Modelo de Clasificación Logistica

El objetivo de este cuaderno es explorar la regresión logística y la ingeniería de características con las funciones de machine learning.

En este cuaderno, se utilizará la sabana de notas de los estudiantes para predecir la nota de matematicas determinando si paso o no la asignatura.

- Se usa SFrames para hacer algunas funciones de ingeniería
- Se entrena un modelo de regresión logística para predecir la calificacion de matematicas de los estudiantes.
- Se inspeccionan los pesos (coeficientes) de un modelo entrenado de regresión logística.
- Se hace una predicción (tanto de clase como de probabilidad) de la calificación para una nueva nota de la asignatura.
- Dados los pesos de regresión logística, los predictores y las notas de verdad básica, se crea una función para calcular la **precisión** del modelo.
- Se inspeccionan los coeficientes del modelo de regresión logística e interpreta sus significados con el fin de determinar las probabilidades.
- Se comparan dos modelos de regresión logistica para determinar cual es mejor

Importanción de las librerias

In [49]:
```
import pandas as pd
import turicreate as tr
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
```

Preparación y Entendimiento de los datos

Se utilizara un conjunto de datos que consiste en la sabana de notas de los estudiantes.

In [50]:
```
ruta = 'https://drive.google.com/uc?id=12y4Yirhk1B-Cy6jMTeMPa1SHHdxA
students=tr.SFrame.read_csv(ruta, '&')
```

```
Finished parsing file https://drive.google.com/uc?id=12y4Yirhk1B-
Cy6jMTeMPa1SHHdxAuhFm&export=download&authuser=0

Parsing completed. Parsed 100 lines in 0.005879 secs.

------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[str,str,str,int,str,int,int,float,int,float,fl
oat,float,float,float,float,float,float,float,float,float,float,f
loat,float,str,str,str,str]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------

Finished parsing file https://drive.google.com/uc?id=12y4Yirhk1B-
Cy6jMTeMPa1SHHdxAuhFm&export=download&authuser=0

Parsing completed. Parsed 214 lines in 0.003314 secs.
```

Ahora se puede tener una vista previa de como se ve el conjunto de datos

In [51]:
```
students.head()
```

Out[51]:

ESTUDIANTE	R	S	CURSO	J10	Estrato	AÑO	ECONOMIA	FÍSICA
AGREDO TORRES KAROL MELISA ...	N	F	1001	M	2	2016	69.7	73
ARIZA MARTINEZ CAMPVERLLY SAMANTHA ...	N	F	1001	M	2	2016	68.7	68
CAÑON SUAREZ JESSICA VIVIANA ...	N	F	1001	M	2	2016	42.7	45
CASALLAS HUERFANO LINA MARIA ...	N	F	1001	M	2	2016	74.7	63
CHUNZA GOMEZ LIZETH ANDREA ...	N	F	1001	M	2	2016	65.0	53
FARFAN PULIDO ANDRES FELIPE ...	N	M	1001	M	2	2016	43.3	43
GALINDO MORENO MARILYN DANIELA ...	N	F	1001	M	2	2016	60.7	60
GAVIRIA HERRERA SANTIAGO	N	M	1001	M	2	2016	62.3	52
GOMEZ ROMERO JONATHAN	N	M	1001	M	2	2016	63.0	55
GONZALEZ RODRIGUEZ HENRY JOEL ...	N	M	1001	M	2	2016	61.7	54

EDUCACION ARTISTICA	RECREACIÓN Y DEPORTES	RELIGION	ETICA	FILOSOFIA	INGLE
78.33333333	62.33333333	68.33333333	74.66666667	80.33333333	65.66666
79.33333333	71.66666667	69.33333333	73.66666667	62.33333333	60.0
40.0	53.0	46.33333333	41.66666667	30.0	42.33333
78.33333333	67.66666667	76.0	75.33333333	71.0	69.0
57.0	52.66666667	66.66666667	70.33333333	61.0	58.66666
56.0	47.66666667	48.66666667	23.33333333	38.33333333	42.0
81.66666667	51.0	68.0	69.66666667	61.66666667	59.0
67.33333333	48.33333333	56.33333333	56.66666667	60.0	53.33333

61.0	66.66666667	60.33333333	73.33333333	60.0	61.0
67.0	66.0	64.33333333	57.0	64.0	60.66666

Se borran los estudiantes que tienen la nota menor a 0 ya que estos estudiantes no continuaron sus estudios

In [52]:
```
students=students[students['CALCULO']>0]
```

Se crea funcion para convertir caracteristicas categoricas en 0 y 1

In [53]:
```
def convertCategory(df, variable):
    #df = df.to_dataframe()
    var = pd.get_dummies(df[variable], prefix=variable)
    df = df.drop(variable, axis=1)
    var = var.iloc[:, [1]]
    df = pd.concat([df,var],axis=1)
    return df
```

Se crea funcion para renombrar las caracteristicas categoricas

In [54]:
```
def renameCategory(df, variable):
    df = df.rename(columns={students.columns[-1]:variable})
    return df
```

In [55]:
```
# Se convierten los datos a DataFrame
students = pd.DataFrame(students)
```

In [56]:
```
#Se convierte y renombra la variable categorica S siendo 0 para muje
students = convertCategory(students, 'S')
students = renameCategory(students, 'S')
#Se convierte y renombra la variable categorica R siendo 1 para Repi
students = convertCategory(students, 'R')
students = renameCategory(students, 'R')
#Se convierte y renombra la variable categorica J10 y J11 siendo 1 p
students = convertCategory(students, 'J10')
students = renameCategory(students, 'J10')
students = convertCategory(students, 'J11')
students = renameCategory(students, 'J11')
```

Se verifican y validan que los tipos de datos de las columnas sehan adecuados para analizar

In [57]:
```
students.dtypes
```

Out[57]:
```
ESTUDIANTE                  object
CURSO                        int64
Estrato                      int64
AÑO                          int64
ECONOMIA                   float64
FÍSICA                       int64
QUÍMICA                    float64
SOCIALES                   float64
EDUCACION ARTISTICA        float64
RECREACIÓN Y DEPORTES      float64
RELIGION                   float64
ETICA                      float64
FILOSOFIA                  float64
INGLES                     float64
LENGUA CASTELLANA          float64
TRIGONOMETRIA              float64
ADMINISTRACION             float64
MANEJO  ORDENADORES        float64
PREPARACIÓN TECNOLÓGICA    float64
CALCULO                    float64
CONVENCIONES MINISTERIO     object
C11                         object
A11                         object
S                            uint8
R                            uint8
J10                          uint8
J11                          uint8
dtype: object
```

Se identifica que las notas de fisica esta en formato int, se convierte a float para mejorar la uniformidad de los datos

