Stats504 Assignment 5: nuMoM2b

November 12, 2022

1 Introduction

Pregnancy can be viewed as window to future health. Along with pregnancy, certain common pregnancy complications have the potential to reveal a woman's vascular or metabolic susceptibility for future diseases, which makes pregnancy potentially risky, especially for pregnant women with no previous pregnancy lasting 20 weeks-0 days or more estimated gestational age (nulliparas). Adverse pregnancy outcomes, including preeclampsia, preterm birth, hypertensive disorders and diabetes, are major health risks for pregnant individuals during the pregnancy and throughout their lifespan. For instance, among pregnant women, 2 - 17.8 % develop gestational diabetes. Therefore, researchers are interested in understanding the underlying, interrelated mechanisms of adverse pregnancy outcomes, and further prevent adverse pregnancy outcomes beforehand.

In this paper, we would only focus on one of the adverse pregnancy outcomes, diabetes. We aim to develop binary classification models, which provide predictions on whether pregnant individuals would suffer from diabetes based on the mother-to-be data collected through interviews, self-administered questionnaires, clinical measurements, ultrasounds, and medical records reviews. Specifically, we develop a classification model that can give accurate classification results, while also providing good interpretability that can help the audience to understand how different factors could influence the possibility of having diabetes during pregnancy.

Concerning the questions of interest, there are some main findings that are worth mentioning: Firstly, based on our analysis, the XGBoost model gives the best performance with 0.20 f1-score and 97% accuracy on the test dataset. In the XGBoost model, the variable representing the pre-pregnancy weight, income, systolic blood measurement taken at the first visit, age of a pregnant woman, and the level of worrying health care are the top 5 variables that contribute the most to the prediction of gestational diabetes in the XGBoost model. Specifically, a higher pre-pregnancy weight systolic blood measurement and age value result in a higher predicted risk of having diabetes, while the higher the family income is, the less likely that the pregnant woman would develop diabetes. It's also worthy of notice that women identified as Black tend to have a higher predicted risk of having diabetes, while those identified as being White, Hispanic, and other races would instead have a lower predicted risk of having diabetes.

2 Method

The data set recorded mother-to-be information such as age, race, systolic, diastolic, income, and whether they have financial support or parent support. Based on these features, we want to predict if they have diabetes. The target variable is binary. A pregnant woman without history of type 1/2 diabetes was denoted as 0, otherwise was denoted as 1. Therefore, we used classification methods such as Logistic regression, Decision Tree, Random Forest, Extreme Gradient Boosting, and Adaptive Gradient Boosting to make the predictions.

From the EDA of the dataset, we learned that the data is highly imbalanced with highly skewed class proportions. The imbalance of data could significantly decrease the power of classification models. With so few positives relative to negatives, the training model would learn on negative examples thus not learn enough from positive ones. Usually, the model could fall into the 'trap' caused by skewed labels: It would always predict everything as the majority class for a decent accuracy. To address this problem, an oversampling method is adapted, which resamples more data from the minority class samples until the enlarged minority class reaches a comparable size as the majority class.

Since there is only one set of training data and only one set of validation data, the performance metrics of our learners rely heavily on those two sets. The learners are only trained and evaluated once, so the performance which only depends on one evaluation may perform very differently when it is trained and evaluated on different subsets of the same data, as the subsets are randomly selected. Furthermore, unlike parameters, the hyperparameters are the parameters that can be set by people before the machine learning model is trained. If those hyperparameters are just set empirically, it would lead to larger bias. In order to reduce this bias, we applied K-fold Cross Validation (CV).

In CV, we still firstly split the data into training and testing. The training data then is split into K number of folds (subsets). Next, CV will iterate through these folds, and at each iteration use one of the K folds as the validation set while using all other remaining folds as the training set at one time. This process is repeated until every fold has been used as a validation set. Fig 1 below shows how this process goes for a 5-fold Cross-Validation:

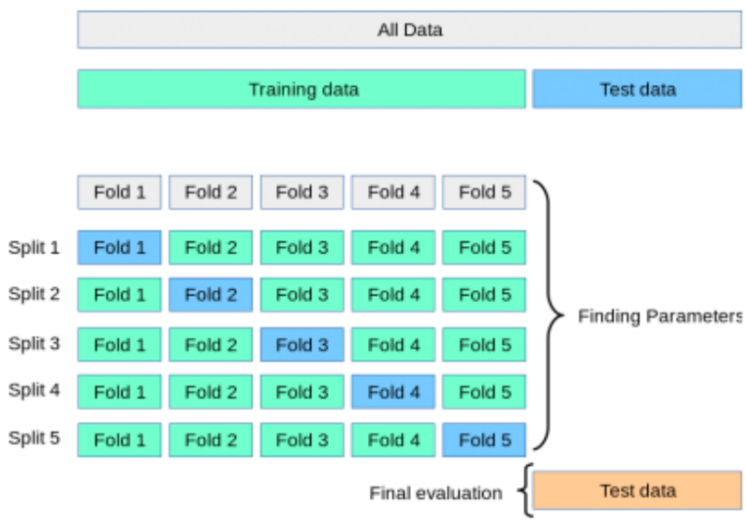


Figure 1: How a 5-Fold CV works

By training and evaluating the model K times on different subsets of the same training data, we are likely to get a better idea that how our model might perform on new data. During the process, we score the model after each iteration based on specific criteria and compute the average of all scores. Based on the score, we are able to find the best combination of hyperparameters before we train the model.

2.1 Model Explanation

In this project, 5 different classification models are adapted for evaluation, interpretation, and comparison. To make those models more accessible for the audience, brief introductions of each model are provided in the following part.

2.1.1 Logistic Regression

Logistic regression is a supervised machine learning algorithm to solve the binary classification problem where the dependent variable is binary, coded as 0 or 1. In logistic regression, the model aims to predict which category a new observation will belong to using a sigmoid function. The function squeezes the output of a linear equation between 0 and 1. When the result of the sigmoid function is greater than 0.5, the label is classified as class 1 (positive class); otherwise, it is classified as class 0 (negative class).

Logistic regression is easy to implement and interpret, and it assumes that there is litter or no multicollinearity between the independent variables. In order to limit the impact of collinearity and avoid overfitting, we used penalized logistic regressions in our project. Ridge penalty (L2 regularization) deals with multicollinearity by penalizing insignificant features while keeping all the variables. Lasso penalty (L1 regularization) shrinks the less important features' coefficients to zero, removing some features altogether, which also works well for feature selection in case we have huge amounts of independent variables. Moreover, we calculated the variance inflation factor first to check the multicollinearity.

Data were pre-processed before implementing the logistic regression models. According to the data dictionary, the "psstotal" variable was categorized into low, moderate and high levels of stress instead of using the stress scores directly. 14 categorical variables including race, emotional support, history of Lupus and etc, were encoded into dummy variables using one-hot encoding. It is also important to have feature scaling that puts all numerical features into the same range in order to avoid the vanishing gradient problem during the training phase. We used standardization to scale our features with a mean of 0 and a standard deviation of 1. In addition, abnormal values that are 0s in the "age" variable were imputed by the mean of age (27).

To handle the class imbalance, we assigned separate weights to the majority and minority classes by tuning the inbuilt "class_weight" parameter. Adding class weights allows us to penalize the minority class for misclassification by setting a higher class weight, while decreasing the weight for the majority class. The optimal logistic regression model was found by tuning the hyperparameters, including the norm of penalty (l1 or l2), regularization term and class weights, using grid search method. The result of the best logistic regression model optimizing for the f1 score is displayed and compared in the result part.

2.1.2 Decision Tree

A Decision Tree is a supervised machine-learning algorithm that can be used for classification problems. It is a flowchart-like tree structure, where each internal node represents a test on a feature, each branch denotes an outcome of the test, and each leaf node holds a class label. The decision tree is constructed by asking sequences of if/else questions that narrow possible values until the model is confident enough to make predictions.

Figure 2 in the next part displays a decision tree with a depth of two, which splits the data set based on the pre-pregnancy weight in lb, income, and prenatal support optimizing for Gini Impurity Score. Intuitively speaking, the more the impurity score decreases after the split, the better the split is. In the split, if an observation satisfies the condition, it goes to the left node. Otherwise, it goes to the right node. After a sufficient number of splits, a sample point would be classified into a class based on the answers to a sequence of if/else questions, which is derived from the values of predictor variables the sample has.

2.1.3 Random Forest

Random Forest (RF) is a supervised machine learning algorithm applied for classification or regression. It is an ensemble learning method by constructing a multitude of decision trees at the training time. For classification problems, the outcome of RF is the class selected by the most trees. The structure of RF is also tree-based, and is very similar to that of decision trees, including decision nodes, leaf nodes, and root nodes. The reason for selecting RF as a candidate model is that the prediction model is easy to interpret. Furthermore, the RF model is less likely to overfit the training data since it samples with replacement. Also, it also tells us the ranking of important features, which helps understand which variable has more weight in the model.

There are 2 important hyperparameters that need to be tuned when building the model. The first one is 'n estimators', which implies how many trees will be in the RF model. The second one is 'max depth', which implies how many splits each tree will have. To tune the hyperparameters of the learner, we applied grid search and used 10-fold cross-validation to find the combination with the best performance.

The RF model makes predictions based on the predictions of the decision trees. It takes the average or the mean of various trees. The logic behind the decision made by each tree is identical to that of the Decision Tree model. The result of the best Random Forest model optimizing for the f1 score is displayed and compared in the result part.

2.1.4 Adaptive Boosting (AdaBoost)

Adaptive Boosting(AdaBoost) is a statistical classification algorithm that can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier.

In this project, AdaBoost is presented for binary classification, in which subsequent weak classifiers(decision tree with one split) are tweaked in favor of those instances misclassified by previous classifiers. At each stage of the AdaBoost algorithm, the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples. Therefore, the loss function for an Adaboost model is the sum of weighted errors of all sample points. Because of the mechanism behind Adaboost, it usually yields a stronger learner with more accurate predictive power. Given that Adaboost is suitable for a binary classification problem and capable of improving the performance of weak learners like decision trees, we decide to include Adaboost as one of the candidate models.

However, the Adaboost model is prone to overfit since it combines multiple weak learners sequentially while shifting more focus on the harder-to-classify examples. Besides, the high dimensionality of the data(with more than 25 predictor variables) also increases the possibility of overfitting. To prevent overfitting, the cross-validation(CV) method is adapted to find the best values for the key hyperparameters, i.e. 'n_estimators' which implies the maximum number of estimators at which boosting is terminated, and 'learning_rate' which implies the weight applied to each classifier at each boosting iteration. The result of the best Adaboost model optimizing for the f1 score is displayed and compared in the result part.

2.1.5 Extreme Gradient Boosting(XGBoost)

Similar to AdaBoosting, Extreme Gradient Boosting (XGBoost) comes from the idea of boosting, which improves a weak model by combining it with some other weak models to create a stronger model. It is a sequential algorithm where errors made in a model are considered when building next models. Gradient boosting uses a gradient descent algorithm to additively generate weak models, where gradient or derivative of the loss function of a previous model is used to improve iterative models. In our case, a gradient boosting train ensemble of shallow decision trees with only a few splits (thus creating a weak model), where in each iteration misclassified data points of the previous model are taken into account when fitting the next model, preventing the next model to make the same mistake the previous model made.

Also similar to AdaBoost, the model can easily overfit, where the model would fit the training data perfectly, but perform poorly in a withheld test set. It is always good to control for overfitting in order to get an accurate prediction. Cross validation with 5 folds was performed to select the best parameter for two of the model's hyperparameters, one that controls for model complexity and one that decides how much to punish wrong classifications on the minority class. Since the data used for the purpose of this paper is also imbalance, the model that was built was catered to classifying imbalance classes. The hyperparameter that gives the model with the best F1 score was selected as the best model for the purpose of this paper.

2.2 Model Evaluation

Many metrics can be used for evaluating the performance of a binary classification model, such as accuracy, f1 score, precision score, recall score, and ROC-AUC curve. From the EDA of the data, we learned that the target variable in the data set is severely imbalanced, in which 98.6% of the participants didn't report any diabetes history during the pregnancy period (class 0), while only 1.4% reported diabetes history(class 1). Also in practical problems, it is almost impossible to maximize both precision and recall at the same time since there is a trade-off between precision and recall. Therefore, to better balance the trade-off between precision and recall, we choose to use the f1 score as the metric when comparing the performance of learners. By definition, the f1 score is a measure of the performance of classification models, which combines the precision and recall of a classifier into a single metric by taking their harmonic mean. To provide a generic evaluation of model performance, metrics including accuracy, f1 score, precision, and recall generated from each model are reported in Table 2 for your reference.

