MEX #7 - Geyzson Kristoffer

SN:2023-21036

https://uvle.upd.edu.ph/mod/assign/view.php?id=547271

Problem #1

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tqdm as notebook_tqdm

diabetes = pd.read_csv('diabetes_data_upload.csv')
diabetes.head()
```

Out[ ]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	s
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	

In [ ]:

```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 520 non-null    int64 
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    object
 3   Polydipsia          520 non-null    object
 4   sudden weight loss  520 non-null    object
 5   weakness            520 non-null    object
 6   Polyphagia          520 non-null    object
 7   Genital thrush      520 non-null    object
 8   visual blurring     520 non-null    object
 9   Itching             520 non-null    object
 10  Irritability        520 non-null    object
 11  delayed healing     520 non-null    object
 12  partial paresis     520 non-null    object
 13  muscle stiffness    520 non-null    object
 14  Alopecia            520 non-null    object
 15  Obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

In [ ]:

```
import optuna
import pandas as pd
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score
```

```
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import make_scorer, f1_score, precision_score, recall_score
from ast import literal_eval

df = diabetes.copy()

df['class'] = df['class'].map({'Positive': 1, 'Negative': 0})
X = df.drop('class', axis=1)
y = df['class']

# 1a.
# encoding the labels and splitting the data into train and test sets
le = LabelEncoder()
X = X.apply(lambda col: le.fit_transform(col.astype(str)), axis=0, result_type='expand')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# running optuna
def objective(trial):
    classifier_name = trial.suggest_categorical('classifier', ['MLPClassifier', 'RandomForest', 'XGBoost', 'Logis

    if classifier_name == 'MLPClassifier':
        hidden_layer_sizes_str = trial.suggest_categorical('hidden_layer_sizes', ['(50, 50, 50)', '(50, 100, 50)
        hidden_layer_sizes = eval(hidden_layer_sizes_str)
        params = {
            'hidden_layer_sizes': hidden_layer_sizes,
            'activation': trial.suggest_categorical('activation', ['tanh', 'relu']),
            'solver': trial.suggest_categorical('solver', ['sgd', 'adam']),
            'max_iter': trial.suggest_int('max_iter', 1000, 2000),
```

```
            'batch_size': trial.suggest_int('batch_size', 1, 100),
            'learning_rate_init': trial.suggest_float('learning_rate_init', 1e-5, 1e-2),
            'alpha': trial.suggest_float('alpha', 1e-5, 1e-2),
            'shuffle': trial.suggest_categorical('shuffle', [True, False]),
            'tol': trial.suggest_float('tol', 1e-5, 1e-2),
            'momentum': trial.suggest_float('momentum', 1e-5, 1e-2),
            'early_stopping': trial.suggest_categorical('early_stopping', [True, False]),
        }
        model = MLPClassifier(**params)

    elif classifier_name == 'RandomForest':
        params = {
            'n_estimators': trial.suggest_int('n_estimators', 10, 1000),
            'max_depth': trial.suggest_int('max_depth', 1, 50),
            'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy']),
            'min_samples_split': trial.suggest_int('min_samples_split', 2, 14),
            'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 14),
            'bootstrap': trial.suggest_categorical('bootstrap', [True, False]),

        }
        model = RandomForestClassifier(**params)

    elif classifier_name == 'XGBoost':
        params = {
            'n_estimators': trial.suggest_int('n_estimators', 10, 1000),
            'max_depth': trial.suggest_int('max_depth', 1, 50),
            'learning_rate': trial.suggest_float('learning_rate', 1e-8, 1.0),
            'booster': trial.suggest_categorical('booster', ['gbtree', 'gblinear', 'dart']),
            'reg_alpha': trial.suggest_float('reg_alpha', 1e-8, 1.0),
            'reg_lambda': trial.suggest_float('reg_lambda', 1e-8, 1.0),
            'scale_pos_weight': trial.suggest_float('scale_pos_weight', 1e-6, 1e6),

        }
        model = xgb.XGBClassifier(**params)
```

```
    elif classifier_name == 'LogisticRegression':
        params = {
            'C': trial.suggest_float('C', 1e-8, 1e2),
            'max_iter': trial.suggest_int('max_iter', 1000, 2000),
        }
        model = LogisticRegression(**params)

    elif classifier_name == 'NaiveBayes':
        params = {
            'var_smoothing': trial.suggest_float('var_smoothing', 1e-8, 1e-2),
        }
        model = GaussianNB(**params)

    elif classifier_name == 'SVM':
        params = {
            'C': trial.suggest_float('C', 1e-8, 1e2),
            'kernel': trial.suggest_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid']),
            'degree': trial.suggest_int('degree', 1, 10),
            'gamma': trial.suggest_categorical('gamma', ['scale', 'auto']),
        }
        model = SVC(**params)

    elif classifier_name == 'kNN':
        params = {
            'n_neighbors': trial.suggest_int('n_neighbors', 1, 50),
            'weights': trial.suggest_categorical('weights', ['uniform', 'distance']),
            'algorithm': trial.suggest_categorical('algorithm', ['auto', 'ball_tree', 'kd_tree', 'brute']),
            'leaf_size': trial.suggest_int('leaf_size', 1, 50),
        }
        model = KNeighborsClassifier(**params)

    f1_scorer = make_scorer(f1_score, average='weighted')
    precision_scorer = make_scorer(precision_score)
    recall_scorer = make_scorer(recall_score)
```

```
    score = cross_val_score(model, X_train, y_train, n_jobs=-1, cv=10).mean()
    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

