82	4.72	
self_employed	12902.00	13.21	3.41	2.26	10.87	
farmer	12902.00	1.86	3.06	0.00	0.04	
race_white	12905.00	69.72	17.96	14.50	56.76	
race_black	12905.00	11.45	12.53	0.06	2.24	
race_asian	12905.00	5.47	6.74	0.00	1.10	
race_native	12905.00	0.90	2.50	0.00	0.21	
race_pacific	12905.00	0.14	0.51	0.00	0.02	
race_other	12905.00	5.68	6.25	0.00	1.31	
race_multiple	12905.00	6.64	3.54	0.43	3.92	
hispanic	12905.00	18.46	17.03	0.19	4.71	
disabled	12905.00	13.34	3.69	4.60	10.27	
poverty	12902.00	13.41	5.22	3.43	9.66	
limited_english	12902.00	4.47	4.84	0.00	0.99	
commute_time	12905.00	27.98	5.08	12.46	24.93	

	count	mean	std	min	25%
health_uninsured	12905.00	8.58	4.20	2.44	5.62
veteran	12905.00	7.08	3.11	1.20	4.93
Ozone	12877.00	39.82	3.56	30.94	37.70
PM25	12877.00	7.48	1.52	2.64	6.65
N02	12877.00	16.10	5.84	2.76	11.28
DiagPeriodL90D	12906.00	0.62	0.48	0.00	0.00

Key Insights from Dataset Summary

- **Patient Data Coverage**: The dataset contains 12,906 entries, with patients' ages ranging from 18 to 91 years, indicating a broad age distribution. The gender distribution is nearly balanced with approximately 50% male and 50% female.
- **Geographic Diversity**: Zip codes range from 101 to 999, suggesting a wide geographic spread across different areas.
- **Environmental Indicators**: Average values for environmental factors like Ozone (39.82), PM2.5 (7.48), and NO2 (16.10) vary, which may impact health outcomes analyzed.
- **Economic Indicators**: Median household income is about $74,374 with a high homeownership rate at around 66%, pointing to a relatively affluent demographic.
- **Educational Attainment**: Around 31.53% of the population holds a college degree or higher, suggesting a well-educated cohort.
- **Health Metrics**: Only a subset of patients (3,941) have BMI data recorded, with an average BMI of 28.98, indicating a tendency towards overweight conditions.
- **Data Completeness**: Significant data gaps exist, notably in `bmi` and `metastatic_first_novel_treatment` fields, which are crucial for detailed health analyses.

In [8]:
```
#Categorical Columns
train.select_dtypes(include=['object']).describe().T
```

Out[8]:

	count	unique	top	freq
patient_race	6521	5	White	3588
payer_type	11103	3	COMMERCIAL	6032
patient_state	12855	50	CA	2438
patient_gender	12906	1	F	12906
breast_cancer_diagnosis_code	12906	50	1749	1982
breast_cancer_diagnosis_desc	12906	50	Malignant neoplasm of breast (female), unspeci...	1982
metastatic_cancer_diagnosis_code	12906	43	C773	7052
metastatic_first_novel_treatment	24	2	PEMBROLIZUMAB	13
metastatic_first_novel_treatment_type	24	1	Antineoplastics	24
Region	12854	4	South	3919
Division	12854	9	East North Central	2923

Summary of Key Categorical Data

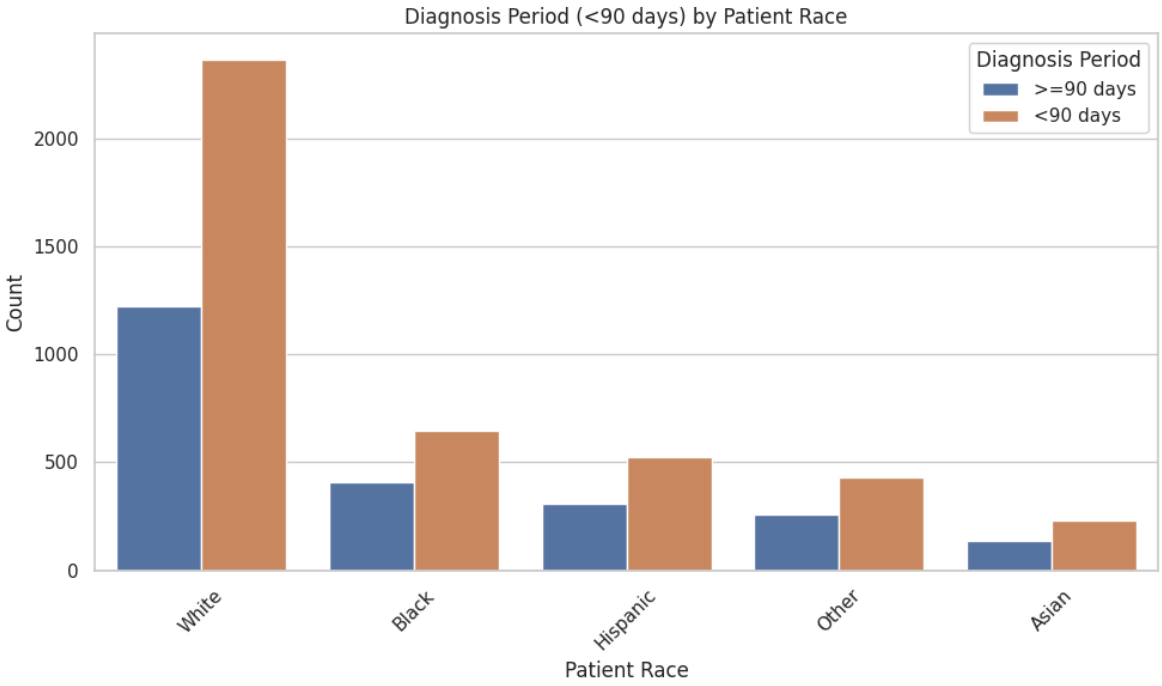
The following highlights address important trends and notable observations within key categorical columns of the dataset:

- **Patient Race**:
  - Available for 6,521 patients, with 'White' as the predominant race, accounting for 3,558 occurrences. This suggests a significant representation of this demographic.
- **Payer Type**:
  - Among 11,103 entries with data, 'COMMERCIAL' is the most common payer type, indicating that a large portion of patients have commercial health insurance.
- **Patient State**:
  - Data spans all 50 states with California (CA) having the highest concentration of cases (2,438), pointing to higher data collection or incidence rates in this state.
- **Breast Cancer Diagnosis**:
  - The dataset details 50 unique breast cancer diagnosis codes. The code '1749' and its corresponding description 'Malignant neoplasm of breast (female), unspecified' appear most frequently (1,982 times), highlighting commonality in diagnosis entries.
- **Metastatic Cancer Diagnosis**:

- 'C773' is notably frequent among the 43 unique metastatic cancer diagnosis codes, appearing 7,052 times, indicating a prevalent focus within the dataset.

Visualizations

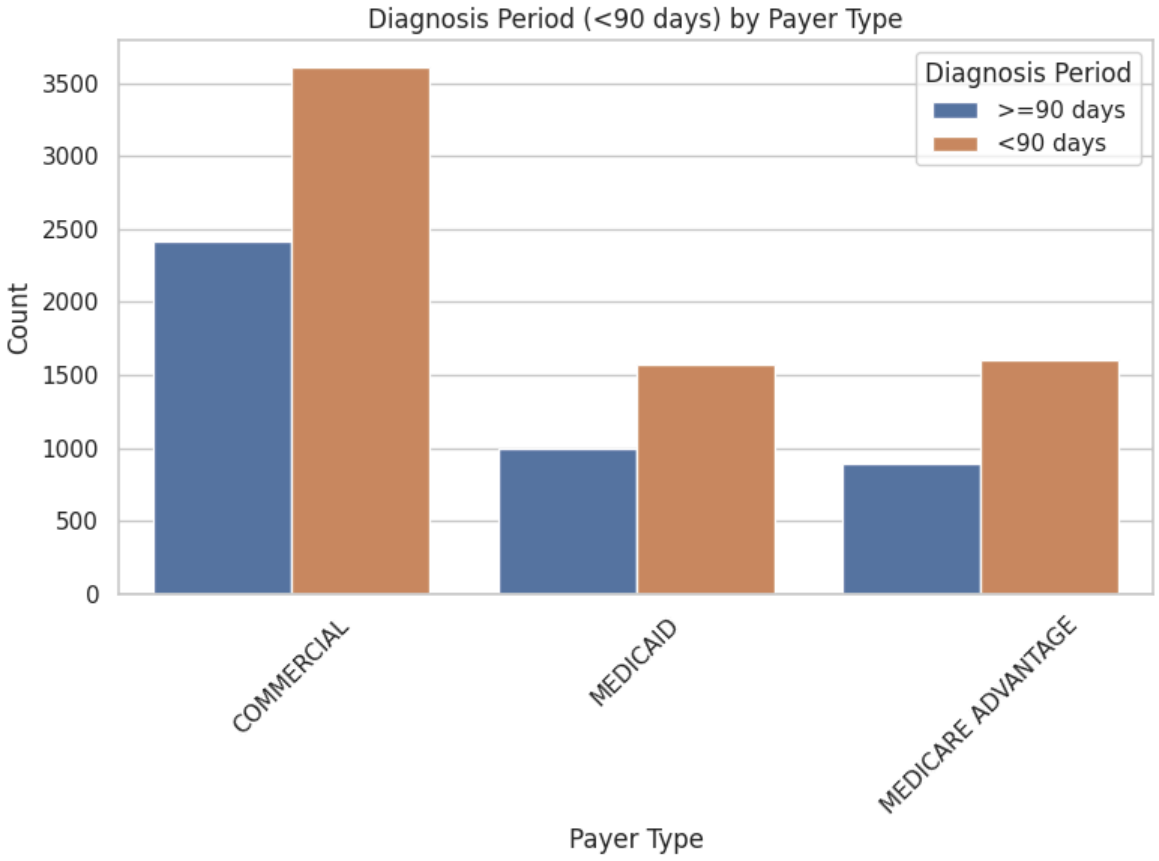
In [9]:
```
sns.set(style="whitegrid")

#Visualization 1: Diagnosis Period by Patient Race
plt.figure(figsize=(10, 6))
sns.countplot(data=train, x='patient_race', hue='DiagPeriodL90D',
              order=train['patient_race'].value_counts().index)
plt.title("Diagnosis Period (<90 days) by Patient Race")
plt.xlabel("Patient Race")
plt.ylabel("Count")
plt.legend(title="Diagnosis Period", labels=[">=90 days", "<90 days"])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This bar chart compares the number of patients diagnosed within 90 days (<90 days) versus those diagnosed later (>=90 days) across different racial groups. We observe that White patients form the largest group overall and show a high count of diagnoses within 90 days. Other racial groups, such as Black, Hispanic, Asian, and Other, have smaller total counts and varying distributions between <90 days and >=90 days. This difference may suggest disparities in timely diagnosis across different demographic groups; however, further investigation is necessary to draw more definitive conclusions about any potential healthcare inequity.

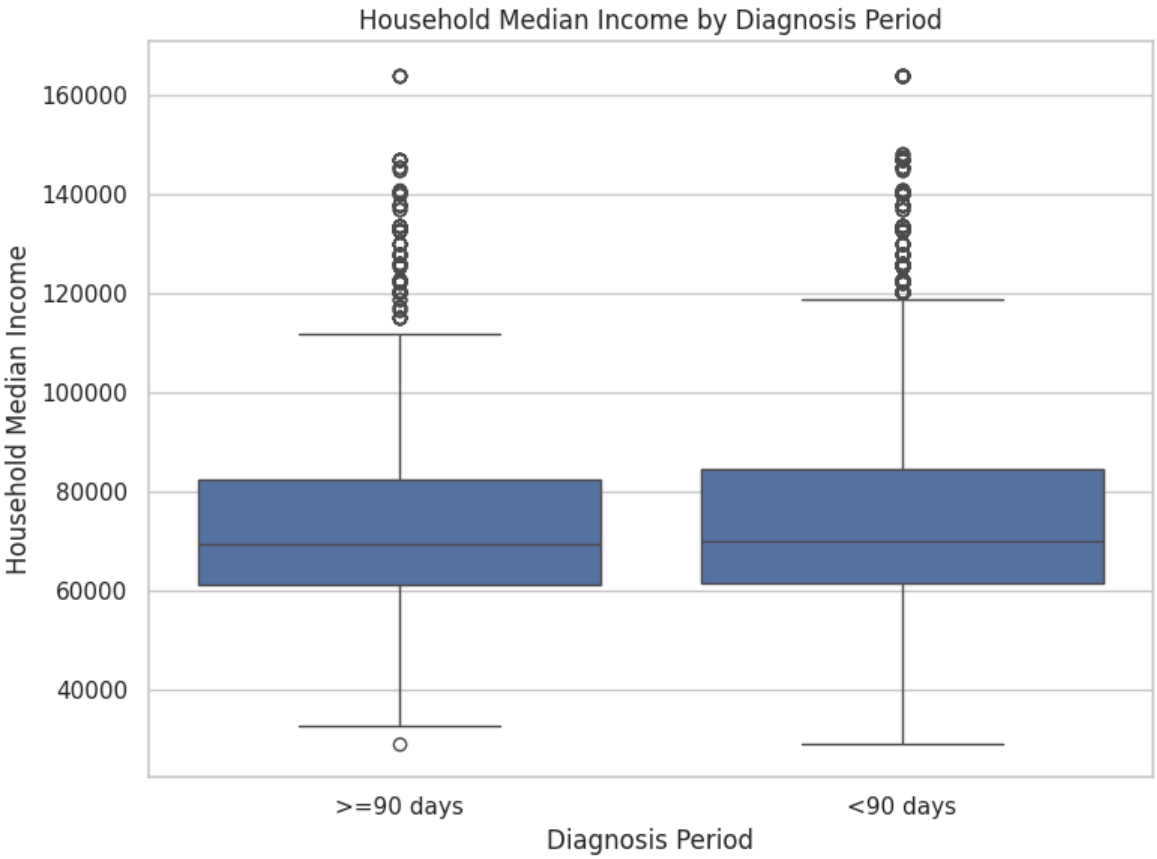
In [10]:
```
#Visualization 2: Diagnosis Period by Payer Type
plt.figure(figsize=(8, 6))
sns.countplot(data=train, x='payer_type', hue='DiagPeriodL90D',
              order=train['payer_type'].value_counts().index)
plt.title("Diagnosis Period (<90 days) by Payer Type")
plt.xlabel("Payer Type")
```