3 Result

3.1 Data Overview

The data set contains 7626 rows of observations and 31 features, 25 of which were used as predictor variables in our project. Each row of observations represents a mother-to-be's personal and physical information such as age, race, emotional support, financial support, delivery support, perceived prenatal stress total(psstotal), systolic, diastolic, and whether the patient has diabetes. The data was collected through interviews, self-administered questionnaires, clinical measurements, ultrasounds, and medical records. The summary statistics of predictor variables are displayed in the baseline table (Table 1) by showing the median or percentage of the features in the data set.

Based on the exploring data analysis, we found that the data suffers severe imbalance problems. Due to an imbalance, the model becomes biased toward the majority class. Therefore, we utilized the upsampling technique to create artificial data points of the minority class to balance the target class label. Furthermore, we used the cross-validation technique to prevent overfitting and construct models with higher generalization ability.

See more summary statistics of predictor variables in Table 1. Note that for better display, variables with more than 7 unique values are summarized in a categorical variable style in this table, but are not necessarily treated as categorical variables in the modeling.

Variable Name	Median (IQR) or Percent
History of type 1 or type 2 diabetes, noted in visit 1(percentage,1:Yes, 0:No)	1:1.42, 0:98.57
Age(year)	23.0(28.0,31.0)
Race created from race/ethnicity questions being asked at different time points,screening and visit 1.(text)	Diabetes:(white:41.28,black:34.86,hispanic:11.01, other:9.17,native:3.67), Non-Diabetes:(white:51.16,black:13.96,hispanic:13.18, other:20.23,native:1.46)
Do you expect your partner to give you emotional support during this pregnancy(percentage, 1='yes', 2='no or not applicable')	Diabetes:(1: 88.99, 0:11.01), Non-Diabetes:(1:94.44, 0:5.56)
Do you expect your partner to give you prenatal visit support during this pregnancy(percentage, 1='yes', 2='no or not applicable')	Diabetes:(1: 85.32, 0:14.68), Non-Diabetes:(1:90.3, 0: 9.7)
Do you expect your partner to give you financial support during this pregnancy(percentage, 1='yes', 2='no or not applicable')	Diabetes:(1: 73.39, 0:26.61), Non-Diabetes:(1: 89.0, 0: 11.0)
Do you expect your partner to give you delivery support during this pregnancy(percentage, 1='yes', 2='no or not applicable')	Diabetes:(1: 88.99, 0:11.01), Non-Diabetes:(1:94.32, 0:5.68)
Perceived Stress Scale total score(level)	30.0(28.0,32.0)
State Trait Anxiety-Trait Subscale total score(level)	34.0(30.0,40.0)
Family related worries total score derived from 3 kinds of worries(level)	5.0(4.0,5.0)
Did you participate in any physical activityies or exercises in the last 4 weeks(percentage, 1: yes, 2: no)	Diabetes:(2:74.31,1:25.69), Non-Diabetes:(2:72.02,1:27.98)

Systolic blood pressure measurement taken at visit 1(mmHg)	110.0(100.0,117.0)
Diastolic blood pressure measurement taken at visit 1(mmHg)	68.0(60.0,72.0)
Edinburgh Postpartum Depression Survey total score, taken at visit 1(level)	5.0(3.0,8.0)
The level of worries related to healthcare issues in pregnancy composed of two related worry-based questions(level)	Diabetes:(2:52.29,3:26.61,4:12.84,5:5.50, 6:2.75), Non-Diabetes:(0:0.03,1:0.11,2;56.50,3:24.92, 4:12.31,5:4.35,6:1.80)
The level of worries related to physical symptoms in pregnancy composed of six related worry-based questions(level)	9.0(7.0,10.0)
The mean score among 12 questions in the Multidimensional Scale of Perceived Social Support Questionnaire. The higher, the more the participant agrees with the questions(level)	6.59(6.0,7.0)
Pre-pregnancy weight(lbs)	140.0(125.0,167.0)
Is there anyone in your family that ever had preeclampsia(percentage, 1:Yes, 2:No, 0:Don't know/Missing)	Diabetes:(3:77.06, 2:18.35,1:4.59), Non-Diabetes:(3:84.12,2:10.63,1:5.25)
The income level of the total family income in the past 12 months. The higher the level is, the higher the income is(level, 0:Don't know/Refused)	10.0(4.0,12.0)
History of kidney disease noted in visit 1(percentage,1='yes', 2='no or not applicable')	Diabetes:(2:98.17,1:1.83), Non-Diabetes:(2:98.35,1:1.65)
History of Lupus disease noted in visit 1(percentage,1='yes', 2='no or not applicable')	Diabetes:(2:99.08,1:0.92), Non-Diabetes:(2:99.80,1:0.20)
History of collagen vasucular disease (autoimmune disease) noted in visit 1(percentage,1='yes', 2='no or not applicable')	Diabetes:(2:97.25,1:2.75), Non-Diabetes:(2:98.30,1:1.70)
History of Crohns disease/Ulcerative colitis noted in visit 1(percentage,1='yes', 2='no or not applicable')	Diabetes:(2:99.08,1:0.92), Non-Diabetes:(2:99.14,1:0.86)
History of PCOS (polycystic ovarian syndrome) or other gynecological conditions noted in visit 1(percentage,1='yes', 2='no or not applicable'):	Diabetes:(2:86.24,1:13.76), Non-Diabetes:(2:95.70,1:4.30)
Sum of yes/no answers to Major Experiences of Discrimination (NSAL and SASH version). Higher score means more instances of experiencing discrimination	1.0(1.0,2.0)
Non-Diabetes:(3:85.79,2:8.69,1:5.52)	

Table 1: Median and interquartile range for numeric variables, or percent of each categorical variable of all explanatory variables used for modeling

3.2 Model Interpretation

Table 2 summarizes the performance of all the best models from each of the five learners on the test set.

As we mentioned before, f1-score is adapted for the evaluation across different models. Based on the results displayed in Table 2, the best predictive model is XGBoost model. However, as a boosting model that ensembles multiple decision trees with different weights, XGBoost model is not easily interpretable. Therefore, we would provide interpretations of both the best decision tree model obtained and the best XGBoost model in the following part.

		Logistic Regression	Decision Tree	Random Forest	AdaBoost	XGBoost
Overall	Accuracy	0.97	0.98	0.98	0.93	0.97
	F1 score	**0.14**	**0.10**	**0.17**	**0.08**	**0.20**
Class 0	Precision	0.99	0.99	0.99	0.99	0.99
	Recall	0.98	0.99	1.00	0.94	0.98
	F1 score	0.98	0.99	0.99	0.96	0.99
Class 1	Precision	0.11	0.11	1.00	0.05	0.18
	Recall	0.18	0.09	0.09	0.22	0.23
	F1 score	0.14	0.10	0.17	0.08	0.20

Table 2: Summary of the evaluation results of each learner on the test set

3.2.1 Decision Tree

To interpret the result obtained from a decision tree, let's first take a look at Figure 2 as an example to interpret how a decision tree (with depth=2) makes predictions. In Figure 2, the splits represent that, for a pregnant woman with pre-pregnancy weight(prepreglbs) smaller than 207.5lb and income less than level 4.5, or pre-pregnancy weight(prepreglbs) is more than 207.5lb, the patient will be classified into class 1 (prone to having diabetes). On the other hand, if with prepreglbs less than 207.5lb and income greater than level 4.5, the patient will be classified class=0 (not likely to have diabetes).

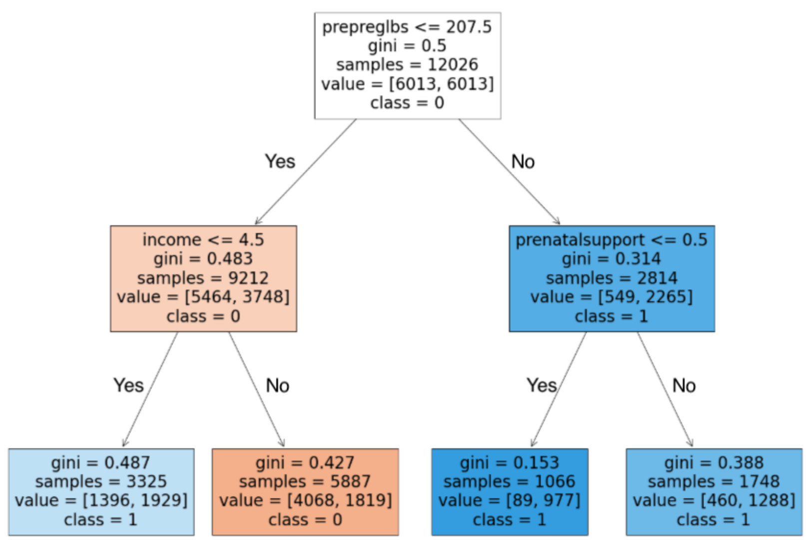


Figure 2: A Decision Tree Process Visualization

However, it's not easy to interpret a decision tree with a higher depth. Alternatively, we can study which feature is more important when making the classification, and how each variable contributes to the response variable via Shapley Additive exPlanations (SHAP) value, which can quantify how a variable contributes to the response variable. Intuitively speaking, negative/positive SHAP values correspond to how the increase of a predictor variable would negatively/positively influence the response variable. The larger the absolute SHAP values is, the more impact a variable has on the response variable.

Figure 3 and Figure 4 are generated based on the best decision tree we've developed so far. In Figure 3, the x-axis represents the average of the absolute SHAP value, and y-axis represents the feature names. SHAP plot orders feature importance from the highest to the lowest. The higher the absolute SHAP value, the more contribution the feature made to the model, which could either be positive or native. In Figure 3, we can learn that pre-pregnancy weight measured in lbs (prepreglbs) contributes the most to the model, followed by income, diastolic, systolic, and perceived Stress Scale total score (psstotal).

Figure 4 better illustrates the positiveness and negativeness of the feature contribution to the response variable. In this plot, the collection of dots represents individual sample points in the training data. X-axis represents the SHAP value, and the y-axis represents the feature names, while the color of the dots encodes the values of each variable. We can learn from Figure 4 that, for the prepregleb predictor, the red dots are more concentrated on the right hand, while the blue dots are concentrated on the left hand. This tendency indicates that pregnant women with higher pre-pregnancy weight tend to have higher SHAP values, thus are more prone to be class 1 (with the history of diabetes). This finding aligns well with the common sense we have about the relationship between weight and the risk of having diabetes.

Since race variable is treated as separate binary variables in the decision tree model, interpretations on that can be different from the others. For example, the race_black variable is a categorical feature that indicates whether the patient's race is black or not. For this variable, red dots are concentrated on the right-hand side of the x-axis, while blue dots are concentrated on the left side. This tendency indicates that the patients identified as black are more likely to be predicted as class 1. Similarly, we can easily get interpretations of how the other variables contribute to the response variables.