```
[I 2024-01-23 04:13:03,199] A new study created in memory with name: no-name-373802b0-5de5-46d8-a3ba-862fc63116dc
[I 2024-01-23 04:13:04,083] Trial 0 finished with value: 0.9179442508710803 and parameters: {'classifier': 'RandomForest', 'n_estimators': 714, 'max_depth': 13, 'criterion': 'entropy', 'min_samples_split': 10, 'min_samples_leaf': 7, 'bootstrap': True}. Best is trial 0 with value: 0.9179442508710803.
[I 2024-01-23 04:13:04,111] Trial 1 finished with value: 0.870150987224158 and parameters: {'classifier': 'kNN', 'n_neighbors': 28, 'weights': 'uniform', 'algorithm': 'kd_tree', 'leaf_size': 8}. Best is trial 0 with value: 0.9179442508710803.
[I 2024-01-23 04:13:04,124] Trial 2 finished with value: 0.92061556329849 and parameters: {'classifier': 'LogisticRegression', 'C': 27.595157339340947, 'max_iter': 1996}. Best is trial 2 with value: 0.92061556329849.
[I 2024-01-23 04:13:09,391] Trial 3 finished with value: 0.5986062717770034 and parameters: {'classifier': 'XGBoost', 'n_estimators': 334, 'max_depth': 24, 'learning_rate': 0.0012840005839864727, 'booster': 'dart', 'reg_alpha': 0.9937318537545634, 'reg_lambda': 0.42952362274475325, 'scale_pos_weight': 462376.23478665506}. Best is trial 2 with value: 0.92061556329849.
[I 2024-01-23 04:13:09,420] Trial 4 finished with value: 0.5986062717770034 and parameters: {'classifier': 'XGBoost', 'n_estimators': 87, 'max_depth': 33, 'learning_rate': 0.012497103059991016, 'booster': 'gbtree', 'reg_alpha': 0.6783680116086803, 'reg_lambda': 0.21517235739232524, 'scale_pos_weight': 917579.8056591077}. Best is trial 2 with value: 0.92061556329849.
[I 2024-01-23 04:13:14,958] Trial 5 finished with value: 0.9591173054587688 and parameters: {'classifier': 'XGBoost', 'n_estimators': 338, 'max_depth': 23, 'learning_rate': 0.9940632260094652, 'booster': 'dart', 'reg_alpha': 0.5798397611120658, 'reg_lambda': 0.4914104804487928, 'scale_pos_weight': 633309.4550088117}. Best is trial 5 with value: 0.9591173054587688.
[I 2024-01-23 04:13:15,882] Trial 6 finished with value: 0.9179442508710803 and parameters: {'classifier': 'RandomForest', 'n_estimators': 870, 'max_depth': 30, 'criterion': 'gini', 'min_samples_split': 6, 'min_samples_leaf': 8, 'bootstrap': True}. Best is trial 5 with value: 0.9591173054587688.
[I 2024-01-23 04:13:15,905] Trial 7 finished with value: 0.8990127758420442 and parameters: {'classifier': 'kNN', 'n_neighbors': 22, 'weights': 'uniform', 'algorithm': 'brute', 'leaf_size': 45}. Best is trial 5 with value: 0.9591173054587688.
[I 2024-01-23 04:13:15,919] Trial 8 finished with value: 0.9686411149825783 and parameters: {'classifier': 'SVM', 'C': 73.51594749868863, 'kernel': 'poly', 'degree': 5, 'gamma': 'scale'}. Best is trial 8 with value: 0.968641114
```

