Import packages

In [47]:
```
from EZpreprocess import PreProcessor, RowSelector, Scaler

import numpy as np
import warnings
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.cluster import hierarchy
from scipy.stats import spearmanr
from collections import defaultdict
from patsy import dmatrices
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor


from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyRegressor, DummyClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm

from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve, roc_auc_score
import heapq


from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.manifold import TSNE
import xgboost as xgb

from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import RFECV
from sklearn.inspection import permutation_importance
from sklearn.feature_selection import SequentialFeatureSelector

from sklearn.model_selection import GridSearchCV

import shap
```

Preprocessing

In [9]:
```
def make_response_class(df, response_var, thresholds):

    """
    Change respose variable to binary class

    Args:
        df: dataset (pd.DataFrame)
        response_var: response variable name (str)
        threshold: a list of thresholds (float)

    Returns:
        modified df (pd.DataFrame)

    """

    na_vals = df[response_var].isna()
    for i,val in enumerate(df[response_var]):
        class_num = 0
        for k in range(len(thresholds)):
            if val > thresholds[k]:
                class_num +=1
            else:
                break
        df[response_var].iloc[i] = class_num
    df[response_var].iloc[na_vals] = np.nan

    return df
```

In [18]:
```
raw_file_path = r"gap718_LABELENC_CONTINUOUS.csv"
prefilttered_file_path = r"gap718_LABELENC_PREFILTER_CANCER_CODED.csv"
feature_file_path = r"Feature_Selection_Result.csv"
```

In [19]:
```
warnings.filterwarnings("ignore")
df = pd.read_csv(raw_file_path)
df = df.iloc[:,2:]
df = make_response_class(df, "rdi", [0.65])
df.head()
```

Out[19]:

	rdi	AdItotal	CalcBOMCTotal	Age	Gender	agecat	racecat	educat	maritalcat	livec
0	0.0	4.0	8	84.0	1.0	1.0	Others	2.0	Married/Domestic Partnership	Livi w parer
1	0.0	6.0	0	70.0	1.0	0.0	Non Hispanic White	2.0	Married/Domestic Partnership	Livi w parer
2	1.0	6.0	0	76.0	0.0	0.0	Non Hispanic White	0.0	Separated/ Widowed/ Divorced	LiveAlo
3	1.0	5.0	7	71.0	0.0	0.0	Non Hispanic White	1.0	Married/Domestic Partnership	Livi w parer
4	NaN	6.0	2	76.0	0.0	0.0	Non Hispanic White	2.0	Married/Domestic Partnership	Livi w parer

5 rows × 120 columns

Split train, test

In [20]:
```
#Code to be added
myRowSelector =  RowSelector(df=df, response_column="rdi", random_state=0)
train_df, test_df = myRowSelector.train_test_split(test_size = 0.2)
```

Split k-fold cross validation

In [21]:
```
myFolds = myRowSelector.kfold_split(n_fold = 5, df = train_df)
```

Define util functions

In [22]:
```
def cal_mean_2se(cv_list):

    """
    Calculate mean and 2 standard error of cross validation results on multiple p

    Args:
        cv_list: list of list of predictions. Example [[Fold_1_accuracies on 24 p
        [Fold_2_accuracies on 24 pipelines], [Fold_3_accuracies on 24 pipelines]]

    Returns:
        2 list: a list of means of different pipelines and a list of 2 Standard e
     """
    results = []
    error = []
    for i in np.array(cv_list).T:
        results.append(np.mean(i))
        error.append(2*np.std(i)/np.sqrt(5))
    return results, error
```

In [23]:
```
def plot_line(ax, lab, val_list, err_list):
    """
    Plot accuracies result

    Args:
        ax: axis object
        lab: name of model
        val_list: means of each pipeline
        err_list: 2 Standard error of each pipline

    Returns:
        plot k-fold cross validation result of 24 pipelines
     """
    ax.errorbar(np.arange(len(val_list)) ,y = val_list, yerr = err_list, label =
             elinewidth=0.5, capsize=5, markeredgewidth=0.1, alpha = 1,linestyle
    ax.set_xticks(np.arange(24))
    ax.legend()
```

In [24]:
```
def bound(pred):


    """
    Make sure prediction value is in range [0,1], Values abov 1 will be 1 and bel

    Args:
        pred: prediction result (list)

    Returns:
        modified values within range [0,1]
     """

    bound_pred = []
    for i in pred:
        if i < 0:
            bound_pred.append(0.)
        elif i > 1:
            bound_pred.append(1.)
        else:
            bound_pred.append(i)
    return bound_pred
```

Iterate through multiple preprocessing pipelines

In [25]:
```
dummy_cls = DummyClassifier(strategy= "most_frequent")
nb = GaussianNB()
rf_cls = RandomForestClassifier()
lr = LogisticRegression()

model_result_train = []
model_result_val = []
for m, model in enumerate((dummy_cls, nb, lr, rf_cls)):
    print(m, model)
    model_result_train.append([])
    model_result_val.append([])

    accuracy_train_results = []
    accuracy_val_results = []
    # Iterate through each fold
    for fold, (train, val) in enumerate(myFolds):
        accuracy_train_results.append([])
        accuracy_val_results.append([])

        #Specify pipeline
        for pl in PreProcessor.get_all_combinations(drop = ["PCA", "ICA", "Kernel

            myPreprocessor = PreProcessor(
                pipeline= pl,
                response_var = "rdi"
        )


            # fit pipeline to train split
            myPreprocessor.fit(train)
            # transform train set
            preprocessed_train = myPreprocessor.transform(train)
            # transform validation set
            preprocessed_val = myPreprocessor.transform(val, val = True)

            preprocessed_val = make_response_class(preprocessed_val, "rdi", [0.65
            preprocessed_train = make_response_class(preprocessed_train, "rdi", [

            X_train, y_train = preprocessed_train.loc[:, preprocessed_train.colum
            X_val, y_val = preprocessed_val.loc[:, preprocessed_val.columns != "r

            model.fit(X_train, y_train)

            train_predictions = model.predict(X_train)
            train_predictions = bound(train_predictions)


            val_predictions = model.predict(X_val)
            val_predictions = bound(val_predictions)

            accuracy_train = accuracy_score(train_predictions, y_train)
            accuracy_val = accuracy_score(val_predictions, y_val)

            accuracy_train_results[fold].append(accuracy_train)
            accuracy_val_results[fold].append(accuracy_val)
```

```
        model_result_train[m].append(accuracy_train_results)
        model_result_val[m].append(accuracy_val_results)
```

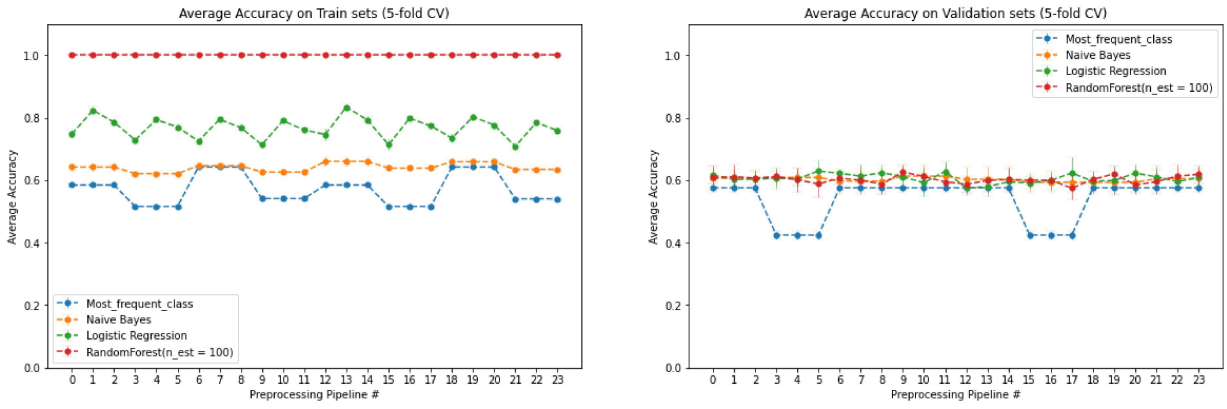
```
0 DummyClassifier(strategy='most_frequent')
1 GaussianNB()
2 LogisticRegression()
3 RandomForestClassifier()
```

Plot 5 fold cv result

In [26]:
```
name = ["Most_frequent_class", "Naive Bayes", "Logistic Regression", "RandomFores
fig = plt.figure(figsize = (20,6))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
ax1.set_title("Average Accuracy on Train sets (5-fold CV)")
ax2.set_title("Average Accuracy on Validation sets (5-fold CV)")
ax1.set_xlabel("Preprocessing Pipeline #")
ax2.set_xlabel("Preprocessing Pipeline #")
ax1.set_ylabel("Average Accuracy")
ax2.set_ylabel("Average Accuracy")
for i, mod in enumerate(model_result_train):
    val, err = cal_mean_2se(mod)
    ax1.set_ylim(0.,1.1)
    plot_line(ax1,name[i],val,err)

plt.legend()
for i, mod in enumerate(model_result_val):
    val, err = cal_mean_2se(mod)
    ax2.set_ylim(0.,1.1)
    plot_line(ax2,name[i],val,err)
plt.legend()
```

```
No handles with labels found to put in legend.
```

Out[26]:
```
<matplotlib.legend.Legend at 0x289dfea9400>
```



Preprocess train and test data

In [27]:
```
myPreprocessor = PreProcessor(
    pipeline= {
         "Encoder": "OneHot",
         "Imputer": "Median",
         "Scaler": "MinMax",
         "FeatureSelection": "All",
         "DimReduction": "Skip"
    },
    response_var = "rdi"
)

#Code to be added
myRowSelector =  RowSelector(df=df, response_column="rdi", random_state=0)
train_df, test_df = myRowSelector.train_test_split(test_size = 0.2)
# fit pipeline to train split
myPreprocessor.fit(train_df)
# transform train set
preprocessed_train = myPreprocessor.transform(train_df)
X_train, y_train = preprocessed_train.iloc[:,1:], preprocessed_train["rdi"]
# transform train set
preprocessed_test = myPreprocessor.transform(test_df)
X_test, y_test = preprocessed_test.iloc[:,1:], preprocessed_test["rdi"]
```

