

Description of the notebook

- Covers the different models used for the project
- We tried the ensemble models and the sampling algorithm to see how the different models work on the imbalanced dataset.
- Incorporated evaluation metric and feature importance graph for the dataset across different years.

Imports

```
import pandas as pd
from google.colab import files
import io
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from xgboost import plot_importance
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
from sklearn.metrics import roc_auc_score, roc_curve, auc
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.metrics import average_precision_score # good for heavily imbalanced data
from sklearn.model_selection import GridSearchCV
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from imblearn.ensemble import RUSBoostClassifier

#from keras.utils import to_categorical

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 500)
```

Running the model on 2020 dataset

```
year = "20"
```

```
encoded_df_20 = pd.read_csv('/content/2020_1_updated.csv')
#df_20.to_csv("data_20.csv", index=False)
```

```
encoded_df_20.head()
```

	imputed_neighpovgroup	imputed_povertygroup	imputed_povgroup	imputed_pov	generalhealth	pcp	didntgetcare	toldhighbp	toldprescrip
0	3	1	1	1	3	1	2	2	
1	3	1	1	1	4	1	2	2	
2	1	4	3	2	2	1	2	2	
3	2	1	1	1	2	1	2	2	
4	2	4	3	2	3	1	2	1	

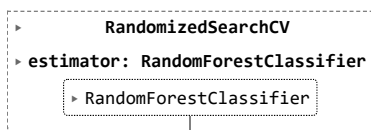
Performing both sampling and splitting the dataset for training and testing

```
X_temp_20 = encoded_df_20.loc[:, encoded_df_20.columns != "diabetes"]
X_20 = X_temp_20.loc[:, X_temp_20.columns != 'cid']
y_20 = encoded_df_20["diabetes"]
y_20 = y_20.map({2:0, 1:1})
X_train_20, X_test_20, y_train_20, y_test_20 = train_test_split(X_20, y_20, test_size=0.33, random_state=42)
oversample = RandomOverSampler(sampling_strategy=0.5)
X_re_20, y_re_20 = oversample.fit_resample(X_train_20, y_train_20)
undersample = RandomUnderSampler(sampling_strategy=0.8)
X_re_20, y_re_20 = undersample.fit_resample(X_re_20, y_re_20)
X_train_over_20, y_train_over_20 = X_re_20, y_re_20
```

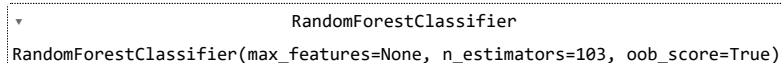
▼ Model after performing sampling

```
model = RandomForestClassifier()
m_params = { "n_estimators" : np.linspace(2, 500, 500, dtype = "int"),
             "max_depth": [5, 20, 30, None],
             "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
             "max_features": ["sqrt", "log2", 10, 20, None],
             "oob_score": [True],
             "bootstrap": [True]
           }
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_20 = RandomizedSearchCV(model,
                                     param_distributions = m_params,
                                     n_iter = 20,
                                     scoring = scoreFunction,
                                     refit = "recall",
                                     return_train_score = True,
                                     random_state = 42,
                                     cv = 5)

#trains and optimizes the model
random_search_20.fit(X_train_over_20, y_train_over_20)
```



```
random_search_20.best_estimator_
```



```
#recover the best model
model_20 = random_search_20.best_estimator_

# model = RandomForestClassifier(n_estimators = 120,
#                               max_depth=15,
#                               criterion = 'entropy',
#                               random_state = 0)
model_20.fit(X_train_over_20, y_train_over_20)
preds_over_20 = model_20.predict(X_test_20)
print("Accuracy:", accuracy_score(y_test_20, preds_over_20))
```

```
Accuracy: 0.8535371489512975
```

```
prob_over_20 = model_20.predict_proba(X_test_20)
```

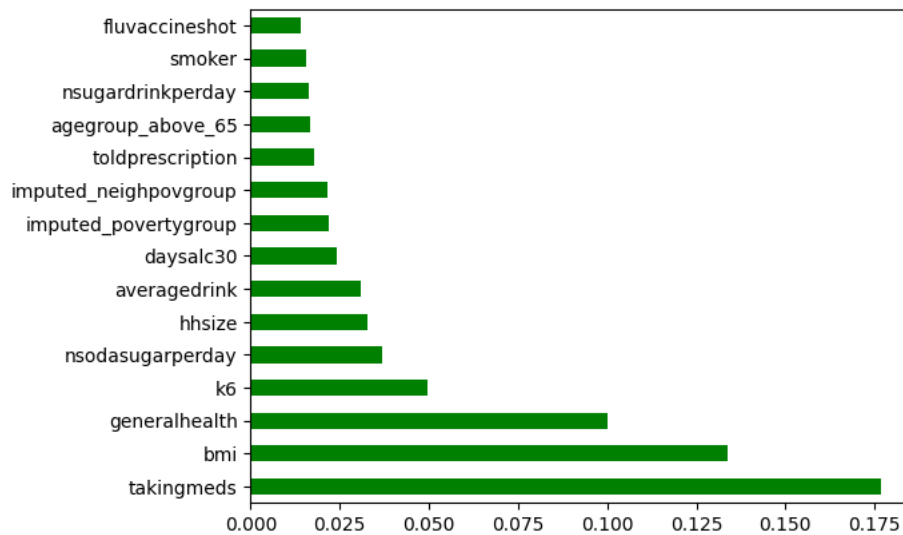
```
y_test_20.value_counts()
```

```
0    2470
1     343
```

Name: diabetes, dtype: int64

```
global_importances_20 = pd.Series(model_20.feature_importances_, index=X_train_over_20.columns)
global_importances_20.sort_values(ascending=True, inplace=True)
global_importances_20[::-1][:15].plot.barh(color='green')
```

<Axes: >



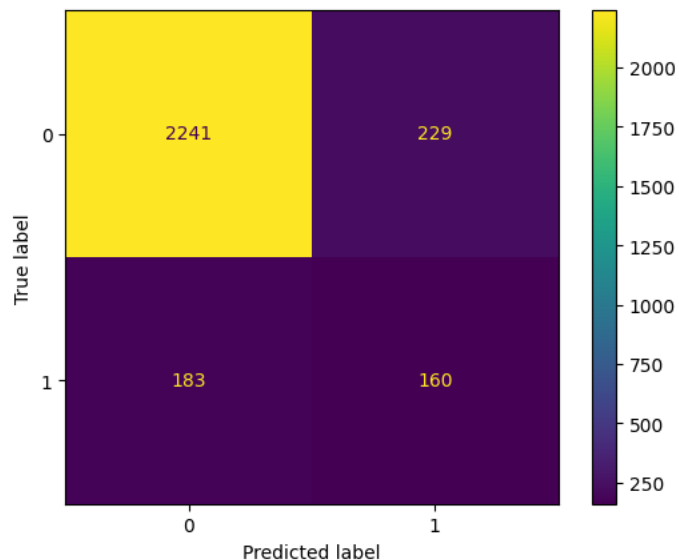
```
viz_20 = pd.DataFrame(global_importances_20[::-1][:15]).transpose()
```

viz_20

	takingmeds	bmi	generalhealth	k6	nsodasugarperday	hysize	averagedrink	daysalc30	imputed_povertygroup	imputed_neighp
0	0.176749	0.133738	0.099971	0.04974	0.036711	0.032743	0.030841	0.024303	0.02194	(

```
cm = confusion_matrix(y_test_20, preds_over_20, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a3d48a9b0>



```

tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
print("f1 score", f1_score(y_test_20, preds_over_20))
print("average precision score", average_precision_score(y_test_20, preds_over_20))
print("sensitivity", (tp / (tp+fn)))

```

```

precision 0.41131105398457585
recall 0.46647230320699706
f1 score 0.4371584699453552
average precision score 0.2569203160020043
sensitivity 0.46647230320699706

```

```

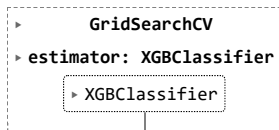
param_grid = {
    "max_depth": [3, 4, 5, 7],
    "learning_rate": [0.1, 0.01, 0.05],
    "gamma": [0, 0.25, 1],
    "reg_lambda": [0, 1, 10],
    "scale_pos_weight": [1, 3, 5],
    "subsample": [0.8],
    "colsample_bytree": [0.5],
}

```

```

xgb_cl = xgb.XGBClassifier()
grid_cv = GridSearchCV(xgb_cl, param_grid, n_jobs=-1, cv=3, scoring="precision")
grid_cv.fit(X_train_over_20, y_train_over_20)

```