In [58]:
```
students['FÍSICA'] = students['FÍSICA'].astype(float)
```

Se visualizan la cantidad de filas y columna, que se encuentran en la sábana de notas

In [59]:
```
students.shape
```

Out[59]:
```
(182, 27)
```

En la siguiente tabla se pueden ver información estadistica de las columnas que tienen valores numericos

In [60]:
```
students.describe()
```

Out[60]:

	CURSO	Estrato	AÑO	ECONOMIA	FÍSICA	QUÍMICA	SOCI
count	182.000000	182.000000	182.000000	182.000000	182.000000	182.000000	182.0
mean	1002.280220	1.873626	2016.417582	65.382418	65.719780	63.880952	68.7
std	1.015648	0.333187	0.494521	6.876521	8.148046	8.860483	8.1
min	1001.000000	1.000000	2016.000000	45.300000	43.000000	39.000000	31.1
25%	1001.000000	2.000000	2016.000000	61.000000	61.000000	60.000000	64.3
50%	1002.000000	2.000000	2016.000000	63.700000	65.000000	62.000000	68.3
75%	1003.000000	2.000000	2017.000000	69.925000	71.000000	69.000000	74.3
max	1004.000000	2.000000	2017.000000	87.000000	89.000000	93.000000	89.7

8 rows × 23 columns

Se identifica la correlación de la variables de las notas.

In [61]:
```
Notas = students
Notas = Notas.drop(columns=['CURSO','AÑO','Estrato','S','R','J10','
Notas.corr()
```

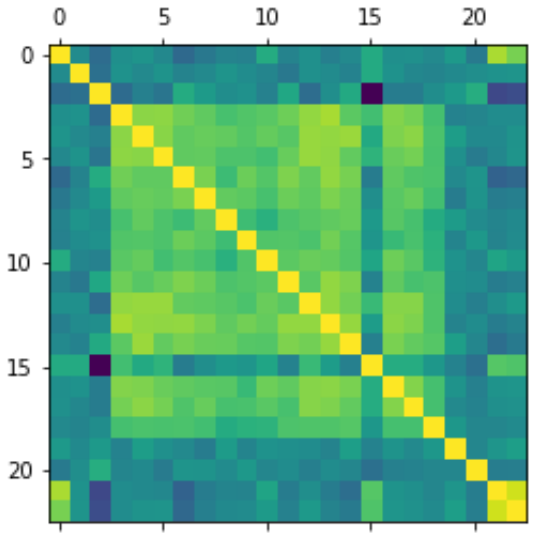
Out[61]:

	ECONOMIA	FÍSICA	QUÍMICA	SOCIALES	EDUCACION ARTISTICA	RECREACI Y DEPORT
ECONOMIA	1.000000	0.654560	0.662660	0.568937	0.512738	0.4199
FÍSICA	0.654560	1.000000	0.646565	0.519180	0.536394	0.5254
QUÍMICA	0.662660	0.646565	1.000000	0.515767	0.523293	0.4469
SOCIALES	0.568937	0.519180	0.515767	1.000000	0.598828	0.3757
EDUCACION ARTISTICA	0.512738	0.536394	0.523293	0.598828	1.000000	0.5274
RECREACIÓN Y DEPORTES	0.419977	0.525484	0.446926	0.375750	0.527466	1.0000
RELIGION	0.464416	0.468681	0.447482	0.556417	0.497146	0.3825
ETICA	0.492083	0.532020	0.402974	0.470298	0.424762	0.2795
FILOSOFIA	0.559481	0.488587	0.541596	0.620119	0.556738	0.4492
INGLES	0.682503	0.697686	0.690920	0.525902	0.507173	0.4865
LENGUA CASTELLANA	0.740452	0.680097	0.668528	0.681563	0.606110	0.5129
TRIGONOMETRIA	0.504239	0.699965	0.524839	0.593761	0.549845	0.5368
ADMINISTRACION	0.402085	0.220494	0.304610	-0.147166	-0.005689	0.0877
MANEJO ORDENADORES	0.630284	0.615061	0.548704	0.520013	0.496597	0.4863
PREPARACIÓN TECNOLÓGICA	0.587149	0.656335	0.584660	0.452037	0.536868	0.4200
CALCULO	0.413420	0.438967	0.431705	0.438208	0.403474	0.2370

Representación grafica de la correlación de las variables, las correlaciones positivas estan en amarillo y los colores mas claros indican una correlacion baja o nula.

In [62]:
```
plt.matshow(students.corr())
```

Out[62]:
```
<matplotlib.image.AxesImage at 0x7fb6940b0550>
```



In [63]:
```
# Se convierten los datos a SFrame
students = tr.SFrame(students)
```

En la siguiente tabla se pueden ver información estadistica de la caracteristica objetivo

CALCULO

In [64]:
```
students['CALCULO'].summary()
```

Out[64]:

```
+--------------------+--------------------+----------+
|        item        |       value        | is exact |
+--------------------+--------------------+----------+
|       Length       |        182         |   Yes    |
|        Min         |        40.0        |   Yes    |
|        Max         |        93.0        |   Yes    |
|        Mean        |  68.1056776558242  |   Yes    |
|        Sum         |   12395.23333336   |   Yes    |
|      Variance      | 110.79237924410957 |   Yes    |
| Standard Deviation | 10.525795895993308 |   Yes    |
|  # Missing Values  |         0          |   Yes    |
|  # unique values   |         86         |    No    |
+--------------------+--------------------+----------+

Most frequent items:
+-------------+-------+
|    value    | count |
+-------------+-------+
|     54.3    |   40  |
|     75.0    |   6   |
|     64.0    |   5   |
|     72.0    |   5   |
|     67.7    |   4   |
|     68.3    |   4   |
| 64.66666667 |   3   |
| 73.33333333 |   3   |
|     70.0    |   3   |
|     67.0    |   3   |
+-------------+-------+

Quantiles:
+------+------+------+-------------+------+-------------+------+-
-----+------+
|  0%  |  1%  |  5%  |     25%     | 50%  |     75%     | 95%  |
99%  | 100% |
+------+------+------+-------------+------+-------------+------+-
-----+------+
| 40.0 | 48.7 | 54.3 | 58.66666667 | 68.3 | 75.66666667 | 85.7 |
92.0 | 93.0 |
+------+------+------+-------------+------+-------------+------+-
-----+------+
```

Histograma con regla de Sturges para las notas de Calculo

Se puede apreciar en el histograma que los datos tienen una distribución sin muchos valores atipicos

In [65]:
```
plt.hist(students['CALCULO'], bins= 1 + int(np.ceil(np.log2(len(stuc
plt.xlabel('Notas')
plt.ylabel('Frecuencia')
plt.title('Histograma Notas de Calculo')
```