EDA

In [28]:
```
def get_high_vif_features(df, threshold):

    """
    Obtain features has VIF higer than a threshold
    using sequential dropping method


    Args:
        df: X matrix dataset (pd.DataFrame)
        threshold: features with VIF higher than this number will be dropped (flo


    Returns:
        list of dropped features(list)

    """

    cols = df.columns
    variables = np.arange(df.shape[1])
    dropped=True
    drop_list = []
    while dropped:
        print(drop_list)
        dropped=False
        c = df[cols[variables]].values
        vif = [variance_inflation_factor(c, ix) for ix in np.arange(c.shape[1])]

        maxloc = vif.index(max(vif))
        if max(vif) > threshold:
            drop_list.append(df[cols[variables]].columns[maxloc])
            variables = np.delete(variables, maxloc)
            dropped=True

    return drop_list
```

Highly correlated features

In [29]:
```
# Takes about 10 mins to run
highly_correlated_features = get_high_vif_features(X_train, 5)
```

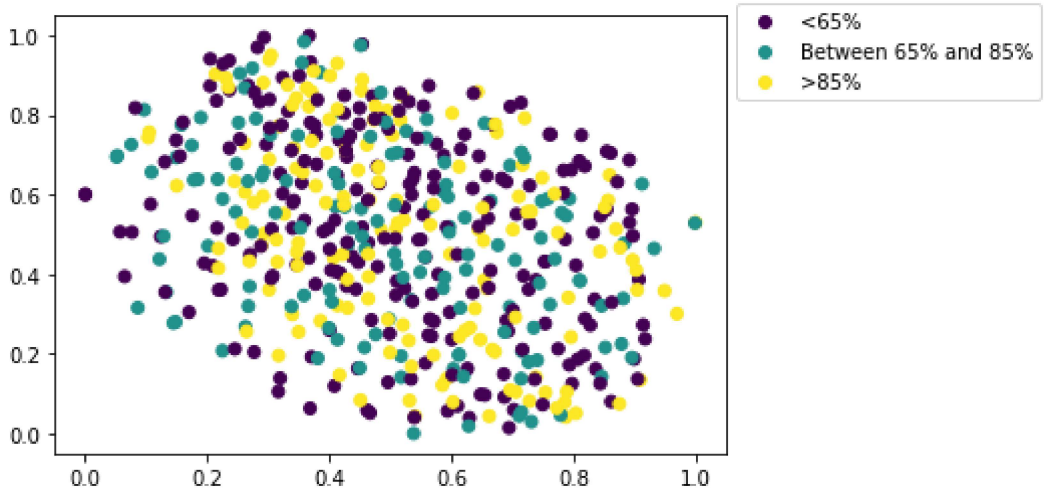
```
entrationSev', 'SOBSev', 'sure_total', 'AppetiteSev', 'PainSev', 'MemorySev',
'TasteSev', 'treatment_type', 'Age', 'InsomniaSev', 'NumbnessSev', 'Comorbidi
tysumCALC', 'MouthSoresSev', 'DizzinessSev', 'FatigueYN', 'NauseaSev', 'stag
e', 'VisionSev', 'livecat_Living with parents', 'SPPBTotalSumcalc', 'ArmSwell
ingSev', 'Weight6MonthsAgo', 'HeadachesSev', 'DryMouthYN']
['calcmss', 'GI_Lung', 'CurrentWeight', 'cognition', 'calcmnascore', 'iadltot
al', 'chemo', 'KPS', 'BMI', 'Adltotal', 'Week_practice', 'Age_Phy', 'BMIRang
e', 'FatigueSev', 'Physical_performance', 'racecat_Non Hispanic White', 'Conc
entrationSev', 'SOBSev', 'sure_total', 'AppetiteSev', 'PainSev', 'MemorySev',
'TasteSev', 'treatment_type', 'Age', 'InsomniaSev', 'NumbnessSev', 'Comorbidi
tysumCALC', 'MouthSoresSev', 'DizzinessSev', 'FatigueYN', 'NauseaSev', 'stag
e', 'VisionSev', 'livecat_Living with parents', 'SPPBTotalSumcalc', 'ArmSwell
ingSev', 'Weight6MonthsAgo', 'HeadachesSev', 'DryMouthYN', 'HandFootYN']
['calcmss', 'GI_Lung', 'CurrentWeight', 'cognition', 'calcmnascore', 'iadltot
al', 'chemo', 'KPS', 'BMI', 'Adltotal', 'Week_practice', 'Age_Phy', 'BMIRang
e', 'FatigueSev', 'Physical_performance', 'racecat_Non Hispanic White', 'Conc
entrationSev', 'SOBSev', 'sure_total', 'AppetiteSev', 'PainSev', 'MemorySev',
'TasteSev', 'treatment_type', 'Age', 'InsomniaSev', 'NumbnessSev', 'Comorbidi
tysumCALC', 'MouthSoresSev', 'DizzinessSev', 'FatigueYN', 'NauseaSev', 'stag
```

In [30]:
```
def plot_tSNE(df, response_var, pca_components_num):

    """
    Plot plot_tSNE using certain number of principle components with labeled clas


    Args:
        df: dataset (pd.DataFrame)
        response_var: response variable name (str)
        pca_components_num: number of PCA components used (int)

    Returns:
        t_SNE plot with labeled classes

    """

    rdi = df[response_var]
    rdi = rdi.fillna(-1)

    X = df.drop(response_var, axis = 1)

    labels = ['<65%', 'Between 65% and 85%', '>85%']

    rdicat = pd.cut(
    rdi,
    [0, 0.65, 0.85, np.inf],
    labels=labels
    )

    rdi_df = pd.concat([rdi, rdicat.rename('rdicat')], axis=1)
    rdi_df.head()

    rdi_exists = rdi_df[rdicat!='Missing']
    rdi_exists.head()



    #PCA
    pca = PCA(n_components = pca_components_num)
    pca_fit = pca.fit(X)
    pca_fit.explained_variance_ratio_.cumsum()
    new_feat = pca_fit.transform(X) #Transformed df

    #TSNE
    tsne = TSNE(n_components=2, perplexity = 40, random_state=0).fit_transform(ne

    tx = tsne[:, 0]
    ty = tsne[:, 1]

    #TX
    value_range = (np.max(tx) - np.min(tx))
    starts_from_zero = tx - np.min(tx)
    tx = starts_from_zero / value_range

    #TY
    value_range = (np.max(ty) - np.min(ty))
```

```
    starts_from_zero = ty - np.min(ty)
    ty = starts_from_zero / value_range

    rdi_new = rdi_exists.rdicat.replace({'<65%': 0, 'Between 65% and 85%': 1, '>8
    rdi_new = rdi_new.values.ravel()

    #Plot
    scatter = plt.scatter(tx, ty, c=rdi_new)
    handles, _ = scatter.legend_elements(prop='colors')
    plt.legend(handles, labels, bbox_to_anchor=(1, 1.05))
```

Plot t-SNE

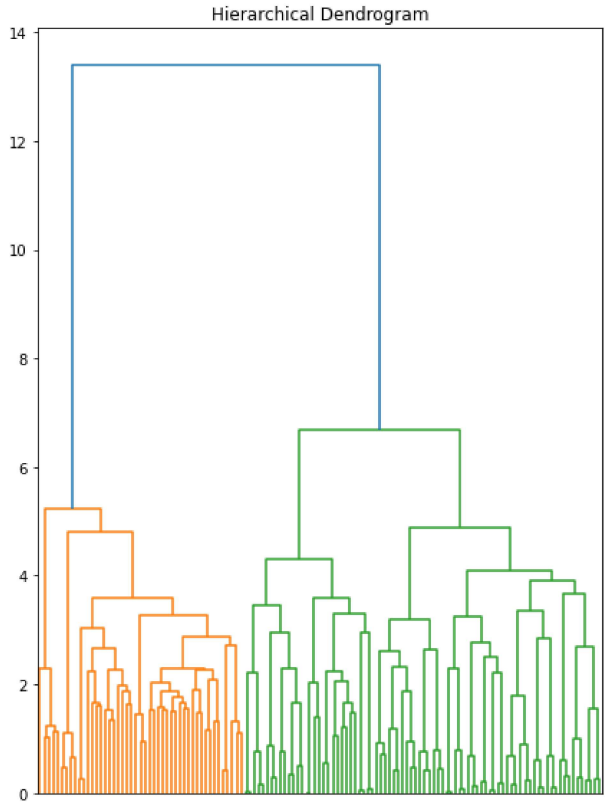
In [32]:
```
df = pd.read_csv(r"gap718_LABELENC_CONTINUOUS.csv")
rdi = df["rdi"]
tSNE_df = preprocessed_train.copy()
tSNE_df["rdi"] = rdi
tSNE_df = tSNE_df[tSNE_df['rdi'].notna()]
plot_tSNE(df = tSNE_df, response_var="rdi", pca_components_num=40)
```

