Final Assignment for the Supervised Machine Learning - Classification module under the IBM Machine Learning Professional Certificate program

Supervised Learning : Exploring Classification Models for Predicting Categorical Responses

---

Carmen Paz

---

1. Table of Contents

This project implements different predictive modelling procedures for dichotomous categorical responses using various helpful packages in **Python**. Models applied in the analysis to predict dichotomous categorical responses included the **Logistic Regression**, **Decision Trees**, **Random Forest** and **Support Vector Machine** algorithms. Remedial procedures on addressing class imbalance including **Class Weighting**, **Synthetic Minority Oversampling Technique** and **Condensed Nearest Neighbors** were similarly considered, as applicable. Ensemble learning using **Stacking** which consolidate many different models types on the same data and using another model to learn how to best combine the predictions was also explored. All results were consolidated in a **Summary** presented at the end of the document.

Binary classification learning refers to a predictive modelling problem where only two class labels are predicted for a given sample of input data. These models use the training data set and calculate how to best map instances of input data to the specific class labels. Typically, binary classification tasks involve one class that is the normal state (assigned the class label 0) and another class that is the abnormal state (assigned the class label 1). It is common to structure a binary classification task with a model that predicts a Bernoulli probability distribution for each instance. The Bernoulli distribution is a discrete probability distribution that covers a case where an event will have a binary outcome as either a 0 or 1. For a binary

classification, this means that the model predicts a probability of an instance belonging to class 1, or the abnormal state. The algorithms applied in this study attempt to categorize the input data and form dichotomous groups based on their similarities.

1.1. Data Background

Datasets used for the analysis were separately gathered and consolidated from various sources including:

1. Cancer Rates from World Population Review
2. Social Protection and Labor Indicator from World Bank
3. Education Indicator from World Bank
4. Economy and Growth Indicator from World Bank
5. Environment Indicator from World Bank
6. Climate Change Indicator from World Bank
7. Agricultural and Rural Development Indicator from World Bank view=chart
8. Social Development Indicator from World Bank
9. Health Indicator from World Bank
10. Science and Technology Indicator from World Bank
11. Urban Development Indicator from World Bank
12. Human Development Indices from Human Development Reports
13. Environmental Performance Indices from Yale Center for Environmental Law and Policy

This study hypothesized that various global development indicators and indices influence cancer rates across countries.

The target variable for the study is:

- CANRAT - Dichotomized category based on age-standardized cancer rates, per 100K population (2022)

The predictor variables for the study are:

- GDPPER - GDP per person employed, current US Dollars (2020)
- URBPOP - Urban population, % of total population (2020)
- PATRES - Patent applications by residents, total count (2020)
- RNDGDP - Research and development expenditure, % of GDP (2020)
- POPGRO - Population growth, annual % (2020)
- LIFEXP - Life expectancy at birth, total in years (2020)
- TUBINC - Incidence of tuberculosis, per 100K population (2020)
- DTHCMD - Cause of death by communicable diseases and maternal, prenatal and nutrition conditions, % of total (2019)
- AGRLND - Agricultural land, % of land area (2020)
- GHGEMI - Total greenhouse gas emissions, kt of CO2 equivalent (2020)

- RELOUT - Renewable electricity output, % of total electricity output (2015)
- METEMI - Methane emissions, kt of CO2 equivalent (2020)
- FORARE - Forest area, % of land area (2020)
- CO2EMI - CO2 emissions, metric tons per capita (2020)
- PM2EXP - PM2.5 air pollution, population exposed to levels exceeding WHO guideline value, % of total (2017)
- POPDEN - Population density, people per sq. km of land area (2020)
- GDPCAP - GDP per capita, current US Dollars (2020)
- ENRTER - Tertiary school enrollment, % gross (2020)
- HDICAT - Human development index, ordered category (2020)
- EPISCO - Environment performance index , score (2022)

1.2. Data Description

1. The dataset is comprised of:
    - **177 rows** (observations)
    - **22 columns** (variables)
        - **1/22 metadata** (object)
            - COUNTRY
        - **1/22 target** (categorical)
            - CANRAT
        - **19/22 predictor** (numeric)
            - GDPPER
            - URBPOP
            - PATRES
            - RNDGDP
            - POPGRO
            - LIFEXP
            - TUBINC
            - DTHCMD
            - AGRLND
            - GHGEMI
            - RELOUT
            - METEMI
            - FORARE
            - CO2EMI
            - PM2EXP
            - POPDEN
            - GDPCAP
            - ENRTER
            - EPISCO
        - **1/22 predictor** (categorical)

- HDICAT

In [1]:
```
###################################
# Setting up compatibility issues
# between the scikit-learn and imblearn packages
###################################
# !pip uninstall scikit-learn --yes
# !pip uninstall imblearn --yes
# !pip install scikit-learn==1.2.2
# !pip install imblearn
```

In [2]:
```
###################################
# Loading Python Libraries
###################################
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import itertools
%matplotlib inline

from operator import add,mul,truediv
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import StandardScaler
from scipy import stats

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split, GridSearchCV
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import CondensedNearestNeighbour
from sklearn.ensemble import StackingClassifier
```

In [3]:
```
###################################
# Loading the dataset
###################################
cancer_rate = pd.read_csv('CategoricalCancerRates.csv')
```

In [4]:
```
###################################
# Performing a general exploration of the dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate.shape)
```

```
Dataset Dimensions:
(177, 22)
```

In [5]:
```
###################################
# Listing the column names and data types
```

```
###################################
print('Column Names and Data Types:')
display(cancer_rate.dtypes)
```

```
Column Names and Data Types:
COUNTRY     object
CANRAT      object
GDPPER     float64
URBPOP     float64
PATRES     float64
RNDGDP     float64
POPGRO     float64
LIFEXP     float64
TUBINC     float64
DTHCMD     float64
AGRLND     float64
GHGEMI     float64
RELOUT     float64
METEMI     float64
FORARE     float64
CO2EMI     float64
PM2EXP     float64
POPDEN     float64
ENRTER     float64
GDPCAP     float64
HDICAT      object
EPISCO     float64
dtype: object
```

In [6]:
```
###################################
# Taking a snapshot of the dataset
###################################
cancer_rate.head()
```

Out[6]:

	COUNTRY	CANRAT	GDPPER	URBPOP	PATRES	RNDGDP	POPGRO	LIFEXP
0	Australia	High	98380.63601	86.241	2368.0	NaN	1.235701	83.200000
1	New Zealand	High	77541.76438	86.699	348.0	NaN	2.204789	82.256098
2	Ireland	High	198405.87500	63.653	75.0	1.23244	1.029111	82.556098
3	United States	High	130941.63690	82.664	269586.0	3.42287	0.964348	76.980488
4	Denmark	High	113300.60110	88.116	1261.0	2.96873	0.291641	81.602439

5 rows × 22 columns

In [7]:
```
###################################
# Setting the levels of the categorical variables
###################################
cancer_rate['CANRAT'] = cancer_rate['CANRAT'].astype('category')
cancer_rate['CANRAT'] = cancer_rate['CANRAT'].cat.set_categories(['Low', 'High'], o
```

```
cancer_rate['HDICAT'] = cancer_rate['HDICAT'].astype('category')
cancer_rate['HDICAT'] = cancer_rate['HDICAT'].cat.set_categories(['L', 'M', 'H', 'V
```

In [8]:
```
###################################
# Performing a general exploration of the numeric variables
###################################
print('Numeric Variable Summary:')
display(cancer_rate.describe(include='number').transpose())
```

Numeric Variable Summary:

	count	mean	std	min	25%	50%	
GDPPER	165.0	45284.424283	3.941794e+04	1718.804896	13545.254510	34024.900890	66
URBPOP	174.0	59.788121	2.280640e+01	13.345000	42.432750	61.701500	
PATRES	108.0	20607.388889	1.340683e+05	1.000000	35.250000	244.500000	1
RNDGDP	74.0	1.197474	1.189956e+00	0.039770	0.256372	0.873660	
POPGRO	174.0	1.127028	1.197718e+00	-2.079337	0.236900	1.179959	
LIFEXP	174.0	71.746113	7.606209e+00	52.777000	65.907500	72.464610	
TUBINC	174.0	105.005862	1.367229e+02	0.770000	12.000000	44.500000	
DTHCMD	170.0	21.260521	1.927333e+01	1.283611	6.078009	12.456279	
AGRLND	174.0	38.793456	2.171551e+01	0.512821	20.130276	40.386649	
GHGEMI	170.0	259582.709895	1.118550e+06	179.725150	12527.487367	41009.275980	116
RELOUT	153.0	39.760036	3.191492e+01	0.000296	10.582691	32.381668	
METEMI	170.0	47876.133575	1.346611e+05	11.596147	3662.884908	11118.976025	32
FORARE	173.0	32.218177	2.312001e+01	0.008078	11.604388	31.509048	
CO2EMI	170.0	3.751097	4.606479e+00	0.032585	0.631924	2.298368	
PM2EXP	167.0	91.940595	2.206003e+01	0.274092	99.627134	100.000000	
POPDEN	174.0	200.886765	6.453834e+02	2.115134	27.454539	77.983133	
ENRTER	116.0	49.994997	2.970619e+01	2.432581	22.107195	53.392460	
GDPCAP	170.0	13992.095610	1.957954e+04	216.827417	1870.503029	5348.192875	17
EPISCO	165.0	42.946667	1.249086e+01	18.900000	33.000000	40.900000	

In [9]:
```
###################################
# Performing a general exploration of the object variable
###################################
print('Object Variable Summary:')
display(cancer_rate.describe(include='object').transpose())
```

Object Variable Summary:

	count	unique	top	freq
COUNTRY	177	177	Australia	1

In [10]:
```
###################################
# Performing a general exploration of the categorical variables
###################################
print('Categorical Variable Summary:')
display(cancer_rate.describe(include='category').transpose())
```

Categorical Variable Summary:

	count	unique	top	freq
CANRAT	177	2	Low	132
HDICAT	167	4	VH	59

In [11]:
```
###################################
# Performing a general exploration of the response variable
###################################
cancer_rate.CANRAT.value_counts(normalize = True)
```

Out[11]:
```
CANRAT
Low     0.745763
High    0.254237
Name: proportion, dtype: float64
```

1.3. Data Quality Assessment

Data quality findings based on assessment are as follows:

1. No duplicated rows observed.
2. Missing data noted for 20 variables with Null.Count>0 and Fill.Rate<1.0.
   - RNDGDP: Null.Count = 103, Fill.Rate = 0.418
   - PATRES: Null.Count = 69, Fill.Rate = 0.610
   - ENRTER: Null.Count = 61, Fill.Rate = 0.655
   - RELOUT: Null.Count = 24, Fill.Rate = 0.864
   - GDPPER: Null.Count = 12, Fill.Rate = 0.932
   - EPISCO: Null.Count = 12, Fill.Rate = 0.932
   - HDICAT: Null.Count = 10, Fill.Rate = 0.943
   - PM2EXP: Null.Count = 10, Fill.Rate = 0.943
   - DTHCMD: Null.Count = 7, Fill.Rate = 0.960
   - METEMI: Null.Count = 7, Fill.Rate = 0.960
   - CO2EMI: Null.Count = 7, Fill.Rate = 0.960
   - GDPCAP: Null.Count = 7, Fill.Rate = 0.960
   - GHGEMI: Null.Count = 7, Fill.Rate = 0.960
   - FORARE: Null.Count = 4, Fill.Rate = 0.977

- TUBINC: Null.Count = 3, Fill.Rate = 0.983
- AGRLND: Null.Count = 3, Fill.Rate = 0.983
- POPGRO: Null.Count = 3, Fill.Rate = 0.983
- POPDEN: Null.Count = 3, Fill.Rate = 0.983
- URBPOP: Null.Count = 3, Fill.Rate = 0.983
- LIFEXP: Null.Count = 3, Fill.Rate = 0.983

3. 120 observations noted with at least 1 missing data. From this number, 14 observations reported high Missing.Rate>0.2.
   - COUNTRY=Guadeloupe: Missing.Rate= 0.909
   - COUNTRY=Martinique: Missing.Rate= 0.909
   - COUNTRY=French Guiana: Missing.Rate= 0.909
   - COUNTRY=New Caledonia: Missing.Rate= 0.500
   - COUNTRY=French Polynesia: Missing.Rate= 0.500
   - COUNTRY=Guam: Missing.Rate= 0.500
   - COUNTRY=Puerto Rico: Missing.Rate= 0.409
   - COUNTRY=North Korea: Missing.Rate= 0.227
   - COUNTRY=Somalia: Missing.Rate= 0.227
   - COUNTRY=South Sudan: Missing.Rate= 0.227
   - COUNTRY=Venezuela: Missing.Rate= 0.227
   - COUNTRY=Libya: Missing.Rate= 0.227
   - COUNTRY=Eritrea: Missing.Rate= 0.227
   - COUNTRY=Yemen: Missing.Rate= 0.227
4. Low variance observed for 1 variable with First.Second.Mode.Ratio>5.
   - PM2EXP: First.Second.Mode.Ratio = 53.000
5. No low variance observed for any variable with Unique.Count.Ratio>10.
6. High skewness observed for 5 variables with Skewness>3 or Skewness<(-3).
   - POPDEN: Skewness = +10.267
   - GHGEMI: Skewness = +9.496
   - PATRES: Skewness = +9.284
   - METEMI: Skewness = +5.801
   - PM2EXP: Skewness = -3.141

In [12]:
```
###################################
# Counting the number of duplicated rows
###################################
cancer_rate.duplicated().sum()
```

Out[12]:
```
np.int64(0)
```

In [13]:
```
###################################
# Gathering the data types for each column
###################################
data_type_list = list(cancer_rate.dtypes)
```

In [14]:
```
###################################
# Gathering the variable names for each column
###################################
variable_name_list = list(cancer_rate.columns)
```

In [15]:
```
###################################
# Gathering the number of observations for each column
###################################
row_count_list = list([len(cancer_rate)] * len(cancer_rate.columns))
```

In [16]:
```
###################################
# Gathering the number of missing data for each column
###################################
null_count_list = list(cancer_rate.isna().sum(axis=0))
```

In [17]:
```
###################################
# Gathering the number of non-missing data for each column
###################################
non_null_count_list = list(cancer_rate.count())
```

In [18]:
```
###################################
# Gathering the missing data percentage for each column
###################################
fill_rate_list = map(truediv, non_null_count_list, row_count_list)
```

In [19]:
```
###################################
# Formulating the summary
# for all columns
###################################
all_column_quality_summary = pd.DataFrame(zip(variable_name_list,
                                              data_type_list,
                                              row_count_list,
                                              non_null_count_list,
                                              null_count_list,
                                              fill_rate_list),
                                          columns=['Column.Name',
                                                   'Column.Type',
                                                   'Row.Count',
                                                   'Non.Null.Count',
                                                   'Null.Count',
                                                   'Fill.Rate'])
display(all_column_quality_summary)
```

	Column.Name	Column.Type	Row.Count	Non.Null.Count	Null.Count	Fill.Rate
0	COUNTRY	object	177	177	0	1.000000
1	CANRAT	category	177	177	0	1.000000
2	GDPPER	float64	177	165	12	0.932203
3	URBPOP	float64	177	174	3	0.983051
4	PATRES	float64	177	108	69	0.610169
5	RNDGDP	float64	177	74	103	0.418079
6	POPGRO	float64	177	174	3	0.983051
7	LIFEXP	float64	177	174	3	0.983051
8	TUBINC	float64	177	174	3	0.983051
9	DTHCMD	float64	177	170	7	0.960452
10	AGRLND	float64	177	174	3	0.983051
11	GHGEMI	float64	177	170	7	0.960452
12	RELOUT	float64	177	153	24	0.864407
13	METEMI	float64	177	170	7	0.960452
14	FORARE	float64	177	173	4	0.977401
15	CO2EMI	float64	177	170	7	0.960452
16	PM2EXP	float64	177	167	10	0.943503
17	POPDEN	float64	177	174	3	0.983051
18	ENRTER	float64	177	116	61	0.655367
19	GDPCAP	float64	177	170	7	0.960452
20	HDICAT	category	177	167	10	0.943503
21	EPISCO	float64	177	165	12	0.932203

In [20]:
```
###################################
# Counting the number of columns
# with Fill.Rate < 1.00
###################################
len(all_column_quality_summary[(all_column_quality_summary['Fill.Rate']<1)])
```

Out[20]: 20

In [21]:
```
###################################
# Identifying the columns
# with Fill.Rate < 1.00
###################################
display(all_column_quality_summary[(all_column_quality_summary['Fill.Rate']<1)].sor
```

	Column.Name	Column.Type	Row.Count	Non.Null.Count	Null.Count	Fill.Rate
5	RNDGDP	float64	177	74	103	0.418079
4	PATRES	float64	177	108	69	0.610169
18	ENRTER	float64	177	116	61	0.655367
12	RELOUT	float64	177	153	24	0.864407
21	EPISCO	float64	177	165	12	0.932203
2	GDPPER	float64	177	165	12	0.932203
16	PM2EXP	float64	177	167	10	0.943503
20	HDICAT	category	177	167	10	0.943503
15	CO2EMI	float64	177	170	7	0.960452
13	METEMI	float64	177	170	7	0.960452
11	GHGEMI	float64	177	170	7	0.960452
9	DTHCMD	float64	177	170	7	0.960452
19	GDPCAP	float64	177	170	7	0.960452
14	FORARE	float64	177	173	4	0.977401
6	POPGRO	float64	177	174	3	0.983051
3	URBPOP	float64	177	174	3	0.983051
17	POPDEN	float64	177	174	3	0.983051
10	AGRLND	float64	177	174	3	0.983051
7	LIFEXP	float64	177	174	3	0.983051
8	TUBINC	float64	177	174	3	0.983051

In [22]:
```
###################################
# Identifying the rows
# with Fill.Rate < 0.90
###################################
column_low_fill_rate = all_column_quality_summary[(all_column_quality_summary['Fill
```

In [23]:
```
###################################
# Gathering the metadata labels for each observation
###################################
row_metadata_list = cancer_rate["COUNTRY"].values.tolist()
```

In [24]:
```
###################################
# Gathering the number of columns for each observation
###################################
column_count_list = list([len(cancer_rate.columns)] * len(cancer_rate))
```

In [25]:
```
###################################
# Gathering the number of missing data for each row
###################################
null_row_list = list(cancer_rate.isna().sum(axis=1))
```

In [26]:
```
###################################
# Gathering the missing data percentage for each column
###################################
missing_rate_list = map(truediv, null_row_list, column_count_list)
```

In [27]:
```
###################################
# Identifying the rows
# with missing data
###################################
all_row_quality_summary = pd.DataFrame(zip(row_metadata_list,
                                           column_count_list,
                                           null_row_list,
                                           missing_rate_list),
                                       columns=['Row.Name',
                                                'Column.Count',
                                                'Null.Count',
                                                'Missing.Rate'])
display(all_row_quality_summary)
```

	Row.Name	Column.Count	Null.Count	Missing.Rate
0	Australia	22	1	0.045455
1	New Zealand	22	2	0.090909
2	Ireland	22	0	0.000000
3	United States	22	0	0.000000
4	Denmark	22	0	0.000000
...	...	...	...	...
172	Congo Republic	22	3	0.136364
173	Bhutan	22	2	0.090909
174	Nepal	22	2	0.090909
175	Gambia	22	4	0.181818
176	Niger	22	2	0.090909

177 rows × 4 columns

In [28]:
```
###################################
# Counting the number of rows
# with Missing.Rate > 0.00
###################################
len(all_row_quality_summary[(all_row_quality_summary['Missing.Rate']>0.00)])
```

Out[28]: 120

In [29]:
```
###################################
# Counting the number of rows
# with Missing.Rate > 0.20
###################################
len(all_row_quality_summary[(all_row_quality_summary['Missing.Rate']>0.20)])
```

Out[29]: 14

In [30]:
```
###################################
# Identifying the rows
# with Missing.Rate > 0.20
###################################
row_high_missing_rate = all_row_quality_summary[(all_row_quality_summary['Missing.R
```

In [31]:
```
###################################
# Identifying the rows
# with Missing.Rate > 0.20
###################################
display(all_row_quality_summary[(all_row_quality_summary['Missing.Rate']>0.20)].sor
```

	Row.Name	Column.Count	Null.Count	Missing.Rate
35	Guadeloupe	22	20	0.909091
39	Martinique	22	20	0.909091
56	French Guiana	22	20	0.909091
13	New Caledonia	22	11	0.500000
44	French Polynesia	22	11	0.500000
75	Guam	22	11	0.500000
53	Puerto Rico	22	9	0.409091
85	North Korea	22	6	0.272727
168	South Sudan	22	6	0.272727
132	Somalia	22	6	0.272727
117	Libya	22	5	0.227273
73	Venezuela	22	5	0.227273
161	Eritrea	22	5	0.227273
164	Yemen	22	5	0.227273

In [32]:
```
###################################
# Formulating the dataset
# with numeric columns only
###################################
cancer_rate_numeric = cancer_rate.select_dtypes(include='number')
```

In [33]:
```
###################################
# Gathering the variable names for each numeric column
###################################
numeric_variable_name_list = cancer_rate_numeric.columns
```

In [34]:
```
###################################
# Gathering the minimum value for each numeric column
###################################
numeric_minimum_list = cancer_rate_numeric.min()
```

In [35]:
```
###################################
# Gathering the mean value for each numeric column
###################################
numeric_mean_list = cancer_rate_numeric.mean()
```

In [36]:
```
###################################
# Gathering the median value for each numeric column
###################################
numeric_median_list = cancer_rate_numeric.median()
```

In [37]:
```
###################################
# Gathering the maximum value for each numeric column
###################################
numeric_maximum_list = cancer_rate_numeric.max()
```

In [38]:
```
###################################
# Gathering the first mode values for each numeric column
###################################
numeric_first_mode_list = [cancer_rate[x].value_counts(dropna=True).index.tolist()[
```

In [39]:
```
###################################
# Gathering the second mode values for each numeric column
###################################
numeric_second_mode_list = [cancer_rate[x].value_counts(dropna=True).index.tolist()
```

In [40]:
```
###################################
# Gathering the count of first mode values for each numeric column
###################################
numeric_first_mode_count_list = [cancer_rate_numeric[x].isin([cancer_rate[x].value_
```

In [41]:
```
###################################
# Gathering the count of second mode values for each numeric column
###################################
numeric_second_mode_count_list = [cancer_rate_numeric[x].isin([cancer_rate[x].value
```

In [42]:
```
###################################
# Gathering the first mode to second mode ratio for each numeric column
###################################
numeric_first_second_mode_ratio_list = map(truediv, numeric_first_mode_count_list,
```

In [43]:
```
###################################
# Gathering the count of unique values for each numeric column
###################################
numeric_unique_count_list = cancer_rate_numeric.nunique(dropna=True)
```

In [44]:
```
###################################
# Gathering the number of observations for each numeric column
###################################
numeric_row_count_list = list([len(cancer_rate_numeric)] * len(cancer_rate_numeric.
```

In [45]:
```
###################################
# Gathering the unique to count ratio for each numeric column
###################################
numeric_unique_count_ratio_list = map(truediv, numeric_unique_count_list, numeric_r
```

In [46]:
```
###################################
# Gathering the skewness value for each numeric column
###################################
numeric_skewness_list = cancer_rate_numeric.skew()
```

In [47]:
```
###################################
# Gathering the kurtosis value for each numeric column
###################################
numeric_kurtosis_list = cancer_rate_numeric.kurtosis()
```

In [48]:
```
numeric_column_quality_summary = pd.DataFrame(zip(numeric_variable_name_list,
                                                  numeric_minimum_list,
                                                  numeric_mean_list,
                                                  numeric_median_list,
                                                  numeric_maximum_list,
                                                  numeric_first_mode_list,
                                                  numeric_second_mode_list,
                                                  numeric_first_mode_count_list,
                                                  numeric_second_mode_count_list,
                                                  numeric_first_second_mode_ratio_lis
                                                  numeric_unique_count_list,
                                                  numeric_row_count_list,
                                                  numeric_unique_count_ratio_list,
                                                  numeric_skewness_list,
                                                  numeric_kurtosis_list),
                                              columns=['Numeric.Column.Name',
                                                       'Minimum',
                                                       'Mean',
                                                       'Median',
                                                       'Maximum',
                                                       'First.Mode',
                                                       'Second.Mode',
                                                       'First.Mode.Count',
                                                       'Second.Mode.Count',
```

```
                                                'First.Second.Mode.Ratio',
                                                'Unique.Count',
                                                'Row.Count',
                                                'Unique.Count.Ratio',
                                                'Skewness',
                                                'Kurtosis'])
display(numeric_column_quality_summary)
```

	Numeric.Column.Name	Minimum	Mean	Median	Maximum	First
0	GDPPER	1718.804896	45284.424283	34024.900890	2.346469e+05	98380.
1	URBPOP	13.345000	59.788121	61.701500	1.000000e+02	100.
2	PATRES	1.000000	20607.388889	244.500000	1.344817e+06	6.
3	RNDGDP	0.039770	1.197474	0.873660	5.354510e+00	1.
4	POPGRO	-2.079337	1.127028	1.179959	3.727101e+00	1.
5	LIFEXP	52.777000	71.746113	72.464610	8.456000e+01	83.
6	TUBINC	0.770000	105.005862	44.500000	5.920000e+02	12.
7	DTHCMD	1.283611	21.260521	12.456279	6.520789e+01	4.
8	AGRLND	0.512821	38.793456	40.386649	8.084112e+01	46.
9	GHGEMI	179.725150	259582.709895	41009.275980	1.294287e+07	571903.
10	RELOUT	0.000296	39.760036	32.381668	1.000000e+02	100.
11	METEMI	11.596147	47876.133575	11118.976025	1.186285e+06	131484.
12	FORARE	0.008078	32.218177	31.509048	9.741212e+01	17.
13	CO2EMI	0.032585	3.751097	2.298368	3.172684e+01	14.
14	PM2EXP	0.274092	91.940595	100.000000	1.000000e+02	100.
15	POPDEN	2.115134	200.886765	77.983133	7.918951e+03	3.
16	ENRTER	2.432581	49.994997	53.392460	1.433107e+02	110.
17	GDPCAP	216.827417	13992.095610	5348.192875	1.173705e+05	51722.
18	EPISCO	18.900000	42.946667	40.900000	7.790000e+01	29.

In [49]:
```
###################################
# Counting the number of numeric columns
# with First.Second.Mode.Ratio > 5.00
###################################
len(numeric_column_quality_summary[(numeric_column_quality_summary['First.Second.Mo
```

Out[49]: 1

In [50]:
```
###################################
# Identifying the numeric columns
```

```
# with First.Second.Mode.Ratio > 5.00
###################################
display(numeric_column_quality_summary[(numeric_column_quality_summary['First.Secon
```

	Numeric.Column.Name	Minimum	Mean	Median	Maximum	First.Mode	Second.M
14	PM2EXP	0.274092	91.940595	100.0	100.0	100.0	1(

In [51]:
```
###################################
# Counting the number of numeric columns
# with Unique.Count.Ratio > 10.00
###################################
len(numeric_column_quality_summary[(numeric_column_quality_summary['Unique.Count.Ra
```

Out[51]: 0

In [52]:
```
###################################
# Counting the number of numeric columns
# with Skewness > 3.00 or Skewness < -3.00
###################################
len(numeric_column_quality_summary[(numeric_column_quality_summary['Skewness']>3) |
```

Out[52]: 5

In [53]:
```
###################################
# Identifying the numeric columns
# with Skewness > 3.00 or Skewness < -3.00
###################################
display(numeric_column_quality_summary[(numeric_column_quality_summary['Skewness']>
```

	Numeric.Column.Name	Minimum	Mean	Median	Maximum	First.I
15	POPDEN	2.115134	200.886765	77.983133	7.918951e+03	3.3
9	GHGEMI	179.725150	259582.709895	41009.275980	1.294287e+07	571903.1
2	PATRES	1.000000	20607.388889	244.500000	1.344817e+06	6.0(
11	METEMI	11.596147	47876.133575	11118.976025	1.186285e+06	131484.7(
14	PM2EXP	0.274092	91.940595	100.000000	1.000000e+02	100.0(

In [54]:
```
###################################
# Formulating the dataset
# with object column only
###################################
cancer_rate_object = cancer_rate.select_dtypes(include='object')
```

In [55]:
```
###################################
# Gathering the variable names for the object column
###################################
object_variable_name_list = cancer_rate_object.columns
```

In [56]:
```
###################################
# Gathering the first mode values for the object column
###################################
object_first_mode_list = [cancer_rate[x].value_counts().index.tolist()[0] for x in
```

In [57]:
```
###################################
# Gathering the second mode values for each object column
###################################
object_second_mode_list = [cancer_rate[x].value_counts().index.tolist()[1] for x in
```

In [58]:
```
###################################
# Gathering the count of first mode values for each object column
###################################
object_first_mode_count_list = [cancer_rate_object[x].isin([cancer_rate[x].value_co
```

In [59]:
```
###################################
# Gathering the count of second mode values for each object column
###################################
object_second_mode_count_list = [cancer_rate_object[x].isin([cancer_rate[x].value_c
```

In [60]:
```
###################################
# Gathering the first mode to second mode ratio for each object column
###################################
object_first_second_mode_ratio_list = map(truediv, object_first_mode_count_list, ob
```

In [61]:
```
###################################
# Gathering the count of unique values for each object column
###################################
object_unique_count_list = cancer_rate_object.nunique(dropna=True)
```

In [62]:
```
###################################
# Gathering the number of observations for each object column
###################################
object_row_count_list = list([len(cancer_rate_object)] * len(cancer_rate_object.col
```

In [63]:
```
###################################
# Gathering the unique to count ratio for each object column
###################################
object_unique_count_ratio_list = map(truediv, object_unique_count_list, object_row_
```

In [64]:
```
object_column_quality_summary = pd.DataFrame(zip(object_variable_name_list,
                                                 object_first_mode_list,
                                                 object_second_mode_list,
                                                 object_first_mode_count_list,
                                                 object_second_mode_count_list,
                                                 object_first_second_mode_ratio_lis
                                                 object_unique_count_list,
                                                 object_row_count_list,
                                                 object_unique_count_ratio_list),
                                            columns=['Object.Column.Name',
                                                     'First.Mode',
                                                     'Second.Mode',
                                                     'First.Mode.Count',
```

```
                                                'Second.Mode.Count',
                                                'First.Second.Mode.Ratio',
                                                'Unique.Count',
                                                'Row.Count',
                                                'Unique.Count.Ratio'])
display(object_column_quality_summary)
```

	Object.Column.Name	First.Mode	Second.Mode	First.Mode.Count	Second.Mode.Count	Fi
0	COUNTRY	Australia	New Zealand	1	1	

In [65]:
```
###################################
# Counting the number of object columns
# with First.Second.Mode.Ratio > 5.00
###################################
len(object_column_quality_summary[(object_column_quality_summary['First.Second.Mode
```

Out[65]: 0

In [66]:
```
###################################
# Counting the number of object columns
# with Unique.Count.Ratio > 10.00
###################################
len(object_column_quality_summary[(object_column_quality_summary['Unique.Count.Rati
```

Out[66]: 0

In [67]:
```
###################################
# Formulating the dataset
# with categorical columns only
###################################
cancer_rate_categorical = cancer_rate.select_dtypes(include='category')
```

In [68]:
```
###################################
# Gathering the variable names for the categorical column
###################################
categorical_variable_name_list = cancer_rate_categorical.columns
```

In [69]:
```
###################################
# Gathering the first mode values for each categorical column
###################################
categorical_first_mode_list = [cancer_rate[x].value_counts().index.tolist()[0] for
```

In [70]:
```
###################################
# Gathering the second mode values for each categorical column
###################################
categorical_second_mode_list = [cancer_rate[x].value_counts().index.tolist()[1] for
```

In [71]:
```
###################################
# Gathering the count of first mode values for each categorical column
###################################
categorical_first_mode_count_list = [cancer_rate_categorical[x].isin([cancer_rate[x
```

In [72]:
```
###################################
# Gathering the count of second mode values for each categorical column
###################################
categorical_second_mode_count_list = [cancer_rate_categorical[x].isin([cancer_rate[
```

In [73]:
```
###################################
# Gathering the first mode to second mode ratio for each categorical column
###################################
categorical_first_second_mode_ratio_list = map(truediv, categorical_first_mode_coun
```

In [74]:
```
###################################
# Gathering the count of unique values for each categorical column
###################################
categorical_unique_count_list = cancer_rate_categorical.nunique(dropna=True)
```

In [75]:
```
###################################
# Gathering the number of observations for each categorical column
###################################
categorical_row_count_list = list([len(cancer_rate_categorical)] * len(cancer_rate_
```

In [76]:
```
###################################
# Gathering the unique to count ratio for each categorical column
###################################
categorical_unique_count_ratio_list = map(truediv, categorical_unique_count_list, c
```

In [77]:
```
categorical_column_quality_summary = pd.DataFrame(zip(categorical_variable_name_lis
                                                      categorical_first_mode_list,
                                                      categorical_second_mode_list,
                                                      categorical_first_mode_count_li
                                                      categorical_second_mode_count_l
                                                      categorical_first_second_mode_r
                                                      categorical_unique_count_list,
                                                      categorical_row_count_list,
                                                      categorical_unique_count_ratio_
                                                columns=['Categorical.Column.Name',
                                                         'First.Mode',
                                                         'Second.Mode',
                                                         'First.Mode.Count',
                                                         'Second.Mode.Count',
                                                         'First.Second.Mode.Ratio',
                                                         'Unique.Count',
                                                         'Row.Count',
                                                         'Unique.Count.Ratio'])
display(categorical_column_quality_summary)
```

	Categorical.Column.Name	First.Mode	Second.Mode	First.Mode.Count	Second.Mode.Coun
0	CANRAT	Low	High	132	4
1	HDICAT	VH	H	59	3

In [78]:
```
###################################
# Counting the number of categorical columns
# with First.Second.Mode.Ratio > 5.00
###################################
len(categorical_column_quality_summary[(categorical_column_quality_summary['First.S
```

Out[78]: 0

In [79]:
```
###################################
# Counting the number of categorical columns
# with Unique.Count.Ratio > 10.00
###################################
len(categorical_column_quality_summary[(categorical_column_quality_summary['Unique.
```

Out[79]: 0

1.4. Data Preprocessing

1.4.1 Data Cleaning

1. Subsets of rows and columns with high rates of missing data were removed from the dataset:
   - 4 variables with Fill.Rate<0.9 were excluded for subsequent analysis.
     - RNDGDP: Null.Count = 103, Fill.Rate = 0.418
     - PATRES: Null.Count = 69, Fill.Rate = 0.610
     - ENRTER: Null.Count = 61, Fill.Rate = 0.655
     - RELOUT: Null.Count = 24, Fill.Rate = 0.864
   - 14 rows with Missing.Rate>0.2 were exluded for subsequent analysis.
     - COUNTRY=Guadeloupe: Missing.Rate= 0.909
     - COUNTRY=Martinique: Missing.Rate= 0.909
     - COUNTRY=French Guiana: Missing.Rate= 0.909
     - COUNTRY=New Caledonia: Missing.Rate= 0.500
     - COUNTRY=French Polynesia: Missing.Rate= 0.500
     - COUNTRY=Guam: Missing.Rate= 0.500
     - COUNTRY=Puerto Rico: Missing.Rate= 0.409
     - COUNTRY=North Korea: Missing.Rate= 0.227
     - COUNTRY=Somalia: Missing.Rate= 0.227
     - COUNTRY=South Sudan: Missing.Rate= 0.227
     - COUNTRY=Venezuela: Missing.Rate= 0.227
     - COUNTRY=Libya: Missing.Rate= 0.227
     - COUNTRY=Eritrea: Missing.Rate= 0.227
     - COUNTRY=Yemen: Missing.Rate= 0.227
2. No variables were removed due to zero or near-zero variance.
3. The cleaned dataset is comprised of:
   - **163 rows** (observations)

- **18 columns** (variables)
  - **1/18 metadata** (object)
    - COUNTRY
  - **1/18 target** (categorical)
    - CANRAT
  - **15/18 predictor** (numeric)
    - GDPPER
    - URBPOP
    - POPGRO
    - LIFEXP
    - TUBINC
    - DTHCMD
    - AGRLND
    - GHGEMI
    - METEMI
    - FORARE
    - CO2EMI
    - PM2EXP
    - POPDEN
    - GDPCAP
    - EPISCO
  - **1/18 predictor** (categorical)
    - HDICAT

In [80]:
```
###################################
# Performing a general exploration of the original dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate.shape)
```

```
Dataset Dimensions:
(177, 22)
```

In [81]:
```
###################################
# Filtering out the rows with
# with Missing.Rate > 0.20
###################################
cancer_rate_filtered_row = cancer_rate.drop(cancer_rate[cancer_rate.COUNTRY.isin(ro
```

In [82]:
```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_filtered_row.shape)
```

```
Dataset Dimensions:
(163, 22)
```

In [83]:
```
###################################
# Filtering out the columns with
# with Fill.Rate < 0.90
###################################
cancer_rate_filtered_row_column = cancer_rate_filtered_row.drop(column_low_fill_rat
```

In [84]:
```
###################################
# Formulating a new dataset object
# for the cleaned data
###################################
cancer_rate_cleaned = cancer_rate_filtered_row_column
```

In [85]:
```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_cleaned.shape)
```

```
Dataset Dimensions:
(163, 18)
```

1.4.2 Missing Data Imputation

Iterative Imputeris based on the Multivariate Imputation by Chained Equations (MICE) algorithm - an imputation method based on fully conditional specification, where each incomplete variable is imputed by a separate model. As a sequential regression imputation technique, the algorithm imputes an incomplete column (target column) by generating plausible synthetic values given other columns in the data. Each incomplete column must act as a target column, and has its own specific set of predictors. For predictors that are incomplete themselves, the most recently generated imputations are used to complete the predictors prior to prior to imputation of the target columns.

