In [1]:
```python
import pandas as pd
from nltk.tokenize import word_tokenize
import nltk
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MultiLabelBinarizer
from readability import Readability
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /Users/conradhalle/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Out[1]: True

In [2]:
```python
raw_data = pd.read_csv("answerList_data.csv")
# add type-token ratio
explanations = [str(e) for e in list(raw_data["Answer.explanation"])]
scores = [len(list(set(word_tokenize(e)))) / len(word_tokenize(e)) for e in
raw_data.loc[:, "explanation.TTR"] = scores
# add Flesch-Kincaid readability score
scores = [Readability(e*10).flesch_kincaid().score for e in explanations]
raw_data.loc[:, "explanation.FK"] = scores
# add explanation size (length of explanation in words)
raw_data["explanation.size"] = raw_data["Answer.explanation"].apply(lambda a
```

In [3]:
```python
wherelearnedtocode_mapping = {'Other self taught': 'Self-taught',
 'Other Self-study': 'Self-taught',
 'Other autodidact': 'Self-taught',
 'Other self-taught': 'Self-taught',
 'Other self study': 'Self-taught',
 'Other books': 'Books and Paper Resources',
 'Other found a book': 'Books and Paper Resources',
 'Other had a book on BASIC when I was a kid': 'Books and Paper Resources',
 'Other by myself from books': 'Self-taught',
 'Other Taught myself the basics as a kid': 'Self-taught',
 'Other Instructional books': 'Books and Paper Resources',
 'Other Self Help Books': 'Self-taught',
 'Other private institute': 'Self-taught',
 'Other private': 'Self-taught',
 'Other Self Taught': 'Self-taught',
 'Other Self Study': 'Self-taught',
 'Other Hobbyist': 'Self-taught',
 'Other as long as i can remember honestly': 'Self-taught',
 'Other i really dont remember a time when i couldnt': 'Self-taught',
 'Other Self-taught as a child': 'Self-taught',
 'Other Self taught': 'Self-taught',
 'Web': 'Web',
 'University': 'University',
 'High School': 'High School',
 'Middle school': 'Middle School',
 'Other junior high computer class': 'Middle School',
 'Other Middle school': 'Middle School',
 'Other elementary school': 'Elementary School',
 'Other Diploma': 'Diploma or Certification',
```

```
 'Other at work': 'Workplace Learning',
 'Other on the job': 'Workplace Learning',
 'Other Through work': 'Workplace Learning',
 'Other Employer': 'Workplace Learning',
 'Other Professionally': 'Workplace Learning',
 'Other Java while on the job': 'Workplace Learning',
 'Other Work': 'Workplace Learning',
 'Other Leanred more experts at work': 'Workplace Learning',
 'Other US Army': 'Workplace Learning',
 'Other On the job': 'Workplace Learning',
 'Other work': 'Workplace Learning',
 'Other Professional': 'Workplace Learning',
 'Other MOOC': 'MOOCs and Online Resources',
 'Other Training classes': 'MOOCs and Online Resources',
 'Other Private Institute': 'MOOCs and Online Resources',
 " there was no 'internet'. I had a TRS-80 for gosh sake. We learned from bo
 'Other Books': 'Books and Paper Resources',
 ' books': 'Books and Paper Resources',
 ' there was no "internet". I had a TRS-80 for gosh sake. We learned from bo
 'Other FIRST Robotics': 'Extracurricular Activities',
 'Other Summer Camp': 'Extracurricular Activities',
 'Other Hobby': 'Extracurricular Activities',
 'Other na': 'Unknown/Other',
 'Other': 'Unknown/Other',
 ' no?': 'Unknown/Other',
 'Other forever': 'Unknown/Other',
 'Other When I started programming (At age 9)': 'Unknown/Other'}

programminglanguage_mapping = {
    'Other Java while on the job': 'Java',
    'Other Instructional books': 'Other',
    '': 'None',
    'php and C++': ['PHP', 'C++'],
    '.net': '.NET',
    'Matlab': 'MATLAB',
    ' c++': 'C++',
    'Other autodidact': 'Other',
    ' Java': 'Java',
    ' no?': 'None',
    ' java': 'Java',
    'Other Professional': 'Other',
    ' PHP': 'PHP',
    'visual foxpro': 'Visual FoxPro',
    'Object Pascal': 'Object Pascal',
    ' js': 'JavaScript',
    'C++': 'C++',
    'SAS': 'SAS',
    ' C#': 'C#',
    'visual fox pro': 'Visual FoxPro',
    'Web': 'Other',
    ' C+': 'C++',
    'c# vb.net java': ['C#', 'VB.NET', 'Java'],
    ' Ruby': 'Ruby',
    ' ASP': 'ASP',
    'python': 'Python',
    'Other Books': 'Other',
```

```
    'Ruby & JavaScript': ['Ruby', 'JavaScript'],
    'BASH': 'Bash',
    ' MQL4': 'MQL4',
    ' plsql': 'PL/SQL',
    ' C': 'C',
    'Other self taught': 'Other',
    'PHP': 'PHP',
    'NONE': 'None',
    'C': 'C',
    'Other When I started programming (At age 9)': 'Other',
    'nan': 'None',
    'ASP.NET': 'ASP.NET',
    'Other Private Institute': 'Other',
    'HTML/CSS (formerly C# and Java in previous jobs)': ['HTML', 'CSS', 'C#'
    'Java Python': ['Java', 'Python'],
    ' HTML': 'HTML',
    'Other as long as i can remember honestly': 'Other',
    'Other Leanred more experts at work': 'Other',
    'matlab': 'MATLAB',
    ' Php': 'PHP',
    'Other Middle school': 'Other',
    'VisualBasic': 'Visual Basic',
    ' JAVA': 'Java',
    'Other private institute': 'Other',
    ' Visual Basic': 'Visual Basic',
    'University': 'Other',
    ' autoit': 'AutoIt',
    ' peoplesoft': 'PeopleSoft',
    'Other On the job': 'Other',
    'Other private': 'Other',
    'Other Through work': 'Other',
    'dot net and core java': ['.NET', 'Java'],
    'none': 'None',
    ' Scheme': 'Scheme',
    'JAVA': 'Java',
    'JavaScript': 'JavaScript',
    'Other na': 'Other',
    'Other books': 'Other',
    ' Elixir': 'Elixir',
    ' javascript': 'JavaScript',
    ' go': 'Go',
    'Swift': 'Swift',
    ' AHK': 'AutoHotkey',
    'Other self study': 'Other',
    'Other at work': 'Other',
    'Various': 'Other',
    'Other Diploma': 'Other',
    ' mysql': 'MySQL',
    ' scala': 'Scala',
    ' C ': 'C',
    'C/C++': ['C', 'C++'],
    'Other self-taught': 'Other',
    ' there was no "internet". I had a TRS-80 for gosh sake. We learned from
    'Python': 'Python',
    'Other Work': 'Other',
    'developing': 'Other',
```