```
plt.ylabel("Count")
plt.legend(title="Diagnosis Period", labels=[">=90 days", "<90 days"])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This chart shows how many patients were diagnosed within 90 days (<90 days) versus those diagnosed later (>=90 days), broken down by payer type. We can see that Commercial insurance covers the largest group of patients overall, and in that group, more individuals receive a diagnosis in less than 90 days than those who wait 90 days or longer. Similarly, for Medicaid and Medicare Advantage, there are still more patients diagnosed in under 90 days, although the total numbers are smaller.

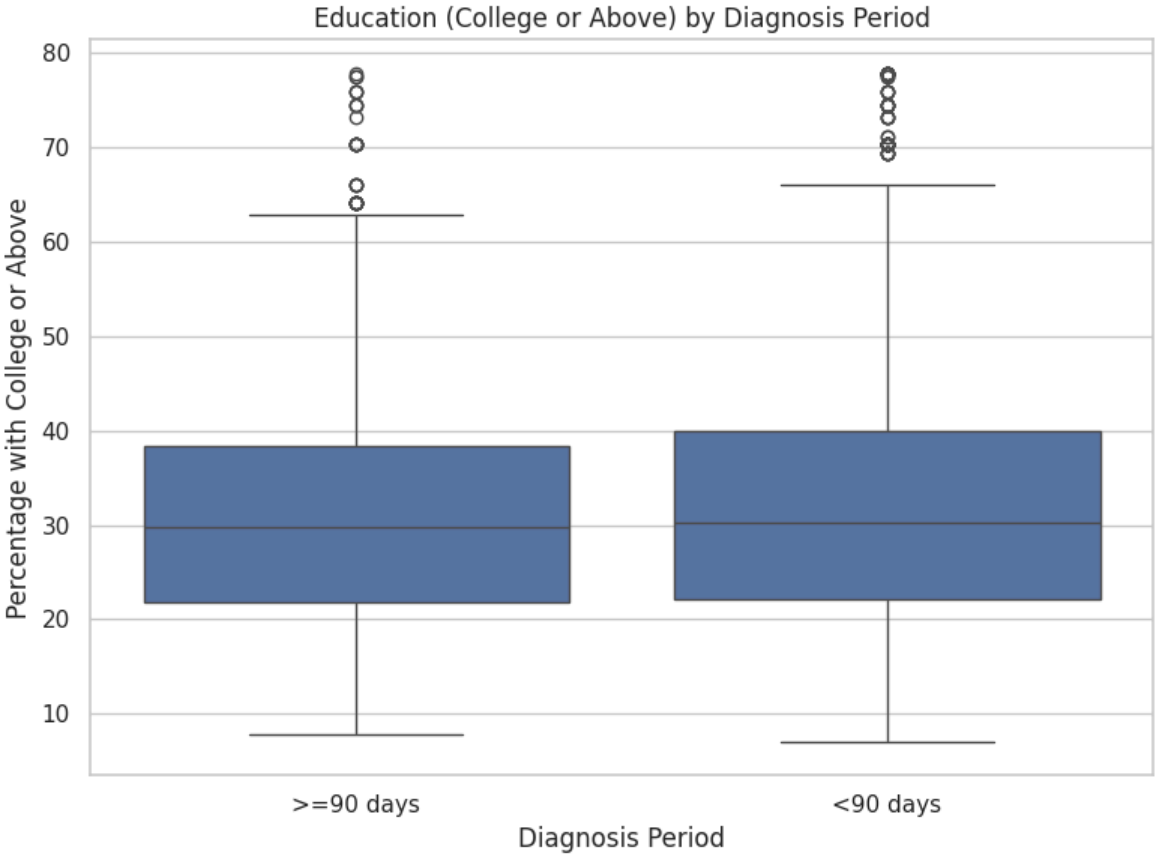
In [11]:
```
#Visualization 3: Household Median Income by Diagnosis Period
plt.figure(figsize=(8, 6))
sns.boxplot(data=train, x='DiagPeriodL90D', y='income_household_median')
plt.title("Household Median Income by Diagnosis Period")
plt.xlabel("Diagnosis Period")
plt.ylabel("Household Median Income")

plt.xticks([0, 1], [">=90 days", "<90 days"])
plt.tight_layout()
plt.show()
```



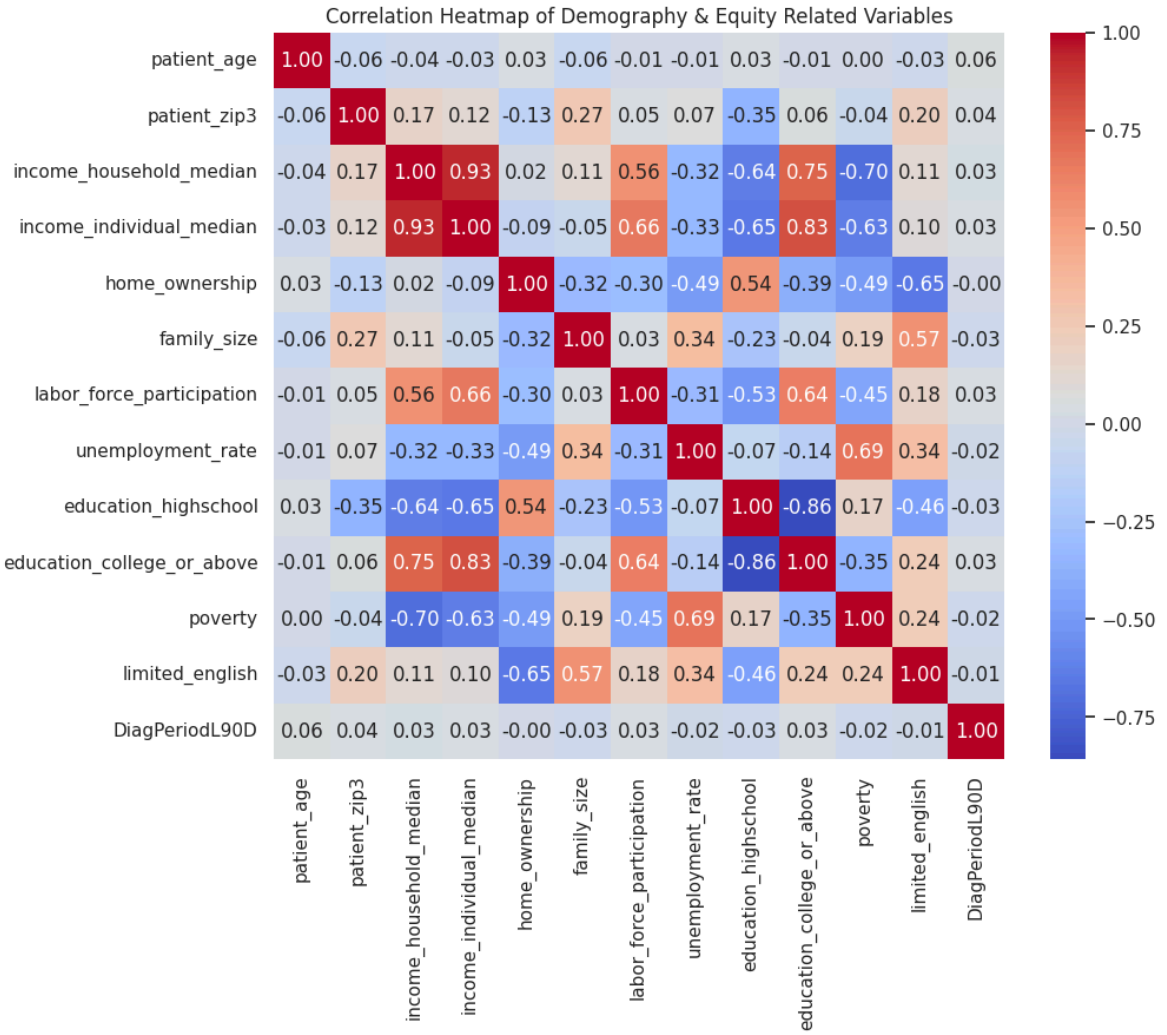
This boxplot compares household median income for patients diagnosed within 90 days versus those diagnosed after 90 days. Each box shows how incomes are spread out (from the 25th to the 75th percentile), with the line in the middle representing the median.

We see that the two groups have somewhat similar income ranges, and there are outliers with very high household incomes in both groups.