Out[65]:
```
Text(0.5, 1.0, 'Histograma Notas de Calculo')
```

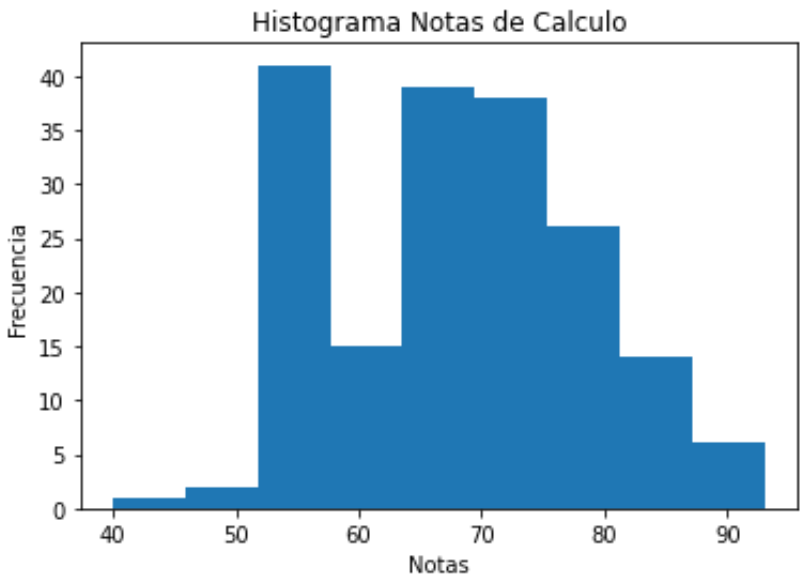
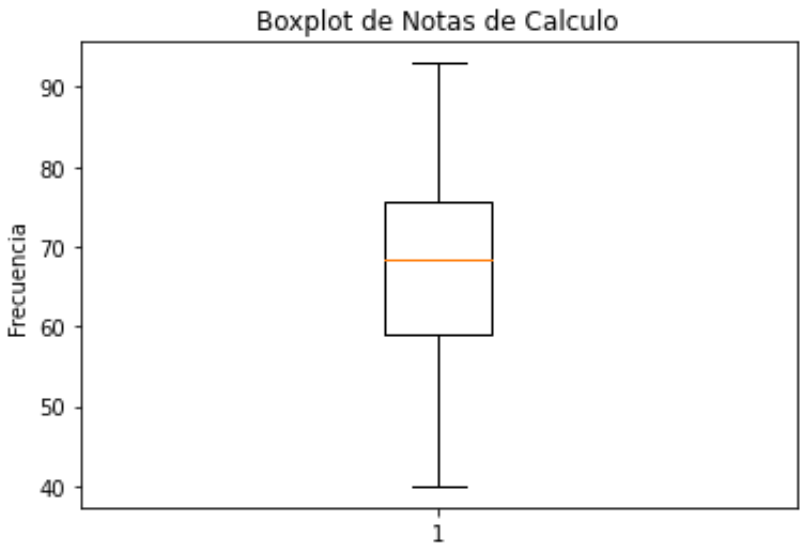


Diagrama de caja para las notas de Calculo

En este diagrama se puede identificar la nota media y donde se concentran la mayoria de los datos

In [66]:
```
plt.boxplot(students['CALCULO'])
plt.ylabel('Frecuencia')
plt.title('Boxplot de Notas de Calculo')
```

Out[66]:
```
Text(0.5, 1.0, 'Boxplot de Notas de Calculo')
```



De los anteriores diagramas se puede determinar que los datos de las notas de Calculo son viable para analisis ya que no son aleatorios y la mayoria de las notas se encuentran entre 57 y 75.

Una vez analizados los datos se crea una columna donde las notas que tengan una calificación de más de 60 para se revisaran como *positivas*, mientras que las que tengan una calificación menor de 60 seran *negativas*. Para la columna de Fracaso, se usa +1 para la revisión de clase positiva y -1 para la etiqueta de clase negativa.

In [67]:
```
students['Fracaso'] = students['CALCULO'].apply(lambda CALCULO : +1
```

Se puede visualizar que el conjunto de datos contiene una columna adicional llamada Fracaso que es positivo(+1) o negativo (-1)

In [68]:
```
fracaso = [ len(students[students['Fracaso'] == 1]) , len(students[s
fracaso
```

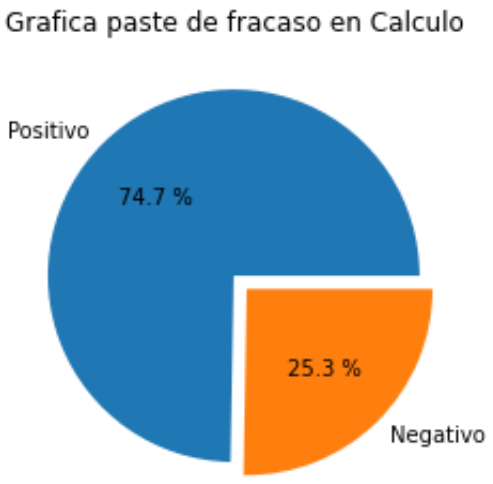
Out[68]:
```
[136, 46]
```

Grafica de pastel de columna fracaso

En esta grafica se puede visualizar que la cuarta parte de los estudiantes tiende a perder la asignatura de Calculo, lo cual representará dificultades y/o posible desercion en su formación de educación superior en carreras que requieran de buenos conocimientos matematicos como los son las ingenierias.

In [69]:
```
desfase = (0,0.10)
plt.pie(fracaso, labels= ["Positivo","Negativo"], autopct="%0.1f %%"
plt.title('Grafica paste de fracaso en Calculo')
```

Out[69]:
```
Text(0.5, 1.0, 'Grafica paste de fracaso en Calculo')
```



División de los datos en conjuntos de entrenamiento y de prueba

Se realiza la división de los datos a los que se realizara el estrenamiento y prueba, donde se utilizara el 80% de los datos en el conjunto de entrenamiento y el 20% de los datos en el conjunto de prueba. Se usa seed = 1 para que los datos obtengan el mismo resultado.

In [70]:
```
train_data, test_data = students.random_split(.7, seed=1)
print(len(train_data))
print(len(test_data))
```

```
118
64
```

Se realiza la selección de caracteristicas teniendo en cuenta las variable dicotomicas, sin tener en cuenta el nombre de los estudiantes, la caracteristica `CALCULO` y la caracteristica `CONVENCIONES MINISTERIO` ya que se creo la caracteristica objetivo Fracaso.