```
    'vbs': 'VBScript',
    ' .net': '.NET',
    'Ruby': 'Ruby',
    ' vba': 'VBA',
    ' groovy': 'Groovy',
    'Other Self Taught': 'Other',
    ' CSS': 'CSS',
    'Other Taught myself the basics as a kid': 'Other',
    ' MySQL': 'MySQL',
    'Other junior high computer class': 'Other',
    'Other on the job': 'Other',
    'VB.net': 'VB.NET',
    ' VBA': 'VBA',
    'XML': 'XML',
    ' c/c++': ['C', 'C++'],
    ' VB.NET': 'VB.NET',
    'BASH and Powershell': ['Bash', 'PowerShell'],
    'Other FIRST Robotics': 'Other',
    'html': 'HTML',
    ' CSS3': 'CSS',
    ' php': 'PHP',
    'HTML': 'HTML',
    ' Pl/SQL': 'PL/SQL',
    'Other MOOC': 'Other',
    ' JS': 'JavaScript',
    'JSP': 'JSP',
    'c': 'C',
    'Java ': 'Java',
    'RPG': 'RPG',
    ' assembly': 'Assembly',
    'High School': 'Other',
    ' python': 'Python',
    ' PL/SQL': 'PL/SQL',
    'Other Employer': 'Other',
    ' Assembly': 'Assembly',
    ' vb.net': 'VB.NET',
    'SQL': 'SQL',
    'jav': 'Java',
    ' Matlab': 'MATLAB',
    'Other elementary school': 'Other',
    'Other Self-study': 'Other',
    'vb .net java': ['VB.NET', 'Java'],
    ' Ruby on Rails': 'Ruby on Rails',
    'C/C++ and some python and java': ['C', 'C++', 'Python', 'Java'],
    ' Cuda': 'CUDA',
    'Other i really dont remember a time when i couldnt': 'Other',
    ' books': 'Other',
    ' most C and ksh in past': 'C',
    'Other Self Help Books': 'Other',
    'c sharp': 'C#',
    'Ada': 'Ada',
    ' C++': 'C++',
    'JAVASCRIPT': 'JavaScript',
    'C#': 'C#',
    'Perl': 'Perl',
    ' vbscript': 'VBScript',
```

```
    'Other Hobbyist': 'Other',
    ' HTML5': 'HTML',
    'Other Self-taught as a child': 'Other',
    'Javascript': 'JavaScript',
    'Other forever': 'Other',
    'AJAX': 'AJAX',
    '6': 'Other',
    ' Perl': 'Perl',
    'perl': 'Perl',
    'php': 'PHP',
    'Other Self Study': 'Other',
    'MAGIC': 'Other',
    'Euphoria': 'Euphoria',
    'java': 'Java',
    'java c++': ['Java', 'C++'],
    ' JEE': 'JEE',
    ' R': 'R',
    'Php/Javascript': ['PHP', 'JavaScript'],
    'Other work': 'Other',
    'Other Self taught': 'Other',
    'VB.NET': 'VB.NET',
    'javascript': 'JavaScript',
    'bash': 'Bash',
    'Other Training classes': 'Other',
    'MySQL': 'MySQL',
    'c++': 'C++',
    'Other had a book on BASIC when I was a kid': 'Other',
    'Other found a book': 'Other',
    'Other US Army': 'Other',
    'Java': 'Java',
    ' SQL': 'SQL',
    'vb.net': 'VB.NET',
    'tera data': 'Teradata',
    'R': 'R',
    '.NET': '.NET',
    '5': 'Other',
    ' VHDL': 'VHDL',
    'Other Hobby': 'Other',
    'Other Professionally': 'Other',
    ' Python': 'Python',
    ' JavaScript': 'JavaScript',
    'BASIC': 'BASIC',
    ' Javascript': 'JavaScript',
    'VBA': 'VBA',
    'none at present': 'None',
    '3': 'Other',
    ' c': 'C',
    ' asp.net': 'ASP.NET',
    'Other Summer Camp': 'Other',
    'Do not currently use': 'None',
    'Other': 'Other',
    'Other by myself from books': 'Other',
    'C++ and Python': ['C++', 'Python'],
    ' a little bit of Java': 'Java',
    'c#': 'C#',
    'VB.Net': 'VB.NET',
```