In [12]:
```
#Visualization 4: Education (College or Above) by Diagnosis Period

plt.figure(figsize=(8, 6))
sns.boxplot(data=train, x='DiagPeriodL90D', y='education_college_or_above')
plt.title("Education (College or Above) by Diagnosis Period")
plt.xlabel("Diagnosis Period")
plt.ylabel("Percentage with College or Above")
plt.xticks([0, 1], [">=90 days", "<90 days"])
plt.tight_layout()
plt.show()
```



In [13]:
```
#Visualization 5: Correlation Heatmap of Demography & Equity Related Variables
equity_cols = [
    'patient_age',
    'patient_zip3',
    'income_household_median',
    'income_individual_median',
    'home_ownership',
    'family_size',
    'labor_force_participation',
    'unemployment_rate',
    'education_highschool',
    'education_college_or_above',
    'poverty',
    'limited_english'
]

cols_for_heatmap = equity_cols + ['DiagPeriodL90D']

cols_for_heatmap = [col for col in cols_for_heatmap if col in train.columns]

#Compute the correlation matrix for the selected columns
corr_matrix = train[cols_for_heatmap].corr()

#Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Demography & Equity Related Variables")
plt.show()
```



Handling Missing Values

In [14]:
```
#Clean up data by making sure 'nan' string values are converted to actual NaN va
train.replace('nan', np.nan, inplace=True)

#Calculate total missing values
total_missing = train.isnull().sum().sum()
total_cells = np.product(train.shape)
missing_percentage = (total_missing / total_cells) * 100

#Print the summary of missing data
print("The total number of missing values are {0}{1}{2}{3}, which is {0}{4:.2f}%
```

The total number of missing values are **43292**, which is **4.04%** of total data.

In [15]:
```
#Calculate percentage of missing values for each column
missing_info = train.isna().mean() * 100
missing_info = missing_info[missing_info > 0].sort_values(ascending=False)

#Define text styles for printing
RED, BOLD, RESET = '\033[91m', '\033[1m', '\033[0m'

#Print formatted output
for column, missing_percentage in missing_info.items():
    missing_count = train[column].isna().sum()
```

```
    print(f"{BOLD}{column}{RESET} has {BOLD}{RED}{missing_count}{RESET} missing
          f"which is {BOLD}{RED}{missing_percentage:.2f}%{RESET} of the column."
```

metastatic_first_novel_treatment has **12882** missing values, which is **99.81%** of the column.
metastatic_first_novel_treatment_type has **12882** missing values, which is **99.81%** of the column.
bmi has **8965** missing values, which is **69.46%** of the column.
patient_race has **6385** missing values, which is **49.47%** of the column.
payer_type has **1803** missing values, which is **13.97%** of the column.
Region has **52** missing values, which is **0.40%** of the column.
Division has **52** missing values, which is **0.40%** of the column.
patient_state has **51** missing values, which is **0.40%** of the column.
PM25 has **29** missing values, which is **0.22%** of the column.
Ozone has **29** missing values, which is **0.22%** of the column.
N02 has **29** missing values, which is **0.22%** of the column.
income_household_75_to_100 has **4** missing values, which is **0.03%** of the column.
income_household_150_over has **4** missing values, which is **0.03%** of the column.
income_household_15_to_20 has **4** missing values, which is **0.03%** of the column.
income_household_20_to_25 has **4** missing values, which is **0.03%** of the column.
income_household_25_to_35 has **4** missing values, which is **0.03%** of the column.
income_household_35_to_50 has **4** missing values, which is **0.03%** of the column.
income_household_50_to_75 has **4** missing values, which is **0.03%** of the column.
income_household_100_to_150 has **4** missing values, which is **0.03%** of the column.
income_household_six_figure has **4** missing values, which is **0.03%** of the column.
income_household_under_5 has **4** missing values, which is **0.03%** of the column.
home_ownership has **4** missing values, which is **0.03%** of the column.
home_value has **4** missing values, which is **0.03%** of the column.
rent_median has **4** missing values, which is **0.03%** of the column.
rent_burden has **4** missing values, which is **0.03%** of the column.
farmer has **4** missing values, which is **0.03%** of the column.
self_employed has **4** missing values, which is **0.03%** of the column.
income_household_5_to_10 has **4** missing values, which is **0.03%** of the column.
income_household_10_to_15 has **4** missing values, which is **0.03%** of the column.
income_household_median has **4** missing values, which is **0.03%** of the column.
family_dual_income has **4** missing values, which is **0.03%** of the column.
limited_english has **4** missing values, which is **0.03%** of the column.
poverty has **4** missing values, which is **0.03%** of the column.
family_size has **4** missing values, which is **0.03%** of the column.
race_native has **1** missing values, which is **0.01%** of the column.
race_white has **1** missing values, which is **0.01%** of the column.
labor_force_participation has **1** missing values, which is **0.01%** of the column.
unemployment_rate has **1** missing values, which is **0.01%** of the column.
population has **1** missing values, which is **0.01%** of the column.
density has **1** missing values, which is **0.01%** of the column.
veteran has **1** missing values, which is **0.01%** of the column.
health_uninsured has **1** missing values, which is **0.01%** of the column.
commute_time has **1** missing values, which is **0.01%** of the column.
education_college_or_above has **1** missing values, which is **0.01%** of the column.
race_pacific has **1** missing values, which is **0.01%** of the column.
race_black has **1** missing values, which is **0.01%** of the column.
disabled has **1** missing values, which is **0.01%** of the column.
hispanic has **1** missing values, which is **0.01%** of the column.
race_asian has **1** missing values, which is **0.01%** of the column.
race_multiple has **1** missing values, which is **0.01%** of the column.
race_other has **1** missing values, which is **0.01%** of the column.
education_stem_degree has **1** missing values, which is **0.01%** of the column.
age_under_10 has **1** missing values, which is **0.01%** of the column.
education_graduate has **1** missing values, which is **0.01%** of the column.
education_bachelors has **1** missing values, which is **0.01%** of the column.
age_20s has **1** missing values, which is **0.01%** of the column.
age_30s has **1** missing values, which is **0.01%** of the column.
age_40s has **1** missing values, which is **0.01%** of the column.

```
age_50s has 1 missing values, which is 0.01% of the column.
age_60s has 1 missing values, which is 0.01% of the column.
age_70s has 1 missing values, which is 0.01% of the column.
age_over_80 has 1 missing values, which is 0.01% of the column.
male has 1 missing values, which is 0.01% of the column.
female has 1 missing values, which is 0.01% of the column.
married has 1 missing values, which is 0.01% of the column.
divorced has 1 missing values, which is 0.01% of the column.
never_married has 1 missing values, which is 0.01% of the column.
widowed has 1 missing values, which is 0.01% of the column.
age_median has 1 missing values, which is 0.01% of the column.
income_individual_median has 1 missing values, which is 0.01% of the column.
age_10_to_19 has 1 missing values, which is 0.01% of the column.
education_less_highschool has 1 missing values, which is 0.01% of the column.
education_highschool has 1 missing values, which is 0.01% of the column.
education_some_college has 1 missing values, which is 0.01% of the column.
housing_units has 1 missing values, which is 0.01% of the column.
```

Summary of Missing Data in Key Columns

The dataset exhibits significant data sparsity in several critical fields, particularly related to health treatments and demographic information. Here are the columns with the most substantial missing data:

- **Metastatic First Novel Treatment**:
  - **Missing Values**: 12,882 (99.81% of the column)
  - This column records the first novel treatment for metastatic cancer, crucial for advanced treatment analysis.
- **Metastatic First Novel Treatment Type**:
  - **Missing Values**: 12,882 (99.81% of the column)
  - Indicates the type of first novel treatment for metastatic cancer, similarly critical for understanding treatment patterns.
- **BMI (Body Mass Index)**:
  - **Missing Values**: 8,965 (69.46% of the column)
  - BMI is a key health indicator, and its absence can significantly impact studies related to patient health outcomes.
- **Patient Race**:
  - **Missing Values**: 6,385 (49.47% of the column)
  - Patient race is vital for demographic studies and understanding disparities in health outcomes.
- **Payer Type**:
  - **Missing Values**: 1,803 (13.97% of the column)
  - Reflects the type of health insurance coverage, important for financial and access-to-care analyses.

In [16]:
```
#Dropping columns with more than 65% missing values
threshold = 65
```

```
cols_to_drop = train.columns[train.isnull().mean() > (threshold / 100)]
print(cols_to_drop)
train.drop(columns=cols_to_drop, inplace=True)
```

```
Index(['bmi', 'metastatic_first_novel_treatment',
       'metastatic_first_novel_treatment_type'],
      dtype='object')
```

In [17]:
```
#Numeric columns: Impute with the median
for col in train.select_dtypes(include=['float64', 'int64']).columns:
    train[col].fillna(train[col].median(), inplace=True)

#Categorical columns: Impute with the mode
for col in train.select_dtypes(include=['object']).columns:
    train[col].fillna(train[col].mode()[0], inplace=True)
```

In [18]:
```
#Check for any remaining missing values in each column
remaining_missing = train.isnull().sum()
remaining_missing = remaining_missing[remaining_missing > 0]  # Filter columns t
if remaining_missing.empty:
    print("No missing values found in the dataset.")
else:
    print("Remaining Missing Values Per Column:")
    print(remaining_missing)
```

```
No missing values found in the dataset.
```

In [19]:
```
train.describe().T.style.background_gradient().format("{:.2f}")
```

Out[19]:

	count	mean	std	min	25%	
patient_id	12906.00	547381.20	260404.96	100063.00	321517.00	54
patient_zip3	12906.00	573.75	275.45	101.00	331.00	
patient_age	12906.00	59.18	13.34	18.00	50.00	
population	12906.00	20744.32	13886.37	635.55	9463.90	19
density	12906.00	1581.88	2966.20	0.92	171.86	
age_median	12906.00	40.50	4.04	20.60	37.13	
age_under_10	12906.00	11.12	1.51	0.00	10.16	
age_10_to_19	12906.00	12.95	1.92	6.31	11.74	
age_20s	12906.00	13.29	3.35	5.92	11.01	
age_30s	12906.00	12.86	2.32	1.50	11.29	
age_40s	12906.00	12.07	1.25	0.80	11.34	
age_50s	12906.00	13.44	1.64	0.00	12.30	
age_60s	12906.00	12.62	2.57	0.20	10.62	
age_70s	12906.00	7.65	2.15	0.00	6.01	
age_over_80	12906.00	4.00	1.24	0.00	3.28	
male	12906.00	50.10	1.66	39.73	49.16	
female	12906.00	49.90	1.66	38.40	49.06	
married	12906.00	47.68	7.49	0.90	42.93	
divorced	12906.00	12.67	2.05	0.20	11.16	
never_married	12906.00	33.83	8.02	13.44	27.53	
widowed	12906.00	5.81	1.54	0.00	4.76	
family_size	12906.00	3.20	0.22	2.55	3.04	
family_dual_income	12906.00	51.85	6.81	19.31	47.64	
income_household_median	12906.00	74372.95	20709.54	29222.00	61284.45	69
income_household_under_5	12906.00	3.27	1.45	0.75	2.26	
income_household_5_to_10	12906.00	2.51	1.34	0.36	1.54	
income_household_10_to_15	12906.00	4.14	1.77	1.02	2.90	
income_household_15_to_20	12906.00	3.93	1.45	1.03	2.92	
income_household_20_to_25	12906.00	4.07	1.35	1.10	3.17	
income_household_25_to_35	12906.00	8.40	2.21	2.65	6.78	
income_household_35_to_50	12906.00	11.56	2.58	1.70	9.90	
income_household_50_to_75	12906.00	16.89	2.72	4.95	15.34	
income_household_75_to_100	12906.00	12.66	1.83	4.73	11.35	