```
grid_cv.best_params_
```

```

{'colsample_bytree': 0.5,
 'gamma': 0,
 'learning_rate': 0.1,
 'max_depth': 7,
 'reg_lambda': 0,
 'scale_pos_weight': 1,
 'subsample': 0.8}

```

```

# Fit
xgb_cl = xgb.XGBClassifier(**grid_cv.best_params_)
xgb_cl.fit(X_train_over_20, y_train_over_20)

# Predict
preds_over_xgb = xgb_cl.predict(X_test_20)
print("accuracy score", accuracy_score(y_test_20, preds_over_xgb))

```

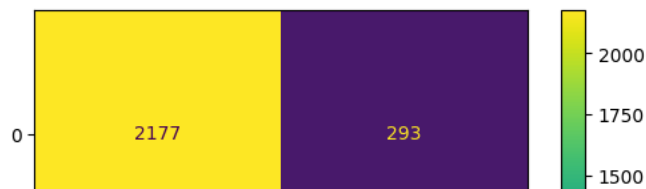
```
accuracy score 0.8418059011731248
```

```

cm = confusion_matrix(y_test_20, preds_over_xgb, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()

```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a3d3cddb0>
```



```
tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
print("f1 score", f1_score(y_test_20, preds_over_xgb))
print("average precision score", average_precision_score(y_test_20, preds_over_xgb))
print("sensitivity", (tp / (tp+fn)))
```

```
precision 0.39462809917355374
recall 0.5568513119533528
f1 score 0.46191051995163246
average precision score 0.2737840130094288
sensitivity 0.5568513119533528
```

✓ Ensemble model before sampling

```
brf = BalancedRandomForestClassifier()
m_params = {"n_estimators": np.linspace(2, 500, 500, dtype = "int"),
            "max_depth": [5, 20, 30, None],
            "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
            "max_features": ["sqrt", "log2", 10, 20, None],
            "oob_score": [True],
            "bootstrap": [True],
            "replacement": [True, False],
            "criterion": ["gini", "entropy"],
            "sampling_strategy": ["majority", "not minority", "not majority", "all", "auto"]}
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_brfc_20 = RandomizedSearchCV(brf,
                                           param_distributions = m_params,
                                           n_iter = 20,
                                           scoring = scoreFunction,
                                           refit = "recall",
                                           return_train_score = True,
                                           random_state = 42,
                                           cv = 5)
random_search_brfc_20.fit(X_train_20, y_train_20)
brf_20 = random_search_brfc_20.best_estimator_
```

```

warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(

```

```

brf_20.fit(X_train_20, y_train_20)
y_pred_brfc_20 = brf_20.predict(X_test_20)
cm = confusion_matrix(y_test_20, y_pred_brfc_20, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot()

```

1

[illegible]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----

[illegible]

[illegible]

```

/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated: The function

```


[illegible]

```

/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(

```

```

tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
print("f1 score", f1_score(y_test_20, preds_over_20))
print("average precision score", average_precision_score(y_test_20, preds_over_20))
print("accuracy", accuracy_score(y_test_20, preds_over_20))
print("sensitivity", (tp / (tp+fn)))

```

```

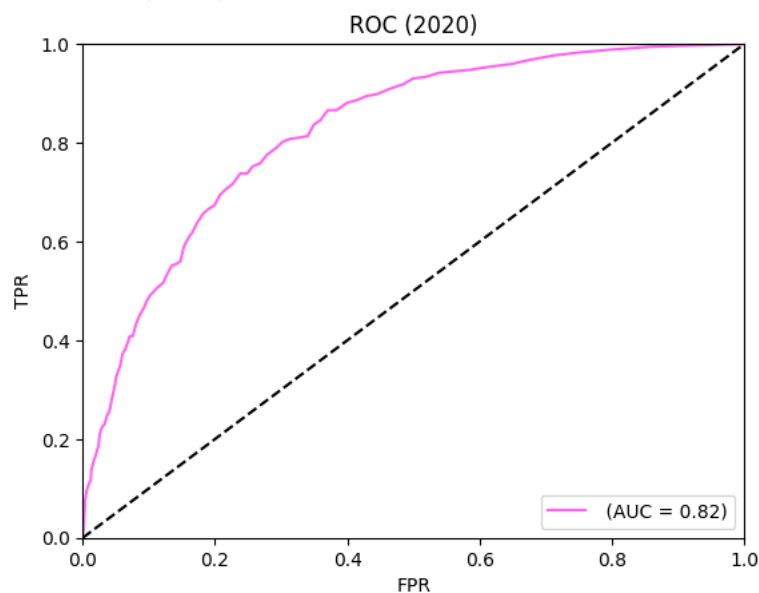
precision 0.2927835051546392
recall 0.8279883381924198
f1 score 0.4371584699453552
average precision score 0.2569203160020043
accuracy 0.8535371489512975
sensitivity 0.8279883381924198

<sklearn.metrics.plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a3d23+d90>

```

```
# def plotAUC(truth, pred, lab):
fpr, tpr, thresholds = roc_curve(y_test_20, prob_over_20[:,1])
roc_auc = auc(fpr, tpr)
c = (np.random.rand(), np.random.rand(), np.random.rand())
plt.plot(fpr, tpr, color=c, label= ' (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC (2020)')
plt.legend(loc="lower right")
# plotAUC(y_test_over_20, prob_over_20[:,1], "All")
```

<matplotlib.legend.Legend at 0x7e7a3d17dba0>



```
rub_clf_20 = RUBBoostClassifier(random_state=0,
                                n_estimators=25,
                                sampling_strategy='majority',
                                replacement=True,
                                learning_rate=1)
rub_clf_20.fit(X_train_20, y_train_20)
y_preds_rub_20 = rub_clf_20.predict(X_test_20)
cm = confusion_matrix(y_test_20, y_preds_rub_20, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()

f1_score(y_test_20, y_preds_rub_20)
print("accuracy", accuracy_score(y_test_20, y_preds_rub_20))
print("sensitivity", (tp/ (tp+fn)))
```


accuracy 0.7511553501599716
sensitivity 0.8279883381924198

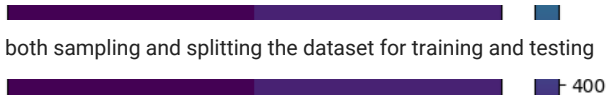


✓ Running the model on 2019 dataset

year = "19"

1000

```
encoded_df_19 = pd.read_csv('/content/2019_updated.csv')
#df_19.to_csv("data.csv", index=False)
```



Performing both sampling and splitting the dataset for training and testing

```
X_temp_19 = encoded_df_19.loc[:, encoded_df_19.columns != "diabetes"]
X_19 = X_temp_19.loc[:, X_temp_19.columns != 'cid']
y_19 = encoded_df_19["diabetes"]
y_19 = y_19.map({2:0, 1:1})
X_train_19, X_test_19, y_train_19, y_test_19 = train_test_split(X_19, y_19, test_size=0.33, random_state=42)
oversample = RandomOverSampler(sampling_strategy=0.5)
X_re_19, y_re_19 = oversample.fit_resample(X_train_19, y_train_19)
undersample = RandomUnderSampler(sampling_strategy=0.8)
X_re_19, y_re_19 = undersample.fit_resample(X_re_19, y_re_19)
X_train_over_19, y_train_over_19 = X_re_19, y_re_19
```

✓ Models after performing sampling

```
model = RandomForestClassifier()
m_params = { "n_estimators": np.linspace(2, 500, 500, dtype = "int"),
             "max_depth": [5, 20, 30, None],
             "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
             "max_features": ["sqrt", "log2", 10, 20, None],
             "oob_score": [True],
             "bootstrap": [True]
           }
```

```
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_19 = RandomizedSearchCV(model,
                                     param_distributions = m_params,
                                     n_iter = 20,
                                     scoring = scoreFunction,
                                     refit = "recall",
                                     return_train_score = True,
                                     random_state = 42,
                                     cv = 5)
```

```
#trains and optimizes the model
random_search_19.fit(X_train_over_19, y_train_over_19)
```