```
    'SQL (Teradata/MS-SQL)': 'SQL',
    'ruby': 'Ruby'
}

mapped_data = raw_data.copy()
mapped_data["Worker.whereLearnedToCode"]=mapped_data["Worker.whereLearnedToC
mapped_data["Worker.whereLearnedToCode"]=mapped_data["Worker.whereLearnedToC

def flatten(lst):
    flat_list = []
    for item in lst:
        if isinstance(item, list):  # Check if the item is a list
            flat_list.extend(flatten(item))  # Recursively flatten the list
        else:
            flat_list.append(item)  # Add non-list items directly
    return flat_list

def map_column_values(values):
    if type(values) != list:
        values = [values]
    for value in values:
        if str(value) not in programminglanguage_mapping:
            print(f"Missing mapping for value: {value}")
    return flatten([programminglanguage_mapping.get(str(value)) for value ir

mapped_data["Worker.programmingLanguage"]=mapped_data["Worker.programmingLar
mapped_data["Worker.programmingLanguage"]=mapped_data["Worker.programmingLar

mlb = MultiLabelBinarizer()
whereLearnedToCodeVector = mlb.fit_transform(mapped_data.loc[:, "Worker.wher
whereLearnedToCodeDf = pd.DataFrame(whereLearnedToCodeVector, columns=[f"whe
mapped_data = pd.concat([mapped_data, whereLearnedToCodeDf], axis=1)

mlb = MultiLabelBinarizer()
programmingLanguageVector = mlb.fit_transform(mapped_data.loc[:, "Worker.prc
programmingLanguageDf = pd.DataFrame(programmingLanguageVector, columns=[f"p
mapped_data = pd.concat([mapped_data, programmingLanguageDf], axis=1)

mapped_data.drop(["Worker.programmingLanguage", "Worker.whereLearnedToCode"]

mapped_data.columns
```

Out[3]:
```
Index(['Answer.ID', 'FailingMethod', 'Question.ID', 'Answer.duration',
       'Answer.confidence', 'Answer.difficulty', 'GroundTruth', 'TP', 'TN',
       'FN', 'FP', 'Answer.option', 'Answer.order', 'Answer.explanation',
       'Code.LOC', 'Code.complexity', 'Worker.ID', 'Worker.score',
       'Worker.profession', 'Worker.yearsOfExperience', 'Worker.age',
       'Worker.gender', 'Worker.country', 'explanation.TTR', 'explanation.F
K',
       'explanation.size', 'where_learned_to_code.Books and Paper Resource
s',
       'where_learned_to_code.Diploma or Certification',
       'where_learned_to_code.Elementary School',
       'where_learned_to_code.Extracurricular Activities',
       'where_learned_to_code.High School',
       'where_learned_to_code.MOOCs and Online Resources',
       'where_learned_to_code.Middle School',
       'where_learned_to_code.Self-taught', 'where_learned_to_code.Universi
ty',
       'where_learned_to_code.Unknown/Other', 'where_learned_to_code.Web',
       'where_learned_to_code.Workplace Learning', 'programming_language..N
ET',
       'programming_language.AJAX', 'programming_language.ASP',
       'programming_language.ASP.NET', 'programming_language.Ada',
       'programming_language.Assembly', 'programming_language.AutoHotkey',
       'programming_language.AutoIt', 'programming_language.BASIC',
       'programming_language.Bash', 'programming_language.C',
       'programming_language.C#', 'programming_language.C++',
       'programming_language.CSS', 'programming_language.CUDA',
       'programming_language.Elixir', 'programming_language.Euphoria',
       'programming_language.Go', 'programming_language.Groovy',
       'programming_language.HTML', 'programming_language.JEE',
       'programming_language.JSP', 'programming_language.Java',
       'programming_language.JavaScript', 'programming_language.MATLAB',
       'programming_language.MQL4', 'programming_language.MySQL',
       'programming_language.None', 'programming_language.Object Pascal',
       'programming_language.Other', 'programming_language.PHP',
       'programming_language.PL/SQL', 'programming_language.PeopleSoft',
       'programming_language.Perl', 'programming_language.PowerShell',
       'programming_language.Python', 'programming_language.R',
       'programming_language.RPG', 'programming_language.Ruby',
       'programming_language.Ruby on Rails', 'programming_language.SAS',
       'programming_language.SQL', 'programming_language.Scala',
       'programming_language.Scheme', 'programming_language.Swift',
       'programming_language.Teradata', 'programming_language.VB.NET',
       'programming_language.VBA', 'programming_language.VBScript',
       'programming_language.VHDL', 'programming_language.Visual Basic',
       'programming_language.Visual FoxPro', 'programming_language.XML'],
      dtype='object')
```

In [4]:
```
categorical_data = mapped_data.select_dtypes(include=['object', 'category'])
numerical_data = mapped_data.select_dtypes(include=['number'])

label_encoded_data = categorical_data.apply(LabelEncoder().fit_transform)

encoded_data = pd.concat([numerical_data, label_encoded_data, categorical_da
```

In [5]:
```
feature_names = ['GroundTruth','FailingMethod', 'Answer.duration',
       'Answer.confidence', 'Answer.difficulty', 'Answer.option', 'Answer.or
       'Code.LOC', 'Code.complexity', 'Worker.score', 'Worker.yearsOfExperie
       'Worker.gender', 'Worker.country', 'explanation.TTR', 'explanation.FK
       'explanation.size']
feature_names += [column for column in encoded_data.columns if column.starts
filtered_data = encoded_data[feature_names]
```

In [6]:
```
# filter data by worker.profession in ['Graduate', 'Undergraduate']
filtered_data.loc[:,"Worker.is_student"] = encoded_data["Worker.profession_c
student_data = filtered_data[filtered_data["Worker.is_student"]].drop("Worke
```

```
/var/folders/gp/jdvwzltn6bv17g9lp1cy3kj40000gn/T/ipykernel_67680/1918298919.
py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_data.loc[:,"Worker.is_student"] = encoded_data["Worker.profession
_categorical"].isin(['Graduate_Student', 'Undergraduate_Student'])
```

In [7]:
```
# create training set
bug_reports = list(set(filtered_data["FailingMethod"]))
training_set_reports = [0,1,4,5,6,7]
training_set = student_data[student_data["FailingMethod"].isin(training_set_
y_train = training_set["GroundTruth"]
X_train = training_set.drop(["GroundTruth", "FailingMethod"], axis=1)