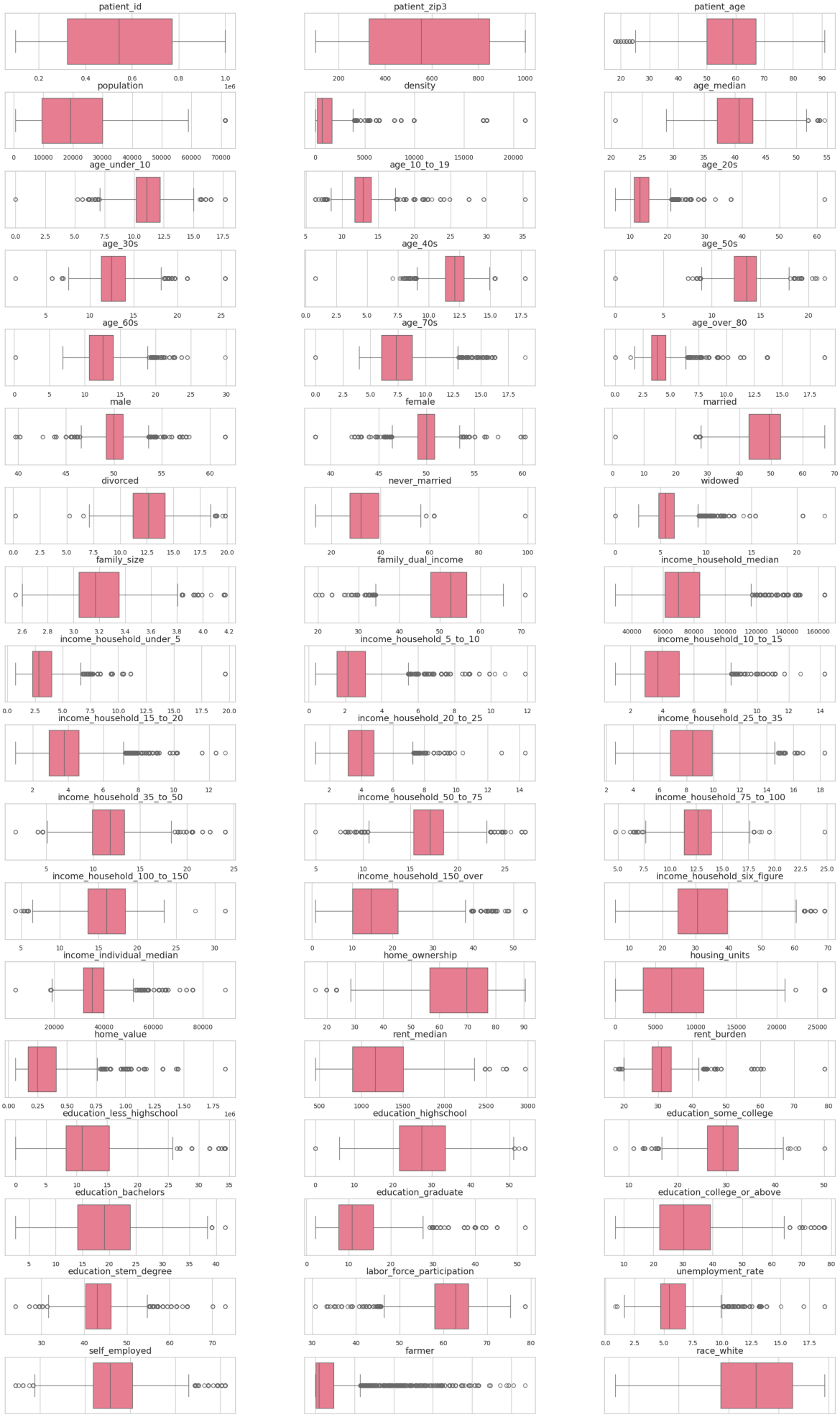
	count	mean	std	min	25%	
income_household_100_to_150	12906.00	15.83	3.17	4.29	13.62	
income_household_150_over	12906.00	16.72	8.94	0.84	10.07	
income_household_six_figure	12906.00	32.55	11.13	5.69	24.57	
income_individual_median	12906.00	36520.42	8194.85	4316.00	31575.26	3
home_ownership	12906.00	66.00	14.12	15.85	56.53	
housing_units	12906.00	7575.66	4916.72	0.00	3447.51	
home_value	12906.00	339788.66	251663.80	60629.00	167760.45	24
rent_median	12906.00	1237.28	427.77	448.40	895.79	
rent_burden	12906.00	31.34	4.78	17.42	28.20	
education_less_highschool	12906.00	11.99	5.14	0.00	8.25	
education_highschool	12906.00	27.56	7.99	0.00	21.64	
education_some_college	12906.00	28.92	4.89	7.20	26.03	
education_bachelors	12906.00	19.27	6.17	2.47	14.02	
education_graduate	12906.00	12.26	5.94	2.09	7.65	
education_college_or_above	12906.00	31.53	11.63	7.05	22.09	
education_stem_degree	12906.00	43.39	4.64	23.91	40.35	
labor_force_participation	12906.00	61.62	5.95	30.70	57.94	
unemployment_rate	12906.00	5.95	1.95	0.82	4.72	
self_employed	12906.00	13.21	3.41	2.26	10.87	
farmer	12906.00	1.86	3.06	0.00	0.04	
race_white	12906.00	69.72	17.96	14.50	56.76	
race_black	12906.00	11.45	12.53	0.06	2.24	
race_asian	12906.00	5.47	6.74	0.00	1.10	
race_native	12906.00	0.90	2.50	0.00	0.21	
race_pacific	12906.00	0.14	0.51	0.00	0.02	
race_other	12906.00	5.68	6.25	0.00	1.31	
race_multiple	12906.00	6.64	3.54	0.43	3.92	
hispanic	12906.00	18.46	17.03	0.19	4.71	
disabled	12906.00	13.34	3.69	4.60	10.27	
poverty	12906.00	13.41	5.22	3.43	9.66	
limited_english	12906.00	4.47	4.84	0.00	0.99	
commute_time	12906.00	27.98	5.08	12.46	24.93	
health_uninsured	12906.00	8.58	4.20	2.44	5.62	

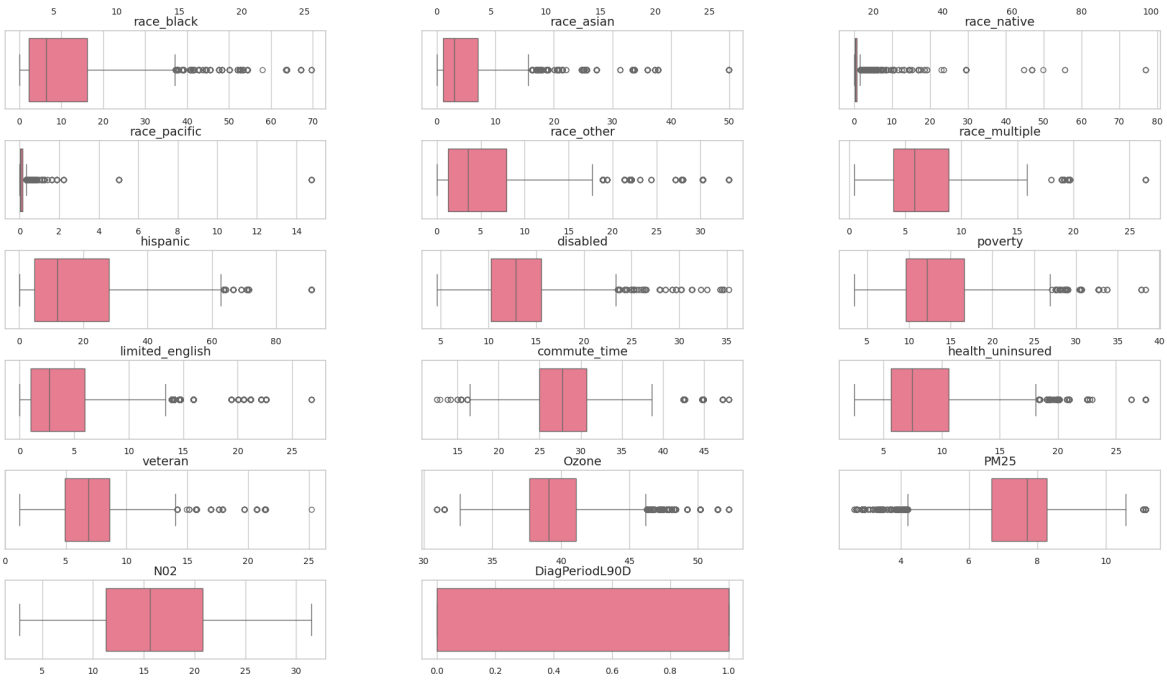
	count	mean	std	min	25%
veteran	12906.00	7.08	3.11	1.20	4.93
Ozone	12906.00	39.82	3.56	30.94	37.70
PM25	12906.00	7.48	1.51	2.64	6.65
N02	12906.00	16.10	5.84	2.76	11.28
DiagPeriodL90D	12906.00	0.62	0.48	0.00	0.00

Checking the distribution of numerical columns

In [20]:
```
#Selecting numerical columns only
train_num = train.select_dtypes(include=['int64', 'float64'])

#Setting plot parameters
plt.rcParams['axes.facecolor'] = 'white'
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10

#Define the figure for plotting
fig, axes = plt.subplots(nrows=len(train_num.columns) // 3 + 1, ncols=3, figsize
fig.suptitle('Boxplot of All Numerical Columns', fontsize=18, fontweight='bold')
fig.subplots_adjust(top=0.95, hspace=0.4, wspace=0.3)

#Loop through the numerical columns and create a boxplot for each
for i, col in enumerate(train_num.columns):
    row = i // 3
    col_num = i % 3
    sns.boxplot(data=train, x=train_num[col], ax=axes[row, col_num], palette="hu
    axes[row, col_num].set_xlabel('')
    axes[row, col_num].set_ylabel('')

#Hide empty subplots
for j in range(i + 1, len(axes.flatten())):
    axes.flatten()[j].set_visible(False)

plt.show()
```

Boxplot of All Numerical Columns





Checking the distribution of Categorical Columns in the dataset

In [21]:
```
#Function to print value counts in markdown table format
def print_markdown_table(df, column_name):
    print(f"Value counts for column '{column_name}':")
    print("+--------------------+-------+------------+")
    print("|       Value        | Count | Percentage |")
    print("+--------------------+-------+------------+")

    #Calculating value counts and percentages
    counts = df[column_name].value_counts(dropna=False).head(10)
    percentages = df[column_name].value_counts(normalize=True, dropna=False) .he

    for value, count in counts.items():
        percentage = percentages[value]
        print(f"| {value:<18} | {count:<5} | {percentage:10.2f}% |")

    print("+--------------------+-------+------------+")
    print("=" * 48)


#List of categorical columns to process
categorical_cols = train.select_dtypes(include=['object']).columns

#Generate markdown tables for each categorical column
for col in categorical_cols:
    if col in train.columns:
        print_markdown_table(train, col)
```

Value counts for column 'patient_race':

Value	Count	Percentage
White	9973	77.27%
Black	1056	8.18%
Hispanic	829	6.42%
Other	683	5.29%
Asian	365	2.83%

==========

Value counts for column 'payer_type':

Value	Count	Percentage
COMMERCIAL	7835	60.71%
MEDICAID	2569	19.91%
MEDICARE ADVANTAGE	2502	19.39%

==========

Value counts for column 'patient_state':

Value	Count	Percentage
CA	2489	19.29%
TX	1155	8.95%
NY	1041	8.07%
MI	858	6.65%
IL	782	6.06%
OH	754	5.84%
FL	609	4.72%
GA	551	4.27%
PA	483	3.74%
MN	377	2.92%

==========

Value counts for column 'patient_gender':

Value	Count	Percentage
F	12906	100.00%

==========

Value counts for column 'breast_cancer_diagnosis_code':

