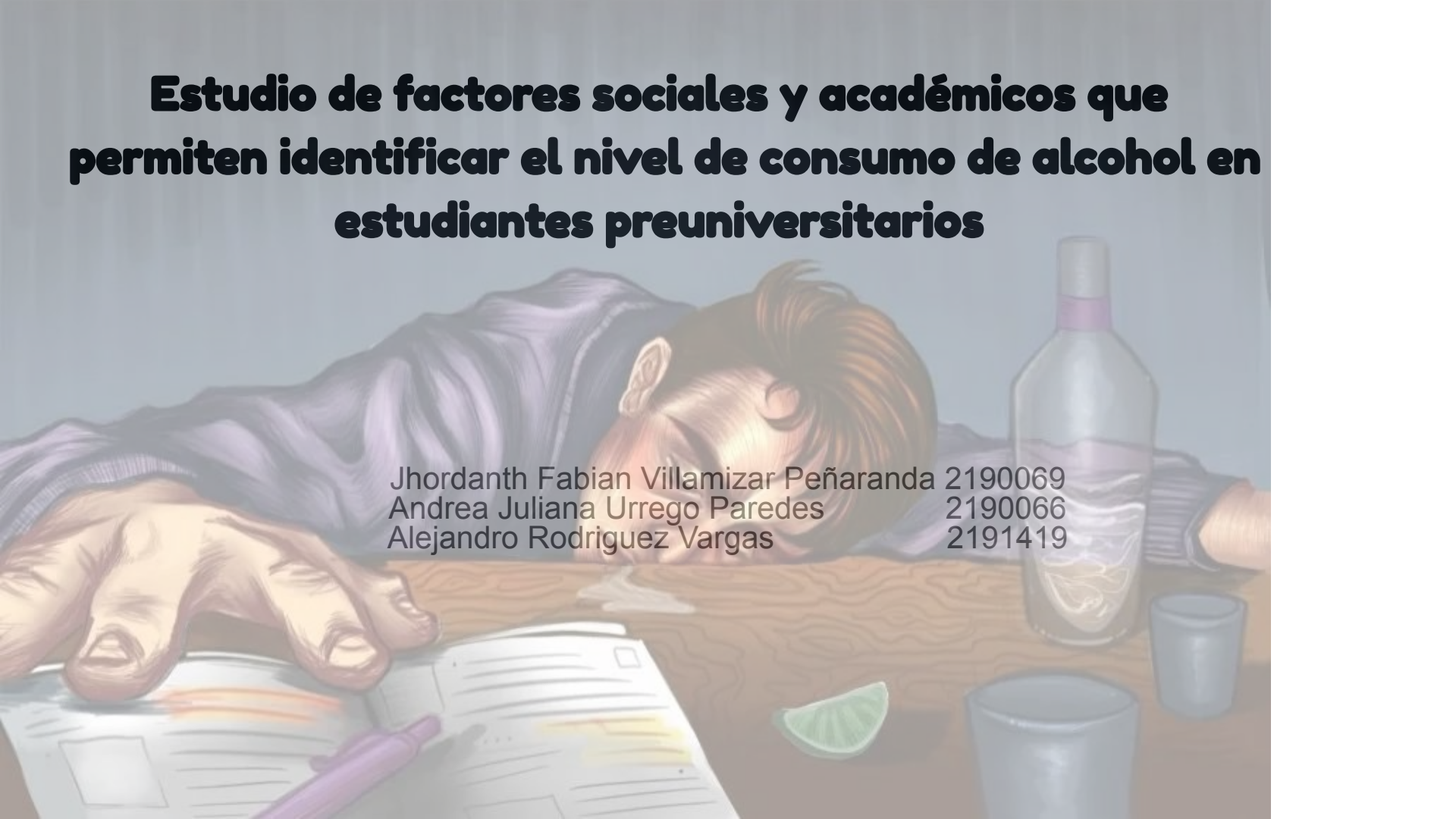


Estudio de factores sociales y académicos que permiten identificar el nivel de consumo de alcohol en estudiantes preuniversitarios

An illustration of a young man with brown hair, wearing a purple sweater, sleeping at a wooden desk. His head is resting on his hand, which is holding a pen. On the desk, there is an open book with a purple pen resting on it. To the right of the student, there is a clear glass bottle of alcohol, a small glass, a larger glass, and a slice of lime. The background is a simple, light blue wall.