# create test set
test_set_reports = [2,3]
test_set = filtered_data[filtered_data["FailingMethod"].isin(test_set_report
y_test = test_set["GroundTruth"]
X_test = test_set.drop(["GroundTruth", "FailingMethod"], axis=1)
```

In [8]:
```
bug_reports
```

Out[8]:
```
[0, 1, 2, 3, 4, 5, 6, 7]
```

Train Classifier

In [9]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.utils import class_weight
from sklearn.metrics import classification_report
!export LDFLAGS="-L/opt/homebrew/opt/libomp/lib"
!export CPPFLAGS="-I/opt/homebrew/opt/libomp/include"
from xgboost import XGBClassifier, plot_importance
from sklearn.metrics import precision_score, recall_score, accuracy_score, f
from sklearn.model_selection import GridSearchCV, KFold
```

In [10]:
```
def fit_random_forest(X_train, y_train, class_weights):
    classifier = RandomForestClassifier()
```

```
    param_grid = {
        "n_estimators": [100, 200, 500, 1000],
        "max_depth": [None,3,4,5,10,12,15,20],
        "bootstrap": [False,True],
    }
    grid_search = GridSearchCV(classifier, param_grid=param_grid, cv=KFold(r
    grid_search.fit(X_train, y_train, sample_weight=class_weights)
    return grid_search

def fit_xgboost(X_train, y_train, class_weights):
    classifier = XGBClassifier(learning_rate=1, objective='binary:logistic')
    param_grid = {
        'n_estimators': [200, 500, 1000],
        'max_depth': [2,3,5,6,7],
        'subsample': [0.6,0.7,0.9],
        'colsample_bytree': [0.5, 0.7, 0.9],
        'gamma': [0.4,0.9,2,4,5],
    }
    grid_search = GridSearchCV(
        estimator=classifier,
        param_grid=param_grid,
        scoring='f1',
        cv=KFold(n_splits=5, shuffle=True, random_state=42),
        verbose=2,
        n_jobs=-1,
    )
    grid_search.fit(X_train, y_train, sample_weight=class_weights)
    return grid_search
```

In [ ]:
```
class_weights = class_weight.compute_sample_weight(
    class_weight='balanced',
    y=y_train
)
grid_search = fit_xgboost(X_train, y_train, class_weights)
grid_search
```

In [12]:
```
print("Best parameters found: ", grid_search.best_params_)
print("Best f1 found: ", grid_search.best_score_)
print("Best model: ", grid_search.best_estimator_)
print("Best model f1 on full training set: ", grid_search.best_estimator_.sc
print("Best model precision on full training set: ", precision_score(y_trair
print("Best model recall on full training set: ", recall_score(y_train, gric
print("Best model accuracy on full training set: ", accuracy_score(y_train,
```

```
Best parameters found:  {'colsample_bytree': 0.9, 'gamma': 5, 'max_depth': 
5, 'n_estimators': 1000, 'subsample': 0.6}
Best f1 found:  0.4024700122399021
Best model:  XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.9, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=5, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=1000, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
Best model f1 on full training set:  0.9365079365079365
Best model precision on full training set:  0.7692307692307693
Best model recall on full training set:  0.989010989010989
Best model accuracy on full training set:  0.9365079365079365
```

In [13]:

```
# train the best model on the full training set
grid_search.best_estimator_.fit(X_train, y_train, sample_weight=class_weight
# evaluate the best model on the test set
print("Best model f1 on full training set: ", grid_search.best_estimator_.sc
print("Best model precision on full training set: ", precision_score(y_trair
print("Best model recall on full training set: ", recall_score(y_train, gric
print("Best model accuracy on full training set: ", accuracy_score(y_train,
```

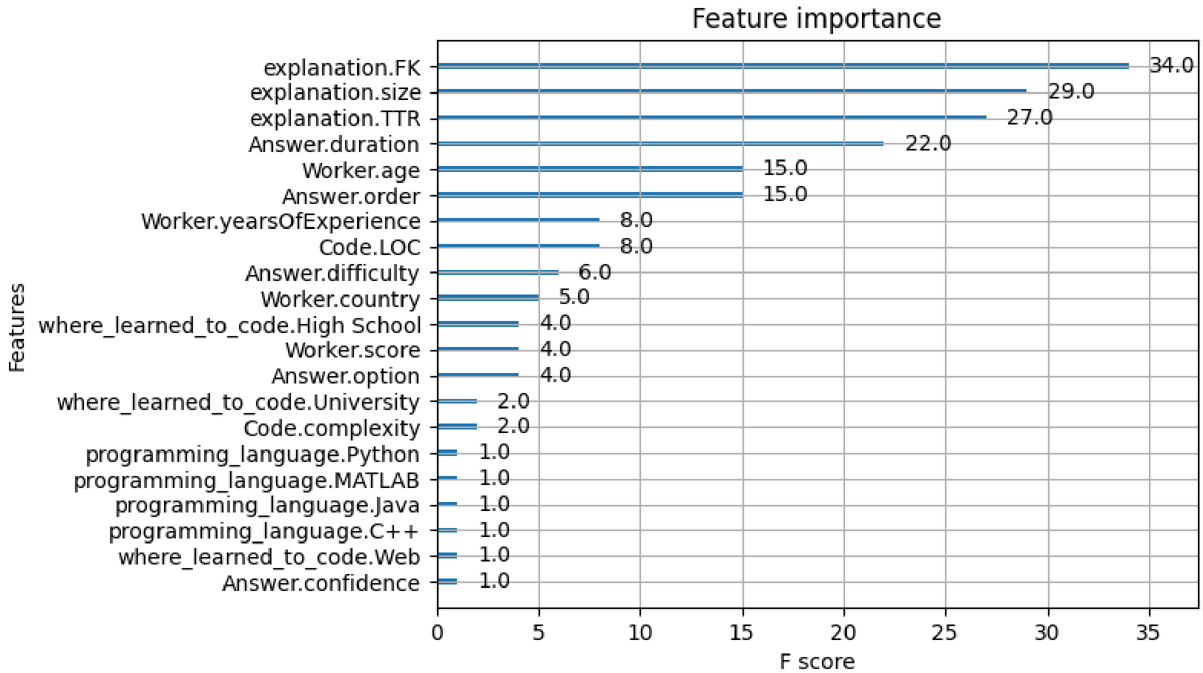
```
Best model f1 on full training set:  0.9365079365079365
Best model precision on full training set:  0.7692307692307693
Best model recall on full training set:  0.989010989010989
Best model accuracy on full training set:  0.9365079365079365
```