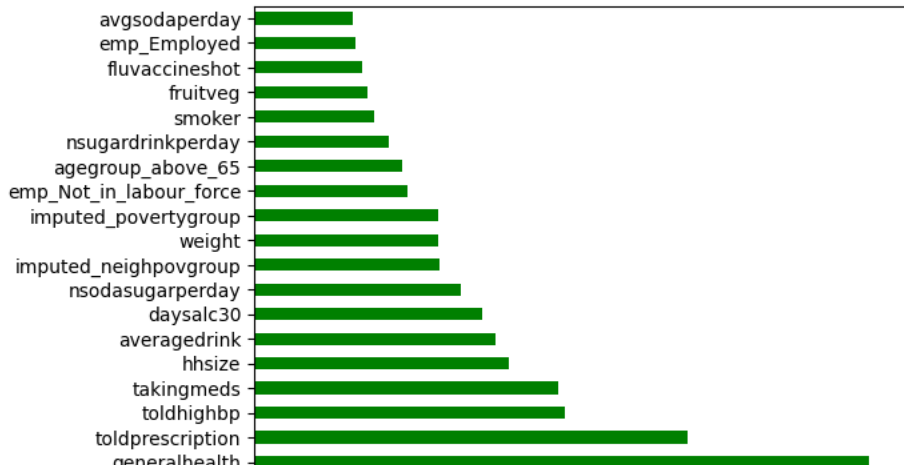
```
#recover the best model
model = random_search_19.best_estimator_
```

```
model.fit(X_train_over_19, y_train_over_19)
preds_over = model.predict(X_test_19)
print("Accuracy:", accuracy_score(y_test_19, preds_over))
```

Accuracy: 0.8368005738880918

```
global_importances_19 = pd.Series(model.feature_importances_, index=X_train_over_19.columns)
global_importances_19.sort_values(ascending=True, inplace=True)
global_importances_19[::-1][:20].plot.barh(color='green')
```

<Axes: >



```
viz_19 = pd.DataFrame(global_importances_19[:, :-1][:20]).transpose()
```

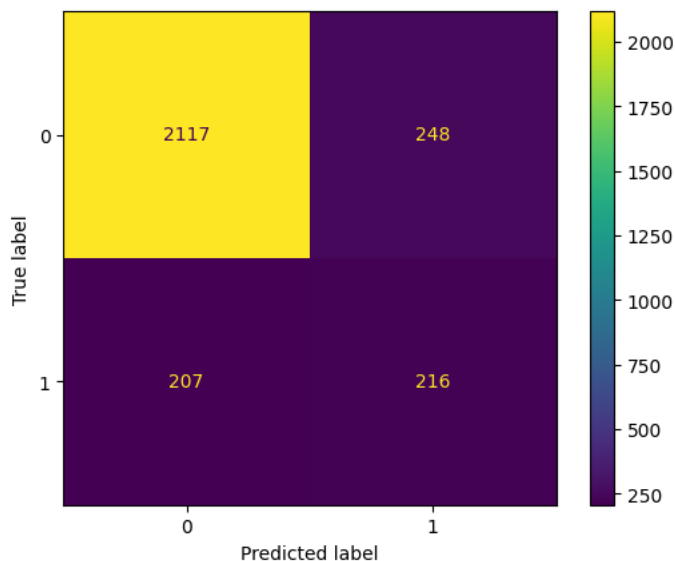
```
0.00 0.02 0.04 0.06 0.08
```

```
viz_19
```

	bmi	generalhealth	toldprescription	toldhighbp	takingmeds	hhsize	averagedrink	daysalc30	nsodasugarperday	imputed_neighp
0	0.090708	0.088517	0.062412	0.044757	0.043778	0.036576	0.034698	0.032867	0.029733	0

```
cm = confusion_matrix(y_test_19, preds_over, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a3d0268c0>
```

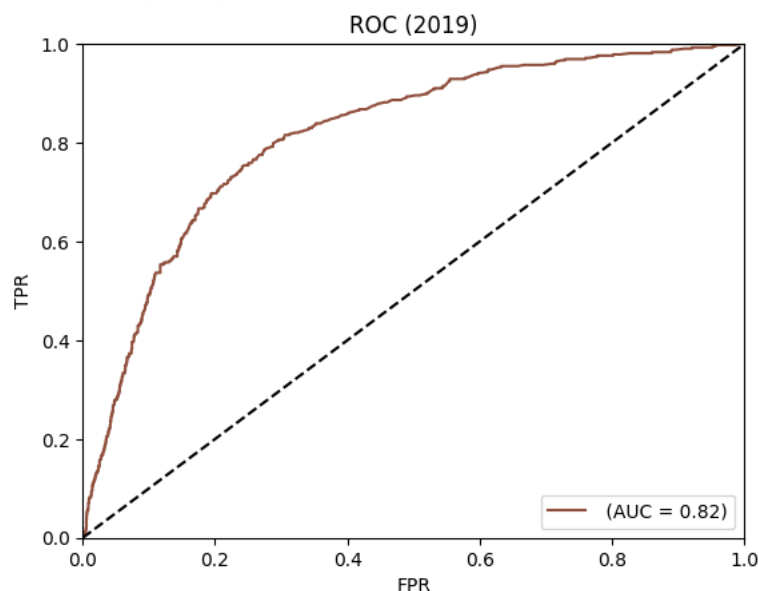


```
tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
print("f1 score", f1_score(y_test_19, preds_over))
```

```
precision 0.46551724137931033
recall 0.5106382978723404
f1 score 0.4870349492671927
```

```
# def plotAUC(truth, pred, lab):
prob_over_19 = model.predict_proba(X_test_19)
fpr, tpr, thresholds = roc_curve(y_test_19, prob_over_19[:,1])
roc_auc = auc(fpr, tpr)
c = (np.random.rand(), np.random.rand(), np.random.rand())
plt.plot(fpr, tpr, color=c, label= ' (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC (2019)')
plt.legend(loc="lower right")
# plotAUC(y_test_over_20, prob_over_20[:,1], "All")
```

<matplotlib.legend.Legend at 0x7e7a3cf06860>

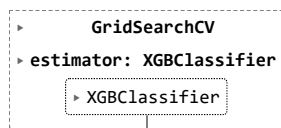


```
y_test_19.value_counts()
```

```
0    2365
1     423
Name: diabetes, dtype: int64
```

```
param_grid = {
    "max_depth": [3, 4, 5, 7],
    "learning_rate": [0.1, 0.01, 0.05],
    "gamma": [0, 0.25, 1],
    "reg_lambda": [0, 1, 10],
    "scale_pos_weight": [1, 3, 5],
    "subsample": [0.8],
    "colsample_bytree": [0.5],
}
```

```
xgb_cl_19 = xgb.XGBClassifier()
grid_cv_19 = GridSearchCV(xgb_cl_19, param_grid, n_jobs=-1, cv=3, scoring="precision")
grid_cv_19.fit(X_train_over_19, y_train_over_19)
```



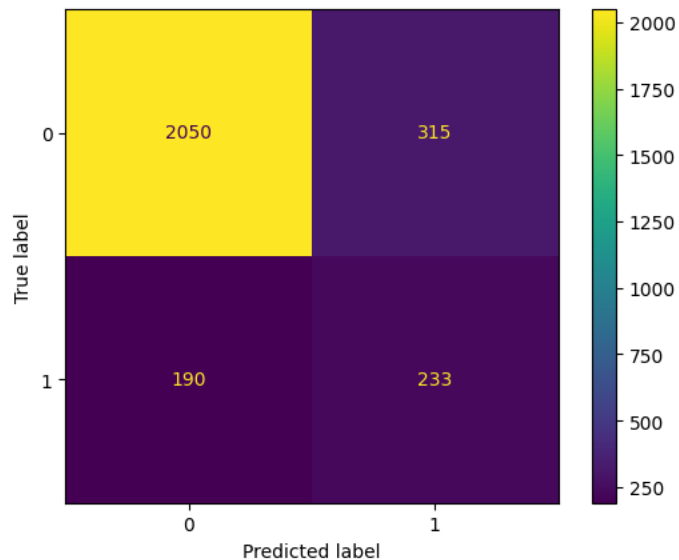
```
# Fit
xgb_cl_19 = xgb.XGBClassifier(**grid_cv_19.best_params_)
xgb_cl_19.fit(X_train_over_19, y_train_over_19)

# Predict
preds_over_xgb_19 = xgb_cl_19.predict(X_test_19)
accuracy_score(y_test_19, preds_over_xgb_19)

0.8188665710186513

cm = confusion_matrix(y_test_19, preds_over_xgb_19, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a3ce57490>



```
tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
#print(tn, fp, fn, tp)
print("f1 score", f1_score(y_test_19, preds_over_xgb_19))
print("average precision", average_precision_score(y_test_19, preds_over_xgb_19))

precision 0.4251824817518248
recall 0.5508274231678487
f1 score 0.4799176107106076
average precision 0.3023513817033423
```

✓ Ensemble models before sampling

```
y_train_19.value_counts()