Linear Regression explores the linear relationship between a scalar response and one or more covariates by having the conditional mean of the dependent variable be an affine function of the independent variables. The relationship is modeled through a disturbance term which represents an unobserved random variable that adds noise. The algorithm is typically formulated from the data using the least squares method which seeks to estimate the coefficients by minimizing the squared residual function. The linear equation assigns one scale factor represented by a coefficient to each covariate and an additional coefficient called the intercept or the bias coefficient which gives the line an additional degree of freedom allowing to move up and down a two-dimensional plot.

1. Missing data for numeric variables were imputed using the iterative imputer algorithm with a linear regression estimator. * GDPPER: Null.Count = 1 * FORARE: Null.Count = 1 * PM2EXP: Null.Count = 5
2. Missing data for categorical variables were imputed using the most frequent value. * HDICAP: Null.Count = 1

In [86]:
```
###################################
# Formulating the summary
# for all cleaned columns
###################################
cleaned_column_quality_summary = pd.DataFrame(zip(list(cancer_rate_cleaned.columns)
                                                  list(cancer_rate_cleaned.dtypes),
                                                  list([len(cancer_rate_cleaned)] *
                                                  list(cancer_rate_cleaned.count())
                                                  list(cancer_rate_cleaned.isna().s
                                              columns=['Column.Name',
                                                       'Column.Type',
                                                       'Row.Count',
                                                       'Non.Null.Count',
                                                       'Null.Count'])
display(cleaned_column_quality_summary)
```

	Column.Name	Column.Type	Row.Count	Non.Null.Count	Null.Count
0	COUNTRY	object	163	163	0
1	CANRAT	category	163	163	0
2	GDPPER	float64	163	162	1
3	URBPOP	float64	163	163	0
4	POPGRO	float64	163	163	0
5	LIFEXP	float64	163	163	0
6	TUBINC	float64	163	163	0
7	DTHCMD	float64	163	163	0
8	AGRLND	float64	163	163	0
9	GHGEMI	float64	163	163	0
10	METEMI	float64	163	163	0
11	FORARE	float64	163	162	1
12	CO2EMI	float64	163	163	0
13	PM2EXP	float64	163	158	5
14	POPDEN	float64	163	163	0
15	GDPCAP	float64	163	163	0
16	HDICAT	category	163	162	1
17	EPISCO	float64	163	163	0

In [87]:
```
###################################
# Formulating the cleaned dataset
# with categorical columns only
```

```
###################################
cancer_rate_cleaned_categorical = cancer_rate_cleaned.select_dtypes(include='object
```

In [88]:
```
###################################
# Formulating the cleaned dataset
# with numeric columns only
###################################
cancer_rate_cleaned_numeric = cancer_rate_cleaned.select_dtypes(include='number')
```

In [89]:
```
###################################
# Taking a snapshot of the cleaned dataset
###################################
cancer_rate_cleaned_numeric.head()
```

Out[89]:

	GDPPER	URBPOP	POPGRO	LIFEXP	TUBINC	DTHCMD	AGRLND	GHGEM
0	98380.63601	86.241	1.235701	83.200000	7.2	4.941054	46.252480	5.719031e+0
1	77541.76438	86.699	2.204789	82.256098	7.2	4.354730	38.562911	8.015803e+0
2	198405.87500	63.653	1.029111	82.556098	5.3	5.684596	65.495718	5.949773e+0
3	130941.63690	82.664	0.964348	76.980488	2.3	5.302060	44.363367	5.505181e+0
4	113300.60110	88.116	0.291641	81.602439	4.1	6.826140	65.499675	4.113555e+0

In [90]:
```
###################################
# Defining the estimator to be used
# at each step of the round-robin imputation
###################################
lr = LinearRegression()
```

In [91]:
```
###################################
# Defining the parameter of the
# iterative imputer which will estimate
# the columns with missing values
# as a function of the other columns
# in a round-robin fashion
###################################
iterative_imputer = IterativeImputer(
    estimator = lr,
    max_iter = 10,
    tol = 1e-10,
    imputation_order = 'ascending',
    random_state=88888888
)
```

In [92]:
```
###################################
# Implementing the iterative imputer
###################################
cancer_rate_imputed_numeric_array = iterative_imputer.fit_transform(cancer_rate_cle
```

In [93]:
```
###################################
# Transforming the imputed data
```

```
# from an array to a dataframe
###################################
cancer_rate_imputed_numeric = pd.DataFrame(cancer_rate_imputed_numeric_array,
                                           columns = cancer_rate_cleaned_numeric.co
```

In [94]:
```
###################################
# Taking a snapshot of the imputed dataset
###################################
cancer_rate_imputed_numeric.head()
```

Out[94]:

	GDPPER	URBPOP	POPGRO	LIFEXP	TUBINC	DTHCMD	AGRLND	GHGEM
0	98380.63601	86.241	1.235701	83.200000	7.2	4.941054	46.252480	5.719031e+0
1	77541.76438	86.699	2.204789	82.256098	7.2	4.354730	38.562911	8.015803e+0
2	198405.87500	63.653	1.029111	82.556098	5.3	5.684596	65.495718	5.949773e+0
3	130941.63690	82.664	0.964348	76.980488	2.3	5.302060	44.363367	5.505181e+0
4	113300.60110	88.116	0.291641	81.602439	4.1	6.826140	65.499675	4.113555e+0

In [95]:
```
###################################
# Formulating the cleaned dataset
# with categorical columns only
###################################
cancer_rate_cleaned_categorical = cancer_rate_cleaned.select_dtypes(include='catego
```

In [96]:
```
###################################
# Imputing the missing data
# for categorical columns with
# the most frequent category
###################################
cancer_rate_cleaned_categorical['HDICAT'].fillna(cancer_rate_cleaned_categorical['H
cancer_rate_imputed_categorical = cancer_rate_cleaned_categorical.reset_index(drop=
```

```
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\787869593.py:6: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cancer_rate_cleaned_categorical['HDICAT'].fillna(cancer_rate_cleaned_categorical
['HDICAT'].mode()[0], inplace=True)
```

In [97]:
```
###################################
# Formulating the imputed dataset
###################################
cancer_rate_imputed = pd.concat([cancer_rate_imputed_numeric,cancer_rate_imputed_ca
```

In [98]:
```
###################################
# Gathering the data types for each column
###################################
data_type_list = list(cancer_rate_imputed.dtypes)
```

In [99]:
```
###################################
# Gathering the variable names for each column
###################################
variable_name_list = list(cancer_rate_imputed.columns)
```

In [100…
```
###################################
# Gathering the number of observations for each column
###################################
row_count_list = list([len(cancer_rate_imputed)] * len(cancer_rate_imputed.columns))
```

In [101…
```
###################################
# Gathering the number of missing data for each column
###################################
null_count_list = list(cancer_rate_imputed.isna().sum(axis=0))
```

In [102…
```
###################################
# Gathering the number of non-missing data for each column
###################################
non_null_count_list = list(cancer_rate_imputed.count())
```

In [103…
```
###################################
# Gathering the missing data percentage for each column
###################################
fill_rate_list = map(truediv, non_null_count_list, row_count_list)
```

In [104…
```
###################################
# Formulating the summary
# for all imputed columns
###################################
imputed_column_quality_summary = pd.DataFrame(zip(variable_name_list,
                                                  data_type_list,
                                                  row_count_list,
                                                  non_null_count_list,
                                                  null_count_list,
                                                  fill_rate_list),
                                              columns=['Column.Name',
                                                       'Column.Type',
                                                       'Row.Count',
                                                       'Non.Null.Count',
                                                       'Null.Count',
                                                       'Fill.Rate'])
display(imputed_column_quality_summary)
```

	Column.Name	Column.Type	Row.Count	Non.Null.Count	Null.Count	Fill.Rate
0	GDPPER	float64	163	163	0	1.0
1	URBPOP	float64	163	163	0	1.0
2	POPGRO	float64	163	163	0	1.0
3	LIFEXP	float64	163	163	0	1.0
4	TUBINC	float64	163	163	0	1.0
5	DTHCMD	float64	163	163	0	1.0
6	AGRLND	float64	163	163	0	1.0
7	GHGEMI	float64	163	163	0	1.0
8	METEMI	float64	163	163	0	1.0
9	FORARE	float64	163	163	0	1.0
10	CO2EMI	float64	163	163	0	1.0
11	PM2EXP	float64	163	163	0	1.0
12	POPDEN	float64	163	163	0	1.0
13	GDPCAP	float64	163	163	0	1.0
14	EPISCO	float64	163	163	0	1.0
15	CANRAT	category	163	163	0	1.0
16	HDICAT	category	163	163	0	1.0

1.4.3 Outlier Detection

1. High number of outliers observed for 5 numeric variables with Outlier.Ratio>0.10 and marginal to high Skewness.
   - PM2EXP: Outlier.Count = 37, Outlier.Ratio = 0.226, Skewness=-3.061
   - GHGEMI: Outlier.Count = 27, Outlier.Ratio = 0.165, Skewness=+9.299
   - GDPCAP: Outlier.Count = 22, Outlier.Ratio = 0.134, Skewness=+2.311
   - POPDEN: Outlier.Count = 20, Outlier.Ratio = 0.122, Skewness=+9.972
   - METEMI: Outlier.Count = 20, Outlier.Ratio = 0.122, Skewness=+5.688
2. Minimal number of outliers observed for 5 numeric variables with Outlier.Ratio<0.10 and normal Skewness.
   - TUBINC: Outlier.Count = 12, Outlier.Ratio = 0.073, Skewness=+1.747
   - CO2EMI: Outlier.Count = 11, Outlier.Ratio = 0.067, Skewness=+2.693
   - GDPPER: Outlier.Count = 3, Outlier.Ratio = 0.018, Skewness=+1.554
   - EPISCO: Outlier.Count = 3, Outlier.Ratio = 0.018, Skewness=+0.635
   - CANRAT: Outlier.Count = 2, Outlier.Ratio = 0.012, Skewness=+0.910

In [105…
```
###################################
# Formulating the imputed dataset
# with numeric columns only
###################################
cancer_rate_imputed_numeric = cancer_rate_imputed.select_dtypes(include='number')
```

In [106…
```
###################################
# Gathering the variable names for each numeric column
###################################
numeric_variable_name_list = list(cancer_rate_imputed_numeric.columns)
```

In [107…
```
###################################
# Gathering the skewness value for each numeric column
###################################
numeric_skewness_list = cancer_rate_imputed_numeric.skew()
```

In [108…
```
###################################
# Computing the interquartile range
# for all columns
###################################
cancer_rate_imputed_numeric_q1 = cancer_rate_imputed_numeric.quantile(0.25)
cancer_rate_imputed_numeric_q3 = cancer_rate_imputed_numeric.quantile(0.75)
cancer_rate_imputed_numeric_iqr = cancer_rate_imputed_numeric_q3 - cancer_rate_impu
```

In [109…
```
###################################
# Gathering the outlier count for each numeric column
# based on the interquartile range criterion
###################################
numeric_outlier_count_list = ((cancer_rate_imputed_numeric < (cancer_rate_imputed_n
```

In [110…
```
###################################
# Gathering the number of observations for each column
###################################
numeric_row_count_list = list([len(cancer_rate_imputed_numeric)] * len(cancer_rate_
```

In [111…
```
###################################
# Gathering the unique to count ratio for each categorical column
###################################
numeric_outlier_ratio_list = map(truediv, numeric_outlier_count_list, numeric_row_c
```

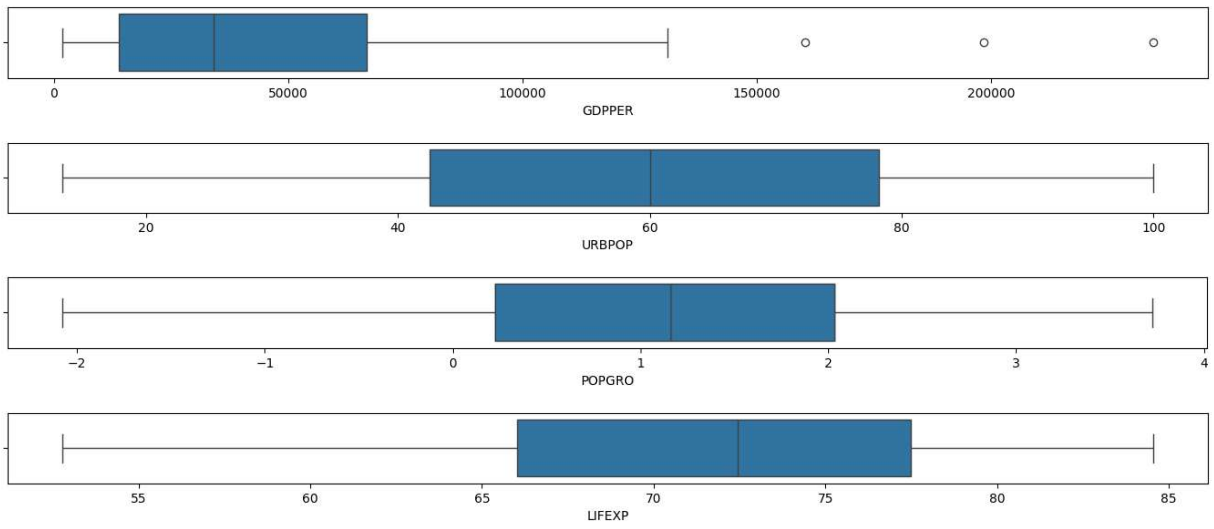
In [112…
```
###################################
# Formulating the outlier summary
# for all numeric columns
###################################
numeric_column_outlier_summary = pd.DataFrame(zip(numeric_variable_name_list,
                                                  numeric_skewness_list,
                                                  numeric_outlier_count_list,
                                                  numeric_row_count_list,
                                                  numeric_outlier_ratio_list),
                                              columns=['Numeric.Column.Name',
                                                       'Skewness',
                                                       'Outlier.Count',
                                                       'Row.Count',
```

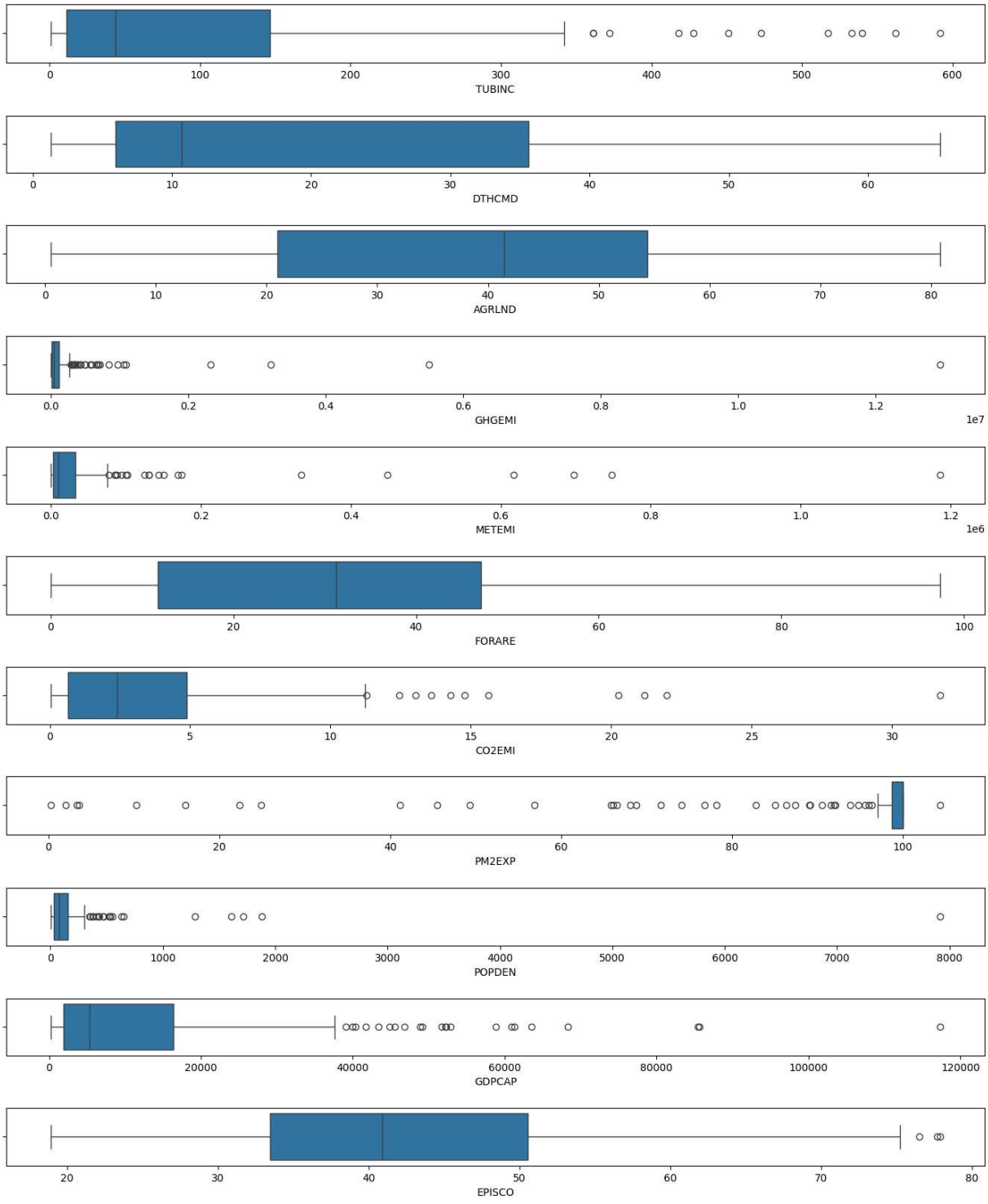
```
                                                    'Outlier.Ratio'])
display(numeric_column_outlier_summary)
```

	Numeric.Column.Name	Skewness	Outlier.Count	Row.Count	Outlier.Ratio
0	GDPPER	1.554457	3	163	0.018405
1	URBPOP	-0.212327	0	163	0.000000
2	POPGRO	-0.181666	0	163	0.000000
3	LIFEXP	-0.329704	0	163	0.000000
4	TUBINC	1.747962	12	163	0.073620
5	DTHCMD	0.930709	0	163	0.000000
6	AGRLND	0.035315	0	163	0.000000
7	GHGEMI	9.299960	27	163	0.165644
8	METEMI	5.688689	20	163	0.122699
9	FORARE	0.563015	0	163	0.000000
10	CO2EMI	2.693585	11	163	0.067485
11	PM2EXP	-3.088403	37	163	0.226994
12	POPDEN	9.972806	20	163	0.122699
13	GDPCAP	2.311079	22	163	0.134969
14	EPISCO	0.635994	3	163	0.018405

In [113…

```
###################################
# Formulating the individual boxplots
# for all numeric columns
###################################
for column in cancer_rate_imputed_numeric:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=cancer_rate_imputed_numeric, x=column)
```





1.4.4 Collinearity

Pearson's Correlation Coefficient is a parametric measure of the linear correlation for a pair of features by calculating the ratio between their covariance and the product of their standard deviations. The presence of high absolute correlation values indicate the univariate association between the numeric predictors and the numeric response.

1. Majority of the numeric variables reported moderate to high correlation which were statistically significant.

2. Among pairwise combinations of numeric variables, high Pearson.Correlation.Coefficient values were noted for:
    - GDPPER and GDPCAP: Pearson.Correlation.Coefficient = +0.921
    - GHGEMI and METEMI: Pearson.Correlation.Coefficient = +0.905
3. Among the highly correlated pairs, variables with the lowest correlation against the target variable were removed.
    - GDPPER: Pearson.Correlation.Coefficient = +0.690
    - METEMI: Pearson.Correlation.Coefficient = +0.062
4. The cleaned dataset is comprised of:
    - **163 rows** (observations)
    - **16 columns** (variables)
        - **1/16 metadata** (object)
            - COUNTRY
        - **1/16 target** (categorical)
            - CANRAT
        - **13/16 predictor** (numeric)
            - URBPOP
            - POPGRO
            - LIFEXP
            - TUBINC
            - DTHCMD
            - AGRLND
            - GHGEMI
            - FORARE
            - CO2EMI
            - PM2EXP
            - POPDEN
            - GDPCAP
            - EPISCO
        - **1/16 predictor** (categorical)
            - HDICAT

In [114…

```
###################################
# Formulating a function
# to plot the correlation matrix
# for all pairwise combinations
# of numeric columns
###################################
def plot_correlation_matrix(corr, mask=None):
    f, ax = plt.subplots(figsize=(11, 9))
    sns.heatmap(corr,
                ax=ax,
                mask=mask,
                annot=True,
                vmin=-1,
                vmax=1,
```

```
                center=0,
                cmap='coolwarm',
                linewidths=1,
                linecolor='gray',
                cbar_kws={'orientation': 'horizontal'})
```

In [115…

```
###################################
# Computing the correlation coefficients
# and correlation p-values
# among pairs of numeric columns
###################################
cancer_rate_imputed_numeric_correlation_pairs = {}
cancer_rate_imputed_numeric_columns = cancer_rate_imputed_numeric.columns.tolist()
for numeric_column_a, numeric_column_b in itertools.combinations(cancer_rate_impute
    cancer_rate_imputed_numeric_correlation_pairs[numeric_column_a + '_' + numeric_
        cancer_rate_imputed_numeric.loc[:, numeric_column_a],
        cancer_rate_imputed_numeric.loc[:, numeric_column_b])
```

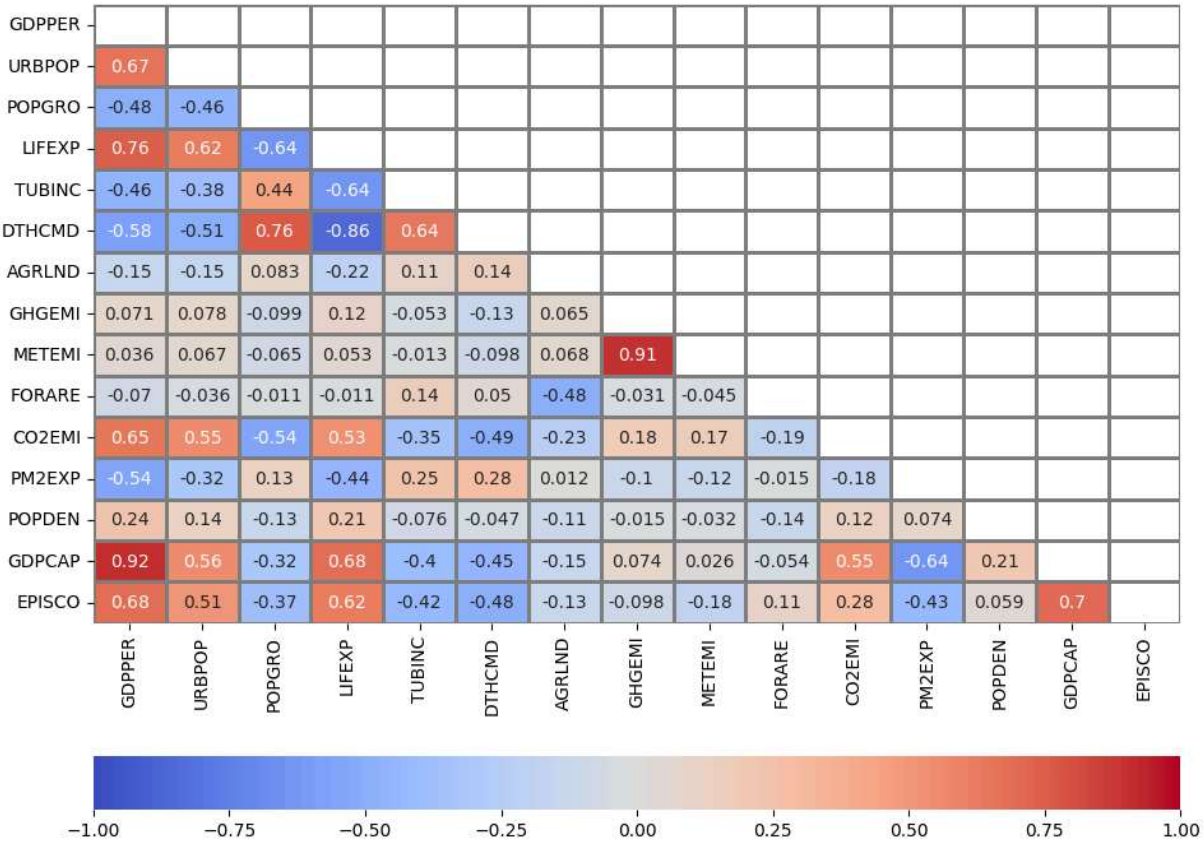
In [116…

```
###################################
# Formulating the pairwise correlation summary
# for all numeric columns
###################################
cancer_rate_imputed_numeric_summary = cancer_rate_imputed_numeric.from_dict(cancer_
cancer_rate_imputed_numeric_summary.columns = ['Pearson.Correlation.Coefficient', '
display(cancer_rate_imputed_numeric_summary.sort_values(by=['Pearson.Correlation.Co
```

	Pearson.Correlation.Coefficient	Correlation.PValue
GDPPER_GDPCAP	0.921010	8.158179e-68
GHGEMI_METEMI	0.905121	1.087643e-61
POPGRO_DTHCMD	0.759470	7.124695e-32
GDPPER_LIFEXP	0.755787	2.055178e-31
GDPCAP_EPISCO	0.696707	5.312642e-25
LIFEXP_GDPCAP	0.683834	8.321371e-24
GDPPER_EPISCO	0.680812	1.555304e-23
GDPPER_URBPOP	0.666394	2.781623e-22
GDPPER_CO2EMI	0.654958	2.450029e-21
TUBINC_DTHCMD	0.643615	1.936081e-20
URBPOP_LIFEXP	0.623997	5.669778e-19
LIFEXP_EPISCO	0.620271	1.048393e-18
URBPOP_GDPCAP	0.559181	8.624533e-15
CO2EMI_GDPCAP	0.550221	2.782997e-14
URBPOP_CO2EMI	0.550046	2.846393e-14
LIFEXP_CO2EMI	0.531305	2.951829e-13
URBPOP_EPISCO	0.510131	3.507463e-12
POPGRO_TUBINC	0.442339	3.384403e-09
DTHCMD_PM2EXP	0.283199	2.491837e-04
CO2EMI_EPISCO	0.282734	2.553620e-04

In [117…

```
###################################
# Plotting the correlation matrix
# for all pairwise combinations
# of numeric columns
###################################
cancer_rate_imputed_numeric_correlation = cancer_rate_imputed_numeric.corr()
mask = np.triu(cancer_rate_imputed_numeric_correlation)
plot_correlation_matrix(cancer_rate_imputed_numeric_correlation,mask)
plt.show()
```

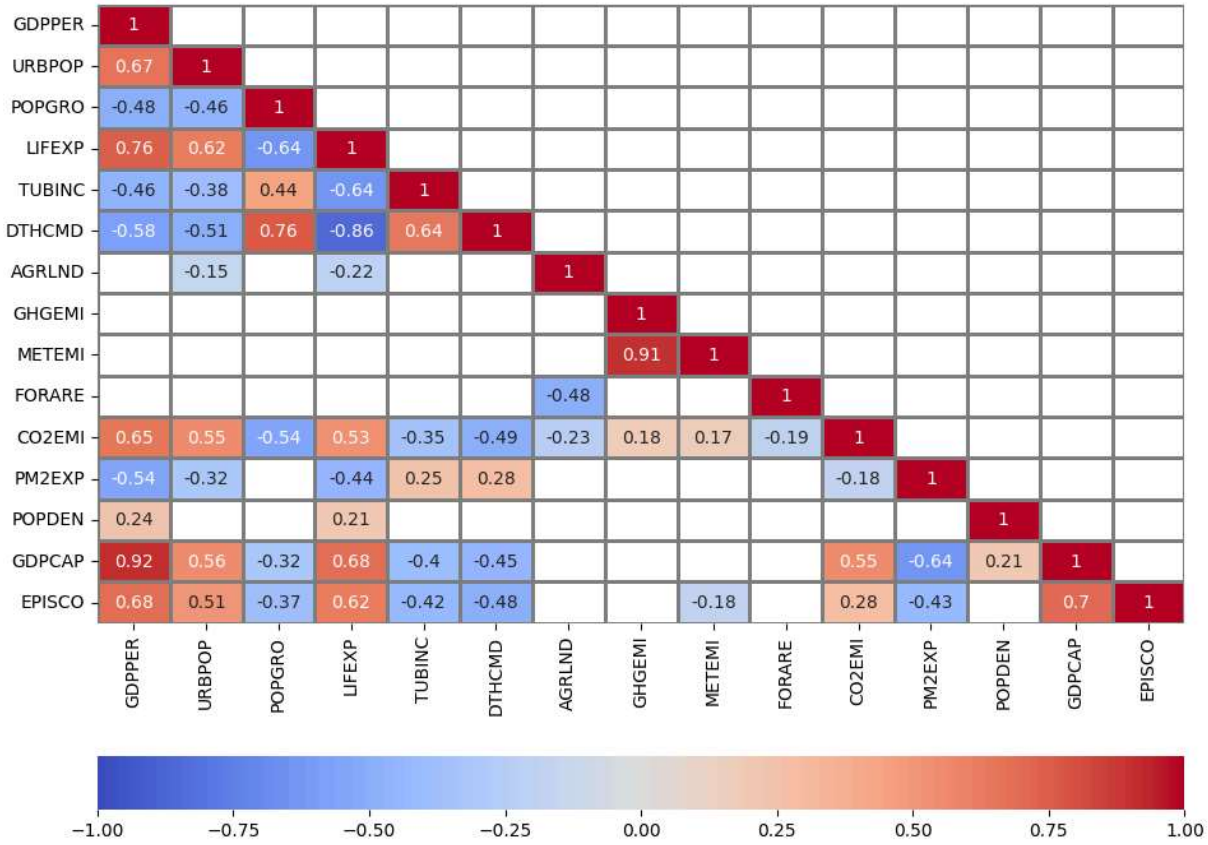


In [118…

```
###################################
# Formulating a function
# to plot the correlation matrix
# for all pairwise combinations
# of numeric columns
# with significant p-values only
###################################
def correlation_significance(df=None):
    p_matrix = np.zeros(shape=(df.shape[1],df.shape[1]))
    for col in df.columns:
        for col2 in df.drop(col,axis=1).columns:
            _ , p = stats.pearsonr(df[col],df[col2])
            p_matrix[df.columns.to_list().index(col),df.columns.to_list().index(col
    return p_matrix
```

In [119…

```
###################################
# Plotting the correlation matrix
# for all pairwise combinations
# of numeric columns
# with significant p-values only
###################################
cancer_rate_imputed_numeric_correlation_p_values = correlation_significance(cancer_
mask = np.invert(np.tril(cancer_rate_imputed_numeric_correlation_p_values<0.05))
plot_correlation_matrix(cancer_rate_imputed_numeric_correlation,mask)
```



In [120…

```
###################################
# Filtering out one among the
# highly correlated variable pairs with
# lesser Pearson.Correlation.Coefficient
# when compared to the target variable
###################################
cancer_rate_imputed_numeric.drop(['GDPPER','METEMI'], inplace=True, axis=1)
```

In [121…

```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_imputed_numeric.shape)
```

```
Dataset Dimensions:
(163, 13)
```

1.4.5 Shape Transformation

Yeo-Johnson Transformation applies a new family of distributions that can be used without restrictions, extending many of the good properties of the Box-Cox power family. Similar to the Box-Cox transformation, the method also estimates the optimal value of lambda but has the ability to transform both positive and negative values by inflating low variance data and deflating high variance data to create a more uniform data set. While there are no restrictions in terms of the applicable values, the interpretability of the transformed values is more diminished as compared to the other methods.