In [14]:

```
# visualize feature importance
plot_importance(grid_search.best_estimator_, max_num_features=200)
```

Out[14]:

```
<Axes: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Fe
atures'>
```



Evaluate Classifier

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
import tqdm
```

Exercise 1.1

Excercise Question: For the impact of 5% and 10% loss on precision and recall, what is the min number of “Non-Students” added on average to the holdout set?

The recall on a set of data points is defined as TP / (TP + FN). Now we define TP = TP1 + (x/N2) * TP2 and FP = FP1 + (x/N2) * FP2. Where TP1 and TP2 are the number of true positives achieved on the student and non student data sets respectively. Analogously for FP1 and FP2. Furthermore x is the number of non-students added to the holdout set and N2 is the amount of non-students. Note that we can define TP and FP in this way because we are interested in the average number of non-students added to the holdout set.

Now substituting TP and FP in the definition of recall we get: recall_combined = (TP1 + (x / N2) * TP2) / (TP1 + (x / N2) * TP2 + FN1 + (x / N2) * FN2)

By solving this equation for x we can find the average minimum number of non-students that would need to be added to the holdout set to achieve the desired recall: x = (TP1 - TP1 * recall_combined - FN1 * recall_combined) * N2 / (TP2 * recall_combined + FN2 * recall_combined - TP2)

The same can be done for precision, where precision is defined as TP / (TP + FP). The formula for the average minimum number of non-students that would need to be added to the holdout set to achieve the desired precision is: x = (TP1 - TP1 * precision_combined - FP1 * precision_combined) * N2 / (TP2 * precision_combined + FP2 * precision_combined - TP2)

In [16]:
```
# first evaluate on the test set that consists only of students
X_test_students = X_test[X_test["Worker.is_student"]].drop(["Worker.is_stude
y_test_students = y_test[X_test["Worker.is_student"]]
students_f1 = grid_search.best_estimator_.score(X_test_students, y_test_stuc
students_precision = precision_score(y_test_students, grid_search.best_estim
students_recall = recall_score(y_test_students, grid_search.best_estimator_.
students_accuracy = accuracy_score(y_test_students, grid_search.best_estimat
print(f"Students f1: {students_f1}")
print(f"Students precision: {students_precision}")
print(f"Students recall: {students_recall}")
print(f"Students accuracy: {students_accuracy}")
```

```
Students f1: 0.6939890710382514
Students precision: 0.18518518518518517
Students recall: 0.2459016393442623
Students accuracy: 0.6939890710382514
```

In [17]:
```
# second evaluate on the test set that consists of all non-students
X_test_non_students = X_test[~X_test["Worker.is_student"]].drop(["Worker.is_
y_test_non_students = y_test[~X_test["Worker.is_student"]]
non_students_f1 = grid_search.best_estimator_.score(X_test_non_students, y_t
non_students_precision = precision_score(y_test_non_students, grid_search.be
non_students_recall = recall_score(y_test_non_students, grid_search.best_est
non_students_accuracy = accuracy_score(y_test_non_students, grid_search.best
print(f"Non-students f1: {non_students_f1}")
print(f"Non-students precision: {non_students_precision}")
print(f"Non-students recall: {non_students_recall}")
print(f"Non-students accuracy: {non_students_accuracy}")
```

```
Non-students f1: 0.6456582633053222
Non-students precision: 0.18981481481481483
Non-students recall: 0.3445378151260504
Non-students accuracy: 0.6456582633053222
```

In [18]:
```
# compare the percentage of positives in the student and non-student test se
print("Percentage of positives in student test set: ", y_test_students.sum()
print("Percentage of positives in non-student test set: ", y_test_non_studer
print("Percentage of positives in student training set: ", y_train.sum() / l
```

```
Percentage of positives in student test set:  0.16666666666666666
Percentage of positives in non-student test set:  0.16666666666666666
Percentage of positives in student training set:  0.20634920634920634
```

From the results here we can already see that the x we are going to compute has to be negative. This is due to the fact that the precision and recall in the non-student set are actually higher than in the student set. Though this is unexpected, it indicates that the student property is likely not the most important feature in the dataset.

In [19]:
```
# recall_combined = (TP1 + (x / N2) * TP2) / (TP1 + (x / N2) * TP2 + FN1 + (
# r = (A + (x / N) * B) / (A + (x / N) * B + C + (x / N) * D)
# x = (A * N - A * N * r - N * C * r) / (B * r + D * r - B)
# x = (TP1 * N2 - TP1 * N2 * recall_combined - FN1 * N2 * recall_combined) /
# x = (TP1 - TP1 * recall_combined - FN1 * recall_combined) * N2 / (TP2 * re

student_predictions = grid_search.best_estimator_.predict(X_test_students)
non_student_predictions = grid_search.best_estimator_.predict(X_test_non_stu

TP_1 = np.sum(student_predictions & y_test_students)
FN_1 = np.sum(~student_predictions & y_test_students)
TP_2 = np.sum(non_student_predictions & y_test_non_students)
FN_2 = np.sum(~non_student_predictions & y_test_non_students)
N2 = len(y_test_non_students)
students_recall = recall_score(y_test_students, student_predictions)
recall_95 = 0.95 * students_recall
recall_90 = 0.9 * students_recall

x_95 = (TP_1 - TP_1 * recall_95 - FN_1 * recall_95) * N2 / (TP_2 * recall_95
x_90 = (TP_1 - TP_1 * recall_90 - FN_1 * recall_90) * N2 / (TP_2 * recall_90
print(f"Need to add {x_95} non-students to achieve a 5% increase in recall a