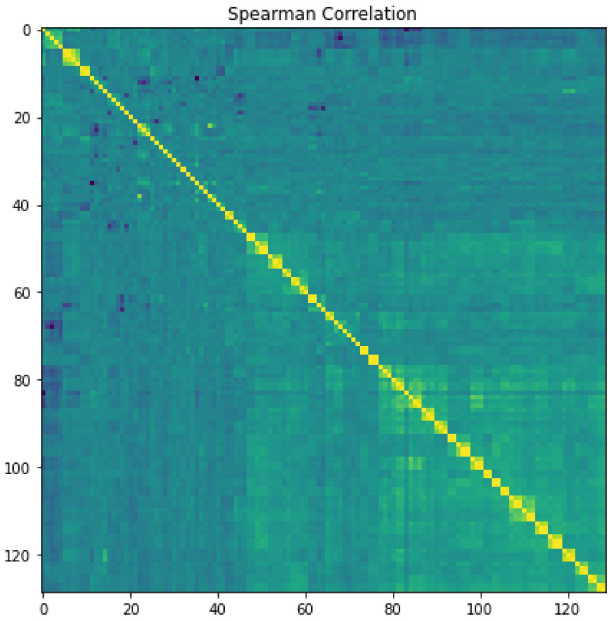


In [33]:
```
def plot_dendrogram(df):

    """
    Plot dendrogram using Ward distance


    Args:
        df: X matrix dataset (pd.DataFrame)
        response_var: response variable name (str)
        include_response: include response in dendrogram (boolean)


    Returns:
        Dendrogram plot

    """

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 8))
    corr = spearmanr(df).correlation
    corr_linkage = hierarchy.ward(corr)
    dendro = hierarchy.dendrogram(
    corr_linkage,labels = list(df.columns.values),ax=ax1 ,leaf_rotation=90
    )
    dendro_idx = np.arange(0, len(dendro['ivl']))

    ax1.get_xaxis().set_visible(False)

    ax2.imshow(corr[dendro['leaves'], :][:, dendro['leaves']])
    ax1.title.set_text('Hierarchical Dendrogram')
    ax2.title.set_text("Spearman Correlation")
    ax1.tick_params(axis="x",bottom = False)

    fig.tight_layout()
    plt.show()
```

Dendrogram and corrlation plot

In [34]:
```
plot_dendrogram(X_train)
```





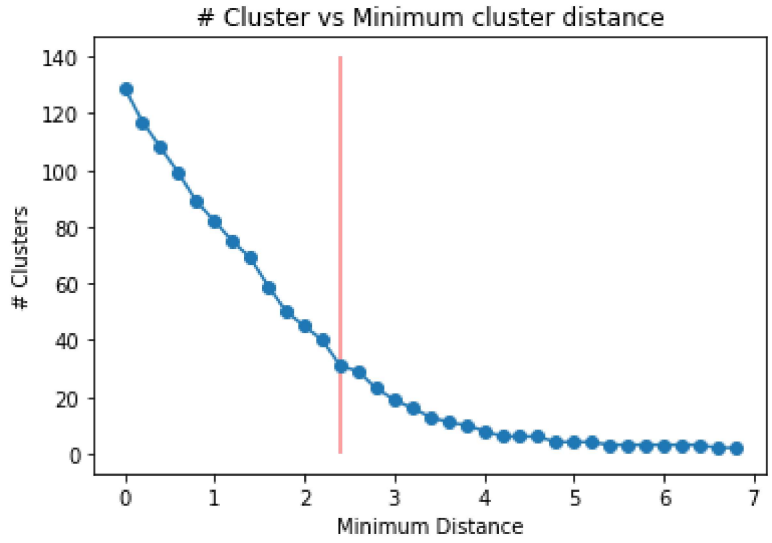
Feature Selection

Clusters # vs Dendrogram height

In [35]:
```
def plot_clusters_number(df):

    """
    Plot cluster numbers as a function of dendrogram height

    Args:
        df: dataset without response variable

    Returns:
        Cluster# vs dendrogram height plot


    """
    corr = spearmanr(df).correlation
    corr_linkage = hierarchy.ward(corr)
    clust_num = []
    distance = []
    for i in range(0,70,2):
        cluster_ids = hierarchy.fcluster(corr_linkage, i/10, criterion='distance'
        distance.append(i/10)
        cluster_id_to_feature_ids = defaultdict(list)
        for idx, cluster_id in enumerate(cluster_ids):
            cluster_id_to_feature_ids[cluster_id].append(idx)
        clust_num.append(len(cluster_id_to_feature_ids))
    plt.plot(distance, clust_num, marker = "o")
    plt.ylabel("# Clusters")
    plt.xlabel("Minimum Distance")
    plt.title("# Cluster vs Minimum cluster distance")
```

In [36]:
```
plot_clusters_number(preprocessed_train)
plt.vlines(2.4,0,140, alpha = 0.5, color = "red")
# red line show cut height of dendrogram
```

Out[36]: <matplotlib.collections.LineCollection at 0x289e054ad30>



In [37]:
```
# Prefiltered dataset
df_prefilterred = pd.read_csv(prefilttered_file_path)
df_prefilterred = df_prefilterred.iloc[:,2:]
df_prefilterred = make_response_class(df_prefilterred, "rdi", [0.65])
#Code to be added
myRowSelector =  RowSelector(df=df_prefilterred, response_column="rdi", random_st
train_df, test_df = myRowSelector.train_test_split(test_size = 0.2)
# fit pipeline to train split
myPreprocessor.fit(train_df)
# transform train set
preprocessed_train = myPreprocessor.transform(train_df)
X_train, y_train = preprocessed_train.iloc[:,1:], preprocessed_train["rdi"]
preprocessed_test = myPreprocessor.transform(test_df)
X_test, y_test = preprocessed_test.iloc[:,1:], preprocessed_test["rdi"]


# COntimuous response version
df_cont = pd.read_csv(prefilttered_file_path)
df_cont = df_cont.iloc[:,2:]
myRowSelector =  RowSelector(df=df_cont, response_column="rdi", random_state=0)
train_df_cont, test_df_cont = myRowSelector.train_test_split(test_size = 0.2)
myPreprocessor.fit(train_df_cont)
preprocessed_train_cont = myPreprocessor.transform(train_df_cont)
```

Absolute Correlation as a selection metric

In [38]:
```
def calculate_absolute_corr_coef(df, response_var, option = None):

    """
    calculate absolute correlation between predictors and response variable


    Args:
        df: dataset with response variable has continuous values (pd.DataFrame)
        response_var: response variable name (str)
        option: "spearman" or "pearson"

    Returns:
        a dictionary with keys are predictors and values are absolute correlatic
        between response variable and predictors

    """

    if option == "spearman":
        corr = abs(df.corr(method = "spearman"))
        abs_corr_coef_dict = corr[response_var].to_dict()

    if option == "pearson":
        corr = abs(df.corr(method = "pearson"))
        abs_corr_coef_dict = corr[response_var].to_dict()
    if option == None:
      #default is pearson
        corr = abs(df.corr(method = "pearson"))
        abs_corr_coef_dict = corr[response_var].to_dict()

    return abs_corr_coef_dict
```

In [39]:
```
### TEST CASE: using continuous response variable

calculate_absolute_corr_coef(preprocessed_train_cont, "rdi")
```

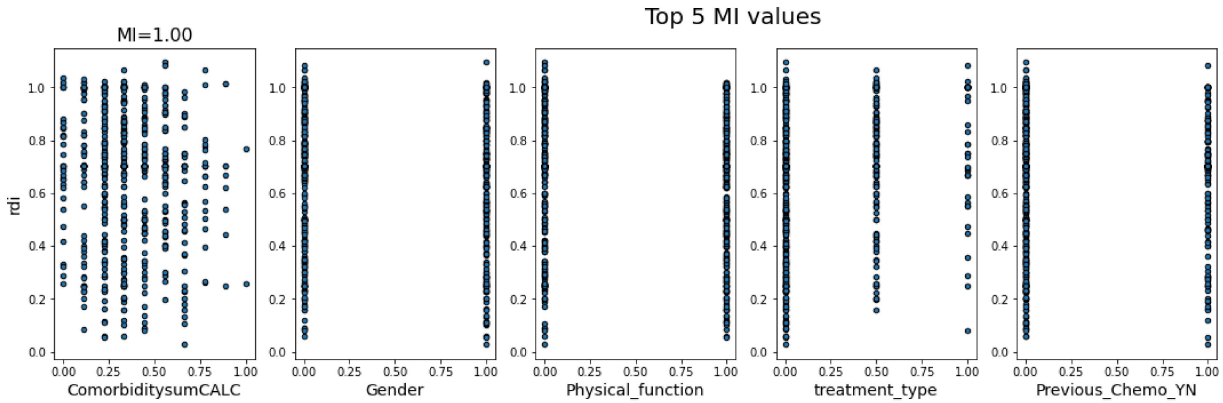
Out[39]:
```
{'rdi': 1.0,
 'Adltotal': 0.04543025592490981,
 'CalcBOMCTotal': 0.028778698192345107,
 'Age': 0.0456503915039278,
 'Gender': 0.07775805365759599,
 'educat': 0.029225154886848896,
 'FH1': 0.08163050738656727,
 'Age_Phy': 0.09302408126116637,
 'Gender_Phy': 0.06573843075307192,
 'Week_practice': 0.0013157682511719176,
 'Previous_Chemo_YN': 0.003823041787216808,
 'gad7TotalCalc': 0.06468145807694622,
 'gds_total_calc': 0.1457117448345113,
 'iadltotal': 0.12611446117966155,
 'KPS': 0.19571375748482162,
 'Weight change_6mo': 0.08144534325158519,
 'BMI': 0.004914308267327329,
 'calcmnascore': 0.142765915233397,
 'ComorbiditysumCALC': 0.06309745209380492,
 'phsum': 0.10400210723162615,
 'ArmSwellingIntrf': 0.1028389337260645,
 'PainIntrf': 0.2128641263611865,
 'HeadachesIntrf': 0.09925518711053911,
 'NauseaSev': 0.14791111889226116,
 'DiarrheaSev': 0.07249708860440175,
 'UrinationControlIntrf': 0.060028854695078446,
 'FatigueIntrf': 0.16403665514565632,
 'AppetiteIntrf': 0.18092493318984218,
 'NumbnessIntrf': 0.10729558887820398,
 'VisionIntrf': 0.0006933957765109049,
 'SOBIntrf': 0.07373954713658847,
 'InsomniaIntrf': 0.10408208500840183,
 'TasteIntrf': 0.12054121794789219,
 'DizzinessIntrf': 0.16579091881669783,
 'MouthSoresIntrf': 0.06097207514609333,
 'ConcentrationIntrf': 0.05640288498446237,
 'MemoryIntrf': 0.06480917988087916,
 'ConstipationSev': 0.15959127490998015,
 'SwallowingSev': 0.13028160772117622,
 'DryMouthSev': 0.19926898333744378,
 'HandFootSev': 0.02654649029181159,
 'RingEarsSev': 0.07019920255337382,
 'SkinSev': 0.07198013605427858,
 'HairLossSev': 0.0208044267106236,
 'StudyArm': 0.05780462308414705,
 'Physical_function': 0.06073654838893616,
 'Physical_performance': 0.10305257062370748,
 'Comorbidity': 0.024639423834530896,
 'Polypharmacy': 0.06314840389986413,
 'nutrition': 0.11965207642143944,
 'Social_support': 0.014433911697795015,
 'psychological': 0.13662660691953446,
```

```
 'cognition': 0.06590943526343586,
 'SPPBTotalSumcalc': 0.13514340480396153,
 'stdofcare': 0.26891838703008036,
 'sure_total': 8.634921917492872e-05,
 'stage': 0.02711338537798917,
 'chemo': 0.09365361371054216,
 'radiation_tx': 0.009405845917460728,
 'treatment_type': 0.2205932026777588,
 'Time_TUG': 0.06834542311336798,
 'racecat_Non Hispanic White': 0.010840904762296662,
 'racecat_Others': 0.055590641978626586,
 'maritalcat_Separated/ Widowed/ Divorced': 0.0032409802241504084,
 'maritalcat_Single, Never Married': 0.01438008992655748,
 'livecat_Living with parents': 0.010056746776589295,
 'incomecat_>50000': 0.07310903190379094,
 'incomecat_Decline to answer': 0.005104623118123385,
 'CalcMinicogScore_1-2 words + clock Normal - Not impaired': 0.02711243986507
45,
 'CalcMinicogScore_3 words - Not impaired': 0.04622493178232455,
 'CalcMinicogScore_No words - IMPAIRED': 0.03490793510672097,
 'GI_Lung_GU': 0.19674331443597842,
 'GI_Lung_Lung': 0.08083037390535819,
 'GI_Lung_Other': 0.022436316347692484}
```