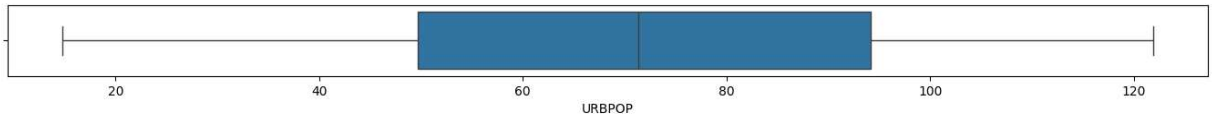
1. A Yeo-Johnson transformation was applied to all numeric variables to improve distributional shape.
2. Most variables achieved symmetrical distributions with minimal outliers after transformation.
3. One variable which remained skewed even after applying shape transformation was removed.
    - PM2EXP
4. The transformed dataset is comprised of:
    - **163 rows** (observations)
    - **15 columns** (variables)
        - **1/15 metadata** (object)
            - COUNTRY
        - **1/15 target** (categorical)
            - CANRAT
        - **12/15 predictor** (numeric)
            - URBPOP
            - POPGRO
            - LIFEXP
            - TUBINC
            - DTHCMD
            - AGRLND
            - GHGEMI
            - FORARE
            - CO2EMI
            - POPDEN
            - GDPCAP
            - EPISCO
        - **1/15 predictor** (categorical)
            - HDICAT

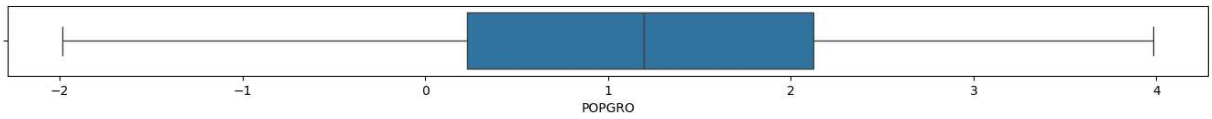
In [122…
```
###################################
# Conducting a Yeo-Johnson Transformation
# to address the distributional
# shape of the variables
###################################
yeo_johnson_transformer = PowerTransformer(method='yeo-johnson',
                                           standardize=False)
cancer_rate_imputed_numeric_array = yeo_johnson_transformer.fit_transform(cancer_ra
```

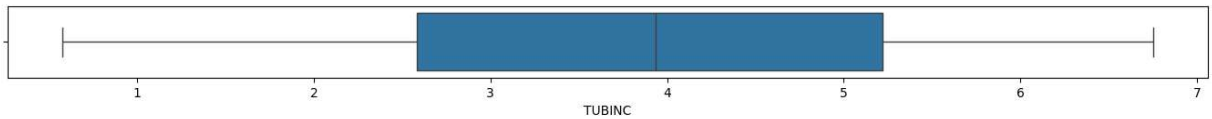
In [123…
```
###################################
# Formulating a new dataset object
# for the transformed data
###################################
cancer_rate_transformed_numeric = pd.DataFrame(cancer_rate_imputed_numeric_array,
                                               columns=cancer_rate_imputed_numeric.
```

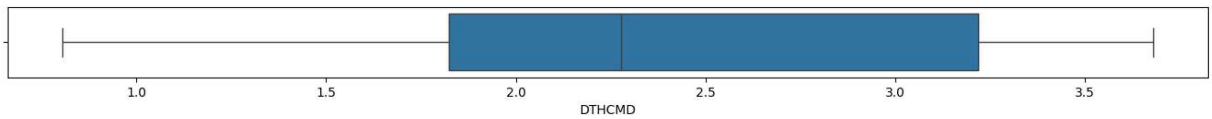
In [124…

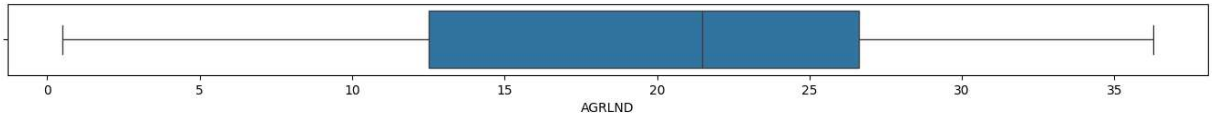
```
###################################
# Formulating the individual boxplots
# for all transformed numeric columns
###################################
for column in cancer_rate_transformed_numeric:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=cancer_rate_transformed_numeric, x=column)
```

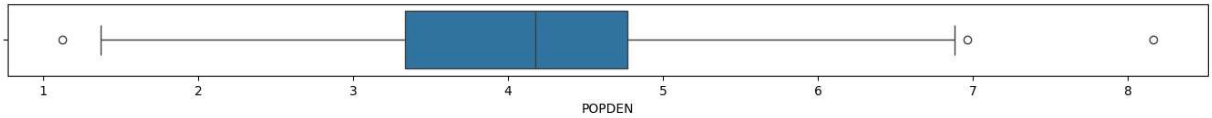


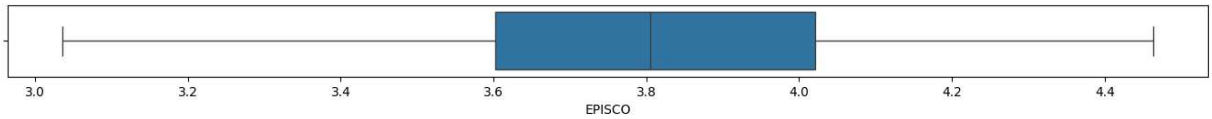












In [125…
```
###################################
# Filtering out the column
# which remained skewed even
# after applying shape transformation
###################################
cancer_rate_transformed_numeric.drop(['PM2EXP'], inplace=True, axis=1)
```

In [126…
```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_transformed_numeric.shape)
```

```
Dataset Dimensions:
(163, 12)
```

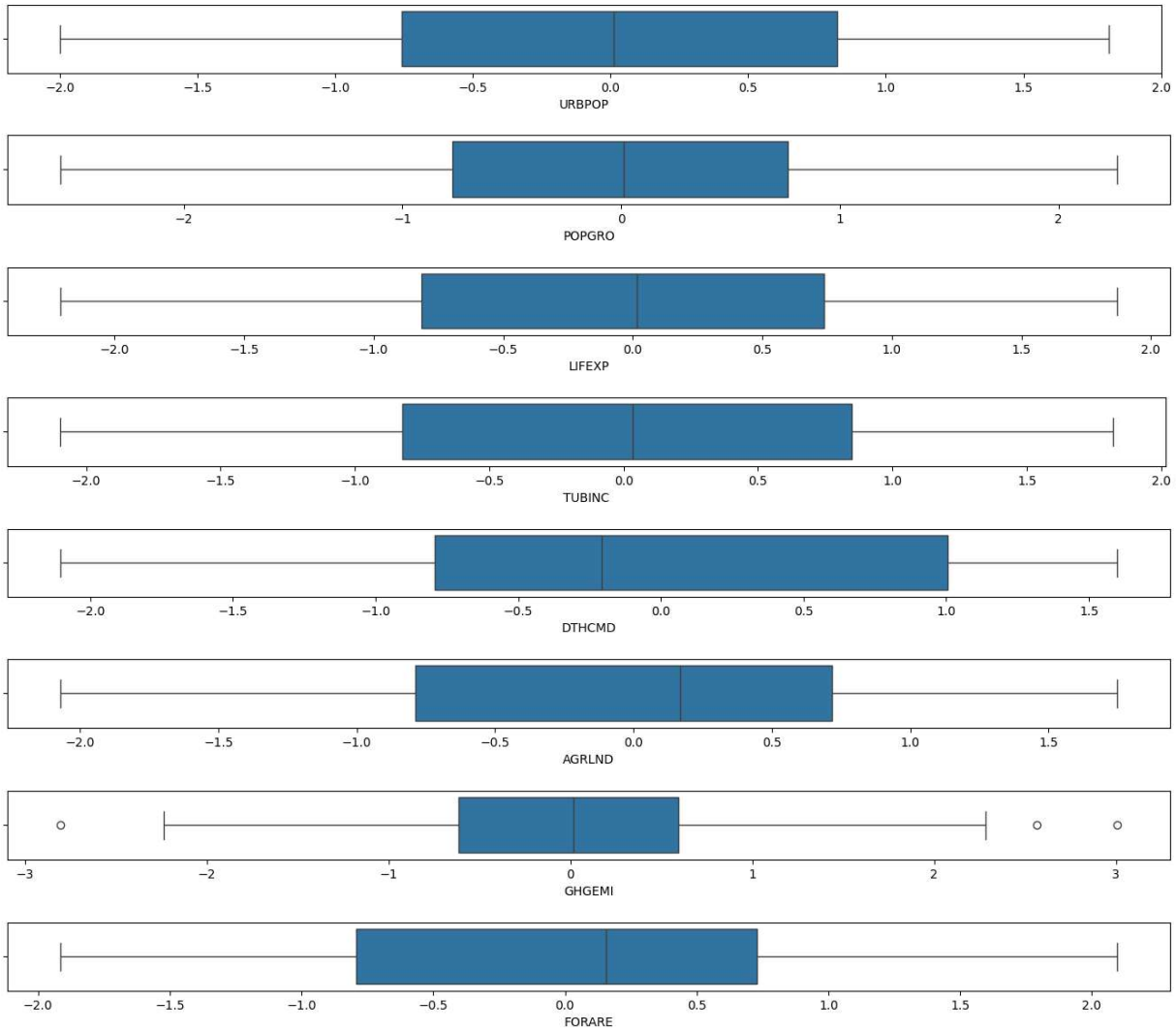
1.4.6 Centering and Scaling

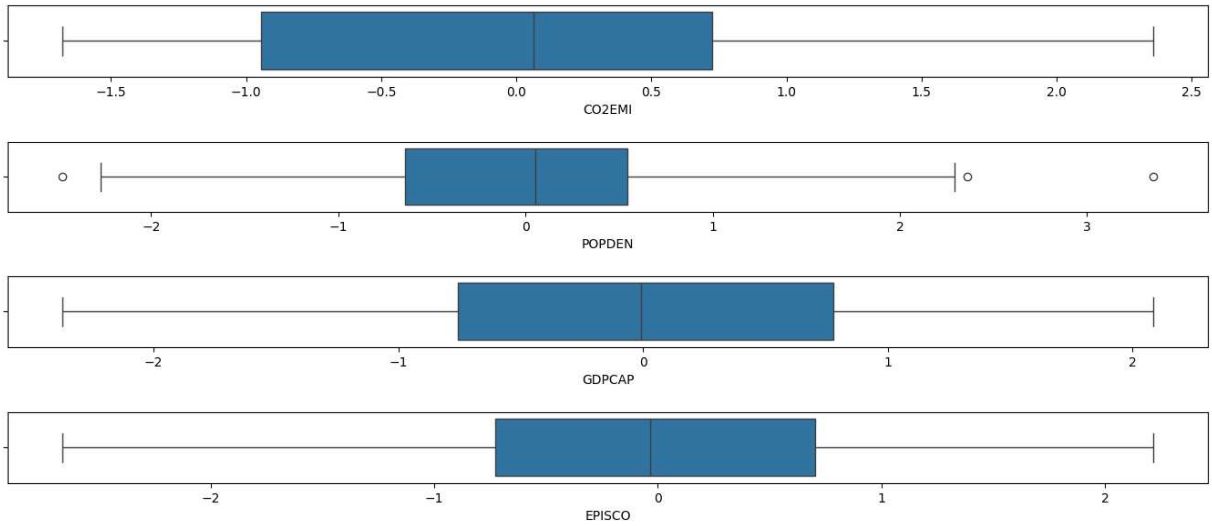
1. All numeric variables were transformed using the standardization method to achieve a comparable scale between values.
2. The scaled dataset is comprised of:
    - **163 rows** (observations)
    - **15 columns** (variables)
        - **1/15 metadata** (object)
            - COUNTRY
        - **1/15 target** (categorical)
            - CANRAT
        - **12/15 predictor** (numeric)
            - URBPOP
            - POPGRO
            - LIFEXP
            - TUBINC
            - DTHCMD
            - AGRLND
            - GHGEMI
            - FORARE
            - CO2EMI
            - POPDEN
            - GDPCAP
            - EPISCO
        - **1/15 predictor** (categorical)
            - HDICAT

In [127…
```
###################################
# Conducting standardization
# to transform the values of the
# variables into comparable scale
###################################
standardization_scaler = StandardScaler()
cancer_rate_transformed_numeric_array = standardization_scaler.fit_transform(cancer
```

In [128…
```
###################################
# Formulating a new dataset object
# for the scaled data
###################################
cancer_rate_scaled_numeric = pd.DataFrame(cancer_rate_transformed_numeric_array,
                                          columns=cancer_rate_transformed_numeric.c
```

In [129…
```
###################################
# Formulating the individual boxplots
# for all transformed numeric columns
###################################
for column in cancer_rate_scaled_numeric:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=cancer_rate_scaled_numeric, x=column)
```





1.4.7 Data Encoding

1. One-hot encoding was applied to the HDICAP_VH variable resulting to 4 additional columns in the dataset:
    - HDICAP_L
    - HDICAP_M
    - HDICAP_H
    - HDICAP_VH

In [130…
```
###################################
# Formulating the categorical column
# for encoding transformation
###################################
cancer_rate_categorical_encoded = pd.DataFrame(cancer_rate_cleaned_categorical.loc[
                                               columns=['HDICAT'])
```

In [131…
```
###################################
# Applying a one-hot encoding transformation
# for the categorical column
###################################
cancer_rate_categorical_encoded = pd.get_dummies(cancer_rate_categorical_encoded, c
```

1.4.8 Preprocessed Data Description

1. The preprocessed dataset is comprised of:
    - **163 rows** (observations)
    - **18 columns** (variables)
        - **1/18 metadata** (object)
            - COUNTRY
        - **1/18 target** (categorical)
            - CANRAT
        - **12/18 predictor** (numeric)

        - URBPOP
        - POPGRO
        - LIFEXP
        - TUBINC
        - DTHCMD
        - AGRLND
        - GHGEMI
        - FORARE
        - CO2EMI
        - POPDEN
        - GDPCAP
        - EPISCO
    - **4/18 predictor** (categorical)
        - HDICAT_L
        - HDICAT_M
        - HDICAT_H
        - HDICAT_VH

In [132…
```
###################################
# Consolidating both numeric columns
# and encoded categorical columns
###################################
cancer_rate_preprocessed = pd.concat([cancer_rate_scaled_numeric,cancer_rate_catego
```

In [133…
```
###################################
# Performing a general exploration of the consolidated dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_preprocessed.shape)
```

```
Dataset Dimensions:
(163, 16)
```

1.5. Data Exploration

1.5.1 Exploratory Data Analysis

1. Bivariate analysis identified individual predictors with generally positive association to the target variable based on visual inspection.
2. Higher values or higher proportions for the following predictors are associated with the CANRAT HIGH category:
    - URBPOP
    - LIFEXP
    - CO2EMI
    - GDPCAP
    - EPISCO

- HDICAP_VH=1

3. Decreasing values or smaller proportions for the following predictors are associated with the CANRAT LOW category:
   - POPGRO
   - TUBINC
   - DTHCMD
   - HDICAP_L=0
   - HDICAP_M=0
   - HDICAP_H=0
4. Values for the following predictors are not associated with the CANRAT HIGH or LOW categories:
   - AGRLND
   - GHGEMI
   - FORARE
   - POPDEN

In [134…
```
###################################
# Segregating the target
# and predictor variable lists
###################################
cancer_rate_preprocessed_target = cancer_rate_filtered_row['CANRAT'].to_frame()
cancer_rate_preprocessed_target.reset_index(inplace=True, drop=True)
cancer_rate_preprocessed_categorical = cancer_rate_preprocessed[cancer_rate_categor
cancer_rate_preprocessed_categorical_combined = cancer_rate_preprocessed_categorica
cancer_rate_preprocessed = cancer_rate_preprocessed.drop(cancer_rate_categorical_en
cancer_rate_preprocessed_predictors = cancer_rate_preprocessed.columns
cancer_rate_preprocessed_combined = cancer_rate_preprocessed.join(cancer_rate_prepr
cancer_rate_preprocessed_all = cancer_rate_preprocessed_combined.join(cancer_rate_c
```

In [135…
```
###################################
# Segregating the target
# and predictor variable names
###################################
y_variable = 'CANRAT'
x_variables = cancer_rate_preprocessed_predictors
```

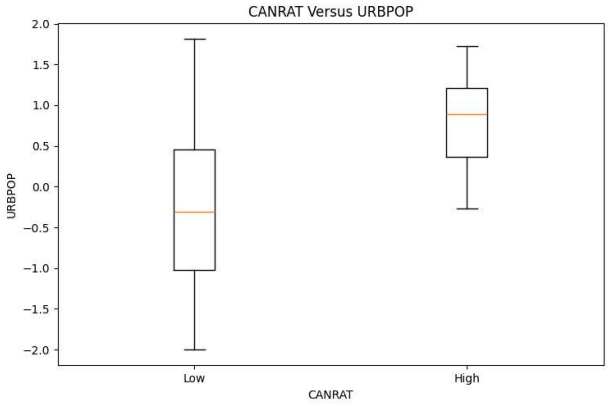
In [136…
```
###################################
# Defining the number of
# rows and columns for the subplots
###################################
num_rows = 6
num_cols = 2
```

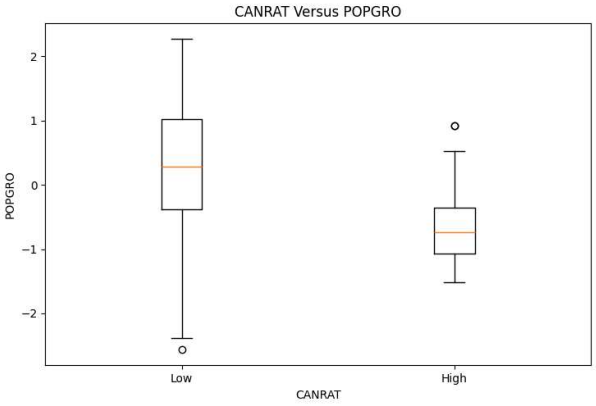
In [137…
```
###################################
# Formulating the subplot structure
###################################
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 30))

###################################
# Flattening the multi-row and
```

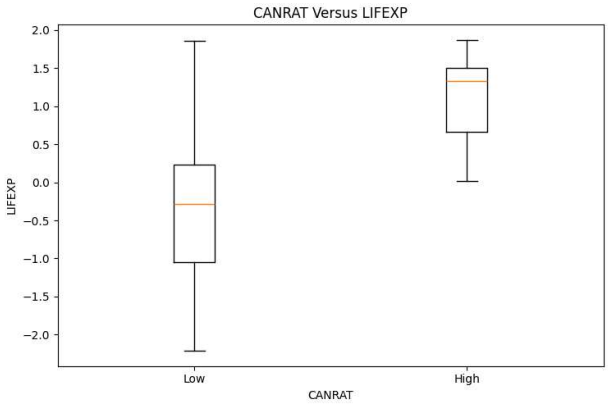
```
# multi-column axes
###################################
axes = axes.ravel()

###################################
# Formulating the individual boxplots
# for all scaled numeric columns
###################################
for i, x_variable in enumerate(x_variables):
    ax = axes[i]
    ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combi
    ax.set_title(f'{y_variable} Versus {x_variable}')
    ax.set_xlabel(y_variable)
    ax.set_ylabel(x_variable)
    ax.set_xticks(range(1, len(cancer_rate_preprocessed_combined[y_variable].unique

###################################
# Adjusting the subplot layout
###################################
plt.tight_layout()

###################################
# Presenting the subplots
###################################
plt.show()
```

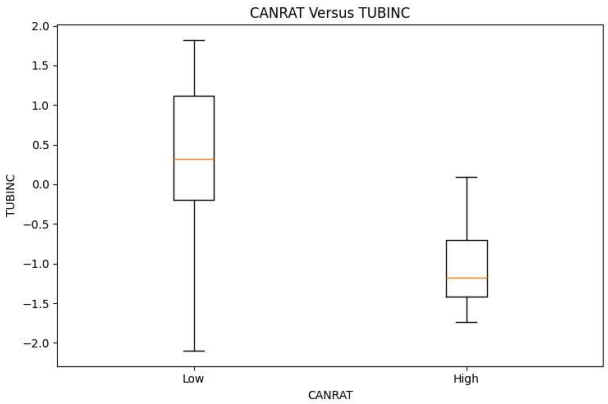
```
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
```

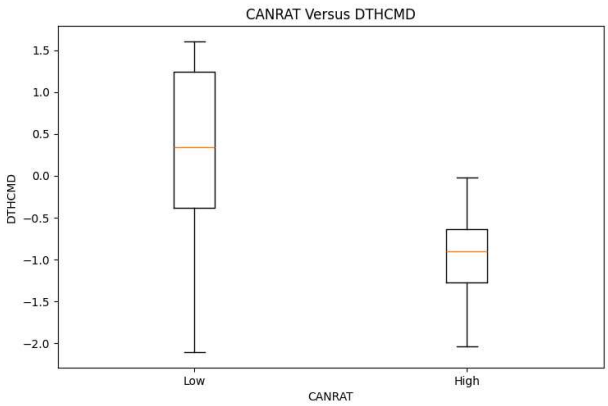
```
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\3632850337.py:18: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=Tru
e to adopt the future default and silence this warning.
  ax.boxplot([group[x_variable] for name, group in cancer_rate_preprocessed_combine
d.groupby(y_variable)])
```

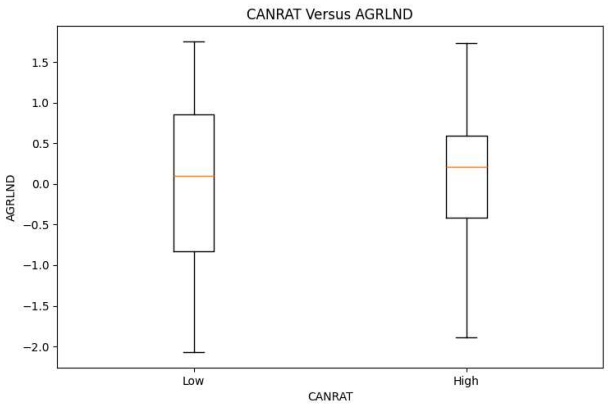

CANRAT Versus URBPOP
URBPOP
CANRAT
Low
High

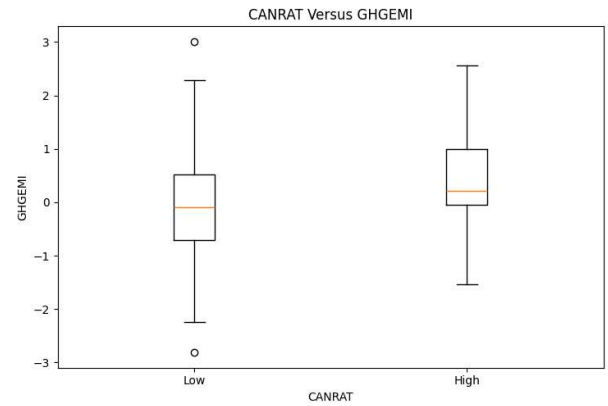

CANRAT Versus POPGRO
POPGRO
CANRAT
Low
High

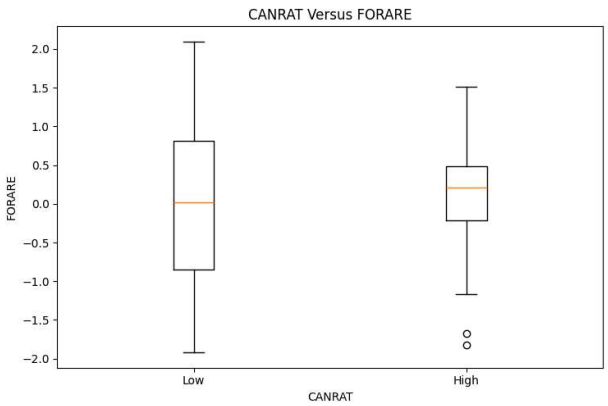

CANRAT Versus LIFEXP
LIFEXP
CANRAT
Low
High

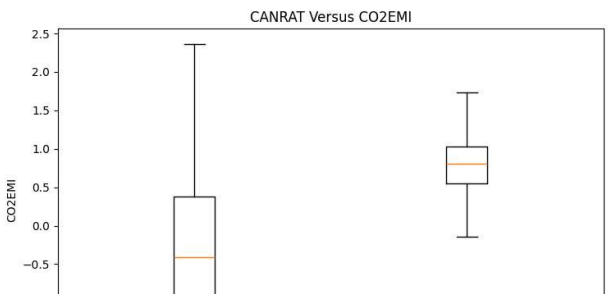

CANRAT Versus TUBINC
TUBINC
CANRAT
Low
High

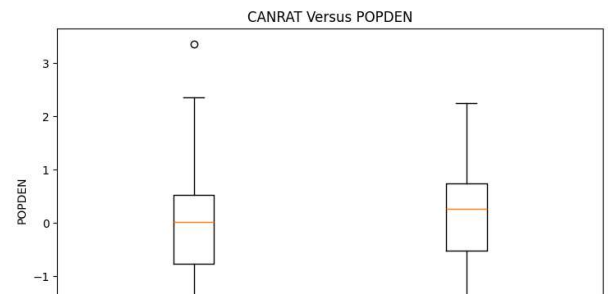

CANRAT Versus DTHCMD
DTHCMD
CANRAT
Low
High

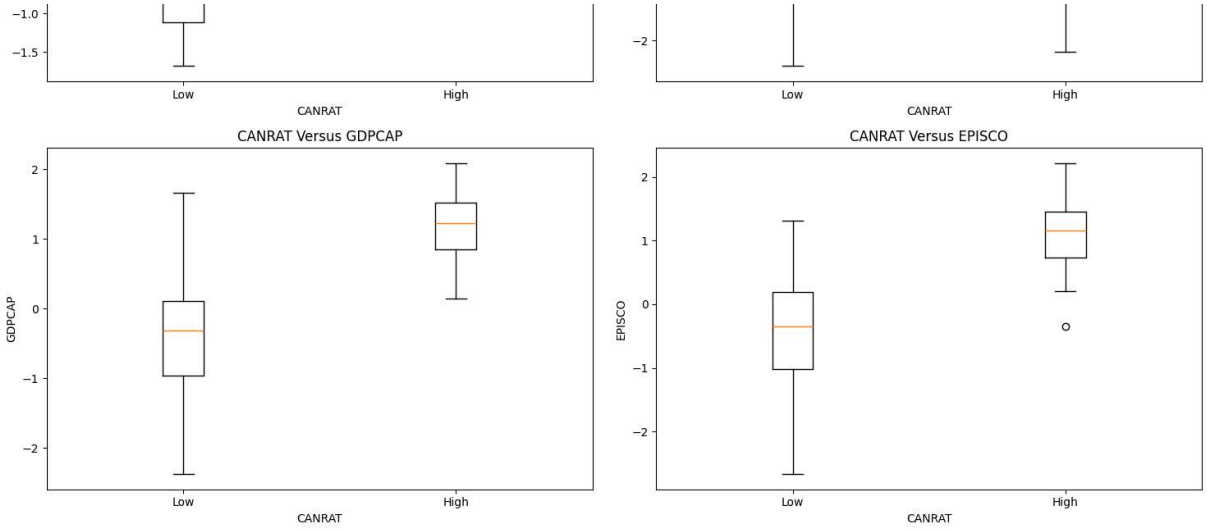

CANRAT Versus AGRLND
AGRLND
CANRAT
Low
High


CANRAT Versus GHGEMI
GHGEMI
CANRAT
Low
High


CANRAT Versus FORARE
FORARE
CANRAT
Low
High


CANRAT Versus CO2EMI
CO2EMI


CANRAT Versus POPDEN
POPDEN

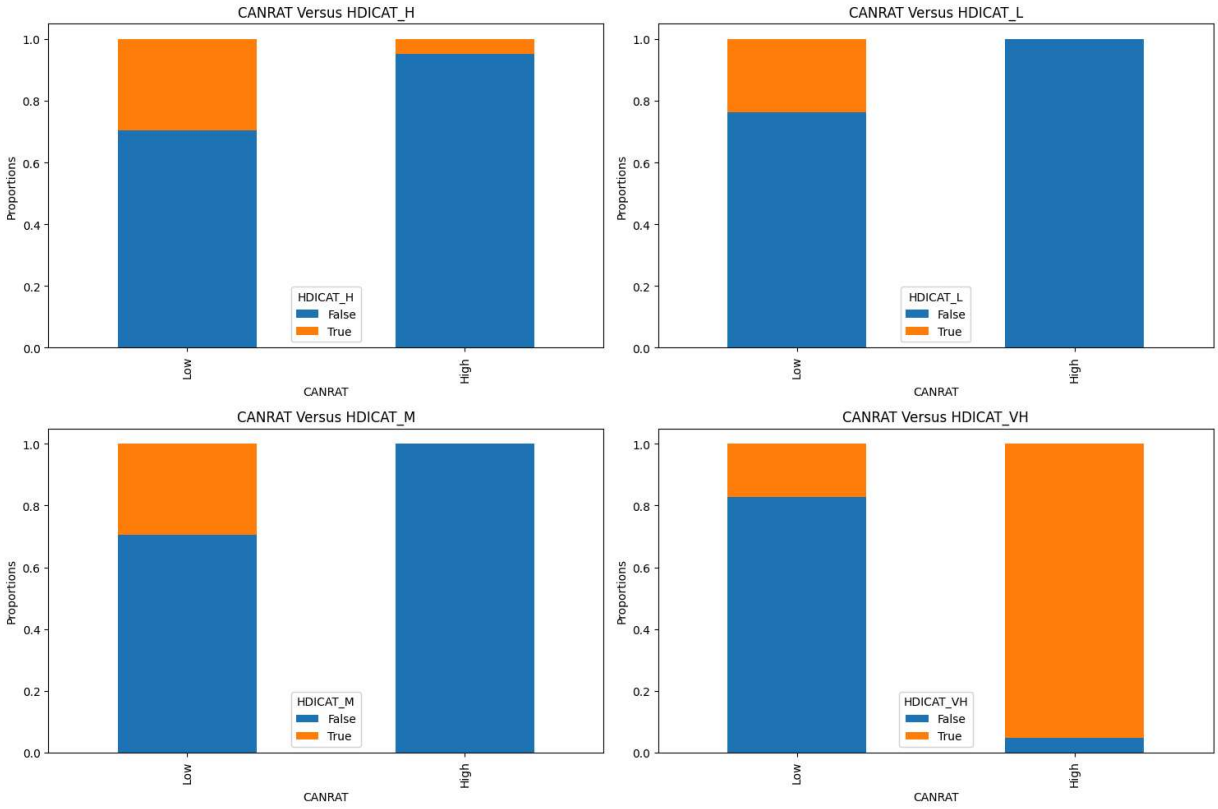


In [138…

```
###################################
# Segregating the target
# and predictor variable names
###################################
y_variables = cancer_rate_preprocessed_categorical.columns
x_variable = 'CANRAT'

###################################
# Defining the number of
# rows and columns for the subplots
###################################
num_rows = 2
num_cols = 2

###################################
# Formulating the subplot structure
###################################
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

###################################
# Flattening the multi-row and
# multi-column axes
###################################
axes = axes.ravel()

###################################
# Formulating the individual stacked column plots
# for all categorical columns
###################################
for i, y_variable in enumerate(y_variables):
    ax = axes[i]
    category_counts = cancer_rate_preprocessed_categorical_combined.groupby([x_vari
    category_proportions = category_counts.div(category_counts.sum(axis=1), axis=0)
    category_proportions.plot(kind='bar', stacked=True, ax=ax)
    ax.set_title(f'{x_variable} Versus {y_variable}')
    ax.set_xlabel(x_variable)
    ax.set_ylabel('Proportions')

###################################
```

```
# Adjusting the subplot layout
##################################
plt.tight_layout()

##################################
# Presenting the subplots
##################################
plt.show()
```

```
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\744851797.py:32: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  category_counts = cancer_rate_preprocessed_categorical_combined.groupby([x_variabl
e, y_variable]).size().unstack(fill_value=0)
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\744851797.py:32: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  category_counts = cancer_rate_preprocessed_categorical_combined.groupby([x_variabl
e, y_variable]).size().unstack(fill_value=0)
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\744851797.py:32: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  category_counts = cancer_rate_preprocessed_categorical_combined.groupby([x_variabl
e, y_variable]).size().unstack(fill_value=0)
C:\Users\117100631\AppData\Local\Temp\ipykernel_7964\744851797.py:32: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  category_counts = cancer_rate_preprocessed_categorical_combined.groupby([x_variabl
e, y_variable]).size().unstack(fill_value=0)
```



1.5.2 Hypothesis Testing

1. The relationship between the numeric predictors to the CANRAT target variable was statistically evaluated using the following hypotheses:
   - **Null**: Difference in the means between groups LOW and HIGH is equal to zero
   - **Alternative**: Difference in the means between groups LOW and HIGH is not equal to zero
2. There is sufficient evidence to conclude of a statistically significant difference between the means of the numeric measurements obtained from LOW and HIGH groups of the CANRAT target variable in 9 of the 12 numeric predictors given their high t-test statistic values with reported low p-values less than the significance level of 0.05.
   - GDPCAP: T.Test.Statistic=-11.937, T.Test.PValue=0.000
   - EPISCO: T.Test.Statistic=-11.789, T.Test.PValue=0.000
   - LIFEXP: T.Test.Statistic=-10.979, T.Test.PValue=0.000
   - TUBINC: T.Test.Statistic=+9.609, T.Test.PValue=0.000
   - DTHCMD: T.Test.Statistic=+8.376, T.Test.PValue=0.000
   - CO2EMI: T.Test.Statistic=-7.031, T.Test.PValue=0.000
   - URBPOP: T.Test.Statistic=-6.541, T.Test.PValue=0.000
   - POPGRO: T.Test.Statistic=+4.905, T.Test.PValue=0.000
   - GHGEMI: T.Test.Statistic=-2.243, T.Test.PValue=0.026
3. The relationship between the categorical predictors to the CANRAT target variable was statistically evaluated using the following hypotheses:
   - **Null**: The categorical predictor is independent of the categorical target variable

- **Alternative**: The categorical predictor is dependent of the categorical target variable

4. There is sufficient evidence to conclude of a statistically significant relationship difference between the categories of the categorical predictors and the LOW and HIGH groups of the CANRAT target variable in all 4 categorical predictors given their high chisquare statistic values with reported low p-values less than the significance level of 0.05.
    - HDICAT_VH: ChiSquare.Test.Statistic=76.764, ChiSquare.Test.PValue=0.000
    - HDICAT_H: ChiSquare.Test.Statistic=13.860, ChiSquare.Test.PValue=0.000
    - HDICAT_M: ChiSquare.Test.Statistic=10.286, ChiSquare.Test.PValue=0.001
    - HDICAT_L: ChiSquare.Test.Statistic=9.081, ChiSquare.Test.PValue=0.002

In [139…

```
###################################
# Computing the t-test
# statistic and p-values
# between the target variable
# and numeric predictor columns
###################################
cancer_rate_preprocessed_numeric_ttest_target = {}
cancer_rate_preprocessed_numeric = cancer_rate_preprocessed_combined
cancer_rate_preprocessed_numeric_columns = cancer_rate_preprocessed_predictors
for numeric_column in cancer_rate_preprocessed_numeric_columns:
    group_0 = cancer_rate_preprocessed_numeric[cancer_rate_preprocessed_numeric.loc
    group_1 = cancer_rate_preprocessed_numeric[cancer_rate_preprocessed_numeric.loc
    cancer_rate_preprocessed_numeric_ttest_target['CANRAT_' + numeric_column] = sta
        group_0[numeric_column],
        group_1[numeric_column],
        equal_var=True)
```

In [140…

```
###################################
# Formulating the pairwise ttest summary
# between the target variable
# and numeric predictor columns
###################################
cancer_rate_preprocessed_numeric_summary = cancer_rate_preprocessed_numeric.from_di
cancer_rate_preprocessed_numeric_summary.columns = ['T.Test.Statistic', 'T.Test.PVa
display(cancer_rate_preprocessed_numeric_summary.sort_values(by=['T.Test.PValue'],
```

	T.Test.Statistic	T.Test.PValue
CANRAT_GDPCAP	-11.936988	6.247937e-24
CANRAT_EPISCO	-11.788870	1.605980e-23
CANRAT_LIFEXP	-10.979098	2.754214e-21
CANRAT_TUBINC	9.608760	1.463678e-17
CANRAT_DTHCMD	8.375558	2.552108e-14
CANRAT_CO2EMI	-7.030702	5.537463e-11
CANRAT_URBPOP	-6.541001	7.734940e-10
CANRAT_POPGRO	4.904817	2.269446e-06
CANRAT_GHGEMI	-2.243089	2.625563e-02
CANRAT_FORARE	-1.174143	2.420717e-01
CANRAT_POPDEN	-0.495221	6.211191e-01
CANRAT_AGRLND	-0.047628	9.620720e-01

In [141…

```
###################################
# Computing the chisquare
# statistic and p-values
# between the target variable
# and categorical predictor columns
###################################
cancer_rate_preprocessed_categorical_chisquare_target = {}
cancer_rate_preprocessed_categorical = cancer_rate_preprocessed_categorical_combine
cancer_rate_preprocessed_categorical_columns = ['HDICAT_L','HDICAT_M','HDICAT_H','H
for categorical_column in cancer_rate_preprocessed_categorical_columns:
    contingency_table = pd.crosstab(cancer_rate_preprocessed_categorical[categorica
                                    cancer_rate_preprocessed_categorical['CANRAT'])
    cancer_rate_preprocessed_categorical_chisquare_target['CANRAT_' + categorical_c
        contingency_table)[0:2]
```

In [142…

```
###################################
# Formulating the pairwise chisquare summary
# between the target variable
# and categorical predictor columns
###################################
cancer_rate_preprocessed_categorical_summary = cancer_rate_preprocessed_categorical
cancer_rate_preprocessed_categorical_summary.columns = ['ChiSquare.Test.Statistic',
display(cancer_rate_preprocessed_categorical_summary.sort_values(by=['ChiSquare.Tes
```

	ChiSquare.Test.Statistic	ChiSquare.Test.PValue
CANRAT_HDICAT_VH	76.764134	1.926446e-18
CANRAT_HDICAT_M	13.860367	1.969074e-04
CANRAT_HDICAT_L	10.285575	1.340742e-03
CANRAT_HDICAT_H	9.080788	2.583087e-03

1.6. Model Development With Hyperparameter Tuning

Hyperparameter tuning is an iterative process that involves experimenting with different hyperparameter combinations, evaluating the model's performance, and refining the hyperparameter values to achieve the best possible performance on new, unseen data - aimed at building effective and well-generalizing machine learning models. A model's performance depends not only on the learned parameters (weights) during training but also on hyperparameters, which are external configuration settings that cannot be learned from the data.