```
M', 'C': 27.704642683069313, 'kernel': 'poly', 'degree': 7, 'gamma': 'scale'}. Best is trial 72 with value: 0.973
4610917537747.
[I 2024-01-23 04:13:27,853] Trial 91 finished with value: 0.96869918699187 and parameters: {'classifier': 'SVM',
'C': 29.41524851447025, 'kernel': 'poly', 'degree': 5, 'gamma': 'scale'}. Best is trial 72 with value: 0.97346109
17537747.
[I 2024-01-23 04:13:27,871] Trial 92 finished with value: 0.96869918699187 and parameters: {'classifier': 'SVM',
'C': 32.428161703792775, 'kernel': 'poly', 'degree': 5, 'gamma': 'scale'}. Best is trial 72 with value: 0.9734610
917537747.
[I 2024-01-23 04:13:27,891] Trial 93 finished with value: 0.96869918699187 and parameters: {'classifier': 'SVM',
'C': 24.70164581031176, 'kernel': 'poly', 'degree': 5, 'gamma': 'scale'}. Best is trial 72 with value: 0.97346109
17537747.
[I 2024-01-23 04:13:27,975] Trial 94 finished with value: 0.9639372822299652 and parameters: {'classifier': 'XGBo
ost', 'n_estimators': 459, 'max_depth': 20, 'learning_rate': 0.21262116901970451, 'booster': 'gbtree', 'reg_alph
a': 0.004478368589993398, 'reg_lambda': 0.7087295921203218, 'scale_pos_weight': 995491.8979060121}. Best is trial
72 with value: 0.9734610917537747.
[I 2024-01-23 04:13:28,104] Trial 95 finished with value: 0.9182346109175377 and parameters: {'classifier': 'SV
M', 'C': 38.30714314119352, 'kernel': 'linear', 'degree': 6, 'gamma': 'scale'}. Best is trial 72 with value: 0.97
34610917537747.
[I 2024-01-23 04:13:29,151] Trial 96 finished with value: 0.9253193960511034 and parameters: {'classifier': 'Rand
omForest', 'n_estimators': 959, 'max_depth': 36, 'criterion': 'gini', 'min_samples_split': 8, 'min_samples_leaf':
8, 'bootstrap': True}. Best is trial 72 with value: 0.9734610917537747.
[I 2024-01-23 04:13:29,168] Trial 97 finished with value: 0.96869918699187 and parameters: {'classifier': 'SVM',
'C': 6.736812205057969, 'kernel': 'poly', 'degree': 5, 'gamma': 'scale'}. Best is trial 72 with value: 0.97346109
17537747.
[I 2024-01-23 04:13:29,188] Trial 98 finished with value: 0.9567363530778163 and parameters: {'classifier': 'SV
M', 'C': 86.71665705653152, 'kernel': 'poly', 'degree': 4, 'gamma': 'auto'}. Best is trial 72 with value: 0.97346
10917537747.
[I 2024-01-23 04:13:29,207] Trial 99 finished with value: 0.9400696864111499 and parameters: {'classifier': 'kN
N', 'n_neighbors': 33, 'weights': 'distance', 'algorithm': 'brute', 'leaf_size': 32}. Best is trial 72 with valu
e: 0.9734610917537747.
```

In [ ]:
```
best_trial = study.best_trial
print(f"Best trial final score: {best_trial.value}")
for key, value in best_trial.params.items():
    print(f"{key}: {value}")
```

```
best_classifier = best_trial.params['classifier']
```

```
Best trial final score: 0.9734610917537747
classifier: SVM
C: 2.9090105588450754
kernel: poly
degree: 5
gamma: scale
```

In [ ]:
```
best_params = study.best_trial.params
best_params
```

Out[ ]:
```
{'classifier': 'SVM',
 'C': 2.9090105588450754,
 'kernel': 'poly',
 'degree': 5,
 'gamma': 'scale'}
```

In [ ]:
```
if best_params['classifier'] == 'MLPClassifier':
    model = MLPClassifier(**{k: v for k, v in best_params.items() if k != 'classifier'})
elif best_params['classifier'] == 'RandomForest':
    model = RandomForestClassifier(**{k: v for k, v in best_params.items() if k != 'classifier'})
elif best_params['classifier'] == 'XGBoost':
    model = xgb.XGBClassifier(**{k: v for k, v in best_params.items() if k != 'classifier'})
elif best_params['classifier'] == 'LogisticRegression':
    model = LogisticRegression(**{k: v for k, v in best_params.items() if k != 'classifier'})
elif best_params['classifier'] == 'NaiveBayes':
    model = GaussianNB()
elif best_params['classifier'] == 'SVM':
    model = SVC(**{k: v for k, v in best_params.items() if k != 'classifier'})
elif best_params['classifier'] == 'kNN':
    model = KNeighborsClassifier(**{k: v for k, v in best_params.items() if k != 'classifier'})