Mutual Information as a selection metric

In [40]:
```
def calculate_MI(df, response_var):

    """
    calculate mutual information score between predictors and response variable


    Args:
        df: dataset with response variable has continuous values (pd.DataFrame)
        response_var: response variable name (str)

    Returns:
        a dictionary with keys are predictors and values are mutual information b

    """


    y = df[response_var]
    X = df.drop(response_var, axis = 1)
    mi = mutual_info_regression(X, y)
    mi /= np.max(mi)
    MI_dict = {}
    for i in range(len(X.columns)):
        MI_dict[X.iloc[:,i].name] = mi[i]

    topm = {}
    m = {k:x for (k,x) in zip(list(MI_dict.keys()),list(MI_dict.values()))}
    m1 = m.copy()
    for i in range(5):
        topm[max(m, key = lambda key: m[key])] = m[max(m, key = lambda key: m[key
        del m[max(m, key = lambda key: m[key])]
    X1= X[list(topm.keys())]
    plt.figure(figsize=(22, 5)).suptitle("Top 5 MI values", fontsize=20)
    for i in range(5):
        plt.subplot(1, 6, i + 1)
        plt.scatter(X1.iloc[:, i], y, edgecolor='black', s=20)
        plt.xlabel(X1.iloc[:,i].name, fontsize=14)
        if i == 0:
            plt.ylabel("rdi", fontsize=14)
            plt.title("MI={:.2f}".format(topm[X1.iloc[:,i].name]),
             fontsize=16)

    plt.show()
    return MI_dict
```

In [41]:
```
### TEST CASE
calculate_MI(preprocessed_train_cont, "rdi")
```



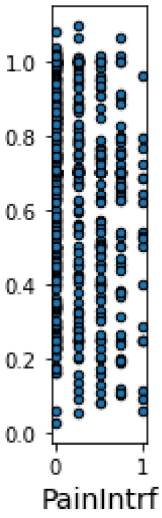
Out[41]:
```
{'Adltotal': 0.0,
 'CalcBOMCTotal': 0.0,
 'Age': 0.0,
 'Gender': 0.8491747049885965,
 'educat': 0.0,
 'FH1': 0.5557515034482139,
 'Age_Phy': 0.7244579912503308,
 'Gender_Phy': 0.3258989350048069,
 'Week_practice': 0.1366447916607252,
 'Previous_Chemo_YN': 0.8086416640228673,
 'gad7TotalCalc': 0.0,
 'gds_total_calc': 0.03065742040132314,
 'iadltotal': 0.5285798021784429,
 'KPS': 0.6035734490535437,
 'Weight change_6mo': 0.0,
 'BMI': 0.6881410809151193,
 'calcmnascore': 0.055655221608193556,
 'ComorbiditysumCALC': 1.0,
 'phsum': 0.7226890665348736,
 'ArmSwellingIntrf': 0.019314032865162173,
 'PainIntrf': 0.0,
 'HeadachesIntrf': 0.7035711826066151,
 'NauseaSev': 0.05541141894155505,
 'DiarrheaSev': 0.4616761586370553,
 'UrinationControlIntrf': 0.3012148541869603,
 'FatigueIntrf': 0.0,
 'AppetiteIntrf': 0.0,
 'NumbnessIntrf': 0.0,
 'VisionIntrf': 0.0,
 'SOBIntrf': 0.0,
 'InsomniaIntrf': 0.0,
 'TasteIntrf': 0.0,
 'DizzinessIntrf': 0.3274669926209625,
 'MouthSoresIntrf': 0.0,
 'ConcentrationIntrf': 0.0,
 'MemoryIntrf': 0.0,
 'ConstipationSev': 0.7184259753045107,
 'SwallowingSev': 0.41738751079771785,
 'DryMouthSev': 0.5147957581437558,
 'HandFootSev': 0.0,
 'RingEarsSev': 0.028681637112609015,
```

```
 'SkinSev': 0.024325570817886052,
 'HairLossSev': 0.0,
 'StudyArm': 0.1441275506976982,
 'Physical_function': 0.832469999254732,
 'Physical_performance': 0.0,
 'Comorbidity': 0.10279009214510292,
 'Polypharmacy': 0.04664318231178632,
 'nutrition': 0.15946203945539547,
 'Social_support': 0.2896245083316883,
 'psychological': 0.1302846042813068,
 'cognition': 0.21216259787086475,
 'SPPBTotalSumcalc': 0.0,
 'stdofcare': 0.7654371811916213,
 'sure_total': 0.0,
 'stage': 0.19917037824590184,
 'chemo': 0.5111374844243206,
 'radiation_tx': 0.0,
 'treatment_type': 0.8252319144694251,
 'Time_TUG': 0.1864912520641886,
 'racecat_Non Hispanic White': 0.6325765691798046,
 'racecat_Others': 0.0,
 'maritalcat_Separated/ Widowed/ Divorced': 0.0,
 'maritalcat_Single, Never Married': 0.13478606445440028,
 'livecat_Living with parents': 0.0,
 'incomecat_>50000': 0.5003812303398988,
 'incomecat_Decline to answer': 0.0,
 'CalcMinicogScore_1-2 words + clock Normal - Not impaired': 0.11368836961227
895,
 'CalcMinicogScore_3 words - Not impaired': 0.011416924221170304,
 'CalcMinicogScore_No words - IMPAIRED': 0.0,
 'GI_Lung_GU': 0.46395253623593735,
 'GI_Lung_Lung': 0.4662992996646474,
 'GI_Lung_Other': 0.4552298881302215}
```

In [48]:
```
def calculate_F_score(df, response_var):

    """
    calculate F score between predictors and response variable


    Args:
        df: dataset with response variable has continuous values (pd.DataFrame)
        response_var: response variable name (str)

    Returns:
        a dictionary with keys are predictors and values are F_score between resp

    """
    y = df[response_var]
    X = df.drop(response_var, axis = 1)
    f = f_regression(X, y)[0]
    f /= np.max(f)

    F_score_dict = {}
    print(X.iloc[:,1].name)
    for i in range(len(X.columns)):
        F_score_dict[X.iloc[:,i].name] = f[i]
    topf = {}
    f = {k:x for (k,x) in zip(list(F_score_dict.keys()),list(F_score_dict.values(
    f1 = f.copy()
    for i in range(5):
        topf[max(f, key = lambda key: f[key])] = f[max(f, key = lambda key: f[key
        del f[max(f, key = lambda key: f[key])]
    X1= X[list(topf.keys())]
    plt.figure(figsize=(22, 5)).suptitle("Top 5 F-test values", fontsize=20)
    for i in range(5):
        plt.subplot(1, 6, i + 1)
        plt.scatter(X1.iloc[:, i], y, edgecolor='black', s=20)
        plt.xlabel(X1.iloc[:,i].name, fontsize=14)
        if i == 0:
            plt.ylabel("rdi", fontsize=14)
        plt.title("F-test={:.2f}".format(topf[X1.iloc[:,i].name]),
             fontsize=16)

        plt.show()
    return F_score_dict
```