1.6.1 Premodelling Data Description

1. Among the 9 numeric variables determined to have a statistically significant difference between the means of the numeric measurements obtained from LOW and HIGH groups of the CANRAT target variable, only 7 were retained with absolute T-Test statistics greater than 5.
   - GDPCAP: T.Test.Statistic=-11.937, T.Test.PValue=0.000
   - EPISCO: T.Test.Statistic=-11.789, T.Test.PValue=0.000
   - LIFEXP: T.Test.Statistic=-10.979, T.Test.PValue=0.000
   - TUBINC: T.Test.Statistic=+9.609, T.Test.PValue=0.000
   - DTHCMD: T.Test.Statistic=+8.376, T.Test.PValue=0.000
   - CO2EMI: T.Test.Statistic=-7.031, T.Test.PValue=0.000
   - URBPOP: T.Test.Statistic=-6.541, T.Test.PValue=0.000
2. Among the 4 categorical predictors determined to have a statistically significant relationship difference between the categories of the categorical predictors and the LOW and HIGH groups of the CANRAT target variable, only 1 was retained with absolute Chi-Square statistics greater than 15.
   - HDICAT_VH: ChiSquare.Test.Statistic=76.764, ChiSquare.Test.PValue=0.000
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.

In [143…

```
###################################
# Consolidating relevant numeric columns
# and encoded categorical columns
```

```
# after hypothesis testing
###################################
cancer_rate_premodelling = cancer_rate_preprocessed_all.drop(['AGRLND','POPDEN','GH
```

In [144…

```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_premodelling.shape)
```

```
Dataset Dimensions:
(163, 9)
```

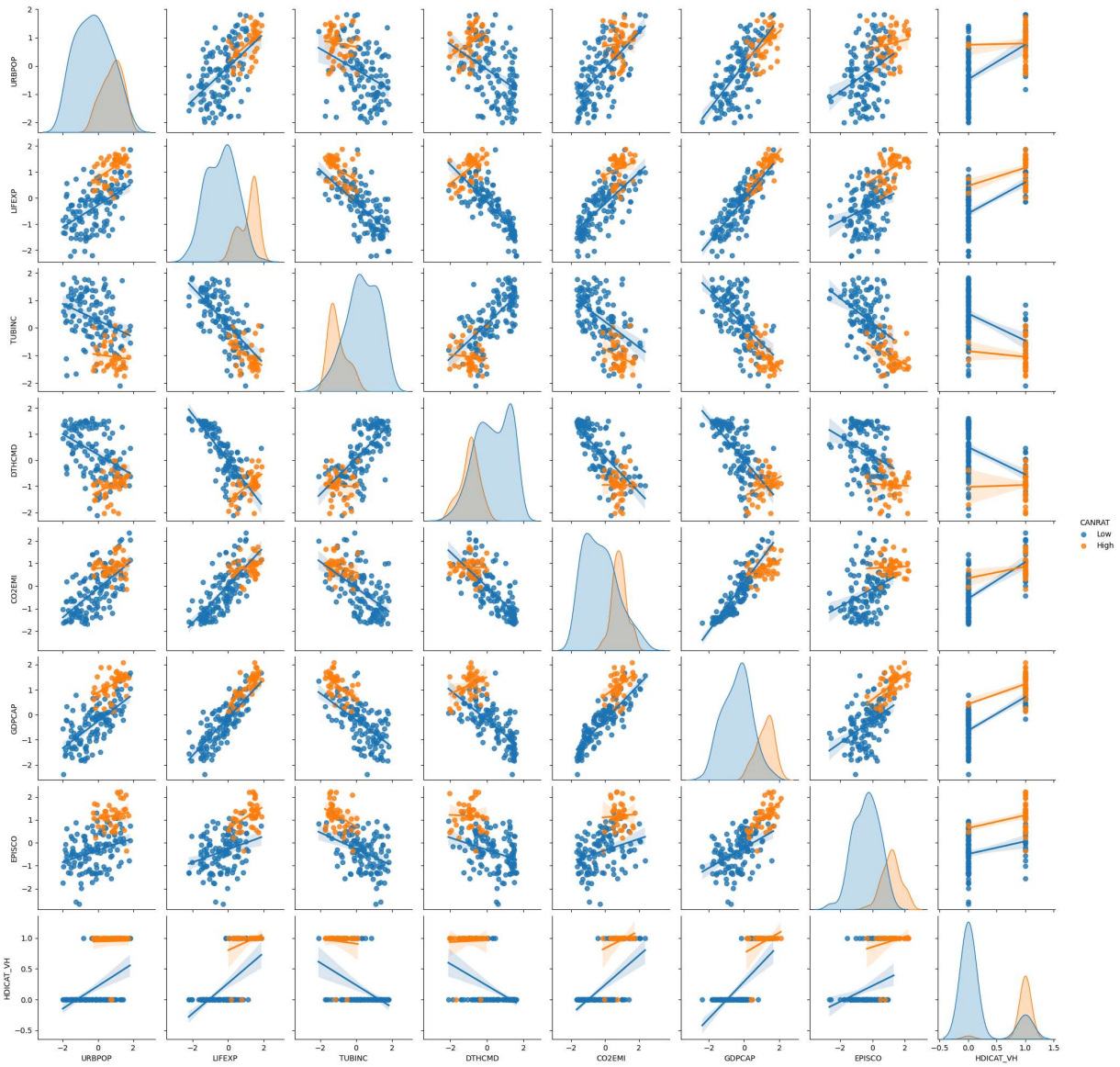
In [145…

```
###################################
# Listing the column names and data types
###################################
print('Column Names and Data Types:')
display(cancer_rate_premodelling.dtypes)
```

```
Column Names and Data Types:
URBPOP        float64
LIFEXP        float64
TUBINC        float64
DTHCMD        float64
CO2EMI        float64
GDPCAP        float64
EPISCO        float64
CANRAT       category
HDICAT_VH        bool
dtype: object
```

In [146…

```
###################################
# Gathering the pairplot for all variables
###################################
sns.pairplot(cancer_rate_premodelling,
             kind='reg',
             hue='CANRAT');
plt.show()
```



In [147…
```
###################################
# Separating the target
# and predictor columns
###################################
X = cancer_rate_premodelling.drop('CANRAT', axis = 1)
y = cancer_rate_premodelling['CANRAT'].cat.codes
```

In [148…
```
###################################
# Formulating the train and test data
# using a 70-30 ratio
###################################
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_s
```

In [149…
```
###################################
# Performing a general exploration of the train dataset
###################################
print('Dataset Dimensions: ')
display(X_train.shape)
```

Dataset Dimensions:

```
(114, 8)
```

In [150…
```
###################################
# Validating the class distribution of the train dataset
###################################
y_train.value_counts(normalize = True)
```

Out[150…
```
0    0.745614
1    0.254386
Name: proportion, dtype: float64
```

In [151…
```
###################################
# Performing a general exploration of the test dataset
###################################
print('Dataset Dimensions: ')
display(X_test.shape)
```

```
Dataset Dimensions:
(49, 8)
```

In [152…
```
###################################
# Validating the class distribution of the test dataset
###################################
y_test.value_counts(normalize = True)
```

Out[152…
```
0    0.755102
1    0.244898
Name: proportion, dtype: float64
```

In [153…
```
###################################
# Defining a function to compute
# model performance
###################################
def model_performance_evaluation(y_true, y_pred):
    metric_name = ['Accuracy','Precision','Recall','F1','AUROC']
    metric_value = [accuracy_score(y_true, y_pred),
                    precision_score(y_true, y_pred),
                    recall_score(y_true, y_pred),
                    f1_score(y_true, y_pred),
                    roc_auc_score(y_true, y_pred)]
    metric_summary = pd.DataFrame(zip(metric_name, metric_value),
                                  columns=['metric_name','metric_value'])
    return(metric_summary)
```

1.6.2 Logistic Regression

Logistic Regression models the relationship between the probability of an event (among two outcome levels) by having the log-odds of the event be a linear combination of a set of predictors weighted by their respective parameter estimates. The parameters are estimated via maximum likelihood estimation by testing different values through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter

estimates. Given the optimal parameters, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability.

1. The logistic regression model from the **sklearn.linear_model** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - C = inverse of regularization strength held constant at a value of 1
   - penalty = penalty norm made to vary between L1 and L2
   - solver = algorithm used in the optimization problem made to vary between Saga and Liblinear
   - class_weight = weights associated with classes held constant at a value of None
   - max_iter = maximum number of iterations taken for the solvers to converge held constant at a value of 500
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - C = 1
   - penalty = L1 norm
   - solver = Liblinear
   - class_weight = None
   - max_iter = 500
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9473
   - **Precision** = 0.8709
   - **Recall** = 0.9310
   - **F1 Score** = 0.9000
   - **AUROC** = 0.9419
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.8979
   - **Precision** = 0.8889
   - **Recall** = 0.6667
   - **F1 Score** = 0.7619
   - **AUROC** = 0.8198
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [154…

```
###################################
# Creating an instance of the
# Logistic Regression model
###################################
logistic_regression = LogisticRegression()

###################################
# Defining the hyperparameters for the
```

```
# Logistic Regression model
###################################
hyperparameter_grid = {
    'C': [1.0],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear','saga'],
    'class_weight': [None],
    'max_iter': [500],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Logistic Regression model
###################################
optimal_logistic_regression = GridSearchCV(estimator = logistic_regression,
                                           param_grid = hyperparameter_grid,
                                           n_jobs = -1,
                                           scoring='f1')

###################################
# Fitting the optimal Logistic Regression model
###################################
optimal_logistic_regression.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Logistic Regression model
###################################
optimal_logistic_regression.best_score_
optimal_logistic_regression.best_params_
```

Out[154…

```
{'C': 1.0,
 'class_weight': None,
 'max_iter': 500,
 'penalty': 'l1',
 'random_state': 88888888,
 'solver': 'liblinear'}
```

In [155…

```
###################################
# Evaluating the optimal Logistic Regression model
# on the train set
###################################
optimal_logistic_regression_y_hat_train = optimal_logistic_regression.predict(X_tra

###################################
# Gathering the model evaluation metrics
###################################
optimal_logistic_regression_performance_train = model_performance_evaluation(y_trai
optimal_logistic_regression_performance_train['model'] = ['optimal_logistic_regress
optimal_logistic_regression_performance_train['set'] = ['train'] * 5
print('Optimal Logistic Regression Model Performance on Train Data: ')
display(optimal_logistic_regression_performance_train)
```

Optimal Logistic Regression Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.947368	optimal_logistic_regression	train
1	Precision	0.870968	optimal_logistic_regression	train
2	Recall	0.931034	optimal_logistic_regression	train
3	F1	0.900000	optimal_logistic_regression	train
4	AUROC	0.941988	optimal_logistic_regression	train

In [156…

```
###################################
# Evaluating the optimal Logistic Regression model
# on the test set
###################################
optimal_logistic_regression_y_hat_test = optimal_logistic_regression.predict(X_test

###################################
# Gathering the model evaluation metrics
###################################
optimal_logistic_regression_performance_test = model_performance_evaluation(y_test,
optimal_logistic_regression_performance_test['model'] = ['optimal_logistic_regressi
optimal_logistic_regression_performance_test['set'] = ['test'] * 5
print('Optimal Logistic Regression Model Performance on Test Data: ')
display(optimal_logistic_regression_performance_test)
```

```
Optimal Logistic Regression Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	optimal_logistic_regression	test
1	Precision	0.888889	optimal_logistic_regression	test
2	Recall	0.666667	optimal_logistic_regression	test
3	F1	0.761905	optimal_logistic_regression	test
4	AUROC	0.819820	optimal_logistic_regression	test

1.6.3 Decision Trees

Decision trees create a model that predicts the class label of a sample based on input features. A decision tree consists of nodes that represent decisions or choices, edges which connect nodes and represent the possible outcomes of a decision and leaf (or terminal) nodes which represent the final decision or the predicted class label. The decision-making process involves feature selection (at each internal node, the algorithm decides which feature to split on based on a certain criterion including gini impurity or entropy), splitting criteria (the splitting criteria aim to find the feature and its corresponding threshold that best separates the data into different classes. The goal is to increase homogeneity within each resulting subset), recursive splitting (the process of feature selection and splitting continues recursively, creating a tree structure. The dataset is partitioned at each internal node based

on the chosen feature, and the process repeats for each subset) and stopping criteria (the recursion stops when a certain condition is met, known as a stopping criterion. Common stopping criteria include a maximum depth for the tree, a minimum number of samples required to split a node, or a minimum number of samples in a leaf node.)

1. The decision tree model from the **sklearn.tree** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
   - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
   - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
   - class_weight = weights associated with classes held constant at a value of None
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - criterion = Entropy
   - max_depth = 5
   - min_samples_leaf = 3
   - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9736
   - **Precision** = 1.0000
   - **Recall** = 0.8965
   - **F1 Score** = 0.9454
   - **AUROC** = 0.9482
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.8571
   - **Precision** = 0.8571
   - **Recall** = 0.5000
   - **F1 Score** = 0.6315
   - **AUROC** = 0.7364
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [157…

```
##################################
# Creating an instance of the
# Decision Tree model
##################################
decision_tree = DecisionTreeClassifier()

##################################
# Defining the hyperparameters for the
# Decision Tree model
```

```
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Decision Tree model
###################################
optimal_decision_tree = GridSearchCV(estimator = decision_tree,
                                     param_grid = hyperparameter_grid,
                                     n_jobs = -1,
                                     scoring='f1')

###################################
# Fitting the optimal Decision Tree model
###################################
optimal_decision_tree.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Decision Tree model
###################################
optimal_decision_tree.best_score_
optimal_decision_tree.best_params_
```

Out[157…

```
{'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 5,
 'min_samples_leaf': 3,
 'random_state': 88888888}
```

In [158…

```
###################################
# Evaluating the optimal decision tree model
# on the train set
###################################
optimal_decision_tree_y_hat_train = optimal_decision_tree.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
optimal_decision_tree_performance_train = model_performance_evaluation(y_train, opt
optimal_decision_tree_performance_train['model'] = ['optimal_decision_tree'] * 5
optimal_decision_tree_performance_train['set'] = ['train'] * 5
print('Optimal Decision Tree Model Performance on Train Data: ')
display(optimal_decision_tree_performance_train)
```

Optimal Decision Tree Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.973684	optimal_decision_tree	train
1	Precision	1.000000	optimal_decision_tree	train
2	Recall	0.896552	optimal_decision_tree	train
3	F1	0.945455	optimal_decision_tree	train
4	AUROC	0.948276	optimal_decision_tree	train

In [159…

```
###################################
# Evaluating the optimal decision tree model
# on the test set
###################################
optimal_decision_tree_y_hat_test = optimal_decision_tree.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
optimal_decision_tree_performance_test = model_performance_evaluation(y_test, optim
optimal_decision_tree_performance_test['model'] = ['optimal_decision_tree'] * 5
optimal_decision_tree_performance_test['set'] = ['test'] * 5
print('Optimal Decision Tree Model Performance on Test Data: ')
display(optimal_decision_tree_performance_test)
```

```
Optimal Decision Tree Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.857143	optimal_decision_tree	test
1	Precision	0.857143	optimal_decision_tree	test
2	Recall	0.500000	optimal_decision_tree	test
3	F1	0.631579	optimal_decision_tree	test
4	AUROC	0.736486	optimal_decision_tree	test

1.6.4 Random Forest

Random Forest is an ensemble learning method made up of a large set of small decision trees called estimators, with each producing its own prediction. The random forest model aggregates the predictions of the estimators to produce a more accurate prediction. The algorithm involves bootstrap aggregating (where smaller subsets of the training data are repeatedly subsampled with replacement), random subspacing (where a subset of features are sampled and used to train each individual estimator), estimator training (where unpruned decision trees are formulated for each estimator) and inference by aggregating the predictions of all estimators.

1. The random forest model from the **sklearn.ensemble** Python library API was implemented.
2. The model contains 5 hyperparameters:
    - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
    - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
    - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
    - n_estimators = number of trees in the forest made to vary between 100, 150 and 200
    - max_features = number of features to consider when looking for the best split made to vary between Sqrt and Log2 of n_estimators
    - class_weight = weights associated with classes held constant at a value of None
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - criterion = Gini
    - max_depth = 3
    - min_samples_leaf = 3
    - n_estimators = 100
    - max_features = Sqrt n_estimators
    - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9561
    - **Precision** = 0.9285
    - **Recall** = 0.8965
    - **F1 Score** = 0.9122
    - **AUROC** = 0.9365
6. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.8775
    - **Precision** = 0.8750
    - **Recall** = 0.5833
    - **F1 Score** = 0.7000
    - **AUROC** = 0.7781
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [160…

```
#################################
# Creating an instance of the
# Random Forest model
#################################
random_forest = RandomForestClassifier()
```

```
###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'n_estimators': [100,150,200],
    'max_features':['sqrt', 'log2'],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
optimal_random_forest = GridSearchCV(estimator = random_forest,
                                     param_grid = hyperparameter_grid,
                                     n_jobs = -1,
                                     scoring='f1')

###################################
# Fitting the optimal Random Forest model
###################################
optimal_random_forest.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Random Forest model
###################################
optimal_random_forest.best_score_
optimal_random_forest.best_params_
```

Out[160…

```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': 3,
 'max_features': 'sqrt',
 'min_samples_leaf': 3,
 'n_estimators': 100,
 'random_state': 88888888}
```

In [161…

```
###################################
# Evaluating the optimal Random Forest model
# on the train set
###################################
optimal_random_forest_y_hat_train = optimal_random_forest.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
optimal_random_forest_performance_train = model_performance_evaluation(y_train, opt
optimal_random_forest_performance_train['model'] = ['optimal_random_forest'] * 5
optimal_random_forest_performance_train['set'] = ['train'] * 5
print('Optimal Random Forest Model Performance on Train Data: ')
display(optimal_random_forest_performance_train)
```

Optimal Random Forest Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.956140	optimal_random_forest	train
1	Precision	0.928571	optimal_random_forest	train
2	Recall	0.896552	optimal_random_forest	train
3	F1	0.912281	optimal_random_forest	train
4	AUROC	0.936511	optimal_random_forest	train

In [162…

```
###################################
# Evaluating the optimal Random Forest model
# on the test set
###################################
optimal_random_forest_y_hat_test = optimal_random_forest.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
optimal_random_forest_performance_test = model_performance_evaluation(y_test, optim
optimal_random_forest_performance_test['model'] = ['optimal_random_forest'] * 5
optimal_random_forest_performance_test['set'] = ['test'] * 5
print('Optimal Random Forest Model Performance on Test Data: ')
display(optimal_random_forest_performance_test)
```

Optimal Random Forest Model Performance on Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.877551	optimal_random_forest	test
1	Precision	0.875000	optimal_random_forest	test
2	Recall	0.583333	optimal_random_forest	test
3	F1	0.700000	optimal_random_forest	test
4	AUROC	0.778153	optimal_random_forest	test

1.6.5 Support Vector Machine

Support Vector Machine plots each observation in an N-dimensional space corresponding to the number of features in the data set and finds a hyperplane that maximally separates the different classes by a maximally large margin (which is defined as the distance between the hyperplane and the closest data points from each class). The algorithm applies kernel transformation by mapping non-linearly separable data using the similarities between the points in a high-dimensional feature space for improved discrimination.

1. The support vector machine model from the **sklearn.svm** Python library API was implemented.

2. The model contains 5 hyperparameters:
    - C = inverse of regularization strength held constant at a value of 1
    - kernel = kernel type to be used in the algorithm made to vary between Linear, Poly, RBF and Sigmoid
    - class_weight = weights associated with classes held constant at a value of None
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - C = 1
    - kernel = Poly
    - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9473
    - **Precision** = 0.9600
    - **Recall** = 0.8275
    - **F1 Score** = 0.8888
    - **AUROC** = 0.9079
6. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.8571
    - **Precision** = 0.8571
    - **Recall** = 0.5000
    - **F1 Score** = 0.6315
    - **AUROC** = 0.7364
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [163…

```
###################################
# Creating an instance of the
# Support Vector Machine model
###################################
support_vector_machine = SVC()

###################################
# Defining the hyperparameters for the
# Support Vector Machine model
###################################
hyperparameter_grid = {
    'C': [1.0],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Support Vector Machine model
###################################
```

```
optimal_support_vector_machine = GridSearchCV(estimator = support_vector_machine,
                                              param_grid = hyperparameter_grid,
                                              n_jobs = -1,
                                              scoring='f1')

###################################
# Fitting the optimal Support Vector Machine model
###################################
optimal_support_vector_machine.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Support Vector Machine model
###################################
optimal_support_vector_machine.best_score_
optimal_support_vector_machine.best_params_
```

Out[163…

```
{'C': 1.0, 'class_weight': None, 'kernel': 'poly', 'random_state': 88888888}
```

In [164…

```
###################################
# Evaluating the optimal Support Vector Machine model
# on the train set
###################################
optimal_support_vector_machine_y_hat_train = optimal_support_vector_machine.predict

###################################
# Gathering the model evaluation metrics
###################################
optimal_support_vector_machine_performance_train = model_performance_evaluation(y_t
optimal_support_vector_machine_performance_train['model'] = ['optimal_support_vecto
optimal_support_vector_machine_performance_train['set'] = ['train'] * 5
print('Optimal Support Vector Machine Model Performance on Train Data: ')
display(optimal_support_vector_machine_performance_train)
```

```
Optimal Support Vector Machine Model Performance on Train Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.947368	optimal_support_vector_machine	train
1	Precision	0.960000	optimal_support_vector_machine	train
2	Recall	0.827586	optimal_support_vector_machine	train
3	F1	0.888889	optimal_support_vector_machine	train
4	AUROC	0.907911	optimal_support_vector_machine	train

In [165…

```
###################################
# Evaluating the optimal Support Vector Machine model
# on the test set
###################################
optimal_support_vector_machine_y_hat_test = optimal_support_vector_machine.predict(

###################################
# Gathering the model evaluation metrics
###################################
```

```
optimal_support_vector_machine_performance_test = model_performance_evaluation(y_te
optimal_support_vector_machine_performance_test['model'] = ['optimal_support_vector
optimal_support_vector_machine_performance_test['set'] = ['test'] * 5
print('Optimal Support Vector Machine Model Performance on Test Data: ')
display(optimal_support_vector_machine_performance_test)
```

```
Optimal Support Vector Machine Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.857143	optimal_support_vector_machine	test
1	Precision	0.857143	optimal_support_vector_machine	test
2	Recall	0.500000	optimal_support_vector_machine	test
3	F1	0.631579	optimal_support_vector_machine	test
4	AUROC	0.736486	optimal_support_vector_machine	test

1.7. Model Development With Class Weights

Class weights are used to assign different levels of importance to different classes when the distribution of instances across different classes in a classification problem is not equal. By assigning higher weights to the minority class, the model is encouraged to give more attention to correctly predicting instances from the minority class. Class weights are incorporated into the loss function during training. The loss for each instance is multiplied by its corresponding class weight. This means that misclassifying an instance from the minority class will have a greater impact on the overall loss than misclassifying an instance from the majority class. The use of class weights helps balance the influence of each class during training, mitigating the impact of class imbalance. It provides a way to focus the learning process on the classes that are underrepresented in the training data.

1.7.1 Premodelling Data Description

1. Among the 9 numeric variables determined to have a statistically significant difference between the means of the numeric measurements obtained from LOW and HIGH groups of the CANRAT target variable, only 7 were retained with absolute T-Test statistics greater than 5.
   - GDPCAP: T.Test.Statistic=-11.937, T.Test.PValue=0.000
   - EPISCO: T.Test.Statistic=-11.789, T.Test.PValue=0.000
   - LIFEXP: T.Test.Statistic=-10.979, T.Test.PValue=0.000
   - TUBINC: T.Test.Statistic=+9.609, T.Test.PValue=0.000
   - DTHCMD: T.Test.Statistic=+8.376, T.Test.PValue=0.000
   - CO2EMI: T.Test.Statistic=-7.031, T.Test.PValue=0.000
   - URBPOP: T.Test.Statistic=-6.541, T.Test.PValue=0.000
2. Among the 4 categorical predictors determined to have a statistically significant relationship difference between the categories of the categorical predictors and the

LOW and HIGH groups of the CANRAT target variable, only 1 was retained with absolute Chi-Square statistics greater than 15.

- HDICAT_VH: ChiSquare.Test.Statistic=76.764, ChiSquare.Test.PValue=0.000

3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.

In [166…
```
###################################
# Consolidating relevant numeric columns
# and encoded categorical columns
# after hypothesis testing
###################################
cancer_rate_premodelling = cancer_rate_preprocessed_all.drop(['AGRLND','POPDEN','GH
```

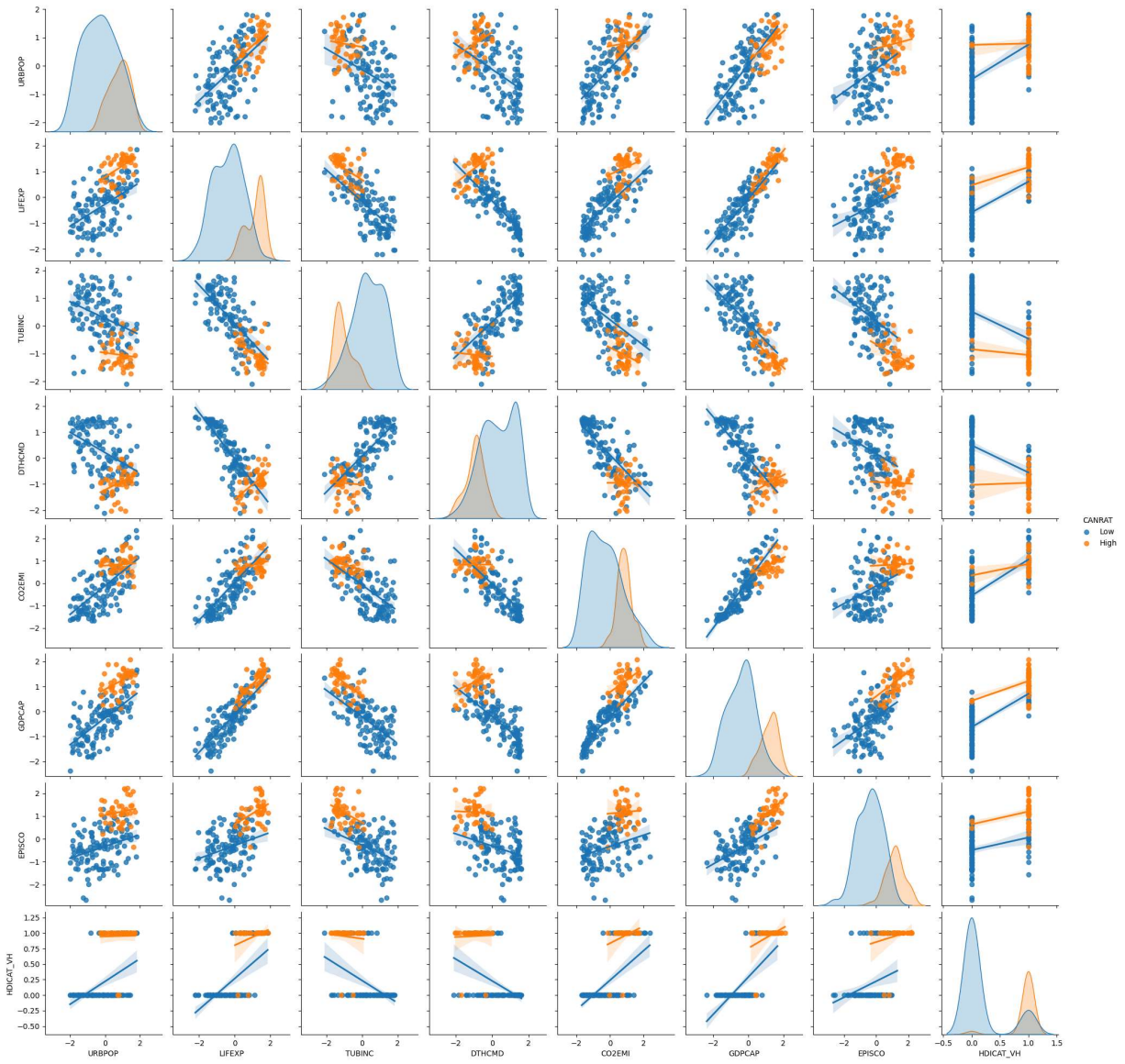
In [167…
```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_premodelling.shape)
```

```
Dataset Dimensions:
(163, 9)
```

In [168…
```
###################################
# Listing the column names and data types
###################################
print('Column Names and Data Types:')
display(cancer_rate_premodelling.dtypes)
```

```
Column Names and Data Types:
URBPOP        float64
LIFEXP        float64
TUBINC        float64
DTHCMD        float64
CO2EMI        float64
GDPCAP        float64
EPISCO        float64
CANRAT       category
HDICAT_VH        bool
dtype: object
```

In [169…
```
###################################
# Gathering the pairplot for all variables
###################################
sns.pairplot(cancer_rate_premodelling,
             kind='reg',
             hue='CANRAT');
plt.show()
```



In [170…
```
###################################
# Separating the target
# and predictor columns
###################################
X = cancer_rate_premodelling.drop('CANRAT', axis = 1)
y = cancer_rate_premodelling['CANRAT'].cat.codes
```

In [171…
```
###################################
# Formulating the train and test data
# using a 70-30 ratio
###################################
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_s
```

In [172…
```
###################################
# Performing a general exploration of the train dataset
###################################
print('Dataset Dimensions: ')
display(X_train.shape)
```

Dataset Dimensions:

```
(114, 8)
```

In [173…
```
###################################
# Validating the class distribution of the train dataset
###################################
y_train.value_counts(normalize = True)
```

Out[173…
```
0    0.745614
1    0.254386
Name: proportion, dtype: float64
```

In [174…
```
###################################
# Performing a general exploration of the test dataset
###################################
print('Dataset Dimensions: ')
display(X_test.shape)
```

```
Dataset Dimensions:
(49, 8)
```

In [175…
```
###################################
# Validating the class distribution of the test dataset
###################################
y_test.value_counts(normalize = True)
```

Out[175…
```
0    0.755102
1    0.244898
Name: proportion, dtype: float64
```

In [176…
```
###################################
# Defining a function to compute
# model performance
###################################
def model_performance_evaluation(y_true, y_pred):
    metric_name = ['Accuracy','Precision','Recall','F1','AUROC']
    metric_value = [accuracy_score(y_true, y_pred),
                    precision_score(y_true, y_pred),
                    recall_score(y_true, y_pred),
                    f1_score(y_true, y_pred),
                    roc_auc_score(y_true, y_pred)]
    metric_summary = pd.DataFrame(zip(metric_name, metric_value),
                                  columns=['metric_name','metric_value'])
    return(metric_summary)
```

1.7.2 Logistic Regression

Logistic Regression models the relationship between the probability of an event (among two outcome levels) by having the log-odds of the event be a linear combination of a set of predictors weighted by their respective parameter estimates. The parameters are estimated via maximum likelihood estimation by testing different values through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter

estimates. Given the optimal parameters, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability.

1. The logistic regression model from the **sklearn.linear_model** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - C = inverse of regularization strength held constant at a value of 1
   - penalty = penalty norm made to vary between L1 and L2
   - solver = algorithm used in the optimization problem made to vary between Saga and Liblinear
   - class_weight = weights associated with classes held constant at a value of 25-75 between classes 0 and 1
   - max_iter = maximum number of iterations taken for the solvers to converge held constant at a value of 500
3. The original data reflecting a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - C = 1
   - penalty = L1 norm
   - solver = Liblinear
   - class_weight = 25-75 between classes 0 and 1
   - max_iter = 500
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.8947
   - **Precision** = 0.7073
   - **Recall** = 1.0000
   - **F1 Score** = 0.8285
   - **AUROC** = 0.9294
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.9387
   - **Precision** = 0.8461
   - **Recall** = 0.9167
   - **F1 Score** = 0.8800
   - **AUROC** = 0.9313
7. Considerable difference in the apparent and independent test model performance observed, indicative of the presence of moderate model overfitting.