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print('The accuracy of the and F1-score of the best model in the test data ')
print(f"Test Set Evaluation Metrics:\nAccuracy: {accuracy}\nF1 Score: {f1}\nPrecision: {precision}\nRecall: {reca
```

```
The accuracy of the and F1-score of the best model in the test data
Test Set Evaluation Metrics:
Accuracy: 0.9807692307692307
F1 Score: 0.980602297008547
Precision: 0.9812961011591148
Recall: 0.9807692307692307
```

In [ ]:

```
def objective2(trial):

    params = {
        'n_estimators': trial.suggest_int('n_estimators', 10, 1000),
        'max_depth': trial.suggest_int('max_depth', 1, 50),
        'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy']),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 14),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 14),
        'bootstrap': trial.suggest_categorical('bootstrap', [True, False]),

    }
    model = RandomForestClassifier(**params)

    f1_scorer = make_scorer(f1_score, average='weighted')
    precision_scorer = make_scorer(precision_score)
    recall_scorer = make_scorer(recall_score)

    score = cross_val_score(model, X_train, y_train, n_jobs=-1, cv=10, scoring=f1_scorer).mean()
    return score
```

```
study2 = optuna.create_study(direction='maximize')
study2.optimize(objective2, n_trials=100)

best_trial2 = study2.best_trial
print(f"Best trial final score: {best_trial2.value}")
for key, value in best_trial2.params.items():
    print(f"{key}: {value}")
```

```
[I 2024-01-23 04:17:31,795] A new study created in memory with name: no-name-041dc1fb-a486-4389-b543-b6fe4a02ac25
[I 2024-01-23 04:17:31,914] Trial 0 finished with value: 0.9468575229369947 and parameters: {'n_estimators': 69, 'max_depth': 16, 'criterion': 'entropy', 'min_samples_split': 13, 'min_samples_leaf': 5, 'bootstrap': False}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:32,922] Trial 1 finished with value: 0.8937037041242073 and parameters: {'n_estimators': 969, 'max_depth': 1, 'criterion': 'gini', 'min_samples_split': 7, 'min_samples_leaf': 13, 'bootstrap': True}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:33,295] Trial 2 finished with value: 0.915081379664103 and parameters: {'n_estimators': 400, 'max_depth': 21, 'criterion': 'entropy', 'min_samples_split': 2, 'min_samples_leaf': 14, 'bootstrap': False}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:33,429] Trial 3 finished with value: 0.9420767735497775 and parameters: {'n_estimators': 90, 'max_depth': 49, 'criterion': 'gini', 'min_samples_split': 3, 'min_samples_leaf': 5, 'bootstrap': True}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:34,023] Trial 4 finished with value: 0.9444386624438087 and parameters: {'n_estimators': 660, 'max_depth': 50, 'criterion': 'gini', 'min_samples_split': 4, 'min_samples_leaf': 5, 'bootstrap': False}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:34,087] Trial 5 finished with value: 0.9153006353294927 and parameters: {'n_estimators': 52, 'max_depth': 29, 'criterion': 'entropy', 'min_samples_split': 7, 'min_samples_leaf': 14, 'bootstrap': False}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:34,870] Trial 6 finished with value: 0.915657123714514 8 and parameters: {'n_estimators': 693, 'max_depth': 24, 'criterion': 'gini', 'min_samples_split': 8, 'min_samples_leaf': 9, 'bootstrap': True}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:35,308] Trial 7 finished with value: 0.9250992205872816 and parameters: {'n_estimators': 481, 'max_depth': 35, 'criterion': 'entropy', 'min_samples_split': 4, 'min_samples_leaf': 11, 'bootstrap': False}. Best is trial 0 with value: 0.9468575229369947.
[I 2024-01-23 04:17:35,973] Trial 8 finished with value: 0.9395264252291453 and parameters: {'n_estimators': 697, 'max_depth': 12, 'criterion': 'entropy', 'min_samples_split': 7, 'min_samples_leaf': 8, 'bootstrap': False}. Best
```