Value	Count	Percentage
1749	1982	15.36%
C50911	1797	13.92%
C50912	1712	13.27%
C50919	1467	11.37%
C50411	978	7.58%
C50412	877	6.80%
C50811	491	3.80%
C50812	419	3.25%
1744	389	3.01%
1748	307	2.38%

==========

Value counts for column 'breast_cancer_diagnosis_desc':

```
+--------------------+-------+------------+
|       Value        | Count | Percentage |
+--------------------+-------+------------+
| Malignant neoplasm of breast (female), unspecified | 1982  |      15.36% |
| Malignant neoplasm of unsp site of right female breast | 1797  |      13.92% |
| Malignant neoplasm of unspecified site of left female breast | 1712  |      13.
27% |
| Malignant neoplasm of unsp site of unspecified female breast | 1467  |      11.
37% |
| Malig neoplm of upper-outer quadrant of right female breast | 978   |       7.5
8% |
| Malig neoplasm of upper-outer quadrant of left female breast | 877   |       6.
80% |
| Malignant neoplasm of ovrlp sites of right female breast | 491   |       3.80%
|
| Malignant neoplasm of ovrlp sites of left female breast | 419   |       3.25% |
| Malignant neoplasm of upper-outer quadrant of female breast | 389   |       3.0
1% |
| Malignant neoplasm of other specified sites of female breast | 307   |       2.
38% |
+--------------------+-------+------------+
================================================
Value counts for column 'metastatic_cancer_diagnosis_code':
+--------------------+-------+------------+
|       Value        | Count | Percentage |
+--------------------+-------+------------+
| C773               | 7052  |      54.64% |
| C7951              | 1842  |      14.27% |
| C779               | 764   |       5.92% |
| C7981              | 467   |       3.62% |
| C7800              | 414   |       3.21% |
| C787               | 362   |       2.80% |
| C7989              | 344   |       2.67% |
| C799               | 282   |       2.19% |
| C7931              | 282   |       2.19% |
| C792               | 166   |       1.29% |
+--------------------+-------+------------+
================================================
Value counts for column 'Region':
+--------------------+-------+------------+
|       Value        | Count | Percentage |
+--------------------+-------+------------+
| South              | 3971  |      30.77% |
| West               | 3735  |      28.94% |
| Midwest            | 3650  |      28.28% |
| Northeast          | 1550  |      12.01% |
+--------------------+-------+------------+
================================================
Value counts for column 'Division':
+--------------------+-------+------------+
|       Value        | Count | Percentage |
+--------------------+-------+------------+
| East North Central | 2975  |      23.05% |
| Pacific            | 2754  |      21.34% |
| South Atlantic     | 1972  |      15.28% |
| Middle Atlantic    | 1545  |      11.97% |
| West South Central | 1450  |      11.24% |
| Mountain           | 981   |       7.60% |
| West North Central | 727   |       5.63% |
| East South Central | 497   |       3.85% |
```

```
| New England         | 5     |       0.04% |
+--------------------+-------+------------+
================================================
```

We will drop the patient gender column as there is only 1 gender present and it adds no variability to the data.

In [22]:
```
#Drop the 'patient_gender' and 'patient_id' column
train.drop('patient_gender', axis=1, inplace=True)
train.drop('patient_id', axis=1, inplace=True)
```

Approach 1

Using simple models like Logistic Regression, Random Forest, SVM, Gradient Boosting.

Model Training and Evaluation

In [23]:
```
#Selecting categorical columns
categorical_cols = train.columns[train.dtypes == object]

#Apply Label Encoding to each categorical column
for col in categorical_cols:
    le = LabelEncoder()
    train[col] = le.fit_transform(train[col].astype(str))

#Define features and target
X = train.drop(['DiagPeriodL90D'], axis=1)
y = train['DiagPeriodL90D']

#Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

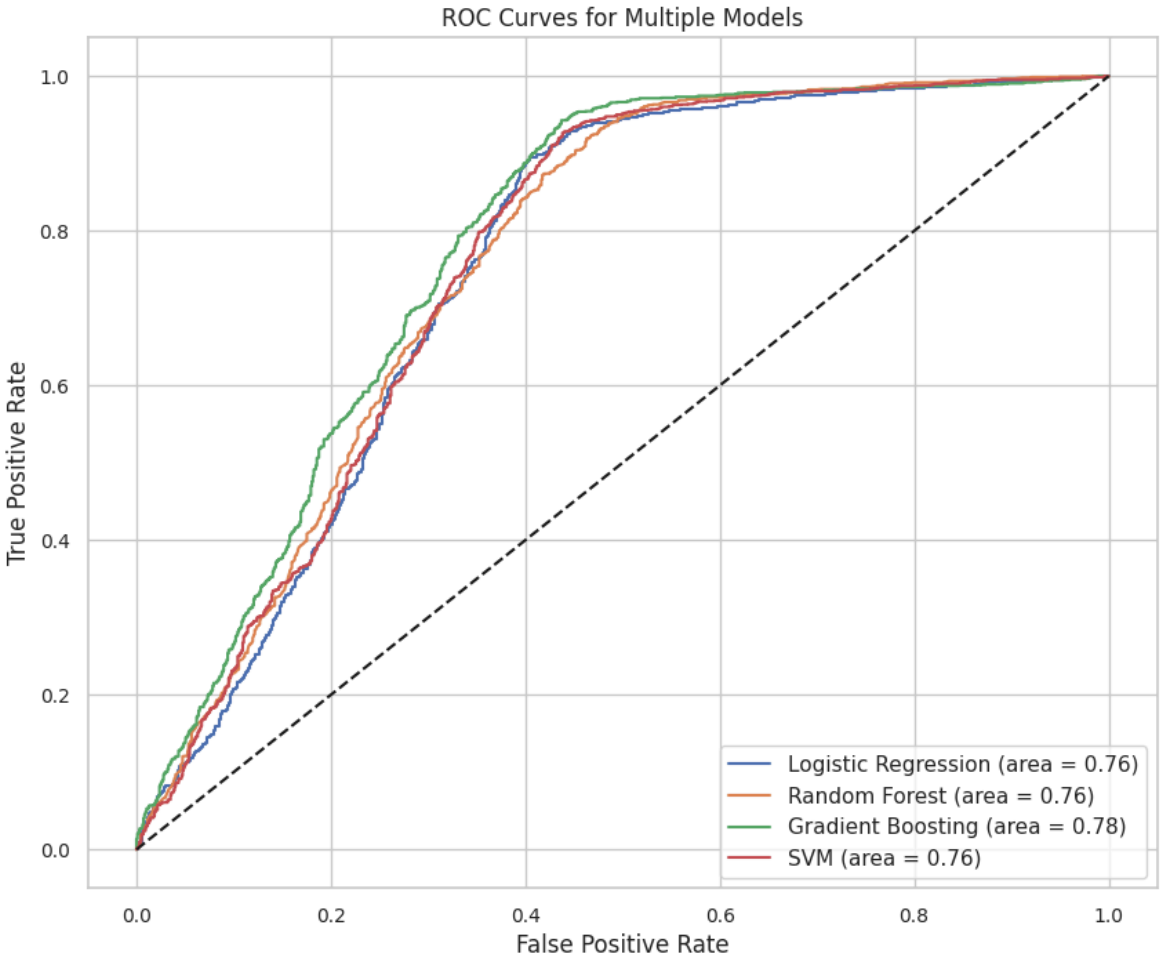
In [24]:
```
#List of models to evaluate
models = [
    ('Logistic Regression', LogisticRegression(random_state=42, max_iter=1000)),
    ('Random Forest', RandomForestClassifier(n_estimators=1000, random_state=42)
    ('Gradient Boosting', GradientBoostingClassifier(n_estimators=1000, random_s
    ('SVM', make_pipeline(StandardScaler(), SVC(probability=True, random_state=4
]
```

In [25]:
```
#Dictionary to hold ROC data
roc_data = {}

#Train each model and calculate ROC AUC
for name, model in models:
    model.fit(X_train, y_train)
    probs = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)
    roc_data[name] = (fpr, tpr, roc_auc)
    print(f"{name} AUC: {roc_auc:.2f}")
    print(classification_report(y_test, model.predict(X_test)))
```

```
Logistic Regression AUC: 0.76
              precision    recall  f1-score   support

           0       0.72      0.61      0.66       970
           1       0.79      0.86      0.82      1612

    accuracy                           0.76      2582
   macro avg       0.75      0.73      0.74      2582
weighted avg       0.76      0.76      0.76      2582

Random Forest AUC: 0.76
              precision    recall  f1-score   support

           0       0.73      0.58      0.65       970
           1       0.78      0.87      0.82      1612

    accuracy                           0.76      2582
   macro avg       0.75      0.73      0.74      2582
weighted avg       0.76      0.76      0.76      2582

Gradient Boosting AUC: 0.78
              precision    recall  f1-score   support

           0       0.79      0.58      0.67       970
           1       0.78      0.91      0.84      1612

    accuracy                           0.79      2582
   macro avg       0.79      0.75      0.76      2582
weighted avg       0.79      0.79      0.78      2582

SVM AUC: 0.76
              precision    recall  f1-score   support

           0       0.77      0.58      0.66       970
           1       0.78      0.90      0.83      1612

    accuracy                           0.78      2582
   macro avg       0.77      0.74      0.75      2582
weighted avg       0.78      0.78      0.77      2582
```

In [26]:

```python
#Plot ROC Curves for all models
plt.figure(figsize=(10, 8))
for name, (fpr, tpr, roc_auc) in roc_data.items():
    plt.plot(fpr, tpr, label=f'{name} (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Multiple Models')
plt.legend(loc='lower right')
plt.show()
```



Model Training and Evaluation

Overview

We train and evaluate several machine learning models to predict whether the diagnosis period for a patient is less than 90 days based on their health and demographic data. The models include Logistic Regression, Random Forest, Gradient Boosting, and Support Vector Machine (SVM).

Training Process

1. **Feature Selection**: We first separate our features (`X`) from the target variable (`y`), where `y` is the diagnosis period classification.
2. **Data Splitting**: The dataset is split into training and testing sets, maintaining a 20% proportion for the test set. This split is stratified by the target variable to ensure that the distribution of the target is similar across both training and test datasets.
3. **Model Training**: Each model is trained on the training data. We use a loop to process each model, where the model is fitted using the training features and labels.
4. **Prediction and Evaluation**: Post-training, each model predicts the probabilities on the test data. The probabilities are then used to calculate the ROC AUC score, which serves as the evaluation metric. This metric helps in assessing the models' ability to distinguish between the two classes (less than 90 days and not).

Results Visualization

- **ROC Curve**: For each model, a Receiver Operating Characteristic (ROC) curve is plotted. This curve illustrates the diagnostic ability of each classifier by plotting the true positive rate against the false positive rate at various threshold settings.
- **Comparison**: The ROC curves for all models are displayed together to compare their performance visually. The area under the ROC curve (AUC) is used to quantify the overall ability of each model to discriminate between the outcomes.

Therefore, Gradient Boosting Model gave a better performance of the total 4 models.

Approach 2

In this approach, I expanded the model selection to include a variety of classifiers and implemented hyperparameter tuning to optimize their performance.

Note: I trained the CatBoost model without encoding categorical variables to allow for straightforward analysis of feature importance directly from the raw features. This helps in clearly understanding which features are most influential without the complexity introduced by encoded variables.

Following the CatBoost analysis, I trained additional models using one-hot encoded features to prepare for a comprehensive comparison and feature importance evaluation. This step ensures that all nuances of categorical data across different models are captured.

Lastly, I employed an ensemble technique to see if combining the predictions from multiple models would enhance the overall accuracy. This ensemble method uses the strengths of individual models to achieve more reliable prediction outcome.

In [27]:
```
#Loading the original dataset
df_original = pd.read_csv('/content/training.csv')

df_preproc = df_original.copy()