In [71]:
```
# Se crea una variable donde se encontraran las columnas
caracteristicas = students.column_names()[1:27]

column_excluir = ["CALCULO", "CONVENCIONES MINISTERIO"]

# Se elimina la caracteristica CALCULO y CONVENCIONES MINISTERIO
caracteristicas = sublist = [x for x in caracteristicas if x not in

caracteristicas
```

Out[71]:
```
['CURSO',
 'Estrato',
 'AÑO',
 'ECONOMIA',
 'FÍSICA',
 'QUÍMICA',
 'SOCIALES',
 'EDUCACION ARTISTICA',
 'RECREACIÓN Y DEPORTES',
 'RELIGION',
 'ETICA',
 'FILOSOFIA',
 'INGLES',
 'LENGUA CASTELLANA',
 'TRIGONOMETRIA',
 'ADMINISTRACION',
 'MANEJO  ORDENADORES',
 'PREPARACIÓN TECNOLÓGICA',
 'C11',
 'A11',
 'S',
 'R',
 'J10',
 'J11']
```

Entrenamiento del clasificador de estudiantes con regresión logística

Ahora se utiliza la regresión logística para crear un clasificador de estudiantes en los datos de entrenamiento. Este modelo utilizará los valores students como una característica y la columna de Fracaso como objetivo. Se usará validation_set = None para obtener los mismos resultados que todos los demás.

In [72]:
```
students_model = tr.logistic_classifier.create(train_data,
                                                target = 'Fr
                                                features = c
                                                validation_s
```

```
Logistic regression:

--------------------------------------------------------

Number of examples          : 118

Number of classes           : 2

Number of feature columns   : 24

Number of unpacked features : 24

Number of coefficients      : 26

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+-------------------+

| Iteration | Passes   | Elapsed Time | Training Accuracy |

+-----------+----------+--------------+-------------------+

| 1         | 2        | 1.005747     | 0.923729          |

| 2         | 3        | 1.009920     | 0.932203          |

| 3         | 4        | 1.011362     | 0.932203          |

| 4         | 5        | 1.012548     | 0.940678          |

| 5         | 6        | 1.013912     | 0.932203          |

| 6         | 7        | 1.015472     | 0.932203          |

+-----------+----------+--------------+-------------------+

SUCCESS: Optimal solution found.
```

Ahora se puede visualizar el clasificador donde se evidencia que se encontro una solucion óptima.

In [73]:
```
students_model
```

Out[73]:
```
Class                          : LogisticClassifier

Schema
------
Number of coefficients         : 26
Number of examples             : 118
Number of classes              : 2
Number of feature columns      : 24
Number of unpacked features    : 24

Hyperparameters
---------------
L1 penalty                     : 0.0
L2 penalty                     : 0.01

Training Summary
----------------
Solver                         : newton
Solver iterations              : 6
Solver status                  : SUCCESS: Optimal solution found.
Training time (sec)            : 1.0171

Settings
--------
Log-likelihood                 : 21.4187

Highest Positive Coefficients
-----------------------------
(intercept)                    : 17.0598
J10                            : 1.3396
S                              : 0.1473
FILOSOFIA                      : 0.0662
SOCIALES                       : 0.0355

Lowest Negative Coefficients
----------------------------
C11[1102]                      : -10.369
C11[1103]                      : -9.0237
A11[2018]                      : -8.0075
R                              : -5.2243
J11                            : -3.8427
```

Cuando se tiene el modelo se pueden extraer los pesos (coeficientes) como un SFrame

In [74]:
```
coeficientes = students_model.coefficients
coeficientes
```

Out[74]:

name	index	class	value	stderr
(intercept)	None	1	17.059824721788804	16215.126120211367
CURSO	None	1	6.190432712863723e-05	0.9536800586905394
Estrato	None	1	-1.1840941438085149	1.6687267781903081
AÑO	None	1	2.2577873980147676e-06	8.087552148687868
ECONOMIA	None	1	-0.03368391711600948	0.13970046546597867
FÍSICA	None	1	0.022528114297520118	0.08991329530389969
QUÍMICA	None	1	-0.015623545105073373	0.09467866530180383
SOCIALES	None	1	0.03545297476478828	0.09432698378590193
EDUCACION ARTISTICA	None	1	-0.01036769606464626	0.07648882420073409
RECREACIÓN Y DEPORTES	None	1	0.007549084195047999	0.10447672420666211

[26 rows x 5 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

Hay un total de 25 coeficientes en el modelo. Los coeficientes positivos □□corresponden a pesos que causan notas positivas, mientras que los pesos negativos corresponden a notas negativas.

El siguiente bloque de código calcula cuántos pesos son positivos (> = 0). (Sugerencia: la columna 'valor' en los pesos de SFrame debe ser positiva (> = 0)).

In [75]:
```
num_coeficiente_positivo = 0
num_coeficiente_negativo = 0

for value in coeficientes['value']:

    if  value >= 0:
        num_coeficiente_positivo += 1
    else:
        num_coeficiente_negativo += 1

print("Numero de coeficientes positivos: %s " % num_coeficiente_posi
print("Numero de coeficientes negativos: %s " % num_coeficiente_nega
```

```
Numero de coeficientes positivos: 14
Numero de coeficientes negativos: 12
```

Predicción con Regresión Logistica

Ahora que un modelo está entrenado, se realizan las predicciones sobre los **datos de la prueba**. Se realiza la exploración en el contexto de 3 ejemplos en el conjunto de datos de prueba. Se hace referencia a este conjunto de 3 ejemplos como **muestra_prueba_datos**.

In [76]:
```
test_data['CALCULO']
```

Out[76]:
```
dtype: float
Rows: 64
[69.33333333, 77.0, 75.0, 74.33333333, 70.3, 72.0, 76.66666667, 7
7.0, 73.33333333, 82.66666667, 61.33333333, 78.33333333, 92.0, 6
2.33333333, 74.66666667, 77.66666667, 85.66666667, 54.3, 54.3, 6
7.66666667, 84.3, 63.0, 78.0, 69.66666667, 54.3, 65.33333333, 7
5.0, 54.3, 54.3, 54.3, 54.3, 54.3, 76.3, 72.0, 68.0, 67.7, 48.7,
74.3, 71.7, 68.3, 62.0, 56.3, 79.3, 54.3, 86.0, 82.7, 69.0, 70.0,
68.7, 64.3, 68.3, 67.3, 54.3, 54.3, 68.3, 67.0, 78.0, 78.7, 63.3,
68.3, 77.7, 68.7, 67.7, 82.0]
```

In [77]:
```
muestra_prueba_datos = test_data[13:16]
print(muestra_prueba_datos['CALCULO'])
muestra_prueba_datos
```

```
[62.33333333, 74.66666667, 77.66666667]
```

Out[77]:

ESTUDIANTE	CURSO	Estrato	AÑO	ECONOMIA	FÍSICA	QUÍMICA	SOC
DAZA PANCHE YUDY NATHALIA	1002	2	2016	62.7	64.0	66.0	(
DIAZ ALVARADO NATHALY	1002	2	2016	71.7	80.0	86.0	1
GARCIA RIAÑO NATALIA	1002	2	2016	78.7	72.0	76.0	1

RECREACIÓN Y DEPORTES	RELIGION	ETICA	FILOSOFIA	INGLES	LENGUA CASTELLANA	TRIG
60.3	68.3	74.0	62.3	60.0	66.0	
85.0	77.0	78.0	84.0	75.0	83.0	
63.0	76.0	75.0	85.0	74.0	83.0	

MANEJO ORDENADORES	PREPARACIÓN TECNOLÓGICA	CALCULO	CONVENCIONES MINISTERIO	C11	A11
72.0	61.0	62.33333333	BASICO	1102	2017
96.0	91.0	74.66666667	BASICO	1102	2017
88.0	81.0	77.66666667	BASICO	1102	2017

Fracaso
1
1
1

[3 rows x 28 columns]

Profundizando en la primera fila de **muestra_prueba_datos**. Este seria el detalle completo del estudiante:

In [78]:
```
muestra_prueba_datos[0]
```

Out[78]:
```
{'ESTUDIANTE': 'DAZA PANCHE YUDY NATHALIA',
 'CURSO': 1002,
 'Estrato': 2,
 'AÑO': 2016,
 'ECONOMIA': 62.7,
 'FÍSICA': 64.0,
 'QUÍMICA': 66.0,
 'SOCIALES': 64.7,
 'EDUCACION ARTISTICA': 55.3,
 'RECREACIÓN Y DEPORTES': 60.3,
 'RELIGION': 68.3,
 'ETICA': 74.0,
 'FILOSOFIA': 62.3,
 'INGLES': 60.0,
 'LENGUA CASTELLANA': 66.0,
 'TRIGONOMETRIA': 63.3,
 'ADMINISTRACION': 60.0,
 'MANEJO  ORDENADORES': 72.0,
 'PREPARACIÓN TECNOLÓGICA': 61.0,
 'CALCULO': 62.33333333,
 'CONVENCIONES MINISTERIO': 'BASICO',
 'C11': '1102',
 'A11': '2017',
 'S': 0,
 'R': 0,
 'J10': 0,
 'J11': 0,
 'Fracaso': 1}
```

Esa estudiante perdio calculo adicionalmente es propenso a perder economia y quimica.

En la siguiente fila de **muestra_prueba_datos**. Se puede evidenciar que es un buen estudiante, no perdio ninguna materia y no esta propenso a perder alguna otra.

In [79]:
```
muestra_prueba_datos[1]
```

Out[79]:
```
{'ESTUDIANTE': 'DIAZ ALVARADO NATHALY',
 'CURSO': 1002,
 'Estrato': 2,
 'AÑO': 2016,
 'ECONOMIA': 71.7,
 'FÍSICA': 80.0,
 'QUÍMICA': 86.0,
 'SOCIALES': 73.0,
 'EDUCACION ARTISTICA': 80.0,
 'RECREACIÓN Y DEPORTES': 85.0,
 'RELIGION': 77.0,
 'ETICA': 78.0,
 'FILOSOFIA': 84.0,
 'INGLES': 75.0,
 'LENGUA CASTELLANA': 83.0,
 'TRIGONOMETRIA': 76.7,
 'ADMINISTRACION': 90.0,
 'MANEJO  ORDENADORES': 96.0,
 'PREPARACIÓN TECNOLÓGICA': 91.0,
 'CALCULO': 74.66666667,
 'CONVENCIONES MINISTERIO': 'BASICO',
 'C11': '1102',
 'A11': '2017',
 'S': 0,
 'R': 0,
 'J10': 0,
 'J11': 1,
 'Fracaso': 1}
```

A continuación se realiza la predicción **clase** para **muestra_prueba_datos**. El `students_model` debe predecir **+1** si la nota es positiva y **- 1** si la nota es negativa. El **puntaje** (a veces llamado **margen**) para el modelo de regresión logística se define como:

$$\text{puntaje}_{\square} = \square^{\square} h(\square_{\square})$$

donde $h(\square_{\square})$ representa las características, por ejemplo, $\square$.El siguiente código permitira obtener los **puntajes** usando Turi Create. Para cada fila, el **puntaje** (o margen) es un número en el rango entre **[- inf, inf]**.

In [80]:
```
puntajes = students_model.predict(muestra_prueba_datos, output_type=
print(puntajes)
```

```
[3.4675152931006235, 0.6969034319864065, 4.636478575226176]
```

Prediccion de la Nota

Los puntajes anteriormente obtenidos se pueden usar para hacer predicciones de clase

de la siguiente manera:

$$\hat{y} = \begin{cases} +1 & \hat{f}(x) > 0 \\ -1 & \hat{f}(x) \leq 0 \end{cases}$$

Usando los puntajes, escriba el código para calcular $\hat{y}$ las predicciones de la clase:

In [81]:
```
predicciones = puntajes.apply(lambda x: 1 if x > 0 else -1)
# Se crea una función para usarla mas adelante
def Calcular_Prediccion(puntajes):
    prediccion = []
    for puntaje in puntajes:
        if puntaje>0:
            prediccion.append(1)
        else:prediccion.append(-1)
    return prediccion

prediccion = Calcular_Prediccion(puntajes)
print (prediccion)
print (predicciones)
```

```
[1, 1, 1]
[1, 1, 1]
```

El siguiente código permite verificar que las predicciones de clase obtenidas en los cálculos anteriores sean las mismas que las obtenidas de Turi Create.

In [82]:
```
print(students_model.predict(muestra_prueba_datos))
```

```
[1, 1, 1]
```

Probabilidad de la Predicción

Para calcular las probabilidades de las predicciones de los puntajes se usa:

$$P(\hat{y} = +1 | x, \hat{f}) = \frac{1}{1 + \exp(-\hat{f}(x))}$$

Usando la variable **puntajes** calculada previamente, se puede calcular la probabilidad de que una nota sea positiva usando la fórmula anterior. Para cada fila, las probabilidades deben ser un número en el rango **[0, 1]**.

In [83]:
```
probabilidades = puntajes.apply(lambda x: 1 / (1 + np.math.exp(-x)))
# Se crea una funcion para el calculo de la probabilidad
def Calcular_Probabilidad(puntajes):
    probabilidad = []
    for puntaje in puntajes:
        probabilidad.append( 1/(1+np.math.exp(-puntaje)))
    return probabilidad

probabilidad =  Calcular_Probabilidad(puntajes)
print (probabilidad)
print (probabilidades)
```

```
[0.9697492129705938, 0.6675008659813135, 0.9904012622905511]
[0.9697492129705938, 0.6675008659813135, 0.9904012622905511]
```

El siguiente codigo permite verificar la probabilidad de las predicciones las cuales coinciden con las obtenidas de Turi Create.