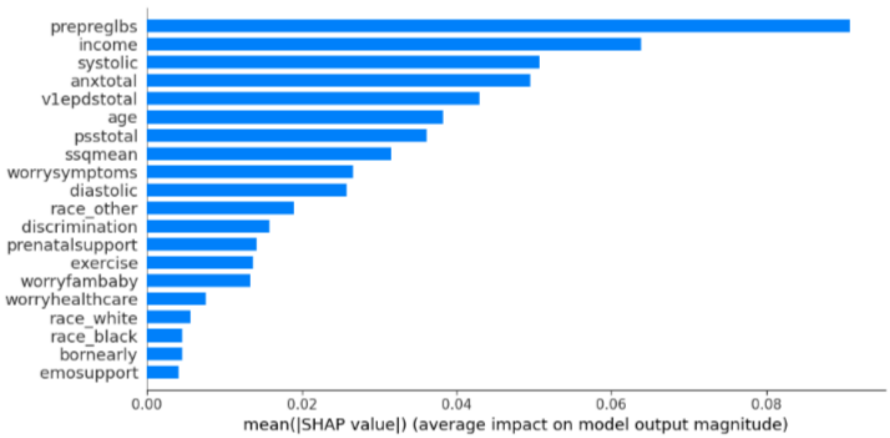


Figure 3: SHAP global feature importance plot for Decision Tree classifier

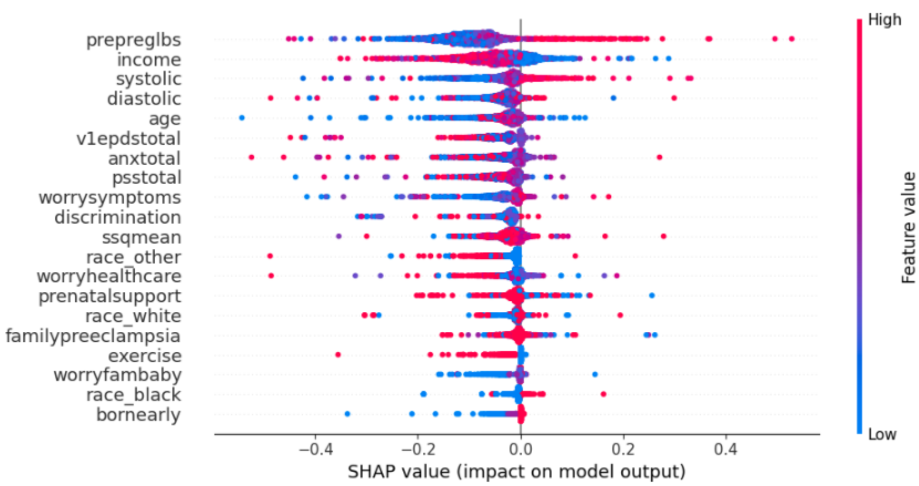


Figure 4: SHAP summary plot for the Decision Tree classifier

3.2.2 XGBoost

As shown in Table 2, XGBoost gives the best performance(the highest f1-score) for predicting gestational diabetes among pregnant women, based on various factors considered. Due to its ensembled nature, XGBoost may not be as easily interpreted. However, a feature importance plot can still be generated for this model, as shown in Figure 5 below. As before, SHAP value was used to indicate the importance of the different variables in predicting gestational diabetes among pregnant women. It can be seen from Figure 5 that pre-pregnancy weight, income, systolic blood measurement taken at the first visit, age of a pregnant woman, and the level of worrying health care are the top 5 variables that contribute the most to the prediction of gestational diabetes.

Similarly, we can learn about the impacts that each predictor variable has on the response variable from Figure 6. According to the XGBoost model, on average, a higher pre-pregnancy weight, systolic blood measurement, and age value results in a higher predicted risk of having diabetes. On the contrary, the higher the family income is, the less likely that the pregnant woman would develop diabetes. As for the race variable, it's worth of notice that, only when a pregnant woman being identified as Black would increase the predicted risk of having diabetes, while those identified as being White, Hispanic, and other races would decrease the predicted risk of having diabetes, according to the XGboost model. As we discussed in class, the bias of the data itself, for example, the sampling method could be biased without reflecting the true distribution of the race group, could account for this discovery. To confirm the validity of this discovery, further investigation on this should be carried out.

Overall, the interpretations of predictor variables obtained from the decision tree and XGBoost models are quite similar, which also align well with the prior knowledge and common sense we hold.

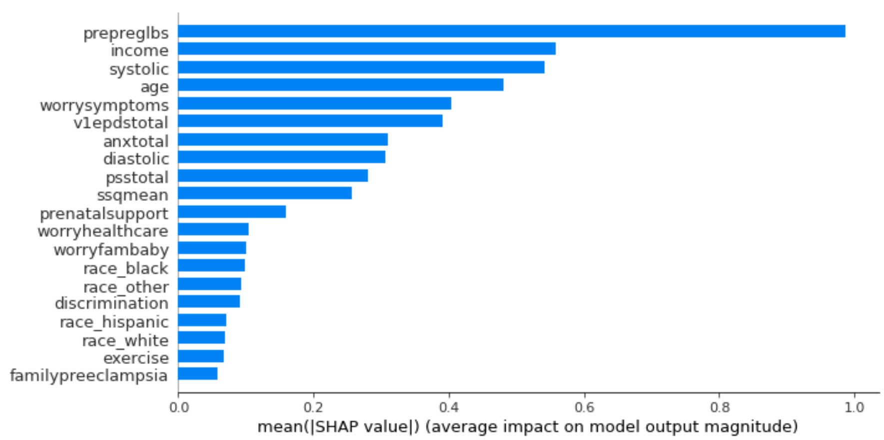


Figure 5: SHAP global feature importance plot for XGBoost model

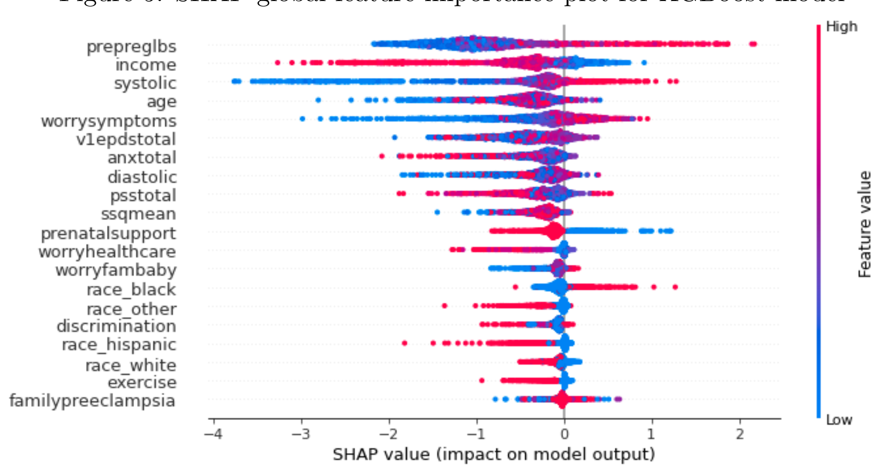


Figure 6: SHAP summary plot for XGBoost model

4 Conclusion

To accomplish the goal of predicting whether a pregnant woman would develop diabetes during the pregnancy, we adapted five learners from each of the logistic regression, decision tree, random forest, AdaBoost and XGBoost based on the mother-to-be data. The best learner was XGBoost which achieved the best performance with the f1-score of 0.20 and the overall accuracy of 0.97 on the test set, while other learners gave relatively lower f1-scores.

According to the best XGBoost model, the first major finding was that the top 5 features that contributed most to predicting the risk of getting diabetes were: pre-pregnancy weight, the level of income, systolic blood pressure taken at the first visit, a pregnant women's age, and the level of worrying health care during pregnancy. Specifically, pregnant women with a higher level of health care worry, pre-pregnant weight, and systolic blood pressure is more likely to develop diabetes on average, while a higher income level could give a lower predicted risk of getting diabetes. It is worth mentioning that a pregnant woman identified as Black is more likely to be predicted to have a higher risk of developing diabetes compared to one identified as another race. However, this finding could be resulted from the bias of data itself, such as the sampling bias.

One possible limitation in our analysis is the severe class imbalance. Though the oversampling technique was applied before the training phase, we still got low precisions and recalls on the minority class, which means the learner is more likely to classify samples that are positive(class 1) as negative(class 0). If time permits, other resampling techniques such as synthetic minority over-sampling could be adapted to achieve better performance in the minority class. Another limitation is derived from the black-box nature of XGBoost model. Even though SHAP plots can help improve the interpretability of ensemble models, the XGBoost model can hardly provide us with quantitative interpretations of the variable effects that are as understandable as linear regression does. This is the compromise we made given that better predictive accuracy is favored than interpretability in this case.

5 Appendix

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
import imblearn
from imblearn.over_sampling import RandomOverSampler
```

1. Read Data

In [2]:
```
drive.mount("/content/drive")

# dispaly data source
from pathlib import Path
!ls "/content/drive/My Drive/Stats504"
```

```
Mounted at /content/drive
 EbayLaptopPriceAnalysis.gdoc   Stats504Project5Reprot.gdoc
 nuMoM2bsubset.csv             'Untitled document (1).gdoc'
 Stats504Proj5.gdoc            'Untitled document.gdoc'
 Stats504Proj5.ipynb
```

In [3]:
```
# load data
path = "/content/drive/My Drive/Stats504/nuMoM2bsubset.csv"
df = pd.read_csv(path)
```

In [4]:
```
df.head(10)
```

Out[4]:

	age	race	dv.gestweeks	dv.v3epdstotal	dv.preeclampsia	emosupport	financialsupport
0	31.0	white	42.0	4.0	1	1.0	1.0
1	26.0	black	37.0	7.0	0	1.0	1.0
2	36.0	white	33.0	13.0	1	1.0	1.0
3	19.0	black	39.0	19.0	0	0.0	0.0
4	20.0	white	38.0	12.0	0	1.0	1.0
5	22.0	white	32.0	12.0	3	1.0	1.0
6	24.0	black	37.0	4.0	1	0.0	0.0
7	20.0	white	38.0	5.0	0	1.0	1.0
8	22.0	white	39.0	0.0	0	1.0	1.0
9	17.0	black	39.0	0.0	0	0.0	1.0

10 rows × 31 columns

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7626 entries, 0 to 7625
Data columns (total 31 columns):
```

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   age                 7626 non-null   float64
 1   race                7626 non-null   object
 2   dv.gestweeks        7626 non-null   float64
 3   dv.v3epdstotal      7626 non-null   float64
 4   dv.preeclampsia     7626 non-null   int64
 5   emosupport          7626 non-null   float64
 6   financialsupport    7626 non-null   float64
 7   prenatalsupport     7626 non-null   float64
 8   deliverysupport     7626 non-null   float64
 9   psstotal            7626 non-null   int64
 10  anxtotal            7626 non-null   float64
 11  worryfambaby        7626 non-null   float64
 12  exercise            7626 non-null   float64
 13  systolic            7626 non-null   float64
 14  diastolic           7626 non-null   float64
 15  v1epdstotal         7626 non-null   float64
 16  worryhealthcare     7626 non-null   float64
 17  worrysymptoms       7626 non-null   float64
 18  ssqmean             7626 non-null   float64
 19  prepreglbs          7626 non-null   float64
 20  familypreeclampsia  7626 non-null   float64
 21  income              7626 non-null   float64
 22  dv.hypertension1    7626 non-null   float64
 23  dv.diabetes1        7626 non-null   float64
 24  kidney1             7626 non-null   float64
 25  lupus1              7626 non-null   float64
 26  collagen1           7626 non-null   float64
 27  crohns1             7626 non-null   float64
 28  pcos1               7626 non-null   float64
 29  discrimination      7626 non-null   float64
 30  bornearly           7626 non-null   float64
dtypes: float64(28), int64(2), object(1)
memory usage: 1.8+ MB
```

In [6]:

```
# check basic stats
df.describe()
```

Out[6]:

	age	dv.gestweeks	dv.v3epdstotal	dv.preeclampsia	emosupport	financialsup
count	7626.000000	7626.000000	7626.000000	7626.000000	7626.000000	7626.00(
mean	27.153029	38.905455	5.437057	0.335169	0.943614	0.90:
std	5.758716	1.773349	4.102719	0.717492	0.230681	0.29
min	0.000000	25.000000	0.000000	0.000000	0.000000	0.00(
25%	23.000000	38.000000	2.000000	0.000000	1.000000	1.00(
50%	28.000000	39.000000	5.000000	0.000000	1.000000	1.00(
75%	31.000000	40.000000	8.000000	0.000000	1.000000	1.00(
max	52.000000	43.000000	28.000000	5.000000	1.000000	1.00(

8 rows × 30 columns

In [7]:

```
# check null value
df.isnull().sum()
```

Out[7]:

```
age                        0
```

```
race                    0
dv.gestweeks            0
dv.v3epdstotal          0
dv.preeclampsia         0
emosupport              0
financialsupport        0
prenatalsupport         0
deliverysupport         0
psstotal                0
anxtotal                0
worryfambaby            0
exercise                0
systolic                0
diastolic               0
v1epdstotal             0
worryhealthcare         0
worrysymptoms           0
ssqmean                 0
prepreglbs              0
familypreeclampsia      0
income                  0
dv.hypertension1        0
dv.diabetes1            0
kidney1                 0
lupus1                  0
collagen1               0
crohns1                 0
pcos1                   0
discrimination          0
bornearly               0
dtype: int64
```

2. Exploring Data Analysis(EDA)

1). Drop Irrelevant Target Variables

In [8]:
```
df = df.drop(columns=['dv.gestweeks', 'dv.v3epdstotal', 'dv.preeclampsia', 'd
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7626 entries, 0 to 7625
Data columns (total 27 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   age                 7626 non-null   float64
 1   race                7626 non-null   object
 2   emosupport          7626 non-null   float64
 3   financialsupport    7626 non-null   float64
 4   prenatalsupport     7626 non-null   float64
 5   deliverysupport     7626 non-null   float64
 6   psstotal            7626 non-null   int64
 7   anxtotal            7626 non-null   float64
 8   worryfambaby        7626 non-null   float64
 9   exercise            7626 non-null   float64
 10  systolic            7626 non-null   float64
 11  diastolic           7626 non-null   float64
 12  v1epdstotal         7626 non-null   float64
 13  worryhealthcare     7626 non-null   float64
```