#Handling Missing Values
threshold = 0.65
cols_to_drop = df_preproc.columns[df_preproc.isnull().mean() > threshold]
df_preproc.drop(columns=cols_to_drop, inplace=True)

#Identifying numerical and categorical columns
numeric_cols = df_preproc.select_dtypes(include=['number']).columns.tolist()
categorical_cols = df_preproc.select_dtypes(include=['object']).columns.tolist()

#Handling Missing values
for col in numeric_cols:
    df_preproc[col].fillna(df_preproc[col].median(), inplace=True)

for col in categorical_cols:
    df_preproc[col].fillna(df_preproc[col].mode()[0], inplace=True)
```

In [28]:
```
#Drop the 'patient_gender' and 'patient_id' column
df_preproc.drop(columns=['patient_gender','patient_id'], axis=1, inplace=True)
```

In [29]:
```
df_preproc.describe().T.style.background_gradient().format("{:.2f}")
```

Out[29]:

	count	mean	std	min	25%	
patient_zip3	12906.00	573.75	275.45	101.00	331.00	5
patient_age	12906.00	59.18	13.34	18.00	50.00	
population	12906.00	20744.32	13886.37	635.55	9463.90	19
density	12906.00	1581.88	2966.20	0.92	171.86	
age_median	12906.00	40.50	4.04	20.60	37.13	
age_under_10	12906.00	11.12	1.51	0.00	10.16	
age_10_to_19	12906.00	12.95	1.92	6.31	11.74	
age_20s	12906.00	13.29	3.35	5.92	11.01	
age_30s	12906.00	12.86	2.32	1.50	11.29	
age_40s	12906.00	12.07	1.25	0.80	11.34	
age_50s	12906.00	13.44	1.64	0.00	12.30	
age_60s	12906.00	12.62	2.57	0.20	10.62	
age_70s	12906.00	7.65	2.15	0.00	6.01	
age_over_80	12906.00	4.00	1.24	0.00	3.28	
male	12906.00	50.10	1.66	39.73	49.16	
female	12906.00	49.90	1.66	38.40	49.06	
married	12906.00	47.68	7.49	0.90	42.93	
divorced	12906.00	12.67	2.05	0.20	11.16	
never_married	12906.00	33.83	8.02	13.44	27.53	
widowed	12906.00	5.81	1.54	0.00	4.76	
family_size	12906.00	3.20	0.22	2.55	3.04	
family_dual_income	12906.00	51.85	6.81	19.31	47.64	
income_household_median	12906.00	74372.95	20709.54	29222.00	61284.45	69
income_household_under_5	12906.00	3.27	1.45	0.75	2.26	
income_household_5_to_10	12906.00	2.51	1.34	0.36	1.54	
income_household_10_to_15	12906.00	4.14	1.77	1.02	2.90	
income_household_15_to_20	12906.00	3.93	1.45	1.03	2.92	
income_household_20_to_25	12906.00	4.07	1.35	1.10	3.17	
income_household_25_to_35	12906.00	8.40	2.21	2.65	6.78	
income_household_35_to_50	12906.00	11.56	2.58	1.70	9.90	
income_household_50_to_75	12906.00	16.89	2.72	4.95	15.34	
income_household_75_to_100	12906.00	12.66	1.83	4.73	11.35	
income_household_100_to_150	12906.00	15.83	3.17	4.29	13.62	

	count	mean	std	min	25%	
income_household_150_over	12906.00	16.72	8.94	0.84	10.07	
income_household_six_figure	12906.00	32.55	11.13	5.69	24.57	
income_individual_median	12906.00	36520.42	8194.85	4316.00	31575.26	352
home_ownership	12906.00	66.00	14.12	15.85	56.53	
housing_units	12906.00	7575.66	4916.72	0.00	3447.51	69
home_value	12906.00	339788.66	251663.80	60629.00	167760.45	247
rent_median	12906.00	1237.28	427.77	448.40	895.79	11
rent_burden	12906.00	31.34	4.78	17.42	28.20	
education_less_highschool	12906.00	11.99	5.14	0.00	8.25	
education_highschool	12906.00	27.56	7.99	0.00	21.64	
education_some_college	12906.00	28.92	4.89	7.20	26.03	
education_bachelors	12906.00	19.27	6.17	2.47	14.02	
education_graduate	12906.00	12.26	5.94	2.09	7.65	
education_college_or_above	12906.00	31.53	11.63	7.05	22.09	
education_stem_degree	12906.00	43.39	4.64	23.91	40.35	
labor_force_participation	12906.00	61.62	5.95	30.70	57.94	
unemployment_rate	12906.00	5.95	1.95	0.82	4.72	
self_employed	12906.00	13.21	3.41	2.26	10.87	
farmer	12906.00	1.86	3.06	0.00	0.04	
race_white	12906.00	69.72	17.96	14.50	56.76	
race_black	12906.00	11.45	12.53	0.06	2.24	
race_asian	12906.00	5.47	6.74	0.00	1.10	
race_native	12906.00	0.90	2.50	0.00	0.21	
race_pacific	12906.00	0.14	0.51	0.00	0.02	
race_other	12906.00	5.68	6.25	0.00	1.31	
race_multiple	12906.00	6.64	3.54	0.43	3.92	
hispanic	12906.00	18.46	17.03	0.19	4.71	
disabled	12906.00	13.34	3.69	4.60	10.27	
poverty	12906.00	13.41	5.22	3.43	9.66	
limited_english	12906.00	4.47	4.84	0.00	0.99	
commute_time	12906.00	27.98	5.08	12.46	24.93	
health_uninsured	12906.00	8.58	4.20	2.44	5.62	
veteran	12906.00	7.08	3.11	1.20	4.93	

	count	mean	std	min	25%
Ozone	12906.00	39.82	3.56	30.94	37.70
PM25	12906.00	7.48	1.51	2.64	6.65
N02	12906.00	16.10	5.84	2.76	11.28
DiagPeriodL90D	12906.00	0.62	0.48	0.00	0.00

In [30]:
```
#Dropping target variable from features
if 'DiagPeriodL90D' in df_preproc.columns:
    X = df_preproc.drop(['DiagPeriodL90D'], axis=1)
    y = df_preproc['DiagPeriodL90D'].values
else:
    X = df_preproc.copy()
    y = None

#Separating categorical and numeric features
categorical_columns = X.select_dtypes(include=['object']).columns.tolist()
numeric_columns = X.select_dtypes(include=['number']).columns.tolist()

#Preprocessor for both types of columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_columns),
        ('cat', OneHotEncoder(), categorical_columns)
    ]
)

#Apply transformation
X_processed = preprocessor.fit_transform(X)
```

Data Preparation and Preprocessing

Dropping Target Variable:

- The target variable `DiagPeriodL90D` is removed from the training set to separate features from the target.

Feature Categorization:

- Features are categorized into numerical and categorical types to apply appropriate preprocessing.

Setting up Preprocessing:

- A `ColumnTransformer` is configured to handle different types of data:
  - Numerical data is passed through without changes.
  - Categorical data is transformed using `OneHotEncoder` to prepare it for model training.

Data Transformation:

- The `ColumnTransformer` applies these transformations to the feature set, and creates a processed dataset ready for training.
- The target variable is extracted into a separate array for model training.

In [31]:
```
#Setting up the model base and hyperparameters
models_params = {
    'Logistic Regression': {
        'model': LogisticRegression(max_iter=1000, random_state=42),
        'params': {'C': [0.1, 1, 10]}
    },
    'Random Forest': {
        'model': RandomForestClassifier(random_state=42),
        'params': {'n_estimators': [100, 200], 'max_depth': [10, 20]}
    },
    'Gradient Boosting': {
        'model': GradientBoostingClassifier(random_state=42),
        'params': {'n_estimators': [100, 200], 'learning_rate': [0.05, 0.1]}
    },
    'SVM': {
    'model': Pipeline([
        ('scaler', StandardScaler(with_mean=False)),
        ('svc', SVC(probability=True, random_state=42))
    ]),
    'params': {
        'svc__C': [0.1, 1],
        'svc__gamma': ['scale', 'auto']
    }
},
    'XGBoost': {
        'model': XGBClassifier(use_label_encoder=False, eval_metric='logloss', r
        'params': {'n_estimators': [100, 200], 'learning_rate': [0.05, 0.1]}
    },
    'AdaBoost': {
        'model': AdaBoostClassifier(random_state=42),
        'params': {'n_estimators': [50, 100], 'learning_rate': [0.5, 1.0]}
    },
    'CatBoost': {
        'model': CatBoostClassifier(verbose=0, random_state=42),
        'params': {'iterations': [100, 200], 'learning_rate': [0.05, 0.1]}
    },
    'LightGBM': {
        'model': LGBMClassifier(random_state=42),
        'params': {'n_estimators': [100, 200], 'learning_rate': [0.05, 0.1]}
    }
}
```

Model Configuration and Hyperparameter Setup

Setting Up Different Classification Models with Hyperparameters;

- **Logistic Regression**: Tuned for regularization strength with values [0.1, 1, 10].
- **Random Forest**: Configured with a range of trees [100, 200] and maximum depths [10, 20].
- **Gradient Boosting**: Adjusts for number of stages [100, 200] and learning rates [0.05, 0.1].

- **SVM**: Uses a pipeline for scaling and SVM classification, tuning `C` [0.1, 1] and `gamma` [scale, auto].
- **XGBoost**: Focuses on the number of trees [100, 200] and learning rate adjustments [0.05, 0.1].
- **AdaBoost**: Set for different numbers of estimators [50, 100] and learning rates [0.5, 1.0].
- **CatBoost**: Configured for iterations [100, 200] and learning rate [0.05, 0.1].
- **LightGBM**: Parameters include number of trees [100, 200] and learning rate [0.05, 0.1].

Each model is integrated into a systematic parameter optimization framework using **GridSearchCV**. This method performs exhaustive searching over specified parameter values for each model. GridSearchCV processes each combination of parameters, applies cross-validation to evaluate each set, and determines the configuration that produces the best results based on the ROC AUC metric. This approach ensures that each model is finely tuned to provide optimal performance on the given dataset.

In [32]:
```
#Training Catboost without One Hot Encoding
#Identify categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
cat_features_indices = [X.columns.get_loc(col) for col in categorical_cols]

#Define CatBoost model parameters
model_params = {
    'iterations': [100, 200],
    'learning_rate': [0.05, 0.1]
}

#Instantiate cat_boost
cat_model = CatBoostClassifier(
    verbose=0,
    random_state=42,
    cat_features=cat_features_indices
)

#Set up GridSearchCV for hyperparameter tuning
cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
grid_search = GridSearchCV(
    estimator=cat_model,
    param_grid=model_params,
    scoring='roc_auc',
    cv=cv,
    n_jobs=-1
)

#Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

#Train and tune the model
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

catboost_best_params = best_model.get_params()
```

```
#Evaluate the model
probs = best_model.predict_proba(X_test)[:, 1]
preds = best_model.predict(X_test)
fpr, tpr, _ = roc_curve(y_test, probs)
roc_auc = auc(fpr, tpr)

print(f"CatBoost Best AUC: {roc_auc:.2f}")
print(classification_report(y_test, preds))
```

```
CatBoost Best AUC: 0.79
              precision    recall  f1-score   support