In [84]:
```
print (students_model.predict(muestra_prueba_datos, output_type='pro
```

```
[0.9697492129705938, 0.6675008659813135, 0.990401262290551]
```

Encontrar los estudiantes con mayores probabilidades de tener buena nota

Ahora se puede examinar el conjunto de datos de prueba completo, **test_data**, y se usa Turi Create para formar predicciones en todos los puntos de datos de prueba para un rendimiento más rápido.

Usando el `students_model` , se encuentrara las 20 revisiones en todo **test_data** con la **mayor probabilidad** de ser clasificado como **nota positiva**. Se hace referencia a estos como las "notas más positivos".

Para calcular estas 20 notas principales, se realizan los siguientes pasos:

1. Hacer predicciones de probabilidad en **test_data** usando el `students_model` . Se utiliza el metodo `.predict` para hacer predicciones sobre los datos de prueba, adicionalmente se usa la opción `output_type = 'probability'` para generar la probabilidad en lugar de solo la clase más probable).
2. Ordenar los datos de acuerdo con las predicciones y elijir los primeros 20. Se utiliza el método `.topk` en un SFrame para encontrar las k filas superiores ordenadas según el valor de una columna especificada.

In [85]:
```
test_data['probability'] = students_model.predict(test_data, output_
test_data.topk('probability', k=20, reverse=False).print_rows(num_r
```

```
+--------------------------------+-------+---------+------+----
------+---------+
|           ESTUDIANTE           | CURSO | Estrato | AÑO  | ECO
NOMIA | FÍSICA |
+--------------------------------+-------+---------+------+----
------+---------+
|   RAMIREZ ARANGO TANIA YISEL   |  1001 |    2    | 2016 |   6
8.7   |  64.0  |
| MANZANO PATIÑO JULIAN ALEX...  |  1001 |    2    | 2016 |   6
1.3   |  69.0  |
|  LUIS LEGUIZAMON DAYANA SOFIA  |  1001 |    2    | 2016 |   7
6.3   |  66.0  |
|  RICO ESPITIA JOHAN FERNANDO   |  1001 |    2    | 2016 |   7
5.0   |  71.0  |
| BARRIOS GARCIA SHELCY VIVIANA  |  1002 |    2    | 2016 |   6
8.3   |  61.0  |
| MARTINEZ LOAIZA JESSICA AL...  |  1001 |    2    | 2016 |   5
9.7   |  64.0  |
| ARIZA MARTINEZ CAMPVERLLY ...  |  1001 |    2    | 2016 |   6
```

Encontrar los estudiantes con menores probabilidades de tener buena nota

Para encontrar las "notas más negativas" se usan las probabilidades de predicción para encontrar las 20 revisiones en test_data con la probabilidad más baja de ser clasificada como una nota positiva.

In [86]:
```
test_data.topk('probability', k=20, reverse=True).print_rows(num_rov
```

```
+--------------------------------+-------+---------+------+----
------+---------+
|            ESTUDIANTE          | CURSO | Estrato | AÑO  | ECO
NOMIA | FÍSICA |
+--------------------------------+-------+---------+------+----
------+---------+
|   VILLALOBOS MOLINA NATALIA    |  1004 |    2    | 2016 |   6
1.7   |  51.0  |
|   TORRES OCHOA BRAYAN STIVEN   |  1003 |    2    | 2016 |   5
1.0   |  57.0  |
|    TORRES PARRA LAURA PAOLA    |  1002 |    2    | 2017 |   6
3.3   |  61.0  |
|    TORRES PARRA LAURA PAOLA    |  1002 |    2    | 2016 |   7
1.7   |  67.0  |
|   VELA ACERO EVELYN YOMAIRA    |  1004 |    2    | 2016 |   7
0.7   |  59.0  |
|   ROJAS ALDANA DIANA MARCELA   |  1004 |    2    | 2016 |   4
6.3   |  47.0  |
| SAENZ QUINTERO SAIRA VALENTINA |  1002 |    2    | 2016 |   6
```

Evaluar el Modelo, calcular la precisión y exactitud del clasificador

A continuación se evaluara la precisión del clasificador entrenado. La precisión viene dada por:

$$\text{precisión} = \frac{\text{\# ejemplos correctamente clasificados}}{\text{\# total de ejemplos}}$$

Esto se puede calcular de la siguiente manera:

- **Paso 1:** Usar el modelo entrenado para calcular las predicciones de clase usa el método `predic`.
- **Paso 2:** Contar el número de puntos de datos cuando las notas de clase predichas coincidan con las notas de verdad básica las cuales seran llamadas `notas_verdaderas`.
- **Paso 3:** Dividir el número total de predicciones correctas por el número total de puntos de datos en el conjunto de datos.

La siguiente función calcula la precisión de clasificación:

In [87]:
```
def get_Precision_Clasificacion(model, data, true_notes):
    # Primero se obtienen las predicciones
    puntaje = model.predict(data, output_type='margin')

    # Se calcula el número de ejemplos correctamente clasificados
    countador_muestras_clasificadas_correctas = 0
    notaPrediccion =  Calcular_Prediccion(puntaje)

    for i in range(len(puntaje)):
        if notaPrediccion[i] == true_notes[i]:
            countador_muestras_clasificadas_correctas+=1

    # Luego se calcula la precisión dividiendo num_correcto por el r
    precision = countador_muestras_clasificadas_correctas/(len(punta

    return precision
    return precision
```

Ahora, se calcula la precisión de clasificación del **students_model** en **test_data**.

In [88]:
```
get_Precision_Clasificacion(students_model, train_data, train_data[
```

Out[88]:
```
0.9322033898305084
```

La precisión del modelo tambien se puede obtener con el metodo `evaluate` en el cual se evalua la precision y se realiza una matriz de confusión, donde se tienen en cuenta los datos de prueba y los datos de entrenamiento razon por la cual tiene un poco mas de precisión.