```
 14  worrysymptoms       7626 non-null   float64
 15  ssqmean             7626 non-null   float64
 16  prepreglbs          7626 non-null   float64
 17  familypreeclampsia  7626 non-null   float64
 18  income              7626 non-null   float64
 19  dv.diabetes1        7626 non-null   float64
 20  kidney1             7626 non-null   float64
 21  lupus1              7626 non-null   float64
 22  collagen1           7626 non-null   float64
 23  crohns1             7626 non-null   float64
 24  pcos1               7626 non-null   float64
 25  discrimination      7626 non-null   float64
 26  bornearly           7626 non-null   float64
dtypes: float64(25), int64(1), object(1)
memory usage: 1.6+ MB
```

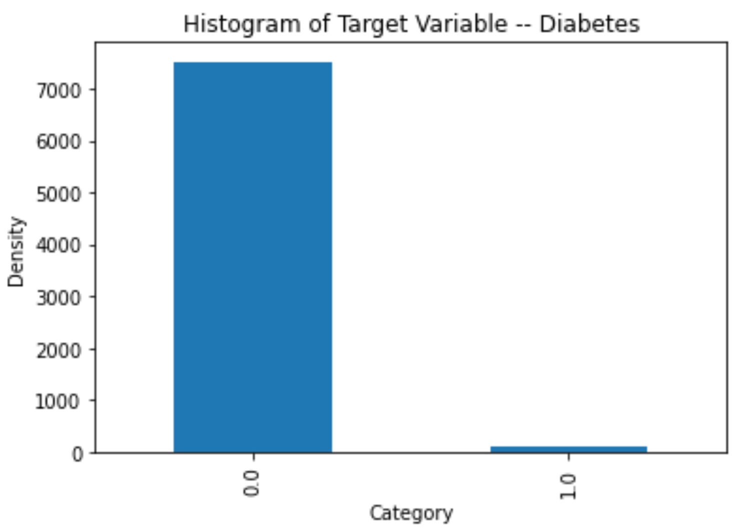
2). Historgam of Target Variable

In [10]:

```
ax = df['dv.diabetes1'].value_counts().plot(kind='bar', title = 'Histogram of
ax.set_xlabel('Category')
ax.set_ylabel('Density')
```

Out[10]:

```
Text(0, 0.5, 'Density')
```



In [11]:

```
df['dv.diabetes1'].value_counts(1)
```

Out[11]:

```
0.0    0.985707
1.0    0.014293
Name: dv.diabetes1, dtype: float64
```

In [12]:

```
# visualizaiton for numerical cols
plt.rcParams['figure.figsize'] = [30, 23]
plt.rcParams['figure.dpi'] = 100
def draw_histograms(df, variables, names, goal, n_rows, n_cols):
    fig=plt.figure()
    for i, var_name in enumerate(variables):
        ax=fig.add_subplot(n_rows,n_cols,i+1)
#         if df[var_name].nunique()>=10:
        up_lim = np.quantile(df[var_name], 0.98)
        sns.kdeplot(data=df[df[goal]==1][var_name], ax=ax, fill=True, label='
        sns.kdeplot(data=df[df[goal]==0][var_name], ax=ax, fill=True,label='D
        ax.set_xlim(left=0, right=up_lim)
#         else:
```

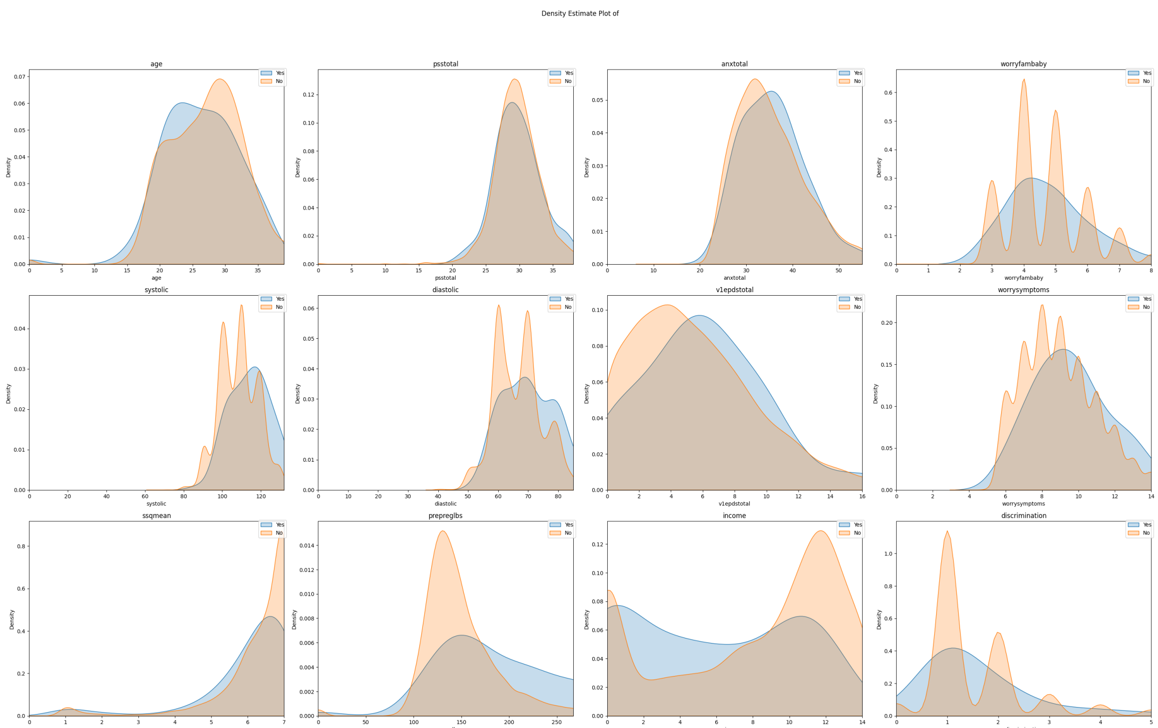
```
#             sns.histplot(data=df[df[goal]==1], ax=ax,x=df[var_name], label=
#             sns.histplot(data=df[df[goal]==0],ax=ax, x=df[var_name], label=
        ax.set_xlabel(names[i])
        ax.set_title(names[i])
        ax.legend(labels=['Yes','No'], bbox_to_anchor=(1.02, 1.02), loc='uppe
    fig.tight_layout()  # Improves appearance a bit.
    plt.suptitle('Density Estimate Plot of',y=1.05)
    plt.show()
```

In [13]:

```
con_vars = list(df.columns[df.nunique()>=8])
y= 'dv.diabetes1'
```

In [14]:

```
# Draw continuous variables
draw_histograms(df, con_vars, con_vars, y, 4, 4)
```



3. Generate Baseline Table

In [15]:

```
df.describe().transpose()
```

Out[15]:

	count	mean	std	min	25%	50%	75%	max
age	7626.0	27.153029	5.758716	0.0	23.0	28.000000	31.0	52.0000
emosupport	7626.0	0.943614	0.230681	0.0	1.0	1.000000	1.0	1.0000
financialsupport	7626.0	0.902308	0.296917	0.0	1.0	1.000000	1.0	1.0000
prenatalsupport	7626.0	0.887752	0.315692	0.0	1.0	1.000000	1.0	1.0000
deliverysupport	7626.0	0.942434	0.232937	0.0	1.0	1.000000	1.0	1.0000
psstotal	7626.0	29.742722	3.555042	0.0	28.0	30.000000	32.0	50.0000
anxtotal	7626.0	35.368870	7.672727	10.0	30.0	34.000000	40.0	72.0000

	count	mean	std	min	25%	50%	75%	max
worryfambaby	7626.0	4.705481	1.237158	0.0	4.0	5.000000	5.0	9.0000
exercise	7626.0	1.279439	0.448753	1.0	1.0	1.000000	2.0	2.0000
systolic	7626.0	109.085497	10.764119	66.0	100.0	110.000000	117.0	165.0000
diastolic	7626.0	67.102282	8.374436	40.0	60.0	68.000000	72.0	111.0000
v1epdstotal	7626.0	5.633360	4.119349	0.0	3.0	5.000000	8.0	25.0000
worryhealthcare	7626.0	2.697482	0.969644	0.0	2.0	2.000000	3.0	6.0000
worrysymptoms	7626.0	9.060451	2.130655	4.0	7.0	9.000000	10.0	18.0000
ssqmean	7626.0	6.214092	1.164837	0.0	6.0	6.583333	7.0	7.0000
prepreglbs	7626.0	150.211372	41.550357	0.0	125.0	140.000000	167.0	374.7854
familypreeclampsia	7626.0	2.787700	0.521698	1.0	3.0	3.000000	3.0	3.0000
income	7626.0	8.001049	4.789058	0.0	4.0	10.000000	12.0	14.0000
dv.diabetes1	7626.0	0.014293	0.118705	0.0	0.0	0.000000	0.0	1.0000
kidney1	7626.0	1.983478	0.127482	1.0	2.0	2.000000	2.0	2.0000
lupus1	7626.0	1.997902	0.045760	1.0	2.0	2.000000	2.0	2.0000
collagen1	7626.0	1.982822	0.129943	1.0	2.0	2.000000	2.0	2.0000
crohns1	7626.0	1.991345	0.092633	1.0	2.0	2.000000	2.0	2.0000
pcos1	7626.0	1.955678	0.205823	1.0	2.0	2.000000	2.0	2.0000
discrimination	7626.0	1.636638	1.187262	0.0	1.0	1.000000	2.0	11.0000
bornearly	7626.0	2.801862	0.519472	1.0	3.0	3.000000	3.0	3.0000

In [16]:

```
# For categorical vars: get percentage of each leve,grouped by y
cat_vars = list(set(df.columns).difference(set(con_vars+[y])))
def get_categorical_percentages(df, cat_vars,y):
    for var in cat_vars:
        df_temp = pd.concat([df[df[y]==1][var].value_counts()/ df[df[y]==1][v
                              df[df[y]==0][var].value_counts()/ df[df[y]==0][v
                             ],
                             axis=1
                             )
        df_temp.columns=[var+'_y=1', var+'_y=0']
        df_temp=np.round(df_temp,4)*100
        print(df_temp.transpose())

get_categorical_percentages(df,cat_vars,y)
```

```
              2.0   1.0
lupus1_y=1  99.08  0.92
lupus1_y=0  99.80  0.20
               2.0   1.0
crohns1_y=1  99.08  0.92
crohns1_y=0  99.14  0.86
                1.0    2.0
exercise_y=1  74.31  25.69
exercise_y=0  72.02  27.98
                       1.0    0.0
deliverysupport_y=1  88.99  11.01
deliverysupport_y=0  94.32   5.68
                         1.0    0.0
```

```
financialsupport_y=1  85.32  14.68
financialsupport_y=0  90.30   9.70
                  2.0   1.0
kidney1_y=1  98.17  1.83
kidney1_y=0  98.35  1.65
            white  black  hispanic  other  native
race_y=1   41.28  34.86     11.01   9.17    3.67
race_y=0   51.16  13.96     13.18  20.23    1.46
                         3.0    2.0   1.0
familypreeclampsia_y=1  77.06  18.35  4.59
familypreeclampsia_y=0  84.12  10.63  5.25
                    2.0   1.0
collagen1_y=1  97.25  2.75
collagen1_y=0  98.30  1.70
                     1.0    0.0
emosupport_y=1  88.99  11.01
emosupport_y=0  94.44   5.56
                    3.0    2.0   1.0
bornearly_y=1  81.65  11.01  7.34
bornearly_y=0  85.79   8.69  5.52
                          1.0    0.0
prenatalsupport_y=1  73.39  26.61
prenatalsupport_y=0  89.00  11.00
                2.0    1.0
pcos1_y=1  86.24  13.76
pcos1_y=0  95.70   4.30
                      0.0    1.0    2.0    3.0    4.0   5.0   6.0
worryhealthcare_y=1   NaN    NaN  52.29  26.61  12.84  5.50  2.75
worryhealthcare_y=0  0.03   0.11  56.50  24.92  12.31  4.35  1.80
```

4. Data Preprocessing

1). Split Data

In [17]:

```python
from sklearn.model_selection import train_test_split

X = df.drop(columns=['dv.diabetes1'])
y = df['dv.diabetes1'].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
```

2). One-Hot-Encoding

In [18]:

```python
X_train = pd.get_dummies(X_train, columns=['race'])
X_test = pd.get_dummies(X_test, columns=['race'])
```

3). Oversampling

In [19]:

```python
oversample = RandomOverSampler(sampling_strategy='minority')
X_train_over, y_train_over = oversample.fit_resample(X_train, y_train)
```

5. Train Models

(1). Decision Tree

In [20]:
```
# import imblearn
# from imblearn.over_sampling import RandomOverSampler

# oversample = RandomOverSampler(sampling_strategy='minority')
# X_train_over, y_train_over = oversample.fit_resample(X_train, y_train)
```

In [21]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import tree

clf = tree.DecisionTreeClassifier(random_state=42)
params = {'max_depth': [int(x) for x in np.linspace(10, 110, num = 11)]}

gdcv_clf = GridSearchCV(clf, params, verbose=1, scoring='f1', cv=5).fit(X_tra

gdcv_clf.best_params_
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
```

Out[21]:
```
{'max_depth': 40}
```

In [22]:
```
tuned_DT  = tree.DecisionTreeClassifier(random_state=42, max_depth=40).fit(X_

y_pred = tuned_DT.predict(X_test)
print("Testing Accuracy = ", tuned_DT.score(X_test, y_test))
print("F1 score = ", f1_score(y_test, y_pred))
print("Precision = ", precision_score(y_test, y_pred))
print("Recall = ", recall_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Testing Accuracy =  0.9705111402359109
F1 score =  0.08163265306122448
Precision =  0.07407407407407407
Recall =  0.09090909090909091
              precision    recall  f1-score   support

         0.0       0.99      0.98      0.99      1504
         1.0       0.07      0.09      0.08        22

    accuracy                           0.97      1526
   macro avg       0.53      0.54      0.53      1526
weighted avg       0.97      0.97      0.97      1526
```

In [23]:
```
!pip install shap
# SHAP plot of XGboost model: X-axis:  predicted y; color: value of predictor
```

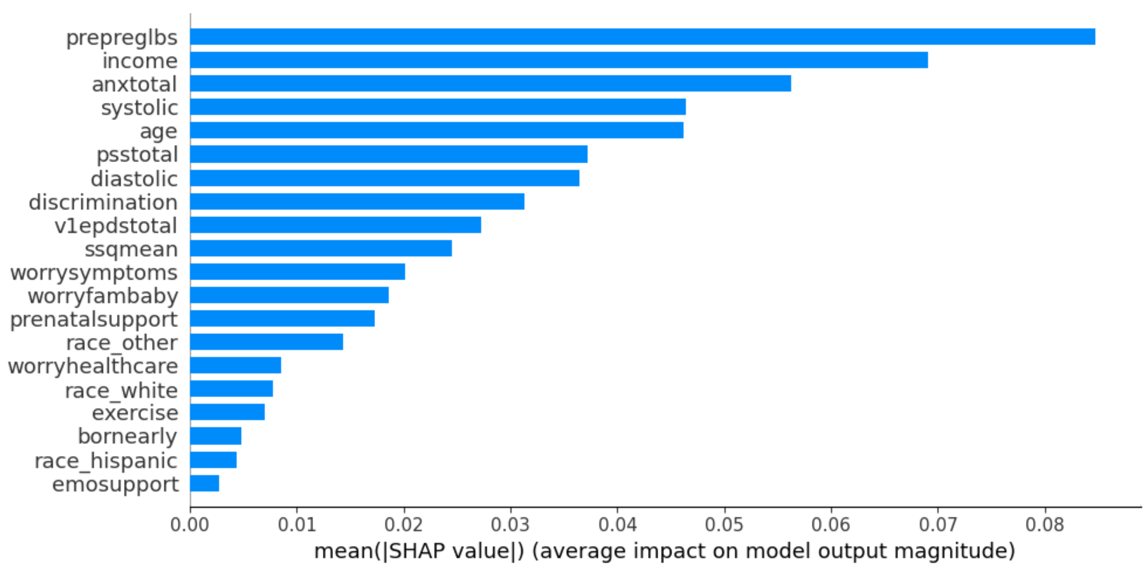
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
heels/public/simple/
Collecting shap
  Downloading shap-0.41.0-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_6
4.whl (569 kB)
     |████████████████████████████████| 569 kB 32.3 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages
(from shap) (1.7.3)
Collecting slicer==0.0.7
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.7/dist-pa
ckages (from shap) (1.5.0)
```

```
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.7/dist
-packages (from shap) (21.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-p
ackages (from shap) (1.0.2)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.7/dist-pa
ckages (from shap) (4.64.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from shap) (1.21.6)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-package
s (from shap) (1.3.5)
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages
(from shap) (0.56.4)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/pyth
on3.7/dist-packages (from packaging>20.9->shap) (3.0.9)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/py
thon3.7/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/
dist-packages (from numba->shap) (4.13.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-pac
kages (from numba->shap) (57.4.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/pyth
on3.7/dist-packages (from importlib-metadata->numba->shap) (4.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-pack
ages (from importlib-metadata->numba->shap) (3.10.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python
3.7/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-p
ackages (from pandas->shap) (2022.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packa
ges (from python-dateutil>=2.7.3->pandas->shap) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-p
ackages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
7/dist-packages (from scikit-learn->shap) (3.1.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.41.0 slicer-0.0.7
```

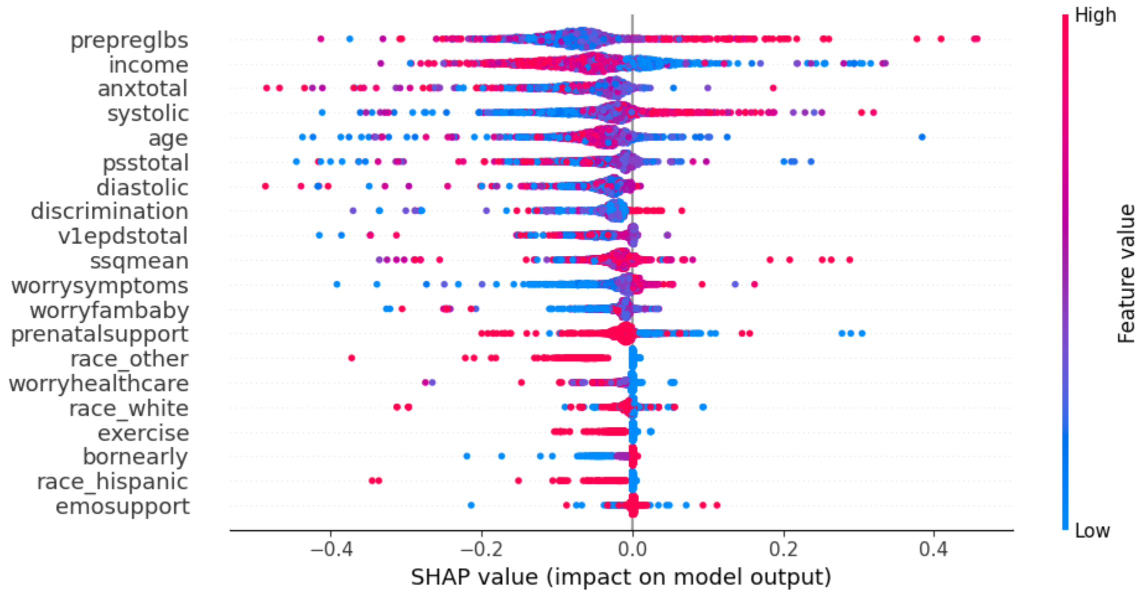
In [25]:

```
import shap

explainer = shap.TreeExplainer(tuned_DT)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[0], X_test, plot_type='bar', plot_size=(10,5))
```



In [26]:
```
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.colum
```



In [27]:
```
# from sklearn.tree import plot_tree
# features = X_train.columns
# target = ['0','1']

# fig = plt.figure(figsize=(25,20))
# _ = tree.plot_tree(tuned_DT,
#                    feature_names=features,
#                    class_names=target,
#                    filled=True)
```

(2). Random Forest

In [28]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rfc = RandomForestClassifier(random_state = 42)
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 1, stop = 50, num = 10)]
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 50, num = 10)]
max_depth.append(None)
random_grid = {'n_estimators': n_estimators,
               'max_depth': max_depth,
}

rfc_random = RandomizedSearchCV(estimator = rfc, param_distributions = random_
rfc_random.fit(X_train_over, y_train_over)
rfc_random.best_params_
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[28]:
```
{'n_estimators': 44, 'max_depth': 39}
```

In [29]:
```
rfc_tuned = RandomForestClassifier(random_state = 42, n_estimators = 44, max_
rfc_tuned.fit(X_train_over, y_train_over)
```

Out[29]:
```
RandomForestClassifier(max_depth=39, n_estimators=44, random_state=42)
```

In [30]:

```
y_train_pred = rfc_tuned.predict(X_train)
y_train_pred_over = rfc_tuned.predict(X_train_over)
y_test_pred = rfc_tuned.predict(X_test)
rfc_train = accuracy_score(y_train, y_train_pred)
rfc_train_over = accuracy_score(y_train_over, y_train_pred_over)
rfc_test = accuracy_score(y_test, y_test_pred)

print(f"Random Forest train/train after oversampling/test accuracies:{rfc_tra
print("F1 score = ", f1_score(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
```

```
Random Forest train/train after oversampling/test accuracies:1.000/1.000/0.987
F1 score =  0.16666666666666669
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      1504
         1.0       1.00      0.09      0.17        22

    accuracy                           0.99      1526
   macro avg       0.99      0.55      0.58      1526
weighted avg       0.99      0.99      0.98      1526
```

(3). Adaboost

In [31]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
le = LabelEncoder()
```

In [32]:

```
import imblearn
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import classification_report
```

In [33]:

```
# Highly imbalanced data =>
# oversampling to balance the labels
oversample = RandomOverSampler(sampling_strategy='minority')
X_train_over, y_train_over = oversample.fit_resample(X_train, y_train)
```

In [34]:

```
# Model tuning
#  tune_ada = AdaBoostClassifier(base_estimator=ada,random_state=42)
# parameters = {
#                'n_estimators':[3000,4000, 5000],
#                'learning_rate':[0.8,1.0,1.2]
# }
# clf = GridSearchCV(tune_ada, parameters, verbose=3, scoring='f1',n_jobs=-1,
# clf.fit(X_train, y_train)
# clf.best_estimator_
```

In [35]:

```
# the best Adaboost model we got so far; More tuning required(but time-consum
ada = AdaBoostClassifier(
           n_estimators=5000,
           learning_rate=1.0,
           random_state=42)
```

```
ada = ada.fit(X_train_over, y_train_over)
# make predictions on train/test data
y_train_pred = ada.predict(X_train)
y_train_pred_over = ada.predict(X_train_over)
y_test_pred = ada.predict(X_test)
ada_train = accuracy_score(y_train, y_train_pred)
ada_train_over = accuracy_score(y_train_over, y_train_pred_over)
ada_test = accuracy_score(y_test, y_test_pred)
print(f"Adaboost train/train after oversampling/test accuracies:{ada_train_ov
print("F1 score = ", f1_score(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
```

```
Adaboost train/train after oversampling/test accuracies:0.972/0.944/0.926
F1 score =  0.08130081300813008
              precision    recall  f1-score   support

         0.0       0.99      0.94      0.96      1504
         1.0       0.05      0.23      0.08        22

    accuracy                           0.93      1526
   macro avg       0.52      0.58      0.52      1526
weighted avg       0.97      0.93      0.95      1526
```

(4). XGBoost

In [36]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
```

Using `GridSearchCV` to run cross validation to select which parameters give the model with the best F1 score.

In [37]:

```
xgboost = XGBClassifier(random_state=1)
param_grid = {'n_estimators': [700],
              'objective': ['binary:logistic'],
              'max_depth': [15],
             'min_child_weight':[1],
             'gamma':[15,20,25,30],
             'scale_pos_weight':[2,3,4]}

xgb_cv = GridSearchCV(estimator=xgboost,
                      param_grid=param_grid,
                      n_jobs=-1,
                      cv=5,
                      scoring='f1_micro')
xgb_cv.fit(X_train_over,y_train_over)
xgb_cv.best_params_
```

Out[37]:

```
{'gamma': 15,
 'max_depth': 15,
 'min_child_weight': 1,
 'n_estimators': 700,
 'objective': 'binary:logistic',
 'scale_pos_weight': 2}
```

Fitting model with the best parameters derived above

In [38]:

```
xgb = XGBClassifier(random_state=1, gamma=15,
 max_depth= 15,
 min_child_weight=1,
```

```
 n_estimators = 700,
 objective = 'binary:logistic',
    scale_pos_weight = 2)
xgb.fit(X_train_over, y_train_over)
```

Out[38]:
```
XGBClassifier(gamma=15, max_depth=15, n_estimators=700, random_state=1,
              scale_pos_weight=2)
```

In [39]:
```
y_train_pred = xgb.predict(X_train)
y_train_pred_over = xgb.predict(X_train_over)
y_test_pred = xgb.predict(X_test)
xgboost_train = accuracy_score(y_train, y_train_pred)
xgboost_train_over = accuracy_score(y_train_over, y_train_pred_over)
xgboost_test = accuracy_score(y_test, y_test_pred)
print("F1 score = ", f1_score(y_test, y_test_pred))
print(f"Adaboost train/train after oversampling/test accuracies:{xgboost_trai
print(classification_report(y_test, y_test_pred))
```

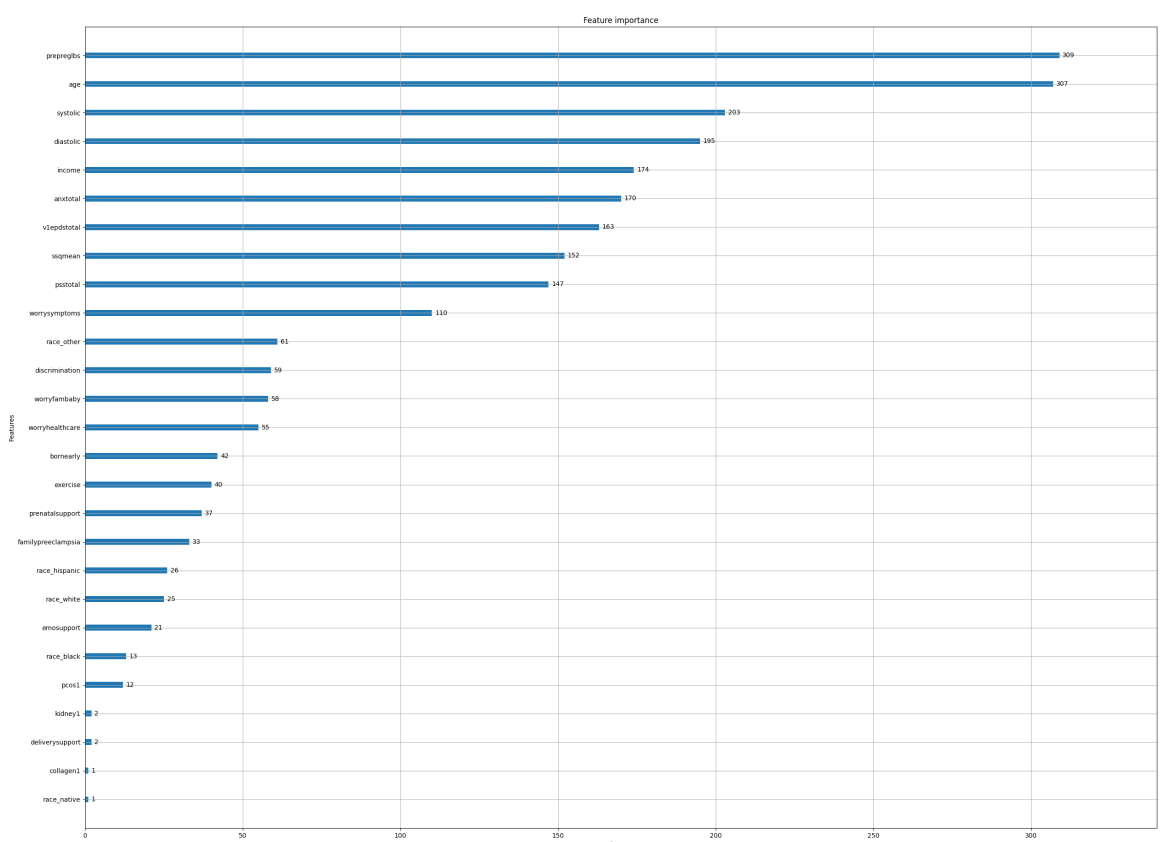
```
F1 score =  0.1702127659574468
Adaboost train/train after oversampling/test accuracies:0.997/0.993/0.974
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99      1504
         1.0       0.16      0.18      0.17        22

    accuracy                           0.97      1526
   macro avg       0.57      0.58      0.58      1526
weighted avg       0.98      0.97      0.98      1526
```

In [40]:
```
from xgboost import plot_importance
plot_importance(xgb)
```

Out[40]:
```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7fde34160650&gt;
```



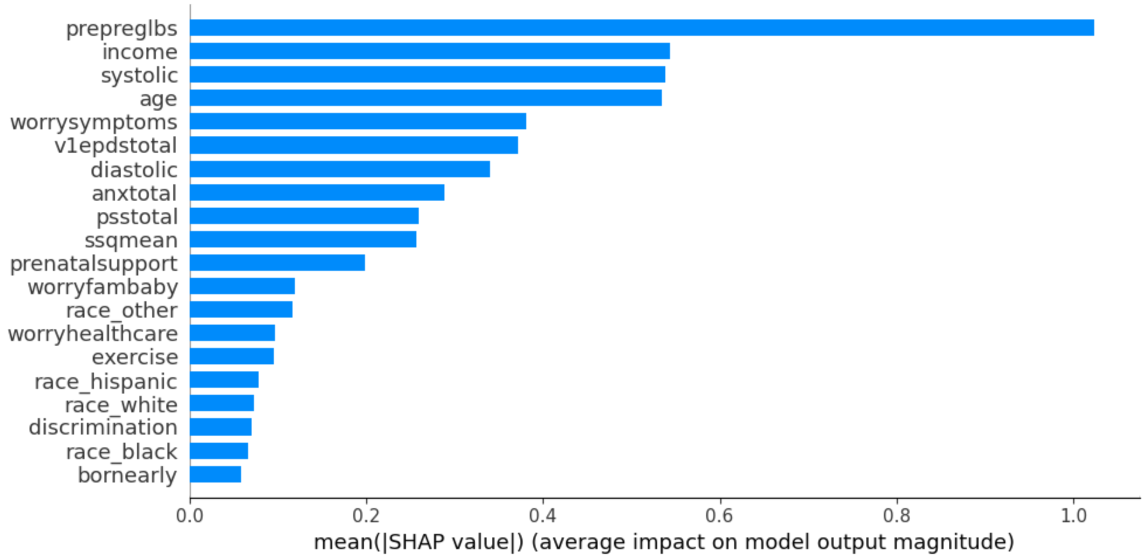
In [ ]:

```
# import shap

# explainer = shap.TreeExplainer(tuned_DT)
# shap_values = explainer.shap_values(X_test)
# shap.summary_plot(shap_values, X_test, plot_type='bar', plot_size=(10,5))
```

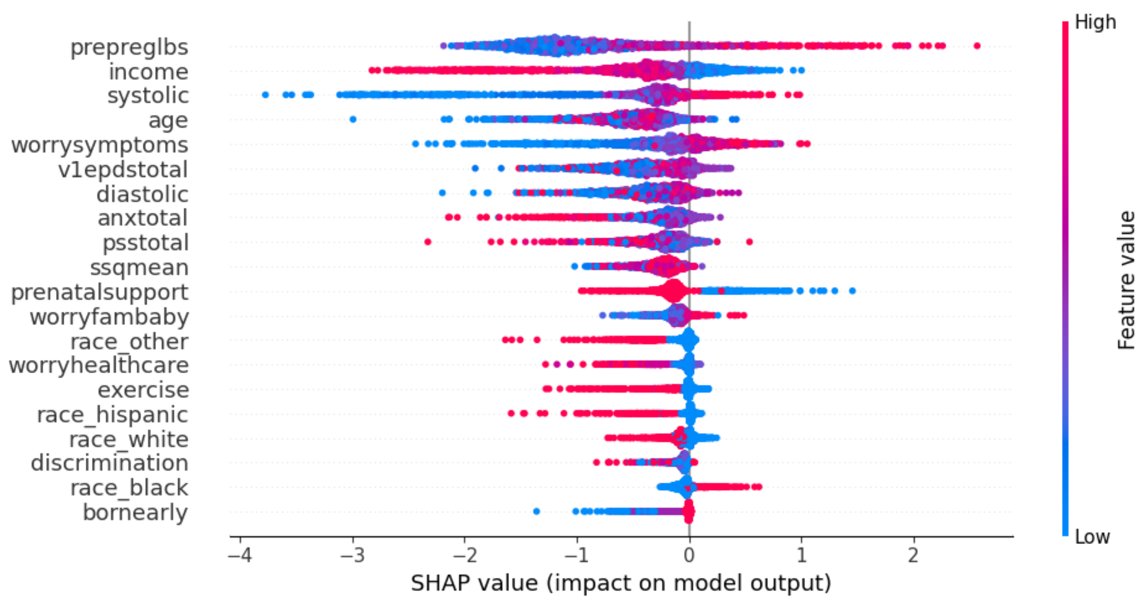
In [41]:

```
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test, plot_type='bar', plot_size=(10,5))
```



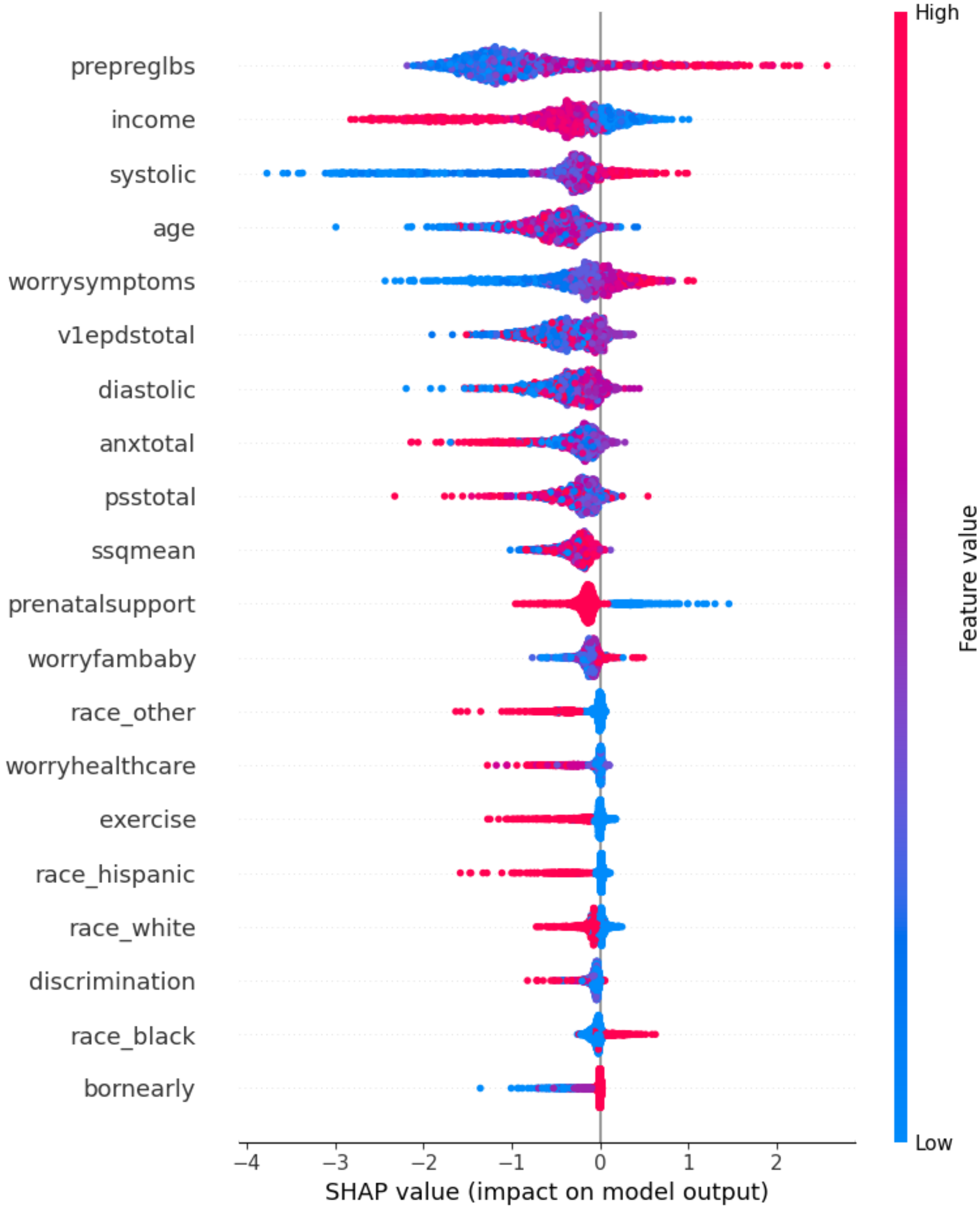
In [42]:

```
shap.summary_plot(shap_values, X_test.values, feature_names = X_test.columns,
```



In [43]:

```
# !pip install shap
# SHAP plot of XGboost model: X-axis: predicted y; color: value of predictor
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```


prepreglbs
income
systolic
age
worrysymptoms
v1epdstotal
diastolic
anxtotal
psstotal
ssqmean
prenatalsupport
worryfambaby
race_other
worryhealthcare
exercise
race_hispanic
race_white
discrimination
race_black
bornearly
−4
−3
−2
−1
0
1
2
SHAP value (impact on model output)
High
Feature value
Low

Logistic Regression

1 Imputation

In [44]:
```
df[df['age']==0] # 29 - unknown or refuse to answer
```

Out[44]:

	age	race	emosupport	financialsupport	prenatalsupport	deliverysupport	psstotal	anxtotal	worryfambaby	exercise	...	familypreecla
3307	0.0	white	1.0	1.0	0.0	1.0	33	51.0	8.0	1.0	...	
3782	0.0	hispanic	1.0	1.0	1.0	1.0	23	34.0	4.0	1.0	...	
3830	0.0	hispanic	1.0	1.0	1.0	1.0	30	23.0	4.0	1.0	...	
3919	0.0	other	1.0	1.0	1.0	1.0	30	47.0	5.0	1.0	...	
3920	0.0	other	1.0	1.0	1.0	1.0	30	47.0	5.0	1.0	...	
5418	0.0	other	1.0	0.0	1.0	1.0	29	31.0	5.0	1.0	...	
5517	0.0	white	1.0	1.0	1.0	1.0	29	37.0	5.0	2.0	...	
6056	0.0	other	1.0	1.0	1.0	1.0	30	24.0	4.0	1.0	...	
6059	0.0	hispanic	1.0	1.0	1.0	1.0	33	31.0	5.0	2.0	...	
6120	0.0	hispanic	1.0	1.0	1.0	1.0	25	23.0	4.0	2.0	...	
6123	0.0	white	1.0	1.0	1.0	1.0	31	26.0	4.0	1.0	...	
6126	0.0	other	1.0	1.0	1.0	1.0	26	34.0	4.0	1.0	...	
6128	0.0	other	1.0	1.0	1.0	1.0	30	35.0	5.0	1.0	...	
6389	0.0	white	1.0	1.0	1.0	1.0	35	41.0	6.0	1.0	...	
6604	0.0	hispanic	1.0	1.0	1.0	1.0	24	36.0	5.0	1.0	...	
7052	0.0	white	1.0	1.0	1.0	1.0	27	37.0	4.0	1.0	...	
7176	0.0	white	1.0	1.0	1.0	1.0	27	41.0	4.0	1.0	...	
7438	0.0	other	1.0	1.0	1.0	1.0	27	30.0	4.0	2.0	...	
7441	0.0	white	1.0	1.0	1.0	1.0	31	36.0	6.0	2.0	...	
7442	0.0	white	1.0	1.0	1.0	1.0	31	36.0	6.0	2.0	...	
7462	0.0	other	1.0	1.0	1.0	1.0	29	26.0	4.0	1.0	...	
7477	0.0	hispanic	1.0	1.0	1.0	1.0	30	31.0	4.0	2.0	...	
7478	0.0	other	1.0	1.0	1.0	1.0	35	46.0	4.0	1.0	...	
7491	0.0	other	1.0	1.0	1.0	1.0	29	26.0	6.0	1.0	...	
7498	0.0	other	1.0	1.0	1.0	1.0	28	34.0	4.0	2.0	...	
7500	0.0	other	1.0	1.0	1.0	1.0	33	32.0	6.0	1.0	...	
7528	0.0	other	1.0	1.0	1.0	1.0	29	28.0	6.0	1.0	...	
7554	0.0	other	1.0	1.0	1.0	1.0	27	30.0	4.0	1.0	...	
7572	0.0	other	1.0	1.0	1.0	1.0	30	38.0	5.0	1.0	...	

29 rows × 27 columns

In [50]:
```
age_mean = df[df['age'] != 0]['age'].mean()
age_mean
```

Out[50]: 27.25668026852705

In [47]:
```
df['age'].mean()
```

Out[47]: 27.15302911093627

In [51]:
```
df['age'].unique()
```

Out[51]:
```
array([31., 26., 36., 19., 20., 22., 24., 17., 21., 18., 33., 28., 35.,
       27., 25., 29., 30., 32., 23., 40., 34., 38., 16., 39., 37., 15.,
       41., 42., 44., 50., 52., 13., 14.,  0., 43., 45.])
```

In [61]:

```
# impute by mean
df_imputed = df.copy()
df_imputed['age'] = np.where(df_imputed['age'] == 0, 27, df_imputed['age'])
df_imputed['age'].mean()
```

Out[61]: 27.255704169944924

2 Check VIF

In [19]:

```
# reduce the categories of psstotal according to the data dictionary
df1= df.copy()
df1['psstotal'] = np.where(df['psstotal'] <= 13, 'low', df1['psstotal'])
df1['psstotal'] = np.where((df['psstotal'] > 13) & (df['psstotal'] <= 26), 'moderate', df1['psstotal'])
df1['psstotal'] = np.where(df['psstotal'] > 26, 'high', df1['psstotal'])
```

In [20]:

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df2 = df1.copy()
df2['race'] = label_encoder.fit_transform(df2['race'])
df2['psstotal'] = label_encoder.fit_transform(df2['psstotal'])
```

In [21]:

```
cols =  df2.columns.drop('dv.diabetes1')
X = df2.loc[:, list(cols)] # all features
```

In [22]:
```
#calculate Variance Inflation Factor
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_scores = pd.DataFrame()
vif_scores["Attribute"] = X.columns

# calculating VIF for each feature
vif_scores["VIF Scores"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]

display(vif_scores) # >10 strongly correlated
```

	Attribute	VIF Scores
0	age	36.317604
1	race	5.965398
2	emosupport	81.355202
3	financialsupport	20.414518
4	prenatalsupport	16.623742
5	deliverysupport	82.257258
6	psstotal	1.238376
7	anxtotal	46.701654
8	worryfambaby	24.655324
9	exercise	9.781021
10	systolic	162.952384
11	diastolic	101.227872
12	v1epdstotal	5.757395
13	worryhealthcare	11.994058
14	worrysymptoms	28.587209
15	ssqmean	32.659996
16	prepreglbs	17.099090
17	familypreeclampsia	30.063360
18	income	7.127112
19	kidney1	231.787188
20	lupus1	812.782626
21	collagen1	220.602517
22	crohns1	399.111682
23	pcos1	90.473656
24	discrimination	3.143673
25	bornearly	31.497966

In [64]:
```
# after imputation
df11= df_imputed.copy()
df11['psstotal'] = np.where(df_imputed['psstotal'] <= 13, 'low', df11['psstotal'])
df11['psstotal'] = np.where((df_imputed['psstotal'] > 13) & (df_imputed['psstotal'] <= 26), 'moderate', df1
df11['psstotal'] = np.where(df_imputed['psstotal'] > 26, 'high', df11['psstotal'])

df3 = df11.copy()
df3['race'] = label_encoder.fit_transform(df3['race'])
df3['psstotal'] = label_encoder.fit_transform(df3['psstotal'])

cols =  df3.columns.drop('dv.diabetes1')
X1 = df3.loc[:, list(cols)] # all features

vif_scores1 = pd.DataFrame()
vif_scores1["Attribute"] = X1.columns

# calculating VIF for each feature
vif_scores1["VIF Scores"] = [variance_inflation_factor(X1.values, i) for i in range(len(X1.columns))]

display(vif_scores1) # >10 strongly correlated
# VIF of age increases
```

	Attribute	VIF Scores
0	age	42.590463
1	race	5.966523
2	emosupport	81.362088
3	financialsupport	20.416366
4	prenatalsupport	16.623040
5	deliverysupport	82.259714
6	psstotal	1.238372
7	anxtotal	46.701044
8	worryfambaby	24.679624
9	exercise	9.781225
10	systolic	162.950455
11	diastolic	101.226600
12	v1epdstotal	5.757975
13	worryhealthcare	11.995465
14	worrysymptoms	28.609225
15	ssqmean	32.643134
16	prepreglbs	17.112231
17	familypreeclampsia	30.065622
18	income	7.495516
19	kidney1	231.806394
20	lupus1	814.381265
21	collagen1	220.594378
22	crohns1	399.147699
23	pcos1	90.436615
24	discrimination	3.145087
25	bornearly	31.502776

3 Encoding Categorical Variables

In [23]:
```
df1['bornearly'].unique()
```
Out[23]:
```
array([3., 2., 1.])
```

In [24]:
```
cat_vars = ['race', 'emosupport', 'financialsupport', 'prenatalsupport', 'deliverysupport', 'psstotal', 'ex
            'familypreeclampsia', 'kidney1', 'lupus1', 'collagen1', 'crohns1', 'pcos1', 'bornearly']
for var in cat_vars:
    cat_list = 'var'+'_'+var
    cat_list = pd.get_dummies(df1[var], prefix=var, drop_first=True)
    df_after_dummy = df1.join(cat_list)
    df1 = df_after_dummy
```

In [25]:
```
to_keep = [i for i in list(df_after_dummy.columns) if i not in cat_vars]
df_final = df1[to_keep]
```

In [26]:
```
df_final.columns
```
Out[26]:
```
Index(['age', 'anxtotal', 'worryfambaby', 'systolic', 'diastolic',
       'v1epdstotal', 'worryhealthcare', 'worrysymptoms', 'ssqmean',
       'prepreglbs', 'income', 'dv.diabetes1', 'discrimination',
       'race_hispanic', 'race_native', 'race_other', 'race_white',
       'emosupport_1.0', 'financialsupport_1.0', 'prenatalsupport_1.0',
       'deliverysupport_1.0', 'psstotal_low', 'psstotal_moderate',
       'exercise_2.0', 'familypreeclampsia_2.0', 'familypreeclampsia_3.0',
       'kidney1_2.0', 'lupus1_2.0', 'collagen1_2.0', 'crohns1_2.0',
       'pcos1_2.0', 'bornearly_2.0', 'bornearly_3.0'],
      dtype='object')
```

In [65]:
```
# after imputation
df_final_imputed = df_final.copy()
df_final_imputed['age'] = df_imputed['age']
```

4 Split Data

In [27]:
```
from sklearn.model_selection import train_test_split

X = df_final.drop(columns=['dv.diabetes1'])
y = df_final['dv.diabetes1'].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
```

In [68]:
```
# after imputation
X_imputed = df_final_imputed.drop(columns=['dv.diabetes1'])
y_imputed = df_final_imputed['dv.diabetes1'].copy()