0    4846
1     814
Name: diabetes, dtype: int64
```

```

brf = BalancedRandomForestClassifier()
m_params = {"n_estimators": np.linspace(2, 500, 500, dtype = "int"),
            "max_depth": [5, 20, 30, None],
            "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
            "max_features": ["sqrt", "log2", 10, 20, None],
            "oob_score": [True],
            "bootstrap": [True],
            "replacement": [True, False],
            "criterion": ["gini", "entropy"],
            "sampling_strategy": ["majority", "not minority", "not majority", "all", "auto"]}

scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_brfc_19 = RandomizedSearchCV(brf,
                                           param_distributions = m_params,
                                           n_iter = 20,
                                           scoring = scoreFunction,
                                           refit = "recall",
                                           return_train_score = True,
                                           random_state = 42,
                                           cv = 5)

random_search_brfc_19.fit(X_train_19, y_train_19)
brf_19 = random_search_brfc_19.best_estimator_

warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(

```

```
brf_19.fit(X_train_19, y_train_19)
y_pred_brfc_19 = brf_19.predict(X_test_19)
cm = confusion_matrix(y_test_19, y_pred_brfc_19, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot()
```

▲

[illegible]

<https://colab.research.google.com/drive/1Cci3VSNw3pVWcthnOJJ81Cli2WimmUDF#scrollTo=o2jTpmbhWol&printMode=true>

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Warning: FutureWarning: category=FutureWarning)

Running on 2018 dataset

```
year = "18"
```

```
encoded_df_18 = pd.read_csv('/content/2018_updated.csv')
#df_18.to_csv("data_18.csv", index=False)
```

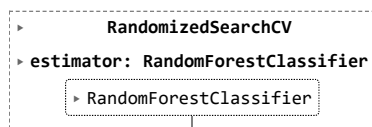
Performing both sampling and splitting the dataset for training and testing

```
X_temp_18 = encoded_df_18.loc[:, encoded_df_18.columns != "diabetes"]
X_18 = X_temp_18.loc[:, X_temp_18.columns != 'cid']
y_18 = encoded_df_18["diabetes"]
y_18 = y_18.map({2:0, 1:1})
X_train_18, X_test_18, y_train_18, y_test_18 = train_test_split(X_18, y_18, test_size=0.33, random_state=42)
oversample = RandomOverSampler(sampling_strategy=0.5)
X_re_18, y_re_18 = oversample.fit_resample(X_train_18, y_train_18)
undersample = RandomUnderSampler(sampling_strategy=0.8)
X_re_18, y_re_18 = undersample.fit_resample(X_re_18, y_re_18)
X_train_over_18, y_train_over_18 = X_re_18, y_re_18
```

Model on sampled data

```
model = RandomForestClassifier()
m_params = { "n_estimators": np.linspace(2, 500, 500, dtype = "int"),
             "max_depth": [5, 20, 30, None],
             "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
             "max_features": ["sqrt", "log2", 10, 20, None],
             "oob_score": [True],
             "bootstrap": [True]
           }
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_18 = RandomizedSearchCV(model,
                                     param_distributions = m_params,
                                     n_iter = 20,
                                     scoring = scoreFunction,
                                     refit = "recall",
                                     return_train_score = True,
                                     random_state = 42,
                                     cv = 5)

#trains and optimizes the model
random_search_18.fit(X_train_over_18, y_train_over_18)
```



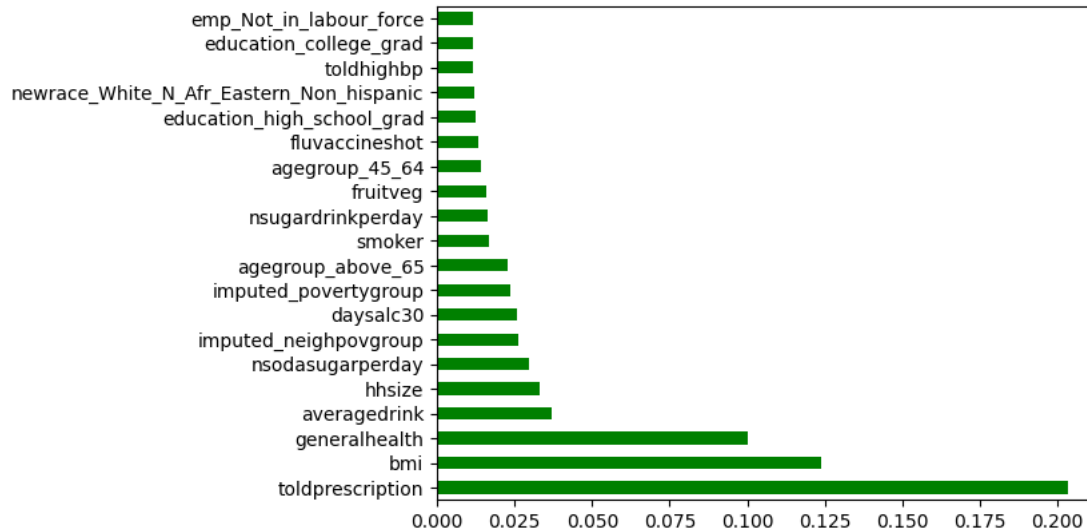

```
#recover the best model
model_18 = random_search_18.best_estimator_

# model = RandomForestClassifier(n_estimators = 120,
#                               max_depth=15,
#                               criterion = 'entropy',
#                               random_state = 0)
model_18.fit(X_train_over_18, y_train_over_18)
preds_over = model_18.predict(X_test_18)
print("Accuracy:", accuracy_score(y_test_18, preds_over))
```

Accuracy: 0.806502031884964

```
global_importances_18 = pd.Series(model_18.feature_importances_, index=X_train_over_18.columns)
global_importances_18.sort_values(ascending=True, inplace=True)
global_importances_18[::-1][:20].plot.barh(color='green')
```

<Axes: >



```
viz_18 = pd.DataFrame(global_importances_18[::-1][:20]).transpose()
viz_18
```

	toldprescription	bmi	generalhealth	averagedrink	hhsz	nsodasugarperday	imputed_neighpovgroup	daysalc30	imputed_poverty
0	0.203368	0.123897	0.100361	0.036775	0.033243	0.029421	0.026067	0.025755	0.000000

```
cm = confusion_matrix(y_test_18, preds_over, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a392c7760>
```



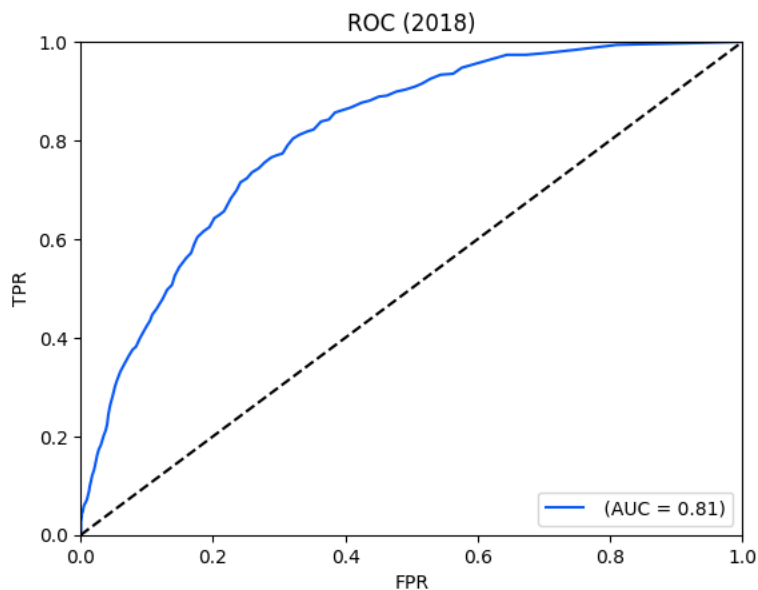
```
tn, fp, fn, tp = cm.ravel()
precision = tp / (tp + fp)
print(precision)
recall = tp / (tp + fn)
print(recall)
print(tn, fp, fn, tp)
print(f1_score(y_test_18, preds_over))
```

```
0.40372670807453415
0.5252525252525253
2320 384 235 260
0.45654082528533807
```



```
# def plotAUC(truth, pred, lab):
prob_over_18 = model_18.predict_proba(X_test_18)
fpr, tpr, thresholds = roc_curve(y_test_18, prob_over_18[:,1])
roc_auc = auc(fpr, tpr)
c = (np.random.rand(), np.random.rand(), np.random.rand())
plt.plot(fpr, tpr, color=c, label= ' (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC (2018)')
plt.legend(loc="lower right")
# plotAUC(y_test_over_20, prob_over_20[:,1], "All")
```