In [44]:
```
calculate_F_score(preprocessed_train_cont, 'rdi')
```

F-test=0.61

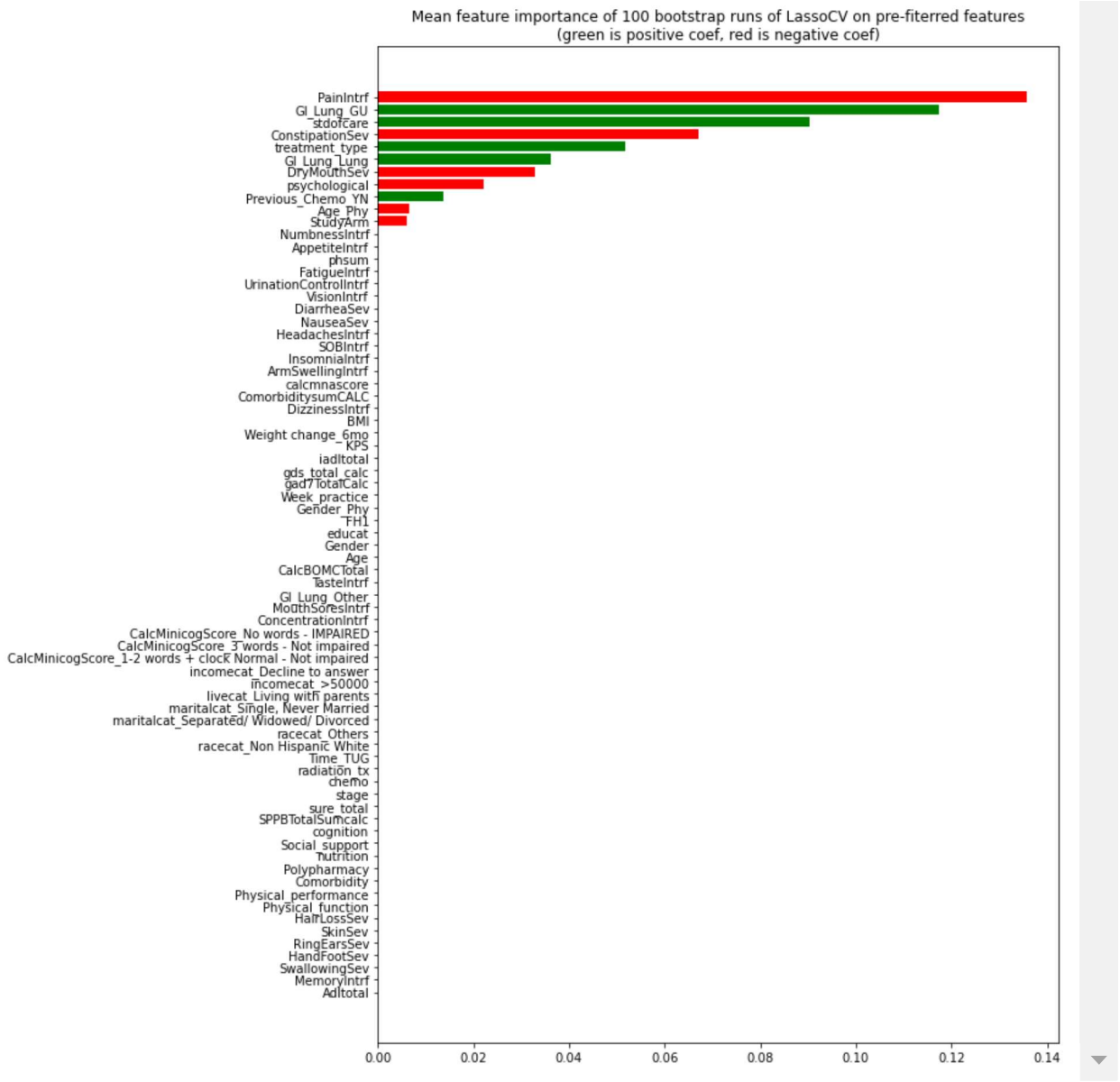


F-test=0.53

In [53]:
```
def model_coef(df, response_var, model, boot_num, option):

    """
    return feature importance based on model coefs or feature importance score


    Args:
        df: dataset(pd.DataFrame)
        response_var: response variable name (str)
        model: machine learning model to fit df
        option: coef_ or features_importance_

    Returns:
        a dictionary with keys are predictors and values are important scores (cc

    """

    bootstrap_result = []
    bootstrap_accumulate = []
    for _ in range(boot_num):
        clf = model



        X_train, y_train = df.loc[:, preprocessed_train.columns != response_var],


        clf.fit(X= X_train, y = y_train)

        accumulate = None
        if option == "coef_":
            accumulate = clf.coef_
            bootstrap_accumulate.append(accumulate)
        elif option == "feature_importances_":
            accumulate = clf.feature_importances_
            bootstrap_accumulate.append(accumulate)

        mean_coef = []
        se_coef = []
        for i in np.array(bootstrap_accumulate).T:
            mean_coef.append(np.mean(i))
            se_coef.append(2*np.std(i)/np.sqrt(100))

    return mean_coef, se_coef
```

In [54]:
```
### TEST CASE

mean_coef, se_coef = model_coef(preprocessed_train, "rdi", linear_model.LassoCV()

color = []
for m in mean_coef:
    if m < 0 :
        color.append("red")
    else:
        color.append("green")

my_lasso_coefs = tuple(zip(np.abs(mean_coef), se_coef ,X_train.columns))
sorted_lasso = np.array(sorted(my_lasso_coefs, key = lambda x: x[0], reverse= Tru

lasso_df = pd.DataFrame([np.abs(mean_coef), se_coef ,X_train.columns, color], inc
lasso_sorted_df = lasso_df.sort_values(by = ["coef"], axis = 0, ascending = True)

plt.figure(figsize = (10,15))
plt.barh(lasso_sorted_df["names"], lasso_sorted_df["coef"], xerr = lasso_sorted_c
plt.title("Mean feature importance of 100 bootstrap runs of LassoCV on pre-fiterr
```

Out[54]:
```
Text(0.5, 1.0, 'Mean feature importance of 100 bootstrap runs of LassoCV on pre
-fiterred features\n (green is positive coef, red is negative coef) ')
```


Mean feature importance of 100 bootstrap runs of LassoCV on pre-fiterred features
(green is positive coef, red is negative coef)
PainIntrf
GI_Lung_GU
stdofcare
ConstipationSev
treatment_type
GI_Lung_Lung
DryMouthSev
psychological
Previous_Chemo_YN
Age_Phy
StudyArm
NumbnessIntrf
AppetiteIntrf
phsum
FatigueIntrf
UrinationControlIntrf
VisionIntrf
DiarrheaSev
NauseaSev
HeadachesIntrf
SOBIntrf
InsomniaIntrf
ArmSwellingIntrf
calcmnascore
ComorbiditysumCALC
DizzinessIntrf
BMI
Weight change_6mo
KPS
iadltotal
gds_total_calc
gad7TotalCalc
Week_practice
Gender_Phy
FH1
educat
Gender
Age
CalcBOMCTotal
TasteIntrf
GI_Lung_Other
MouthSoresIntrf
ConcentrationIntrf
CalcMinicogScore_No words - IMPAIRED
CalcMinicogScore_3 words - Not impaired
CalcMinicogScore_1-2 words + clock Normal - Not impaired
incomecat_Decline to answer
incomecat_>50000
livecat_Living with parents
maritalcat_Single, Never Married
maritalcat_Separated/ Widowed/ Divorced
racecat_Others
racecat_Non Hispanic White
Time_TUG
radiation_tx
chemo
stage
sure_total
SPPBTotalSumcalc
cognition
Social_support
nutrition
Polypharmacy
Comorbidity
Physical_performance
Physical_function
HairLossSev
SkinSev
RingEarsSev
HandFootSev
SwallowingSev
MemoryIntrf
Adltotal
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

In [55]:
```
def get_permutation_importance(df, response_var, model):

    """
    return permutation importanc score of each feature


    Args:
        df: dataset(pd.DataFrame)
        response_var: response variable name (str)
        model: machine learning model to fit df

    Returns:
        a dictionary with keys are predictors and values are permutation_importar

    """

    y = df[response_var]
    X = df.drop(response_var, axis = 1)


    model.fit(X, y)
    result = permutation_importance(model, X, y)

    return result
```

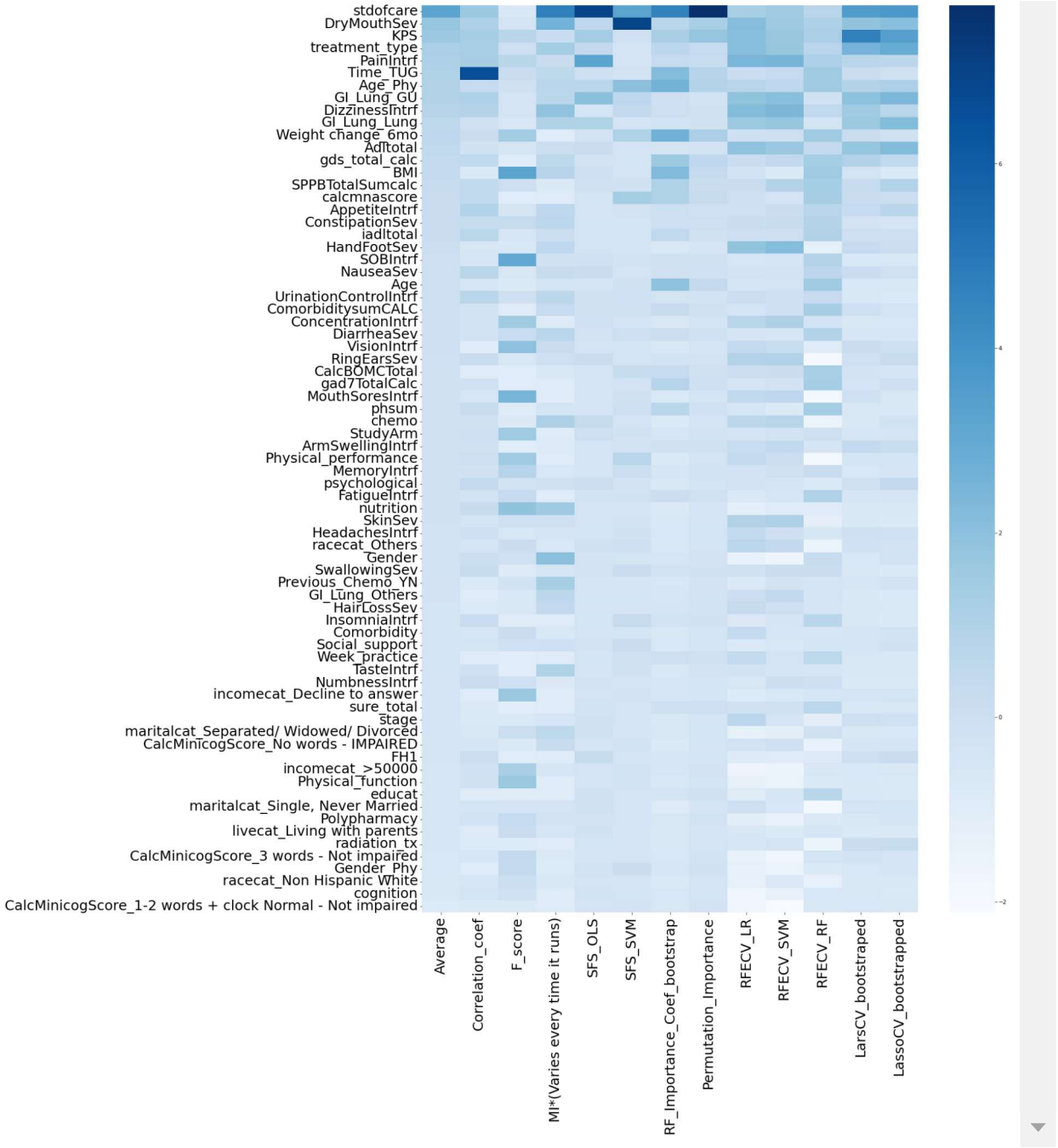
In [56]:
```
get_permutation_importance(preprocessed_train, "rdi", RandomForestClassifier(n_es
```