# precision_combined = (TP1 + (x / N2) * TP2) / (TP1 + (x / N2) * TP2 + FP1
# x = (TP1 * N2 - TP1 * N2 * recall_combined - FP1 * N2 * recall_combined) /
# x = (TP1 - TP1 * recall_combined - FP1 * recall_combined) * N2 / (TP2 * re

# calculate FP1, TP1, FP2, TP2 where 1 is students and 2 is non-students
FP_1 = np.sum(student_predictions & ~y_test_students)
FP_2 = np.sum(non_student_predictions & ~y_test_non_students)
students_precision = precision_score(y_test_students, student_predictions)
precision_95 = 0.95 * students_precision
precision_90 = 0.9 * students_precision

x_95 = (TP_1 - TP_1 * precision_95 - FP_1 * precision_95) * N2 / (TP_2 * pre
x_90 = (TP_1 - TP_1 * precision_90 - FP_1 * precision_90) * N2 / (TP_2 * pre
print(f"Need to add {x_95} non-students to achieve a 5% increase in precisio
```

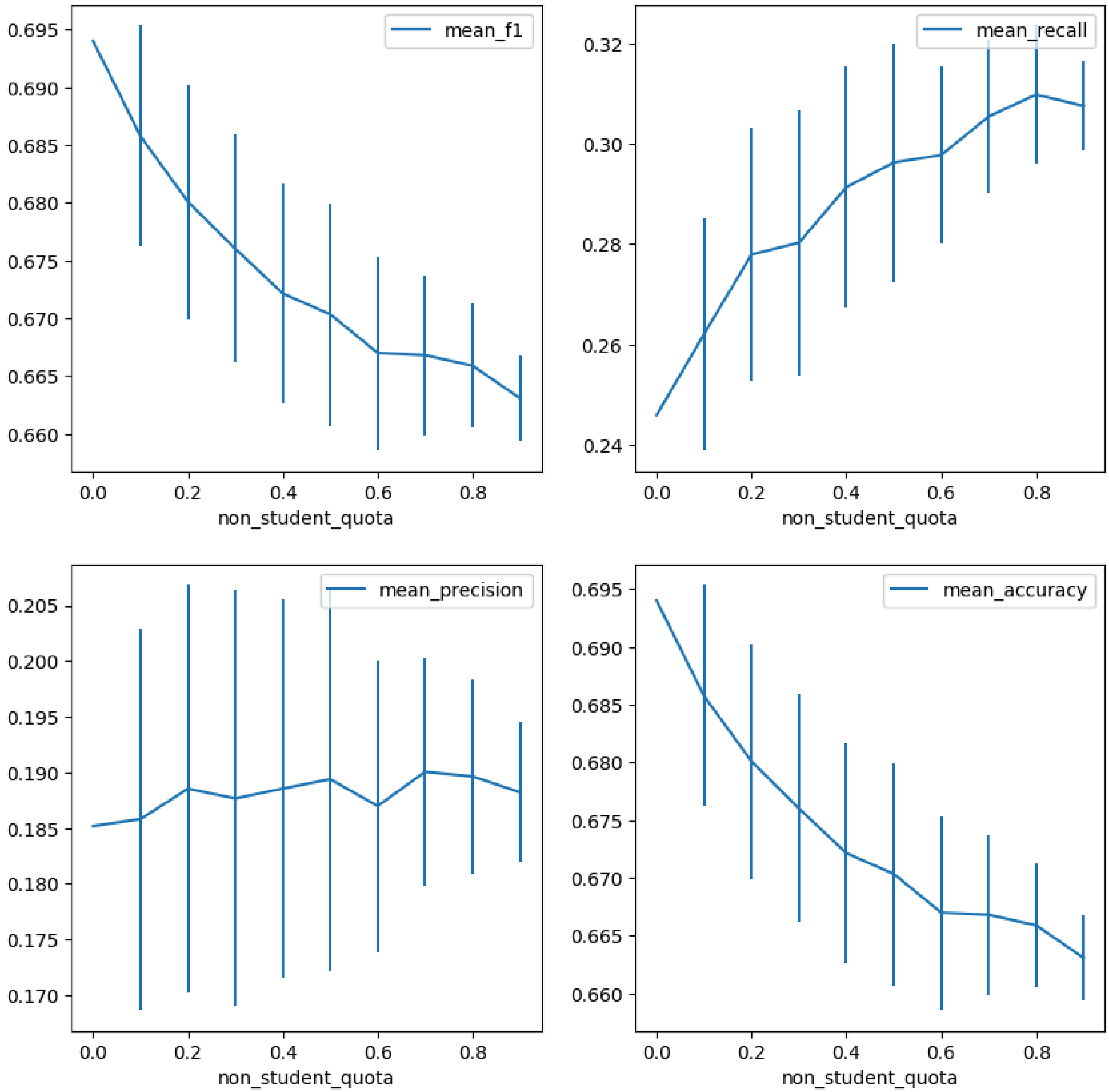
```
Need to add -40.56566283762815 non-students to achieve a 5% increase in reca
ll and -73.0363331470095 for a 10% increase
Need to add -178.50000000000043 non-students to achieve a 5% increase in pre
cision and -214.2 for a 10% increase
```

Below you can see an empirical approach to the problem. We implemented that in the beginning to get a feeling for the problem. The idea is to randomly select different numbers of non-students to add to the student set a hundred times and then calculate the average recall and precision.

In [20]:
```python
# we are trying to figure out what amount of non-students need to be added t
def run_evaluation_for_non_student_quota(non_student_quota, X_test, y_test,
    f1_scores = []
    recall_scores = []
    precision_scores = []
    accuracy_scores = []
    for _ in range(n_probes):
        X_test_non_students = X_test[~X_test["Worker.is_student"]].drop(["Wo
        y_test_non_students = y_test[~X_test["Worker.is_student"]]
        X_test_students = X_test[X_test["Worker.is_student"]].drop(["Worker.
        y_test_students = y_test[X_test["Worker.is_student"]]