           0       0.90      0.55      0.68       970
           1       0.78      0.96      0.86      1612

    accuracy                           0.81      2582
   macro avg       0.84      0.76      0.77      2582
weighted avg       0.82      0.81      0.79      2582
```

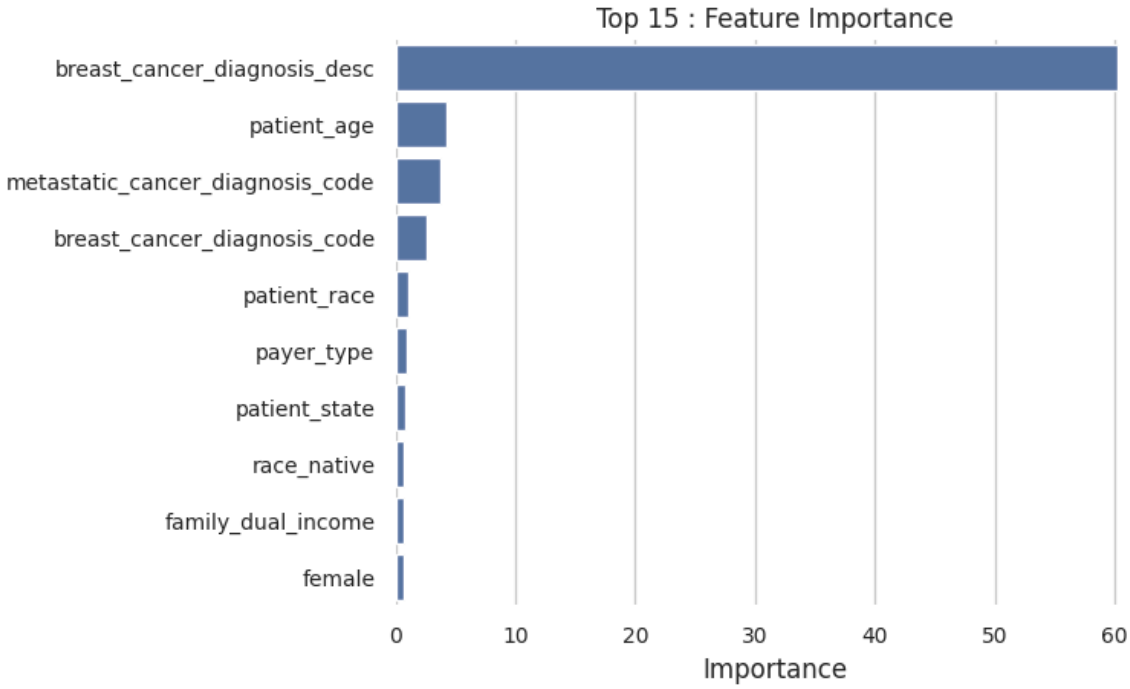
CatBoost Model Evaluation Without One-Hot Encoding

Given CatBoost's strong performance in previous tests, it was trained directly on categorical data without one-hot encoding. This uses CatBoost's built-in handling of such data, simplifying the process. This was also done to extract the feature importance in the original form of column names to understand what factors contribute to the target variable.

Results Summary

The model achieved an AUC of 0.79, showing strong discriminative ability. With 90% precision for the negative class and 96% recall for the positive class, it shows high accuracy in predicting true outcomes. Overall, the model correctly predicted outcomes 81% of the time, proving its effectiveness in using raw categorical data.

In [33]:
```
#Extract feature importances from the best CatBoost model
feature_importances = best_model.feature_importances_

#Create a DataFrame mapping each feature name to its importance
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

top10_sorted = importance_df.head(10)

sns.barplot(x='Importance', y='Feature', data=top10_sorted)
plt.title('Top 15 : Feature Importance')
plt.xlabel('Importance')
plt.ylabel('')
sns.despine(left=True, bottom=True)
plt.show()
```



Feature Importance for CatBoost Model without encoding the categorical variables

The CatBoost model's analysis of feature importance shows the most influential factors in predicting outcomes. The description of breast cancer diagnosis ('breast_cancer_diagnosis_desc') emerged as the most significant feature, followed closely by 'patient_age', indicating the crucial role of diagnosis specifics and age in the model's predictions. Other notable features include various diagnosis codes and patient demographics like 'patient_race' and 'payer_type'.

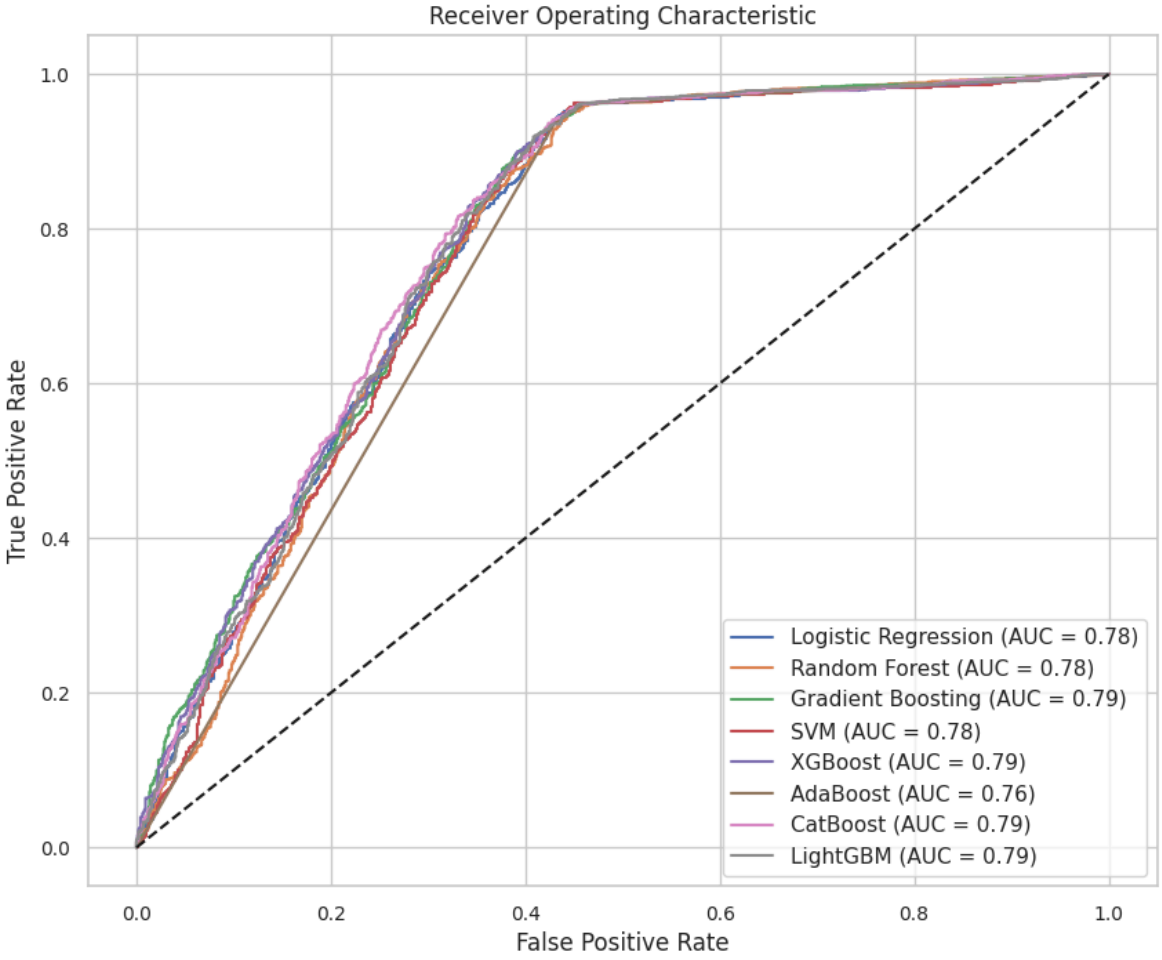
In [34]:
```
#Training and tuning all other models
roc_data = {}
best_models = []
skf = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

for name, info in models_params.items():
    print(f"Training and tuning {name}")
    grid_search = GridSearchCV(info['model'], info['params'], cv=skf, scoring='r
    grid_search.fit(X_processed, y)
    best_model = grid_search.best_estimator_
    best_models.append((name, best_model))

    #Evaluation
    X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_siz
    best_model.fit(X_train, y_train)
    probs = best_model.predict_proba(X_test)[:, 1]
    preds = best_model.predict(X_test)
    fpr, tpr, _ = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)
    roc_data[name] = (fpr, tpr, roc_auc)

    print(f"{name} AUC: {roc_auc:.2f}")
    print(classification_report(y_test, preds))
```

```
Training and tuning Logistic Regression
Logistic Regression AUC: 0.78
              precision    recall  f1-score   support

           0       0.88      0.55      0.68       970
           1       0.78      0.96      0.86      1612

    accuracy                           0.80      2582
   macro avg       0.83      0.75      0.77      2582
weighted avg       0.82      0.80      0.79      2582

Training and tuning Random Forest
Random Forest AUC: 0.78
              precision    recall  f1-score   support

           0       0.90      0.52      0.66       970
           1       0.77      0.96      0.86      1612

    accuracy                           0.80      2582
   macro avg       0.83      0.74      0.76      2582
weighted avg       0.82      0.80      0.78      2582

Training and tuning Gradient Boosting
Gradient Boosting AUC: 0.79
              precision    recall  f1-score   support

           0       0.89      0.54      0.67       970
           1       0.78      0.96      0.86      1612

    accuracy                           0.80      2582
   macro avg       0.83      0.75      0.77      2582
weighted avg       0.82      0.80      0.79      2582

Training and tuning SVM
SVM AUC: 0.78
              precision    recall  f1-score   support

           0       0.90      0.51      0.65       970
           1       0.76      0.97      0.85      1612

    accuracy                           0.79      2582
   macro avg       0.83      0.74      0.75      2582
weighted avg       0.81      0.79      0.78      2582

Training and tuning XGBoost
XGBoost AUC: 0.79
              precision    recall  f1-score   support

           0       0.89      0.55      0.68       970
           1       0.78      0.96      0.86      1612

    accuracy                           0.80      2582
   macro avg       0.83      0.75      0.77      2582
weighted avg       0.82      0.80      0.79      2582

Training and tuning AdaBoost
AdaBoost AUC: 0.76
              precision    recall  f1-score   support

           0       0.89      0.48      0.62       970
```

```
           1       0.75      0.97      0.85      1612

    accuracy                           0.78      2582
   macro avg       0.82      0.72      0.73      2582
weighted avg       0.81      0.78      0.76      2582

Training and tuning CatBoost
CatBoost AUC: 0.79
              precision    recall  f1-score   support

           0       0.88      0.55      0.68       970
           1       0.78      0.95      0.86      1612

    accuracy                           0.80      2582
   macro avg       0.83      0.75      0.77      2582
weighted avg       0.82      0.80      0.79      2582

Training and tuning LightGBM
[LightGBM] [Info] Number of positive: 8060, number of negative: 4846
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.013873 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 17444
[LightGBM] [Info] Number of data points in the train set: 12906, number of used f
eatures: 208
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.624516 -> initscore=0.508760
[LightGBM] [Info] Start training from score 0.508760
[LightGBM] [Info] Number of positive: 6448, number of negative: 3876
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.011045 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 17454
[LightGBM] [Info] Number of data points in the train set: 10324, number of used f
eatures: 205
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.624564 -> initscore=0.508966
[LightGBM] [Info] Start training from score 0.508966
LightGBM AUC: 0.79
              precision    recall  f1-score   support