Out[56]:
```
array([0.        , 0.        , 0.00169492, 0.        , 0.        ,
       0.        , 0.00101695, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00135593, 0.00237288,
       0.00169492, 0.        , 0.        , 0.00135593, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00033898, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00101695, 0.        ,
       0.        , 0.        , 0.        , 0.00033898, 0.00135593,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00033898, 0.        , 0.        ])
```

Generate Feature Consensus Heatmap

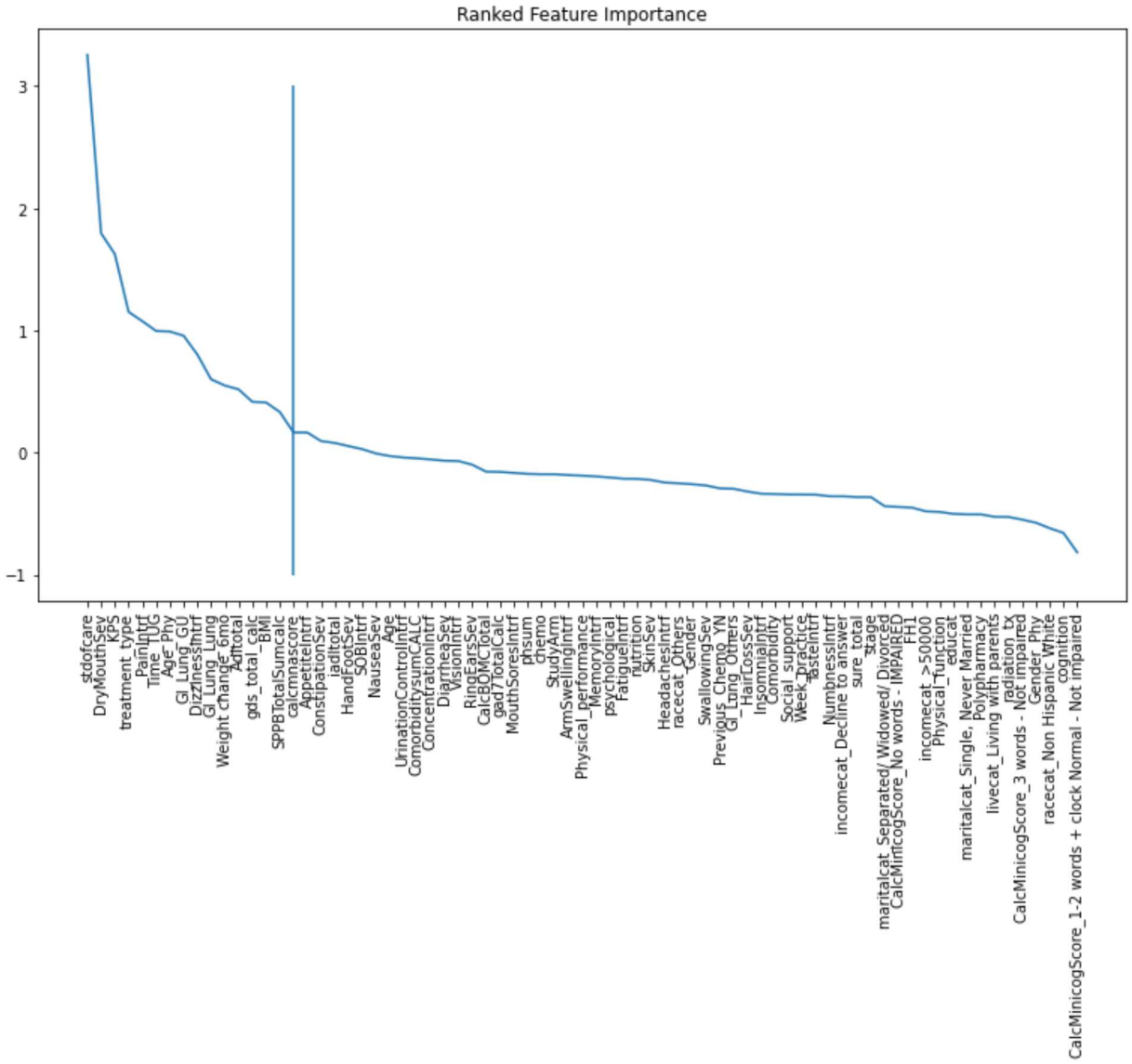
Plot Feature Selection Map

In [57]:
```
f_metrics = pd.read_csv(feature_file_path)
f_metrics["Correlation_coef"] = abs(f_metrics["Correlation_coef"])
f_metrics["SFS_OLS"] = 1/ f_metrics["SFS_OLS"]
f_metrics["SFS_SVM"] = 1/ f_metrics["SFS_SVM"]
f_metrics = f_metrics.set_index(f_metrics["Features"])
f_metrics = f_metrics.iloc[:,1:]
myScaler = Scaler("Standard")
myScaler.fit(f_metrics)
transformed_f_metrics = myScaler.transform(f_metrics)
sum_col = 0
for col in transformed_f_metrics.columns:
    sum_col += np.array(transformed_f_metrics[col])
ave_col = sum_col/12
transformed_f_metrics.insert(0, column ="Average", value= ave_col)
transformed_f_metrics = transformed_f_metrics.sort_values(by = "Average", axis =
transformed_f_metrics= transformed_f_metrics.dropna()
plt.figure(figsize= (20,30))
sns.heatmap(transformed_f_metrics, cmap = "Blues")
plt.xticks(rotation = 90, fontsize = 25)
plt.yticks(fontsize = 25)
plt.ylabel(None)
plt.show()
```


stdofcare
DryMouthSev
KPS
treatment_type
PainIntrf
Time_TUG
Age_Phy
GI_Lung_GU
DizzinessIntrf
GI_Lung_Lung
Weight change_6mo
AdlTotal
gds_total_calc
BMI
SPPBTotalSumcalc
calcmnascore
AppetiteIntrf
ConstipationSev
iadltotal
HandFootSev
SOBIntrf
NauseaSev
Age
UrinationControlIntrf
ComorbiditysumCALC
ConcentrationIntrf
DiarrheaSev
VisionIntrf
RingEarsSev
CalcBOMCTotal
gad7TotalCalc
MouthSoresIntrf
phsum
chemo
StudyArm
ArmSwellingIntrf
Physical_performance
MemoryIntrf
psychological
FatigueIntrf
nutrition
SkinSev
HeadachesIntrf
racecat_Others
Gender
SwallowingSev
Previous_Chemo_YN
GI_Lung_Others
HairLossSev
InsomniaIntrf
Comorbidity
Social_support
Week_practice
TasteIntrf
NumbnessIntrf
incomecat_Decline to answer
sure_total
Stage
maritalcat_Separated/ Widowed/ Divorced
CalcMinicogScore_No words - IMPAIRED
FH1
incomecat_>50000
Physical_function
educat
maritalcat_Single, Never Married
Polypharmacy
livecat_Living with parents
radiation_tx
CalcMinicogScore_3 words - Not impaired
Gender_Phy
racecat_Non Hispanic White
cognition
CalcMinicogScore_1-2 words + clock Normal - Not impaired
Average
Correlation_coef
F_score
MI*(Varies every time it runs)
SFS_OLS
SFS_SVM
RF_Importance_Coef_bootstrap
Permutation_Importance
RFECV_LR
RFECV_SVM
RFECV_RF
LarsCV_bootstraped
LassoCV_bootstrapped
6
4
2
0
−2