```
<matplotlib.legend.Legend at 0x7e7a381930d0>
```



```
param_grid = {
    "max_depth": [3, 4, 5, 7],
    "learning_rate": [0.1, 0.01, 0.05],
    "gamma": [0, 0.25, 1],
    "reg_lambda": [0, 1, 10],
    "scale_pos_weight": [1, 3, 5],
    "subsample": [0.8],
    "colsample_bytree": [0.5],
}

xgb_cl_18 = xgb.XGBClassifier()
grid_cv_18 = GridSearchCV(xgb_cl_18, param_grid, n_jobs=-1, cv=3, scoring="precision")
grid_cv_18.fit(X_train_over_18, y_train_over_18)
```

```

GridSearchCV
estimator: XGBClassifier

```

```

# Fit
xgb_cl_18 = xgb.XGBClassifier(**grid_cv_18.best_params_)
xgb_cl_18.fit(X_train_over_18, y_train_over_18)

```

```

# Predict
preds_over_xgb_18 = xgb_cl_18.predict(X_test_18)
accuracy_score(y_test_18, preds_over_xgb_18)

```

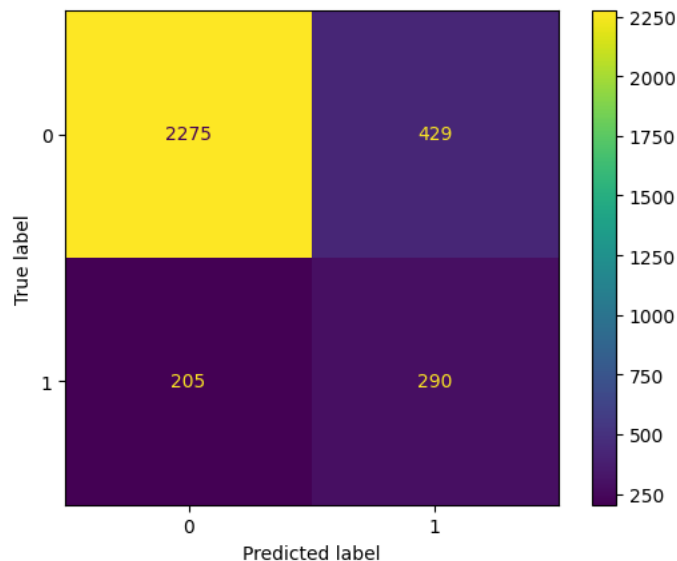
```
0.8018130665833073
```

```

cm = confusion_matrix(y_test_18, preds_over_xgb_18, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()

```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a381dae60>
```



```

tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
#print(tn, fp, fn, tp)
print("f1 score", f1_score(y_test_18, preds_over_xgb_18))
print("average precision", average_precision_score(y_test_18, preds_over_xgb_18))

```

```

precision 0.40333796940194716
recall 0.5858585858585859
f1 score 0.47775947281713343
average precision 0.3003815381662075

```

▼ Ensemble model

```

brf = BalancedRandomForestClassifier()
m_params = {"n_estimators": np.linspace(2, 500, 500, dtype = "int"),
            "max_depth": [5, 20, 30, None],
            "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
            "max_features": ["sqrt", "log2", 10, 20, None],
            "oob_score": [True],
            "bootstrap": [True],
            "replacement": [True, False],
            "criterion": ["gini", "entropy"],
            "sampling_strategy": ["majority", "not minority", "not majority", "all", "auto"]}
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_brf_18 = RandomizedSearchCV(brf,
                                          param_distributions = m_params,
                                          n_iter = 20,
                                          scoring = scoreFunction,
                                          refit = "recall",
                                          return_train_score = True,
                                          random_state = 42,
                                          cv = 5)
random_search_brf_18.fit(X_train_18, y_train_18)
brf_18 = random_search_brf_18.best_estimator_

warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(

```

```
brf_18.fit(X_train_18, y_train_18)
y_pred_brfc_18 = brf_18.predict(X_test_18)
cm = confusion_matrix(y_test_18, y_pred_brfc_18, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot()
print("f1 score", f1_score(y_test_18, y_pred_brfc_18))
```


[illegible]

olab research google.com/drive/1Cci3VSNw3pVWcthpQ_Ul81Cii2WimmlIDE#scrollTo=_o2iTpmbbWol&printMode=true 4

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

olab.research.google.com/drive/1Cci3VSNw3pVWcthnOJJ81Cij2WimmUDF#scrollTo= o2iTpmbhWol&printMode=true 5

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
rub_clf_18 = RUBBoostClassifier(random_state=0,
                                n_estimators=25,
                                sampling_strategy='majority',
                                replacement=True,
                                learning_rate=1)
rub_clf_18.fit(X_train_18, y_train_18)
y_preds_rub_18 = rub_clf_18.predict(X_test_18)
cm = confusion_matrix(y_test_18, y_preds_rub_18, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()

f1_score(y_test_18, y_preds_rub_18)
```

```
0.48716266697827958, category=futurewarning)
```

Running the model on 2017 data

```
year = "17"

encoded_df_17 = pd.read_csv('/content/2017_updated.csv')
#df_17.to_csv("data_17.csv", index=False)

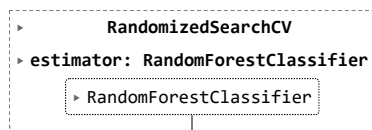
Performing both sampling and splitting the dataset for training and testing

X_temp_17 = encoded_df_17.loc[:, encoded_df_17.columns != "diabetes"]
X_17 = X_temp_17.loc[:, X_temp_17.columns != 'cid']
y_17 = encoded_df_17["diabetes"]
y_17 = y_17.map({2:0, 1:1})
X_train_17, X_test_17, y_train_17, y_test_17 = train_test_split(X_17, y_17, test_size=0.33, random_state=42)
oversample = RandomOverSampler(sampling_strategy=0.5)
X_re_17, y_re_17 = oversample.fit_resample(X_train_17, y_train_17)
undersample = RandomUnderSampler(sampling_strategy=0.8)
X_re_17, y_re_17 = undersample.fit_resample(X_re_17, y_re_17)
X_train_over_17, y_train_over_17 = X_re_17, y_re_17
```

Model on sampled data

```
model = RandomForestClassifier()
m_params = { "n_estimators" : np.linspace(2, 500, 500, dtype = "int"),
             "max_depth": [5, 20, 30, None],
             "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
             "max_features": ["sqrt", "log2", 10, 20, None],
             "oob_score": [True],
             "bootstrap": [True]
           }
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_17 = RandomizedSearchCV(model,
                                     param_distributions = m_params,
                                     n_iter = 20,
                                     scoring = scoreFunction,
                                     refit = "recall",
                                     return_train_score = True,
                                     random_state = 42,
                                     cv = 5)

#trains and optimizes the model
random_search_17.fit(X_train_over_17, y_train_over_17)
```



```
#recover the best model
model_17 = random_search_17.best_estimator_

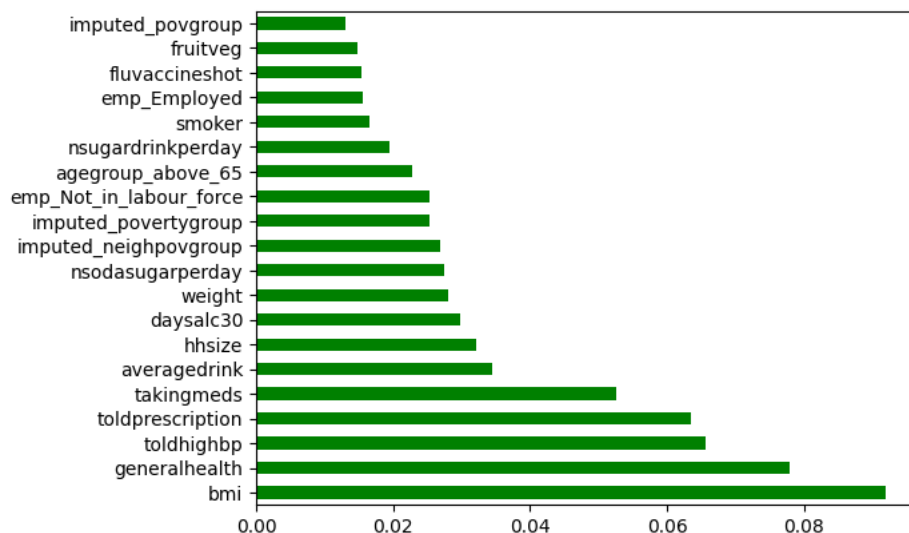
# model = RandomForestClassifier(n_estimators = 120,
#                               max_depth=15,
#                               criterion = 'entropy',
#                               random_state = 0)
model_17.fit(X_train_over_17, y_train_over_17)
preds_over_17 = model_17.predict(X_test_17)
print("Accuracy:", accuracy_score(y_test_17, preds_over_17))
```