Evaluación del modelo con los datos de entrenamiento

In [89]:
```
students_model.evaluate(train_data)
```

Out[89]:
```
{'accuracy': 0.9322033898305084,
 'auc': 0.9729055258467016,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

 Rows: 4

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |      1       |        1        |   82  |
 |      -1      |        -1       |   28  |
 |      -1      |        1        |   5   |
 |      1       |        -1       |   3   |
 +--------------+-----------------+-------+
 [4 rows x 3 columns],
 'f1_score': 0.9534883720930232,
 'log_loss': 0.18151477606880836,
 'precision': 0.9425287356321839,
 'recall': 0.9647058823529412,
 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int

 Rows: 100001

 Data:
 +-----------+-----+-----+----+----+
 | threshold | fpr | tpr | p  | n  |
 +-----------+-----+-----+----+----+
 |    0.0    | 1.0 | 1.0 | 85 | 33 |
 |   1e-05   | 1.0 | 1.0 | 85 | 33 |
 |   2e-05   | 1.0 | 1.0 | 85 | 33 |
 |   3e-05   | 1.0 | 1.0 | 85 | 33 |
 |   4e-05   | 1.0 | 1.0 | 85 | 33 |
 |   5e-05   | 1.0 | 1.0 | 85 | 33 |
 |   6e-05   | 1.0 | 1.0 | 85 | 33 |
 |   7e-05   | 1.0 | 1.0 | 85 | 33 |
 |   8e-05   | 1.0 | 1.0 | 85 | 33 |
 |   9e-05   | 1.0 | 1.0 | 85 | 33 |
 +-----------+-----+-----+----+----+
 [100001 rows x 5 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more
rows and columns.}
```

Evaluación del modelo con los datos de prueba

In [90]:
```
students_model.evaluate(test_data)
```

Out[90]:
```
{'accuracy': 0.90625,
 'auc': 0.9713423831070889,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

 Rows: 4

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |      1       |        -1       |   5   |
 |      1       |        1        |   46  |
 |      -1      |        -1       |   12  |
 |      -1      |        1        |   1   |
 +--------------+-----------------+-------+
 [4 rows x 3 columns],
 'f1_score': 0.9387755102040817,
 'log_loss': 0.22282073710616115,
 'precision': 0.9787234042553191,
 'recall': 0.9019607843137255,
 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int

 Rows: 100001

 Data:
 +-----------+--------------------+-----+----+----+
 | threshold |        fpr         | tpr | p  | n  |
 +-----------+--------------------+-----+----+----+
 |    0.0    |        1.0         | 1.0 | 51 | 13 |
 |   1e-05   |        1.0         | 1.0 | 51 | 13 |
 |   2e-05   |        1.0         | 1.0 | 51 | 13 |
 |   3e-05   |        1.0         | 1.0 | 51 | 13 |
 |   4e-05   |        1.0         | 1.0 | 51 | 13 |
 |   5e-05   |        1.0         | 1.0 | 51 | 13 |
 |   6e-05   | 0.9230769230769231 | 1.0 | 51 | 13 |
 |   7e-05   | 0.9230769230769231 | 1.0 | 51 | 13 |
 |   8e-05   | 0.9230769230769231 | 1.0 | 51 | 13 |
 |   9e-05   | 0.9230769230769231 | 1.0 | 51 | 13 |
 +-----------+--------------------+-----+----+----+
 [100001 rows x 5 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more
rows and columns.}
```

Consolidado de los datos de entrenamiento y de prueba

In [91]:
```
students_model.evaluate(students)
```

Out[91]:
```
{'accuracy': 0.9230769230769231,
 'auc': 0.9675511508951402,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

 Rows: 4

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |      1       |        -1       |   8   |
 |      1       |        1        |  128  |
 |      -1      |        -1       |   40  |
 |      -1      |        1        |   6   |
 +--------------+-----------------+-------+
 [4 rows x 3 columns],
 'f1_score': 0.9481481481481482,
 'log_loss': 0.19603994918084452,
 'precision': 0.9552238805970149,
 'recall': 0.9411764705882353,
 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int

 Rows: 100001

 Data:
 +-----------+--------------------+-----+-----+----+
 | threshold |        fpr         | tpr |  p  | n  |
 +-----------+--------------------+-----+-----+----+
 |    0.0    |        1.0         | 1.0 | 136 | 46 |
 |   1e-05   |        1.0         | 1.0 | 136 | 46 |
 |   2e-05   |        1.0         | 1.0 | 136 | 46 |
 |   3e-05   |        1.0         | 1.0 | 136 | 46 |
 |   4e-05   |        1.0         | 1.0 | 136 | 46 |
 |   5e-05   |        1.0         | 1.0 | 136 | 46 |
 |   6e-05   | 0.9782608695652174 | 1.0 | 136 | 46 |
 |   7e-05   | 0.9782608695652174 | 1.0 | 136 | 46 |
 |   8e-05   | 0.9782608695652174 | 1.0 | 136 | 46 |
 |   9e-05   | 0.9782608695652174 | 1.0 | 136 | 46 |
 +-----------+--------------------+-----+-----+----+
 [100001 rows x 5 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more
rows and columns.}
```

Se evalua la precisión con los datos de prueba y los datos de entrenamiento, evidenciando que es la misma que se dio como resultado la funcion `evaluate`

In [92]:
```
get_Precision_Clasificacion(students_model, students, students['Frac
```

Out[92]:
```
0.9230769230769231
```

Se realiza la predicción sobre todo el conjunto de datos de los estudiantes

In [93]:
```
students["NotaPredicha"] = students_model.predict(students, output_t
```

Entrenar otro clasificador con menos caracteristicas

Se usan todas la caracteristicas en el modelo anterior. Ahora se entrenará un modelo de regresión logística más simple utilizando solo un subconjunto de caracteristicas que aparecen en la sabana de notas. Para esta tarea, se seleccionaran solo las asignaturas. Estos son:

In [94]:
```
Asignaturas = ['ECONOMIA', 'FÍSICA', 'QUÍMICA', 'SOCIALES', 'EDUCACI
      'ETICA', 'FILOSOFIA', 'INGLES', 'LENGUA CASTELLANA', 'TRIGONOM
      'PREPARACIÓN TECNOLÓGICA']
```

In [95]:
```
len(Asignaturas)
```

Out[95]:
```
15
```

Entrenar un modelo de regresión logística en un subconjunto de datos

Ahora se crea un clasificador con **Asignaturas** como característica y ** Fracaso ** como objetivo.