        X_test_non_students = X_test_non_students.sample(frac=non_student_qu
        y_test_non_students = y_test_non_students[X_test_non_students.index]

        X_test_combined = pd.concat([X_test_students, X_test_non_students])
        y_test_combined = pd.concat([y_test_students, y_test_non_students])

        f1_scores.append(grid_search.best_estimator_.score(X_test_combined,
        recall_scores.append(recall_score(y_test_combined, grid_search.best_
        precision_scores.append(precision_score(y_test_combined, grid_search
        accuracy_scores.append(accuracy_score(y_test_combined, grid_search.b
    return f1_scores, recall_scores, precision_scores, accuracy_scores

results_df = pd.DataFrame(columns=["non_student_quota", "mean_f1", "std_f1",
for non_student_quota in np.arange(0.0, 1.0, 0.1):
    random_generator = np.random.default_rng(seed=42)
    f1_scores, recall_scores, precision_scores, accuracy_scores = run_evalua
    results_df = pd.concat([results_df, pd.DataFrame({
        "non_student_quota": [non_student_quota],
        "mean_f1": [np.mean(f1_scores)],
        "std_f1": [np.std(f1_scores)],
        "mean_recall": [np.mean(recall_scores)],
        "std_recall": [np.std(recall_scores)],
        "mean_precision": [np.mean(precision_scores)],
        "std_precision": [np.std(precision_scores)],
        "mean_accuracy": [np.mean(accuracy_scores)],
        "std_accuracy": [np.std(accuracy_scores)]
    })])
```

```
/var/folders/gp/jdvwzltn6bv17g9lp1cy3kj40000gn/T/ipykernel_67680/4274380840.
py:29: FutureWarning: The behavior of DataFrame concatenation with empty or
all-NA entries is deprecated. In a future version, this will no longer exclu
de empty or all-NA columns when determining the result dtypes. To retain the
old behavior, exclude the relevant entries before the concat operation.
  results_df = pd.concat([results_df, pd.DataFrame({
```

In [21]:
```python
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
results_df.plot(x="non_student_quota", y="mean_f1", yerr="std_f1", ax=ax[0,
results_df.plot(x="non_student_quota", y="mean_recall", yerr="std_recall", a
results_df.plot(x="non_student_quota", y="mean_precision", yerr="std_precisi
results_df.plot(x="non_student_quota", y="mean_accuracy", yerr="std_accuracy
plt.show()
```



Exercise 1.2

Exercise Question: What is the min number of “Non-Students” to train a model that produces similar outcome to the model trained on mixed data (from mini project 2)?

To answer the Question we create each 10 different combinations of training sets for different numbers of non-student samples to be contained in the training set. We then train a model on each of these training sets and evaluate the model on the mixed data holdout set. We then calculate the average recall and precision (and the std) for each amount of non-student samples. We plot the the results to see how the recall and precision change with the number of non-student samples in the training set.

In [30]:
```
def train_and_eval_classifier(
    X_train_1,
```

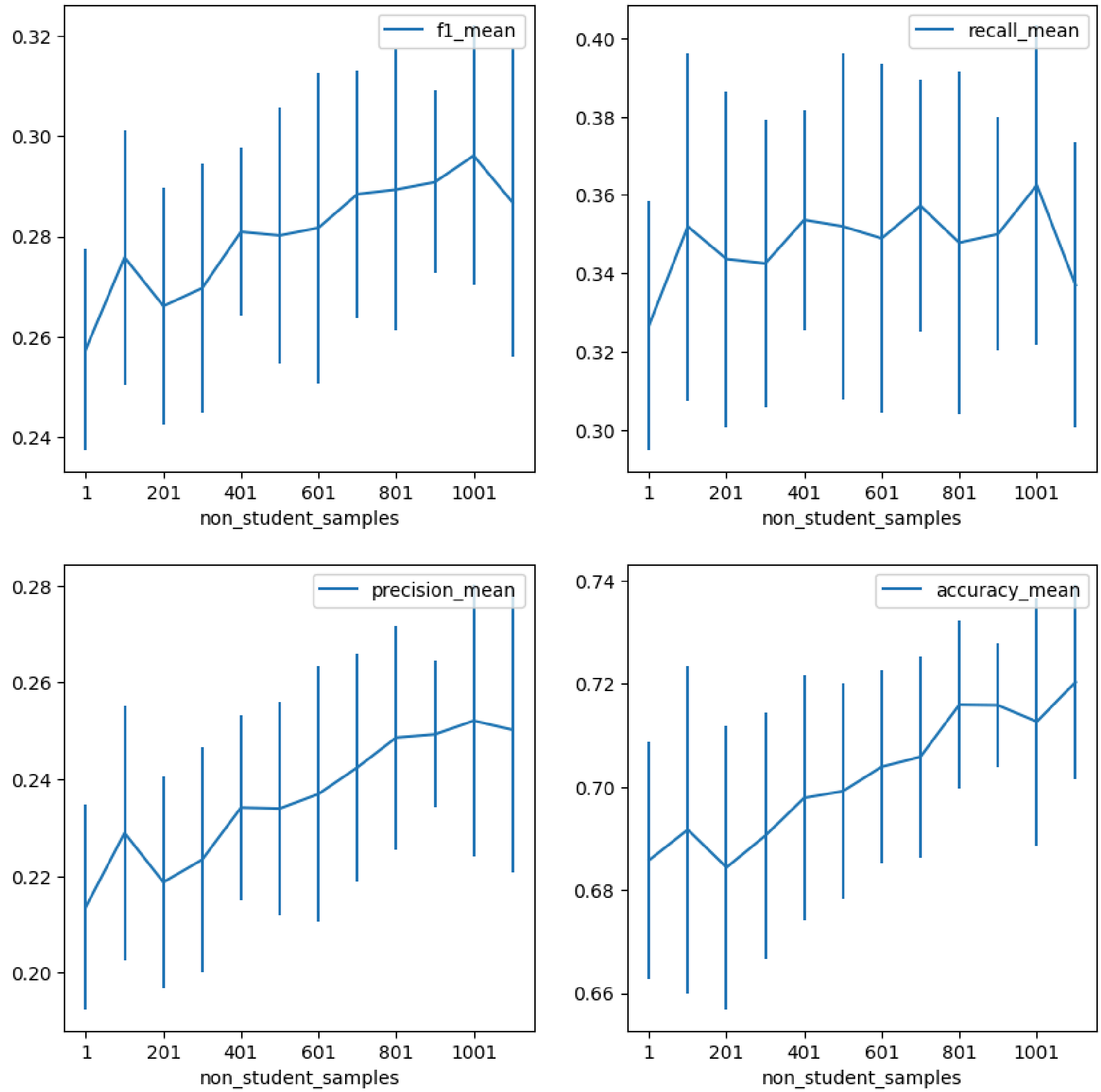
```
    y_train_1,
    X_train_2,
    y_train_2,
    samples_2,
    X_test,
    y_test,
    classifier,
    n_probes,
):
    random_generator = np.random.default_rng(seed=42)
    f1_scores = []
    recall_scores = []
    precision_scores = []
    accuracy_scores = []
    for _ in range(n_probes):
        # create X_train by adding samples_2 of X_train_2 to X_train_1
        X_train_2_sampled = X_train_2.sample(n=samples_2, random_state=rando
        X_train = pd.concat([X_train_1, X_train_2_sampled])
        y_train = pd.concat([y_train_1, y_train_2.loc[X_train_2_sampled.inde