In [177…

```
###################################
# Creating an instance of the
# Logistic Regression model
###################################
logistic_regression = LogisticRegression()
```

```
###################################
# Defining the hyperparameters for the
# Logistic Regression model
###################################
hyperparameter_grid = {
    'C': [1.0],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear','saga'],
    'class_weight': [{0:0.25, 1:0.75}],
    'max_iter': [500],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Logistic Regression model
###################################
weighted_logistic_regression = GridSearchCV(estimator = logistic_regression,
                                             param_grid = hyperparameter_grid,
                                             n_jobs = -1,
                                             scoring='f1')

###################################
# Fitting the weighted Logistic Regression model
###################################
weighted_logistic_regression.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Logistic Regression model
###################################
weighted_logistic_regression.best_score_
weighted_logistic_regression.best_params_
```

Out[177…

```
{'C': 1.0,
 'class_weight': {0: 0.25, 1: 0.75},
 'max_iter': 500,
 'penalty': 'l2',
 'random_state': 88888888,
 'solver': 'liblinear'}
```

In [178…

```
###################################
# Evaluating the weighted Logistic Regression model
# on the train set
###################################
weighted_logistic_regression_y_hat_train = weighted_logistic_regression.predict(X_t

###################################
# Gathering the model evaluation metrics
###################################
weighted_logistic_regression_performance_train = model_performance_evaluation(y_tra
weighted_logistic_regression_performance_train['model'] = ['weighted_logistic_regre
weighted_logistic_regression_performance_train['set'] = ['train'] * 5
print('Weighted Logistic Regression Model Performance on Train Data: ')
display(weighted_logistic_regression_performance_train)
```

Weighted Logistic Regression Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.894737	weighted_logistic_regression	train
1	Precision	0.707317	weighted_logistic_regression	train
2	Recall	1.000000	weighted_logistic_regression	train
3	F1	0.828571	weighted_logistic_regression	train
4	AUROC	0.929412	weighted_logistic_regression	train

In [179…

```
###################################
# Evaluating the weighted Logistic Regression model
# on the test set
###################################
weighted_logistic_regression_y_hat_test = weighted_logistic_regression.predict(X_te

###################################
# Gathering the model evaluation metrics
###################################
weighted_logistic_regression_performance_test = model_performance_evaluation(y_test
weighted_logistic_regression_performance_test['model'] = ['weighted_logistic_regres
weighted_logistic_regression_performance_test['set'] = ['test'] * 5
print('Weighted Logistic Regression Model Performance on Test Data: ')
display(weighted_logistic_regression_performance_test)
```

```
Weighted Logistic Regression Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.938776	weighted_logistic_regression	test
1	Precision	0.846154	weighted_logistic_regression	test
2	Recall	0.916667	weighted_logistic_regression	test
3	F1	0.880000	weighted_logistic_regression	test
4	AUROC	0.931306	weighted_logistic_regression	test

1.7.3 Decision Trees

Decision trees create a model that predicts the class label of a sample based on input features. A decision tree consists of nodes that represent decisions or choices, edges which connect nodes and represent the possible outcomes of a decision and leaf (or terminal) nodes which represent the final decision or the predicted class label. The decision-making process involves feature selection (at each internal node, the algorithm decides which feature to split on based on a certain criterion including gini impurity or entropy), splitting criteria (the splitting criteria aim to find the feature and its corresponding threshold that best separates the data into different classes. The goal is to increase homogeneity within each resulting subset), recursive splitting (the process of feature selection and splitting continues recursively, creating a tree structure. The dataset is partitioned at each internal node based

on the chosen feature, and the process repeats for each subset) and stopping criteria (the recursion stops when a certain condition is met, known as a stopping criterion. Common stopping criteria include a maximum depth for the tree, a minimum number of samples required to split a node, or a minimum number of samples in a leaf node.)

1. The decision tree model from the **sklearn.tree** Python library API was implemented.
2. The model contains 5 hyperparameters:
    - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
    - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
    - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
    - class_weight = weights associated with classes held constant at a value of 25-75 between classes 0 and 1
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - criterion = Gini
    - max_depth = 3
    - min_samples_leaf = 3
    - class_weight = 25-75 between classes 0 and 1
5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9736
    - **Precision** = 1.0000
    - **Recall** = 0.8965
    - **F1 Score** = 0.9454
    - **AUROC** = 0.9482
6. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.8571
    - **Precision** = 0.8571
    - **Recall** = 0.5000
    - **F1 Score** = 0.6315
    - **AUROC** = 0.7364
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [180…

```
##################################
# Creating an instance of the
# Decision Tree model
##################################
decision_tree = DecisionTreeClassifier()

##################################
# Defining the hyperparameters for the
```

```
# Decision Tree model
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'class_weight': [{0:0.25, 1:0.75}],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Decision Tree model
###################################
weighted_decision_tree = GridSearchCV(estimator = decision_tree,
                                      param_grid = hyperparameter_grid,
                                      n_jobs = -1,
                                      scoring='f1')

###################################
# Fitting the weighted Decision Tree model
###################################
weighted_decision_tree.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Decision Tree model
###################################
weighted_decision_tree.best_score_
weighted_decision_tree.best_params_
```

Out[180…

```
{'class_weight': {0: 0.25, 1: 0.75},
 'criterion': 'gini',
 'max_depth': 3,
 'min_samples_leaf': 3,
 'random_state': 88888888}
```

In [181…

```
###################################
# Evaluating the weighted decision tree model
# on the train set
###################################
weighted_decision_tree_y_hat_train = weighted_decision_tree.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
weighted_decision_tree_performance_train = model_performance_evaluation(y_train, we
weighted_decision_tree_performance_train['model'] = ['weighted_decision_tree'] * 5
weighted_decision_tree_performance_train['set'] = ['train'] * 5
print('Weighted Decision Tree Model Performance on Train Data: ')
display(weighted_decision_tree_performance_train)
```

Weighted Decision Tree Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.956140	weighted_decision_tree	train
1	Precision	0.852941	weighted_decision_tree	train
2	Recall	1.000000	weighted_decision_tree	train
3	F1	0.920635	weighted_decision_tree	train
4	AUROC	0.970588	weighted_decision_tree	train

In [182…

```
###################################
# Evaluating the weighted decision tree model
# on the test set
###################################
weighted_decision_tree_y_hat_test = weighted_decision_tree.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
weighted_decision_tree_performance_test = model_performance_evaluation(y_test, weig
weighted_decision_tree_performance_test['model'] = ['weighted_decision_tree'] * 5
weighted_decision_tree_performance_test['set'] = ['test'] * 5
print('Weighted Decision Tree Model Performance on Test Data: ')
display(weighted_decision_tree_performance_test)
```

```
Weighted Decision Tree Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	weighted_decision_tree	test
1	Precision	0.769231	weighted_decision_tree	test
2	Recall	0.833333	weighted_decision_tree	test
3	F1	0.800000	weighted_decision_tree	test
4	AUROC	0.876126	weighted_decision_tree	test

1.7.4 Random Forest

Random Forest is an ensemble learning method made up of a large set of small decision trees called estimators, with each producing its own prediction. The random forest model aggregates the predictions of the estimators to produce a more accurate prediction. The algorithm involves bootstrap aggregating (where smaller subsets of the training data are repeatedly subsampled with replacement), random subspacing (where a subset of features are sampled and used to train each individual estimator), estimator training (where unpruned decision trees are formulated for each estimator) and inference by aggregating the predictions of all estimators.

1. The random forest model from the **sklearn.ensemble** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
   - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
   - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
   - n_estimators = number of trees in the forest made to vary between 100, 150 and 200
   - max_features = number of features to consider when looking for the best split made to vary between Sqrt and Log2 of n_estimators
   - class_weight = weights associated with classes held constant at a value of 25-75 between classes 0 and 1
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - criterion = Gini
   - max_depth = 5
   - min_samples_leaf = 3
   - n_estimators = 100
   - max_features = Sqrt n_estimators
   - class_weight = 25-75 between classes 0 and 1
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9736
   - **Precision** = 0.9062
   - **Recall** = 1.0000
   - **F1 Score** = 0.9508
   - **AUROC** = 0.9823
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.8979
   - **Precision** = 0.8888
   - **Recall** = 0.6666
   - **F1 Score** = 0.7619
   - **AUROC** = 0.8198
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [183…

```
#################################
# Creating an instance of the
# Random Forest model
#################################
random_forest = RandomForestClassifier()
```

```
###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'n_estimators': [100,150,200],
    'max_features':['sqrt', 'log2'],
    'class_weight': [{0:0.25, 1:0.75}],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
weighted_random_forest = GridSearchCV(estimator = random_forest,
                                      param_grid = hyperparameter_grid,
                                      n_jobs = -1,
                                      scoring='f1')

###################################
# Fitting the weighted Random Forest model
###################################
weighted_random_forest.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Random Forest model
###################################
weighted_random_forest.best_score_
weighted_random_forest.best_params_
```

Out[183…

```
{'class_weight': {0: 0.25, 1: 0.75},
 'criterion': 'gini',
 'max_depth': 5,
 'max_features': 'sqrt',
 'min_samples_leaf': 3,
 'n_estimators': 100,
 'random_state': 88888888}
```

In [184…

```
###################################
# Evaluating the weighted Random Forest model
# on the train set
###################################
weighted_random_forest_y_hat_train = weighted_random_forest.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
weighted_random_forest_performance_train = model_performance_evaluation(y_train, w
weighted_random_forest_performance_train['model'] = ['weighted_random_forest'] * 5
weighted_random_forest_performance_train['set'] = ['train'] * 5
```

```
print('Weighted Random Forest Model Performance on Train Data: ')
display(weighted_random_forest_performance_train)
```

```
Weighted Random Forest Model Performance on Train Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.973684	weighted_random_forest	train
1	Precision	0.906250	weighted_random_forest	train
2	Recall	1.000000	weighted_random_forest	train
3	F1	0.950820	weighted_random_forest	train
4	AUROC	0.982353	weighted_random_forest	train

In [185…

```
###################################
# Evaluating the weighted Random Forest model
# on the test set
###################################
weighted_random_forest_y_hat_test = weighted_random_forest.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
weighted_random_forest_performance_test = model_performance_evaluation(y_test, wei
weighted_random_forest_performance_test['model'] = ['weighted_random_forest'] * 5
weighted_random_forest_performance_test['set'] = ['test'] * 5
print('Weighted Random Forest Model Performance on Test Data: ')
display(weighted_random_forest_performance_test)
```

```
Weighted Random Forest Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	weighted_random_forest	test
1	Precision	0.888889	weighted_random_forest	test
2	Recall	0.666667	weighted_random_forest	test
3	F1	0.761905	weighted_random_forest	test
4	AUROC	0.819820	weighted_random_forest	test

1.7.5 Support Vector Machine

Support Vector Machine plots each observation in an N-dimensional space corresponding to the number of features in the data set and finds a hyperplane that maximally separates the different classes by a maximally large margin (which is defined as the distance between the hyperplane and the closest data points from each class). The algorithm applies kernel transformation by mapping non-linearly separable data using the similarities between the points in a high-dimensional feature space for improved discrimination.

1. The support vector machine model from the **sklearn.svm** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - C = inverse of regularization strength held constant at a value of 1
   - kernel = kernel type to be used in the algorithm made to vary between Linear, Poly, RBF and Sigmoid
   - class_weight = weights associated with classes held constant at a value of 25-75 between classes 0 and 1
3. The original data which reflect a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - C = 1
   - kernel = Poly
   - class_weight = 25-75 between classes 0 and 1
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9649
   - **Precision** = 0.9629
   - **Recall** = 0.8965
   - **F1 Score** = 0.9285
   - **AUROC** = 0.9423
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.8775
   - **Precision** = 0.8750
   - **Recall** = 0.5833
   - **F1 Score** = 0.7000
   - **AUROC** = 0.7781
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [186…

```
###################################
# Creating an instance of the
# Support Vector Machine model
###################################
support_vector_machine = SVC()

###################################
# Defining the hyperparameters for the
# Support Vector Machine model
###################################
hyperparameter_grid = {
    'C': [1.0],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'class_weight': [{0:0.25, 1:0.75}],
    'random_state': [88888888]}
```

```
###################################
# Defining the hyperparameters for the
# Support Vector Machine model
###################################
weighted_support_vector_machine = GridSearchCV(estimator = support_vector_machine,
                                               param_grid = hyperparameter_grid,
                                               n_jobs = -1,
                                               scoring='f1')

###################################
# Fitting the weighted Support Vector Machine model
###################################
weighted_support_vector_machine.fit(X_train, y_train)

###################################
# Determining the optimal hyperparameter
# for the Support Vector Machine model
###################################
weighted_support_vector_machine.best_score_
weighted_support_vector_machine.best_params_
```

Out[186…

```
{'C': 1.0,
 'class_weight': {0: 0.25, 1: 0.75},
 'kernel': 'poly',
 'random_state': 88888888}
```

In [187…

```
###################################
# Evaluating the weighted Support Vector Machine model
# on the train set
###################################
weighted_support_vector_machine_y_hat_train = weighted_support_vector_machine.predict(

###################################
# Gathering the model evaluation metrics
###################################
weighted_support_vector_machine_performance_train = model_performance_evaluation(y
weighted_support_vector_machine_performance_train['model'] = ['weighted_support_ve
weighted_support_vector_machine_performance_train['set'] = ['train'] * 5
print('Weighted Support Vector Machine Model Performance on Train Data: ')
display(weighted_support_vector_machine_performance_train)
```

Weighted Support Vector Machine Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.964912	weighted_support_vector_machine	train
1	Precision	0.962963	weighted_support_vector_machine	train
2	Recall	0.896552	weighted_support_vector_machine	train
3	F1	0.928571	weighted_support_vector_machine	train
4	AUROC	0.942394	weighted_support_vector_machine	train

In [188…

```
###################################
# Evaluating the weighted Support Vector Machine model
```

```
# on the test set
#################################
weighted_support_vector_machine_y_hat_test = weighted_support_vector_machine.predi

#################################
# Gathering the model evaluation metrics
#################################
weighted_support_vector_machine_performance_test = model_performance_evaluation(y_
weighted_support_vector_machine_performance_test['model'] = ['weighted_support_vec
weighted_support_vector_machine_performance_test['set'] = ['test'] * 5
print('Weighted Support Vector Machine Model Performance on Test Data: ')
display(weighted_support_vector_machine_performance_test)
```

```
Weighted Support Vector Machine Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.877551	weighted_support_vector_machine	test
1	Precision	0.875000	weighted_support_vector_machine	test
2	Recall	0.583333	weighted_support_vector_machine	test
3	F1	0.700000	weighted_support_vector_machine	test
4	AUROC	0.778153	weighted_support_vector_machine	test

1.8. Model Development With SMOTE Upsampling

Synthetic Minority Oversampling Technique is specifically designed to increase the representation of the minority class by generating new minority instances between existing instances. The new instances created are not just the copy of existing minority cases, instead for each minority class instance, the algorithm generates synthetic examples by creating linear combinations of the feature vectors between that instance and its k nearest neighbors. The synthetic samples are placed along the line segments connecting the original instance to its neighbors.

1.8.1 Premodelling Data Description

1. Among the 9 numeric variables determined to have a statistically significant difference between the means of the numeric measurements obtained from LOW and HIGH groups of the CANRAT target variable, only 7 were retained with absolute T-Test statistics greater than 5.
    - GDPCAP: T.Test.Statistic=-11.937, T.Test.PValue=0.000
    - EPISCO: T.Test.Statistic=-11.789, T.Test.PValue=0.000
    - LIFEXP: T.Test.Statistic=-10.979, T.Test.PValue=0.000
    - TUBINC: T.Test.Statistic=+9.609, T.Test.PValue=0.000
    - DTHCMD: T.Test.Statistic=+8.376, T.Test.PValue=0.000
    - CO2EMI: T.Test.Statistic=-7.031, T.Test.PValue=0.000

- URBPOP: T.Test.Statistic=-6.541, T.Test.PValue=0.000

2. Among the 4 categorical predictors determined to have a statistically significant relationship difference between the categories of the categorical predictors and the LOW and HIGH groups of the CANRAT target variable, only 1 was retained with absolute Chi-Square statistics greater than 15.
   - HDICAT_VH: ChiSquare.Test.Statistic=76.764, ChiSquare.Test.PValue=0.000
3. The SMOTE algorithm from the **imblearn.over_sampling** Python library API was implemented. The extended model training data by upsampling the minority HIGH CANRAT category applying SMOTE was used.

In [189…
```
###################################
# Consolidating relevant numeric columns
# and encoded categorical columns
# after hypothesis testing
###################################
cancer_rate_premodelling = cancer_rate_preprocessed_all.drop(['AGRLND','POPDEN','C
```

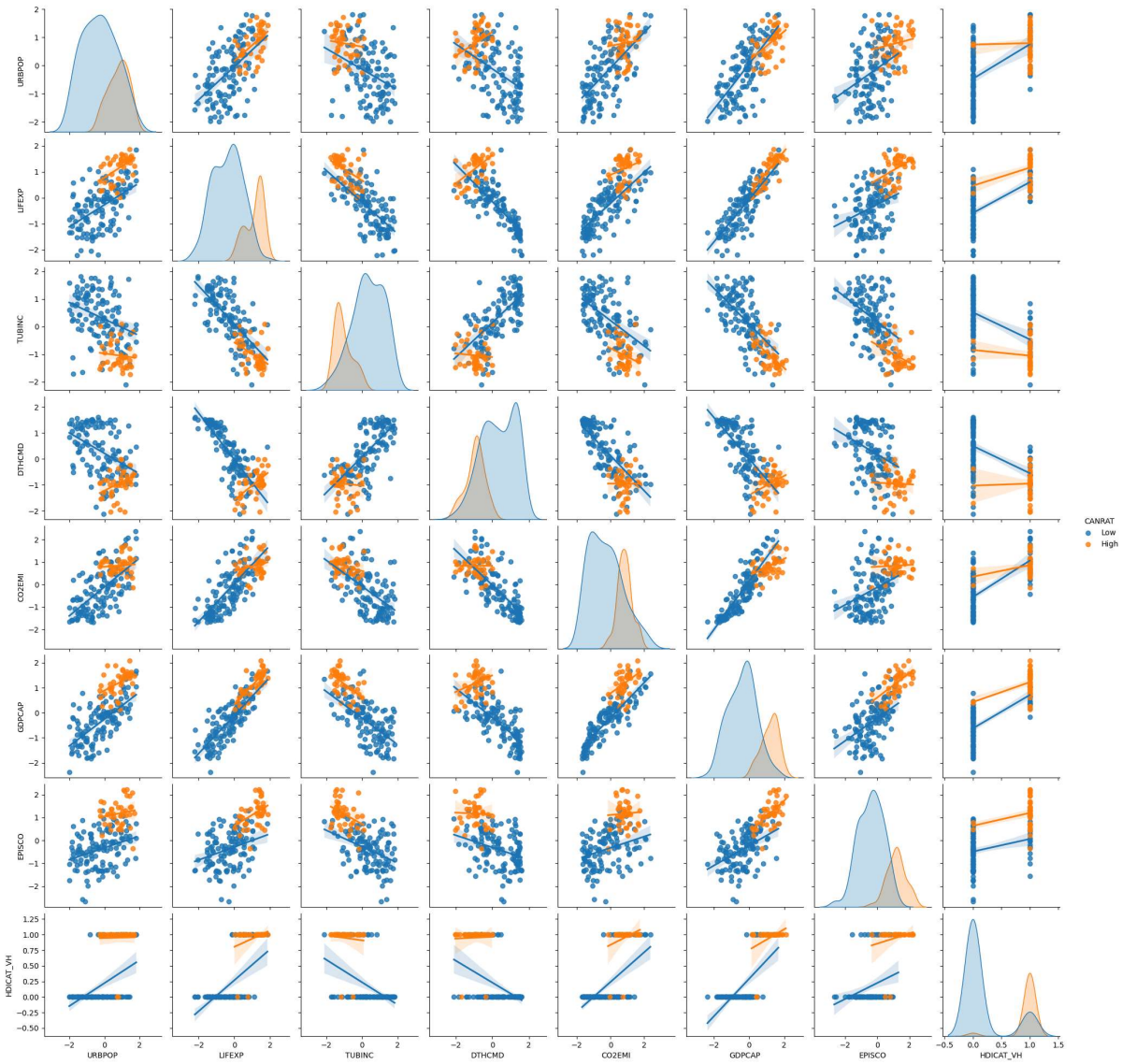
In [190…
```
###################################
# Performing a general exploration of the filtered dataset
###################################
print('Dataset Dimensions: ')
display(cancer_rate_premodelling.shape)
```

```
Dataset Dimensions:
(163, 9)
```

In [191…
```
###################################
# Listing the column names and data types
###################################
print('Column Names and Data Types:')
display(cancer_rate_premodelling.dtypes)
```

```
Column Names and Data Types:
URBPOP        float64
LIFEXP        float64
TUBINC        float64
DTHCMD        float64
CO2EMI        float64
GDPCAP        float64
EPISCO        float64
CANRAT       category
HDICAT_VH        bool
dtype: object
```

In [192…
```
###################################
# Gathering the pairplot for all variables
###################################
sns.pairplot(cancer_rate_premodelling,
             kind='reg',
             hue='CANRAT');
plt.show()
```



In [193…
```
###################################
# Separating the target
# and predictor columns
###################################
X = cancer_rate_premodelling.drop('CANRAT', axis = 1)
y = cancer_rate_premodelling['CANRAT'].cat.codes
```

In [194…
```
###################################
# Formulating the train and test data
# using a 70-30 ratio
###################################
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_
```

In [195…
```
###################################
# Performing a general exploration of the train dataset
###################################
print('Dataset Dimensions: ')
display(X_train.shape)
```

```
Dataset Dimensions:
(114, 8)
```

In [196…
```
###################################
# Validating the class distribution of the train dataset
###################################
y_train.value_counts(normalize = True)
```

Out[196…
```
0    0.745614
1    0.254386
Name: proportion, dtype: float64
```

In [197…
```
###################################
# Initiating an oversampling instance
# on the train data using
# Synthetic Minority Oversampling Technique
###################################
smote = SMOTE(random_state = 88888888)
X_train_smote, y_train_smote = smote.fit_resample(X_train,y_train)
```

In [198…
```
###################################
# Performing a general exploration of the overampled train dataset
###################################
print('Dataset Dimensions: ')
display(X_train_smote.shape)
```

```
Dataset Dimensions:
(170, 8)
```

In [199…
```
###################################
# Validating the class distribution of the overampled train dataset
###################################
y_train_smote.value_counts(normalize = True)
```

Out[199…
```
0    0.5
1    0.5
Name: proportion, dtype: float64
```

In [200…
```
###################################
# Performing a general exploration of the test dataset
###################################
print('Dataset Dimensions: ')
display(X_test.shape)
```

```
Dataset Dimensions:
(49, 8)
```

In [201…
```
###################################
# Validating the class distribution of the test dataset
###################################
y_test.value_counts(normalize = True)
```

Out[201…
```
0    0.755102
1    0.244898
Name: proportion, dtype: float64
```

In [202…
```
###################################
# Defining a function to compute
# model performance
```

```
################################
def model_performance_evaluation(y_true, y_pred):
    metric_name = ['Accuracy','Precision','Recall','F1','AUROC']
    metric_value = [accuracy_score(y_true, y_pred),
                    precision_score(y_true, y_pred),
                    recall_score(y_true, y_pred),
                    f1_score(y_true, y_pred),
                    roc_auc_score(y_true, y_pred)]
    metric_summary = pd.DataFrame(zip(metric_name, metric_value),
                                  columns=['metric_name','metric_value'])
    return(metric_summary)
```

1.8.2 Logistic Regression

Logistic Regression models the relationship between the probability of an event (among two outcome levels) by having the log-odds of the event be a linear combination of a set of predictors weighted by their respective parameter estimates. The parameters are estimated via maximum likelihood estimation by testing different values through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimates. Given the optimal parameters, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability.

1. The logistic regression model from the **sklearn.linear_model** Python library API was implemented.
2. The model contains 5 hyperparameters:
    - C = inverse of regularization strength held constant at a value of 1
    - penalty = penalty norm made to vary between L1 and L2
    - solver = algorithm used in the optimization problem made to vary between Saga and Liblinear
    - class_weight = weights associated with classes held constant at a value of None
    - max_iter = maximum number of iterations taken for the solvers to converge held constant at a value of 500
3. The extended model training data by upsampling the minority HIGH CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - C = 1
    - penalty = L1 norm
    - solver = Saga
    - class_weight = None
    - max_iter = 500
5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9649
    - **Precision** = 0.9032

- **Recall** = 0.9655
- **F1 Score** = 0.9333
- **AUROC** = 0.9651

6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.9183
   - **Precision** = 0.9000
   - **Recall** = 0.7500
   - **F1 Score** = 0.8181
   - **AUROC** = 0.8614
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [203…

```
###################################
# Creating an instance of the
# Logistic Regression model
###################################
logistic_regression = LogisticRegression()

###################################
# Defining the hyperparameters for the
# Logistic Regression model
###################################
hyperparameter_grid = {
    'C': [1.0],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear','saga'],
    'class_weight': [None],
    'max_iter': [500],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Logistic Regression model
###################################
upsampled_logistic_regression = GridSearchCV(estimator = logistic_regression,
                                              param_grid = hyperparameter_grid,
                                              n_jobs = -1,
                                              scoring='f1')

###################################
# Fitting the upsampled Logistic Regression model
###################################
upsampled_logistic_regression.fit(X_train_smote, y_train_smote)

###################################
# Determining the optimal hyperparameter
# for the Logistic Regression model
###################################
upsampled_logistic_regression.best_score_
upsampled_logistic_regression.best_params_
```

Out[203…
```
{'C': 1.0,
 'class_weight': None,
 'max_iter': 500,
 'penalty': 'l1',
 'random_state': 88888888,
 'solver': 'saga'}
```

In [204…
```
###################################
# Evaluating the upsampled Logistic Regression model
# on the train set
###################################
upsampled_logistic_regression_y_hat_train = upsampled_logistic_regression.predict(

###################################
# Gathering the model evaluation metrics
###################################
upsampled_logistic_regression_performance_train = model_performance_evaluation(y_t
upsampled_logistic_regression_performance_train['model'] = ['upsampled_logistic_re
upsampled_logistic_regression_performance_train['set'] = ['train'] * 5
print('Upsampled Logistic Regression Model Performance on Train Data: ')
display(upsampled_logistic_regression_performance_train)
```

Upsampled Logistic Regression Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.964912	upsampled_logistic_regression	train
1	Precision	0.903226	upsampled_logistic_regression	train
2	Recall	0.965517	upsampled_logistic_regression	train
3	F1	0.933333	upsampled_logistic_regression	train
4	AUROC	0.965112	upsampled_logistic_regression	train

In [205…
```
###################################
# Evaluating the upsampled Logistic Regression model
# on the test set
###################################
upsampled_logistic_regression_y_hat_test = upsampled_logistic_regression.predict()

###################################
# Gathering the model evaluation metrics
###################################
upsampled_logistic_regression_performance_test = model_performance_evaluation(y_te
upsampled_logistic_regression_performance_test['model'] = ['upsampled_logistic_reg
upsampled_logistic_regression_performance_test['set'] = ['test'] * 5
print('Upsampled Logistic Regression Model Performance on Test Data: ')
display(upsampled_logistic_regression_performance_test)
```

Upsampled Logistic Regression Model Performance on Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.918367	upsampled_logistic_regression	test
1	Precision	0.900000	upsampled_logistic_regression	test
2	Recall	0.750000	upsampled_logistic_regression	test
3	F1	0.818182	upsampled_logistic_regression	test
4	AUROC	0.861486	upsampled_logistic_regression	test

1.8.3 Decision Trees

Decision trees create a model that predicts the class label of a sample based on input features. A decision tree consists of nodes that represent decisions or choices, edges which connect nodes and represent the possible outcomes of a decision and leaf (or terminal) nodes which represent the final decision or the predicted class label. The decision-making process involves feature selection (at each internal node, the algorithm decides which feature to split on based on a certain criterion including gini impurity or entropy), splitting criteria (the splitting criteria aim to find the feature and its corresponding threshold that best separates the data into different classes. The goal is to increase homogeneity within each resulting subset), recursive splitting (the process of feature selection and splitting continues recursively, creating a tree structure. The dataset is partitioned at each internal node based on the chosen feature, and the process repeats for each subset) and stopping criteria (the recursion stops when a certain condition is met, known as a stopping criterion. Common stopping criteria include a maximum depth for the tree, a minimum number of samples required to split a node, or a minimum number of samples in a leaf node.

1. The decision tree model from the **sklearn.tree** Python library API was implemented.
2. The model contains 5 hyperparameters:
    - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
    - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
    - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
    - class_weight = weights associated with classes held constant at a value of None
3. The extended model training data by upsampling the minority HIGH CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - criterion = Entropy
    - max_depth = 3
    - min_samples_leaf = 5
    - class_weight = None

5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9210
    - **Precision** = 0.7631
    - **Recall** = 1.0000
    - **F1 Score** = 0.8656
    - **AUROC** = 0.9470
6. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.8979
    - **Precision** = 0.7692
    - **Recall** = 0.8333
    - **F1 Score** = 0.8000
    - **AUROC** = 0.8761
7. Considerable difference in the apparent and independent test model performance observed, indicative of the presence of moderate model overfitting.

In [206…

```
###################################
# Creating an instance of the
# Decision Tree model
###################################
decision_tree = DecisionTreeClassifier()

###################################
# Defining the hyperparameters for the
# Decision Tree model
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Decision Tree model
###################################
upsampled_decision_tree = GridSearchCV(estimator = decision_tree,
                                       param_grid = hyperparameter_grid,
                                       n_jobs = -1,
                                       scoring='f1')

###################################
# Fitting the upsampled Decision Tree model
###################################
upsampled_decision_tree.fit(X_train_smote, y_train_smote)

###################################
# Determining the optimal hyperparameter
# for the Decision Tree model
###################################
```

```
upsampled_decision_tree.best_score_
upsampled_decision_tree.best_params_
```

Out[206…

```
{'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_leaf': 5,
 'random_state': 88888888}
```

In [207…

```
###################################
# Evaluating the upsampled Decision Tree model
# on the train set
###################################
upsampled_decision_tree_y_hat_train = upsampled_decision_tree.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
upsampled_decision_tree_performance_train = model_performance_evaluation(y_train,
upsampled_decision_tree_performance_train['model'] = ['upsampled_decision_tree'] *
upsampled_decision_tree_performance_train['set'] = ['train'] * 5
print('Upsampled Decision Tree Model Performance on Train Data: ')
display(upsampled_decision_tree_performance_train)
```

Upsampled Decision Tree Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.921053	upsampled_decision_tree	train
1	Precision	0.763158	upsampled_decision_tree	train
2	Recall	1.000000	upsampled_decision_tree	train
3	F1	0.865672	upsampled_decision_tree	train
4	AUROC	0.947059	upsampled_decision_tree	train

In [208…

```
###################################
# Evaluating the upsampled Decision Tree model
# on the test set
###################################
upsampled_decision_tree_y_hat_test = upsampled_decision_tree.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
upsampled_decision_tree_performance_test = model_performance_evaluation(y_test, up
upsampled_decision_tree_performance_test['model'] = ['upsampled_decision_tree'] *
upsampled_decision_tree_performance_test['set'] = ['test'] * 5
print('Upsampled Decision Tree Model Performance on Test Data: ')
display(upsampled_decision_tree_performance_test)
```

Upsampled Decision Tree Model Performance on Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.897959	upsampled_decision_tree	test
1	Precision	0.769231	upsampled_decision_tree	test
2	Recall	0.833333	upsampled_decision_tree	test
3	F1	0.800000	upsampled_decision_tree	test
4	AUROC	0.876126	upsampled_decision_tree	test

1.8.4 Random Forest

Random Forest is an ensemble learning method made up of a large set of small decision trees called estimators, with each producing its own prediction. The random forest model aggregates the predictions of the estimators to produce a more accurate prediction. The algorithm involves bootstrap aggregating (where smaller subsets of the training data are repeatedly subsampled with replacement), random subspacing (where a subset of features are sampled and used to train each individual estimator), estimator training (where unpruned decision trees are formulated for each estimator) and inference by aggregating the predictions of all estimators.

1. The random forest model from the **sklearn.ensemble** Python library API was implemented.
2. The model contains 5 hyperparameters:
    - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
    - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
    - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
    - n_estimators = number of trees in the forest made to vary between 100, 150 and 200
    - max_features = number of features to consider when looking for the best split made to vary between Sqrt and Log2 of n_estimators
    - class_weight = weights associated with classes held constant at a value of None
3. The extended model training data by upsampling the minority HIGH CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - criterion = Entropy
    - max_depth = 7
    - min_samples_leaf = 3
    - n_estimators = 100
    - max_features = Sqrt n_estimators

- class_weight = None

5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9912
   - **Precision** = 0.9666
   - **Recall** = 1.0000
   - **F1 Score** = 0.9830
   - **AUROC** = 0.9941
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.9183
   - **Precision** = 0.9000
   - **Recall** = 0.7500
   - **F1 Score** = 0.8181
   - **AUROC** = 0.8614
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [209…

```
###################################
# Creating an instance of the
# Random Forest model
###################################
random_forest = RandomForestClassifier()

###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'n_estimators': [100,150,200],
    'max_features':['sqrt', 'log2'],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
upsampled_random_forest = GridSearchCV(estimator = random_forest,
                                       param_grid = hyperparameter_grid,
                                       n_jobs = -1,
                                       scoring='f1')

###################################
# Fitting the upsampled Random Forest model
###################################
upsampled_random_forest.fit(X_train_smote, y_train_smote)

###################################
# Determining the optimal hyperparameter
```

```
# for the Random Forest model
###################################
upsampled_random_forest.best_score_
upsampled_random_forest.best_params_
```

Out[209…

```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': 3,
 'max_features': 'sqrt',
 'min_samples_leaf': 3,
 'n_estimators': 150,
 'random_state': 88888888}
```

In [210…

```
###################################
# Evaluating the upsampled Random Forest model
# on the train set
###################################
upsampled_random_forest_y_hat_train = upsampled_random_forest.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
upsampled_random_forest_performance_train = model_performance_evaluation(y_train,
upsampled_random_forest_performance_train['model'] = ['upsampled_random_forest'] *
upsampled_random_forest_performance_train['set'] = ['train'] * 5
print('Upsampled Random Forest Model Performance on Train Data: ')
display(upsampled_random_forest_performance_train)
```

Upsampled Random Forest Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.973684	upsampled_random_forest	train
1	Precision	0.906250	upsampled_random_forest	train
2	Recall	1.000000	upsampled_random_forest	train
3	F1	0.950820	upsampled_random_forest	train
4	AUROC	0.982353	upsampled_random_forest	train

In [211…

```
###################################
# Evaluating the upsampled Random Forest model
# on the test set
###################################
upsampled_random_forest_y_hat_test = upsampled_random_forest.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
upsampled_random_forest_performance_test = model_performance_evaluation(y_test, up
upsampled_random_forest_performance_test['model'] = ['upsampled_random_forest'] *
upsampled_random_forest_performance_test['set'] = ['test'] * 5
print('Upsampled Random Forest Model Performance on Test Data: ')
display(upsampled_random_forest_performance_test)
```

```
Upsampled Random Forest Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	upsampled_random_forest	test
1	Precision	0.888889	upsampled_random_forest	test
2	Recall	0.666667	upsampled_random_forest	test
3	F1	0.761905	upsampled_random_forest	test
4	AUROC	0.819820	upsampled_random_forest	test

1.8.5 Support Vector Machine

Support Vector Machine plots each observation in an N-dimensional space corresponding to the number of features in the data set and finds a hyperplane that maximally separates the different classes by a maximally large margin (which is defined as the distance between the hyperplane and the closest data points from each class). The algorithm applies kernel transformation by mapping non-linearly separable data using the similarities between the points in a high-dimensional feature space for improved discrimination.