```
Accuracy: 0.8376603065373788
```

```
global_importances = pd.Series(model_17.feature_importances_, index=X_train_over_17.columns)
```

```
global_importances.sort_values(ascending=True, inplace=True)
global_importances[::-1][:20].plot.barh(color='green')
```

<Axes: >

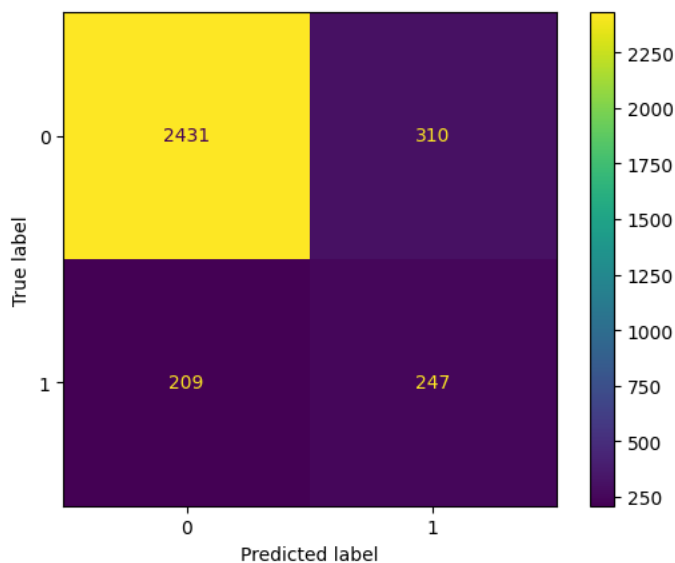


```
viz_17 = pd.DataFrame(global_importances[::-1][:20]).transpose()
viz_17
```

	bmi	generalhealth	toldhighbp	toldprescription	takingmeds	averagedrink	hhsz	daysalc30	weight	nsodasugarperday	imput
0	0.092006	0.077956	0.065589	0.063505	0.052604	0.034368	0.032209	0.029747	0.028112	0.027518	

```
cm = confusion_matrix(y_test_17, preds_over_17, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a2b5e08b0>



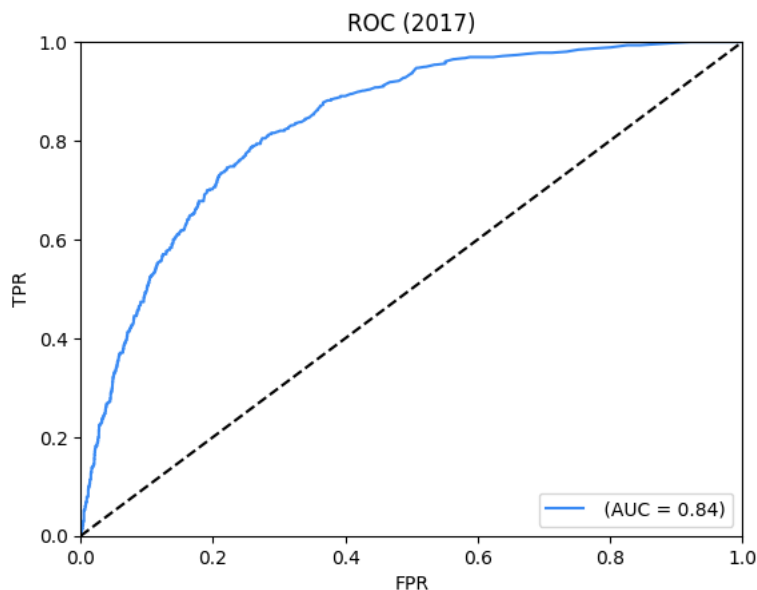
```
tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print(precision)
recall = tp / (tp+fn)
print(recall)
print(tn, fp, fn, tp)
print(f1_score(y_test_17, preds_over_17))
```

```
0.44344703770197486
0.5416666666666666
2431 310 209 247
0.48766041461006904
```



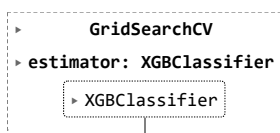
```
# def plotAUC(truth, pred, lab):
prob_over_17 = model_17.predict_proba(X_test_17)
fpr, tpr, thresholds = roc_curve(y_test_17, prob_over_17[:,1])
roc_auc = auc(fpr, tpr)
c = (np.random.rand(), np.random.rand(), np.random.rand())
plt.plot(fpr, tpr, color=c, label= ' (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC (2017)')
plt.legend(loc="lower right")
# plotAUC(y_test_over_20, prob_over_20[:,1], "All")
```

<matplotlib.legend.Legend at 0x7e7a2b347eb0>



```
param_grid = {
    "max_depth": [3, 4, 5, 7],
    "learning_rate": [0.1, 0.01, 0.05],
    "gamma": [0, 0.25, 1],
    "reg_lambda": [0, 1, 10],
    "scale_pos_weight": [1, 3, 5],
    "subsample": [0.8],
    "colsample_bytree": [0.5],
}

xgb_cl_17 = xgb.XGBClassifier()
grid_cv_17 = GridSearchCV(xgb_cl_17, param_grid, n_jobs=-1, cv=3, scoring="precision")
grid_cv_17.fit(X_train_over_17, y_train_over_17)
```



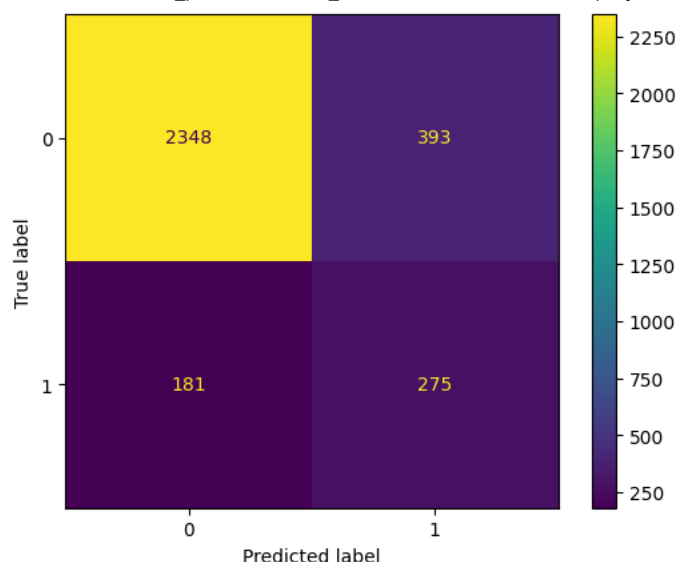
```
# Fit
xgb_cl_17 = xgb.XGBClassifier(**grid_cv_17.best_params_)
xgb_cl_17.fit(X_train_over_17, y_train_over_17)

# Predict
preds_over_xgb_17 = xgb_cl_17.predict(X_test_17)
accuracy_score(y_test_17, preds_over_xgb_17)
```

0.8204566781357523

```
cm = confusion_matrix(y_test_17, preds_over_xgb_17, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
disp.plot()
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e7a2b32b190>



```
tn, fp, fn, tp = cm.ravel()
precision = tp / (tp+fp)
print("precision", precision)
recall = tp / (tp+fn)
print("recall", recall)
#print(tn, fp, fn, tp)
print("f1 score", f1_score(y_test_17, preds_over_xgb_17))
print("average precision", average_precision_score(y_test_17, preds_over_xgb_17))
```

```
precision 0.4116766467065868
recall 0.6030701754385965
f1 score 0.4893238434163701
average precision 0.304885484656849
```

✓ Ensemble model

```
brf = BalancedRandomForestClassifier()
m_params = {"n_estimators": np.linspace(2, 500, 500, dtype = "int"),
            "max_depth": [5, 20, 30, None],
            "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
            "max_features": ["sqrt", "log2", 10, 20, None],
            "oob_score": [True],
            "bootstrap": [True],
            "replacement": [True, False],
            "criterion": ["gini", "entropy"],
            "sampling_strategy": ["majority", "not minority", "not majority", "all", "auto"]}
scoreFunction = {"recall": "recall", "precision": "precision"}
random_search_brf_17 = RandomizedSearchCV(brf,
                                          param_distributions = m_params,
                                          n_iter = 20,
                                          scoring = scoreFunction,
                                          refit = "recall",
                                          return_train_score = True,
                                          random_state = 42,
                                          cv = 5)
random_search_brf_17.fit(X_train_17, y_train_17)
brf_17 = random_search_brf_17.best_estimator_
```

```

warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(

```

```

brf_17.fit(X_train_17, y_train_17)
y_pred_brfc_17 = brf_17.predict(X_test_17)
cm = confusion_matrix(y_test_17, y_pred_brfc_17, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot()
print("f1 score", f1_score(y_test_17, y_pred_brfc_17))

```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated: The function