Elbow plot

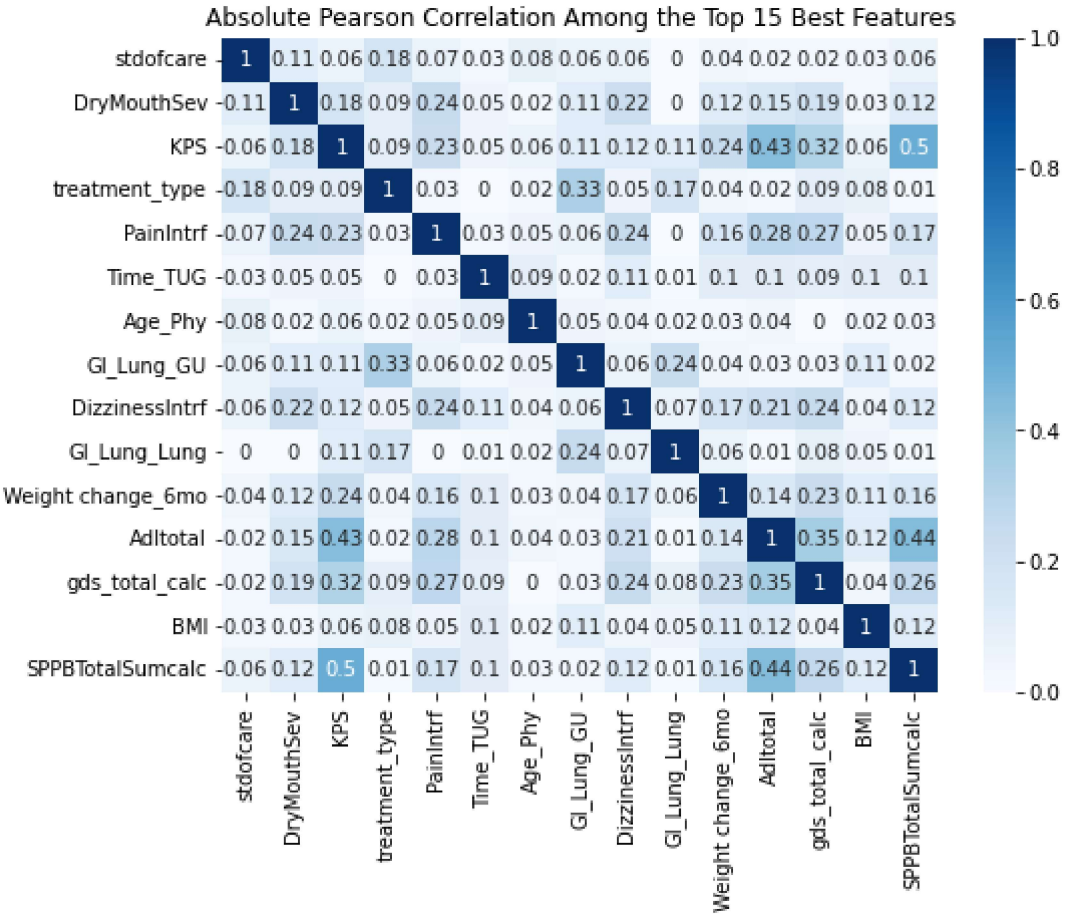
In [58]:
```
plt.figure(figsize = (13,7))
plt.plot(transformed_f_metrics["Average"])
plt.xticks(np.arange(73), rotation = 90)
plt.title("Ranked Feature Importance")
plt.vlines(15,-1,3)
plt.show()
```



Top 15 features correlation

In [59]:
```
selected_features = transformed_f_metrics.T.columns[:15]
plt.figure(figsize = (8,6))
plt.title("Absolute Pearson Correlation Among the Top 15 Best Features")
sns.heatmap(abs(np.round((np.corrcoef(preprocessed_train[selected_features].T)),2
            xticklabels=selected_features, yticklabels=selected_features, annot=T
```

Out[59]:
```
<AxesSubplot:title={'center':'Absolute Pearson Correlation Among the Top 15 Bes
t Features'}>
```



Model Optmization

Preprocess Train set

In [60]:
```
# Code to be added (transform Train set)
```

In [61]:
```
def get_optimized_parameter(model, params_dict, df, response_var):

    """
    Get an optimized set of parameter for a model


    Args:
        model: model to be optimized
        params_dict: a diction nary with keys are hyperparameter andvalues are ra
        df: dataset, train set(pd.DataFrame)
        response_var: response variable name (str)

    Returns:
        parameter dictionary with optimizd value

    """



    X_train, Y_train = df.loc[:, df.columns != response_var], df[response_var]

    CV = GridSearchCV(estimator=model, param_grid=params_dict, cv= 5)
    CV.fit(X_train, Y_train)

    optimized_params_dict = CV.best_params_


    return optimized_params_dict
```

Model fit and metrics

In [62]:
```
def get_model_performance_kfold(model, folds, response_var, metric_list):

    """
    get metrics from tuned models

    Args:
        model: fitted model
        folds: output from RowSelector kfold split
        response_var: response variable name (str)
        metrics: a list of metrics (limited to Accuracy, Precision, Recall, AUC)

    Returns:
        a dictionary of model performance with values are tuples with 2 elements:

    """
    accuracy = []
    recall = []
    precision = []
    auc = []

    # Iterate through each fold
    for fold, (train, val) in enumerate(folds):

      #Specify pipeline
      #for pl in PreProcessor.get_all_combinations(drop = "MICE"):
        myPreprocessor = PreProcessor(
         pipeline= {
            "Encoder": "OneHot",
            "Imputer": "Median",
            "Scaler": "MinMax",
            "FeatureSelection": "All",
            "DimReduction": "Skip"
         },
          response_var = "rdi"
        )


        # fit pipeline to train split
        myPreprocessor.fit(train)

        # transform train set
        preprocessed_train = myPreprocessor.transform(train)
        # transform validation set
        preprocessed_val = myPreprocessor.transform(val, val = True)


        X_train, y_train = preprocessed_train.loc[:, preprocessed_train.columns !
        X_val, y_val = preprocessed_val.loc[:, preprocessed_val.columns != respor

        model = model.fit(X_train, y_train)
        pred = model.predict(X_val)

        auc.append(metrics.roc_auc_score(y_val, pred))
        recall.append(metrics.recall_score(y_val, pred))
        precision.append(metrics.precision_score(y_val, pred))
        accuracy.append(accuracy_score(y_val, pred))
```

```
    performance_dict = {}
    if 'Accuracy' in metric_list:
        performance_dict['Accuracy'] = np.mean(accuracy), 2*np.std(accuracy)/len(

    if 'Precision' in metric_list:
        performance_dict['Precision'] = np.mean(precision), 2*np.std(precision)/l

    if 'Recall' in metric_list:
        performance_dict['Recall'] = np.mean(recall),  2*np.std(recall)/len(recal

    if 'AUC' in metric_list:
        performance_dict['AUC'] = np.mean(auc),  2*np.std(auc)/len(auc)

    return performance_dict
```

In [63]:
```
def get_model_performance(model, df, response_var, metric_list):

    """
    get metrics from tuned models

    Args:
        model: fitted model
        df: dataset, test or train set(pd.DataFrame)
        response_var: response variable name (str)
        metric_list: a list of metrics (limited to Accuracy, Precision, Recall, A

    Returns:
        a dictionary of model performance

    """

    X_val, Y_val = df.loc[:, df.columns != response_var], df[response_var]

    pred = model.predict(X_val)

    performance_dict = {}

    if 'Accuracy' in metric_list:
        performance_dict['Accuracy'] = accuracy_score(Y_val, pred)

    if 'Precision' in metric_list:
        performance_dict['Precision'] = metrics.precision_score(Y_val, pred)

    if 'Recall' in metric_list:
        performance_dict['Recall'] = metrics.recall_score(Y_val, pred)

    if 'AUC' in metric_list:
        performance_dict['AUC'] = metrics.roc_auc_score(Y_val, pred)


    return performance_dict
```

Define range for grid seearch

In [64]:
```
param_grid_lr = {
    'penalty': ['l1','l2'],
    'C': [0.001,0.01,0.1,1,5,10,25]
}
param_grid_rf = {
    'bootstrap': [True],
    'max_depth': [60, 70, 80],
    'max_features': ['sqrt'],
    'min_samples_leaf': [2, 4, 6],
    'min_samples_split': [2],
    'n_estimators': [1400, 1600, 1800]
}

param_grid_xgb = {
    'colsample_bytree': [0.3],
    'n_estimators': range(5,25),
    'max_depth': range(2,17)
}
```

Use top 10 features

In [65]:
```
top10_selected = list(transformed_f_metrics.T.columns[:10])
selected_df_train = preprocessed_train[top10_selected+["rdi"]]
selected_df_test = preprocessed_test[top10_selected+["rdi"]]
```

In [66]:
```
lr_params = get_optimized_parameter(LogisticRegression(),param_grid_lr , selected
print("PASS")
# uncomment to run random forest optimization. This will take quite a long time
#rf_params = get_optimized_parameter(RandomForestClassifier(), param_grid_rf , se
print("PASS")
xgb_params = get_optimized_parameter(xgb.XGBClassifier(), param_grid_xgb, selecte
```

```
PASS
PASS
[22:51:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:51:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:51:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:51:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
```

In [67]:
```
lr_optimized = LogisticRegression(**lr_params)
xgb_optimized = xgb.XGBClassifier(**xgb_params)
rf_optimized = RandomForestClassifier(**{"n_estimators":80})
```

In [68]:
```
for model in (lr_optimized, rf_optimized,xgb_optimized):
    print("Model", model)
    myFolds = myRowSelector.kfold_split(n_fold = 5, df = selected_df_train)
    print("5 fold cross validation performance")
    print(get_model_performance_kfold(model = model, folds = myFolds, response_va
                                      metric_list = ["Accuracy", "Precision", "Re

    print("Performance on test set")
    print(get_model_performance(model = model, df = selected_df_test, response_va
                          metric_list = ["Accuracy", "Precision", "Recall", "AUC'

    print("Performance on train set")
    print(get_model_performance(model = model, df = selected_df_train, response_v
                          metric_list = ["Accuracy", "Precision", "Recall", "AUC'
```