X_train_imputed, X_test_imputed, y_train_imputed, y_test_imputed = train_test_split(X_imputed, y_imputed, t
```

5 Normalization

Normalize the data after splitting to avoid information leaking

In [28]:
```
num_vars = [i for i in list(df.columns.drop('dv.diabetes1')) if i not in cat_vars]
other_vars = [i for i in list(df_final.columns.drop('dv.diabetes1')) if i not in num_vars]
```

In [29]:
```
from sklearn.preprocessing import StandardScaler

scale = StandardScaler()
_ = pd.DataFrame(scale.fit_transform(X_train[num_vars]), columns=num_vars, index=X_train[other_vars].index
X_train_scaled = _.join(X_train[other_vars])
X_test_scaled = pd.DataFrame(scale.transform(X_test[num_vars]), columns=num_vars,
                             index=X_test[other_vars].index).join(X_test[other_vars])
```

In [30]:
```
X_train_scaled
```

Out[30]:

	age	anxtotal	worryfambaby	systolic	diastolic	v1epdstotal	worryhealthcare	worrysymptoms	ssqmean	prepreglbs	...	exercis
5176	-1.574141	0.737676	-1.369109	-0.105032	-0.607330	-1.124826	1.351726	0.436849	0.179216	0.162464	...	
5182	-0.365267	0.477495	0.238662	-1.033443	-1.799936	-0.881814	0.315855	0.436849	-0.600741	-0.246371	...	
2840	1.361696	-0.563227	0.238662	0.080650	0.585276	-1.124826	1.351726	-0.032467	0.675551	-0.414715	...	
3878	0.670911	0.867766	1.042547	-0.476397	-0.845851	1.305292	0.315855	1.375482	0.675551	-0.534961	...	
6025	0.325518	-1.213679	-1.369109	-1.033443	0.108234	-1.124826	-0.720015	-1.440417	0.321026	-0.294469	...	
...	...	...	...	...	...	...	...	...	...	...	...	
79	-0.192571	1.778399	0.238662	0.916220	0.108234	1.791316	-0.720015	-0.971100	-0.742551	1.196577	...	
3927	0.843607	0.477495	0.238662	0.637697	1.539361	-0.152779	-0.720015	0.436849	0.604646	1.124430	...	
5955	-0.019874	-0.172956	-0.565224	1.937472	0.108234	-0.152779	0.315855	-0.501784	0.675551	1.052283	...	
6936	-0.365267	0.347405	-0.565224	1.009061	0.823798	1.062280	-0.720015	1.844798	0.675551	0.475103	...	
5640	0.325518	1.518218	0.238662	0.637697	1.420101	0.333245	0.315855	0.436849	0.533741	-3.613251	...	

6100 rows × 32 columns

In [69]:
```
_ = pd.DataFrame(scale.fit_transform(X_train_imputed[num_vars]), columns=num_vars, index=X_train_imputed[ot
Xi_train_scaled = _.join(X_train_imputed[other_vars])
Xi_test_scaled = pd.DataFrame(scale.transform(X_test_imputed[num_vars]), columns=num_vars,
                              index=X_test_imputed[other_vars].index).join(X_test_imputed[other_vars])
```

In [70]:
```
Xi_train_scaled
```

Out[70]:

	age	anxtotal	worryfambaby	systolic	diastolic	v1epdstotal	worryhealthcare	worrysymptoms	ssqmean	prepreglbs	...	exercis
5176	-1.678697	0.737676	-1.369109	-0.105032	-0.607330	-1.124826	1.351726	0.436849	0.179216	0.162464	...	
5182	-0.406212	0.477495	0.238662	-1.033443	-1.799936	-0.881814	0.315855	0.436849	-0.600741	-0.246371	...	
2840	1.411624	-0.563227	0.238662	0.080650	0.585276	-1.124826	1.351726	-0.032467	0.675551	-0.414715	...	
3878	0.684490	0.867766	1.042547	-0.476397	-0.845851	1.305292	0.315855	1.375482	0.675551	-0.534961	...	
6025	0.320922	-1.213679	-1.369109	-1.033443	0.108234	-1.124826	-0.720015	-1.440417	0.321026	-0.294469	...	
...	...	...	...	...	...	...	...	...	...	...	...	
79	-0.224428	1.778399	0.238662	0.916220	0.108234	1.791316	-0.720015	-0.971100	-0.742551	1.196577	...	
3927	0.866273	0.477495	0.238662	0.637697	1.539361	-0.152779	-0.720015	0.436849	0.604646	1.124430	...	
5955	-0.042645	-0.172956	-0.565224	1.937472	0.108234	-0.152779	0.315855	-0.501784	0.675551	1.052283	...	
6936	-0.406212	0.347405	-0.565224	1.009061	0.823798	1.062280	-0.720015	1.844798	0.675551	0.475103	...	
5640	0.320922	1.518218	0.238662	0.637697	1.420101	0.333245	0.315855	0.436849	0.533741	-3.613251	...	

6100 rows × 32 columns

6 Modelling

6.1 Best fit

In [39]:
```python
# tune hyperparameter
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, StratifiedKFold

lr = LogisticRegression(solver='liblinear')

# handle imbalance
# tuning weight for minority class then weight for majority class will be 1-weight of minority class
# Setting the range for class weights
weights = np.linspace(0.0,0.99,500)

# other parameters
param = {'C': [0.1, 0.5, 1,10,15,20], 'penalty': ['l1', 'l2'],
         "class_weight":[{0: x, 1: 1.0-x} for x in weights]}

# create 5 folds
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

#Gridsearch for hyperparam tuning
lr_model = GridSearchCV(estimator= lr, param_grid=param, scoring="f1", cv=folds, return_train_score=True)
lr_model.fit(X_train_scaled, y_train)
```

Out[39]:
```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [40]:
```python
print("Best F1 score: ", lr_model.best_score_)
print("Best hyperparameters: ", lr_model.best_params_)
```

```
Best F1 score:  0.14809523809523809
Best hyperparameters:  {'C': 20, 'class_weight': {0: 0.069438877755551102, 1: 0.930561122244489}, 'penalt
y': 'l2'}
```

In [79]:
```python
# build model that has the best performance
best_lr = LogisticRegression(class_weight={0: 0.0694, 1: 0.9306}, C=20, penalty='l2',
                            solver='liblinear').fit(X_train_scaled, y_train)
```

In [85]:
```
feature_importance = pd.DataFrame({'feature':list(X_train_scaled.columns),
                                   'feature_importance':[abs(i) for i in best_lr.coef_[0]]})
feature_importance.sort_values('feature_importance',ascending=False)
```

Out[85]:

	feature	feature_importance
18	prenatalsupport_1.0	1.172730
20	psstotal_low	0.869668
12	race_hispanic	0.830604
14	race_other	0.819206
26	lupus1_2.0	0.764835
28	crohns1_2.0	0.614105
29	pcos1_2.0	0.586461
22	exercise_2.0	0.540446
16	emosupport_1.0	0.458528
10	income	0.455659
27	collagen1_2.0	0.440106
13	race_native	0.433133
15	race_white	0.410792
17	financialsupport_1.0	0.401908
7	worrysymptoms	0.394175
1	anxtotal	0.381653
9	prepreglbs	0.354640
3	systolic	0.353740
23	familypreeclampsia_2.0	0.345140
31	bornearly_3.0	0.262011
25	kidney1_2.0	0.175388
5	v1epdstotal	0.120013
2	worryfambaby	0.097542
24	familypreeclampsia_3.0	0.088843
6	worryhealthcare	0.086577
30	bornearly_2.0	0.075790
4	diastolic	0.064933
21	psstotal_moderate	0.052974
11	discrimination	0.044386
8	ssqmean	0.032712
0	age	0.014291
19	deliverysupport_1.0	0.001293

6.1.1 after imputation

In [72]:
```
lr1 = LogisticRegression(solver='liblinear')
lr_model1 = GridSearchCV(estimator= lr1, param_grid=param, scoring="f1", cv=folds, return_train_score=True
lr_model1.fit(Xi_train_scaled, y_train_imputed)
```

Out[72]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [73]:
```
print("Best F1 score: ", lr_model1.best_score_)
print("Best hyperparameters: ", lr_model1.best_params_)
```

```
Best F1 score:  0.14913528164302156
Best hyperparameters:  {'C': 10, 'class_weight': {0: 0.07340681362725451, 1: 0.9265931863727455}, 'penalt
y': 'l2'}
```

In [77]:
```
# build model that has the best performance
best_lr1 = LogisticRegression(class_weight={0: 0.0734, 1: 0.9266}, C=20, penalty='l2',
                              solver='liblinear').fit(Xi_train_scaled, y_train_imputed)
```

In [83]:
```
feature_importance = pd.DataFrame({'feature':list(Xi_train_scaled.columns),
                                   'feature_importance':[abs(i) for i in best_lr1.coef_[0]]})
feature_importance.sort_values('feature_importance',ascending=False)
```

Out[83]:

	feature	feature_importance
18	prenatalsupport_1.0	1.154580
20	psstotal_low	0.889713
14	race_other	0.855292
12	race_hispanic	0.842913
26	lupus1_2.0	0.757078
28	crohns1_2.0	0.618716
29	pcos1_2.0	0.574520
22	exercise_2.0	0.538227
10	income	0.508039
15	race_white	0.431488
27	collagen1_2.0	0.429636
16	emosupport_1.0	0.419614
13	race_native	0.418140
7	worrysymptoms	0.394720
17	financialsupport_1.0	0.389046
1	anxtotal	0.380608
3	systolic	0.352680
9	prepreglbs	0.347589
23	familypreeclampsia_2.0	0.340685
31	bornearly_3.0	0.264727
25	kidney1_2.0	0.169117
5	v1epdstotal	0.122555
0	age	0.119418
24	familypreeclampsia_3.0	0.097609
2	worryfambaby	0.086196
6	worryhealthcare	0.085073
30	bornearly_2.0	0.079195
21	psstotal_moderate	0.060183
4	diastolic	0.053894
11	discrimination	0.033972
8	ssqmean	0.026466
19	deliverysupport_1.0	0.022788

3.6.1.2 Oversampler

In [87]:
```
from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy='minority')
X_train_over, y_train_over = oversample.fit_resample(X_train_imputed, y_train_imputed)
```

In [88]:
```
lr2 = LogisticRegression(solver='liblinear')
param1 = {'C': [0.1, 0.5, 1,10,15,20], 'penalty': ['l1', 'l2']}
lr_model2 = GridSearchCV(estimator= lr2, param_grid=param1, scoring="f1", cv=folds, return_train_score=True
lr_model2.fit(X_train_over, y_train_over)
```

Out[88]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [93]:
```
print("Best F1 score: ", lr_model2.best_score_)
print("Best hyperparameters: ", lr_model2.best_params_)
```

```
Best F1 score:  0.7494015922535697
Best hyperparameters:  {'C': 0.1, 'penalty': 'l1'}
```

In [94]:
```
# build model that has the best performance
best_lr2 = LogisticRegression(C=0.1, penalty='l1',solver='liblinear').fit(X_train_over, y_train_over)
```

6.2 Evaluation

In [80]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report

# care about FN

y_train_pred = best_lr.predict(X_train_scaled)
y_test_pred = best_lr.predict(X_test_scaled)
logr_train = accuracy_score(y_train, y_train_pred)
logr_test = accuracy_score(y_test, y_test_pred)
print(f"Logistic train/test accuracies: {logr_train:.3f}/{logr_test:.3f}")
print(classification_report(y_test, y_test_pred))
```

```
Logistic train/test accuracies: 0.965/0.961
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98      1504
         1.0       0.09      0.18      0.12        22

    accuracy                           0.96      1526
   macro avg       0.54      0.58      0.55      1526
weighted avg       0.97      0.96      0.97      1526
```

In [91]:
```
# after imputation
yi_train_pred = best_lr1.predict(Xi_train_scaled)
yi_test_pred = best_lr1.predict(Xi_test_scaled)
logr_traini = accuracy_score(y_train_imputed, yi_train_pred)
logr_testi = accuracy_score(y_test_imputed, yi_test_pred)
print(f"Logistic train/test accuracies: {logr_traini:.3f}/{logr_testi:.3f}")
print(classification_report(y_test_imputed, yi_test_pred))
```

```
Logistic train/test accuracies: 0.967/0.967
              precision    recall  f1-score   support

         0.0       0.99      0.98      0.98      1504
         1.0       0.11      0.18      0.14        22

    accuracy                           0.97      1526
   macro avg       0.55      0.58      0.56      1526
weighted avg       0.98      0.97      0.97      1526
```

In [95]:
```
# use oversampler (not to tune class weights)
yo_train_pred = best_lr2.predict(X_train_over)
yo_test_pred = best_lr2.predict(Xi_test_scaled)
logr_traino = accuracy_score(y_train_over, yo_train_pred)
logr_testo = accuracy_score(y_test_imputed, yo_test_pred)
print(f"Logistic train/test accuracies: {logr_traino:.3f}/{logr_testo:.3f}")
print(classification_report(y_test_imputed, yo_test_pred))
```

```
Logistic train/test accuracies: 0.749/0.986
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      1504
         1.0       0.00      0.00      0.00        22

    accuracy                           0.99      1526
   macro avg       0.49      0.50      0.50      1526
weighted avg       0.97      0.99      0.98      1526



/opt/anaconda3/envs/504/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMet
ricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/504/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMet
ricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/504/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMet
ricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [96]:
```
from sklearn.metrics import f1_score
f1_score(y_test_imputed, yi_test_pred)
```

Out[96]:
```
0.13793103448275862
```