Jhordanth Fabian Villamizar Peñaranda 2190069
Andrea Juliana Urrego Paredes 2190066
Alejandro Rodriguez Vargas 2191419

Análisis del dataset

Para realizar este estudio se tomó en cuenta las condiciones y comportamientos, de 649 estudiantes de una clase, buscando patrones en los datos que pudiesen indicar la tendencia que tiene un estudiante a consumir bebidas alcohólicas.

Para poder realizar esta diferenciación de patrones de comportamiento se tendrá en cuenta las variables weekend alcohol consumption (Consumo de alcohol el fin de semana) y workday alcohol consumption (consumo de alcohol en días laborales)(**Walc y Dalc en el dataset** respectivamente), las cuales toman valores entre 1 y 5, siendo 5 muy alto y 1 muy bajo, en este análisis se decidió sumar ambas variables para tomarlas como **ground truth**, por tanto ahora nuestro ground truth tomará valores entre 2 y 10.

La información del dataset puede encontrarse en el siguiente link:

<https://www.kaggle.com/datasets/uciml/student-alcohol-consumption>

Eliminación de columnas menos importantes del dataset.

Relacionando los datos para el respectivo análisis que queremos realizar, algunas columnas no tenían mucha relación en cuanto a poder ver el consumo de alcohol de los estudiantes, como por ejemplo el tiempo de viaje.

```
## limpiar dataset

import pandas as pd

dp.drop(['school','reason','traveltime', 'failures','paid','nursery'],axis=1, inplace=True)

dp.info()
```

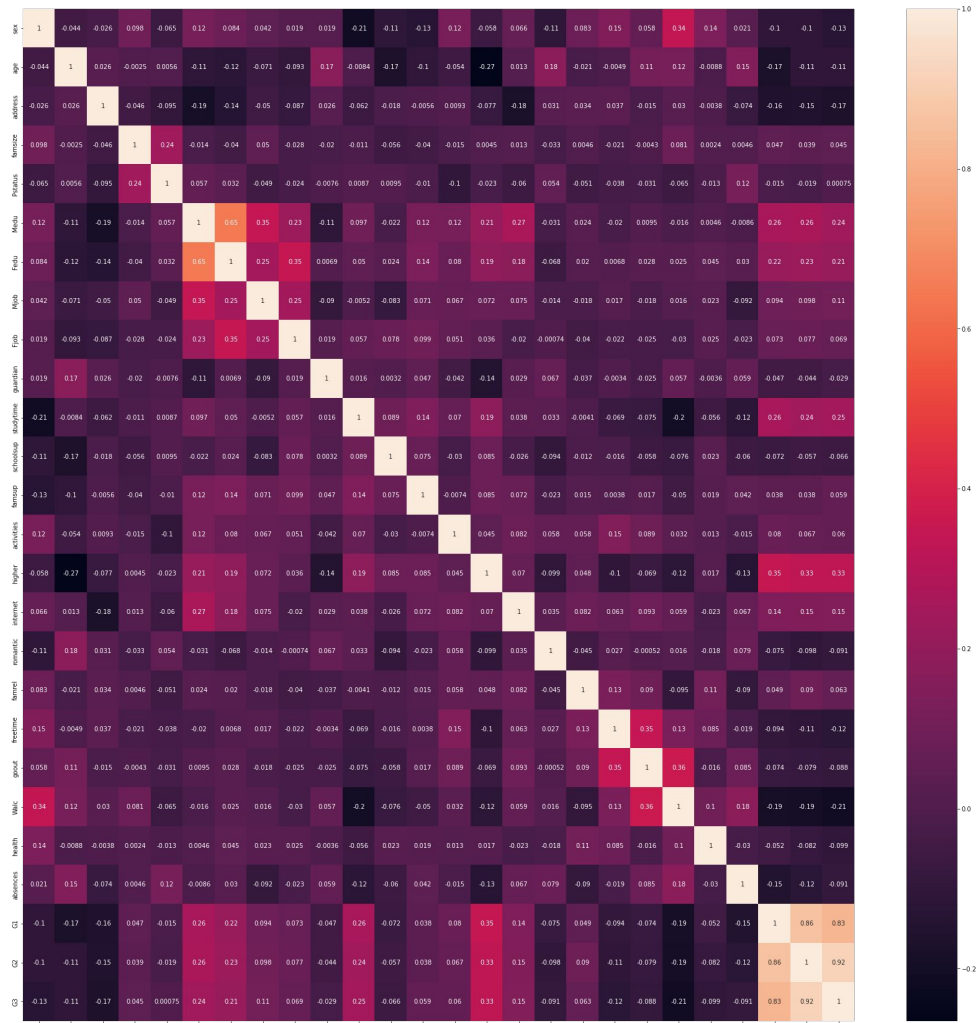
```
✓ ▶ dp['sex'].unique()
dp['sex'] = dp['sex'].map(
    {
        'F': 0,
        'M': 1,
    }
)

dp['address'].unique()
dp['address'] = dp['address'].map(
    {
        'U': 0,
        'R': 1,
    }
)

dp['famsize'].unique()
dp['famsize'] = dp['famsize'].map(
    {
        'GT3': 0,
        'LE3': 1,
    }
)
```