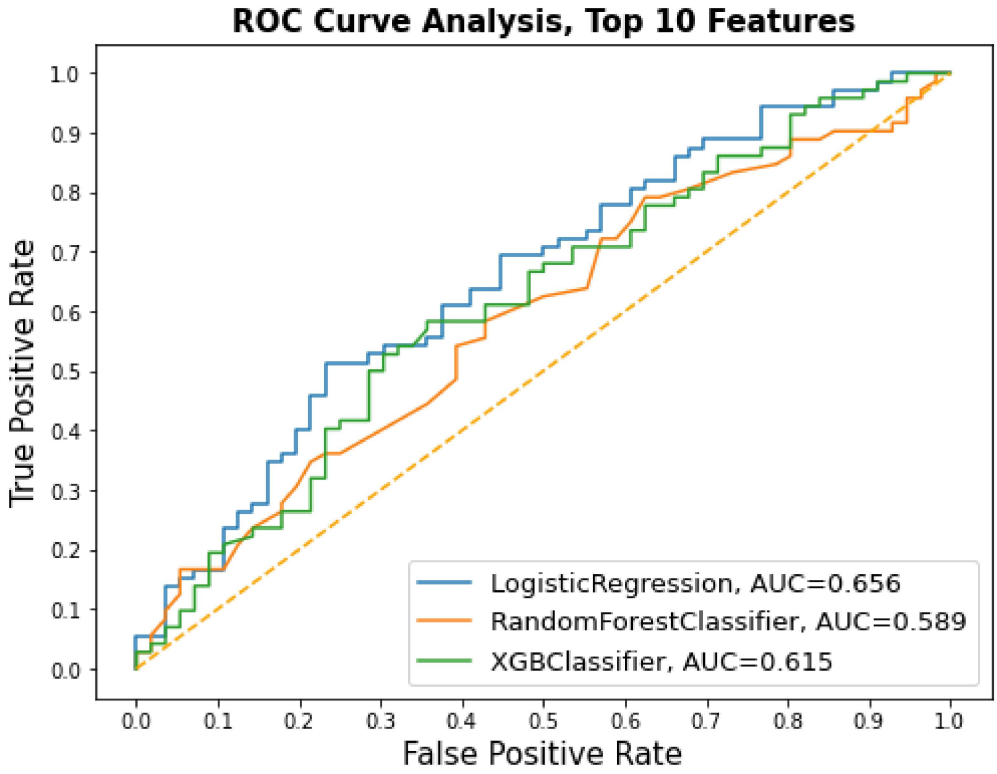
```
Model LogisticRegression(C=1)
5 fold cross validation performance
{'Accuracy': (0.6593220338983051, 0.01444561067977119), 'Precision': (0.68440
99400249991, 0.012260746053999318), 'Recall': (0.8519270823704854, 0.01679385
9357397294), 'AUC': (0.5917951491907637, 0.01170958608488821)}
Performance on test set
{'Accuracy': 0.609375, 'Precision': 0.6018518518518519, 'Recall': 0.902777777
7777778, 'AUC': 0.5674603174603174}
Performance on train set
{'Accuracy': 0.6830508474576271, 'Precision': 0.6876267748478702, 'Recall':
0.9112903225806451, 'AUC': 0.6024341521160106}
Model RandomForestClassifier(n_estimators=80)
5 fold cross validation performance
{'Accuracy': (0.6067796610169492, 0.012427323918524313), 'Precision': (0.6682
673619665489, 0.008097486808219964), 'Recall': (0.7482547033195914, 0.0178025
9701347012), 'AUC': (0.5575918099068626, 0.013063425021798514)}
Performance on test set
{'Accuracy': 0.5078125, 'Precision': 0.5517241379310345, 'Recall': 0.66666666
66666666, 'AUC': 0.48511904761904756}
Performance on train set
{'Accuracy': 0.8983050847457628, 'Precision': 0.8880597014925373, 'Recall':
0.9596774193548387, 'AUC': 0.8766277005031076}
Model XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=0.3,
              enable_categorical=False, gamma=None, gpu_id=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=2,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=22, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, reg_alpha=None,
              reg_lambda=None, scale_pos_weight=None, subsample=None,
              tree_method=None, validate_parameters=None, verbosity=None)
5 fold cross validation performance
[22:52:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:52:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
```

```
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:52:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:52:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[22:52:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
{'Accuracy': (0.6457627118644068, 0.0049820130361691835), 'Precision': (0.672
3618719711686, 0.007395437141737483), 'Recall': (0.8548926540094529, 0.010710
27696753569), 'AUC': (0.5719684112007236, 0.003027691230347149)}
Performance on test set
{'Accuracy': 0.6015625, 'Precision': 0.6, 'Recall': 0.875, 'AUC': 0.5625}
Performance on train set
{'Accuracy': 0.7016949152542373, 'Precision': 0.6936758893280632, 'Recall':
0.9435483870967742, 'AUC': 0.6162696063924238}
```

In [69]:
```
def plot_ROC(models, X_train, y_train, X_test, y_test):

    """
    plot ROC curves with AUC from fitted models


    Args:


        models: a list of fitted machine learning model  (list)

    Returns:
        ROC curves plot

    """



    # Define a result table as a DataFrame
    result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

    # Train the models and record the results
    for cls in models:
        model = cls.fit(X_train, y_train)
        yproba = model.predict_proba(X_test)[::,1]

        fpr, tpr, _ = roc_curve(y_test,  yproba)
        auc = roc_auc_score(y_test, yproba)

        result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                            'fpr':fpr,
                                            'tpr':tpr,
                                            'auc':auc}, ignore_index=True)

        # Set name of the classifiers as index labels
    result_table.set_index('classifiers', inplace=True)
    fig = plt.figure(figsize=(8,6))

    for i in result_table.index:
        plt.plot(result_table.loc[i]['fpr'],
                 result_table.loc[i]['tpr'],
                 label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

    plt.plot([0,1], [0,1], color='orange', linestyle='--')

    plt.xticks(np.arange(0.0, 1.1, step=0.1))
    plt.xlabel("False Positive Rate", fontsize=15)

    plt.yticks(np.arange(0.0, 1.1, step=0.1))
    plt.ylabel("True Positive Rate", fontsize=15)

    plt.title('ROC Curve Analysis, Top 10 Features', fontweight='bold', fontsize=
    plt.legend(prop={'size':13}, loc='lower right')

    plt.show()
```

In [70]:
```
X_train, y_train = selected_df_train.drop("rdi", axis = 1), selected_df_train["rd
X_test, y_test = selected_df_test.drop("rdi", axis = 1), selected_df_test["rdi"]
models = [
       lr_optimized,
       rf_optimized,
       xgb_optimized,
       ]
plot_ROC(models, X_train, y_train, X_test, y_test)
```

```
[22:52:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

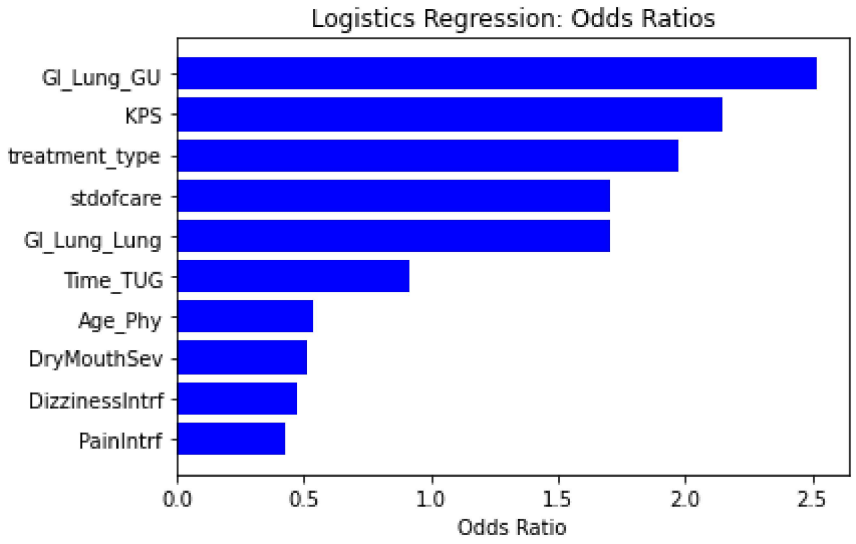


Model Interpretation

Calculate and plot Odd Ratio: Logistics Regression

In [71]:
```
coefs = list(np.exp(lr_optimized.coef_))
score = tuple(zip(coefs[0], X_train.columns))
sorted_score = sorted(score, key = lambda x:abs(x[0]))
sorted_score_t = np.array(sorted_score).T

colors = []
for i in np.array(sorted_score_t[0], float):
    if i > 0:
        colors.append("blue")
    else:
        colors.append("red")

plt.barh(sorted_score_t[1],np.array(sorted_score_t[0], float), color = colors)
plt.title("Logistics Regression: Odds Ratios")
plt.xlabel("Odds Ratio")
```

Out[71]:
```
Text(0.5, 0, 'Odds Ratio')
```



SHAP

In [72]:
```
def plot_SHAPs(tts, response_var, model, option = "beeswarm"):

    """
    plot SHAP values. Please refer to:
    https://shap.readthedocs.io
    /en/latest/example_notebooks/tabular_examples/model_agnostic/Census%20income%


    Args:
        tts: list of: [Xtrain, Xval, ytrain, yval]
        model: fitted machine learning model
        response_var: response variable name (str)
        option: type of SHAP plot to display (waterfall, beeswarm, heatmap, summa

    Returns:
        SHAP plot

    """
    model.fit(tts[0],tts[2])
    f = lambda x: model.predict_proba(x)[:,1]
    med = tts[0].median().values.reshape((1,tts[0].shape[1]))

    explainer = shap.Explainer(f,med)
    shap_values = explainer(tts[0])
    if option == "waterfall":
        shap.plots.waterfall(shap_values[0])
    if option == "beeswarm":
        shap.plots.beeswarm(shap_values)
    if option == "heatmap":
        shap.plots.heatmap(shap_values)
    plt.show()
    return
```

In [73]:
```
plot_SHAPs([X_train, X_test, y_train, y_test], "rdi", lr_optimized)
```