```
8, 'max_depth': 31, 'criterion': 'gini', 'min_samples_split': 6, 'min_samples_leaf': 2, 'bootstrap': False}. Best
is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:05,383] Trial 91 finished with value: 0.9783553974512491 and parameters: {'n_estimators': 61
5, 'max_depth': 39, 'criterion': 'entropy', 'min_samples_split': 5, 'min_samples_leaf': 1, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:05,977] Trial 92 finished with value: 0.9783553974512491 and parameters: {'n_estimators': 62
8, 'max_depth': 33, 'criterion': 'entropy', 'min_samples_split': 5, 'min_samples_leaf': 1, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:06,575] Trial 93 finished with value: 0.9759284399720268 and parameters: {'n_estimators': 61
4, 'max_depth': 34, 'criterion': 'entropy', 'min_samples_split': 5, 'min_samples_leaf': 1, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:07,072] Trial 94 finished with value: 0.9710801316135094 and parameters: {'n_estimators': 54
2, 'max_depth': 39, 'criterion': 'entropy', 'min_samples_split': 5, 'min_samples_leaf': 2, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:07,727] Trial 95 finished with value: 0.9759284399720268 and parameters: {'n_estimators': 68
8, 'max_depth': 37, 'criterion': 'entropy', 'min_samples_split': 4, 'min_samples_leaf': 1, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:08,298] Trial 96 finished with value: 0.9759284399720268 and parameters: {'n_estimators': 63
4, 'max_depth': 43, 'criterion': 'entropy', 'min_samples_split': 5, 'min_samples_leaf': 1, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:08,818] Trial 97 finished with value: 0.9759856744042613 and parameters: {'n_estimators': 56
2, 'max_depth': 33, 'criterion': 'entropy', 'min_samples_split': 4, 'min_samples_leaf': 2, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:09,244] Trial 98 finished with value: 0.9759284399720268 and parameters: {'n_estimators': 45
5, 'max_depth': 30, 'criterion': 'entropy', 'min_samples_split': 5, 'min_samples_leaf': 1, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
[I 2024-01-23 04:18:09,866] Trial 99 finished with value: 0.9685385178934836 and parameters: {'n_estimators': 66
8, 'max_depth': 40, 'criterion': 'entropy', 'min_samples_split': 6, 'min_samples_leaf': 2, 'bootstrap': False}. B
est is trial 82 with value: 0.9831326497607644.
```

```
Best trial final score: 0.9831326497607644
n_estimators: 209
max_depth: 27
criterion: entropy
min_samples_split: 6
min_samples_leaf: 1
bootstrap: False
```

In [ ]:
```
best_classifier2 = best_trial2.params
best_classifier2
```

Out[ ]:
```
{'n_estimators': 209,
 'max_depth': 27,
 'criterion': 'entropy',
 'min_samples_split': 6,
 'min_samples_leaf': 1,
 'bootstrap': False}
```

In [ ]:
```
model2 = RandomForestClassifier(**best_classifier2)
model2
```

Out[ ]:
```
RandomForestClassifier

RandomForestClassifier(bootstrap=False, criterion='entropy', max_depth=27,
                       min_samples_split=6, n_estimators=209)
```

In [ ]:
```
model2.fit(X_train, y_train)

y_pred2 = model2.predict(X_test)
accuracy2 = accuracy_score(y_test, y_pred2)
f1_2 = f1_score(y_test, y_pred2, average='weighted')
precision2 = precision_score(y_test, y_pred2, average='weighted')
recall2 = recall_score(y_test, y_pred2, average='weighted')
```

```
print('The F1-Score of a better RF model in the test data is \n', f1_2)
print()
print(f"Test Set Evaluation Metrics:\nAccuracy: {accuracy2}\nF1 Score: {f1_2}\nPrecision: {precision2}\nRecall: {
```

```
The F1-Score of a better RF model in the test data is
 0.9904222748776574

Test Set Evaluation Metrics:
Accuracy: 0.9903846153846154
F1 Score: 0.9904222748776574
Precision: 0.9906674208144797
Recall: 0.9903846153846154
```

Problem #2

In [ ]:
```
import pandas as pd
import lazypredict

from lazypredict.Supervised import LazyRegressor
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

In [ ]:
```
from ucimlrepo import fetch_ucirepo

student_performance = fetch_ucirepo(id=320)

X_import = student_performance.data.features
y_import = student_performance.data.targets
X_import
```

Out[ ]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	higher	internet	romantic	famrel	free
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	yes	no	no	4	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	yes	yes	no	5	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	yes	yes	no	4	
3	GP	F	15	U	GT3	T	4	2	health	services	...	yes	yes	yes	3	
4	GP	F	16	U	GT3	T	3	3	other	other	...	yes	no	no	4	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
644	MS	F	19	R	GT3	T	2	3	services	other	...	yes	yes	no	5	
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	yes	yes	no	4	
646	MS	F	18	U	GT3	T	1	1	other	other	...	yes	no	no	1	
647	MS	M	17	U	LE3	T	3	1	services	services	...	yes	yes	no	2	
648	MS	M	18	R	LE3	T	3	2	services	other	...	yes	yes	no	4	