Conversión de
las columnas
del dataset a
valores
numéricos.

Correlación de Pearson



Uso del feature importances

Al final nos quedamos con 24 columnas.

```
features= est.feature_importances_  
bestfeat = np.argsort(features[::-1])  
bestcols = bestfeat[0:24]  
X_train_best = X_train.values[:, bestcols]  
X_test_best = X_test.values[:, bestcols]
```

Usando el DecisionTreeClassifier

Para calcular algunas métricas como lo son el recall score, el accuracy sin el uso de feature importances.

```
X = dp[['sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'guardian', 'studytime', 'schoolsup', 'famsup', 'activities', 'higher', 'internet']
y = dp['Walc']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False, random_state=21)

est = DecisionTreeClassifier(random_state=21)
est.fit(X_train, y_train)
pred = est.predict(X_test)
print('prediction: ', pred)
print('grand truth: ', y_test.values)
print('accuracy socre: ', accuracy_score(pred, y_test))
print('recall socre: ', recall_score(y_test, pred, average='macro'))
print('f1 socre: ', f1_score(y_test, pred, average='macro'))
```



```

est = DecisionTreeClassifier(random_state=21, criterion='gini')
est.fit(X_train_best, y_train)
score = est.score(X_test_best, y_test)
pred = est.predict(X_test_best)
print('prediction: ',pred)
print('grand truth: ',y_test.values)
print('accuracy socre: ', accuracy_score(pred, y_test))
print('recall socre: ', recall_score(y_test,pred, average='macro'))
print('f1 socre: ', f1_score(y_test, pred, average='macro'))

```

```

prediction: [ 6  2  2  8  6  3  3  3  5  4  7  3  3  6  4  2  8  2  2  3  9  2  6  4
  7  5  2  2  2  2  3  4  2  3  3  3  7  2  6  2  5 10  6  3  3  8  3
  8  2  3  4  8  2  3  6  2  4  2  3  3  3  2  6  3  2  2  3  2  8  3  3
  4  4  5  4  6  4  2  5  4  3  3  7  4  6  4  6  2  5  4  8  3  2  4  2
  2  2  3  2  6  5  6  4  9  6 10  3  4  4  6  8  2  2  4  6  5  2  6  3
  7  2  3  8  2  4  5  5  8  4  3  5  4  2  2  2  3  2  2  3  2  3  2  5
  9  5  3  6  4  2  2  4  2  3  2  2  8  3  9  2  4  4  5  3  4  2  5  3
  6  3  2  4  3  4  3  6  7  4  5  2  2  2  7  2  2  2  5  4  4  7  2  4
  3  7  4]
test: [ 3  6  3 10  2  2  2  7  3  4  7  5  4  2  3  2  2  5  2  5  8  2  2  6
  2  2  2  2  4  3  3  3  5  2  2  7  2  5  2  8  4  2  4  7  2  3 10  6
  8  2  2  3  6  2  3  2  2  2  2  4  5  5  5  5  4  2  4  2  3 10  2  3
  3  4  3  3 10  6  5  2  7  4  5  3  4  4  4  2  5  4  6  7  2  3  3  5
  4  4  7  2  4  2  6  2  8  2  4  5  2  5  5  6  5  4  2  4  6  4  2  3
  6  7 10  6  5  5  6  2  2  2  2  4  4  3  2  2  4  2  2  3  3  2  3  2
 10  2  7  2  5  2  3  5  4  3  3  2  5  2  4  2  3  3  3  2  5  4  2  3
  6  4  3  3  6  5  2  4  2  3  2  7  3  5  5  4  2  4  7  4  2  4  3  2
  2  7  7]