```

[illegible]

[illegible]

[illegible]

[illegible]

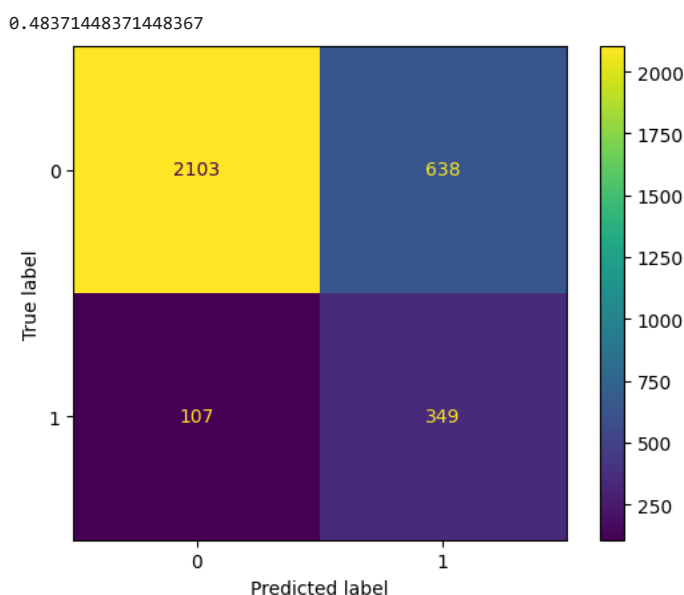
[illegible]

```
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:86: FutureWarning: Function delayed is deprecated; The function
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/parallel.py:114: UserWarning: `sklearn.utils.parallel.delayed` should be used v
warnings.warn(
f1 score 0.472703917790623
```



```
rub_clf_17 = RUBoostClassifier(random_state=0,
                              n_estimators=25,
                              sampling_strategy='majority',
                              replacement=True,
                              learning_rate=1)
rub_clf_17.fit(X_train_17, y_train_17)
y_preds_rub_17 = rub_clf_17.predict(X_test_17)
cm = confusion_matrix(y_test_17, y_preds_rub_17, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
disp.plot()

f1_score(y_test_17, y_preds_rub_17)
```



✓ line chart visualization for feature importance

```
viz_col = ["year"]
def get_all_viz_cols(df):
    for col in df.columns:
        if col not in viz_col:
            viz_col.append(col)

get_all_viz_cols(viz_20)
get_all_viz_cols(viz_19)
get_all_viz_cols(viz_18)
get_all_viz_cols(viz_17)

print(viz_col)

['year', 'takingmeds', 'bmi', 'generalhealth', 'k6', 'nsodasugarperday', 'hhszise', 'averagedrink', 'daysalc30', 'imputed_povertygroup',
viz_df = pd.DataFrame(columns=viz_col)
```

viz_df

year	takingmeds	bmi	generalhealth	k6	nsodasugarperday	hhsiz	averagedrink	daysalc30	imputed_povertygroup	imputed_neighpovgrou
------	------------	-----	---------------	----	------------------	-------	--------------	-----------	----------------------	----------------------

```
def construct_df_for_viz(df, year):
    val_dict={}
    val_dict['year'] = year
    for col in viz_df.columns[1:]:
        if col in df.columns:
            val_dict[col]=df[col][0]
        else:
            val_dict[col]=0

    return val_dict
rows = []
rows.append(construct_df_for_viz(viz_20, "2020"))
rows.append(construct_df_for_viz(viz_19, "2019"))
rows.append(construct_df_for_viz(viz_18, "2018"))
rows.append(construct_df_for_viz(viz_17, "2017"))
```

```
df= pd.DataFrame.from_dict(rows, orient='columns').set_index('year')
```

df

	takingmeds	bmi	generalhealth	k6	nsodasugarperday	hhsiz	averagedrink	daysalc30	imputed_povertygroup	imputed_nei
2020	0.176749	0.133738	0.099971	0.04974	0.036711	0.032743	0.030841	0.024303	0.021940	
2019	0.043778	0.090708	0.088517	0.00000	0.029733	0.036576	0.034698	0.032867	0.026555	
2018	0.000000	0.123897	0.100361	0.00000	0.029421	0.033243	0.036775	0.025755	0.023406	
2017	0.052604	0.092006	0.077956	0.00000	0.027518	0.032209	0.034368	0.029747	0.025380	

```
# ax = df.T.plot(figsize=(len(viz_col[1:]), 7))
# ax.set_ylabel('Absolute Power (log)', fontsize=12)
# ax.set_xlabel('Frequencies', fontsize=12)
# #ax.set_xticks(ticks=viz_col[1:], labels=viz_col[1:], rotation=90, fontsize=12)
# #ax.set_xticklabels(rotation=90)
# plt.show()
```

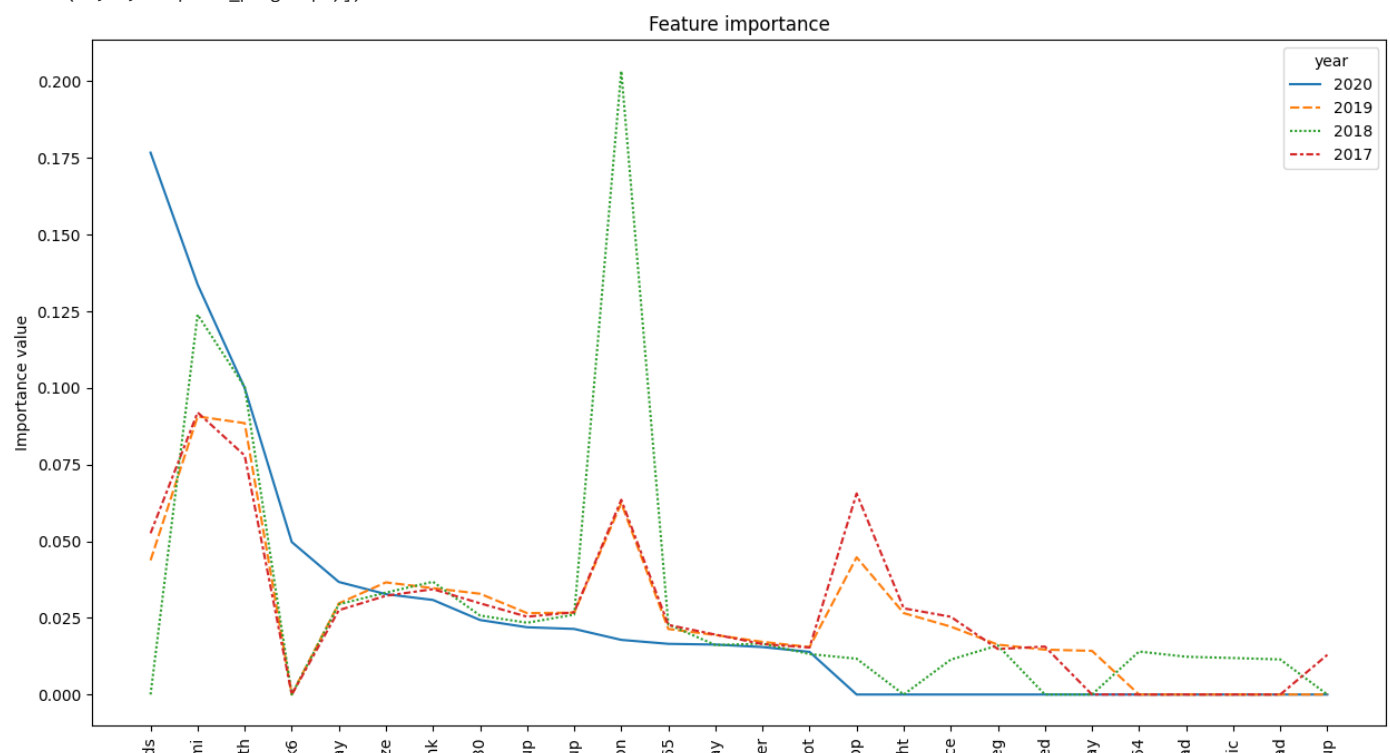
```
# the pivot of your data
df_ = df.T
```

```
# plot your data
plt.figure(figsize=(15,8))
sns.lineplot(data=df_)
plt.title('Feature importance')
plt.xlabel('Top features')
plt.ylabel('Importance value')
plt.xticks(rotation=90)
```

```

([0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
10,
11,
12,
13,
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25],
[Text(0, 0, 'takingmeds'),
 Text(1, 0, 'bmi'),
 Text(2, 0, 'generalhealth'),
 Text(3, 0, 'k6'),
 Text(4, 0, 'nsodasugarperday'),
 Text(5, 0, 'hhsize'),
 Text(6, 0, 'averagedrink'),
 Text(7, 0, 'daysalc30'),
 Text(8, 0, 'imputed_povertygroup'),
 Text(9, 0, 'imputed_neighpovgroup'),
 Text(10, 0, 'toldprescription'),
 Text(11, 0, 'agegroup_above_65'),
 Text(12, 0, 'nsugardrinkperday'),
 Text(13, 0, 'smoker'),
 Text(14, 0, 'fluvaccineshot'),
 Text(15, 0, 'toldhighbp'),
 Text(16, 0, 'weight'),
 Text(17, 0, 'emp_Not_in_labour_force'),
 Text(18, 0, 'fruitveg'),
 Text(19, 0, 'emp_Employed'),
 Text(20, 0, 'avgsodaperday'),
 Text(21, 0, 'agegroup_45_64'),
 Text(22, 0, 'education_high_school_grad'),
 Text(23, 0, 'newrace_White_N_Afr_Eastern_Non_hispanic'),
 Text(24, 0, 'education_college_grad'),
 Text(25, 0, 'imputed_povgroup')])