        class_weights = class_weight.compute_sample_weight(
            class_weight="balanced", y=y_train
        )
        classifier.fit(X_train, y_train, sample_weight=class_weights)
        predictions = classifier.predict(X_test)
        f1_scores.append(f1_score(y_test, predictions))
        recall_scores.append(recall_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions))
        accuracy_scores.append(accuracy_score(y_test, predictions))
    return f1_scores, recall_scores, precision_scores, accuracy_scores


classifier = grid_search.estimator.set_params(**grid_search.best_params_)
training_set = filtered_data[filtered_data["FailingMethod"].isin(training_se
non_student_training_data = training_set[~training_set["Worker.is_student"]]
    "Worker.is_student", axis=1
)
X_train_non_students = non_student_training_data.drop(
    ["GroundTruth", "FailingMethod"], axis=1
)
y_train_non_students = non_student_training_data["GroundTruth"]

results_df = pd.DataFrame(columns=["non_student_samples", "f1_mean", "f1_stc
for non_student_samples in tqdm.tqdm(list(range(1, len(y_train_non_students)
    f1_scores, recall_scores, precision_scores, accuracy_scores = (
        train_and_eval_classifier(
            X_train,
            y_train,
            X_train_non_students,
            y_train_non_students,
            non_student_samples,
            X_test.drop(["Worker.is_student"], axis=1),
            y_test,
            classifier,
            20,
        )
```

```
    )
    results_df = pd.concat(
        [
            results_df,
            pd.DataFrame(
                {
                    "non_student_samples": [non_student_samples],
                    "f1_mean": [np.mean(f1_scores)],
                    "f1_std": [np.std(f1_scores)],
                    "recall_mean": [np.mean(recall_scores)],
                    "recall_std": [np.std(recall_scores)],
                    "precision_mean": [np.mean(precision_scores)],
                    "precision_std": [np.std(precision_scores)],
                    "accuracy_mean": [np.mean(accuracy_scores)],
                    "accuracy_std": [np.std(accuracy_scores)],
                }
            ),
        ]
    )
```

```
  0%|          | 0/12 [00:00<?, ?it/s]/var/folders/gp/jdvwzltn6bv17g9lp1cy3k
j40000gn/T/ipykernel_67680/3919169521.py:60: FutureWarning: The behavior of
DataFrame concatenation with empty or all-NA entries is deprecated. In a fut
ure version, this will no longer exclude empty or all-NA columns when determ
ining the result dtypes. To retain the old behavior, exclude the relevant en
tries before the concat operation.
  results_df = pd.concat(
100%|██████████| 12/12 [01:37<00:00,  8.12s/it]
```

In [31]:

```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
results_df.plot(x="non_student_samples", y="f1_mean", yerr="f1_std", ax=ax[0
results_df.plot(x="non_student_samples", y="recall_mean", yerr="recall_std",
results_df.plot(x="non_student_samples", y="precision_mean", yerr="precision
results_df.plot(x="non_student_samples", y="accuracy_mean", yerr="accuracy_s
plt.show()
```



In [27]:
```
print(f"max number of non-students samples in the training set: {len(y_trair
print(f"number of students samples in the training set: {len(y_train)}")
```

```
max number of non-students samples in the training set: 1059
number of students samples in the training set: 441
```

As we can see from the plot precision and recall seem to be increasing with the number of non-student samples in the training set. However this has to be interpreted with caution since the standard deviation is quite high. Another interesting observation is that both recall and precision seem to decrease again when using all non-student samples in the training set. Therefore judging by the f1 score, the smallest amount of non-student samples in the training set that achieves similar results to training on mixed data likely lies between 600 and 700.

However one needs to be very careful with interpreting any meaning in this result. We don't know the correlation between the student property and the other features in the dataset and we already see quite large standard deviations in the results.