```

Y con el uso
de feature
importances

Uso de RandomForestClassifier sin feature importances

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False, random_state=21)
s1plot= []
s2plot= []
s3plot= []
splot= []
bests1plot= []
bests2plot= []
bests3plot= []
bestsplot= []
for x in n:
    est = RandomForestClassifier(n_estimators=x+1, random_state=21, criterion='gini')
    est.fit(X_train, y_train)
    score = est.score(X_test, y_test)
    pred = est.predict(X_test)
    s1 = jaccard_score(pred, y_test, average='micro')
    s2 = accuracy_score(pred, y_test)
    s3 = balanced_accuracy_score(pred, y_test)
    s1plot.append(s1)
    s2plot.append(s2)
    s3plot.append(s3)
    splot.append(score)
```

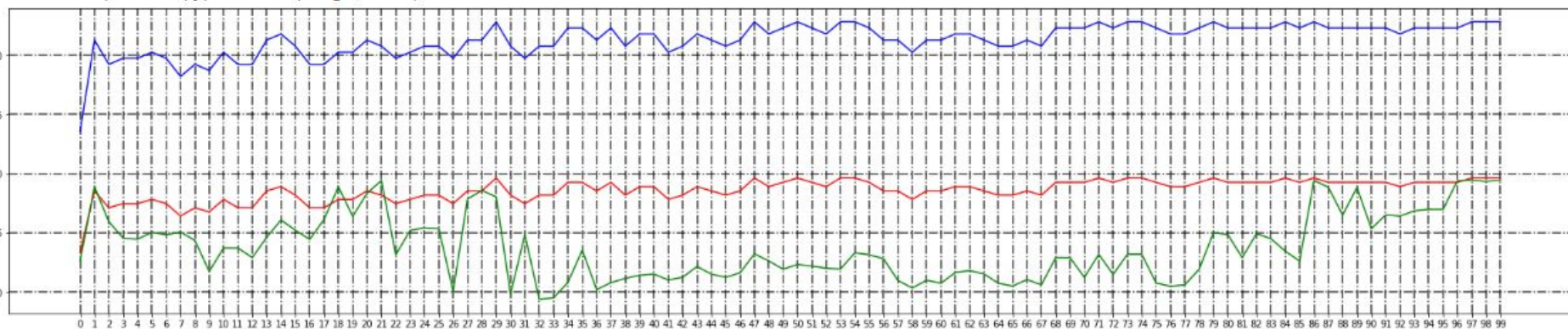
Uso de RandomForestClassifier con feature importances

```
for x in n:
    est = RandomForestClassifier(n_estimators=x+1, random_state=21, criterion='gini')
    est.fit(X_train_best, y_train)
    score = est.score(X_test_best, y_test)
    pred = est.predict(X_test_best)
    s1 = jaccard_score(pred, y_test, average='micro')
    s2 = accuracy_score(pred, y_test)
    s3 = balanced_accuracy_score(pred, y_test)
    bests1plot.append(s1)
    bests2plot.append(s2)
    bests3plot.append(s3)
    bestsplot.append(score)
```

Gráfica de la predicción con feature importances