1. The support vector machine model from the **sklearn.svm** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - C = inverse of regularization strength held constant at a value of 1
   - kernel = kernel type to be used in the algorithm made to vary between Linear, Poly, RBF and Sigmoid
   - class_weight = weights associated with classes held constant at a value of None
3. The extended model training data by upsampling the minority HIGH CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - C = 1
   - kernel = Linear
   - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9736
   - **Precision** = 0.9062
   - **Recall** = 1.0000
   - **F1 Score** = 0.9508
   - **AUROC** = 0.9823
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.8979
   - **Precision** = 0.8181
   - **Recall** = 0.7500

- **F1 Score** = 0.7826
- **AUROC** = 0.8479

7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [212…

```
###################################
# Creating an instance of the
# Support Vector Machine model
###################################
support_vector_machine = SVC()

###################################
# Defining the hyperparameters for the
# Support Vector Machine model
###################################
hyperparameter_grid = {
    'C': [1.0],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Support Vector Machine model
###################################
upsampled_support_vector_machine = GridSearchCV(estimator = support_vector_machine
                                                 param_grid = hyperparameter_grid,
                                                 n_jobs = -1,
                                                 scoring='f1')

###################################
# Fitting the upsampled Support Vector Machine model
###################################
upsampled_support_vector_machine.fit(X_train_smote, y_train_smote)

###################################
# Determining the optimal hyperparameter
# for the Support Vector Machine model
###################################
upsampled_support_vector_machine.best_score_
upsampled_support_vector_machine.best_params_
```

Out[212…

```
{'C': 1.0, 'class_weight': None, 'kernel': 'linear', 'random_state': 88888888}
```

In [213…

```
###################################
# Evaluating the upsampled Support Vector Machine model
# on the train set
###################################
upsampled_support_vector_machine_y_hat_train = upsampled_support_vector_machine.pr

###################################
# Gathering the model evaluation metrics
###################################
upsampled_support_vector_machine_performance_train = model_performance_evaluation(
```

```
upsampled_support_vector_machine_performance_train['model'] = ['upsampled_support_
upsampled_support_vector_machine_performance_train['set'] = ['train'] * 5
print('Upsampled Support Vector Machine Model Performance on Train Data: ')
display(upsampled_support_vector_machine_performance_train)
```

```
Upsampled Support Vector Machine Model Performance on Train Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.973684	upsampled_support_vector_machine	train
1	Precision	0.933333	upsampled_support_vector_machine	train
2	Recall	0.965517	upsampled_support_vector_machine	train
3	F1	0.949153	upsampled_support_vector_machine	train
4	AUROC	0.970994	upsampled_support_vector_machine	train

In [214…

```
###################################
# Evaluating the upsampled Support Vector Machine model
# on the test set
###################################
upsampled_support_vector_machine_y_hat_test = upsampled_support_vector_machine.pre

###################################
# Gathering the model evaluation metrics
###################################
upsampled_support_vector_machine_performance_test = model_performance_evaluation(y
upsampled_support_vector_machine_performance_test['model'] = ['upsampled_support_v
upsampled_support_vector_machine_performance_test['set'] = ['test'] * 5
print('Upsampled Support Vector Machine Model Performance on Test Data: ')
display(upsampled_support_vector_machine_performance_test)
```

```
Upsampled Support Vector Machine Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	upsampled_support_vector_machine	test
1	Precision	0.818182	upsampled_support_vector_machine	test
2	Recall	0.750000	upsampled_support_vector_machine	test
3	F1	0.782609	upsampled_support_vector_machine	test
4	AUROC	0.847973	upsampled_support_vector_machine	test

1.9. Model Development With CNN Downsampling

Condensed Nearest Neighbors is a prototype selection algorithm that aims to select a subset of instances from the original dataset, discarding redundant and less informative instances. The algorithm works by iteratively adding instances to the subset, starting with an empty set. At each iteration, an instance is added if it is not correctly classified by the current subset. The decision to add or discard an instance is based on its performance on a

k-nearest neighbors classifier. If an instance is misclassified by the current subset's k-nearest neighbors, it is added to the subset. The process is repeated until no new instances are added to the subset. The resulting subset is a condensed representation of the dataset that retains the essential information needed for classification.

1.9.1 Premodelling Data Description

1. Among the 9 numeric variables determined to have a statistically significant difference between the means of the numeric measurements obtained from LOW and HIGH groups of the CANRAT target variable, only 7 were retained with absolute T-Test statistics greater than 5.
   - GDPCAP: T.Test.Statistic=-11.937, T.Test.PValue=0.000
   - EPISCO: T.Test.Statistic=-11.789, T.Test.PValue=0.000
   - LIFEXP: T.Test.Statistic=-10.979, T.Test.PValue=0.000
   - TUBINC: T.Test.Statistic=+9.609, T.Test.PValue=0.000
   - DTHCMD: T.Test.Statistic=+8.376, T.Test.PValue=0.000
   - CO2EMI: T.Test.Statistic=-7.031, T.Test.PValue=0.000
   - URBPOP: T.Test.Statistic=-6.541, T.Test.PValue=0.000
2. Among the 4 categorical predictors determined to have a statistically significant relationship difference between the categories of the categorical predictors and the LOW and HIGH groups of the CANRAT target variable, only 1 was retained with absolute Chi-Square statistics greater than 15.
   - HDICAT_VH: ChiSquare.Test.Statistic=76.764, ChiSquare.Test.PValue=0.000
3. The CNN algorithm from the **imblearn.under_sampling** Python library API was implemented. The reduced model training data by downsampling the majority LOW CANRAT category applying CNN was used.

In [215…
```
##################################
# Consolidating relevant numeric columns
# and encoded categorical columns
# after hypothesis testing
##################################
cancer_rate_premodelling = cancer_rate_preprocessed_all.drop(['AGRLND','POPDEN','C
```

In [216…
```
##################################
# Performing a general exploration of the filtered dataset
##################################
print('Dataset Dimensions: ')
display(cancer_rate_premodelling.shape)
```

```
Dataset Dimensions:
(163, 9)
```

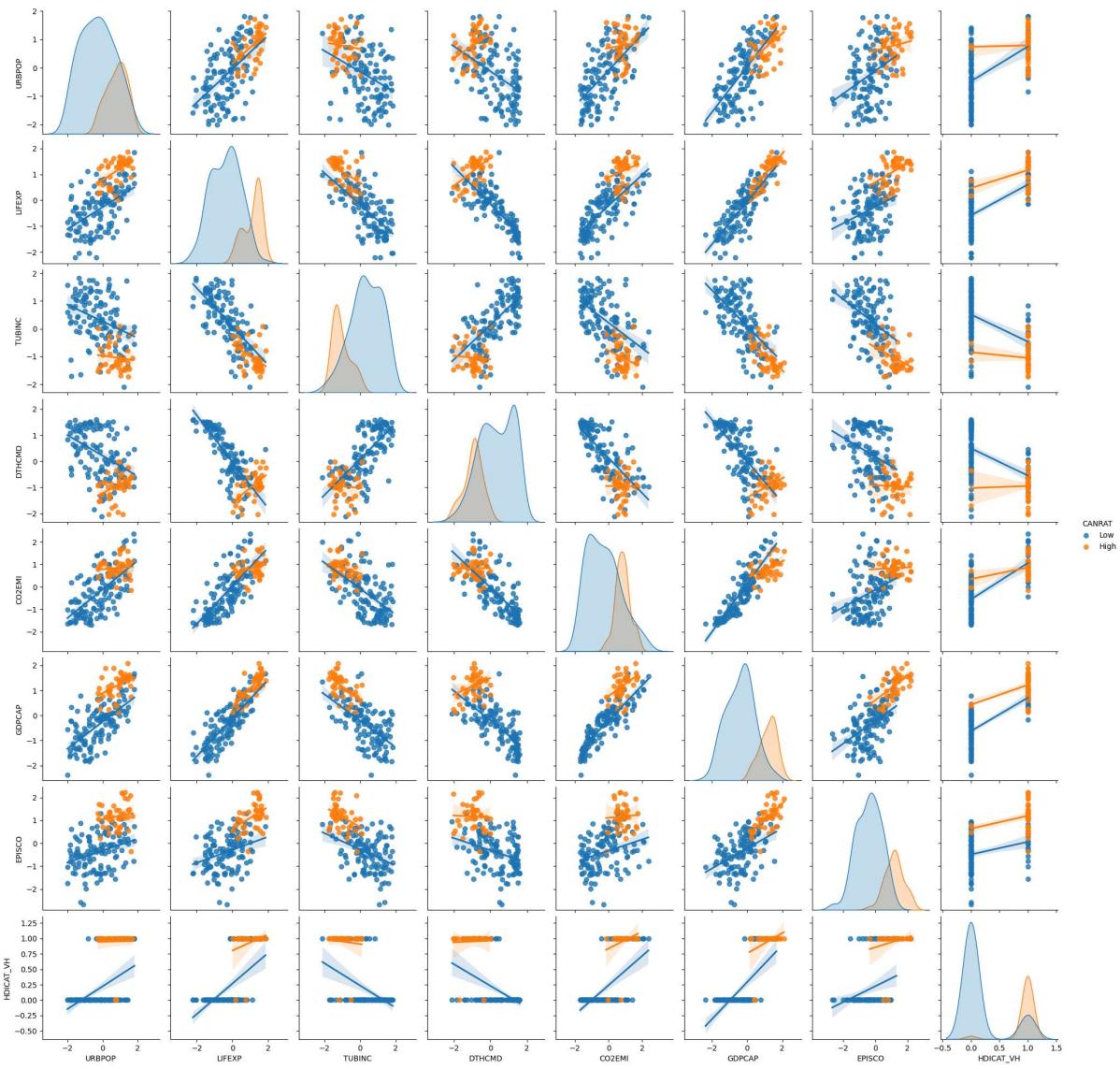
In [217…
```
##################################
# Listing the column names and data types
##################################
```

```
print('Column Names and Data Types:')
display(cancer_rate_premodelling.dtypes)
```

```
Column Names and Data Types:
URBPOP        float64
LIFEXP        float64
TUBINC        float64
DTHCMD        float64
CO2EMI        float64
GDPCAP        float64
EPISCO        float64
CANRAT       category
HDICAT_VH        bool
dtype: object
```

In [218…

```
###################################
# Gathering the pairplot for all variables
###################################
sns.pairplot(cancer_rate_premodelling,
             kind='reg',
             hue='CANRAT');
plt.show()
```



In [219…
```
###################################
# Separating the target
# and predictor columns
###################################
X = cancer_rate_premodelling.drop('CANRAT', axis = 1)
y = cancer_rate_premodelling['CANRAT'].cat.codes
```

In [220…
```
###################################
# Formulating the train and test data
# using a 70-30 ratio
###################################
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_
```

In [221…
```
###################################
# Performing a general exploration of the train dataset
###################################
print('Dataset Dimensions: ')
display(X_train.shape)
```

```
Dataset Dimensions:
(114, 8)
```

In [222…
```
###################################
# Validating the class distribution of the train dataset
###################################
y_train.value_counts(normalize = True)
```

Out[222…
```
0    0.745614
1    0.254386
Name: proportion, dtype: float64
```

In [223…
```
###################################
# Initiating an oversampling instance
# on the train data using
# Condense Nearest Neighbors
###################################
cnn = CondensedNearestNeighbour(random_state = 88888888, n_neighbors=3)
X_train_cnn, y_train_cnn = cnn.fit_resample(X_train,y_train)
```

In [224…
```
###################################
# Performing a general exploration of the overampled train dataset
###################################
print('Dataset Dimensions: ')
display(X_train_cnn.shape)
```

```
Dataset Dimensions:
(50, 8)
```

In [225…
```
###################################
# Validating the class distribution of the overampled train dataset
###################################
y_train_cnn.value_counts(normalize = True)
```

Out[225…
```
1    0.58
0    0.42
Name: proportion, dtype: float64
```

In [226…
```
###################################
# Performing a general exploration of the test dataset
###################################
print('Dataset Dimensions: ')
display(X_test.shape)
```

```
Dataset Dimensions:
(49, 8)
```

In [227…
```
###################################
# Validating the class distribution of the test dataset
###################################
y_test.value_counts(normalize = True)
```

Out[227…
```
0    0.755102
1    0.244898
Name: proportion, dtype: float64
```

In [228…
```
###################################
# Defining a function to compute
# model performance
###################################
def model_performance_evaluation(y_true, y_pred):
    metric_name = ['Accuracy','Precision','Recall','F1','AUROC']
    metric_value = [accuracy_score(y_true, y_pred),
                    precision_score(y_true, y_pred),
                    recall_score(y_true, y_pred),
                    f1_score(y_true, y_pred),
                    roc_auc_score(y_true, y_pred)]
    metric_summary = pd.DataFrame(zip(metric_name, metric_value),
                                  columns=['metric_name','metric_value'])
    return(metric_summary)
```

1.9.2 Logistic Regression

Logistic Regression models the relationship between the probability of an event (among two outcome levels) by having the log-odds of the event be a linear combination of a set of predictors weighted by their respective parameter estimates. The parameters are estimated via maximum likelihood estimation by testing different values through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimates. Given the optimal parameters, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability.

1. The logistic regression model from the **sklearn.linear_model** Python library API was implemented.
2. The model contains 5 hyperparameters:

- C = inverse of regularization strength held constant at a value of 1
- penalty = penalty norm made to vary between L1 and L2
- solver = algorithm used in the optimization problem made to vary between Saga and Liblinear
- class_weight = weights associated with classes held constant at a value of None
- max_iter = maximum number of iterations taken for the solvers to converge held constant at a value of 500

3. The reduced model training data by downsampling the majority LOW CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - C = 1
   - penalty = L1 norm
   - solver = Liblinear
   - class_weight = None
   - max_iter = 500
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9473
   - **Precision** = 0.8484
   - **Recall** = 0.9655
   - **F1 Score** = 0.9032
   - **AUROC** = 0.9533
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.9183
   - **Precision** = 0.9000
   - **Recall** = 0.7500
   - **F1 Score** = 0.8181
   - **AUROC** = 0.8614
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [229…

```
##################################
# Creating an instance of the
# Logistic Regression model
##################################
logistic_regression = LogisticRegression()

##################################
# Defining the hyperparameters for the
# Logistic Regression model
##################################
hyperparameter_grid = {
    'C': [1.0],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear','saga'],
    'class_weight': [None],
```

```
    'max_iter': [500],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Logistic Regression model
###################################
downsampled_logistic_regression = GridSearchCV(estimator = logistic_regression,
                                                param_grid = hyperparameter_grid,
                                                n_jobs = -1,
                                                scoring='f1')

###################################
# Fitting the downsampled Logistic Regression model
###################################
downsampled_logistic_regression.fit(X_train_cnn, y_train_cnn)

###################################
# Determining the optimal hyperparameter
# for the Logistic Regression model
###################################
downsampled_logistic_regression.best_score_
downsampled_logistic_regression.best_params_
```

Out[229…

```
{'C': 1.0,
 'class_weight': None,
 'max_iter': 500,
 'penalty': 'l1',
 'random_state': 88888888,
 'solver': 'liblinear'}
```

In [230…

```
###################################
# Evaluating the downsampled Logistic Regression model
# on the train set
###################################
downsampled_logistic_regression_y_hat_train = downsampled_logistic_regression.prec

###################################
# Gathering the model evaluation metrics
###################################
downsampled_logistic_regression_performance_train = model_performance_evaluation(y
downsampled_logistic_regression_performance_train['model'] = ['downsampled_logisti
downsampled_logistic_regression_performance_train['set'] = ['train'] * 5
print('Downsampled Logistic Regression Model Performance on Train Data: ')
display(downsampled_logistic_regression_performance_train)
```

Downsampled Logistic Regression Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.947368	downsampled_logistic_regression	train
1	Precision	0.848485	downsampled_logistic_regression	train
2	Recall	0.965517	downsampled_logistic_regression	train
3	F1	0.903226	downsampled_logistic_regression	train
4	AUROC	0.953347	downsampled_logistic_regression	train

In [231…

```
###################################
# Evaluating the downsampled Logistic Regression model
# on the test set
###################################
downsampled_logistic_regression_y_hat_test = downsampled_logistic_regression.predi

###################################
# Gathering the model evaluation metrics
###################################
downsampled_logistic_regression_performance_test = model_performance_evaluation(y_
downsampled_logistic_regression_performance_test['model'] = ['downsampled_logistic
downsampled_logistic_regression_performance_test['set'] = ['test'] * 5
print('Downsampled Logistic Regression Model Performance on Test Data: ')
display(downsampled_logistic_regression_performance_test)
```

```
Downsampled Logistic Regression Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.918367	downsampled_logistic_regression	test
1	Precision	0.900000	downsampled_logistic_regression	test
2	Recall	0.750000	downsampled_logistic_regression	test
3	F1	0.818182	downsampled_logistic_regression	test
4	AUROC	0.861486	downsampled_logistic_regression	test

1.9.3 Decision Trees

Decision trees create a model that predicts the class label of a sample based on input features. A decision tree consists of nodes that represent decisions or choices, edges which connect nodes and represent the possible outcomes of a decision and leaf (or terminal) nodes which represent the final decision or the predicted class label. The decision-making process involves feature selection (at each internal node, the algorithm decides which feature to split on based on a certain criterion including gini impurity or entropy), splitting criteria (the splitting criteria aim to find the feature and its corresponding threshold that best separates the data into different classes. The goal is to increase homogeneity within each resulting subset), recursive splitting (the process of feature selection and splitting continues recursively, creating a tree structure. The dataset is partitioned at each internal

node based on the chosen feature, and the process repeats for each subset) and stopping criteria (the recursion stops when a certain condition is met, known as a stopping criterion. Common stopping criteria include a maximum depth for the tree, a minimum number of samples required to split a node, or a minimum number of samples in a leaf node.)

1. The decision tree model from the **sklearn.tree** Python library API was implemented.
2. The model contains 5 hyperparameters:
   - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
   - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
   - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
   - class_weight = weights associated with classes held constant at a value of None
3. The reduced model training data by downsampling the majority LOW CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
   - criterion = Gini
   - max_depth = 3
   - min_samples_leaf = 5
   - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
   - **Accuracy** = 0.9385
   - **Precision** = 0.9230
   - **Recall** = 0.8275
   - **F1 Score** = 0.8727
   - **AUROC** = 0.9020
6. The independent test model performance of the final model is summarized as follows:
   - **Accuracy** = 0.8979
   - **Precision** = 0.8888
   - **Recall** = 0.6666
   - **F1 Score** = 0.7619
   - **AUROC** = 0.8198
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [232…

```
##################################
# Creating an instance of the
# Decision Tree model
##################################
decision_tree = DecisionTreeClassifier()

##################################
# Defining the hyperparameters for the
# Decision Tree model
```

```
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Decision Tree model
###################################
downsampled_decision_tree = GridSearchCV(estimator = decision_tree,
                                         param_grid = hyperparameter_grid,
                                         n_jobs = -1,
                                         scoring='f1')

###################################
# Fitting the downsampled Decision Tree model
###################################
downsampled_decision_tree.fit(X_train_cnn, y_train_cnn)

###################################
# Determining the optimal hyperparameter
# for the Decision Tree model
###################################
downsampled_decision_tree.best_score_
downsampled_decision_tree.best_params_
```

Out[232…

```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': 3,
 'min_samples_leaf': 5,
 'random_state': 88888888}
```

In [233…

```
###################################
# Evaluating the downsampled Decision Tree model
# on the train set
###################################
downsampled_decision_tree_y_hat_train = downsampled_decision_tree.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
downsampled_decision_tree_performance_train = model_performance_evaluation(y_train
downsampled_decision_tree_performance_train['model'] = ['downsampled_decision_tree
downsampled_decision_tree_performance_train['set'] = ['train'] * 5
print('Downsampled Decision Tree Model Performance on Train Data: ')
display(downsampled_decision_tree_performance_train)
```

Downsampled Decision Tree Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.938596	downsampled_decision_tree	train
1	Precision	0.923077	downsampled_decision_tree	train
2	Recall	0.827586	downsampled_decision_tree	train
3	F1	0.872727	downsampled_decision_tree	train
4	AUROC	0.902028	downsampled_decision_tree	train

In [234…

```
###################################
# Evaluating the downsampled Decision Tree model
# on the test set
###################################
downsampled_decision_tree_y_hat_test = downsampled_decision_tree.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
downsampled_decision_tree_performance_test = model_performance_evaluation(y_test,
downsampled_decision_tree_performance_test['model'] = ['downsampled_decision_tree'
downsampled_decision_tree_performance_test['set'] = ['test'] * 5
print('Downsampled Decision Tree Model Performance on Test Data: ')
display(downsampled_decision_tree_performance_test)
```

```
Downsampled Decision Tree Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	downsampled_decision_tree	test
1	Precision	0.888889	downsampled_decision_tree	test
2	Recall	0.666667	downsampled_decision_tree	test
3	F1	0.761905	downsampled_decision_tree	test
4	AUROC	0.819820	downsampled_decision_tree	test

1.9.4 Random Forest

Random Forest is an ensemble learning method made up of a large set of small decision trees called estimators, with each producing its own prediction. The random forest model aggregates the predictions of the estimators to produce a more accurate prediction. The algorithm involves bootstrap aggregating (where smaller subsets of the training data are repeatedly subsampled with replacement), random subspacing (where a subset of features are sampled and used to train each individual estimator), estimator training (where unpruned decision trees are formulated for each estimator) and inference by aggregating the predictions of all estimators.

1. The random forest model from the **sklearn.ensemble** Python library API was implemented.
2. The model contains 5 hyperparameters:
    - criterion = function to measure the quality of a split made to vary between Gini, Entropy and Log-Loss
    - max_depth = maximum depth of the tree made to vary between 3, 5 and 7
    - min_samples_leaf = minimum number of samples required to split an internal node made to vary between 3, 5 and 10
    - n_estimators = number of trees in the forest made to vary between 100, 150 and 200
    - max_features = number of features to consider when looking for the best split made to vary between Sqrt and Log2 of n_estimators
    - class_weight = weights associated with classes held constant at a value of None
3. The reduced model training data by downsampling the majority LOW CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - criterion = Gini
    - max_depth = 3
    - min_samples_leaf = 3
    - n_estimators = 100
    - max_features = Sqrt n_estimators
    - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9649
    - **Precision** = 0.9032
    - **Recall** = 0.9655
    - **F1 Score** = 0.9333
    - **AUROC** = 0.9651
6. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.8979
    - **Precision** = 0.8888
    - **Recall** = 0.6666
    - **F1 Score** = 0.7619
    - **AUROC** = 0.8198
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [235…

```
#################################
# Creating an instance of the
# Random Forest model
#################################
random_forest = RandomForestClassifier()
```

```
###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
hyperparameter_grid = {
    'criterion': ['gini','entropy','log_loss'],
    'max_depth': [3,5,7],
    'min_samples_leaf': [3,5,10],
    'n_estimators': [100,150,200],
    'max_features':['sqrt', 'log2'],
    'class_weight': [None],
    'random_state': [88888888]}

###################################
# Defining the hyperparameters for the
# Random Forest model
###################################
downsampled_random_forest = GridSearchCV(estimator = random_forest,
                                         param_grid = hyperparameter_grid,
                                         n_jobs = -1,
                                         scoring='f1')

###################################
# Fitting the downsampled Random Forest model
###################################
downsampled_random_forest.fit(X_train_cnn, y_train_cnn)

###################################
# Determining the optimal hyperparameter
# for the Random Forest model
###################################
downsampled_random_forest.best_score_
downsampled_random_forest.best_params_
```

Out[235…

```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': 3,
 'max_features': 'sqrt',
 'min_samples_leaf': 3,
 'n_estimators': 100,
 'random_state': 88888888}
```

In [236…

```
###################################
# Evaluating the downsampled Random Forest model
# on the train set
###################################
downsampled_random_forest_y_hat_train = downsampled_random_forest.predict(X_train)

###################################
# Gathering the model evaluation metrics
###################################
downsampled_random_forest_performance_train = model_performance_evaluation(y_train
downsampled_random_forest_performance_train['model'] = ['downsampled_random_forest
downsampled_random_forest_performance_train['set'] = ['train'] * 5
print('Downsampled Random Forest Model Performance on Train Data: ')
display(downsampled_random_forest_performance_train)
```

Downsampled Random Forest Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.964912	downsampled_random_forest	train
1	Precision	0.903226	downsampled_random_forest	train
2	Recall	0.965517	downsampled_random_forest	train
3	F1	0.933333	downsampled_random_forest	train
4	AUROC	0.965112	downsampled_random_forest	train

In [237…

```
###################################
# Evaluating the downsampled Random Forest model
# on the test set
###################################
downsampled_random_forest_y_hat_test = downsampled_random_forest.predict(X_test)

###################################
# Gathering the model evaluation metrics
###################################
downsampled_random_forest_performance_test = model_performance_evaluation(y_test,
downsampled_random_forest_performance_test['model'] = ['downsampled_random_forest'
downsampled_random_forest_performance_test['set'] = ['test'] * 5
print('Downsampled Random Forest Model Performance on Test Data: ')
display(downsampled_random_forest_performance_test)
```

Downsampled Random Forest Model Performance on Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.897959	downsampled_random_forest	test
1	Precision	0.888889	downsampled_random_forest	test
2	Recall	0.666667	downsampled_random_forest	test
3	F1	0.761905	downsampled_random_forest	test
4	AUROC	0.819820	downsampled_random_forest	test

1.9.5 Support Vector Machine

Support Vector Machine plots each observation in an N-dimensional space corresponding to the number of features in the data set and finds a hyperplane that maximally separates the different classes by a maximally large margin (which is defined as the distance between the hyperplane and the closest data points from each class). The algorithm applies kernel transformation by mapping non-linearly separable data using the similarities between the points in a high-dimensional feature space for improved discrimination.

1. The support vector machine model from the **sklearn.svm** Python library API was implemented.

2. The model contains 5 hyperparameters:
    - C = inverse of regularization strength held constant at a value of 1
    - kernel = kernel type to be used in the algorithm made to vary between Linear, Poly, RBF and Sigmoid
    - class_weight = weights associated with classes held constant at a value of None
3. The reduced model training data by downsampling the majority LOW CANRAT category was used.
4. Hyperparameter tuning was conducted using the 5-fold cross-validation method with optimal model performance using the F1 score determined for:
    - C = 1
    - kernel = Linear
    - class_weight = None
5. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9561
    - **Precision** = 0.9285
    - **Recall** = 0.8965
    - **F1 Score** = 0.9122
    - **AUROC** = 0.9365
6. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.8979
    - **Precision** = 0.8888
    - **Recall** = 0.6666
    - **F1 Score** = 0.7619
    - **AUROC** = 0.8198
7. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

In [238…

```
##################################
# Creating an instance of the
# Support Vector Machine model
##################################
support_vector_machine = SVC()

##################################
# Defining the hyperparameters for the
# Support Vector Machine model
##################################
hyperparameter_grid = {
    'C': [1.0],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'class_weight': [None],
    'random_state': [88888888]}

##################################
# Defining the hyperparameters for the
# Support Vector Machine model
##################################
```

```
downsampled_support_vector_machine = GridSearchCV(estimator = support_vector_machi
                                                   param_grid = hyperparameter_grid,
                                                   n_jobs = -1,
                                                   scoring='f1')

###################################
# Fitting the downsampled Support Vector Machine model
###################################
downsampled_support_vector_machine.fit(X_train_cnn, y_train_cnn)

###################################
# Determining the optimal hyperparameter
# for the Support Vector Machine model
###################################
downsampled_support_vector_machine.best_score_
downsampled_support_vector_machine.best_params_
```

Out[238… `{'C': 1.0, 'class_weight': None, 'kernel': 'linear', 'random_state': 88888888}`

In [239…
```
###################################
# Evaluating the downsampled Support Vector Machine model
# on the train set
###################################
downsampled_support_vector_machine_y_hat_train = downsampled_support_vector_machir

###################################
# Gathering the model evaluation metrics
###################################
downsampled_support_vector_machine_performance_train = model_performance_evaluatic
downsampled_support_vector_machine_performance_train['model'] = ['downsampled_supp
downsampled_support_vector_machine_performance_train['set'] = ['train'] * 5
print('Downsampled Support Vector Machine Model Performance on Train Data: ')
display(downsampled_support_vector_machine_performance_train)
```

Downsampled Support Vector Machine Model Performance on Train Data:

	metric_name	metric_value	model	set
0	Accuracy	0.956140	downsampled_support_vector_machine	train
1	Precision	0.928571	downsampled_support_vector_machine	train
2	Recall	0.896552	downsampled_support_vector_machine	train
3	F1	0.912281	downsampled_support_vector_machine	train
4	AUROC	0.936511	downsampled_support_vector_machine	train

In [240…
```
###################################
# Evaluating the downsampled Support Vector Machine model
# on the test set
###################################
downsampled_support_vector_machine_y_hat_test = downsampled_support_vector_machine

###################################
# Gathering the model evaluation metrics
###################################
```

```
downsampled_support_vector_machine_performance_test = model_performance_evaluatior
downsampled_support_vector_machine_performance_test['model'] = ['downsampled_suppo
downsampled_support_vector_machine_performance_test['set'] = ['test'] * 5
print('Downsampled Support Vector Machine Model Performance on Test Data: ')
display(downsampled_support_vector_machine_performance_test)
```

```
Downsampled Support Vector Machine Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.897959	downsampled_support_vector_machine	test
1	Precision	0.888889	downsampled_support_vector_machine	test
2	Recall	0.666667	downsampled_support_vector_machine	test
3	F1	0.761905	downsampled_support_vector_machine	test
4	AUROC	0.819820	downsampled_support_vector_machine	test

1.10. Model Development With Stacking Ensemble Learning

Model Stacking - also known as stacked generalization, is an ensemble approach which involves creating a variety of base learners and using them to create intermediate predictions, one for each learned model. A meta-model is incorporated that gains knowledge of the same target from intermediate predictions. Unlike bagging, in stacking, the models are typically different (e.g. not all decision trees) and fit on the same dataset (e.g. instead of samples of the training dataset). Unlike boosting, in stacking, a single model is used to learn how to best combine the predictions from the contributing models (e.g. instead of a sequence of models that correct the predictions of prior models). Stacking is appropriate when the predictions made by the base learners or the errors in predictions made by the models have minimal correlation. Achieving an improvement in performance is dependent upon the choice of base learners and whether they are sufficiently skillful in their predictions.

1.10.1 Premodelling Data Description

1. Among the formulated versions of the logistic regression model, the model which applied class weights demonstrated the best independent test model performance. Considerable difference in the apparent and independent test model performance was observed, indicative of the presence of moderate model overfitting.
    - **Accuracy** = 0.9387
    - **Precision** = 0.8461
    - **Recall** = 0.9167
    - **F1 Score** = 0.8800
    - **AUROC** = 0.9313

2. Among the formulated versions of the decision tree model, the model which applied upsampling of the minority class using SMOTE demonstrated the best independent test model performance. Considerable difference in the apparent and independent test model performance was observed, indicative of the presence of moderate model overfitting.
    - **Accuracy** = 0.8979
    - **Precision** = 0.7692
    - **Recall** = 0.8333
    - **F1 Score** = 0.8000
    - **AUROC** = 0.8761
3. Among the formulated versions of the random forest model, the model which applied upsampling of the minority class using SMOTE demonstrated the best independent test model performance. High difference in the apparent and independent test model performance was observed, indicative of the presence of excessive model overfitting.
    - **Accuracy** = 0.9387
    - **Precision** = 0.8461
    - **Recall** = 0.9167
    - **F1 Score** = 0.8800
    - **AUROC** = 0.9313
4. Among the formulated versions of the support vector machine model, the model which applied upsampling of the minority class using SMOTE demonstrated the best independent test model performance. High difference in the apparent and independent test model performance was observed, indicative of the presence of excessive model overfitting.
    - **Accuracy** = 0.8979
    - **Precision** = 0.8181
    - **Recall** = 0.7500
    - **F1 Score** = 0.7826
    - **AUROC** = 0.8479
5. All individual formulated models which applied upsampling of the minority class using SMOTE were used to generate the base-learners for the stacking algorithm.

In [241…

```
###################################
# Consolidating all the
# Logistic Regression
# model performance measures
###################################
logistic_regression_performance_comparison = pd.concat([optimal_logistic_regressio
                                                        optimal_logistic_regressio
                                                        weighted_logistic_regressi
                                                        weighted_logistic_regressi
                                                        upsampled_logistic_regress
                                                        upsampled_logistic_regress
                                                        downsampled_logistic_regre
                                                        downsampled_logistic_regre
                                                       ignore_index=True)
```

```
print('Consolidated Logistic Regression Model Performance on Train and Test Data:
display(logistic_regression_performance_comparison)
```

Consolidated Logistic Regression Model Performance on Train and Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.947368	optimal_logistic_regression	train
1	Precision	0.870968	optimal_logistic_regression	train
2	Recall	0.931034	optimal_logistic_regression	train
3	F1	0.900000	optimal_logistic_regression	train
4	AUROC	0.941988	optimal_logistic_regression	train
5	Accuracy	0.897959	optimal_logistic_regression	test
6	Precision	0.888889	optimal_logistic_regression	test
7	Recall	0.666667	optimal_logistic_regression	test
8	F1	0.761905	optimal_logistic_regression	test
9	AUROC	0.819820	optimal_logistic_regression	test
10	Accuracy	0.894737	weighted_logistic_regression	train
11	Precision	0.707317	weighted_logistic_regression	train
12	Recall	1.000000	weighted_logistic_regression	train
13	F1	0.828571	weighted_logistic_regression	train
14	AUROC	0.929412	weighted_logistic_regression	train
15	Accuracy	0.938776	weighted_logistic_regression	test
16	Precision	0.846154	weighted_logistic_regression	test
17	Recall	0.916667	weighted_logistic_regression	test
18	F1	0.880000	weighted_logistic_regression	test
19	AUROC	0.931306	weighted_logistic_regression	test
20	Accuracy	0.964912	upsampled_logistic_regression	train
21	Precision	0.903226	upsampled_logistic_regression	train
22	Recall	0.965517	upsampled_logistic_regression	train
23	F1	0.933333	upsampled_logistic_regression	train
24	AUROC	0.965112	upsampled_logistic_regression	train
25	Accuracy	0.918367	upsampled_logistic_regression	test
26	Precision	0.900000	upsampled_logistic_regression	test
27	Recall	0.750000	upsampled_logistic_regression	test
28	F1	0.818182	upsampled_logistic_regression	test
29	AUROC	0.861486	upsampled_logistic_regression	test

	metric_name	metric_value	model	set
30	Accuracy	0.947368	downsampled_logistic_regression	train
31	Precision	0.848485	downsampled_logistic_regression	train
32	Recall	0.965517	downsampled_logistic_regression	train
33	F1	0.903226	downsampled_logistic_regression	train
34	AUROC	0.953347	downsampled_logistic_regression	train
35	Accuracy	0.918367	downsampled_logistic_regression	test
36	Precision	0.900000	downsampled_logistic_regression	test
37	Recall	0.750000	downsampled_logistic_regression	test
38	F1	0.818182	downsampled_logistic_regression	test
39	AUROC	0.861486	downsampled_logistic_regression	test

In [242…

```
###################################
# Consolidating all the F1 score
# model performance measures
###################################
logistic_regression_performance_comparison_F1 = logistic_regression_performance_co
logistic_regression_performance_comparison_F1_train = logistic_regression_performa
logistic_regression_performance_comparison_F1_test = logistic_regression_performar
```

In [243…