In [96]:
```
simple_model = tr.logistic_classifier.create(train_data, target = 'I
simple_model
```

Out[96]:
```
Class                          : LogisticClassifier

Schema
------
Number of coefficients         : 16
Number of examples             : 118
Number of classes              : 2
Number of feature columns      : 15
Number of unpacked features    : 15

Hyperparameters
---------------
L1 penalty                     : 0.0
L2 penalty                     : 0.01

Training Summary
----------------
Solver                         : newton
Solver iterations              : 3
Solver status                  : SUCCESS: Optimal solution found.
Training time (sec)            : 0.0024

Settings
--------
Log-likelihood                 : 63.7327

Highest Positive Coefficients
-----------------------------
MANEJO  ORDENADORES            : 0.0631
SOCIALES                       : 0.0592
LENGUA CASTELLANA              : 0.042
FÍSICA                         : 0.0095
ADMINISTRACION                 : 0.006

Lowest Negative Coefficients
----------------------------
(intercept)                    : -1.4834
ECONOMIA                       : -0.0458
RECREACIÓN Y DEPORTES          : -0.0388
ETICA                          : -0.0368
RELIGION                       : -0.0193
```

Comparación de Modelos

Ahora se calcula la precisión de clasificación utilizando la función `get_Precision_Clasificacion` que se implementó anteriormente.

In [97]:
```
#Precisión de Modelo con menos caracteristicas
print(get_Precision_Clasificacion(simple_model, test_data, test_data
#Precisión de Modelo con mas caracteristicas
print(get_Precision_Clasificacion(students_model, test_data, test_da
```

```
0.84375
0.90625
```

Ahora se revisa la precisión con los datos de entrenamiento

In [98]:
```
#Precisión de Modelo con menos caracteristicas en los datos de entre
print(get_Precision_Clasificacion(simple_model, train_data, train_da
#Precisión de Modelo con mas caracteristicas en los datos de entrena
print(get_Precision_Clasificacion(students_model, train_data, train_
```

```
0.7372881355932204
0.9322033898305084
```

De lo anterior se puede concluir que entre menos caracteristicas tenga el modelo menor es su precisión.

Se revisan los coeficientes de **simple_model**:

In [99]:
```
simple_model.coefficients
```

Out[99]:

name	index	class	value	stderr
(intercept)	None	1	-1.483389766935871	3.676982996015537
ECONOMIA	None	1	-0.04578722212006558	0.062202171198081226
FÍSICA	None	1	0.0094640231983315	0.0473998748033986
QUÍMICA	None	1	-0.007784139156500198	0.04664424536698715
SOCIALES	None	1	0.05919566379936625	0.05216389826602649
EDUCACION ARTISTICA	None	1	0.0005437232943721551	0.03445356327998263
RECREACIÓN Y DEPORTES	None	1	-0.03875602200759394	0.04331912194084268
RELIGION	None	1	-0.01930382364791221	0.027862771050159187
ETICA	None	1	-0.03682056106269357	0.03769713504086529
FILOSOFIA	None	1	0.0010919691456089447	0.034471833873307954

[16 rows x 5 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

Se ordenan los coeficientes (en orden descendente) por el **valor** para obtener los coeficientes con el efecto más positivo en la nota.

In [100]:
```
simple_model.coefficients.sort('value', ascending=False).print_rows
```

```
+-------------------------+-------+-------+----------------------
-+
|           name          | index | class |         value
|
+-------------------------+-------+-------+----------------------
-+
|   MANEJO  ORDENADORES   |  None |   1   |  0.06313250960005795
|
|         SOCIALES        |  None |   1   |  0.05919566379936625
|
|    LENGUA CASTELLANA    |  None |   1   |  0.04201484580670062
|
|          FÍSICA         |  None |   1   |   0.0094640231983315
|
|      ADMINISTRACION     |  None |   1   |  0.005985018707629899
|
| PREPARACIÓN TECNOLÓGICA |  None |   1   | 0.0020988496251170678
|
|        FILOSOFIA        |  None |   1   | 0.0010919691456089447
|
|   EDUCACION ARTISTICA   |  None |   1   | 0.0005437232943721551
|
|          INGLES         |  None |   1   | -0.003951857552623652
|
|      TRIGONOMETRIA      |  None |   1   | -0.004651165281555013
|
|         QUÍMICA         |  None |   1   | -0.007784139156500198
|
|         RELIGION        |  None |   1   |  -0.01930382364791221
|
|          ETICA          |  None |   1   |  -0.03682056106269357
|
|  RECREACIÓN Y DEPORTES  |  None |   1   |  -0.03875602200759394
|
|         ECONOMIA        |  None |   1   |  -0.04578722212006558
|
|       (intercept)       |  None |   1   |   -1.483389766935871
|
+-------------------------+-------+-------+----------------------
-+
+----------------------+
|        stderr        |
+----------------------+
| 0.04456189368726723  |
| 0.05216389826602649  |
| 0.04799198406441616  |
|  0.0473998748033986  |
| 0.009878961494448942 |
| 0.04049865325416556  |
| 0.034471833873307954 |
| 0.03445356327998263  |
| 0.06472739516370737  |
|  0.0409964995818316  |
| 0.04664424536698715  |
| 0.027862771050159187 |
```

```
| 0.03769713504086529  |
| 0.04331912194084268  |
| 0.062202171198081226 |
|  3.676982996015537   |
+----------------------+
[16 rows x 5 columns]
```

Predicción de clase mayoritaria

Es bastante común usar el **clasificador de clase mayoritaria** como modelo de referencia (o referencia) para compararlo con su modelo de clasificador. El modelo clasificador mayoritario predice la clase mayoritaria para todos los puntos de datos. Por lo menos, debe vencer de manera saludable al clasificador de clase mayoritario, de lo contrario, el modelo (generalmente) no tiene sentido.

A continuación se representa la clase mayoritaria en el **train_data**?

In [101]:
```
num_positive  = (train_data['Fracaso'] == +1).sum()
num_negative = (train_data['Fracaso'] == -1).sum()
print(num_positive)
print(num_negative)
```

```
85
33
```

In [102]:
```
exactitud_entrenamiento = (num_positive)/(num_positive+num_negative)
exactitud_entrenamiento
```

Out[102]:
```
0.7203389830508474
```

Ahora se calcula la precisión del clasificador de clase mayoritaria en **test_data** para evidenciar la relación de los datos.

In [103]:
```
num_positive  = (test_data['Fracaso'] == +1).sum()
num_negative = (test_data['Fracaso'] == -1).sum()
print(num_positive)
print(num_negative)
exactitud_prueba = (num_positive)/(num_positive+num_negative)
exactitud_prueba
```

```
51
13
```

Out[103]:
```
0.796875
```

Para finalizar se exportan los datos resultantes del modelo con mejor precisión y exactitud a formato separado por comas en un documento llamados Estudiantes .csv

In [104]:

```
# Se realiza la conversion de SFrame a dataframe
Estudiantes = students.to_dataframe()
# Se crea una carpeta llamada dir y se va a almacenar los datos resu
ruta = './dir'
# Nombre del Archivo
name = 'Estudiantes.csv'

if not os.path.exists(ruta):
    os.mkdir(ruta)

fullname = os.path.join(ruta, name)
#Exportación de Datos a la ruta indicada
# Estudiantes.to_csv(fullname)
```