```
ax= plt
meanax = plt

ax.figure(figsize=(30,6))
ax.plot(bests1plot, color='red')
ax.plot(bests2plot, color='blue')
ax.plot(bests3plot, color='green')
#ax.plot(bestsplot)
ax.xticks(range(0, 100, 1))
ax.grid(alpha=0.8, color='Black',linestyle='-.',linewidth=1.5)
ax.show
```

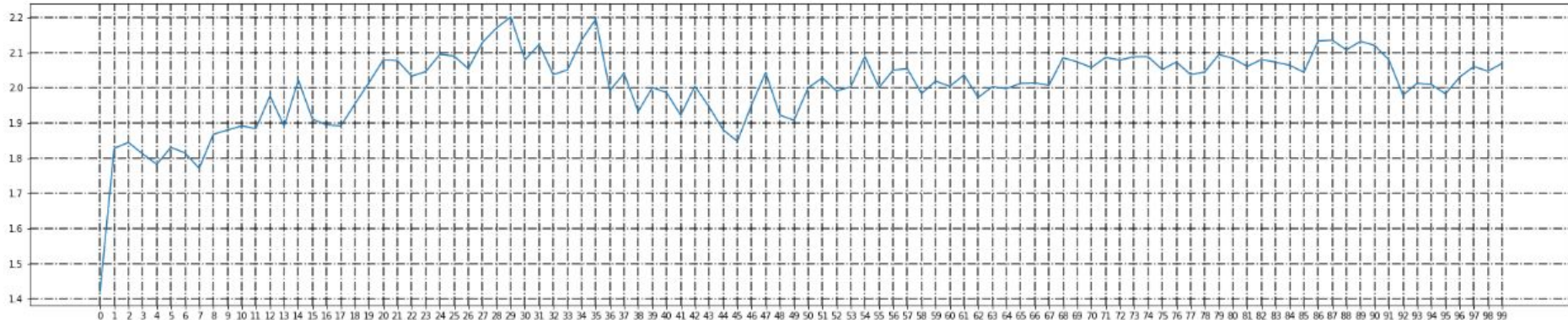


Gráfica de la suma de las dos predicciones

```
mean=[]

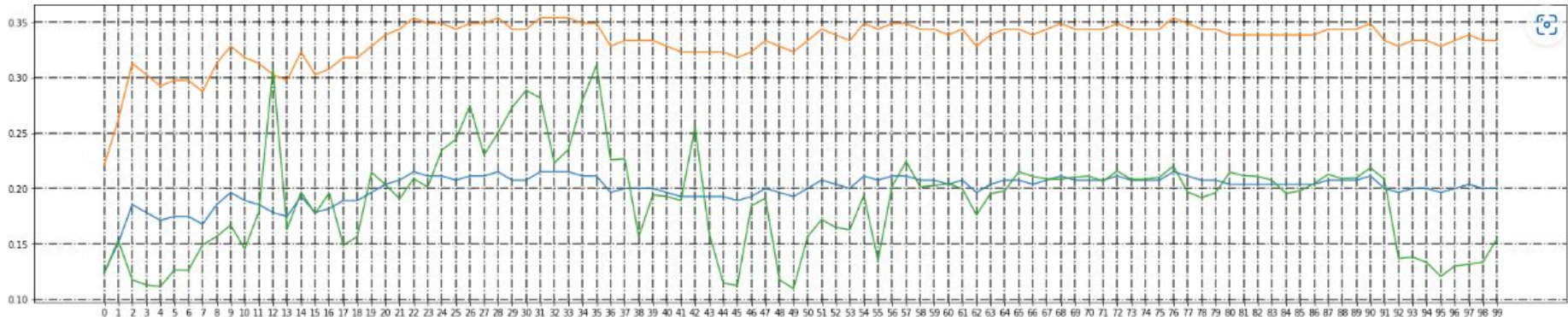
for n in range(100):
    mean.append(bests1plot[n]+bests2plot[n]+bests3plot[n]+bestsplot[n]+s1plot[n]+s2plot[n]+s3plot[n]+splot[n])

meanax.figure(figsize=(30,6))
meanax.plot(mean)
meanax.xticks(range(0, 100, 1))
meanax.grid(alpha=0.8, color='Black',linestyle='-.',linewidth=1.5)
meanax.show
```



Gráfica de las predicciones sin feature importances