```
###################################
# Combining all the F1 score
# model performance measures
# between train and test sets
###################################
logistic_regression_performance_comparison_F1_plot = pd.DataFrame({'train': logist
                                                                   'test': logisti
                                                                  index=logistic_r
logistic_regression_performance_comparison_F1_plot
```

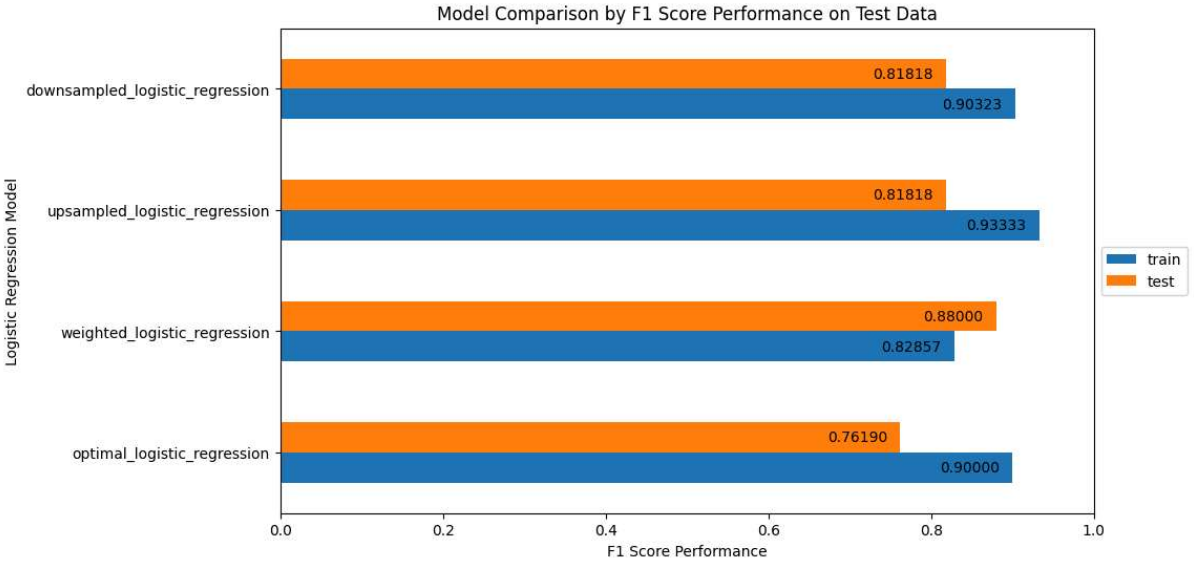
Out[243…

	train	test
optimal_logistic_regression	0.900000	0.761905
weighted_logistic_regression	0.828571	0.880000
upsampled_logistic_regression	0.933333	0.818182
downsampled_logistic_regression	0.903226	0.818182

In [244…

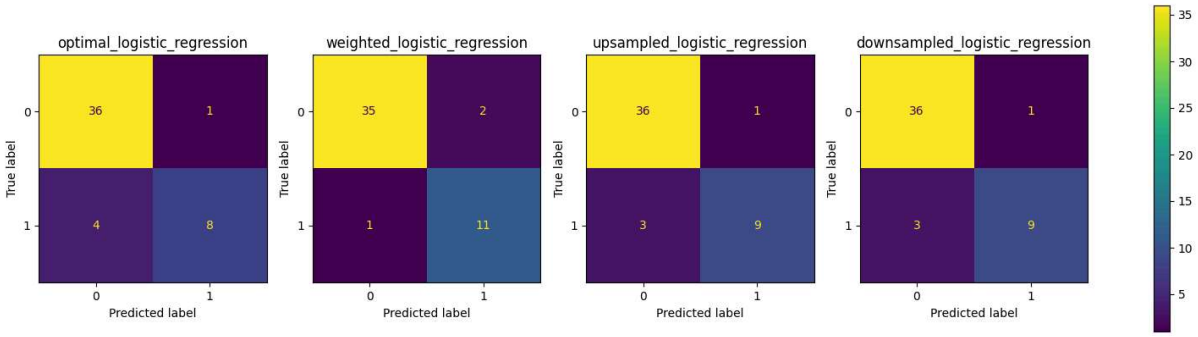
```
###################################
# Plotting all the F1 score
# model performance measures
# between train and test sets
###################################
logistic_regression_performance_comparison_F1_plot = logistic_regression_performar
logistic_regression_performance_comparison_F1_plot.set_xlim(0.00,1.00)
```

```
logistic_regression_performance_comparison_F1_plot.set_title("Model Comparison by
logistic_regression_performance_comparison_F1_plot.set_xlabel("F1 Score Performanc
logistic_regression_performance_comparison_F1_plot.set_ylabel("Logistic Regressior
logistic_regression_performance_comparison_F1_plot.grid(False)
logistic_regression_performance_comparison_F1_plot.legend(loc='center left', bbox_
for container in logistic_regression_performance_comparison_F1_plot.containers:
    logistic_regression_performance_comparison_F1_plot.bar_label(container, fmt='%
```



In [245…

```
###################################
# Plotting the confusion matrices
# for all the Logistic Regression models
###################################
classifiers = {"optimal_logistic_regression": optimal_logistic_regression,
               "weighted_logistic_regression": weighted_logistic_regression,
               "upsampled_logistic_regression": upsampled_logistic_regression,
               "downsampled_logistic_regression": downsampled_logistic_regression}

fig, axes = plt.subplots(1, 4, figsize=(20, 5))
for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cf_matrix)
    disp.plot(ax=axes[i], xticks_rotation=0)
    disp.ax_.grid(False)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()

fig.colorbar(disp.im_, ax=axes)
plt.show()
```



In [246…

```
###################################
# Consolidating all the
# Decision Tree
# model performance measures
###################################
decision_tree_performance_comparison = pd.concat([optimal_decision_tree_performanc
                                                  optimal_decision_tree_performanc
                                                  weighted_decision_tree_performar
                                                  weighted_decision_tree_performar
                                                  upsampled_decision_tree_performa
                                                  upsampled_decision_tree_performa
                                                  downsampled_decision_tree_perfor
                                                  downsampled_decision_tree_perfor
                                                 ignore_index=True)
print('Consolidated Decision Tree Model Performance on Train and Test Data: ')
display(decision_tree_performance_comparison)
```

Consolidated Decision Tree Model Performance on Train and Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.973684	optimal_decision_tree	train
1	Precision	1.000000	optimal_decision_tree	train
2	Recall	0.896552	optimal_decision_tree	train
3	F1	0.945455	optimal_decision_tree	train
4	AUROC	0.948276	optimal_decision_tree	train
5	Accuracy	0.857143	optimal_decision_tree	test
6	Precision	0.857143	optimal_decision_tree	test
7	Recall	0.500000	optimal_decision_tree	test
8	F1	0.631579	optimal_decision_tree	test
9	AUROC	0.736486	optimal_decision_tree	test
10	Accuracy	0.956140	weighted_decision_tree	train
11	Precision	0.852941	weighted_decision_tree	train
12	Recall	1.000000	weighted_decision_tree	train
13	F1	0.920635	weighted_decision_tree	train
14	AUROC	0.970588	weighted_decision_tree	train
15	Accuracy	0.897959	weighted_decision_tree	test
16	Precision	0.769231	weighted_decision_tree	test
17	Recall	0.833333	weighted_decision_tree	test
18	F1	0.800000	weighted_decision_tree	test
19	AUROC	0.876126	weighted_decision_tree	test
20	Accuracy	0.921053	upsampled_decision_tree	train
21	Precision	0.763158	upsampled_decision_tree	train
22	Recall	1.000000	upsampled_decision_tree	train
23	F1	0.865672	upsampled_decision_tree	train
24	AUROC	0.947059	upsampled_decision_tree	train
25	Accuracy	0.897959	upsampled_decision_tree	test
26	Precision	0.769231	upsampled_decision_tree	test
27	Recall	0.833333	upsampled_decision_tree	test
28	F1	0.800000	upsampled_decision_tree	test
29	AUROC	0.876126	upsampled_decision_tree	test

	metric_name	metric_value	model	set
30	Accuracy	0.938596	downsampled_decision_tree	train
31	Precision	0.923077	downsampled_decision_tree	train
32	Recall	0.827586	downsampled_decision_tree	train
33	F1	0.872727	downsampled_decision_tree	train
34	AUROC	0.902028	downsampled_decision_tree	train
35	Accuracy	0.897959	downsampled_decision_tree	test
36	Precision	0.888889	downsampled_decision_tree	test
37	Recall	0.666667	downsampled_decision_tree	test
38	F1	0.761905	downsampled_decision_tree	test
39	AUROC	0.819820	downsampled_decision_tree	test

In [247…

```
###################################
# Consolidating all the F1 score
# model performance measures
###################################
decision_tree_performance_comparison_F1 = decision_tree_performance_comparison[dec
decision_tree_performance_comparison_F1_train = decision_tree_performance_comparis
decision_tree_performance_comparison_F1_test = decision_tree_performance_comparisc
```

In [248…

```
###################################
# Combining all the F1 score
# model performance measures
# between train and test sets
###################################
decision_tree_performance_comparison_F1_plot = pd.DataFrame({'train': decision_tre
                                                             'test': decision_tree
                                                            index=decision_tree_pe
decision_tree_performance_comparison_F1_plot
```

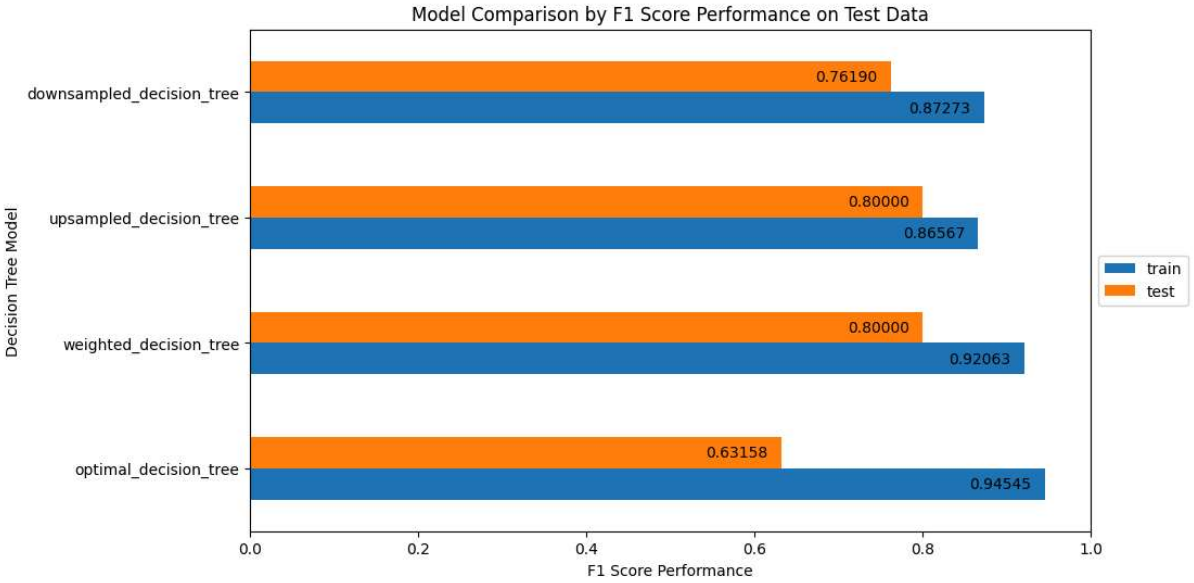
Out[248…

	train	test
optimal_decision_tree	0.945455	0.631579
weighted_decision_tree	0.920635	0.800000
upsampled_decision_tree	0.865672	0.800000
downsampled_decision_tree	0.872727	0.761905

In [249…

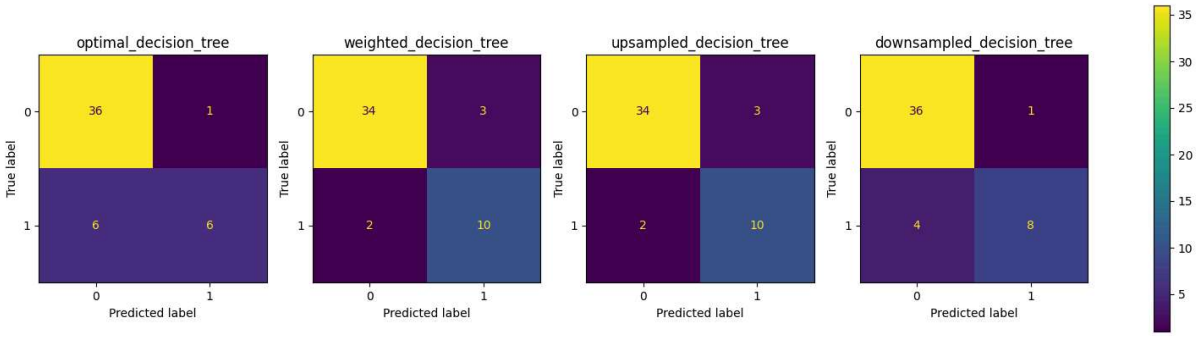
```
###################################
# Plotting all the F1 score
# model performance measures
# between train and test sets
###################################
decision_tree_performance_comparison_F1_plot = decision_tree_performance_comparisc
decision_tree_performance_comparison_F1_plot.set_xlim(0.00,1.00)
```

```
decision_tree_performance_comparison_F1_plot.set_title("Model Comparison by F1 Sc
decision_tree_performance_comparison_F1_plot.set_xlabel("F1 Score Performance")
decision_tree_performance_comparison_F1_plot.set_ylabel("Decision Tree Model")
decision_tree_performance_comparison_F1_plot.grid(False)
decision_tree_performance_comparison_F1_plot.legend(loc='center left', bbox_to_an
for container in decision_tree_performance_comparison_F1_plot.containers:
    decision_tree_performance_comparison_F1_plot.bar_label(container, fmt='%.5f',
```



In [250…

```
###################################
# Plotting the confusion matrices
# for all the Decision Tree models
###################################
classifiers = {"optimal_decision_tree": optimal_decision_tree,
               "weighted_decision_tree": weighted_decision_tree,
               "upsampled_decision_tree": upsampled_decision_tree,
               "downsampled_decision_tree": downsampled_decision_tree}

fig, axes = plt.subplots(1, 4, figsize=(20, 5))
for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cf_matrix)
    disp.plot(ax=axes[i], xticks_rotation=0)
    disp.ax_.grid(False)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()

fig.colorbar(disp.im_, ax=axes)
plt.show()
```



In [251…

```
###################################
# Consolidating all the
# Random Forest
# model performance measures
###################################
random_forest_performance_comparison = pd.concat([optimal_random_forest_performanc
                                                  optimal_random_forest_performanc
                                                  weighted_random_forest_performar
                                                  weighted_random_forest_performar
                                                  upsampled_random_forest_performa
                                                  upsampled_random_forest_performa
                                                  downsampled_random_forest_perfor
                                                  downsampled_random_forest_perfor
                                                 ignore_index=True)
print('Consolidated Random Forest Model Performance on Train and Test Data: ')
display(random_forest_performance_comparison)
```

Consolidated Random Forest Model Performance on Train and Test Data:

	metric_name	metric_value	model	set
0	Accuracy	0.956140	optimal_random_forest	train
1	Precision	0.928571	optimal_random_forest	train
2	Recall	0.896552	optimal_random_forest	train
3	F1	0.912281	optimal_random_forest	train
4	AUROC	0.936511	optimal_random_forest	train
5	Accuracy	0.877551	optimal_random_forest	test
6	Precision	0.875000	optimal_random_forest	test
7	Recall	0.583333	optimal_random_forest	test
8	F1	0.700000	optimal_random_forest	test
9	AUROC	0.778153	optimal_random_forest	test
10	Accuracy	0.973684	weighted_random_forest	train
11	Precision	0.906250	weighted_random_forest	train
12	Recall	1.000000	weighted_random_forest	train
13	F1	0.950820	weighted_random_forest	train
14	AUROC	0.982353	weighted_random_forest	train
15	Accuracy	0.897959	weighted_random_forest	test
16	Precision	0.888889	weighted_random_forest	test
17	Recall	0.666667	weighted_random_forest	test
18	F1	0.761905	weighted_random_forest	test
19	AUROC	0.819820	weighted_random_forest	test
20	Accuracy	0.973684	upsampled_random_forest	train
21	Precision	0.906250	upsampled_random_forest	train
22	Recall	1.000000	upsampled_random_forest	train
23	F1	0.950820	upsampled_random_forest	train
24	AUROC	0.982353	upsampled_random_forest	train
25	Accuracy	0.897959	upsampled_random_forest	test
26	Precision	0.888889	upsampled_random_forest	test
27	Recall	0.666667	upsampled_random_forest	test
28	F1	0.761905	upsampled_random_forest	test
29	AUROC	0.819820	upsampled_random_forest	test

	metric_name	metric_value	model	set
30	Accuracy	0.964912	downsampled_random_forest	train
31	Precision	0.903226	downsampled_random_forest	train
32	Recall	0.965517	downsampled_random_forest	train
33	F1	0.933333	downsampled_random_forest	train
34	AUROC	0.965112	downsampled_random_forest	train
35	Accuracy	0.897959	downsampled_random_forest	test
36	Precision	0.888889	downsampled_random_forest	test
37	Recall	0.666667	downsampled_random_forest	test
38	F1	0.761905	downsampled_random_forest	test
39	AUROC	0.819820	downsampled_random_forest	test

In [252…
```
###################################
# Consolidating all the F1 score
# model performance measures
###################################
random_forest_performance_comparison_F1 = random_forest_performance_comparison[rar
random_forest_performance_comparison_F1_train = random_forest_performance_comparis
random_forest_performance_comparison_F1_test = random_forest_performance_comparisc
```

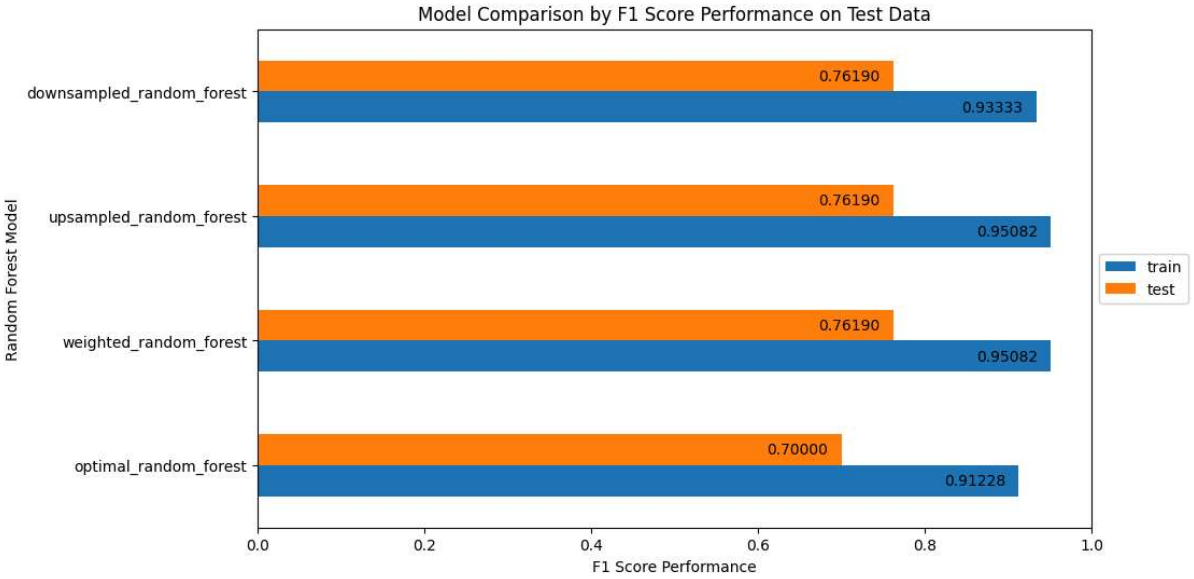
In [253…
```
###################################
# Combining all the F1 score
# model performance measures
# between train and test sets
###################################
random_forest_performance_comparison_F1_plot = pd.DataFrame({'train': random_fores
                                                             'test': random_forest
                                                            index=random_forest_pe
random_forest_performance_comparison_F1_plot
```

Out[253…

	train	test
optimal_random_forest	0.912281	0.700000
weighted_random_forest	0.950820	0.761905
upsampled_random_forest	0.950820	0.761905
downsampled_random_forest	0.933333	0.761905

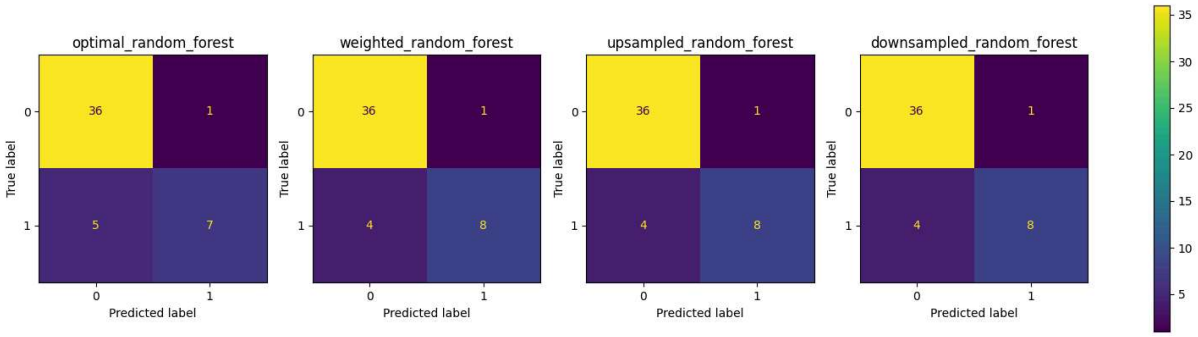
In [254…
```
###################################
# Plotting all the F1 score
# model performance measures
# between train and test sets
###################################
random_forest_performance_comparison_F1_plot = random_forest_performance_comparisc
random_forest_performance_comparison_F1_plot.set_xlim(0.00,1.00)
```

```
random_forest_performance_comparison_F1_plot.set_title("Model Comparison by F1 Sc
random_forest_performance_comparison_F1_plot.set_xlabel("F1 Score Performance")
random_forest_performance_comparison_F1_plot.set_ylabel("Random Forest Model")
random_forest_performance_comparison_F1_plot.grid(False)
random_forest_performance_comparison_F1_plot.legend(loc='center left', bbox_to_anc
for container in random_forest_performance_comparison_F1_plot.containers:
    random_forest_performance_comparison_F1_plot.bar_label(container, fmt='%.5f',
```



In [255…

```
###################################
# Plotting the confusion matrices
# for all the Random Forest models
###################################
classifiers = {"optimal_random_forest": optimal_random_forest,
               "weighted_random_forest": weighted_random_forest,
               "upsampled_random_forest": upsampled_random_forest,
               "downsampled_random_forest": downsampled_random_forest}

fig, axes = plt.subplots(1, 4, figsize=(20, 5))
for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cf_matrix)
    disp.plot(ax=axes[i], xticks_rotation=0)
    disp.ax_.grid(False)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()

fig.colorbar(disp.im_, ax=axes)
plt.show()
```



In [256…

```
###################################
# Consolidating all the
# Support Vector Machine
# model performance measures
###################################
support_vector_machine_performance_comparison = pd.concat([optimal_support_vector_
                                                           optimal_support_vector_machine_p
                                                           weighted_support_vector_machine_
                                                           weighted_support_vector_machine_
                                                           upsampled_support_vector_machine
                                                           upsampled_support_vector_machine
                                                           downsampled_support_vector_machi
                                                           downsampled_support_vector_machi
                                                          ignore_index=True)
print('Consolidated Support Vector Machine Model Performance on Train and Test Dat
display(support_vector_machine_performance_comparison)
```

```
Consolidated Support Vector Machine Model Performance on Train and Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.947368	optimal_support_vector_machine	train
1	Precision	0.960000	optimal_support_vector_machine	train
2	Recall	0.827586	optimal_support_vector_machine	train
3	F1	0.888889	optimal_support_vector_machine	train
4	AUROC	0.907911	optimal_support_vector_machine	train
5	Accuracy	0.857143	optimal_support_vector_machine	test
6	Precision	0.857143	optimal_support_vector_machine	test
7	Recall	0.500000	optimal_support_vector_machine	test
8	F1	0.631579	optimal_support_vector_machine	test
9	AUROC	0.736486	optimal_support_vector_machine	test
10	Accuracy	0.964912	weighted_support_vector_machine	train
11	Precision	0.962963	weighted_support_vector_machine	train
12	Recall	0.896552	weighted_support_vector_machine	train
13	F1	0.928571	weighted_support_vector_machine	train
14	AUROC	0.942394	weighted_support_vector_machine	train
15	Accuracy	0.877551	weighted_support_vector_machine	test
16	Precision	0.875000	weighted_support_vector_machine	test
17	Recall	0.583333	weighted_support_vector_machine	test
18	F1	0.700000	weighted_support_vector_machine	test
19	AUROC	0.778153	weighted_support_vector_machine	test
20	Accuracy	0.973684	upsampled_support_vector_machine	train
21	Precision	0.933333	upsampled_support_vector_machine	train
22	Recall	0.965517	upsampled_support_vector_machine	train
23	F1	0.949153	upsampled_support_vector_machine	train
24	AUROC	0.970994	upsampled_support_vector_machine	train
25	Accuracy	0.897959	upsampled_support_vector_machine	test
26	Precision	0.818182	upsampled_support_vector_machine	test
27	Recall	0.750000	upsampled_support_vector_machine	test
28	F1	0.782609	upsampled_support_vector_machine	test
29	AUROC	0.847973	upsampled_support_vector_machine	test

	metric_name	metric_value	model	set
30	Accuracy	0.956140	downsampled_support_vector_machine	train
31	Precision	0.928571	downsampled_support_vector_machine	train
32	Recall	0.896552	downsampled_support_vector_machine	train
33	F1	0.912281	downsampled_support_vector_machine	train
34	AUROC	0.936511	downsampled_support_vector_machine	train
35	Accuracy	0.897959	downsampled_support_vector_machine	test
36	Precision	0.888889	downsampled_support_vector_machine	test
37	Recall	0.666667	downsampled_support_vector_machine	test
38	F1	0.761905	downsampled_support_vector_machine	test
39	AUROC	0.819820	downsampled_support_vector_machine	test

In [257…
```
###################################
# Consolidating all the F1 score
# model performance measures
###################################
support_vector_machine_performance_comparison_F1 = support_vector_machine_performa
support_vector_machine_performance_comparison_F1_train = support_vector_machine_pe
support_vector_machine_performance_comparison_F1_test = support_vector_machine_per
```

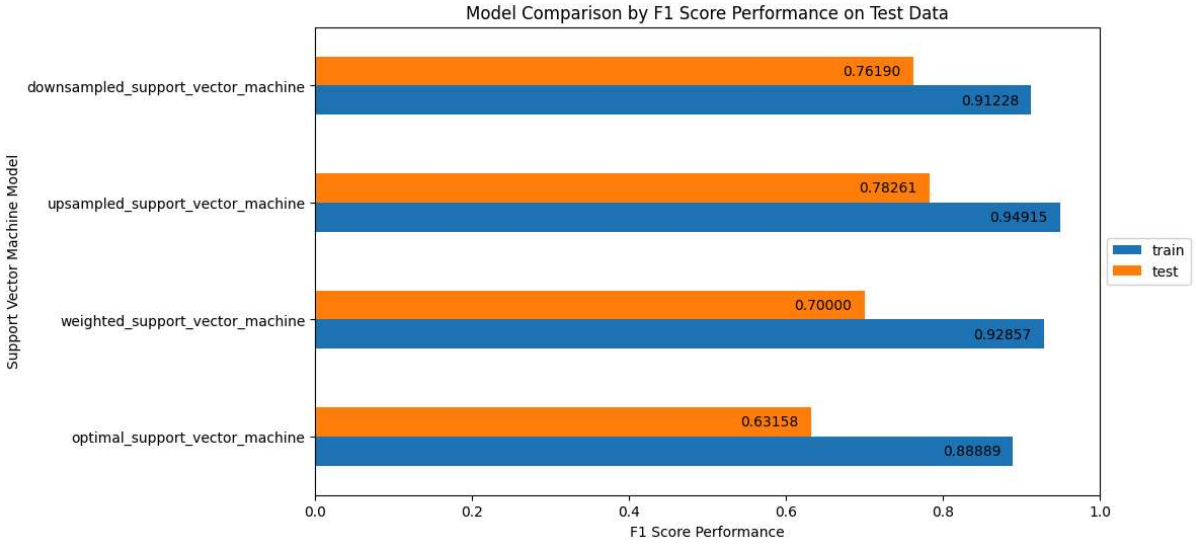
In [258…
```
###################################
# Combining all the F1 score
# model performance measures
# between train and test sets
###################################
support_vector_machine_performance_comparison_F1_plot = pd.DataFrame({'train': sup
                                                                      'test': supp
                                                                     index=support

support_vector_machine_performance_comparison_F1_plot
```

Out[258…

	train	test
optimal_support_vector_machine	0.888889	0.631579
weighted_support_vector_machine	0.928571	0.700000
upsampled_support_vector_machine	0.949153	0.782609
downsampled_support_vector_machine	0.912281	0.761905

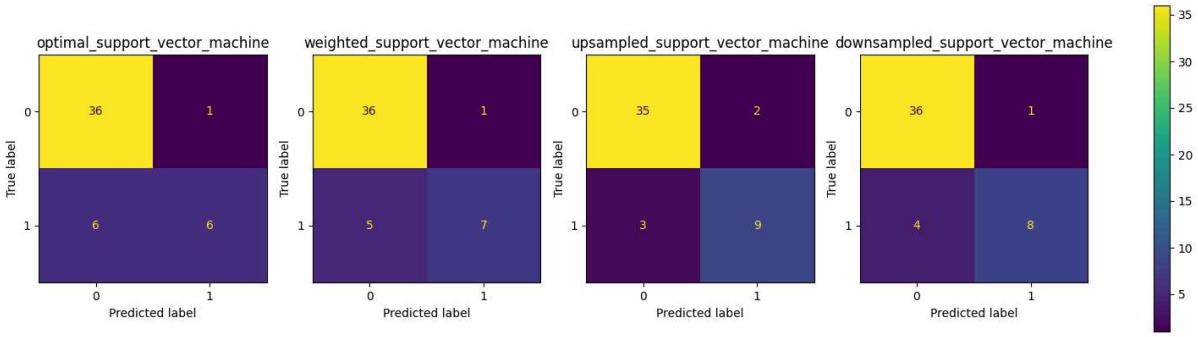
In [259…
```
###################################
# Plotting all the F1 score
# model performance measures
# between train and test sets
###################################
support_vector_machine_performance_comparison_F1_plot = support_vector_machine_per
support_vector_machine_performance_comparison_F1_plot.set_xlim(0.00,1.00)
```

```
support_vector_machine_performance_comparison_F1_plot.set_title("Model Comparison
support_vector_machine_performance_comparison_F1_plot.set_xlabel("F1 Score Perform
support_vector_machine_performance_comparison_F1_plot.set_ylabel("Support Vector M
support_vector_machine_performance_comparison_F1_plot.grid(False)
support_vector_machine_performance_comparison_F1_plot.legend(loc='center left', bb
for container in support_vector_machine_performance_comparison_F1_plot.containers:
    support_vector_machine_performance_comparison_F1_plot.bar_label(container, fmt
```



In [260…

```
###################################
# Plotting the confusion matrices
# for all the Support Vector Machine models
###################################
classifiers = {"optimal_support_vector_machine": optimal_support_vector_machine,
               "weighted_support_vector_machine": weighted_support_vector_machine,
               "upsampled_support_vector_machine": upsampled_support_vector_machir
               "downsampled_support_vector_machine": downsampled_support_vector_ma

fig, axes = plt.subplots(1, 4, figsize=(20, 5))
for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cf_matrix)
    disp.plot(ax=axes[i], xticks_rotation=0)
    disp.ax_.grid(False)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()

fig.colorbar(disp.im_, ax=axes)
plt.show()
```



1.10.2 Logistic Regression

Logistic Regression models the relationship between the probability of an event (among two outcome levels) by having the log-odds of the event be a linear combination of a set of predictors weighted by their respective parameter estimates. The parameters are estimated via maximum likelihood estimation by testing different values through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimates. Given the optimal parameters, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability.

1. The logistic regression model from the **sklearn.linear_model** Python library API was implemented as a meta-learner for the stacking algorithm.
2. The model used default hyperparameters with no tuning applied:
    - C = inverse of regularization strength held constant at a value of 1
    - penalty = penalty norm held constant at a value of L2
    - solver = algorithm used in the optimization problem held constant at a value of Lbfgs
    - class_weight = weights associated with classes held constant at a value of 25-75 between classes 0 and 1
    - max_iter = maximum number of iterations taken for the solvers to converge held constant at a value of 500
3. The original data reflecting a 3:1 class imbalance between the LOW and HIGH CANRAT categories was used for model training and testing.
4. The apparent model performance of the optimal model is summarized as follows:
    - **Accuracy** = 0.9736
    - **Precision** = 0.9062
    - **Recall** = 1.0000
    - **F1 Score** = 0.9508
    - **AUROC** = 0.9823
5. The independent test model performance of the final model is summarized as follows:
    - **Accuracy** = 0.9183
    - **Precision** = 0.9000

- **Recall** = 0.7500
- **F1 Score** = 0.8181
- **AUROC** = 0.8614

6. High difference in the apparent and independent test model performance observed, indicative of the presence of excessive model overfitting.

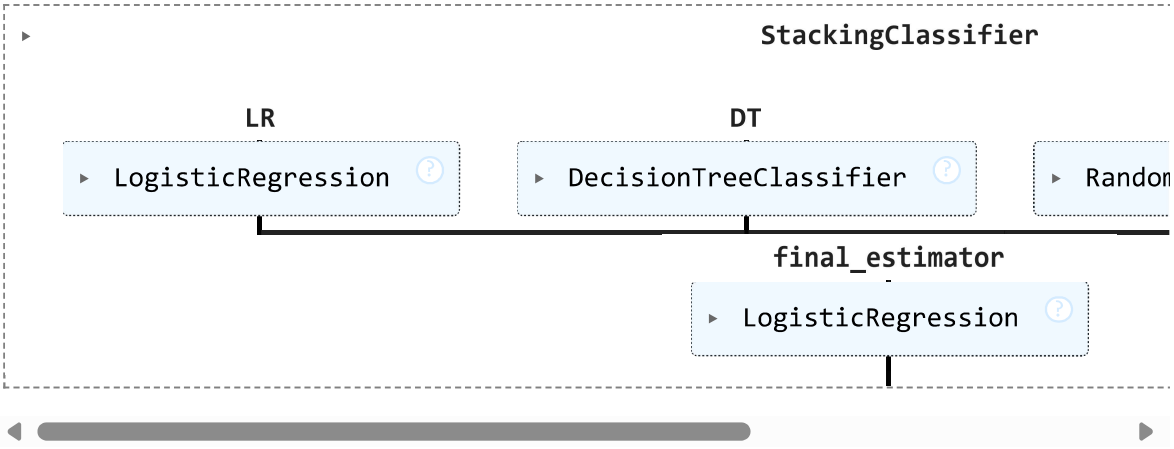
In [261…
```
##################################
# Formulating the base learners
# using the optimal hyperparameters
# for the upsampled models
##################################
base_learners = [('LR', LogisticRegression(C=1.0,
                                           class_weight=None,
                                           max_iter=500,
                                           penalty='l1',
                                           random_state=88888888,
                                           solver='saga')),
                 ('DT', DecisionTreeClassifier(class_weight=None,
                                               criterion='entropy',
                                               max_depth=3,
                                               min_samples_leaf=5,
                                               random_state=88888888)),
                 ('RF', RandomForestClassifier(class_weight=None,
                                               criterion='entropy',
                                               max_depth=7,
                                               max_features='sqrt',
                                               min_samples_leaf=3,
                                               n_estimators=100,
                                               random_state=88888888)),
                ('SVM', SVC(class_weight=None,
                            C=1.0,
                            kernel='linear',
                            random_state=88888888))]
```

In [262…
```
##################################
# Formulating the meta learner
# using default hyperparameters
##################################
meta_learner = LogisticRegression(C=1.0,
                                  class_weight=None,
                                  max_iter=500,
                                  random_state=88888888)
```

In [263…
```
##################################
# Formulating the stacked model
# using the base and meta learners
##################################
stacked_logistic_regression = StackingClassifier(estimators=base_learners, final_e
```

In [264…
```
##################################
# Fitting the meta Logistic Regression model
##################################
stacked_logistic_regression.fit(X_train_smote, y_train_smote)
```

Out[264...