649 rows × 30 columns

In [ ]:

```
X = pd.concat([X_import, y_import.iloc[:, :-1]], axis=1)
X
```

Out[ ]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	romantic	famrel	freetime	goout	Dalc
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	no	4	3	4	1
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	no	5	3	3	1
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	no	4	3	2	2
3	GP	F	15	U	GT3	T	4	2	health	services	...	yes	3	2	2	1
4	GP	F	16	U	GT3	T	3	3	other	other	...	no	4	3	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	F	19	R	GT3	T	2	3	services	other	...	no	5	4	2	1
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	no	4	3	4	1
646	MS	F	18	U	GT3	T	1	1	other	other	...	no	1	1	1	1
647	MS	M	17	U	LE3	T	3	1	services	services	...	no	2	4	5	3
648	MS	M	18	R	LE3	T	3	2	services	other	...	no	4	4	1	3

649 rows × 32 columns

In [ ]:

```
y = y_import.iloc[:, -1]
y
```

Out[ ]:
```
0      11
1      11
2      12
3      14
4      13
       ..
644    10
645    16
646     9
647    10
648    11
Name: G3, Length: 649, dtype: int64
```

In [ ]:
```
le = LabelEncoder()
X = pd.DataFrame(X)
X = X.apply(lambda col: le.fit_transform(col.astype(str)), axis=0, result_type='expand')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

rgsr = LazyRegressor(verbose=0,ignore_warnings=True, custom_metric=None)
models, predictions = rgsr.fit(X_train, X_test, y_train, y_test)
```

```
 98%|█████████▊| 41/42 [00:09<00:00,  4.96it/s]
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000204 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 179
[LightGBM] [Info] Number of data points in the train set: 519, number of used features: 32
[LightGBM] [Info] Start training from score 11.793834
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
100%|██████████| 42/42 [00:09<00:00,  4.33it/s]
```

In [ ]:
```
print(models)
```

```
                            Adjusted R-Squared  R-Squared  RMSE  Time Taken
Model
GradientBoostingRegressor                 0.76       0.82  1.33        0.33
RandomForestRegressor                     0.74       0.80  1.38        0.65
```

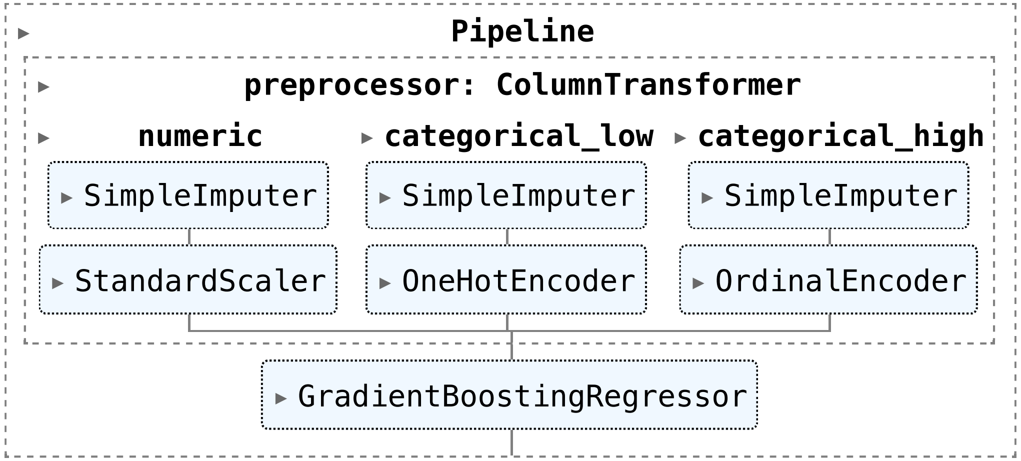
XGBRegressor	0.72	0.79	1.43	0.29
ExtraTreesRegressor	0.71	0.78	1.46	0.95
BaggingRegressor	0.71	0.78	1.47	0.08
HistGradientBoostingRegressor	0.69	0.77	1.50	1.43
LGBMRegressor	0.68	0.76	1.54	0.26
AdaBoostRegressor	0.59	0.69	1.73	0.21
DecisionTreeRegressor	0.28	0.46	2.29	0.02
ExtraTreeRegressor	0.24	0.43	2.37	0.09
NuSVR	0.02	0.26	2.68	0.09
SVR	-0.01	0.24	2.72	0.09
KNeighborsRegressor	-0.07	0.20	2.80	0.13
TweedieRegressor	-0.10	0.17	2.84	0.31
HuberRegressor	-0.11	0.16	2.86	0.09
ElasticNetCV	-0.12	0.16	2.86	0.37
LassoCV	-0.13	0.15	2.88	0.24
LassoLarsCV	-0.13	0.15	2.88	0.15
LarsCV	-0.13	0.15	2.88	0.19
LinearSVR	-0.13	0.15	2.88	0.07
BayesianRidge	-0.13	0.15	2.88	0.02
LassoLarsIC	-0.14	0.15	2.89	0.06
PoissonRegressor	-0.14	0.14	2.89	0.34
RidgeCV	-0.16	0.13	2.92	0.07
SGDRegressor	-0.16	0.13	2.92	0.04
Ridge	-0.16	0.12	2.92	0.03
Lars	-0.16	0.12	2.92	0.04
LinearRegression	-0.16	0.12	2.92	0.04
TransformedTargetRegressor	-0.16	0.12	2.92	0.05
ElasticNet	-0.18	0.11	2.95	0.03
OrthogonalMatchingPursuitCV	-0.27	0.05	3.05	0.04
OrthogonalMatchingPursuit	-0.28	0.04	3.06	0.02
Lasso	-0.29	0.03	3.07	0.03
LassoLars	-0.29	0.03	3.07	0.03
DummyRegressor	-0.37	-0.03	3.17	0.03
MLPRegressor	-0.56	-0.18	3.39	1.94
PassiveAggressiveRegressor	-0.98	-0.49	3.81	0.02

```
RANSACRegressor                           -2.42      -1.57  5.01        0.46
KernelRidge                              -20.01     -14.79 12.41        0.06
GaussianProcessRegressor                 -21.01     -15.55 12.70        0.10
```