```



▼ Data imbalance plot

```
encoded_df_20['diabetes'] = encoded_df_20['diabetes'].map({1: "Have diabetes", 2: "Don't have diabetes"})
encoded_df_19['diabetes'] = encoded_df_19['diabetes'].map({1: "Have diabetes", 2: "Don't have diabetes"})
encoded_df_18['diabetes'] = encoded_df_18['diabetes'].map({1: "Have diabetes", 2: "Don't have diabetes"})
encoded_df_17['diabetes'] = encoded_df_17['diabetes'].map({1: "Have diabetes", 2: "Don't have diabetes"})

target_class_20=encoded_df_20['diabetes'].value_counts().to_dict()
target_class_19=encoded_df_19['diabetes'].value_counts().to_dict()
target_class_18=encoded_df_18['diabetes'].value_counts().to_dict()
target_class_17=encoded_df_17['diabetes'].value_counts().to_dict()

# df_18['diabetes18'].value_counts()

target_class_20['year']=2020
target_class_19['year']=2019
target_class_18['year']=2018
target_class_17['year']=2017

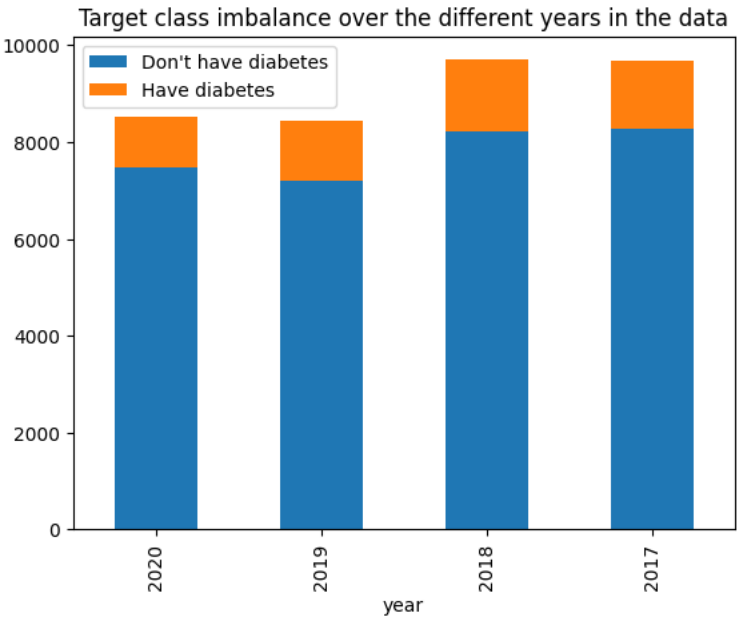
diab = pd.DataFrame([target_class_20, target_class_19, target_class_18, target_class_17])

diab
```

	Don't have diabetes	Have diabetes	year
0	7479	1045	2020
1	7211	1237	2019
2	8210	1483	2018
3	8277	1409	2017

```
diab.plot(x="year", kind='bar', stacked=True,
         title='Target class imbalance over the different years in the data')

<Axes: title={'center': 'Target class imbalance over the different years in the data'}, xlabel='year'>
```



2020 - misc

```

# X_temp_20 = encoded_df_20.loc[:, encoded_df_20.columns != 'diabetes20']
# X_20 = X_temp_20.loc[:, X_temp_20.columns != 'cid']
# y_20 = encoded_df_20['diabetes20']
# X_train_20, X_test_20, y_train_20, y_test_20 = train_test_split(X_20, y_20, test_size=0.33, random_state=42)
# y_train_20 = y_train_20.map({2:0, 1:1})
# y_test_20 = y_test_20.map({2:0, 1:1})

# brf = BalancedRandomForestClassifier()
# m_params = {"n_estimators": np.linspace(2, 500, 500, dtype = "int"),
#             "max_depth": [5, 20, 30, None],
#             "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
#             "max_features": ["sqrt", "log2", 10, 20, None],
#             "oob_score": [True],
#             "bootstrap": [True],
#             "replacement": [True, False],
#             "criterion": ["gini", "entropy"],
#             "sampling_strategy": ["majority", "not minority", "not majority", "all", "auto"]
#             }
# scoreFunction = {"recall": "recall", "precision": "precision"}
# random_search_brf_20 = RandomizedSearchCV(brf,
#                                           param_distributions = m_params,
#                                           n_iter = 20,
#                                           scoring = scoreFunction,
#                                           refit = "recall",
#                                           return_train_score = True,
#                                           random_state = 42,
#                                           cv = 5)
# random_search_brf_20.fit(X_train_20, y_train_20)
# brf_20 = random_search_brf_20.best_estimator_

# brf_20.fit(X_train_20, y_train_20)
# y_pred_brfc = brf_20.predict(X_test_20)
# cm = confusion_matrix(y_test_20, y_pred_brfc, labels=[0, 1])
# disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
# disp.plot()
# f1_score(y_test_20, y_pred_brfc)

# tn, fp, fn, tp = cm.ravel()
# precision = tp / (tp+fp)
# print(precision)
# recall = tp / (tp+fn)
# print(recall)
# print(tn, fp, fn, tp)

# average_precision_score(y_test_20, y_pred_brfc)

# from imblearn.ensemble import RUSBoostClassifier
# rub_clf = RUSBoostClassifier(random_state=0,
#                              n_estimators=25,
#                              sampling_strategy='majority',
#                              replacement=True,
#                              learning_rate=1)
# rub_clf.fit(X_train_20, y_train_20)
# y_preds_rub_20 = rub_clf.predict(X_test_20)
# cm = confusion_matrix(y_test_20, y_preds_rub_20, labels=[0,1])
# disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1])
# disp.plot()

# f1_score(y_test_20, y_preds_rub_20)

# tn, fp, fn, tp = cm.ravel()
# precision = tp / (tp+fp)
# print(precision)
# recall = tp / (tp+fn)
# print(recall)
# print(tn, fp, fn, tp)

```

```

# average_precision_score(y_test_20, y_preds_rub_20)

# xgb_cl = xgb.XGBClassifier()

# # Fit
# xgb_cl.fit(X_train_20, y_train_20)

# # Predict
# preds_over_xgb = xgb_cl.predict(X_test_20)
# accuracy_score(y_test_20, preds_over_xgb)

# cm = confusion_matrix(y_test_20, preds_over_xgb, labels=[0,1])
# disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
# disp.plot()

# f1_score(y_test_20, preds_over_xgb)

# average_precision_score(y_test_20, preds_over_xgb)

# tn, fp, fn, tp = cm.ravel()
# precision = tp/ (tp+fp)
# print(precision)
# recall = tp/(tp+fn)
# print(recall)
# print(tn, fp, fn, tp)
# #print(f1_score(y_test_20, preds_over_20))

# model = RandomForestClassifier()
# m_params = { "n_estimators": np.linspace(2, 500, 500, dtype = "int"),
#             "max_depth": [5, 20, 30, None],
#             "min_samples_split": np.linspace(2, 50, 50, dtype = "int"),
#             "max_features": ["sqrt", "log2",10, 20, None],
#             "oob_score": [True],
#             "bootstrap": [True]
#             }
# scoreFunction = {"recall": "recall", "precision": "precision"}
# random_search = RandomizedSearchCV(model,
#                                     param_distributions = m_params,
#                                     n_iter = 20,
#                                     scoring = scoreFunction,
#                                     refit = "recall",
#                                     return_train_score = True,
#                                     random_state = 42,

```