In [265...

```
###################################
# Evaluating the stacked Logistic Regression model
# on the train set
###################################
stacked_logistic_regression_y_hat_train = stacked_logistic_regression.predict(X_tr
```

In [266...

```
###################################
# Gathering the model evaluation metrics
###################################
stacked_logistic_regression_performance_train = model_performance_evaluation(y_tra
stacked_logistic_regression_performance_train['model'] = ['stacked_logistic_regres
stacked_logistic_regression_performance_train['set'] = ['train'] * 5
print('Stacked Logistic Regression Model Performance on Train Data: ')
display(stacked_logistic_regression_performance_train)
```

```
Stacked Logistic Regression Model Performance on Train Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.973684	stacked_logistic_regression	train
1	Precision	0.933333	stacked_logistic_regression	train
2	Recall	0.965517	stacked_logistic_regression	train
3	F1	0.949153	stacked_logistic_regression	train
4	AUROC	0.970994	stacked_logistic_regression	train

In [267...

```
###################################
# Evaluating the stacked Logistic Regression model
# on the test set
###################################
stacked_logistic_regression_y_hat_test = stacked_logistic_regression.predict(X_tes
```

In [268...

```
###################################
# Gathering the model evaluation metrics
###################################
stacked_logistic_regression_performance_test = model_performance_evaluation(y_test
stacked_logistic_regression_performance_test['model'] = ['stacked_logistic_regress
stacked_logistic_regression_performance_test['set'] = ['test'] * 5
```

```
print('Stacked Logistic Regression Model Performance on Test Data: ')
display(stacked_logistic_regression_performance_test)
```

```
Stacked Logistic Regression Model Performance on Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.918367	stacked_logistic_regression	test
1	Precision	0.900000	stacked_logistic_regression	test
2	Recall	0.750000	stacked_logistic_regression	test
3	F1	0.818182	stacked_logistic_regression	test
4	AUROC	0.861486	stacked_logistic_regression	test

1.11. Consolidated Findings

1. Among the formulated versions of the logistic regression model, the model which applied upsampling of the minority class using SMOTE was used as a base learner for the model stacking algorithm.
    - **Accuracy** = 0.9183
    - **Precision** = 0.9000
    - **Recall** = 0.7500
    - **F1 Score** = 0.8181
    - **AUROC** = 0.8614
2. Among the formulated versions of the decision tree model, the model which applied upsampling of the minority class using SMOTE was used as a base learner for the model stacking algorithm.
    - **Accuracy** = 0.8979
    - **Precision** = 0.7692
    - **Recall** = 0.8333
    - **F1 Score** = 0.8000
    - **AUROC** = 0.8761
3. Among the formulated versions of the random forest model, the model which applied upsampling of the minority class using SMOTE was used as a base learner for the model stacking algorithm.
    - **Accuracy** = 0.9387
    - **Precision** = 0.8461
    - **Recall** = 0.9167
    - **F1 Score** = 0.8800
    - **AUROC** = 0.9313
4. Among the formulated versions of the support vector machine model, the model which applied upsampling of the minority class using SMOTE was used as a base learner for the model stacking algorithm.

- **Accuracy** = 0.8979
- **Precision** = 0.8181
- **Recall** = 0.7500
- **F1 Score** = 0.7826
- **AUROC** = 0.8479

5. The stacked logistic regression model comprised of the individual base learners demonstrated sufficient class discrmination:
   - **Accuracy** = 0.9183
   - **Precision** = 0.9000
   - **Recall** = 0.7500
   - **F1 Score** = 0.8181
   - **AUROC** = 0.8614
6. Comparing all results from the formulated base and stacked models formulated, the which applied class weights still demonstrated the best independent test model performance and was selected as the final model for classification.
   - **Accuracy** = 0.9387
   - **Precision** = 0.8461
   - **Recall** = 0.9167
   - **F1 Score** = 0.8800
   - **AUROC** = 0.9313

In [269…

```
###################################
# Consolidating all the
# base and meta-learner
# model performance measures
###################################
base_meta_learner_performance_comparison = pd.concat([weighted_logistic_regressior
                                                      weighted_logistic_regressior
                                                      upsampled_logistic_regressic
                                                      upsampled_logistic_regressic
                                                      upsampled_decision_tree_perf
                                                      upsampled_decision_tree_perf
                                                      upsampled_random_forest_perf
                                                      upsampled_random_forest_perf
                                                      upsampled_support_vector_mac
                                                      upsampled_support_vector_mac
                                                      stacked_logistic_regression_
                                                      stacked_logistic_regression_
                                                     ignore_index=True)
print('Consolidated Base and Meta Learner Model Performance on Train and Test Data
display(base_meta_learner_performance_comparison)
```

```
Consolidated Base and Meta Learner Model Performance on Train and Test Data:
```

	metric_name	metric_value	model	set
0	Accuracy	0.894737	weighted_logistic_regression	train
1	Precision	0.707317	weighted_logistic_regression	train
2	Recall	1.000000	weighted_logistic_regression	train
3	F1	0.828571	weighted_logistic_regression	train
4	AUROC	0.929412	weighted_logistic_regression	train
5	Accuracy	0.938776	weighted_logistic_regression	test
6	Precision	0.846154	weighted_logistic_regression	test
7	Recall	0.916667	weighted_logistic_regression	test
8	F1	0.880000	weighted_logistic_regression	test
9	AUROC	0.931306	weighted_logistic_regression	test
10	Accuracy	0.964912	upsampled_logistic_regression	train
11	Precision	0.903226	upsampled_logistic_regression	train
12	Recall	0.965517	upsampled_logistic_regression	train
13	F1	0.933333	upsampled_logistic_regression	train
14	AUROC	0.965112	upsampled_logistic_regression	train
15	Accuracy	0.918367	upsampled_logistic_regression	test
16	Precision	0.900000	upsampled_logistic_regression	test
17	Recall	0.750000	upsampled_logistic_regression	test
18	F1	0.818182	upsampled_logistic_regression	test
19	AUROC	0.861486	upsampled_logistic_regression	test
20	Accuracy	0.921053	upsampled_decision_tree	train
21	Precision	0.763158	upsampled_decision_tree	train
22	Recall	1.000000	upsampled_decision_tree	train
23	F1	0.865672	upsampled_decision_tree	train
24	AUROC	0.947059	upsampled_decision_tree	train
25	Accuracy	0.897959	upsampled_decision_tree	test
26	Precision	0.769231	upsampled_decision_tree	test
27	Recall	0.833333	upsampled_decision_tree	test
28	F1	0.800000	upsampled_decision_tree	test
29	AUROC	0.876126	upsampled_decision_tree	test

	metric_name	metric_value	model	set
30	Accuracy	0.973684	upsampled_random_forest	train
31	Precision	0.906250	upsampled_random_forest	train
32	Recall	1.000000	upsampled_random_forest	train
33	F1	0.950820	upsampled_random_forest	train
34	AUROC	0.982353	upsampled_random_forest	train
35	Accuracy	0.897959	upsampled_random_forest	test
36	Precision	0.888889	upsampled_random_forest	test
37	Recall	0.666667	upsampled_random_forest	test
38	F1	0.761905	upsampled_random_forest	test
39	AUROC	0.819820	upsampled_random_forest	test
40	Accuracy	0.973684	upsampled_support_vector_machine	train
41	Precision	0.933333	upsampled_support_vector_machine	train
42	Recall	0.965517	upsampled_support_vector_machine	train
43	F1	0.949153	upsampled_support_vector_machine	train
44	AUROC	0.970994	upsampled_support_vector_machine	train
45	Accuracy	0.897959	upsampled_support_vector_machine	test
46	Precision	0.818182	upsampled_support_vector_machine	test
47	Recall	0.750000	upsampled_support_vector_machine	test
48	F1	0.782609	upsampled_support_vector_machine	test
49	AUROC	0.847973	upsampled_support_vector_machine	test
50	Accuracy	0.973684	stacked_logistic_regression	train
51	Precision	0.933333	stacked_logistic_regression	train
52	Recall	0.965517	stacked_logistic_regression	train
53	F1	0.949153	stacked_logistic_regression	train
54	AUROC	0.970994	stacked_logistic_regression	train
55	Accuracy	0.918367	stacked_logistic_regression	test
56	Precision	0.900000	stacked_logistic_regression	test
57	Recall	0.750000	stacked_logistic_regression	test
58	F1	0.818182	stacked_logistic_regression	test
59	AUROC	0.861486	stacked_logistic_regression	test

In [270…

```
###################################
# Consolidating all the F1 score
# model performance measures
###################################
base_meta_learner_performance_comparison_F1 = base_meta_learner_performance_compar
base_meta_learner_performance_comparison_F1_train = base_meta_learner_performance_
base_meta_learner_performance_comparison_F1_test = base_meta_learner_performance_c
```

In [271…

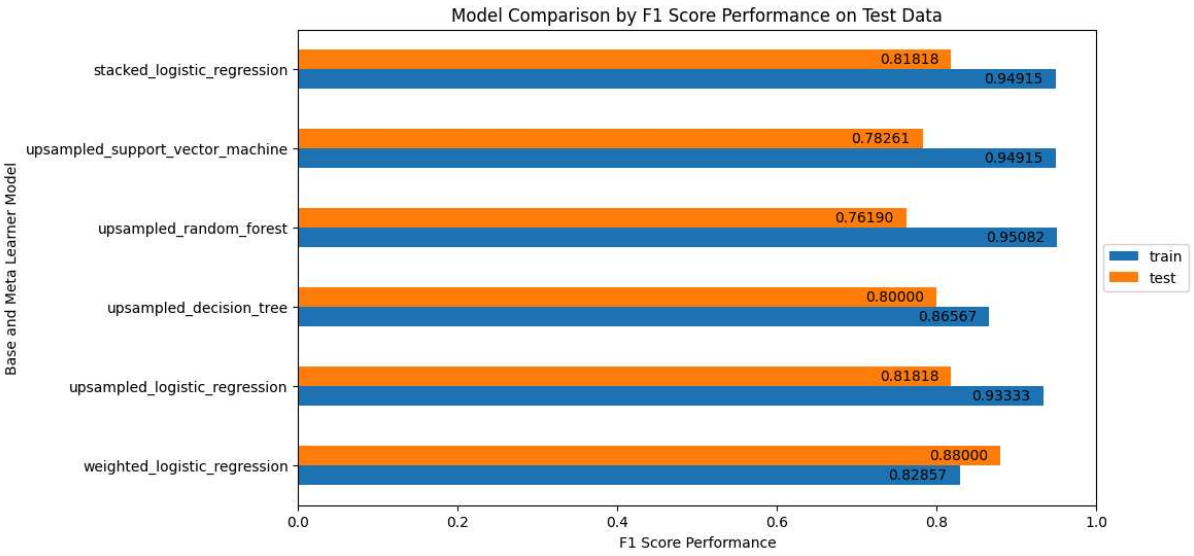
```
###################################
# Combining all the F1 score
# model performance measures
# between train and test sets
###################################
base_meta_learner_performance_comparison_F1_plot = pd.DataFrame({'train': base_met
                                                                 'test': base_meta
                                                                index=base_meta_le
base_meta_learner_performance_comparison_F1_plot
```

Out[271…

	train	test
weighted_logistic_regression	0.828571	0.880000
upsampled_logistic_regression	0.933333	0.818182
upsampled_decision_tree	0.865672	0.800000
upsampled_random_forest	0.950820	0.761905
upsampled_support_vector_machine	0.949153	0.782609
stacked_logistic_regression	0.949153	0.818182

In [272…

```
###################################
# Plotting all the F1 score
# model performance measures
# between train and test sets
###################################
base_meta_learner_performance_comparison_F1_plot = base_meta_learner_performance_c
base_meta_learner_performance_comparison_F1_plot.set_xlim(0.00,1.00)
base_meta_learner_performance_comparison_F1_plot.set_title("Model Comparison by F1
base_meta_learner_performance_comparison_F1_plot.set_xlabel("F1 Score Performance"
base_meta_learner_performance_comparison_F1_plot.set_ylabel("Base and Meta Learner
base_meta_learner_performance_comparison_F1_plot.grid(False)
base_meta_learner_performance_comparison_F1_plot.legend(loc='center left', bbox_to
for container in base_meta_learner_performance_comparison_F1_plot.containers:
    base_meta_learner_performance_comparison_F1_plot.bar_label(container, fmt='%.5
```



In [273…

```
###################################
# Consolidating all score
# model performance measures
###################################
base_meta_learner_performance_comparison_Accuracy_test = base_meta_learner_perform
base_meta_learner_performance_comparison_Precision_test = base_meta_learner_perfor
base_meta_learner_performance_comparison_Recall_test = base_meta_learner_performar
base_meta_learner_performance_comparison_F1_test = base_meta_learner_performance_c
base_meta_learner_performance_comparison_AUROC_test = base_meta_learner_performanc
```

In [274…

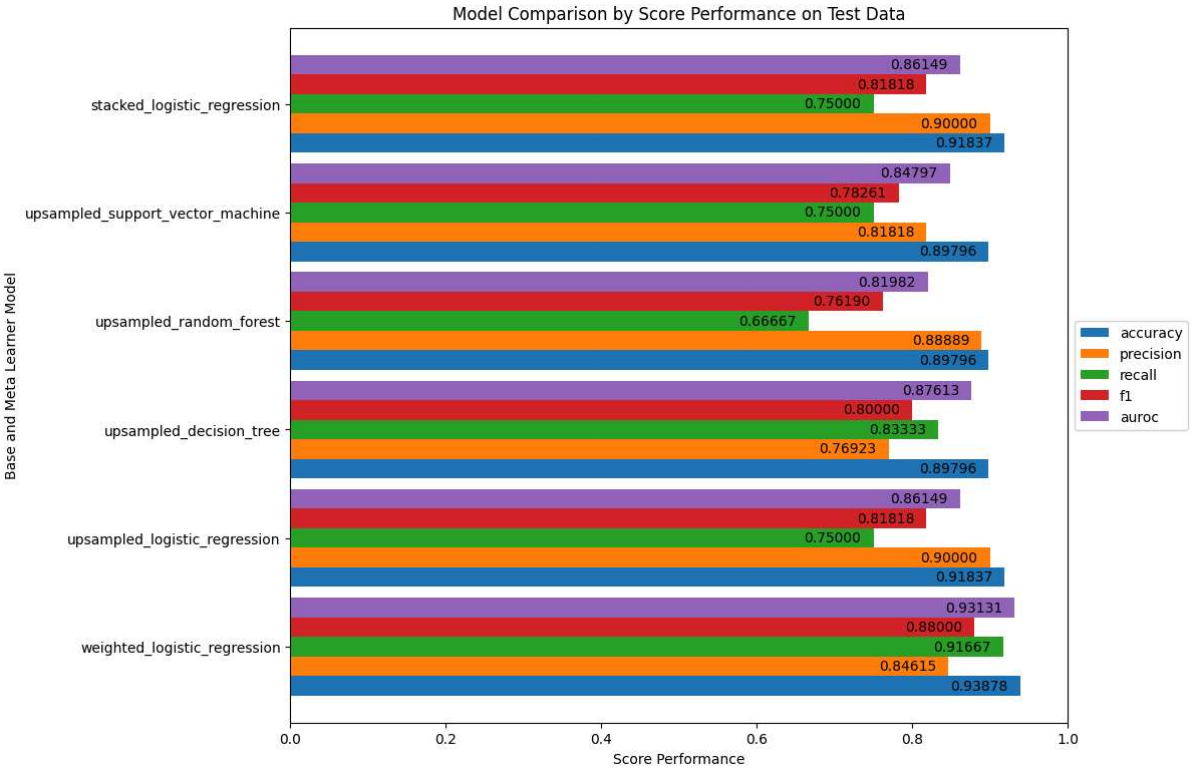
```
###################################
# Combining all the score
# model performance measures
# between train and test sets
###################################
base_meta_learner_performance_comparison_all_plot = pd.DataFrame({'accuracy': base
                                                                  'precision': bas
                                                                  'recall': base_m
                                                                  'f1': base_meta_
                                                                  'auroc': base_me
                                                                 index=base_meta_le

base_meta_learner_performance_comparison_all_plot
```

Out[274…

	accuracy	precision	recall	f1	auroc
weighted_logistic_regression	0.938776	0.846154	0.916667	0.880000	0.931306
upsampled_logistic_regression	0.918367	0.900000	0.750000	0.818182	0.861486
upsampled_decision_tree	0.897959	0.769231	0.833333	0.800000	0.876126
upsampled_random_forest	0.897959	0.888889	0.666667	0.761905	0.819820
upsampled_support_vector_machine	0.897959	0.818182	0.750000	0.782609	0.847973
stacked_logistic_regression	0.918367	0.900000	0.750000	0.818182	0.861486

In [275…

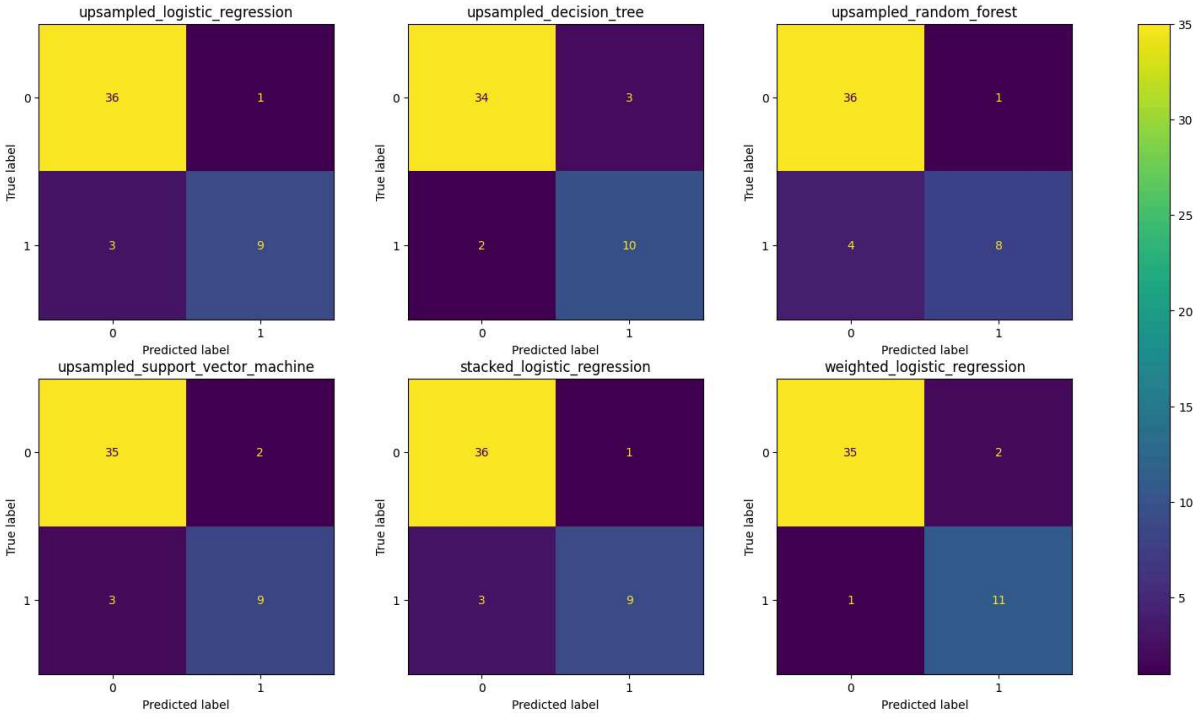
```
###################################
# Plotting all the score
# model performance measures
# between train and test sets
###################################
base_meta_learner_performance_comparison_all_plot = base_meta_learner_performance_
base_meta_learner_performance_comparison_all_plot.set_xlim(0.00,1.00)
base_meta_learner_performance_comparison_all_plot.set_title("Model Comparison by S
base_meta_learner_performance_comparison_all_plot.set_xlabel("Score Performance")
base_meta_learner_performance_comparison_all_plot.set_ylabel("Base and Meta Learne
base_meta_learner_performance_comparison_all_plot.grid(False)
base_meta_learner_performance_comparison_all_plot.legend(loc='center left', bbox_t
for container in base_meta_learner_performance_comparison_all_plot.containers:
    base_meta_learner_performance_comparison_all_plot.bar_label(container, fmt='%.
```



In [276…

```
###################################
# Plotting the confusion matrices
# for all the Support Vector Machine models
###################################
classifiers = {"upsampled_logistic_regression": upsampled_logistic_regression,
               "upsampled_decision_tree": upsampled_decision_tree,
               "upsampled_random_forest": upsampled_random_forest,
               "upsampled_support_vector_machine": upsampled_support_vector_machir
               "stacked_logistic_regression": stacked_logistic_regression,
               "weighted_logistic_regression": weighted_logistic_regression,}

fig, axes = plt.subplots(2, 3, figsize=(20, 10))
axes = axes.ravel()
for i, (key, classifier) in enumerate(classifiers.items()):
    y_pred = classifier.predict(X_test)
    cf_matrix = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cf_matrix)
```

```
    disp.plot(ax=axes[i], xticks_rotation=0)
    disp.ax_.grid(False)
    disp.ax_.set_title(key)
    disp.im_.colorbar.remove()

fig.colorbar(disp.im_, ax=axes)
plt.show()
```

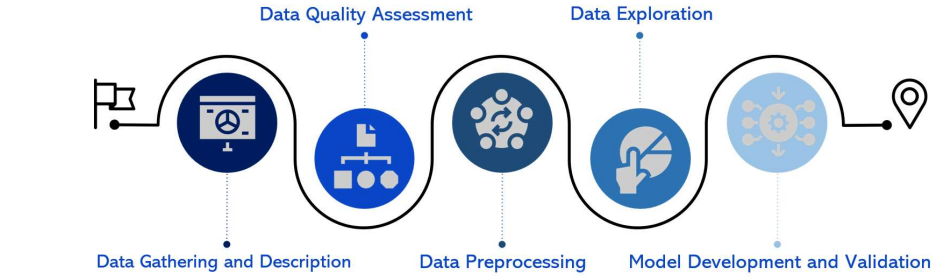


2. Summary

INTRODUCTION

- Cancer – a complex group of diseases characterized by the uncontrolled growth and spread of abnormal cells, has emerged as a leading cause of morbidity and mortality worldwide.
- While the prevalence of cancer varies significantly between countries, anticipating elevated cancer categories given the factors that contribute to these variations is crucial for effective prevention, early detection, and treatment strategies.
- This capstone project generally aims to develop a classification model that could provide robust and reliable estimates of cancer category probabilities from an optimal set of observations and predictors, while delivering accurate predictions when applied to new unseen data.
    - In particular, multiple classification models will be formulated with remedial actions applied to address class imbalance. The final classification model will be selected among candidates based on robust performance estimates. The final model performance and generalization ability will be evaluated through external validation in an independent set.

METHODOLOGY



Data Gathering and Description

- Source assessment
- Data download
- Variable description
- Data dimension
- Column types
- Numeric statistics
- Categorical statistics

Data Quality Assessment

- Row duplicates
- Column fill rate
- Row fill rate
- Low variance
- High data skew

Data Preprocessing

- Data cleaning
- Data imputation
- Outlier treatment
- Collinearity
- Transformation
- Centering and scaling
- Data Encoding

Data Exploration

- Visual exploration
- Hypothesis testing

Model Development and Validation

- Model development
- Hyperparameter tuning
- Class weighting
- Upsampling (SMOTE)
- Downsampling (CNN)
- Model validation
- Model stacking
- Model selection

RESULTS – DATA GATHERING

World Performance Indicators
Social Protection and Labor [Source: World Bank]
Education [Source: World Bank]
Economy and Growth [Source: World Bank]
Environment [Source: World Bank]
Climate Change [Source: World Bank]
Agricultural and Rural Development [Source: World Bank]
Social Development [Source: World Bank]
Health [Source: World Bank]
Science and Technology [Source: World Bank]
Urban Development [Source: World Bank]

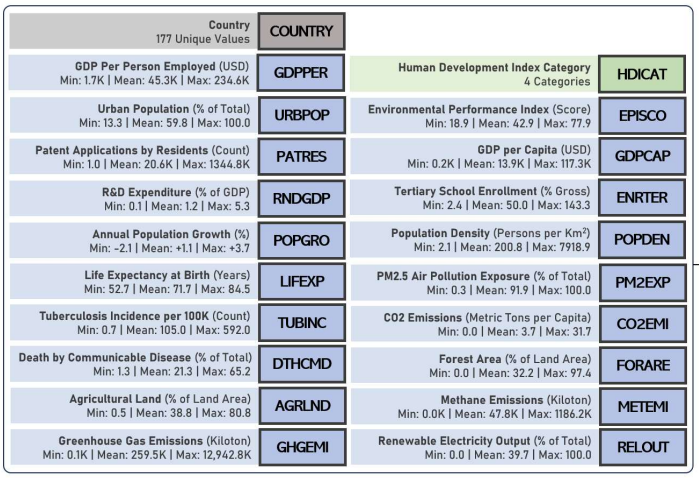
World Performance Indices
Human Development [Source: Human Development Reports]
Environmental Performance [Source: Yale Center]

- The study hypothesizes that **world performance indicators** (social protection and labor, education, economy and growth, environment, climate change, agricultural and rural development, social development, health, science and technology, urban development) and **indices** (human development, environmental performance) directly influence **cancer rates** across countries.

World Cancer Rates
[Source: World Population Review]

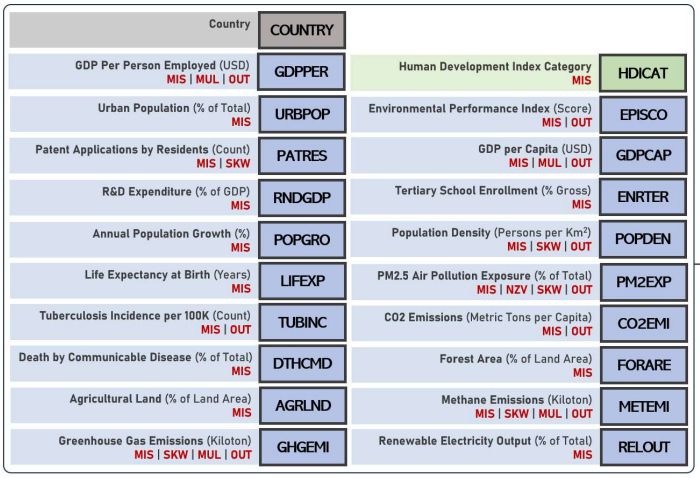
- The objective of the study is to explore and prepare the dataset to determine which among the **world performance indicators** and **indices** are the significant key drivers of **cancer rates** across countries.

RESULTS – DATA DESCRIPTION



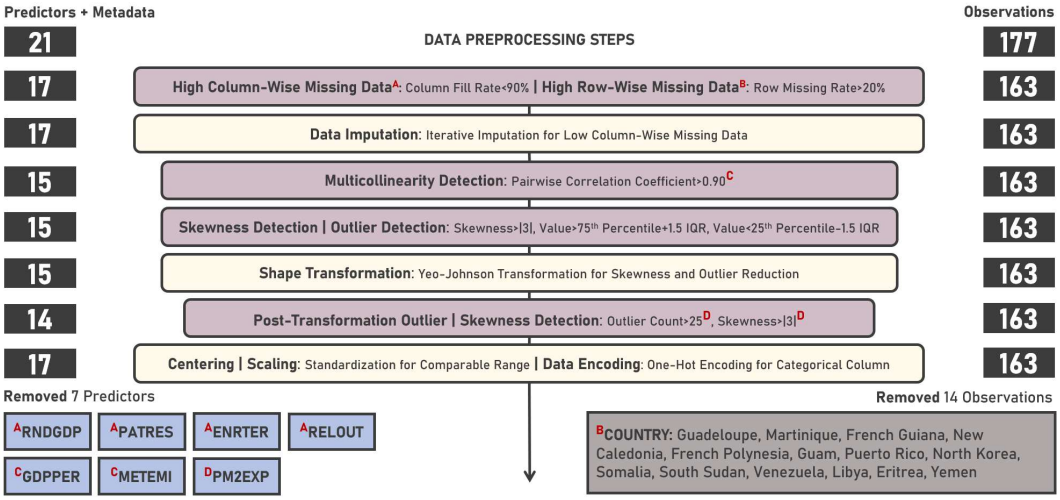
- Original data consisted of:
  - 177 observation rows
  - 1 categorical response column
  - 19 numeric predictor columns
  - 1 categorical predictor column
  - 1 object metadata column
- Numeric data are of different scales.

RESULTS – DATA QUALITY ASSESSMENT

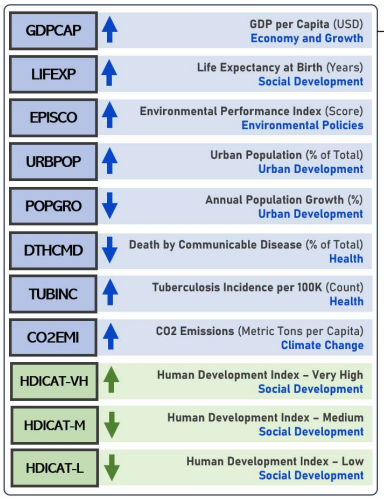


- Data quality issues identified included:
  - Missing data
  - Skewed distributions
  - Near zero variance
  - High multicollinearity
  - High outlier ratio
- Data preprocessing is needed to address data quality issues.

RESULTS – DATA PREPROCESSING



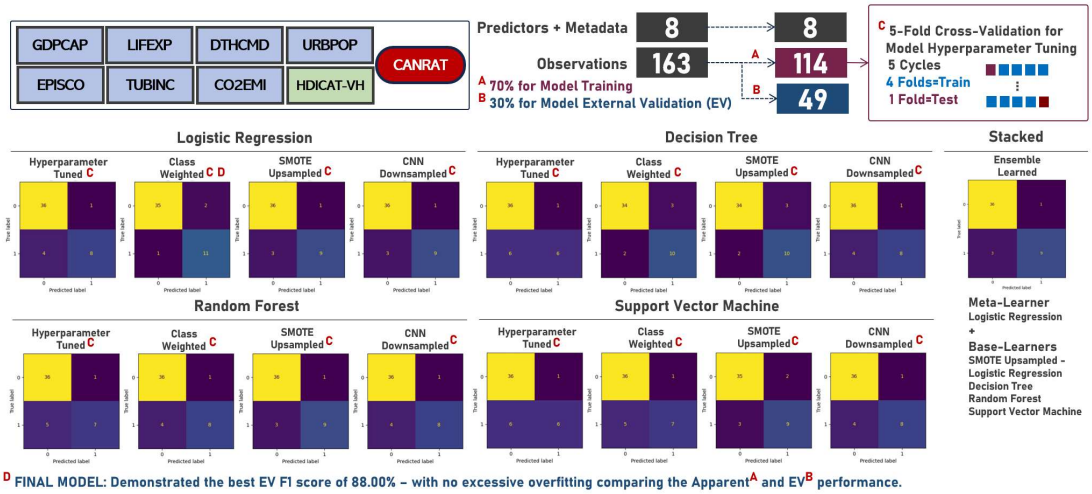
RESULTS – DATA EXPLORATION



Statistically significant associations with predictors were determined.

- Cancer rates categorized as high were generally observed among progressive countries as characterized by the following:
  - Higher economic growth leading to advanced industrialization with increased exposure to pollutants but potentially better environmental policies
  - Better socioeconomic policies to support cancer screening and diagnosis
  - More robust healthcare infrastructure to support the completeness of cancer registries and reporting mechanisms
  - Shifting demographics including a potentially aging population with increased disease incidence trends
- Given the findings, a high level of progressiveness may not inherently imply higher cancer rates; rather, the relationship might have been influenced by a combination of demographic, environmental, and healthcare-related factors associated with developed countries.
- The relationship between a country's level of progressiveness and cancer rates may not be straightforward, and other potential drivers must be considered in the future when exploring this issue including data on lifestyle factors (smoking rates, obesity levels) or genetic diversity (infectious agents).

RESULTS – MODEL DEVELOPMENT



D FINAL MODEL: Demonstrated the best EV F1 score of 88.00% – with no excessive overfitting comparing the Apparent[A] and EV[B] performance.

OVERALL FINDINGS AND IMPLICATIONS

• Key Findings

- Using the F1 score, the best-performing **logistic regression model** applied **class weights**.
- The best-performing **decision tree**, **random forest**, and **support vector machine** models were those applied with **upsampling using SMOTE**, as compared to the other evaluated remedial measures to address class imbalance including **hyperparameter tuning**, **class weights**, and **downsampling using CNN**.
- Using the individual models that applied **upsampling using SMOTE** as base-learners for a stacking ensemble that used logistic regression as a meta-learner demonstrated good discrimination but was still inferior to the **logistic regression model** which applied **class weights**.

• Overall Implications

- The final classification model using the **logistic regression model** which applied **class weights** could provide robust and reliable estimates of cancer categories based on the estimated metrics as follows.
  - Accuracy = 93.87%
  - Precision = 84.61%
  - Recall = 91.67%
  - F1 Score = 88.00%
  - AUROC = 0.9313

CONCLUSION

- Overall Summary
  - Data collection for the analysis involved world performance indicators and indices hypothesized to be directly influencing cancer rates across countries.
  - The quality of gathered data was assessed and potential issues were identified.
  - Appropriate pre-processing methods including remedial procedures to address duplicate, missing, outlying, and non-normalized data were applied to prepare the data for subsequent analysis. Additional data scaling and transformation were implemented.
  - EDA using visualization presented the various distributions and comparisons between the indicators and indices as evaluated against cancer rate categories – eventually identifying the key drivers of higher cancer rates. Statistical hypothesis testing identified the individual drivers that were significantly associated with cancer rate categories.
  - A final classification model was selected among candidates that could provide robust and reliable probability estimates of cancer rate categories from an optimal set of observations and predictors.
  - Overall analysis findings were discussed and their practical implications were highlighted.

In [277…

```
from IPython.display import display, HTML
display(HTML("<style>.rendered_html { font-size: 15px; font-family: 'Trebuchet MS'
```