           0       0.87      0.56      0.68       970
           1       0.78      0.95      0.86      1612

    accuracy                           0.80      2582
   macro avg       0.82      0.75      0.77      2582
weighted avg       0.81      0.80      0.79      2582
```

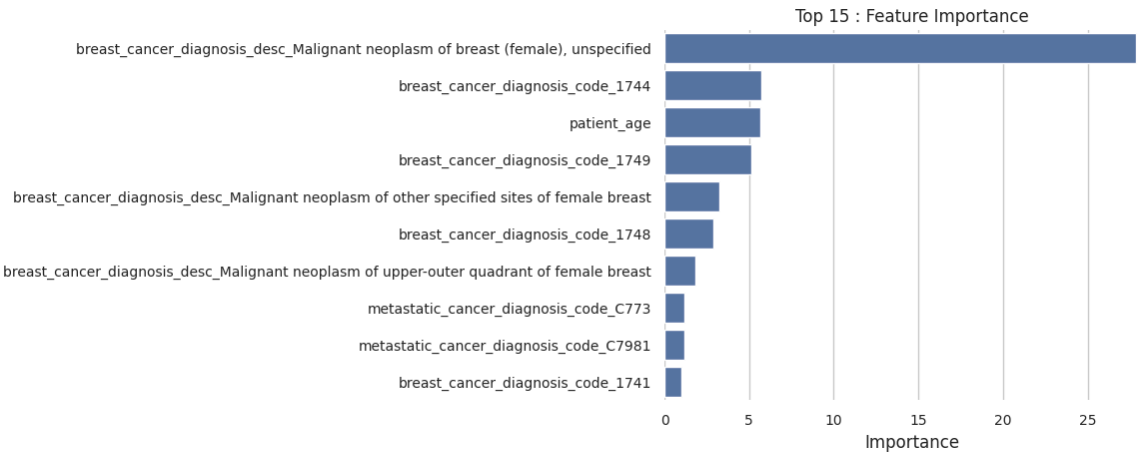
In [35]:

```python
#Plot ROC curves
plt.figure(figsize=(10, 8))
for name, (fpr, tpr, roc_auc) in roc_data.items():
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [36]:
```
#To find feature importance
numeric_features = numeric_columns
if categorical_columns:
    categorical_features = preprocessor.named_transformers_['cat'].get_feature_n
    all_features = np.concatenate([numeric_features, categorical_features])
else:
    all_features = np.array(numeric_features)

#Extract the best CatBoost model from the best_models list
best_catboost_model = next((model for name, model in best_models if name == 'Cat

if best_catboost_model is not None:
    #Get feature importances from the CatBoost model
    feature_importances = best_catboost_model.feature_importances_

    #Create a DataFrame to map features with their importance
    importance_df = pd.DataFrame({
        'Feature': all_features,
        'Importance': feature_importances
    }).sort_values(by='Importance', ascending=False)

    top10_sorted = importance_df.head(10)

    sns.barplot(x='Importance', y='Feature', data=top10_sorted)
    plt.title('Top 15 : Feature Importance')
    plt.xlabel('Importance')
    plt.ylabel('')
    sns.despine(left=True, bottom=True)
    plt.show()
```

```
else:
    print("CatBoost model not found in the best_models list.")
```



Model Training and Evaluation

1. Setup for Model Training and Tuning:

- **Models and Parameters:** A dictionary (`models_params`) defines each model (e.g., Logistic Regression, Random Forest) and its potential hyperparameters for tuning.
- **Stratified K-Fold:** This cross-validation method ensures each fold of data contains a proportionate representation of the target classes, crucial for maintaining balance especially in imbalanced datasets.

2. Hyperparameter Tuning with Cross-Validation:

- **Grid Search:** For each model, `GridSearchCV` is employed to find the best combination of parameters. It tests all possible combinations of the specified hyperparameters across different folds of the data.
- **Best Model Selection:** The best model configuration for each type is determined based on the Area Under the Receiver Operating Characteristic (ROC AUC) score, which measures the model's ability to distinguish between classes.

3. Model Evaluation on New Data:

- **Data Transformation:** The entire dataset is transformed using a predefined preprocessor to ensure the model inputs are correctly formatted.
- **Train-Test Split:** The data is then split into training and testing sets. This split allows the best-tuned model to be trained on one portion of the data and tested on a completely unseen portion.
- **Model Fitting and Prediction:** Each best-tuned model is trained on the training set and then used to predict outcomes on the testing set.
- **Performance Metrics:** The model's effectiveness is evaluated using the ROC AUC score and detailed classification metrics (precision, recall, f1-score) to understand its predictive accuracy and robustness.

Model Evaluation Summary

Model Comparison and Selection

During the assesment of various models, each was evaluated based on key metrics like ROC AUC score, precision, recall, and f1-scores. These metrics help us understand how well each model can correctly identify outcomes in a binary classification setting. Logistic Regression set a solid baseline with an AUC of 0.78. Random Forest and AdaBoost also performed great. SVM followed closely with an AUC of 0.78, benefiting from data normalization.

Gradient Boosting, XGBoost, CatBoost, and LightGBM all showed strong performance with an AUC of 0.79. These models are particularly good at managing the balance between detecting the presence or absence of a condition, which is crucial in medical diagnostics, for example. Among them, CatBoost stood out by consistently providing high precision and recall across various tests, proving its reliability in handling complex datasets.

Given this comprehensive evaluation, it was decided to proceed with **CatBoost** as the model of choice. Its ability to perform well consistently across various metrics and tests makes it the most suitable option for deployment in environments where accurate prediction is critical.

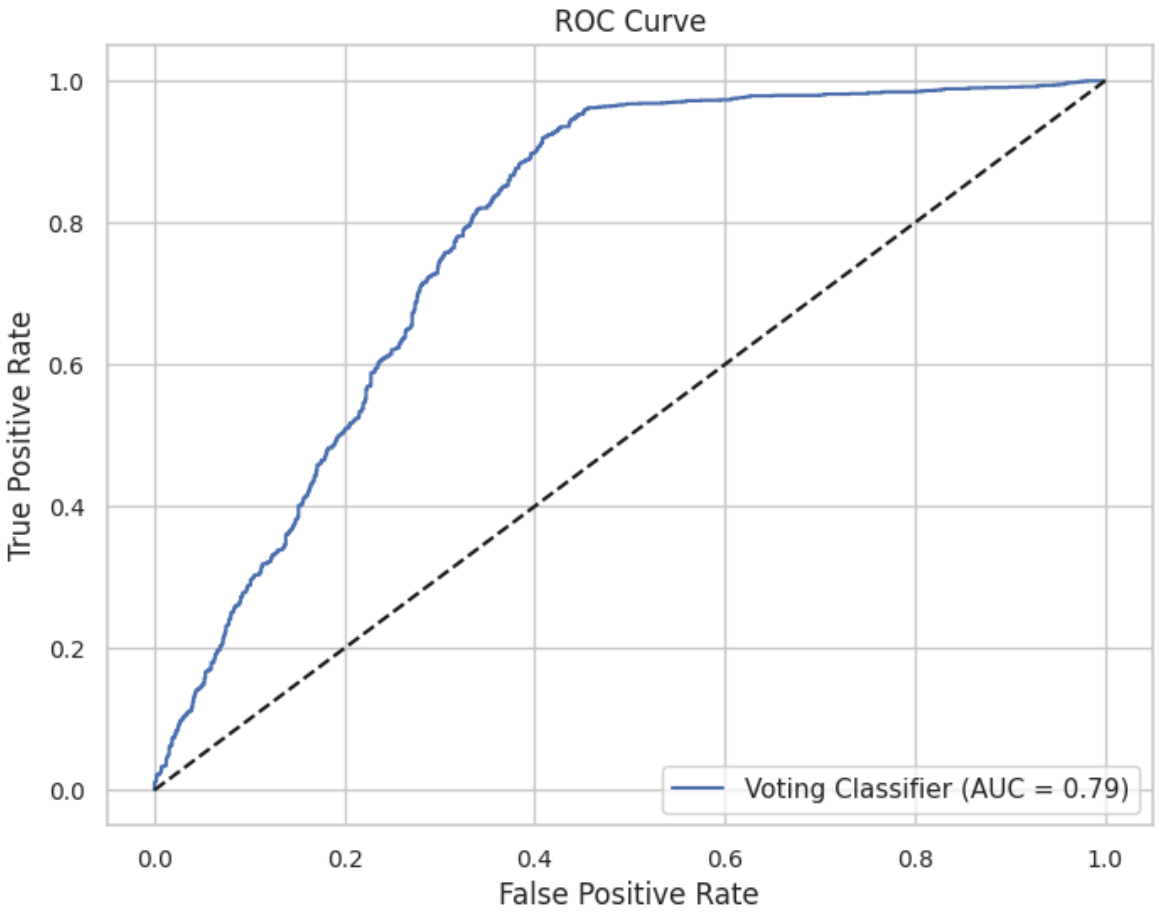
Feature Importance Summary

The CatBoost model, has highlighted the most influential features in predicting the target variable. The analysis shows that the description and codes related to breast cancer diagnosis are highly significant. The top feature, 'Malignant neoplasm of breast (female), unspecified', significantly impacts model predictions, showing that the general category of breast cancer plays a crucial role in outcomes. Other important features include specific breast cancer diagnosis codes like 1744 and 1749, which represent particular types of breast cancer. Additionally, 'patient_age' also features prominently, indicating that age is a vital factor in the predictions.

In [37]:
```
#Ensemble - Voting Classifier with best models
voting_clf = VotingClassifier(estimators=best_models, voting='soft')
voting_clf.fit(X_train, y_train)
voting_probs = voting_clf.predict_proba(X_test)[:, 1]
voting_preds = voting_clf.predict(X_test)
voting_fpr, voting_tpr, _ = roc_curve(y_test, voting_probs)
voting_roc_auc = auc(voting_fpr, voting_tpr)

print("Voting Classifier AUC:", voting_roc_auc)
print(classification_report(y_test, voting_preds))
```

```
[LightGBM] [Info] Number of positive: 6448, number of negative: 3876
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.010871 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 17454
[LightGBM] [Info] Number of data points in the train set: 10324, number of used f
eatures: 205
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.624564 -> initscore=0.508966
[LightGBM] [Info] Start training from score 0.508966
Voting Classifier AUC: 0.79339489908163
              precision    recall  f1-score   support

           0       0.90      0.55      0.68       970
           1       0.78      0.96      0.86      1612

    accuracy                           0.81      2582
   macro avg       0.84      0.75      0.77      2582
weighted avg       0.82      0.81      0.79      2582
```

In [38]:
```
#Plot ROC Curve for ensemble
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Voting Classifier (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



Ensemble Learning and Voting Classifier

Ensemble learning is a technique that involves combining multiple models to improve the overall performance, reduce overfitting, and increase the accuracy of predictions. It uses the strengths of each individual model and often achieves better performance than any single model could on its own.

A Voting Classifier is a type of ensemble model that gathers predictions from multiple models and outputs the prediction that receives the most votes. In the context of classification, it typically uses either:

- **Hard voting:** where the predicted output class is the class with the majority of votes from all the classifiers.
- **Soft voting:** where it predicts the class labels based on the average of probabilities given to that class by each model. Soft voting is often more flexible and achieves higher performance as it takes into account the certainty of each model's predictions.

Using an ensemble like a Voting Classifier can enhance the decision-making process by integrating diverse perspectives from multiple models, thus often leading to better and more stable predictive performance on complex datasets.

Ensemble Model Performance Summary

The **Voting Classifier** combines multiple models to improve prediction accuracy, achieving an AUC of 0.788. This shows good model performance in distinguishing between classes.

Results:

- **Precision**: High accuracy with 0.90 for non-disease predictions and 0.78 for disease predictions.
- **Recall**: Effectively identifies 55% of non-disease cases and 96% of disease cases.
- **F1-Score**: Balanced scores of 0.68 for non-disease and 0.86 for disease, indicating a strong model performance.
- **Overall Accuracy**: The model accurately predicts outcomes 81% of the time, showing its reliability in practical scenarios.