```
plt.figure(figsize=(30,6))
plt.plot(s1plot)
plt.plot(s2plot)
plt.plot(s3plot)
#plt.plot(splot)
plt.xticks(range(0, 100, 1))
plt.grid(alpha=0.8, color='Black',linestyle='-.',linewidth=1.5)
plt.show
```



Mediante las gráficas y usando RandomForest buscamos la mejor cantidad de n estimators.

```
est = RandomForestClassifier(n_estimators=30, random_state=21, criterion='gini')
est.fit(X_train_best, y_train)
score = est.score(X_test_best, y_test)
pred = est.predict(X_test_best)
s1 = jaccard_score(pred, y_test, average='micro')
s2 = accuracy_score(pred, y_test)
s3 = balanced_accuracy_score(pred, y_test)

print('Score = ', score,
      'S1 = ', s1,
      'S2 = ', s2,
      'S3 = ', s3)
```

Aplicamos GaussianNB

Podemos ver que con el GaussianNB sabemos que es un método menos preciso, las métricas fueron mucho peores.

```
X = dp[['sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'guardian', 'studytime', 'schoolsup', 'famsup', 'activities', 'higher', 'internet',  
y = dp['Walc']  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False, random_state=21)  
  
est = GaussianNB()  
est.fit(X_train, y_train)  
pred = est.predict(X_test)  
print('prediction: ', pred)  
print('grand truth: ', y_test.values)  
print('accuracy socre: ', accuracy_score(pred, y_test))  
print('recall socre: ', recall_score(y_test, pred, average='macro'))  
print('f1 socre: ', f1_score(y_test, pred, average='macro'))
```



```

prediction: [ 8 2 3 9 2 2 3 2 2 2 9 8 9 3 9 9 8 8 2 9 9 2 8 9
 2 8 8 8 8 8 8 8 8 8 8 8 8 6 8 8 8 9 3 3 8 9
 9 2 8 8 8 8 9 3 2 8 8 8 8 6 8 8 8 9 8 8 8 8 9
 8 9 8 8 9 9 9 3 8 9 8 8 8 8 2 8 9 9 9 8 9 8 2
 8 9 8 8 8 2 9 8 8 9 9 9 8 9 9 8 8 9 8 9 8 8 3
 7 8 9 9 9 8 9 8 8 8 3 8 10 8 8 8 8 8 8 3 9 3 8
 9 9 8 8 8 8 8 8 3 8 8 8 8 2 9 8 8 8 9 3 8 3 8 2
 9 9 3 8 8 8 8 8 8 2 8 8 3 8 9 9 9 9 2 8 8 8 3
 8 9 9]
grand truth: [ 3 6 3 10 2 2 2 7 3 4 7 5 4 2 3 2 2 5 2 5 8 2 2 6
 2 2 2 2 4 3 3 3 5 2 2 7 2 5 2 8 4 2 4 7 2 3 10 6
 8 2 2 3 6 2 3 2 2 2 2 4 5 5 5 5 4 2 4 2 3 10 2 3
 3 4 3 3 10 6 5 2 7 4 5 3 4 4 4 2 5 4 6 7 2 3 3 5
 4 4 7 2 4 2 6 2 8 2 4 5 2 5 5 6 5 4 2 4 6 4 2 3
 6 7 10 6 5 5 6 2 2 2 2 4 4 3 2 2 4 2 2 3 3 2 3 2
 10 2 7 2 5 2 3 5 4 3 3 2 5 2 4 2 3 3 3 2 5 4 2 3
 6 4 3 3 6 5 2 4 2 3 2 7 3 5 5 4 2 4 7 4 2 4 3 2
 2 7 7]
accuracy socre: 0.09230769230769231
recall socre: 0.06736111111111111
f1 socre: 0.05905353690093004

```