In [ ]:

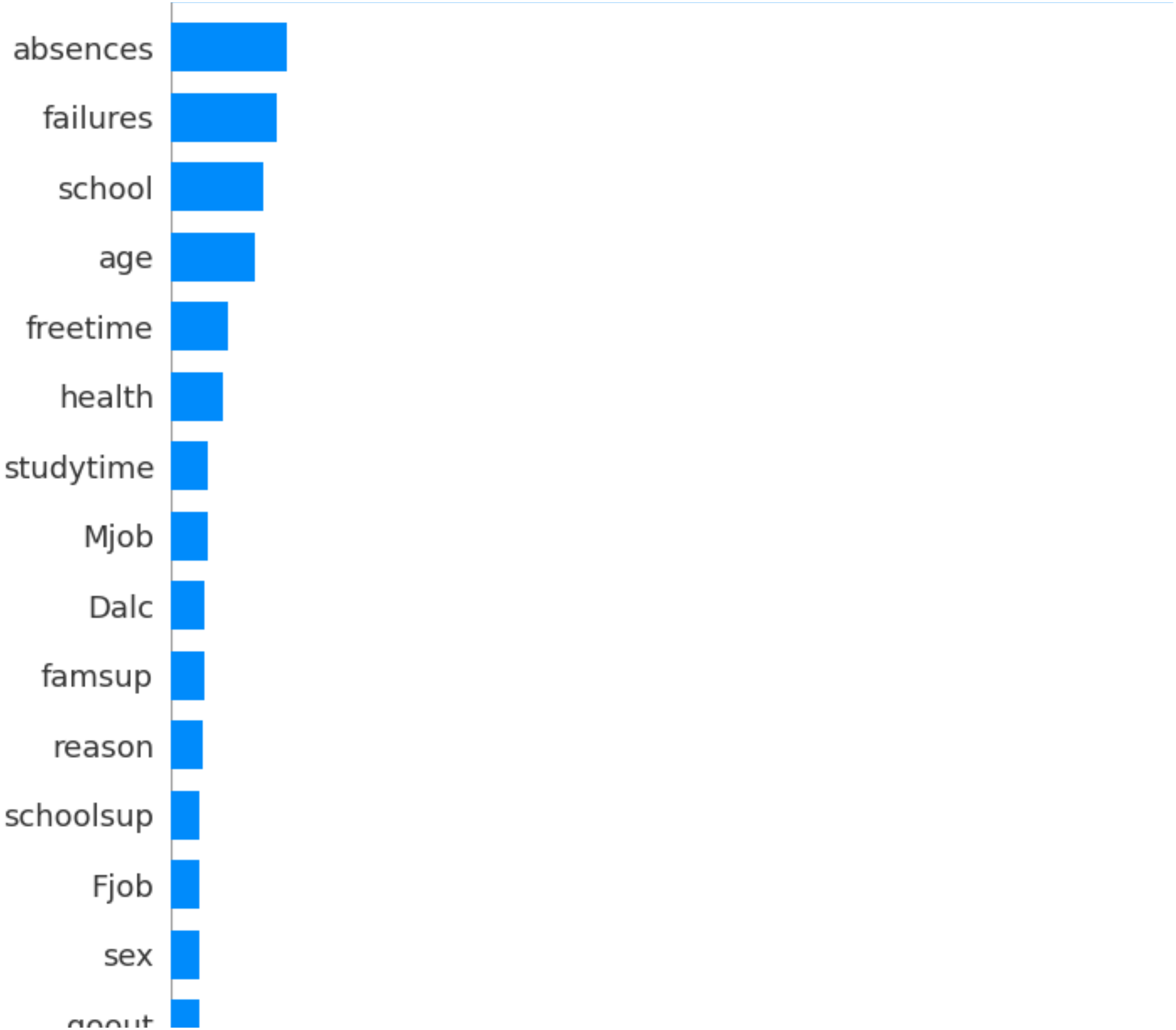
```python
best_model_name = models.sort_values('R-Squared', ascending=False).index[0]

feature_names = X.columns
X_df = pd.DataFrame(X_test, columns=feature_names)
best_model = rgsr.models[best_model_name]
best_model
```

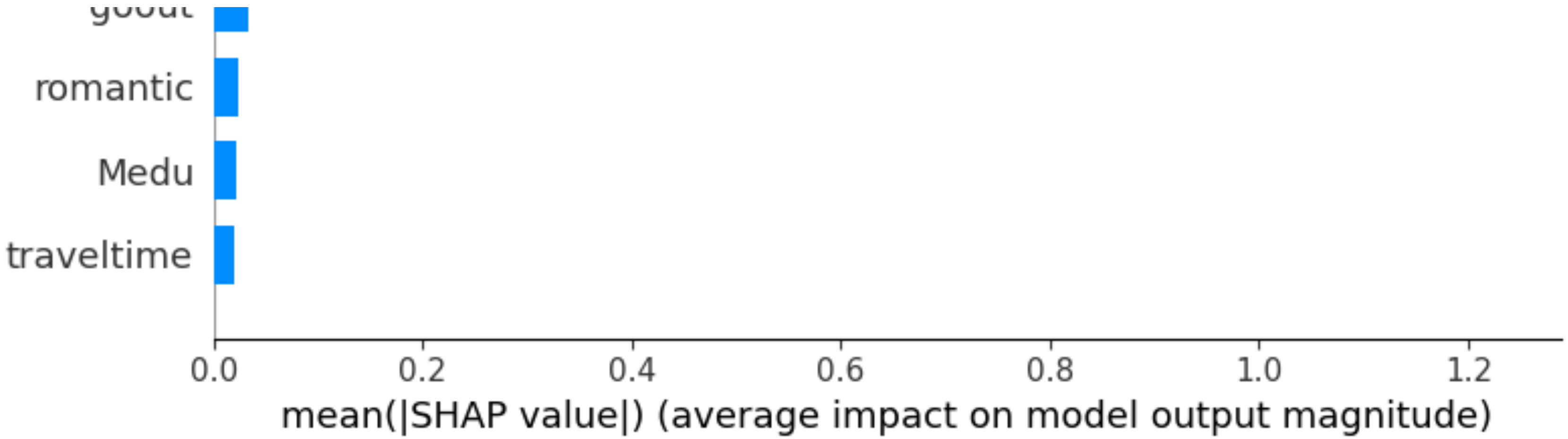
Out[ ]:



In [ ]:

```python
import shap
explainer = shap.TreeExplainer(best_model.named_steps['regressor'])
shap_values = explainer(X_df)
shap.summary_plot(shap_values, X_test, plot_type="bar")
```


absences
failures
school
age
freetime
health
studytime
Mjob
Dalc
famsup
reason
schoolsup
Fjob
sex



Summary

- According to the plot the most influencial features are G1 and G2 which are expected because these are grades from the previous periods.
- This only means that these past grades are really a high predictor of the G3 grade.
- Another is the absences. This suggest that absences directly impact the performance of the G3 of the students.
- Other factors like traveltime, Medu, romantic, and goout, are not very indicative of the performance of the students.